Bayware Documentation

Gregory Walter **Igor Tarasenko** **Adam Dubey**

Dec 12, 2019

Introduction

Welcome to Bayware!

CHAPTER 1

Overview

1.1 Problem

Compute resources have become instantly available and at ever-finer levels of granularity whenever an application requests them. Various orchestration solutions facilitate those requests in private data centers and public clouds. Some solutions go even further and allow applications to seamlessly spin up compute resources across multiple clouds and various virtualization platforms.

While compute resource management has greatly improved, connectivity provisioning in a muiticloud environment, on the contrary, is not so instant, seamless, and granular. Using a declarative language, the application can set up a new workload anywhere in the world without delay, but its connectivity with other workloads will depend on third-party configuration of multiple virtual or physical middleboxes–gateways, routers, load balancers, firewalls–installed in between.

The SDN-style, configuration-centric connectivity model doesn't allow applications to manage the communication environment in the same way as the computational environment. It is impossible for an application to declare a workload connectivity requirement (i.e., intent) one time and then have this desired connectivity applied each time a new workload instance appears in a private datacenter or a public cloud.

A pure application-level approach, in which communication intent translates into only HTTPS connections, also doesn't work. Confining all connectivity provisioning to a layer of mTLS tunnels while removing network checkpoints between workloads oversimplifies inherent application communication environment complexities. True, multi-layered security requires enforcing application intent at multiple levels, which implies middlebox configuration is not eliminated. ‘

To make migration to clouds easier, faster, and more secure, the application connectivity ought to become part of the application deployment code, with communication intent expressed in an infrastructure-agnostic manner. As a result, deploying a workload instance in any cloud, on any virtualization platform would automatically bring up both compute resources and desired connectivity. No middlebox configuration would be required to establish secure connectivity between workloads in different clouds, clusters, or other trust domains.

1.2 Solution

The Service Interconnection Fabric (SIF) is a cloud migration platform that enforces connectivity policy and provides service discovery functions for application services packaged as containers or virtual machines and deployed in private data centers or public clouds. The SIF eliminates network configuration by capturing and applying application communication intent as a **service graph** that is infrastructure-agnostic and based entirely on application service topology.

The SIF is zero-trust right out of the box. An application itself defines its connectivity policy and executes it at the time of workload deployment. The execution output are ephemeral security segments in the cross-cloud network, connecting subsets of workloads together. As such, whenever a workload appears anywhere in the fabric, it automatically receives the desired connectivity with other workloads as specified by the application service graph. Zero-trust ensures no connectivity exists between workloads that was neither specified nor requested.

In the SIF, connectivity is part of application deployment code. Moreover, the fabric itself is an infrastructure-as-code component. As such, connectivity policies and fabric resources easily integrate into the application CI/CD process. Whether rolling out an entire application or just a few microservices, the required resources and connectivity come up automatically. The policy and resource code is reusable and copy-paste portable across clouds.

The figure above presents the SIF software layers and their integration into a broader cloud infrastructure-as-code stack. The SIF upper layer enforces application connectivity policy and provides service discovery functions. The SIF bottom layer controls resource connectivity and facilitates resource deployment. In this way, the application communication environment is fully abstracted from the underlying infrastructure platforms. This allows an application to declare communication intent one time, using service identities instead of infrastructure-dependent IP addresses.

The SIF is fast and easy to deploy. The fabric creates a complete set of identity, security, routing and telemetry services automatically based on the service graph. No SDN or VNF solutions are required to establish secure connectivity between application services in different clouds or clusters. CNIs and complete logging and telemetry are built-in and configured automatically. As such, the fabric is ideal for hybrid clouds and hybrid VM/container deployments.

1.3 Product Architecture

The SIF consists of Fabric Manager and three types of nodes: Orchestrator, Processor, and Workload. Fabric Manager deploys nodes across clouds. Orchestrator controls connectivity policies. Processor secures the trust domain boundaries by enforcing resource connectivity policy on orchestrator requests and service connectivity policy on workload requests. Workload provides application services with connectivity and service discovery functions in strict accordance with an application communication intent.

Bayware software components in the SIF are as follows:

- BWCTL CLI and BWCTL-API CLI tools (Fabric Manager);
- Policy Controller (Orchestrator nodes);
- Policy Engine (Processor nodes);
- Policy Agent (Workload nodes).

All Bayware software components belong to the SIF control plane only. The application data traverses Linux kernel datapaths that are controlled by Policy Agents and Engines. The Policy Agent programs the Extended Berkeley Packet Filter (eBPF) on its workload node and communicates application communication intent to the desired Policy Engines and Agents. The Policy Engine validates connectivity requests, programs the Open Virtual Switch (OVS) on its processor node, and forwards requests.

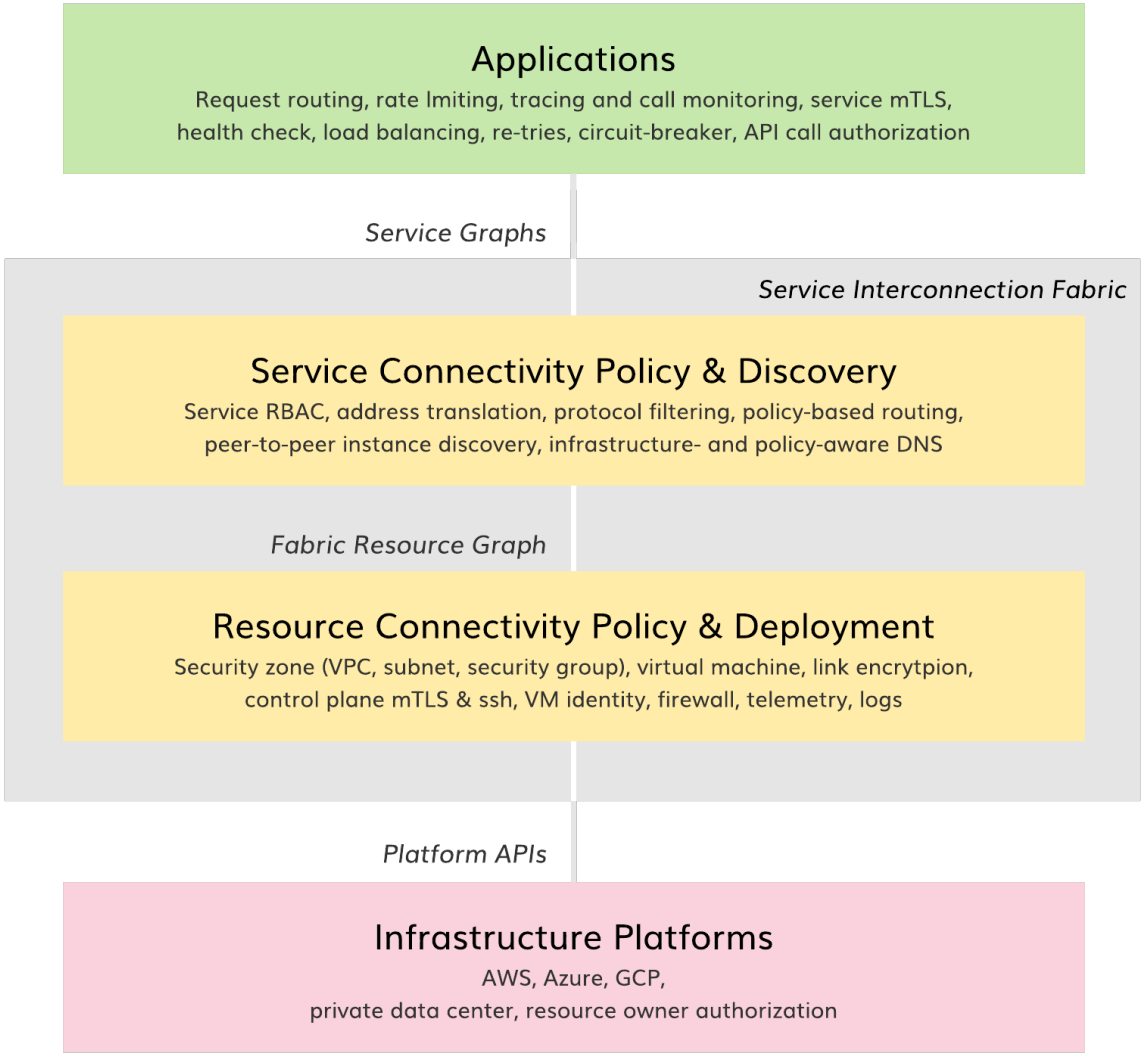


Fig. 1.1: SIF Cloud Software Stack

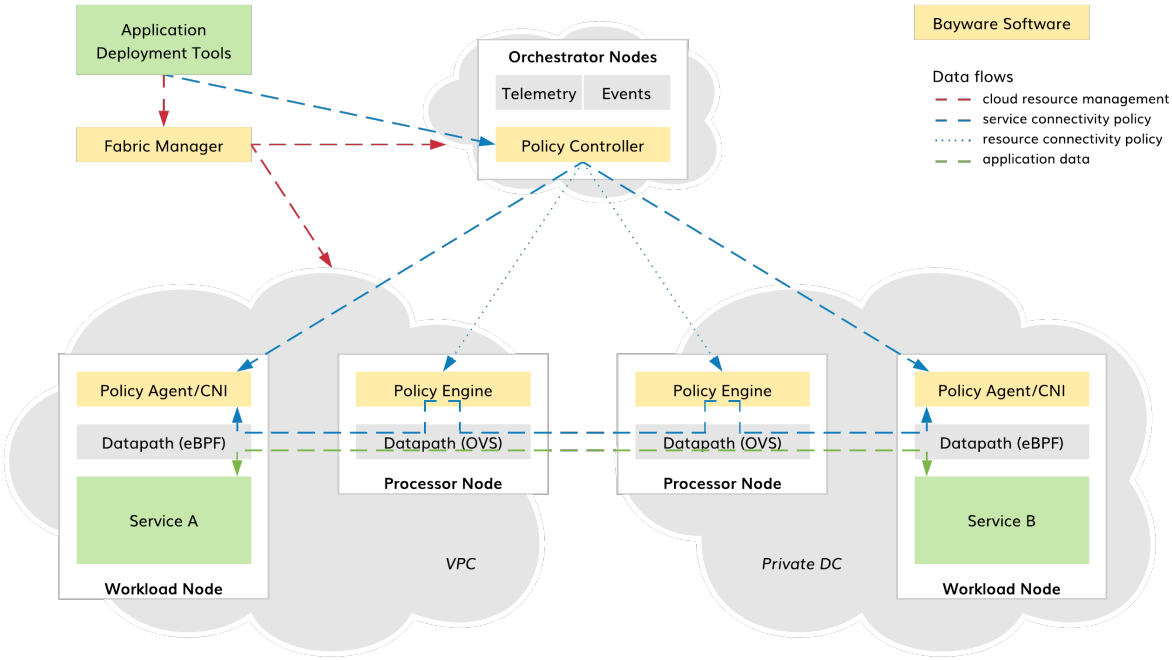


Fig. 1.2: SIF Product Architecture

This unique architecture allows the SIF to automatically generate, configure, and adapt secure connectivity between application services across any set of public, private or hybrid clouds. Application deployment tools communicate application intent to the SIF in a fully infrastructure-agnostic manner, and the SIF takes care of all communication settings: VPCs, subnets, security groups, node firewalls, link encryption, address translation, packet filtering, policy-based routing, and service discovery.

In summary, application connectivity becomes part of the application deployment code. While deploying an application, the required resources and connectivity come up in the SIF automatically. This ensures fast, easy, and secure cloud, multicloud, and hybrid cloud migration.

CHAPTER 2

Managing Cloud Resources

2.1 Rethinking Resource Management

2.1.1 Moving to Clouds

Nowadays, it is just a matter of time before a company on its digitalization path faces a cloud migration project. This endeavour could take many forms:

- moving workloads from a private data center to a public cloud (or back and forth),
- stretching an application across multiple VPCs in the same cloud,
- distributing microservices over various public clouds,
- dispatching workloads from a public cloud to a network edge.

In any of the above mentioned scenarios, the ability to set up, operate and maintain the resource layer of cloud infrastructure in a resilient, secure, and efficient way becomes crucial for the company.

2.1.2 Leaving Configuration Behind

Traditional approaches to the high availability, precise isolation, and maintenance automation of infrastructure resources stop working the very moment these company resources need to operate in a heterogeneous and dynamic environment.

Application services begin communicating over numerous and ever-changing administrative, security, technological, and geographical boundaries. In a traditional paradigm, moving any application service from one walled-garden location to another requires reconfiguration *of several platforms* (data center, public cloud, SD-WAN), *at multiple levels* (computational, network, application), *by multiple teams* (NetOps, SecOps, DevOps, AppDevs).

A long waterfall-defined deployment cycle, wide variety of required skills, and risk of inconsistent policy are among the major drawbacks of bringing old techniques to the cloud world.

2.1.3 Managing Infrastructure as Code

In new realities, the management of the cloud resource layer commonly becomes part of the infrastructure-as-code domain with its declarative language, ability to quickly reproduce deployments, consistency and predictiveness of outcome.

Because the resource layer itself doesn't solve the whole problem of infrastructure setup, operation and maintenance, the cloud resources integrate into a broader cloud infrastructure-as-code stack.

The SIF cloud resource layer utilizes infrastructure platform capabilities, performs specific jobs, and passes abstracted resources to an upper layer as outcome. This approach not only decouples the company infrastructure from various platform implementations, but guarantees **policy consistency with synchronized and instant response to changes across all layers**.

2.2 Resource Deployment

Application services might need to be scaled-out in the same VPC, spread across several VPCs for higher isolation, replicated to a new public cloud for better redundancy, or moved from a test to a production multi-cloud environment. In the SIF, a single resource-copy-and-paste approach enables all of these use cases.

First, the current state of the source VPC, cloud or multi-cloud environment is exported to a file. Next, the resource instance names in the state file are changed to match a target environment. Finally, the current state of the target VPC, cloud or multi-cloud environment is updated with new resources from the state file. It works the same across different clouds, various regions, and multiple VPCs.

The SIF cloud resource layer performs jobs in Azure, AWS, and GCP on the following types of cloud resources:

- VPC,
- gateway,
- subnet,
- security group,
- virtual machine.

Note: It's easy to add new types of managed resources or to support a new cloud platform because the SIF employs *HashiCorp Terraform* to work with infrastructure platform APIs.

After processing, the cloud resources are abstracted in the SIF as follows:

- fabric,
- VPC,
- node (i.e., orchestrator, processor, workload).

By abstracting the underlying infrastructure, the SIF allows a company to manage its cloud resources using **a small set of basic operations: create, show/export, delete**. These operations can be performed on a single node, entire VPC, or multi-cloud deployment (i.e., fabric). With the SIF, the multi-cloud resource deployment becomes reproducible, secure, fast, and simple.

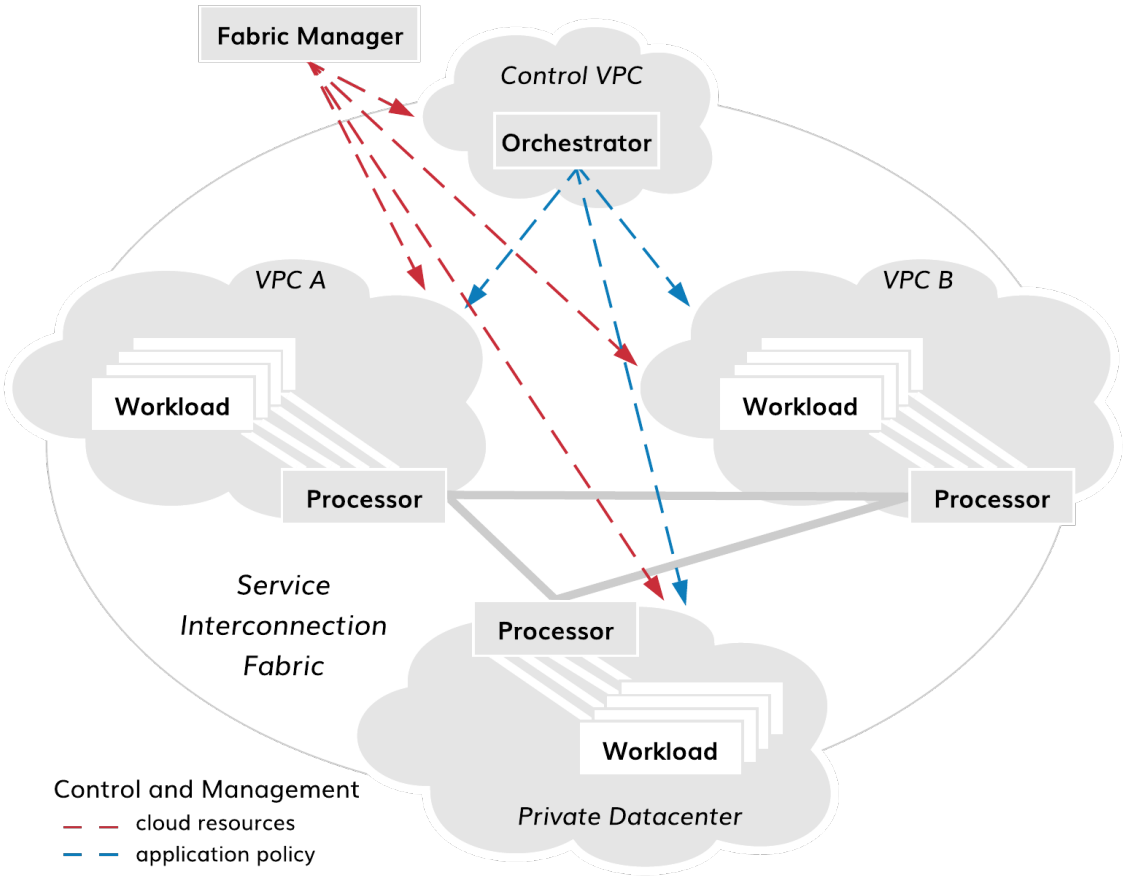


Fig. 2.1: Resource Deployment in SIF

2.3 Maintenance Automation

It is not enough to simply spin up a new virtual machine from a cloud image in order to add it to the company cloud stack for application deployment. The setup of multiple infrastructure services–ssh, PKI, telemetry, logs–often is part of the resource bootstrap. Also, in the course of resource operation, it may be required that the machine change initial settings, restart hosted services, and upgrade software.

While moving to clouds, it is crucial to have in place a maintenance automation tool that allows a company to automatically set up secure access to resources, to provide these resources with infrastructure services, to start/stop services on those resources, and to upgrade software across infrastructure boundaries. No less important is having secure transport between the tool and the distributed resources.

As new resources constantly appear, the maintenance automation tool and multicloud control plane must transport dynamically adapt to changes. The tool and resources may exchange control traffic across public network boundaries, and control flows may terminate in overlapping private IP address spaces. Resource discovery, authentication, and authorization–along with control channel encryption–become necessary components of multicloud maintenance automation.

The SIF offers a complete approach to resource maintenance automation, allowing a company to automatically set up the following infrastructure services:

- SSH access,
- X.509 node certificate,
- control plane mTLS,
- telemetry,
- events,
- software upgrade.

Note: It's easy to add new maintenance procedures or to modify existing maintenance procedures because the SIF employs *RedHat Ansible* for task automation.

Again, all maintenance procedures are executed using **another small set of basic operations: configure, start/stop, update**. Similar to the resource deployment operations, the maintenance can be performed on a single node, entire VPC, or multi-cloud deployment (i.e., fabric).

2.4 Summary

Managing infrastructure resources as code allows a company to quickly generate deployments with predictive outcome in any cloud. With the SIF, cloud resources integrate as a layer into a broader infrastructure-as-code stack, abstracting application communication and computational environment from cloud platforms. As a result, the SIF cloud software stack provides a company with a unified and easy-to-use set of resource management operations–e.g., create, configure, update, delete–across all clouds. As well, the SIF cloud software stack implementation guarantees application policy consistency with synchronized and instant response to changes across all infrastructure layers.

CHAPTER 3

Connectivity Policies

3.1 Layered Security

3.1.1 What is Segmentation?

Wikipedia notes: *"Network segmentation in computer networking is the act or practice of splitting a computer network into subnetworks, each being a network segment. Advantages of such splitting are primarily for boosting performance and improving security."*

Network segmentation has become a widely-used practice since the inception of computer networks and, in recent decades, evolved from the physical separation of Ethernet segments to the splitting of networks into logical IP subnets, and from the filtering of packet flows down to TCP/UDP ports to the mutual authorization of flow endpoints. The ability to restrict communication at the flow level paved the way for the term **microsegmentation**.

Mostly, the network evolution path led to the layering of segmentation techniques, not replacing one with another. On the one hand, the layered approach to segmentation has minimized the risk that a single breach might compromise the entire communication environment. On the other hand, the introduced complexity of managing multiple layers of defense has resulted in packet processing overhead and, much worse, a high risk of getting inconsistent security policy.

In the time of cloud migration and multi-platform application deployment, the approach to network segmentation is being revisited. Stretching VLANs and VPNs across clouds or computational platforms; updating ACLs on hosts and network/cloud firewalls; and employing service discovery mechanisms unaware of service reachability are not viable options for the segmentation any more. Neither is the option to confine the segmentation to a single layer of mTLS tunnels while removing all the network checkpoints between services. Instead, multi-layered segmentation is becoming a part of infrastructure-as-code practice and blending into the application CI/CD pipelines.

3.1.2 Segmentation in Multicloud

Service Connectivity Policy

The approach to microsegmentation in the service interconnection fabric embraces the layered-security concept while enhancing it with a single source of security policy for all segmentation layers. The logical entity called **service graph** fully defines security segments for application services in an abstract, infrastructure-agnostic manner.

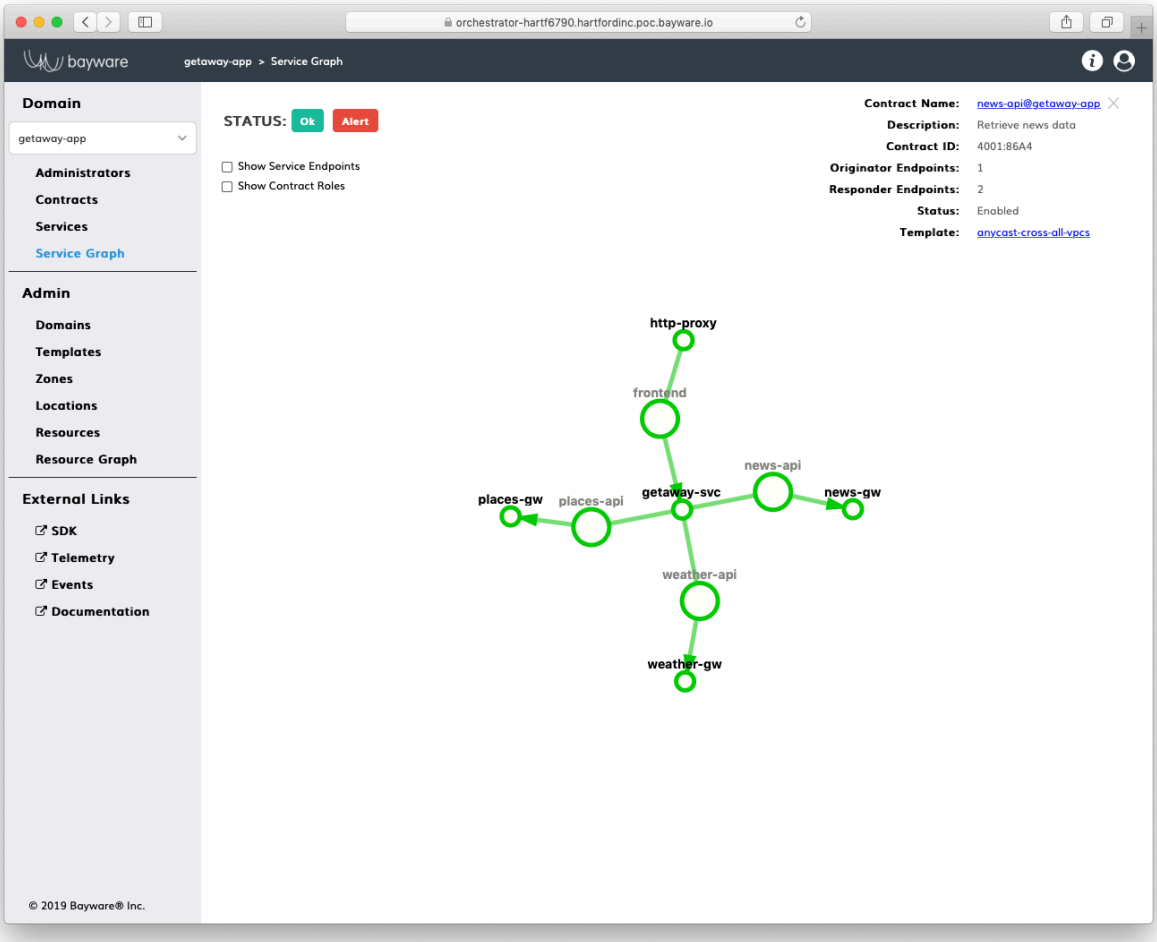


Fig. 3.1: Service Graph

Network connectivity between services, filtering packet flows, and mutual service discovery across various clouds and computational environments (VM and container-based) are all governed in the service interconnection fabric by the application service graph.

Resource Connectivity Policy

Segmentation at the resource layer reinforces the service segmentation. The logical entity called **resource graph** represents abstracted computational resources in the service interconnection fabric.

The resource segmentation isolates each computational resource, e.g. physical server, VM or Kubernetes worker node, from the other and ensures only the application services described in the service graph can reach each other over the top of the resource layer.

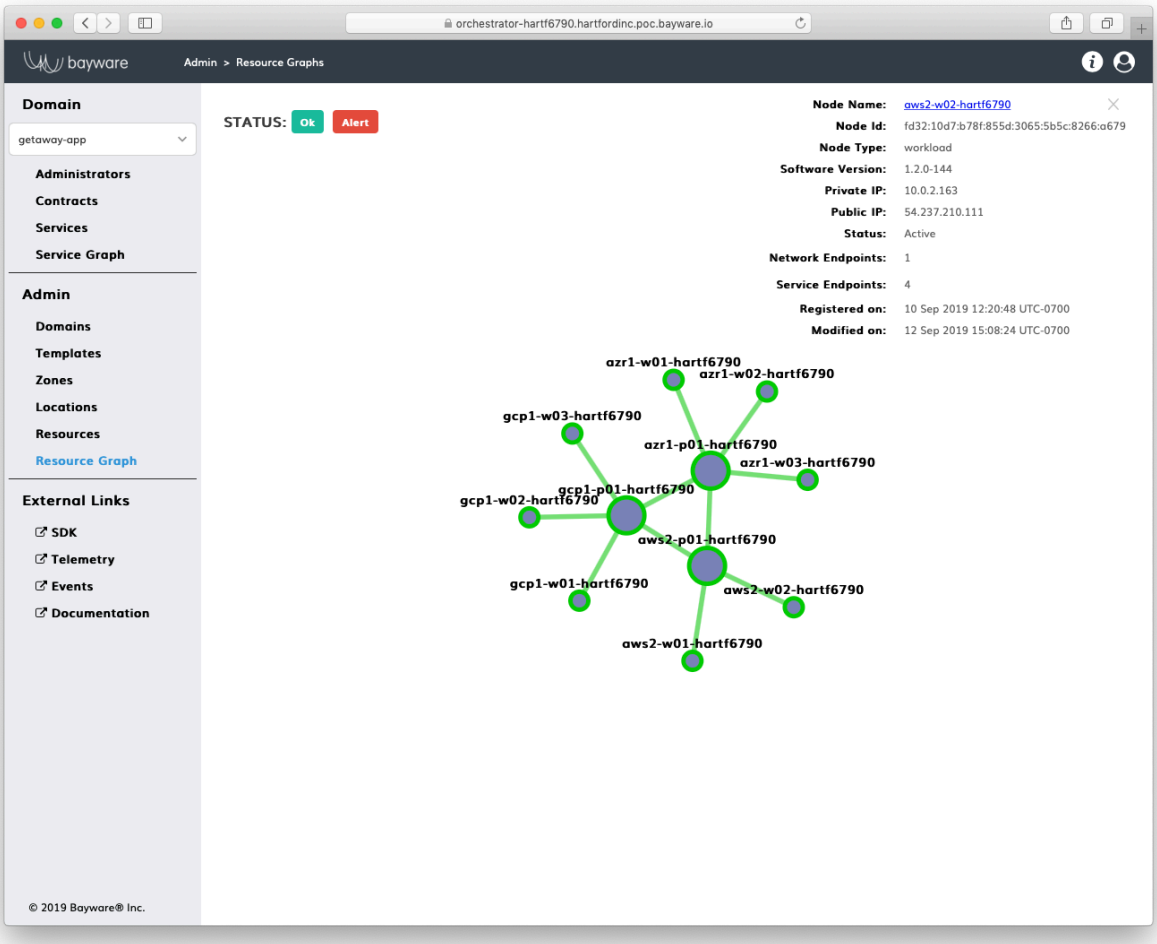


Fig. 3.2: Resource Graph

3.2 Service Connectivity Policy

Microsegmentation in the service interconnection fabric is based on the application service identity and communication intent, as opposed to the use of service IP/MAC addresses and associated routing/switching configuration in traditional networks. This approach allows one to define–all at once and in advance– the security policy for the application. And while the application services might eventually be dispersed across private and public clouds for both VM and container environments, the policy specification remains unchanged because it carries no infrastructure and environment dependencies.

The infrastructure-agnostic security policy uniformly governs application behavior at multiple communication layers. The service identity and communication intent determine:

- Network connectivity – reachability of the service by others in the network;
- Packet filtering – protocols and ports open for packet flows belonging to the service;
- Service discovery – capability of the service to advertise itself and find other services.

The service may have multiple roles, each defining communication intent in a different way. This allows for the service to be exposed in multiple security segments while maintaining connectivity, filtering, and discovery policies in each zone independently. The security segment is a logical entity of the service graph called **contract**. Only the services that become parties in opposite roles in the same contract can communicate with each other.

3.2.1 Network Connectivity

The contract determines IP reachability of one service by another in the service interconnection fabric. Only the services acquiring opposite roles to the same contract can potentially reach each other.

The contract itself might even further restrict the service reachability. As an example, the network connectivity policy might define that the opposite-role services must be within one hop from each other, in other words, in the same VPC.

Another example of the network connectivity policy is unidirectional communication. The policy might define that all services in a given role can receive data from the opposite-role services but are not allowed to send any content into the network. A UDP-streaming service crossing multiple VPCs relies on such a communication pattern.

3.2.2 Packet Filtering

Another part of the segmentation policy covered by the contract is packet filtering. In addition to packet filtering at the protocol and port level, every new packet flow must be cryptographically authorized by every policy processor before opening a connection in the network.

The packet filters at the opposite-role endpoints of the same flow mirror each other. An ingress rule for one endpoint implies an automatically-generated, opposite-role rule. The rules are synchronously applied across all clouds for all service endpoints. Moreover, the ingress and egress rules on both sides of the flow function as stateful firewalls or, more precisely, reflexive ACLs.

Flow authorization happens before protocol and port filtering. It effectively blocks all communication except the packet exchange between the opposite-role services. The flow authorization ensures the flow originator always plays the role assigned by the service graph.

3.2.3 Service Discovery

The application service in the service interconnection fabric is able to discover only the opposite-role services. Moreover, only reachable and already-authorized remote service instances appear in the local service discovery database.

The service discovery segmentation ensures the service at a given workload node can resolve only the opposite-role instances that have been authorized and proved reachable from this particular node.

As an example, the network connectivity policy might specify that communication between services be confined within a single VPC. If two pairs of opposite role service instances are deployed in two separate VPCs, every single service will discover only one instance from its own VPC.

The service discovery segmentation is fully automatic and doesn't require adding any specification to the contract in order to be enacted.

3.3 Resource Connectivity Policy

The resource segmentation in the service interconnection fabric reinforces the service segmentation layer. Splitting computational resources, i.e. workload nodes, into segments offers an additional layer of security for applications running on these nodes and ensures only communication defined by the service graph are present in the fabric.

Each computational resource in the service interconnection fabric possesses an X.509 certificate as a node identity document on which all resource segmentation layers are built:

- Fabric segmentation - sandbox for application deployment with workload nodes isolated from the outside world;
- Zone segmentation - group of workload nodes within the fabric whose inbound and outbound traffic is regulated through policy;
- Workload segmentation - workload node isolation from the nodes in the same group.

The security policy for resource segmentation is infrastructure-agnostic and works the same in all clouds.

3.4 Summary

The service interconnection fabric offers a new approach to network segmentation in public and private clouds for both VM and container environments. The solution is designed to provide high performance and uncompromised security. The segmentation is part of the application deployment process, embedded into infrastructure-as-code rather than coming from a disconnected network configuration system. The single-source, infrastructure-agnostic policy in the form of service identity and communication intent doesn't sacrifice the layered-security approach but governs segmentation across layers in a consistent and real-time manner.

CHAPTER 4

Service Discovery

4.1 Overview

With Service Interconnection Fabric, services may scale in and out across VM and container environments; communicating instances may appear in the same private data center or a thousand miles apart in different clouds; instances may be instantiated in overlapping or even different–IPv4/6–address spaces; and each particular instance-to-instance interaction may have its own policy to support enterprise security or compliance requirements.

In such a dynamic and heterogeneous environment, the DNS service must be intelligent, secure, and scalable. An intelligent DNS reflects infrastructure fluidity at the application level without overburdening applications. A secure DNS ensures only desired communication happens between application services. And a scalable DNS does not have a single point of failure nor does it lock to a particular infrastructure or execution environment.

The DNS service built into Service Interconnection Fabric offers these features without heavy lift-and-shift migration, with practically zero-touch configuration, and with no maintenance required.

4.2 Architecture

Intelligent DNS is a fully distributed system without centralized DNS record management. Each workload node receives its own personalized DNS resolver as a part of Policy Agent functionality.

When the policy controller assigns a communication role to a service instance on a workload node, the workload node starts sending service discovery messages to opposite-role instances to populate their DNS record databases with new entries. The discovery messages, signed by the policy controller, contain an instance Relative Distinguished Name (RDN) and an instance Host Identifier (HID), among other fields.

The RDN identifies an instance service endpoint distinctly in the service interconnection fabric. The policy agent automatically builds the RDN from three components: the host name and host location identifier of the workload node on which the service instance is deployed plus the instance role identifier.

The HID identifies an instance network endpoint distinctly in the service interconnection fabric. The HID is a cryptographically generated address that decouples the application transport layer from the internetworking layer (IP).

The policy agent creates a name resolution record in its database when it receives the service discovery message. The agent forms the name resolution record by augmenting the RDN with the local zone name to form a fully qualified distinguished name (FQDN) and assigning a virtual IP (VIP) to the newly-discovered service instance. Additionally, the policy agent may create two service records: one for the service in a given location and another, more general record, for the service itself.

The policy agent assigns credits to each name resolution record. Records with more credits have a higher priority than records with fewer credits. This mechanism allows the policy agent to redirect a service record to a newly-discovered service instance if the new instance has more credits than the existing instance.

To support zero-touch configuration, the service discovery procedure has both embedded keep-alive and shutdown routines. For example, once a service instance shuts down, the associated records are removed fabric-wide almost immediately.

Intelligent DNS works in private data centers and public clouds for VMs and containers offering seamless and secure application connectivity across various security, technological, geographical, and administrative boundaries.

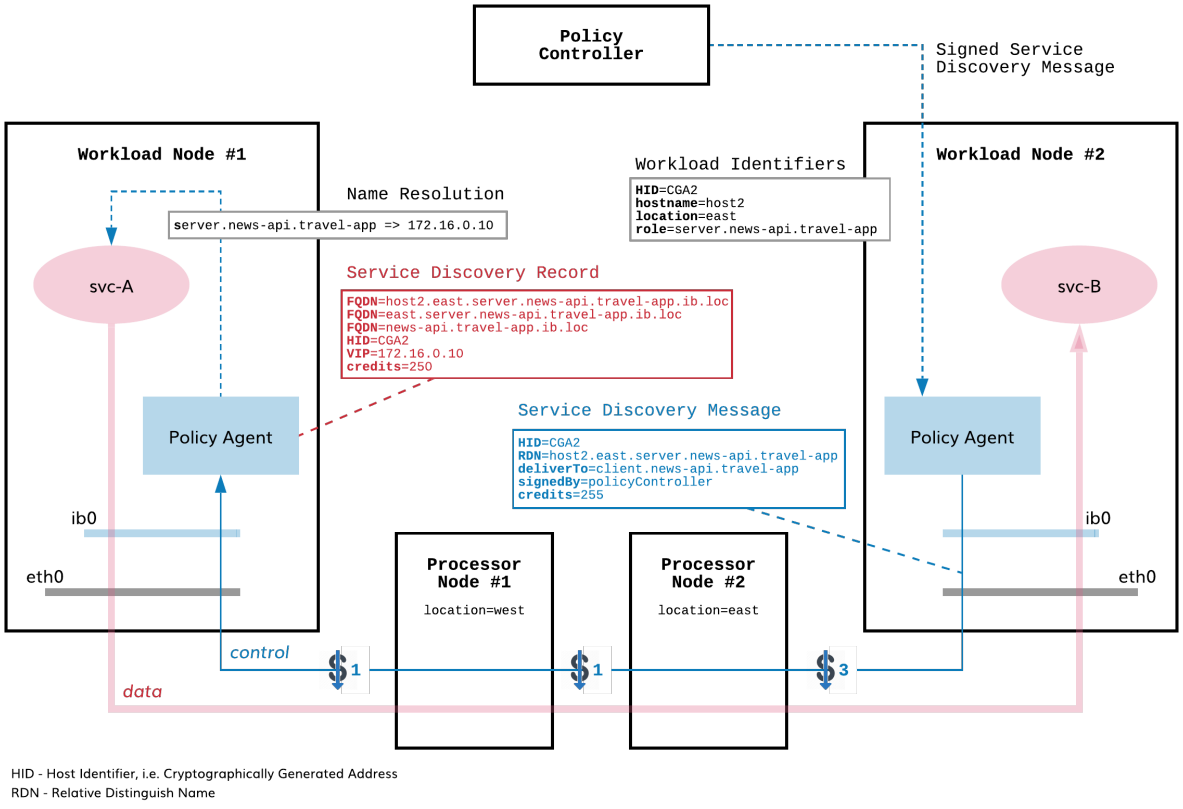


Fig. 4.1: Intelligent DNS Architecture

4.3 Specification

4.3.1 Design Principles

Distributed DNS Secure peer-to-peer service discovery combined with service instance authentication and flow-level microsegmentation interconnects workload node DNS resolvers into a distributed, intelligent DNS system without centralized record management.

Benefit: *Secure, responsive, adjustable, and scalable DNS service with minimal overhead*

Host Identity Services communicate with each other by means of virtual IPs (VIP) that are automatically translated into cryptographically generated host identifiers/addresses (CGA). Communication is built on host identity, not on host locator.

Benefit: *DNS service resolves authenticated hosts only from VIP to CGA*

Secure Service Discovery Workload nodes exchange authorized messages so that each service instance discovers its opposite-role instance.

Benefit: *Only authorized service instances may discover each other*

Personalized Name Space Each workload node places the discovered remote instance names in the zone meaningful to that node only e.g., different zones for private DCs, public cloud, and Kubernetes clusters.

Benefit: *Flexible DNS request routing without environment interdependencies*

Personalized VIP Space Each workload node receives its own personalized address space for VIP allocation.

Benefit: *Communicating services may be in overlapping IP-address spaces*

Personalized Protocol-agnostic Name Resolution Each workload node may switch between IPv4 DNS or IPv6 DNS independently from other nodes (supported in coming version).

Benefit: *Automatic translation between IPv4 and IPv6*

4.3.2 Interface to Applications

Automatic FQDN Builder Each service instance receives a unique FQDN (within the service interconnection fabric) built automatically from host identifier, location identifier, and instance role configuration.

Benefit: *Zero FQDN provisioning*

Alternative Names A service instance can have multiple FQDNs, each corresponding to a different communication role.

Benefit: *Supports flow-level communication policy*

Role-based Access Control Each service instance may propagate only its own role instance names and resolve only opposite-role instance names.

Benefit: *Protection from unsanctioned service discovery*

Instance Affinity Service instances can query DNS using opposite-role instance affinity: `role.namespace`, `location.role.namespace`, `host.location.role.namespace`.

Benefit: *Automatic request re-routing*

Instance proximity DNS records are prioritized based on opposite-role instance proximity to the local service instance. Proximity is defined as the cost of the path between two workload nodes. Cost can be dynamically assigned to each link in service interconnection fabric by an external system and can reflect interconnection quality, instance load, traffic cost, etc.

Benefit: *Infrastructure-aware request routing*

Responsiveness to Connectivity Failure A heartbeat allows detection of remote instance connectivity failure.

Benefit: *Minimize application downtime*

Responsiveness to Instance Failure On instance shutdown, corresponding DNS records are removed fabric-wide almost immediately.

Benefit: *Minimize application downtime*

REST API for Retrieving Local DNS Records The policy agent RESTful API allows for retrieving all records from the DNS resolver database on each workload node.

Benefit: *Open for integration with workload orchestrators*

4.3.3 Deployment and Maintenance

Cloud-agnostic The DNS service is embedded in an infrastructure-agnostic service interconnection fabric so it does not require services specific to any cloud or private data center.

Benefit: *Not locked to any cloud infrastructure*

Supports Both VM and Kubernetes Environments On each workload node, the DNS service is exposed to service instances as a `libnss` library (via `nsswitch.conf` on VMs) or DNS resolver (via CoreDNS on a Kubernetes worker node).

Benefit: *Consistent DNS service across VM and container environments*

Transparent to Applications The DNS service requires no changes to applications apart from updating URLs and then forwarding requests to the service interconnection fabric DNS.

Benefit: *Minimal changes to applications*

DNS Resolver Policy Agent Component DNS resolver installation is part of policy agent deployment.

Benefit: *Minimal footprint, no additional maintenance*

CLI for DNS Service Adjustment The policy agent CLI allows for setting up name and VIP spaces on each workload node.

Benefit: *Minimal or no configuration*

Seamless Migration The DNS service does not interrupt or degrade performance of existing DNS services on workload nodes, allowing for a gradual and automatic migration as service instance communication is switched from traditional networking to the service interconnection fabric.

Benefit: *No lift-and-shift migration required*

CHAPTER 5

Security Model

5.1 Isolation Levels

5.1.1 Concept

With the service interconnection fabric (SIF), organizations can divide application services distributed across private and public clouds into isolated domains without network configuration and irrespective of geographic location, virtualization platform, and IP/Ethernet addresses. Services may be running on physical servers or VMs or containers.

IT teams can explicitly define micorsegments (contracts), specifying interaction patterns, firewall rules, and data pathways. Each permission set within a contract is labeled as a distinct role. These roles, then, can be attached to any application service.

As an application service instance binds to an endpoint on a workload node, the endpoint dynamically obtains permissions based on the service role. In this way, the service instance will be able to establish authorized connectivity with the policy set not only at the network endpoint, but through the entire infrastructure.

When the service instance initiates a new flow, **every processor en route verifies permissions** linked to the service instance role before forwarding packets belonging to the flow. With the SIF, communication policy preserves uniformity over different virtualization environments and across private and public clouds while allowing for personalized forwarding, path protection, and load balancing down to the flow level.

The service-level segmentation is reinforced with the resource segmentation and separation of administration duties. As such, the SIF security model relies on three levels of isolation: fabric, domain, and contract.

5.1.2 Fabric-level Isolation

A fabric isolates all its resources in their own fault domain. Resources within a fabric share a policy controller—usually deployed for high-availability and scalability as a plurality of nodes—dedicated to that fabric. The policy controller does not communicate with resources outside of the fabric.

5.1.3 Domain-level Isolation

Within a fabric, administrative responsibilities are isolated from each other in domains. Multiple applications, for example, may be running on resources in a single fabric. However, the services and contracts for one application may be administratively isolated from the services and contracts for another application by placing them in distinct domains. By doing this, a unique administrative account manages services and contracts for its domain only.

5.1.4 Contract-level Isolation

Resources (workloads) for a given application (isolated within a domain) receive contracts from a policy controller (isolated within a fabric) that dictate allowed paths of communication. These contracts act as waypoints between service endpoints (application microservice instances) running on workloads that determine who can talk to whom. This level of micro-segmentation ensures that no data passes between workloads without authorization from the policy controller.

5.2 Isolation Topology

Fig. 5.1 graphically demonstrates the three levels of isolation as they relate to an application.

- **Resource Graph**: represents fabric-level isolation containing all workload nodes and processor nodes that are under the auspices of the same policy controller
- **Service Graph**: represents domain-level isolation containing services (service graph nodes) within an application that communicate with each other
- **Flow Graph**: represents contract-level isolation containing policy (service graph edges) that allows one service instance to communicate with another service instance

Table 5.1 summarizes the relationship between the Fabric, Domain, and Contract abstraction layers, their graphical representation, and constituent elements.

Table 5.1: Security Model Abstraction Layers

No.	Abstraction Layer	Representation	Node
1	Fabric	Resource graph	Resource
2	Domain	Service graph	Service
3	Contract	Flow graph	Endpoint

5.3 Security Entities

Two APIs facilitate communication with the policy controller: Northbound API and Southbound API.

5.3.1 Northbound API

The Northbound API provides a mechanism for automation systems to interact with the policy controller. Tasks that can be performed via the policy controller GUI can also be performed programmatically through the Northbound API.

A user will typically use this interface extensively when automating tasks.

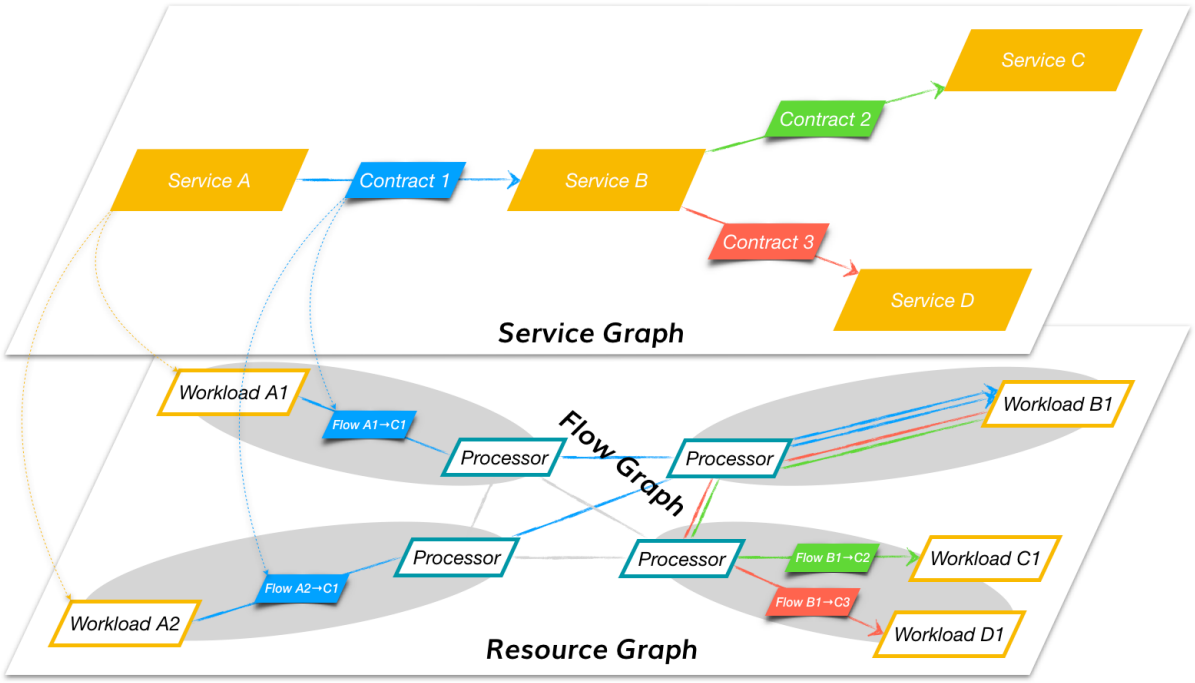


Fig. 5.1: Security Isolation Levels

5.3.2 Southbound API

The Southbound API provides a mechanism for the policy engines and policy agents to communicate with the policy controller.

Communication on this interface is fully automated. A user will not directly interact with it.

As shown in figure below, there are four security entities: Administrator Credentials, Resource Certificate, Service Token, and Flow Signature.

Resource Certificate

Each virtual machine in a given fabric uses an X.509 certificate to verify identity within a fabric. Hence, a policy agent or policy engine can use credentials to establish communication with a policy controller only if the policy agent or policy engine is running on a virtual machine recognized by the policy controller via its X.509 certificate.

Once node identity is established, the X.509 certificate is used to

- create a secure communication control channel with the policy controller using mTLS
- create a secure communication data channel using IPsec
- find neighboring nodes using SeND (Secure Neighbor Discovery) protocol
- generate network endpoint identity using CGAs (Cryptographically Generated Addresses)

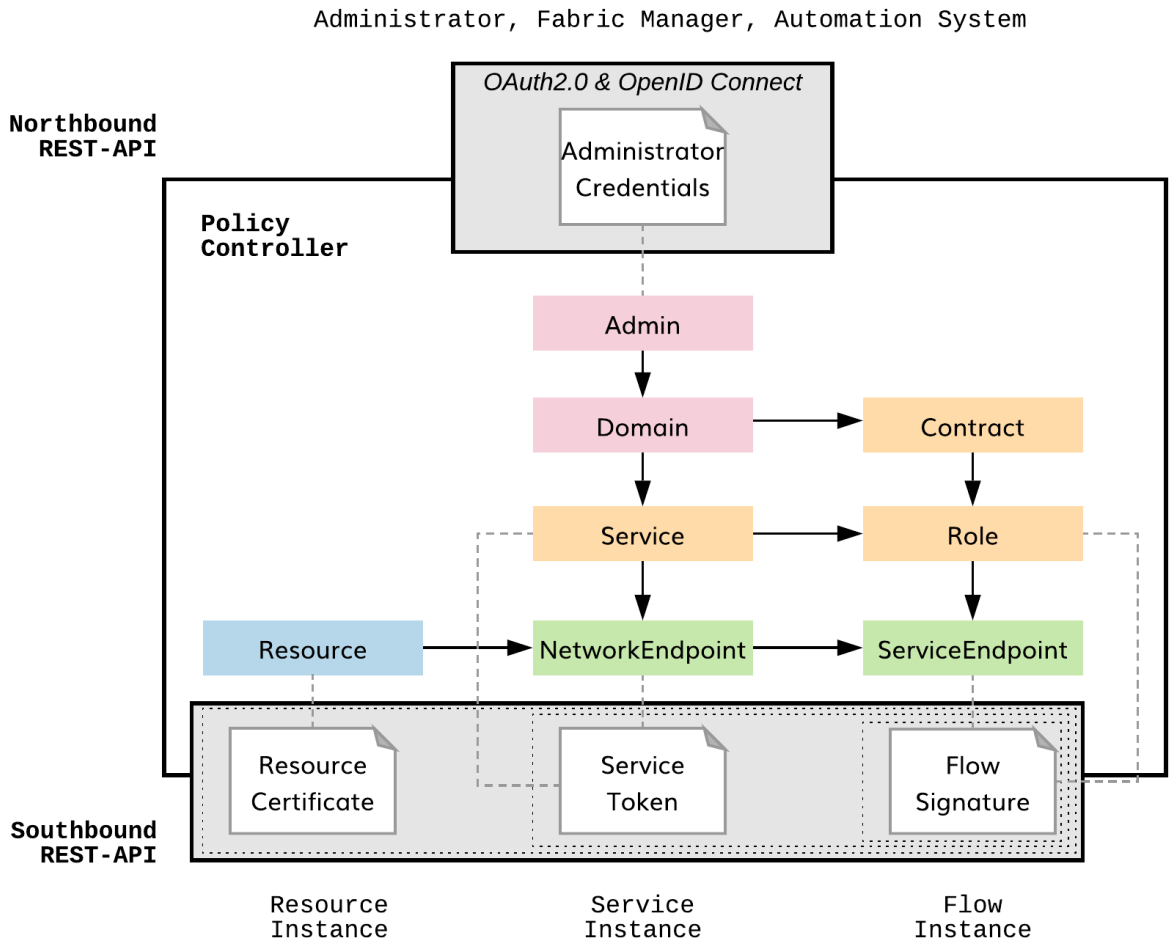


Fig. 5.2: Security Model

Service Token

If the Credential authorizes a policy agent to communicate with the policy controller and the Certificate establishes the identity of a node that can operate within a fabric, then a Token authorizes a service to run on an identified node with an authorized policy agent. When a service is authorized to run on a network endpoint, the corresponding policy agent creates one or more service endpoints.

The domain administrator assigns roles associated with one or more contracts to a service created at the policy controller. An authorized policy agent at a network endpoint creates a service endpoint for each role assigned by the domain administrator to that service.

The domain administrator generates a token at the policy controller for a given service. The token consists of a 128-bit key and a 128-bit value that form a key:value pair. The policy controller stores the key and a hash of the value only. The domain administrator then authorizes a service at a network endpoint by attaching the entire 256-bit key:value pair to a policy agent, which then queries the policy controller. If the policy controller recognizes the key:value pair, it passes contract role information to the policy agent to establish service endpoints. Recall that the contract role dictates how and to whom the service endpoint can communicate.

Best practices necessitate that administrators treat tokens as short-lived security entities, rotating them often and assigning them uniquely to service instances.

Flow Signature

Upon creating a service endpoint for a given contract role, the policy agent requests that the policy controller send the contract policy to the policy agent in the form of a microprogram. This microprogram is cryptographically bound to the service endpoint that is operating at the given network endpoint using an ECDSA signature that covers IPv6 control packet headers. The IPv6 control packet headers carry the microprogram in an extension header along with network endpoint (IPv6 SA), service endpoint (IPv6 FL), and contract ID (IPv6 DA). Policy engines within the fabric use the policy to create flow state to interconnect authorized service endpoints.

5.4 Summary

The security model of the service interconnection fabric relies on three levels of isolation and four security entities. Fabric-level isolation confines resources to a single policy controller; domain-level isolation provides for administrative isolation among applications; and contract-level isolation inserts policy waypoints along communication paths between service endpoints.

These isolation levels are enforced using credentials that authorize administrators or automation systems to communicate with the policy controller. Certificates corroborate the identity of nodes within a service interconnection fabric that ultimately provides for a secure control channel (mTLS), secure data channel (IPsec), secure neighbor discovery (SeND), and secure network endpoint identifiers (CGA). Tokens ensure that only authorized service endpoints communicate through contracts over the service interconnection fabric. Finally, ECDSA signatures provide for unaltered delivery of inband policy information to policy engines.

CHAPTER 6

Introduction

6.1 Overview

The purpose of this guide is to lead you through four steps in creating your service interconnection fabric, such as:

- deploying network and computational resources for your application in public clouds,
- configuring interconnection policy for cloud resources,
- configuring interconnection policy for application services,
- deploying application services on cloud resources.

As a result of following the prescribed procedures, your application services will be secured from each other and the outside world, able to automatically discover each other, tolerant to cloud failures, and easily portable across private and public clouds.

You will achieve this by representing cloud resources and application services–through **resource graph** and **service graph** correspondingly–and making your resource and service segmentation policy fully infrastructure-agnostic.

6.2 Fabric Components

You will deploy your service interconnection fabric using Bayware components as follows:

- Fabric Manager,
- Orchestrator Node,
- Processor Node,
- Workload Node.

You manage your application's infrastructure via the **fabric manager**. Included with the fabric manager are two command-line-interface tools: BWCTL and BWCTL-API. The former allows you to easily manage

cloud resources and the latter to control resource and application policy. Instead of developing and deploying numerous cloud-specific policies, you create infrastructure-agnostic policy once and copy-paste it across private and public clouds, multiple VPCs in the same public cloud, or various public clouds.

The **orchestrator** is a unified point for the resource and application policy management. It might be initially deployed as a single node–offering policy controller functionality only–and later enhanced with telemetry and events nodes. Placing all of these components together, you receive a single source for creating and managing all layers of security for your application in multicloud environment as well as in-depth metrics and detailed logs to easily see at a glance the status of your application infrastructure in its entirety.

At a high level, the **processor** is an infrastructure-as-code component that secures data and control flows between application services in a multicloud environment. The processor plays multiple roles: ssh jump-host, microsegmentation firewall, and inter-VPC VPN gateway among others. However, the most remarkable processor feature is the direct execution of your security policy without requiring any configuration. You can install the processor policy engine with all its dependencies on any VM or physical server and have it serving application traffic in a minute.

Each application **workload** node–either VM or Kubernetes worker–runs *policy agent*, a software driver that connects a workload to your service interconnection fabric. The agent manipulates eBPF programs–which process each and every packet coming to and from your application–all at the interface level. Additionally, the agent has an embedded cross-cloud service discovery mechanism to serve DNS and REST requests from the applications located on the node. The agent deployment and autoconfiguration takes just another minute.

6.3 Behind the Scene

The declarative language of BWCTL and BWCTL-API command-line tools abstracts all the specifics of resource deployment and security policy management in hybrid cloud and multicloud environments. The tools allow you to interact with cloud provider APIs, manage virtual machines and set up security policy. So, a lot of activities happen in the background when you just type, for example,

```
$ bwctl create processor <vpc-name>
$ bwctl-api create link -s <source-processor> -t <target-processor>
```

Here is a brief outline of what happens behind the scene at each stage of service interconnection fabric deployment.

(1) Creating fabric

- Setting up certificate authority
- Setting up Terraform state
- Setting up Ansible inventory
- Setting up ssh transport in jump-host configuration

(2) Creating VPC

- Creating virtual network
- Creating subnets
- Creating security groups

(3) Deploying orchestrator

- Creating VM with appropriate firewall rules
- Setting up policy controller containers

- Deploying InfluxDB/Grafana for telemetry and ELK for events
- Creating DNS records and certificates

(4) Deploying processor or workload

- Creating VM with appropriate firewall rules
- Deploying policy engine or agent
- Deploying Telegraf for telemetry and Filebeat for events
- Deploying certificate for mTLS channel with orchestrator

(5) Setting up and interconnecting security zones

- Assigning processors and workloads to security zones
- Connecting processors
- Setting up IPsec encryption between processors and workloads

(6) Uploading communication rules and creating service graph

- Installing templates
- Setting up domain for application policy
- Specifying contracts by altering templates
- Assigning application services to contracts

(7) Deploying application

- Authorizing service endpoints with tokens
- Authorizing application packets at source and destination workloads and all transit processors
- Discovering service endpoints and auto-configuring local DNS

6.4 Summary

In the next four steps you will create an environment for multicloud application deployment in which the infrastructure is just part of your application code and blends into application CI/CD process. You don't need to configure networks and clouds in advance in order to deploy application services. And when you move services, the policy follows them. Also, a single source for your multilayered security policy ensures there are no gaps or inconsistency in the application defense.

CHAPTER 7

Deploy Resources

7.1 Set up Fabric

7.1.1 Spin up Fabric Manager

The first thing you will need to do is to create and/or choose a VPC for your Fabric Manager deployment. Next, create a VM in this VPC using the Bayware Multicloud Service Mesh image from the Marketplace.

To quickly start from the Azure marketplace offering, simply search for Bayware, and click on the "Get It Now" button to begin the download.

7.1.2 Update BWCTL CLI Tool

Upon successfully completing the creation of the new VM image, it is time to update all necessary packages and dependencies for BWCTL. To do this, you will need to `SSH` into your newly created VM and switch to root level access to update all packages as such:

```
]$ sudo su -
```

Next, to update BWCTL, run the command:

```
]# pip3 install --upgrade bwctl
```

To update the BWCTL-resources package, run the command:

```
]# pip3 install --upgrade bwctl-resources
```

To exit from the current command prompt once you have completed updating, run the command:

```
]# exit
```

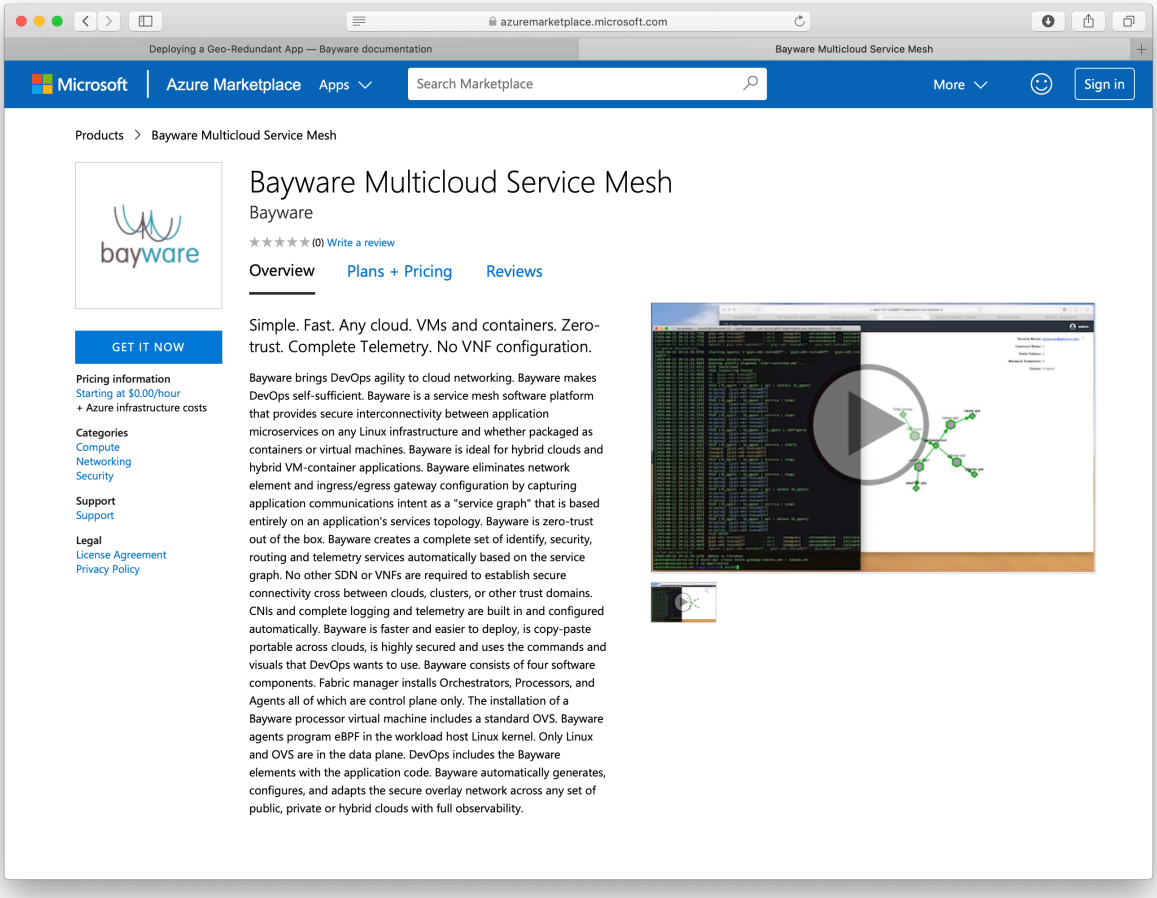


Fig. 7.1: Fig. Azure Fabric Manager marketplace offering

Configure BWCTL

Next, it's time to create the BWCTL environment in the home directory of the current user (`ubuntu`).

First, start BWCTL running the command:

```
]$ bwctl init
```

You should see this output:

```
[2019-09-25 17:30:12.156]  Welcome to bwctl initialization
[2019-09-25 17:30:12.156]  Fabric manager
[2019-09-25 17:30:12.156]   Company name (value is required):
```

In interactive mode, provide all required values when prompted.

Note: Press `<Enter>` to accept the default values.

After the initialization you should have a configuration similar to:

```
[2019-09-25 17:30:12.156]  Welcome to bwctl initialization
[2019-09-25 17:30:12.156]  Fabric manager
[2019-09-25 17:30:12.156]   Company name (value is required): myorg3
[2019-09-25 17:30:30.113]  Global
[2019-09-25 17:30:30.113]   Cloud providers credentials file [~/.bwctl/credentials.yml]:
[2019-09-25 17:30:34.004]   DNS hosted zone (value is required): poc.bayware.io
[2019-09-25 17:30:37.325]   Debug enabled [true]:
[2019-09-25 17:30:42.062]   Production mode enabled [true]:
[2019-09-25 17:30:44.548]   Marketplace images to be used [false]:
[2019-09-25 17:30:48.624]  Components
[2019-09-25 17:30:48.624]   Family version [1.2]:
[2019-09-25 17:30:51.959]  Cloud storage
[2019-09-25 17:30:51.959]   Store bwctl state on AWS S3 [false]:
[2019-09-25 17:30:58.786]   Store terraform state on AWS S3 [true]:
[2019-09-25 17:31:05.633]    AWS S3 bucket name [terraform-states-sandboxes]:
[2019-09-25 17:31:12.933]    AWS region [us-west-1]:
[2019-09-25 17:31:15.876]  SSH keys
[2019-09-25 17:31:15.876]   SSH Private key file []:
[2019-09-25 17:31:21.268]  Configuration is done
```

To view the file with your cloud provider credentials, cat to where the cloud `credentials.yml` file was specified during the initialization by running the command with the path to the file–in this example `/home/ubuntu/.bwctl/credentials.yml` –as argument:

```
]$ cd /home/ubuntu/.bwctl/credentials.yml
```

You should see this output:

```
---

# Add cloud-provider credentials that will be used when creating
# infrastructure and accessing repositories.

aws:
```

(continues on next page)

(continued from previous page)

```
  # In the AWS console, select the IAM service for managing users and keys.
  # Select Users, and then Add User.  Type in a user name and check
  # programmatic access.  Users require access to EC2, S3, and Route53.
  # Copy and paste the secret access key and key ID here.
  aws_secret_access_key:
  aws_access_key_id:
azr:
  # Azure provides detailed steps for generating required credentials
  # on the command line, which you can find at this URL:
  # https://docs.microsoft.com/en-us/azure/virtual-machines/linux/terraform-install-
↪configure#set-up-terraform-access-to-azure
  azr_client_id:
  azr_client_secret:
  azr_resource_group_name:
  azr_subscription_id:
  azr_tennant_id:
gcp:
  # Google uses a GCP Service Account that is granted a limited set of
  # IAM permissions for generating infrastructure.  From the IAM & Admin
  # page, select the service account to use and then click "create key"
  # in the drop-down menu on the right.  The JSON file will be downloaded
  # to your computer.  Put the path to that file here.
  google_cloud_keyfile_json:
```

Use your editor of choice (ex: vim, nano) to add your public cloud credentials to `credentials.yml`.

Create Fabric

The next step is to create a fabric. The fabric acts as a namespace into which your infrastructure components will be deployed.

Note: The fabric manager allows you to create multiple fabrics to isolate various applications or different environments.

To get started, SSH into your Fabric Manager VM and enter the BWCTL command prompt:

```
]$ bwctl
```

You should be at the `bwctl` prompt:

```
(None) bwctl>
```

Now, to create a new fabric, run the command with your fabric name–in this example `myfab2` –as the argument:

```
(None) bwctl> create fabric myfab2
```

You should see output similar to:

```
[2019-09-25 17:33:24.563]  Creating fabric: myfab2...
...
[2019-09-25 17:33:29.901]  Fabric 'myfab21' created successfully
```

To configure the fabric, run the command with your organization name–in this example `myorg2` –as the argument:

```
(None) bwctl> configure fabric myfab2
```

You should see output similar to:

```
[2019-09-25 17:34:29.730]  Install CA for fabric 'myfab2'
...
[2019-09-25 17:34:36.859  Fabric 'myfab2' configured successfully
```

To verify the new fabric has been created with the argument provided, run the command:

```
(None) bwctl> show fabric
```

You should see output similar to:

```
[2019-09-25 17:35:50.356]  Available fabrics listed.  Use "bwctl set fabric FABRIC_NAME"␣
↪to select fabric.
  FABRIC
  myfab2
```

Now, set BWCTL to the new fabric by running this command:

```
(None) bwctl> set fabric myfab2
```

You should see output similar to:

```
[2019-09-25 17:36:22.476]  Active fabric: 'myfab2'
```

Notice that your `bwctl` prompt has changed, now showing the active fabric:

```
(myfab2) bwctl>
```

7.2 Create Orchestrator

Now that you have setup your Fabric Manager, and created a new Fabric, it's time to create a VPC with Policy Orchestrator Nodes.

7.2.1 Create VPC

Once you are in the BWCTL command prompt, show a list of available VPC regions by running this command:

```
(myfab2) bwctl> show vpc --regions
```

You should see the list of the regions, in which you can create your VPC, similar to:

```
aws:
 ap-east-1
 ap-northeast-1
 ap-northeast-2
 ap-south-1
 ap-southeast-1
 ap-southeast-2
 ca-central-1
 eu-central-1
 eu-north-1
 eu-west-1
 eu-west-2
 eu-west-3
 sa-east-1
 us-east-1
 us-east-2
 us-west-1
 us-west-2
azr:
 australiaeast
 australiasoutheast
 brazilsouth
 canadacentral
 centralindia
 centralus
 eastasia
 eastus
 eastus2
 japaneast
 northcentralus
 northeurope
 southcentralus
 southeastasia
 southindia
 westcentralus
 westeurope
 westus
 westus2
gcp:
 asia-east1
 asia-east2
 asia-northeast1
 asia-northeast2
 asia-south1
 asia-southeast1
 australia-southeast1
 europe-north1
 europe-west1
 europe-west2
 europe-west3
 europe-west4
 europe-west6
```

(continues on next page)

(continued from previous page)

```
northamerica-northeast1
southamerica-east1
us-central1
us-east1
us-east4
us-west1
us-west2
```

Now, to create a new VPC for orchestrator nodes, run the command with the cloud and region names–in this example `azr` and `westus`, respectively, as an argument:

```
]$ bwctl> create vpc azr westus
```

You should see output similar to:

```
[2019-09-25 17:36:58.649]  Creating VPC: azr1-vpc-myfab2...
...
[2019-09-25 17:38:26.089]  VPCs ['azr1-vpc-myfab2'] created successfully
```

Note: The VPC name has been autogenerated. Use this name from the command output at the next step.

7.2.2 Create Controller Node

To create a controller node for the orchestrator, run this command with the orchestrator VPC name–in this example `azr1-vpc-myfab2` –as argument:

```
(myfab2) bwctl> create orchestrator controller azr1-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 17:39:48.091]  Creating new orchestrator 'azr1-c01-myfab2'...
...
[2019-09-25 17:43:56.811]  ['azr1-c01-myfab2'] created successfully
[2019-09-25 17:43:56.840]  Generating SSH config...
```

Note: The orchestrator node name has been autogenerated. Use this name at the next step.

Next, configure the orchestrator node by running this command with the orchestrator node name–in this example `azr1-c01-myfab2` –as argument:

```
(myfab2) bwctl> configure orchestrator azr1-c01-myfab2
```

You should see output similar to:

```
[2019-09-25 17:44:38.177]  Setup/check swarm manager on orchestrator 'azr1-c01-myfab2'
...
[2019-09-25 17:50:14.166]  Orchestrators: ['azr1-c01-myfab2'] configured successfully
```

(continues on next page)

(continued from previous page)

```
[2019-09-25 17:50:14.166]  IMPORTANT: Here is administrator's password that was used to␣
↪initialize controller. Please change it after first login
[2019-09-25 17:50:14.166]  Password: RWpoi5RkMDBi
```

Note: Be sure to write down the PASSWORD as it appears on your screen, it will be needed later.

To login to the orchestrator, you will use the FQDN of orchestrator northbound interface (NBI).

The FQDN of orchestrator NBI has been auto-generated on the prior step and in this example has the structure as follows:

```
orchestrator-myfab2.myorg2.poc.bayware.io
```

Note: The FQDN of orchestrator NBI is always defined in the following manner: `orchestrator-<fabric>.<company>.<DNS hosted zone>` wherein `company` and `DNS hosted zone` are from the fabric management configuration and same for all fabrics.

Authenticate into the orchestrator via a web browser and use the following information:

- Orchestrator URL - **FQDN of orchestrator NBI**,
- Domain - **default**,
- Username - **admin**,
- Password - **PASSWORD** from the prior step.

7.3 Create Processor and Workload

The next step is very similar to step 3 where you created a VPC with Policy Orchestrator Nodes, however, now you must create the Processor and Workload nodes within a new VPC. Be aware that some of the commands may appear very similar to prior commands, however they do have different consequences.

7.3.1 Create VPC

To create a new VPC for application deployment, with the cloud and region names– in this example `azr` and `westus`-as an argument:

```
(myfab2) bwctl> create vpc azr westus
```

You should see output similar to:

```
[2019-09-25 17:51:51.688]  Creating VPC: azr2-vpc-myfab2...
...
[2019-09-25 17:52:50.803]  VPCs ['azr2-vpc-myfab2'] created successfully
```

7.3.2 Create Processor Node

Next, to create a processor, run the command with the target VPC name as an argument:

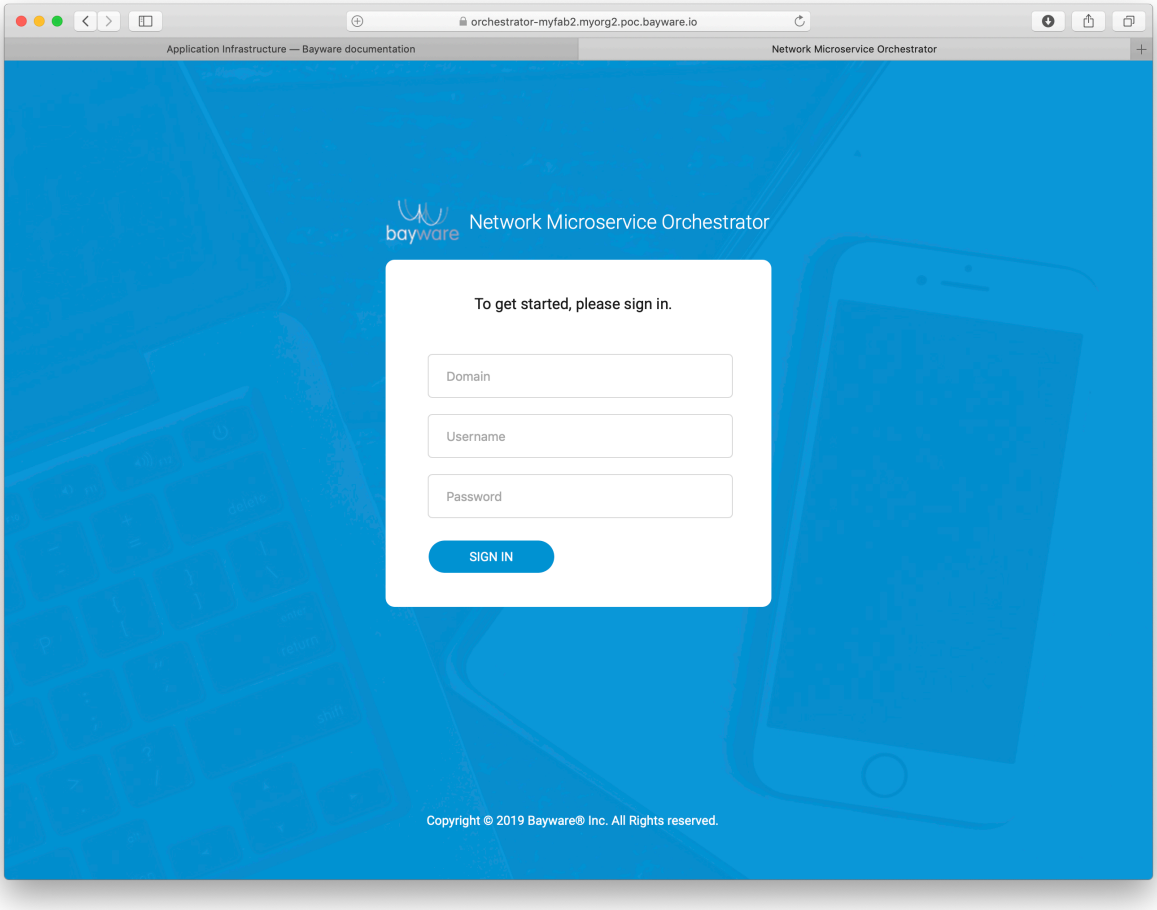


Fig. 7.2: Orchestrator Login Page

```
(myfab2) bwctl> create processor azr2-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 17:53:22.613]  Creating new processor 'azr2-p01-myfab2'...
...
[2019-09-25 17:57:27.735]  ['azr2-p01-myfab2'] created successfully
[2019-09-25 17:57:27.763]  Generating SSH config...
```

To configure the processor, you will use the FQDN of orchestrator southbound interface (SBI).

The FQDN of orchestrator SBI has been auto-generated on the prior step and in this example has the structure as follows:

```
controller-myfab2.myorg2.poc.bayware.io
```

Note: The FQDN of orchestrator SBI is always defined in the following manner: **controller-<fabric>.<company>.<DNS hosted zone>**

To configure the processor, run the command with the FQDN of orchestrator SBI – in this example `controller-myfab2.myorg2.poc.bayware.io` as an argument:

```
(myfab2) bwctl> configure processor azr2-p01-myfab2 --orchestrator-fqdn controller-
↪myfab2.mayorg2.poc.bayware.io
```

You should see output similar to:

```
[2019-09-25 17:58:58.573]  Generate ansible inventory...
...
[2019-09-25 18:00:18.506]  Processors ['azr2-p01-myfab2'] configured successfully
```

To start the processor, run the command:

```
(myfab2) bwctl> start processor azr2-p01-myfab2
```

You should see output similar to:

```
[2019-09-25 18:00:44.719]  Processors to be started: ['azr2-p01-myfab2']
...
[2019-09-25 18:00:47.537]  Processors ['azr2-p01-myfab2'] started successfully
```

7.3.3 Create Workload Node

Now create a new workload in the current VPC, run the command:

```
(myfab2) bwctl> create workload azr2-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 18:03:26.462]  Creating new workload 'azr2-w01-myfab2'...
...
```

(continues on next page)

(continued from previous page)

```
[2019-09-25 18:06:24.269]  ['azr2-w01-myfab2'] created successfully
[2019-09-25 18:06:24.297]  Generating SSH config...
```

To configure the workload, run the command with the FQDN of orchestrator SBI – in this example `controller-myfab2.myorg2.poc.bayware.io` as an argument:

```
(myfab2) bwctl> configure workload azr2-w01-myfab2 --orchestrator-fqdn controller-myfab2.
↪myorg2.poc.bayware.io
```

You should see output similar to:

```
[2019-09-25 18:07:17.658]  Generate ansible inventory...
...
[2019-09-25 18:08:25.858]  Workloads ['azr2-w01-myfab2'] configured successfully
```

To start the workload, run the command:

```
(myfab2) bwctl> start workload azr2-w01-myfab2
```

You should see output similar to:

```
[2019-09-25 18:09:18.375]  Workloads to be started: ['azr2-w01-myfab2']
...
[2019-09-25 18:09:21.495]  Workloads ['azr2-w01-myfab2'] started successfully
```

7.3.4 Check Resource Graph

To verify that both the processor and workload nodes have joined the service interconnection fabric, go to orchestrator and click on **Resource Graph**.

7.4 Summary

7.4.1 Review Steps

You now have an environment for multi-cloud application deployment with two core components–fabric manager and policy orchestrator.

The fabric manager allows you to add or remove cloud resources in AWS, Azure, and GCP to satisfy your application's computational needs. Whereas the policy orchestrator is a tool for over-the-top segmentation of the application services deployed on those resources.

For practice purposes, you have already set up one VPC with the policy processor, securing the location, and one workload node, ready for deployment of an application service. You can now add more workload nodes to the VPC and/or create more VPCs with processor and workload nodes in each.

At this point, you can exit from the BWCTL prompt by running the command:

```
(myfab2) bwctl> quit
```

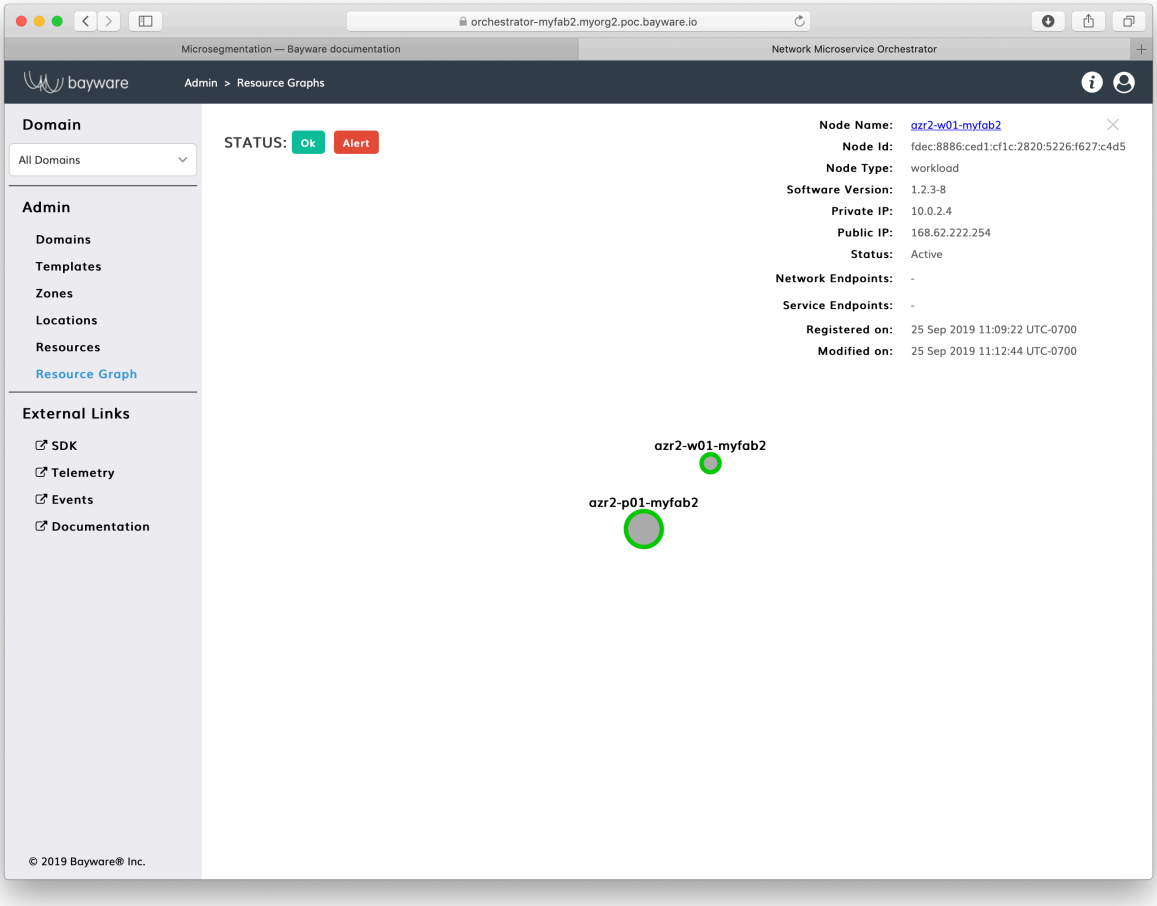


Fig. 7.3: Fig. Orchestrator Resource Graph Page

7.4.2 Next Step

At the next step, you will create a resource interconnection policy for your workload nodes. For now, the nodes you have created are isolated from each other and the external world.

Before you start the deployment of resources in your service interconnection fabric, you need to make sure you have:

- a Microsoft Azure account to obtain a copy of the fabric manager image from the Bayware Multicloud Service Mesh offer in Azure Marketplace;
- an AWS account to backup the fabric manager configuration in s3 and create orchestrator DNS record in Route53.

Note: Each section of this tutorial will take approximately 20 minutes to complete.

CHAPTER 8

Create Resource Connectivity Policy

8.1 Preparation

To set up the resource connectivity policy in your service interconnection fabric you will need an access to the fabric manager and the policy orchestrator created in the prior step.

All the tasks presented in this tutorial can be accomplished using either orchestrator Web-interface or BWCTL-API command-line tool. The tutorial shows how to perform them in BWCTL-API CLI.

Note: This tutorial will take approximately 10 minutes to complete.

8.1.1 Update BWCTL-API Tool

To make sure you are working with the latest version of software, update the BWCTL-API CLI tool already installed on your fabric manager node. To do this, you will need to SSH into the fabric manager node and switch to root level access to update all packages as such:

```
]$ sudo su -
```

Next, to update BWCTL-API, run this commands:

```
]# apt-get update
]# apt-get --only-upgrade install bwctl-api
```

To exit from the current command prompt once you have completed updating, run this command:

```
]# exit
```

8.1.2 Configure BWCTL-API

Before you can run BWCTL-API, you must configure the tool with your orchestrator credentials from the prior step:

- Orchestrator URL - **FQDN of orchestrator NBI**
- Domain - **default**
- Username - **admin**
- Password - **PASSWORD** from the prior step.

You store configuration locally in the file called `config.yaml` located at `~/.bwctl-api/config.yml`.

To edit information in the `config.yml` file, run this command:

```
]$ nano /home/ubuntu/.bwctl-api/config.yml
```

After editing, the `config.yml` file in this example contains:

```
hostname: 'orchestrator-myfab2.myorg2.poc.bayware.io'
domain: 'default'
login:  'admin'
password: 'RWpoi5RkMDBi'
```

8.2 Set up Zone

To set up resource policy for the processor and workload nodes you have already created, all you need is to put them in security zones.

8.2.1 Create Zone

First, create the new zone by running this command with a desired zone name (any string without spaces)–in this example `azure-uswest` as an argument:

```
]$ bwctl-api create zone azure-uswest
```

You should see output similar to this:

```
[2019-09-26 19:26:52.543]  Zone 'azure-uswest' created successfully
```

8.2.2 Add Processor to Zone

Next, assign the processor to the zone by running this command with the processor name–in this example `azr2-p01-myfab2` as an argument:

```
]$ bwctl-api update zone azure-uswest -a azr2-p01-myfab2
```

You should see output similar to this:

```
[2019-09-26 19:27:58.424]  Processor 'azr2-p01-myfab2' assigned to zone 'azure-uswest'
[2019-09-26 19:27:58.424]  Zone 'azure-uswest' updated successfully
```

8.2.3 Add Workload to Zone

Finally, put the location with workload nodes into the zone by running this command with the location name–in this example `azr2` as an argument:

```
]$ bwctl-api update location azr2 -z azure-uswest
```

You should see output similar to this:

```
[2019-09-26 19:29:29.498]  Location 'azr2' updated successfully
```

Note: Node's default location name is the left outmost part of the name of the VPC, in which the node is created, as for example: **vpc-name:** `azr2-vpc-myfab2` ==> **location-name:** `azr2`

At this point, you can open the orchestrator resource graph page and see the workload node now is connected to the processor node.

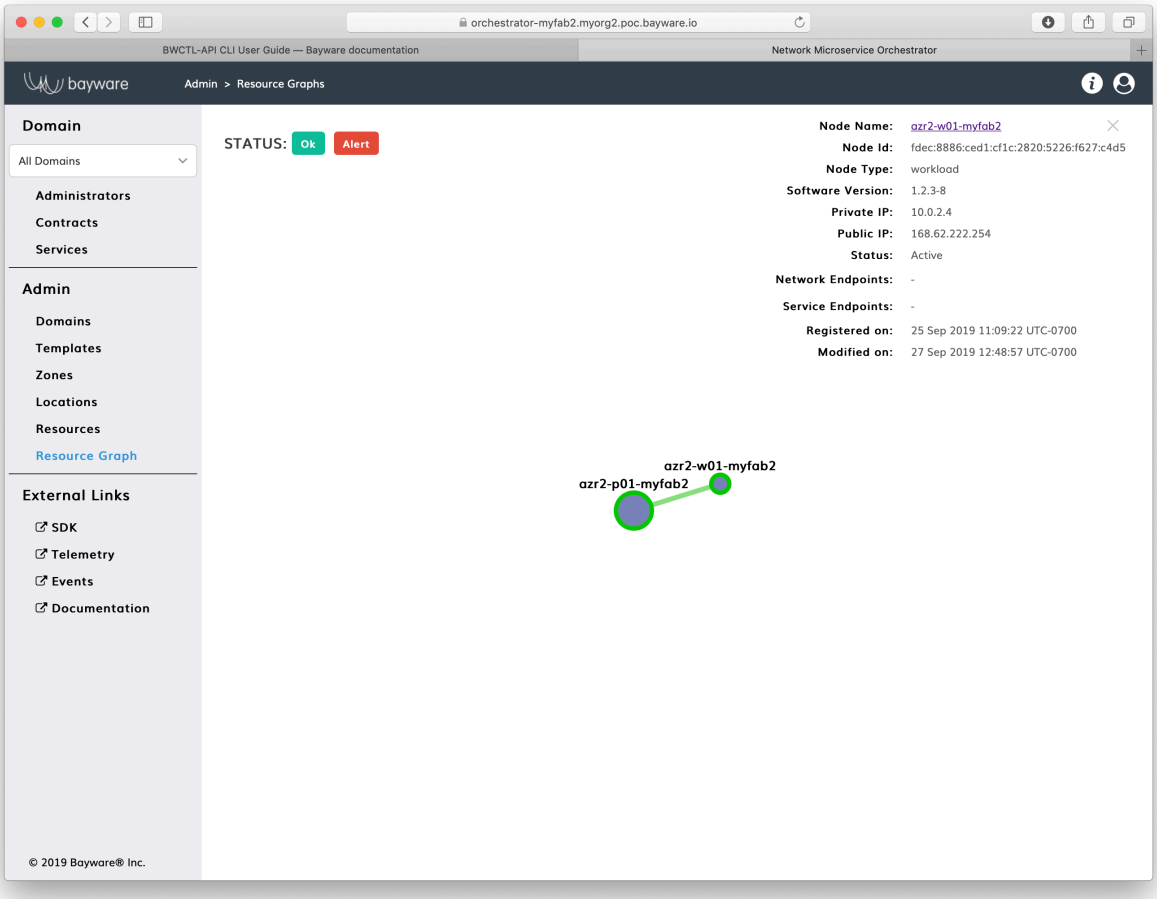


Fig. 8.1: Fig. Orchestrator Resource Graph Page

8.3 Interconnect Zones

To interconnect zones, you must specify a link between two processor nodes located in the zones. In the link specification, you can use references to already deployed nodes or nodes you are planning to create later.

8.3.1 Declare Processor

If the processor node doesn't exist yet, declare the node by running this command with the expected node name–in this example `gcp1-p01-myfab2` as an argument:

```
]$ bwctl-api create resource gcp1-p01-myfab2 -type processor -l default
```

You should see output similar to this:

```
[2019-09-26 19:30:16.487]  Resource 'gcp1-p01-myfab2' created successfully
```

8.3.2 Specify Link

To specify a link between nodes, run this command with the source and target processor node names–in this example `azr2-p01-myfab2` and `gcp1-p01-myfab2` as arguments:

```
]$ bwctl-api create link -s gcp1-p01-myfab2 -t azr2-p01-myfab2
```

You should see output similar to this:

```
[2019-09-26 19:30:52.559]  Link 'azr2-p01-myfab2_gcp1-p01-myfab2' created successfully
```

At this point, you can open the orchestrator page called Resources and see the operational status of your resources– `active` for already running nodes and `init` for declared-only nodes.

8.4 Summary

8.4.1 Review Steps

In a couple steps you have created and applied the resource interconnection policy for your workload nodes.

In step one, the workload node location received permission to join the fabric using a particular processor, so the workload node automatically established a security association with the processor.

In step two, the processor was authorized to automatically connect to another processor–as soon as the latter is deployed–establishing in this way a secure communication channel between clouds for workload nodes.

8.4.2 Next Step

At the next step, you will create infrastructure-agnostic service interconnection policy for your application. As for now, each processor node keeps acting in its default-deny mode by blocking all data communications from the workload nodes attached to it.

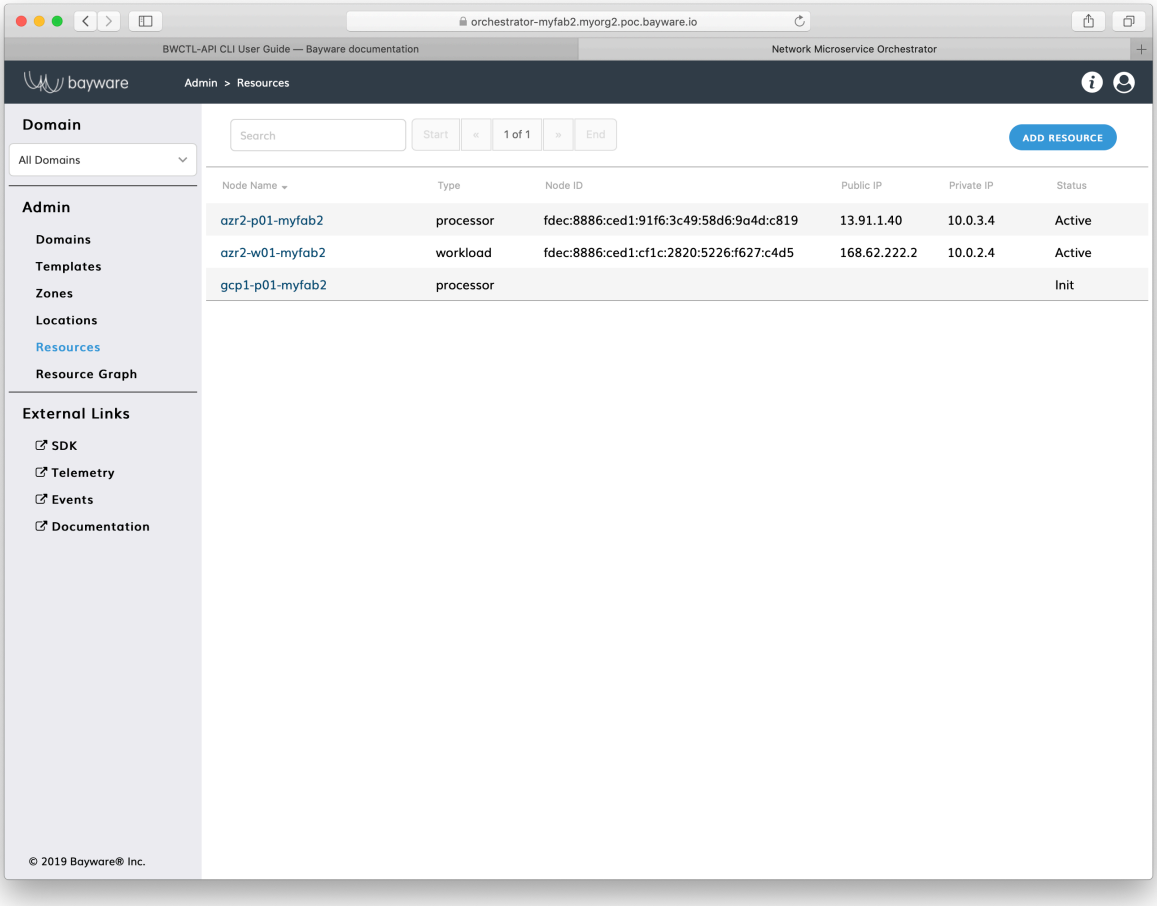


Fig. 8.2: Fig. Orchestrator Resources Page

CHAPTER 9

Create Service Connectivity Policy

9.1 Preparation

To set up the service connectivity policy in your service interconnection fabric you will need an access to the fabric manager and the policy orchestrator.

All the tasks presented in this tutorial can be accomplished using either orchestrator Web-interface or BWCTL-API command-line tool. The tutorial shows how to perform them in BWCTL-API CLI.

Note: This tutorial will take approximately 10 minutes to complete.

9.2 Upload Communication Rules

You can program your own communication rules using the policy orchestrator SDK or start with a default set of rules coming with BWCTL-API tool. To upload the default set of rules, run this command:

```
]$ bwctl-api create template default
```

You should see this output:

```
[2019-09-26 19:41:53.528]  Template 'default' created successfully
```

9.3 Create Service Graph

9.3.1 Create Domain

To create a namespace for your application policy, run this command with a desired domain name (any string without spaces)–in this example `myapp` as an argument:

```
]$ bwctl-api create domain myapp
```

You should see output similar to this:

```
[2019-09-26 19:42:38.726]  Domain 'myapp' created successfully
```

9.3.2 Specify Contract

To specify a security segment in the newly created namespace, run this command with a desired contract name (any string without spaces) preceding the domain name–in this example `frontend@myapp` as an argument:

```
]$ bwctl-api create contract frontend@myapp
```

You should see output similar to this:

```
[2019-09-26 19:43:13.294]  Contract 'frontend@myapp' created successfully
```

9.3.3 Name Service

To name a service in the newly created namespace, run this command with a desired service name (any string without spaces) preceding the domain name–in this example `http-proxy@myapp` as an argument:

```
]$ bwctl-api create service http-proxy@myapp
```

You should see this output:

```
[2019-09-26 19:43:45.779]  Service 'http-proxy@myapp' created successfully
```

9.3.4 Authorize Service

To authorize the newly created service to access the security segment, you have to assign the service a role in the contract.

To check available roles, run this command with the contract name–in this example `frontend@myapp` as an argument:

```
]$ bwctl-api show contract frontend@myapp
```

You should see output similar to this:

```
---
apiVersion: policy.bayware.io/v1
kind: Contract
metadata:
  description: frontend
  domain: myapp
  name: frontend
spec:
 contract_roles:
```

(continues on next page)

(continued from previous page)

```
- cfg_hash: c40f2ddc0843e983a4ea4088e2ea0f8e
  description: null
  id: 1
  ingress_rules:
  - {}
  name: originator
  path_params: {}
  port_mirror_enabled: false
  program_data:
    params:
    - name: hopsCount
      value: 0
    ppl: 0
  propagation_interval: 5
  role_index: 0
  service_rdn: originator.frontend.myapp
  stat_enabled: false
- cfg_hash: 84dcec61d02bb315a50354e38b1e6a0a
  description: null
  id: 2
  ingress_rules:
  - {}
  name: responder
  path_params: {}
  port_mirror_enabled: false
  program_data:
    params:
    - name: hopsCount
      value: 0
    ppl: 0
  propagation_interval: 5
  role_index: 1
  service_rdn: responder.frontend.myapp
  stat_enabled: false
enabled: true
template: default
```

Note: The contract specification always includes two roles. A unique role identifier is built using such notation – **<role_name>:<contract_name>**.

To assign a contract role to the service, run this command with the service name and the contract role–in this example `originator:frontend` as an argument:

```
]$ bwctl-api update service http-proxy@myapp -a originator:frontend
```

You should see output similar to this:

```
[2019-09-26 19:44:23.626]  Service 'http-proxy@myapp' updated successfully
```

To verify that your application policy is now in place, go to orchestrator, select your application domain and click on **Service Graph**.

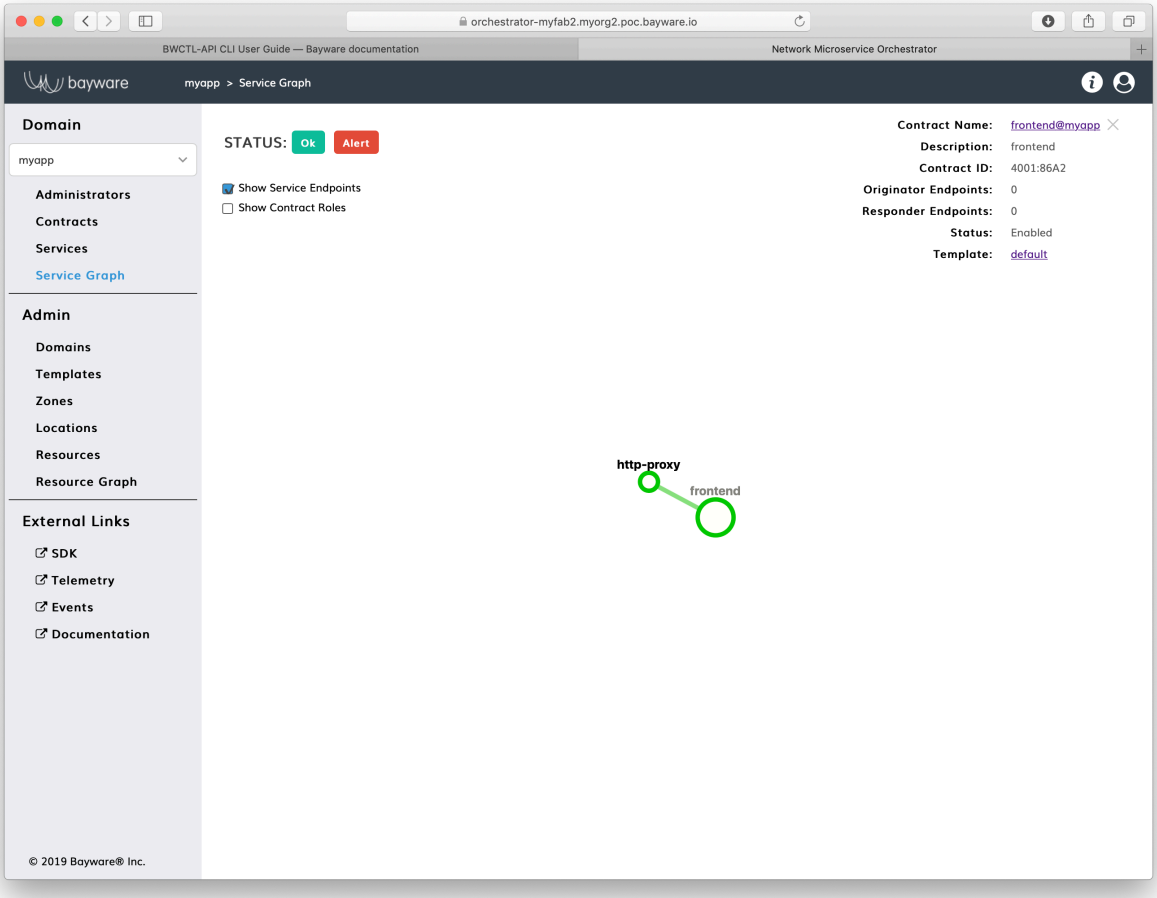


Fig. 9.1: Fig. Application Service Graph Page

9.4 Summary

9.4.1 Review Steps

You now have set up an infrastructure-agnostic segmentation policy for your application.

Firstly, you uploaded a template that implements default communication rules of interaction between application services in multi-cloud environment.

Secondly, you created a security segment–called **contract** using the template and specified that one of your application services is authorized to access this segment in a particular role.

9.4.2 Next Step

At the final step, you will deploy authorized application services on workload nodes in your service interconnection fabric.

No changes to application code, no proxies in between, and no network configuration is needed. Just install existing packages alongside with service authorization tokens and you will automatically receive multicloud secure segmentation, service discovery, and on-the-fly traffic rerouting for disaster recovery and cost optimization.

CHAPTER 10

Deploy Application

10.1 Preparation

To deploy an application in your service interconnection fabric, you will need access to the fabric manager and the policy orchestrator.

Note: This tutorial will take approximately 10 minutes to complete.

10.2 Generate Token

To generate a new authorization token for your application service, run this command using `service_name@contract_name`–in this example `http-proxy@myapp` as an argument:

```
]$ bwctl-api create service_token http-proxy@myapp
```

You should see output similar to this:

```
---
apiVersion: policy.bayware.io/v1
kind: ServiceToken
metadata:
  token_ident: 00c1babd-3197-465a-beec-6d144e53d4ef:46a55e4963263260c3d61eb4b4b67882
spec:
  domain: myapp
  expiry_time: 30 Sep 2020 18:41:52 GMT
  service: http-proxy
  status: Active
```

Warning: Token comprises two parts–token identity and token secret–separated by a colon. This is the only time you can see the token secret. Be sure to copy the entire TOKEN as it appears on your screen, it will be needed later.

10.3 Deploy Service

10.3.1 SSH to Workload Node

To deploy the service on the workload node, first ssh to the workload node from your fabric manager as such:

```
]$ ssh azr2-w01-myfab2
```

Note: SSH service is set up automatically for easy and secure access to workload nodes in your service interconnection fabric.

When you are on the workload node, switch to root level access:

```
[ubuntu@azr2-w01-myfab2]$ sudo su -
```

10.3.2 Add Token

Next, edit the policy agent token file by running this command:

```
]# nano /opt/bayware/ib-agent/conf/tokens.dat
```

Add the token to the `tokens.dat` file and save the file, which in this example will contain after editing:

```
00c1babd-3197-465a-beec-6d144e53d4ef:46a55e4963263260c3d61eb4b4b67882
```

To apply the token, reload the policy agent by running this command:

```
]# systemctl reload ib-agent
```

At this point, you can visit the policy orchestrator and find a registered endpoint on the **Service Graph** page of your application.

10.3.3 Install Service

Now, you can install your application service on this workload node. In this example, a package called `getaway-proxy` installs by running the command:

```
]# apt-get install getaway-proxy
```

The service automatically discovers all remote services sharing the same contract. So, edit the application service configuration file to update the remote service URL, by running this command:

```
]# nano /opt/getaway-proxy/conf/app.conf
```

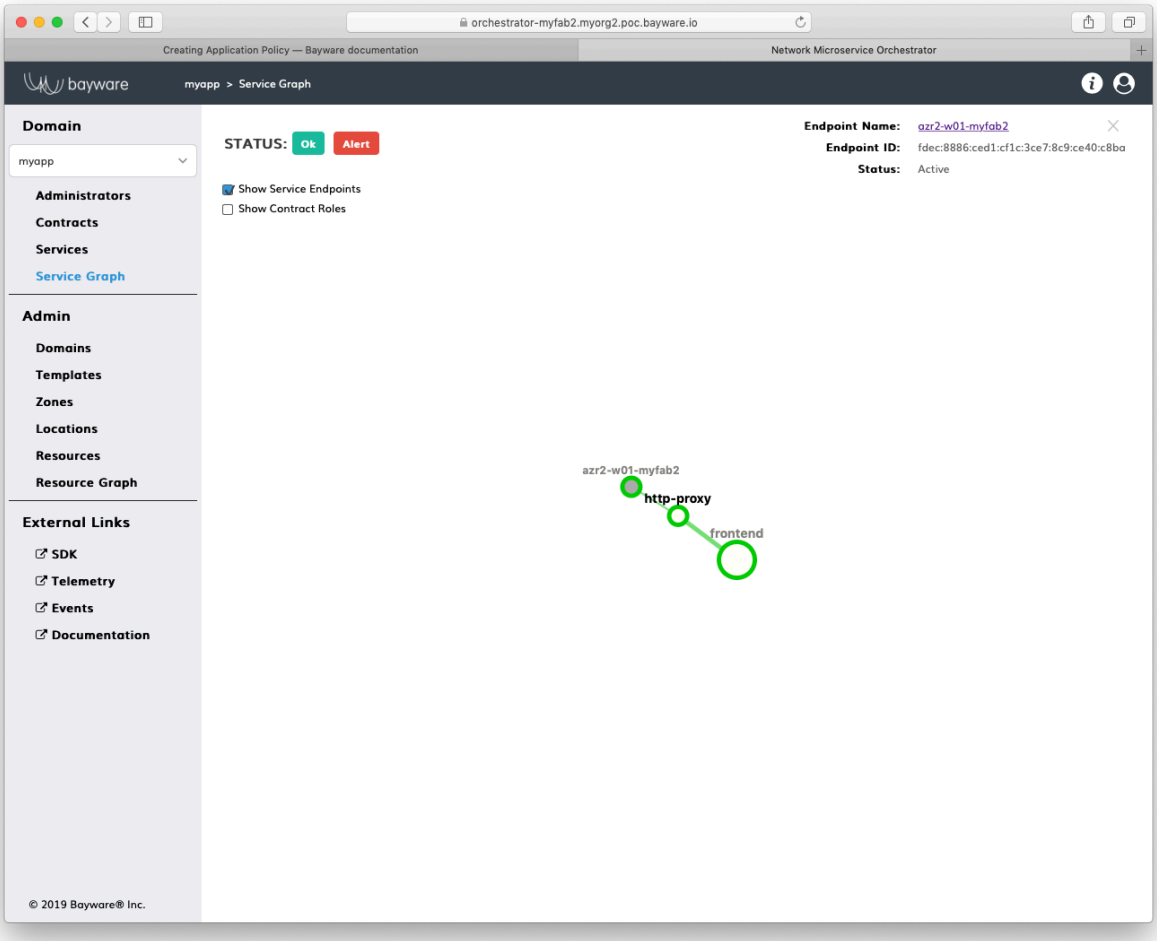


Fig. 10.1: Fig. Service Graph with Registered Service Endpoint

After editing, the service configuration file in this example contains:

```
WS_APP = 'http://responder.frontend.myapp.ib.loc:8080/'
```

Note: The FQDN part of remote service URL is automatically built in such a manner: **<role>.<contract>.<domain>.ib.loc**

Finally, to start the application service, run this command:

```
]# systemctl start getaway-proxy
```

10.4 Summary

You have successfully installed the application service on the workload node in your service interconnection fabric.

Your application policy for this service is infrastructure-agnostic and allows the service to access only particular security segment. You can move this service across workload nodes in multiple clouds and the security segment will expand or shrink automatically.

The service will automatically discover the opposite-role services in its security segment as those services will go up or down across multiple clouds.

CHAPTER 11

Clean up

If you would like to completely delete your current fabric installation, follow these commands in order to do so. Please note, this operation is irreversible and very destructive, do exercise extreme caution when executing this procedure!

To delete the entire service interconnection fabric, first start BWCTL by running this command:

```
]$ bwctl
```

You should see output with your fabric name similar to this:

```
(myfab2) bwctl>
```

Now, export the current fabric configuration by running this command:

```
(myfab2) bwctl> export fabric myfab2.yml
```

You should see output similar to this:

```
[2019-09-27 21:24:52.558]  Exporting to 'myfab2.yml'
[2019-09-27 21:24:52.670]  Fabric configuration exported successfully
```

To delete all fabric components, run this command:

```
(myfab2) bwctl> delete batch myfab2.yml
```

You should see output similar to this:

```
[2019-09-27 21:25:23.189]  Deleting batch: file='myfab2.yml', input=format='yaml', dry-
↪run=False
[2019-09-27 21:25:23.199]  Found batch 'myfab2' (Fabric "myfab2" export at Thu Oct  3␣
↪21:24:52 2019) with 6 objects
[2019-09-27 21:25:23.199]  Fabric: ['myfab2']
[2019-09-27 21:25:23.199]  Vpc: ['azr1-vpc-myfab2', 'azr2-vpc-myfab2']
```

(continues on next page)

(continued from previous page)

```
[2019-09-27 21:25:23.199]  Orchestrator: ['azr1-c01-myfab2']
[2019-09-27 21:25:23.199]  Processor: ['azr2-p01-myfab2']
[2019-09-27 21:25:23.199]  Workload: ['azr2-w01-myfab2']
[2019-09-27 21:25:23.199]  Do you want to delete these objects? [y/N]
```

Type **y** and press Enter:

```
y
```

You should see output similar to this:

```
[2019-09-27 21:25:29.159]  Fabric 'myfab2' is going to be deleted with all nested objects
...
[2019-09-27 21:44:55.341]  Fabric 'myfab2' deleted successfully
```

Once the BWCTL CLI commands finish running, you may safely delete the fabric manager's VM and corresponding components via the Azure Console. Also, you can delete the S3 bucket used for fabric manager's state backup via the AWS Console.

CHAPTER 12

Deploying Service Interconnection Fabric

12.1 Cloud Infrastructure

12.1.1 Introduction

This tutorial gives the user hands-on experience with Bayware's SDK, Orchestrator, Processors, and Agents. By following the steps outlined below, you will create a Bayware network that spans three public cloud providers, manually deploy a microservice-based application called Getaway App, use Ansible to deploy a second microservice-based application called Voting App, and finally use Bayware's Network Microservices to experience the simplicity in managing your network in a hybrid cloud environment.

To use this tutorial, you should have already received an email from Bayware with an attached personal *Sandbox Installation Summary* (SIS) page. If you have not received this email or have not yet contacted Bayware regarding your interest in running this tutorial, please reach out to us from our contacts page. Alternatively, you may follow along by referring to an *example SIS*.

This tutorial assumes you have internet access, a web browser, and a terminal window all running on the same computer. Your computer should have an **`ssh`** client installed and you should be familiar with using and executing commands using a command-line interface. This tutorial runs equally well in Linux, MacOS, and Windows.

12.1.2 Tutorial Outline

This tutorial is broken up into the following steps.

1. Setup Enterprise Cloud Infrastructure
2. Deploy Service Interconnection Fabric for Applications
3. Deploy Application 1 - Getaway App
    a. Manually Install Application Microservices with Bayware Interface
    b. Demonstrate Microservice Mobility

c. Demonstrate Policy Enforcement with Network Microservices

d. Delete Application & Clean VMs

4. Deploy Application 2 - Voting App

a. Use Ansible to Install Application Microservices with Bayware Interface

b. Demonstrate Microservice Mobility

c. Demonstrate Policy Enforcement with Network Microservices

d. Delete Application & Clean VMs

12.1.3 Enterprise Cloud Infrastructure

Compute Resources

You have been tasked with deploying two microservice-based applications in a multi-cloud environment. The cost and service-level differences among public cloud providers are substantial enough that you need to ensure easy mobility of your microservices from one public cloud to another and, moreover, guarantee that your customers are using your resources in an efficient, cost-effective manner. Security requirements and other policy enforcement considerations are tantamount to a successful deployment.

Imagine that you have setup your enterprise cloud infrastructure using 17 virtual machines (VMs) spread across Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft's Azure. For security concerns, you will manage your network from a single node. This node, VM1 or `aws-bastion` (see your *SIS page*), is called the Customer Command Center (CCC). All interaction with VM2 through VM17 takes place from your CCC. As such, you have **`ssh`** password access to your CCC and **`ssh`** key access to all other VMs from your CCC.

Since you'll be referring to it frequently, keep a copy of the *SIS* available that you received in the *Welcome to the Bayware Tutorial* email. (Alternatively, you may refer to the example *SIS* that is in this tutorial. In that case, you will have no VM resources, but you will be able to read through the tutorial in context.)

On your *SIS* page, note that this tutorial may refer to the VMs in the table using either the VM number indicated in column one or the host name indicated in column two. For example, in the next section you will be asked to log into the Bayware processor in GCP. The tutorial will refer to this VM as either VM12 (its VM number) or `gcp-p1-382fd7` (its host name). Since the suffix of each host name is user- (or sandbox, sb-) specific, the tutorial would, for instance, abbreviate the host name associated with VM12 as `gcp-p1`.

From your CCC, you may log into other VMs using their public IP addresses or their truncated host name. When the tutorial uses a public IP address, it will be referred to as VM<num>-IP. So the IP address for VM12 is written as VM12-IP, which can easily be referenced from the table in the *SIS*. The `/etc/hosts` file on the CCC, however, has been pre-configured to map truncated host names with their IP addresses. So you can login to `gcp-12-382fd7`, for instace, using **`ssh`**@gcp-12 from the CCC.

Web pages used throughout the tutorial use FQDNs and are listed near the top of your *SIS*.

Command Center

So let's get started with some typing by logging into your CCC from your local computer using the username (centos) and the password listed in the table on your *SIS* for `aws-bastion`. For your convenience, you can use the bastion's FQDN, which should look something like `ap`*`382fd7`*`.sb.bayware.io`. You can find the full bastion FQDN on your SIS just above table row 1 of the *VM table* (look for *bastion FQDN*). Just be sure to replace the *`382fd7`* with your own sandbox name. Enter the following command and hit `return`:

```
]$ ssh centos@ap382fd7.sb.bayware.io
```

Be aware that the first time you log in to a new machine, you might see a scary warning message that looks like

```
]$ ssh centos@ap382fd7.sb.bayware.io
The authenticity of host 'ap382fd7.sb.bayware.io (13.56.241.123)' can't be established.
ECDSA key fingerprint is SHA256:6LLVP+3QvrIb8FjRGN1eLQRy7zL2eXeNCdOoYRbbxqw.
ECDSA key fingerprint is MD5:7b:fd:15:4c:35:d3:1d:20:fd:3e:3d:b7:1b:14:6a:1b.
```

Where it asks if you wish to contine, just type `yes`.

```
Are you sure you want to continue connecting (yes/no)? yes
```

You will be prompted for your password with the following query

```
centos@ap382fd7.sb.bayware.io password:
```

Type in the password for your `aws-bastion`. Again, this is located in the rightmost column of row one on the *VM table* in your SIS.

If all goes well, you should now see a prompt in your terminal window such as

```
[centos@aws-bastion-382fd7 ~]$
```

That's all for now on your CCC. But don't close the terminal window since you'll use it the next section when you install the service interconnection fabric. And, if it's not completely clear, you can always open up more terminal windows on your local computer and log into your CCC from them so you have more windows to work with.

Good Housekeeping

The 17 VMs running across the three public cloud providers started with stock CentOS 7 images. You can ping your VMs, but not much else, since cloud provider security groups have been set to allow SSH, ICMP, and a couple of UDP ports you'll need for IPsec.

To ensure the operating systems on all VMs are starting from the same point, we've already done a few upgrades for you including the following

- updated all CentOS 7 packages
- added repository `epel-release` and `bayware-repo`
- installed, started, and enabled firewalld
- set firewall rules on public zone, which includes eth0, to allow IPsec and accept GRE tunnels
- and installed orchestrator and telemetry tools: telegraf, InfluxDB, Grafana

Now all your VMs are locked down with only the minimally required access allowed.

Orchestrator

Finally, you will be performing many tasks in this tutorial using the Bayware Orchestrator. As such, open a new tab in your browser and navigate to the *web address for the orchestrator* listed in the table near the top of your *SIS*. The address should look something like `https://c1382fd7.sb.bayware.io`. The login page should appear as

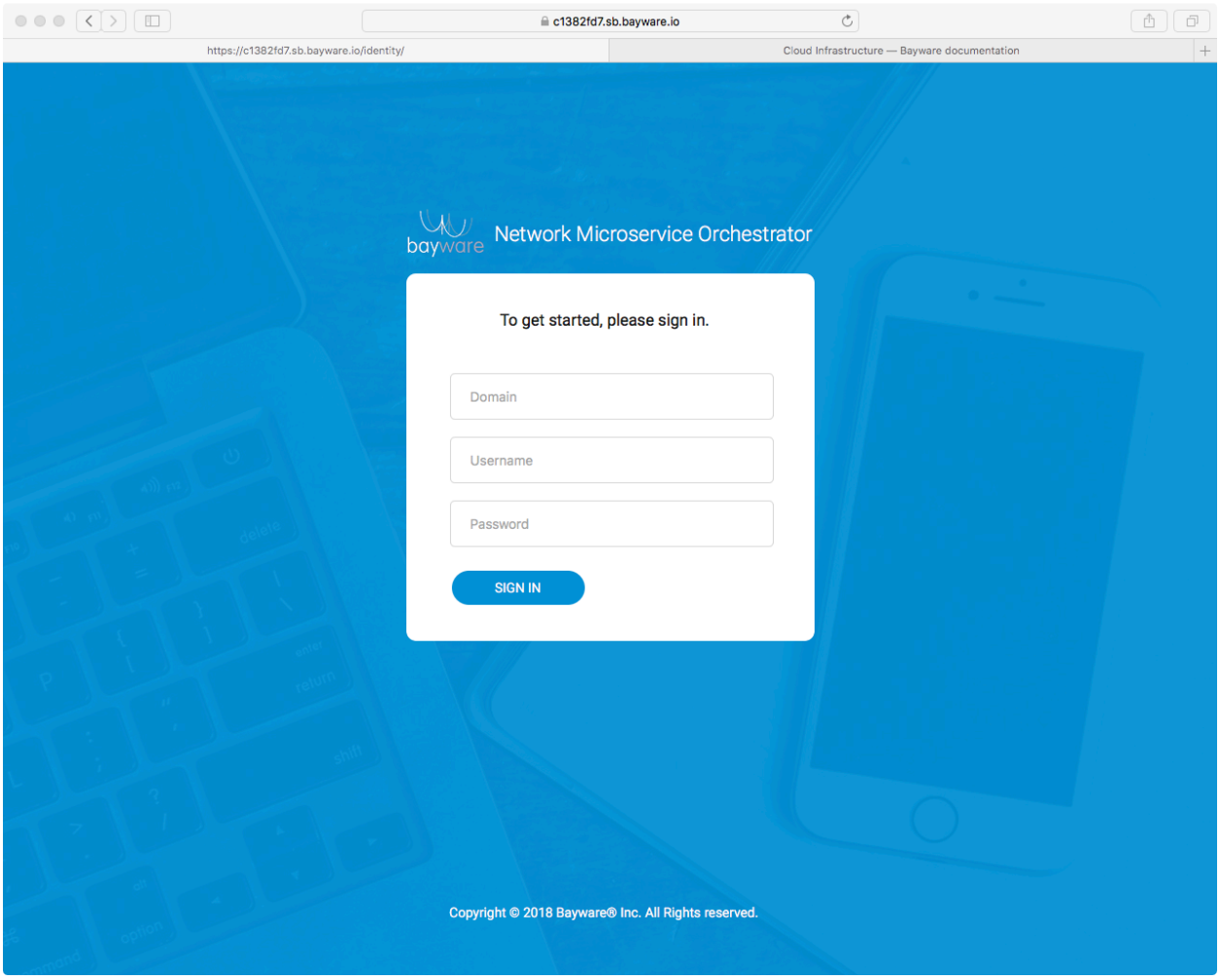


Fig. 12.1: Orchestrator Admin Login Page

Use the *orchestrator credentials* shown in your *SIS* to sign in. You'll find this on row 18 in the tables in your *SIS*. (Note that your orchestrator login credentials are different from the username and password for `aws-c1` shown in the table in your SIS. The former logs you into the orchestrator application running in your browser while the latter, if allowed, would simply log you into the Linux operating system running on `aws-c1`.)

After successfully logging into your orchestrator, you should see a page such as

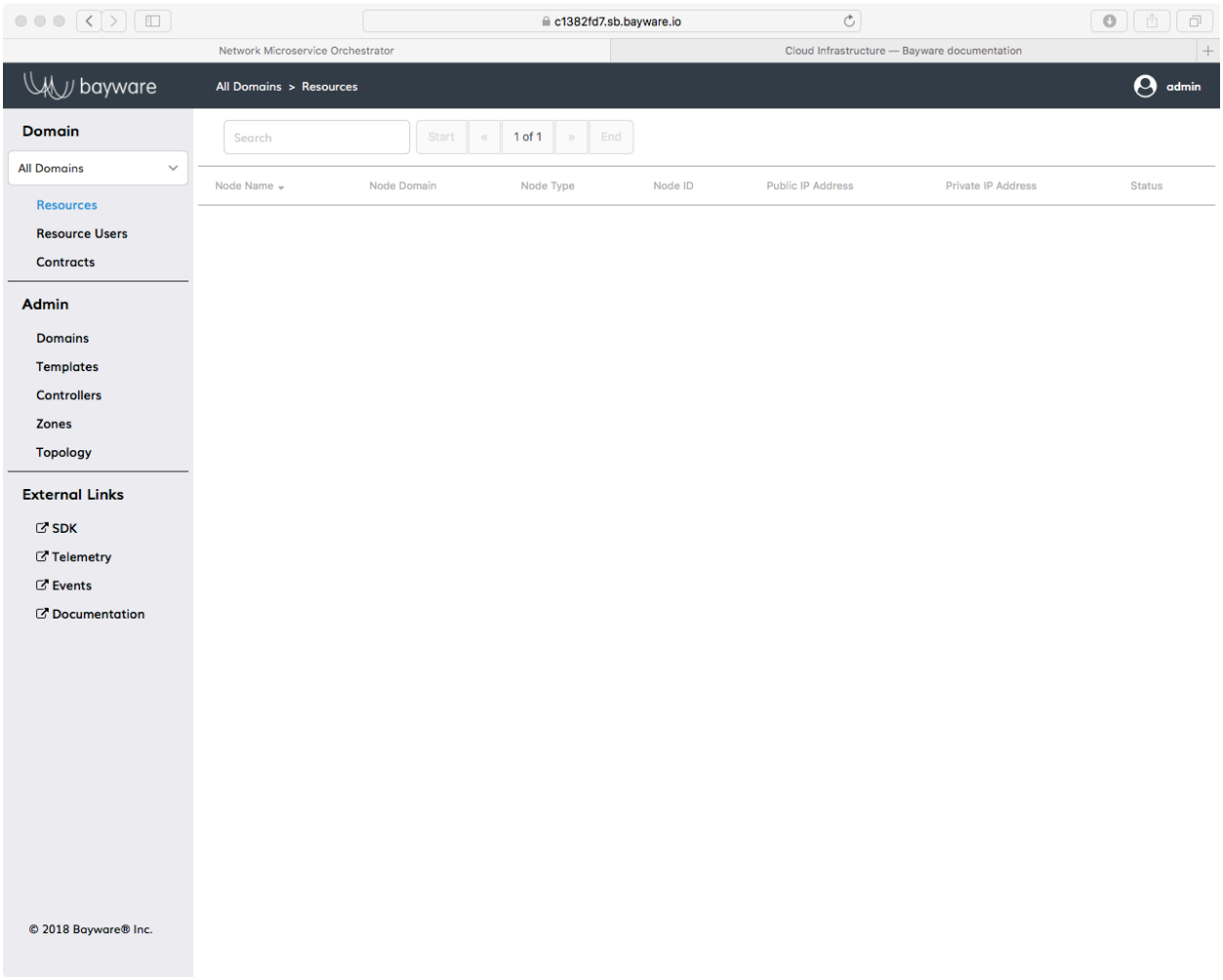


Fig. 12.2: Clean Orchestrator After Login

Now you're all set up for the steps that follow.

Visibility

If you're commanding a giant infrastructure of 17 virtual machines using Bayware technology, it's good to be able to have a look at what they're doing. For that, Bayware has teamed with Grafana to display all types of telemetry information–from the usual processor load and memory usage to Bayware-specific dashboards such as GRE interface usage and IPsec traffic. You can conveniently get to telemetry information by clicking the *Telemetry* link in the left-side navigation bar on the orchestrator as shown in Fig. 12.3.

You can scroll through the dashboard to see telemetry information for the server indicated at the top of the window.

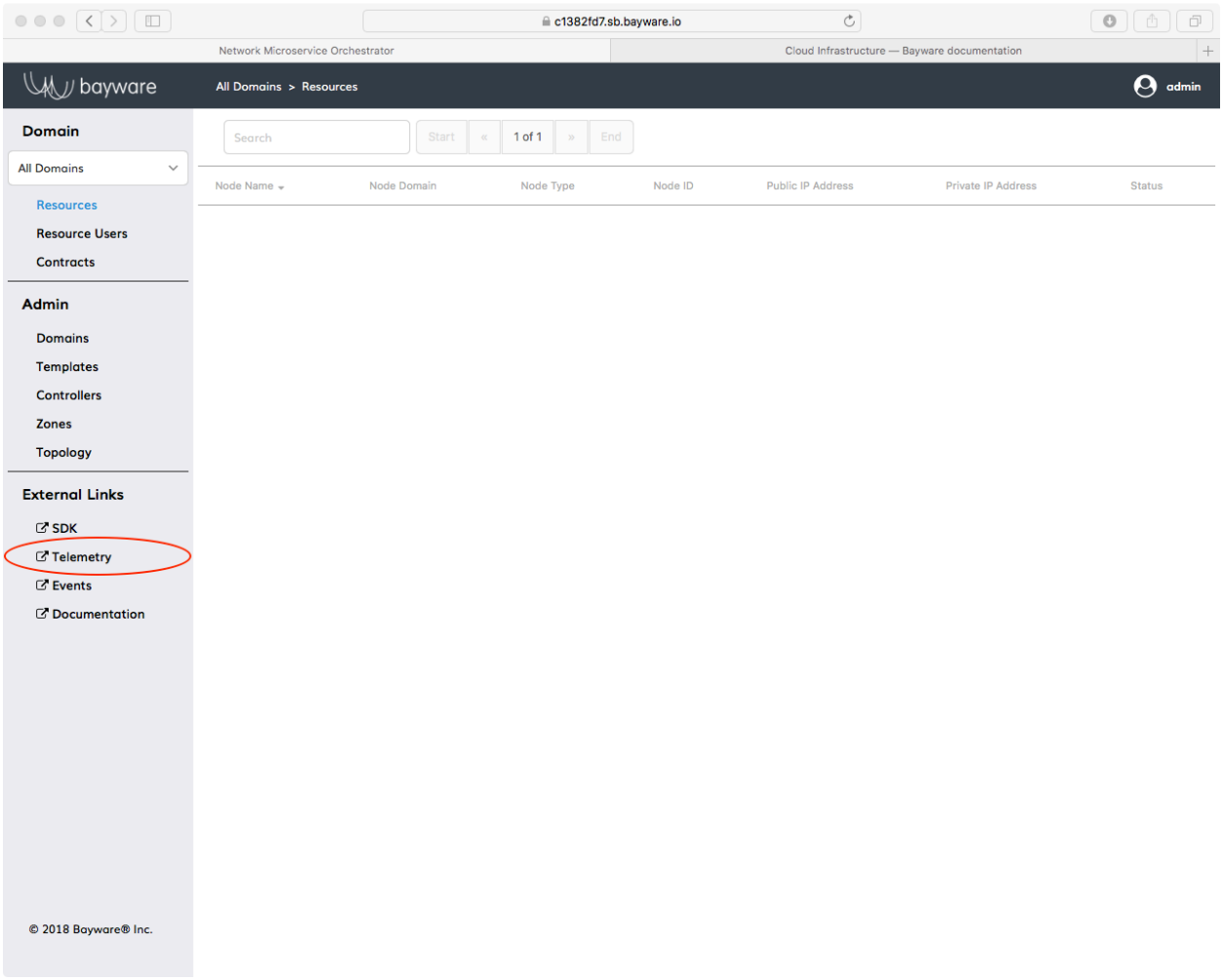


Fig. 12.3: Access Telemetry from the Orchestrator

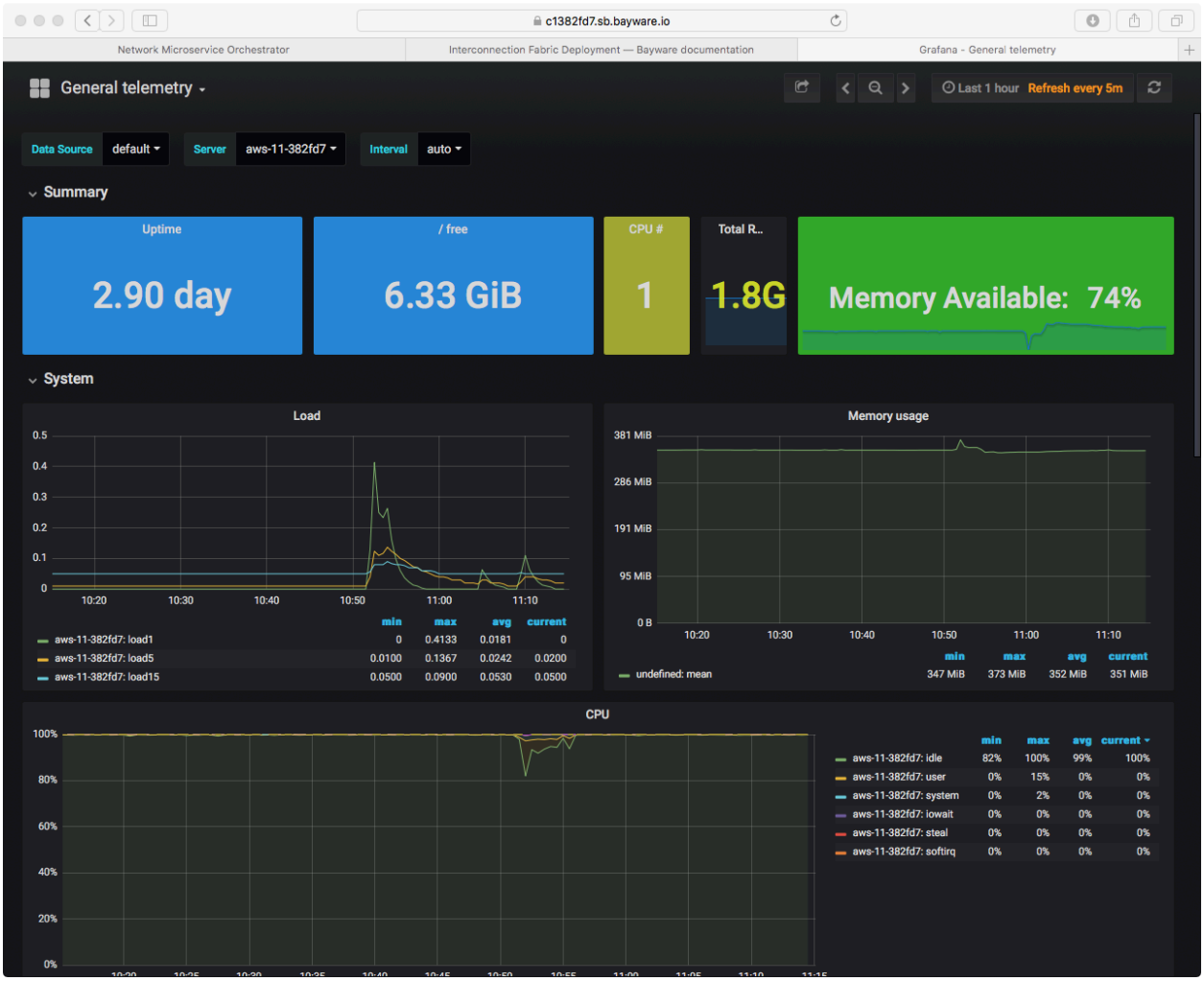


Fig. 12.4: Telemetry Using Grafana

Or use the drop down menu as shown in Fig. 12.5 to check out stats for a different virtual machine.

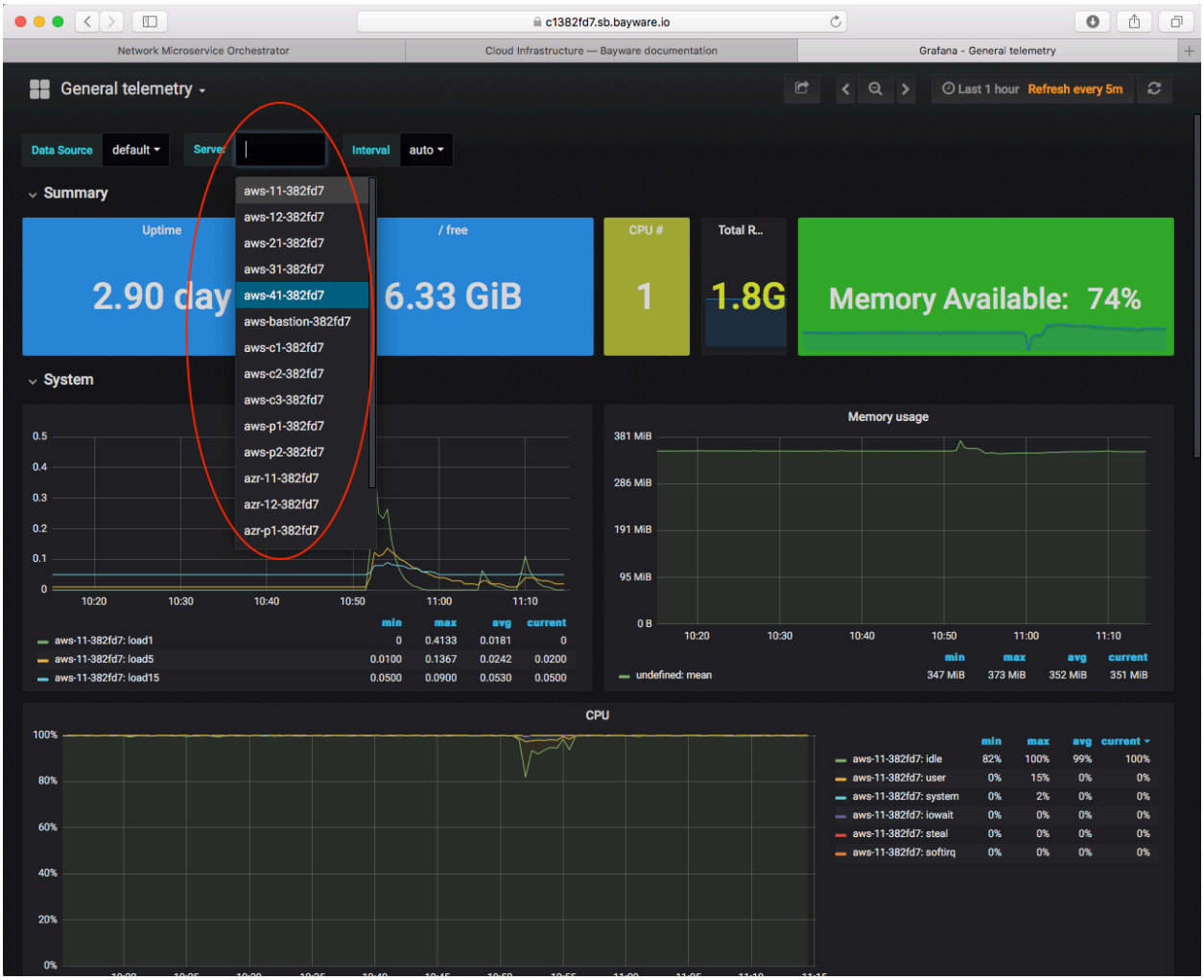


Fig. 12.5: View Telemetry for a VM

12.1.4 Summary

Good job! Now you have an idea where you're headed in the next sections and you've practiced logging into your control center from which you'll manage all your infrastructure resources. You should be comfortable navigating to your orchestrator browser page and know that you can get telemetry information about any of your virtual machines through the Grafana application. And, importantly, you have reviewed some of the information on your Sandbox Information Sheet (SIS) and you know not to confuse the example *SIS* located in this documentation with the personalized version you received from Bayware, which you'll use extensively throughout the remainder of this tutorial.

Next up: create a Bayware service interconnection fabric...

12.2 SIF Deployment

A full Bayware system utilizes an orchestrator, processors, and agents. The processors in your system work together to form a service interconnection fabric between your application microservices. For this tutorial, you will go through the steps to turn four of your VMs (Virtual Machines) into processors: `aws-p1` and `aws-p2` in AWS; `azr-p1` in Azure; and `gcp-p1` in GCP. But to be clear–and recalling the importance of policy enforcement in your deployments–installing a service interconnection fabric presents a zero-trust network in which no communication is allowed between your microservices until contracts are explicitly put in place.

Before we begin, let's use the orchestrator to show that no processors (and no agents) currently exist in your network. To do this, go back to the browser window in which you logged into the orchestrator. In the navigation bar on the left, find the *Topology* button under the *Admin* heading as shown in Fig. 12.6.

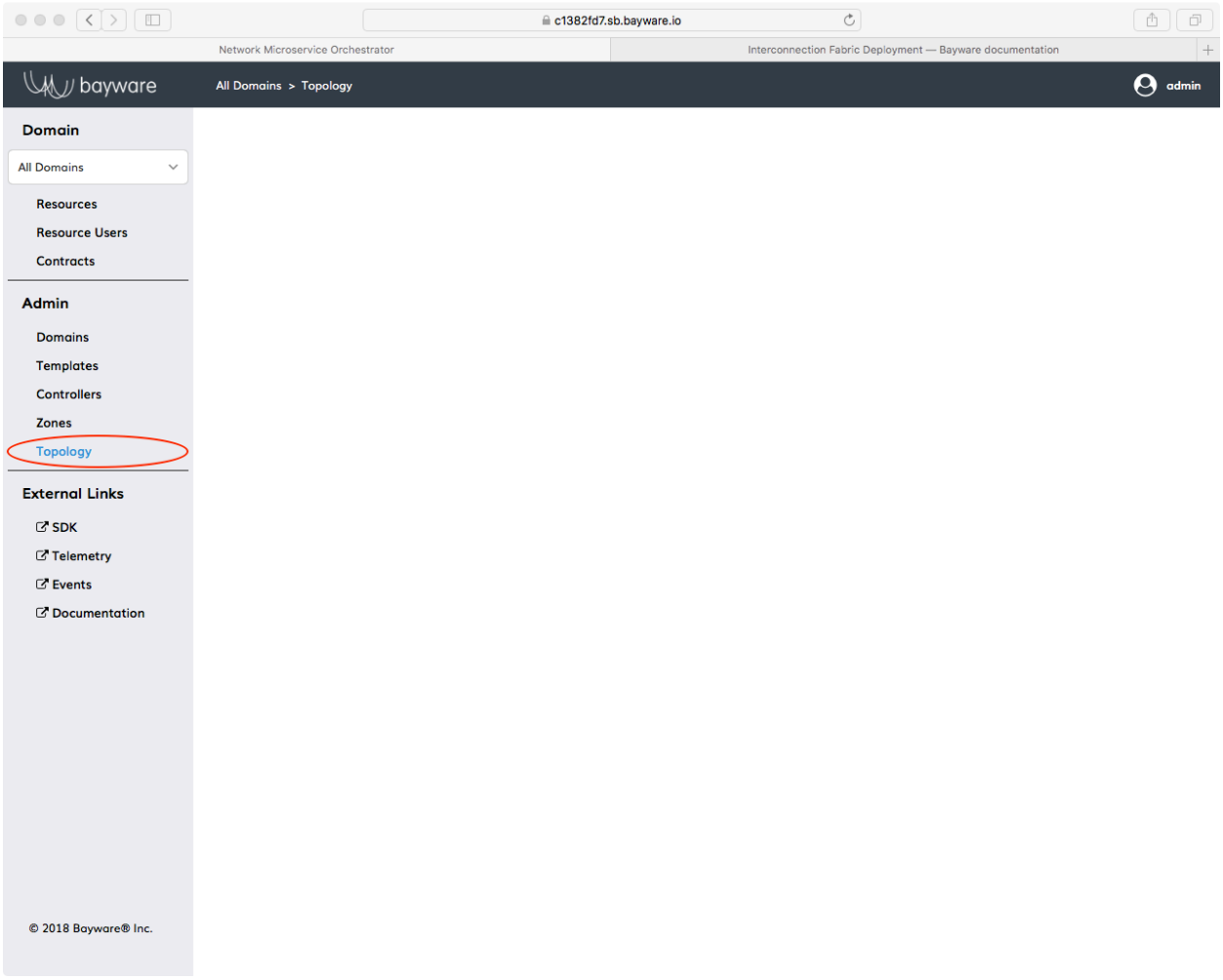


Fig. 12.6: Empty Orchestrator Topology

Click *Topology*. The pane on the right should be empty. Later, you will see that processors appear as large circles in this pane.

The subsequent commands can be broken up into

1. Login to a processor VM's OS
2. Install Bayware's engine and Open vSwitch

3. Configure the engine
4. Repeat steps 1 - 3 for each processor VM
5. Create links between processors

12.2.1 Step 1: SSH to VM

Let's begin with `aws-p1`. From the command-line prompt on your CCC, login

```
]$ ssh centos@aws-p1
```

You will not need a password since a public key has already been installed on `aws-p1`.

You should now have a prompt on `aws-p1` that looks similar to

```
[centos@aws-p1-382fd7 ~]$
```

The following commands require super-user privileges, so become root

```
[centos@aws-p1-382fd7 ~]$ sudo su -
```

which should give you a prompt like

```
[root@aws-p1-382fd7 ~]#
```

The `root` prompt will be abbreviated in the description below to `]#`.

12.2.2 Step 2: Install Bayware's engine and Open vSwitch

Each processor node in a service interconnection fabric is comprised of two pieces: an engine that determines intent and a data path that moves traffic. The engine, part of Bayware's core technology, is an Erlang- and C-based application available from the `bayware-repo`. The data path for this particular implementation utilizes the open-source application, Open vSwitch. The `bayware-repo` and the `epel-release` repo, both required for these applications, have been preinstalled on your virtual machines.

Now install the engine and Open vSwitch on `aws-p1`

```
]# yum install ib_engine openvswitch -y
```

12.2.3 Step 3: Configure the engine

Now you will configure the engine so that it becomes visible to the orchestrator. For security purposes, each engine in the network may have its own login credentials. And that's how this tutorial has been configured. In your *SIS* page, locate the table with *Bayware Processor login credentials* (overall, table rows 19 - 22). Note the `domain`, `username`, and `password` associated with the first processor (row 19) as you'll need it in the next step.

To configure the engine, `cd` to the directory that contains the configuration script, `/opt/ib_engine/bin`.

```
]# cd /opt/ib_engine/bin
```

The script is called `ib_configure`. You can display the usage instructions by typing

```
]# ./ib_configure -h
```

You will run the script now in interactive mode. The script will prompt you to enter the following information

- orchestrator IP or FQDN: use `c1382fd7.sb.bayware.io` as shown in the *URL table* at the top of the *SIS*. Your FQDN prefix will be different than `c1382fd7` shown here. Do *not* include the `https://` that is present in the URL.
- node `domain`: use the domain from *login credentials*, row 19
- node `username`: use the username for this engine from *login credentials*, row 19
- node `password`: use the password for this engine from *login credentials*, row 19
- configure IPsec: answer `YES`

Begin the interactive script now by entering the following at your prompt

```
]# ./ib_configure -i
```

After you work your way through the script and it does its magic, the engine will be configured, but it won't be running. Since the engine is controlled by Linux `systemd`, you should start and enable it with the following

```
]# systemctl start ib_engine
]# systemctl enable ib_engine
```

The `aws-p1` node should now be registered as a processor on the orchestrator. To see this, once again go to the orchestrator tab open in your browser and click *Topology*.

You should see a green circle with the node name of this processor next to it as shown in Fig. 12.7. You can also see this registered resource by clicking on the orchestrator's *Resources* button located near the top of the left-side navigation bar.

12.2.4 Step 4: Repeat steps 1 - 3 for each processor VM

Now that you have successfully installed one processor, repeat steps 1 through 3 above with the three remaining Proc VMs and *processor login credentials* listed in your *SIS*. When you're finished, the four Bayware processors should be running on VMs as shown in the table below.

Table 12.1: Mapping Bayware Processors to Virtual Machines

VM	Processor Login Username
aws-p1	proc-1
aws-p2	proc-2
azr-p1	proc-3
gcp-p1	proc-4

You can do this by logging out of `aws-p1` in the terminal window you used above by exiting from `root` and then exiting from the VM altogether by typing

```
]# exit
]$ exit
```

which should get you back to your CCC VM and its prompt

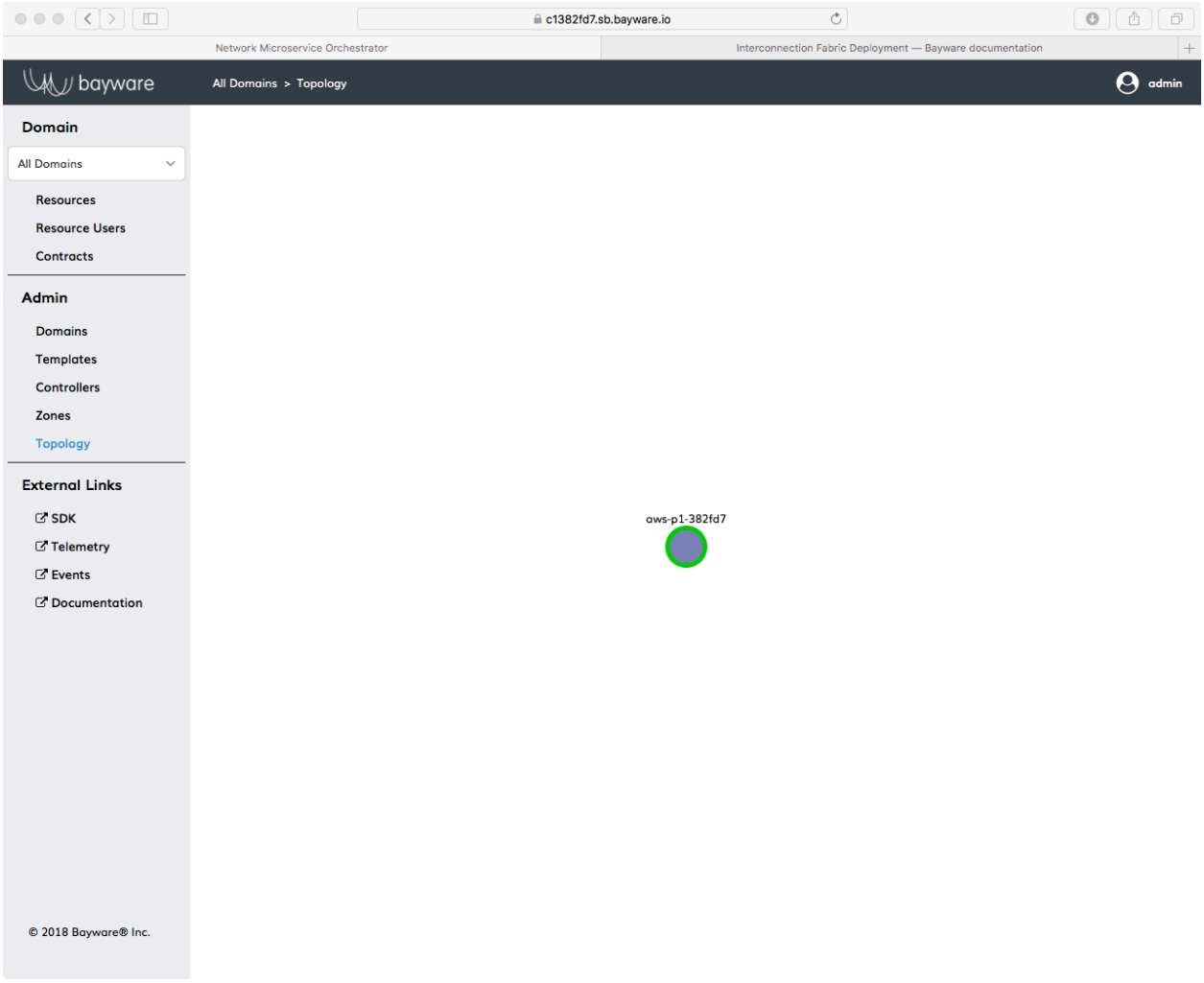


Fig. 12.7: Bayware Processor Installed on `aws-11`

```
[centos@aws-bastion-382fd7 ~]$
```

For the quick studies among you, the essential commands have been reproduced in the *CHEAT SHEET - PROC INSTALL* with a few hints about what goes where. If you're comfortable with the *whys* of all this typing, the cheat sheet simply saves a little scrolling. Otherwise, feel free to go back through each of the steps in detail.

```
CHEAT SHEET - PROC INSTALL

]$ ssh centos@aws-p2                    hint: [ aws-p2, azr-p1, gcp-p1 ]
]$ sudo su -
]# yum install ib_engine openvswitch -y
]# cd /opt/ib_engine/bin
]# ./ib_configure -i                    hint: [ proc-2, proc-3, proc-4 ]
]# systemctl start ib_engine
]# systemctl enable ib_engine
]# exit
]$ exit
```

Once all four engines are installed, return to the orchestrator *Topology* page and *Resources* page to ensure everything went smoothly. You should see that the orchestrator has recognized four processors as shown in Fig. 12.8.

After that, you're ready to move on to creating a full mesh between your processor nodes.

12.2.5 Step 5: Create links between processor

Your service interconnection fabric, currently, consists only of processors with no knowledge of each other nor the ability to communicate with each other. To set up links between the processors that allow communication, you'll work with the orchestrator.

On the browser tab with the orchestrator, click on *Resources* in the left-side navigation bar.

Find the *Node Name* of your first processor node in the first column in the right pane. You're looking for `aws-p1-382fd7`. Click it.

You should see the *Resources* page for node name `aws-p1-382fd7`.

Scroll to the bottom of the page and click *Add Link*.

Fill in the fields as follows (recalling you will have a different *382fd7* suffix):

- *Domain* = `cloud-net`
- *Node* = `aws-p2-382fd7`
- *Tunnel IPs* = `External`
- *IPsec* = `Yes`
- *Admin Status* = `Enabled`

Now click *Submit* in the upper right. You should briefly see a *Success* message and then be directed back to the *Resources* page for `aws-p1-382fd7`. Scroll back to the bottom of the page. There should be a *Link Configuration* entry for `aws-p2-382fd7`.

While you're on the *Resources* page for `aws-p1-382fd7`, go ahead and add two more links, just as you did above, except using *Node Names* `gcp-p1-382fd7` for one link and `azr-p1-328fd7` for the other link.

When you've finished, you should see three entries under *Link Configuration* for `aws-p1-382fd7`.

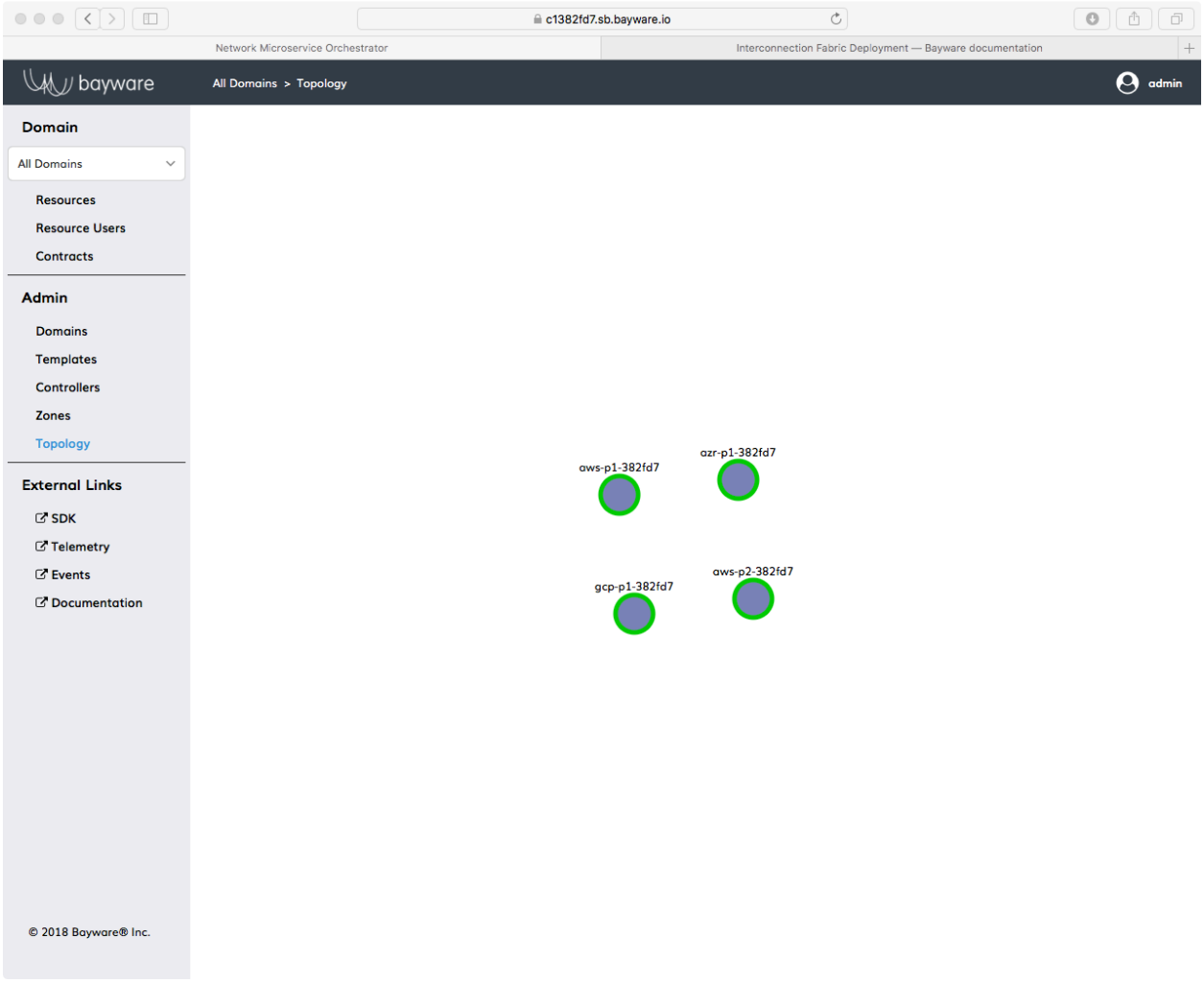


Fig. 12.8: Four Processors on the Orchestrator *Topology* Page

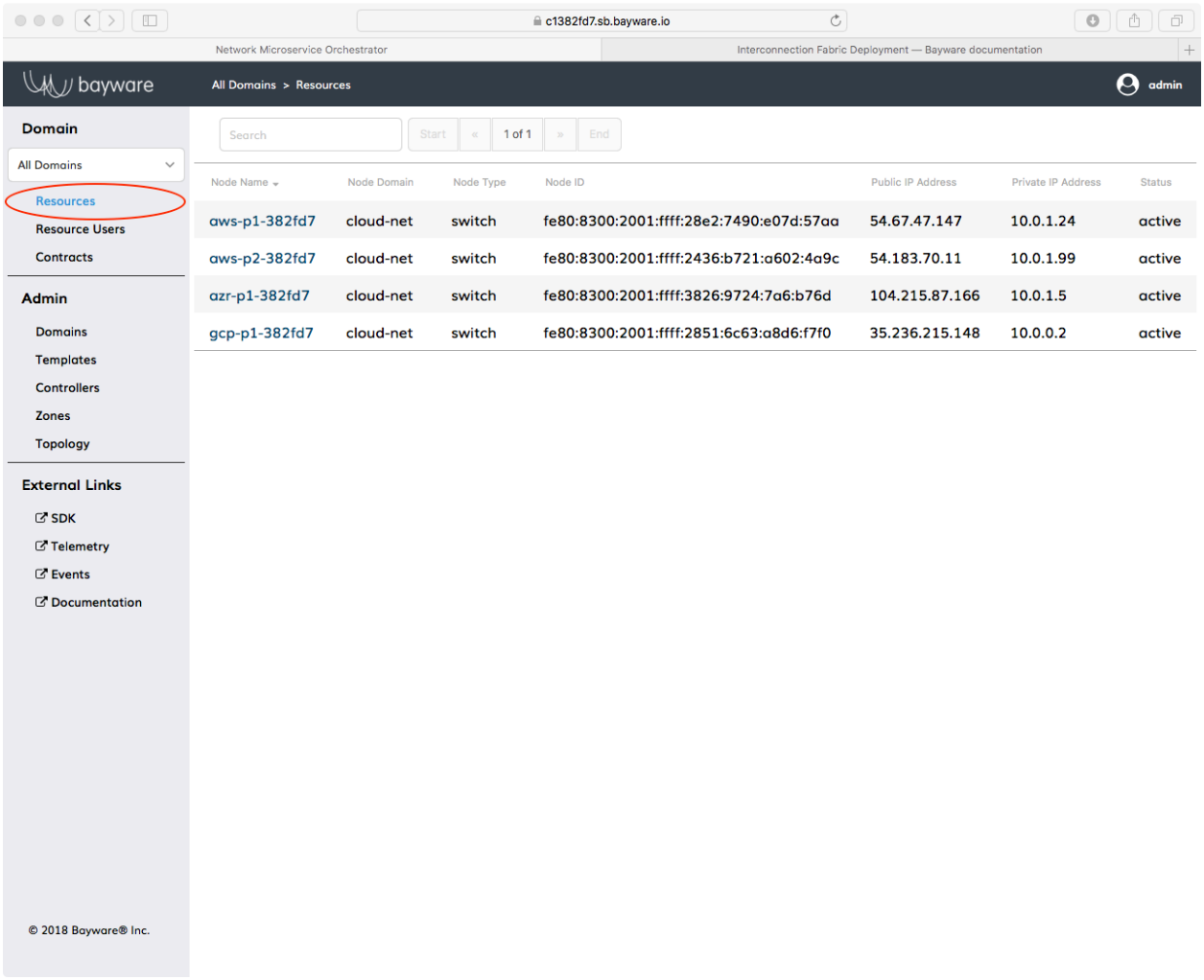


Fig. 12.9: Bayware Orchestrator Resources Button

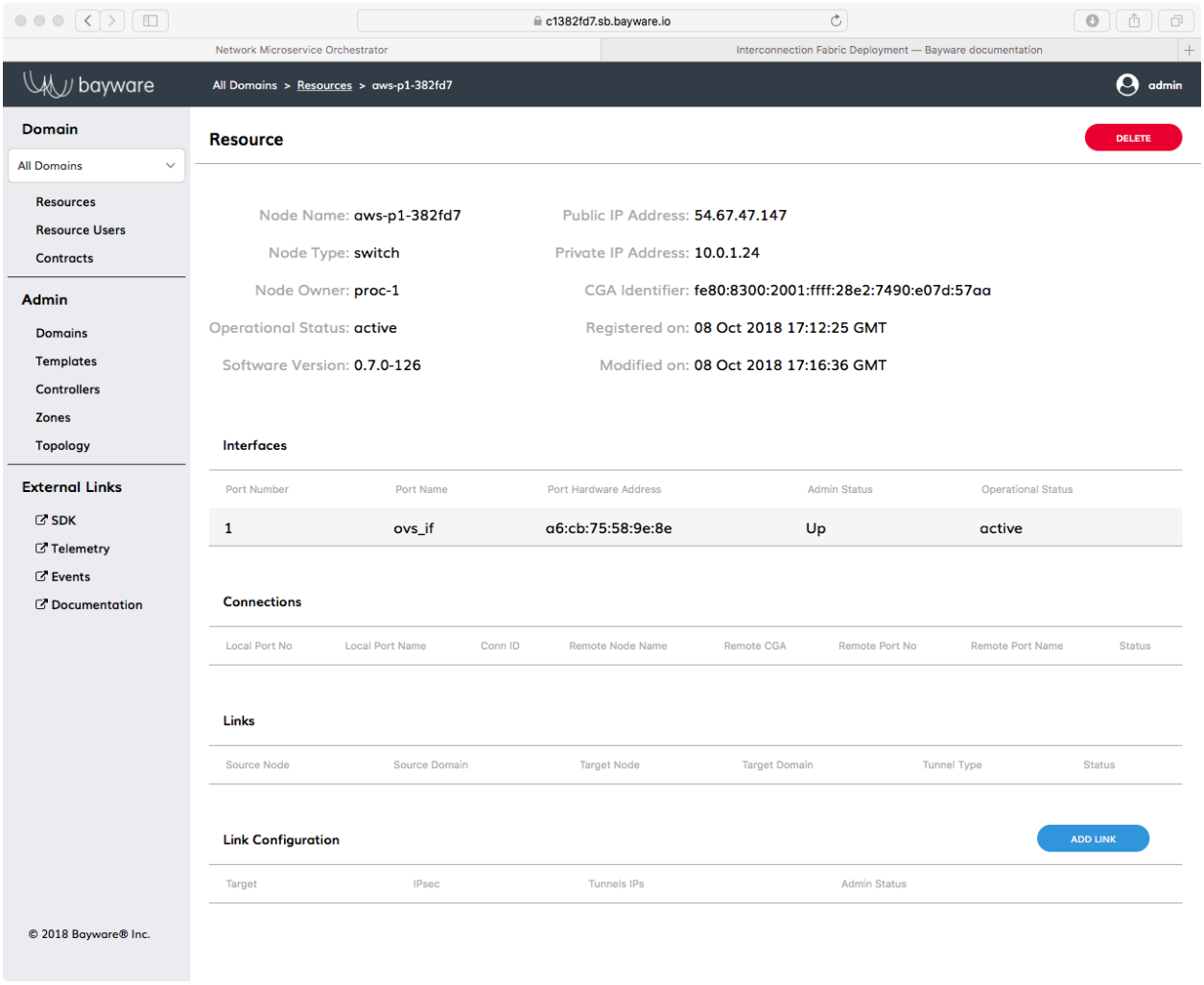


Fig. 12.10: Resources Page For aws-p1-382fd7

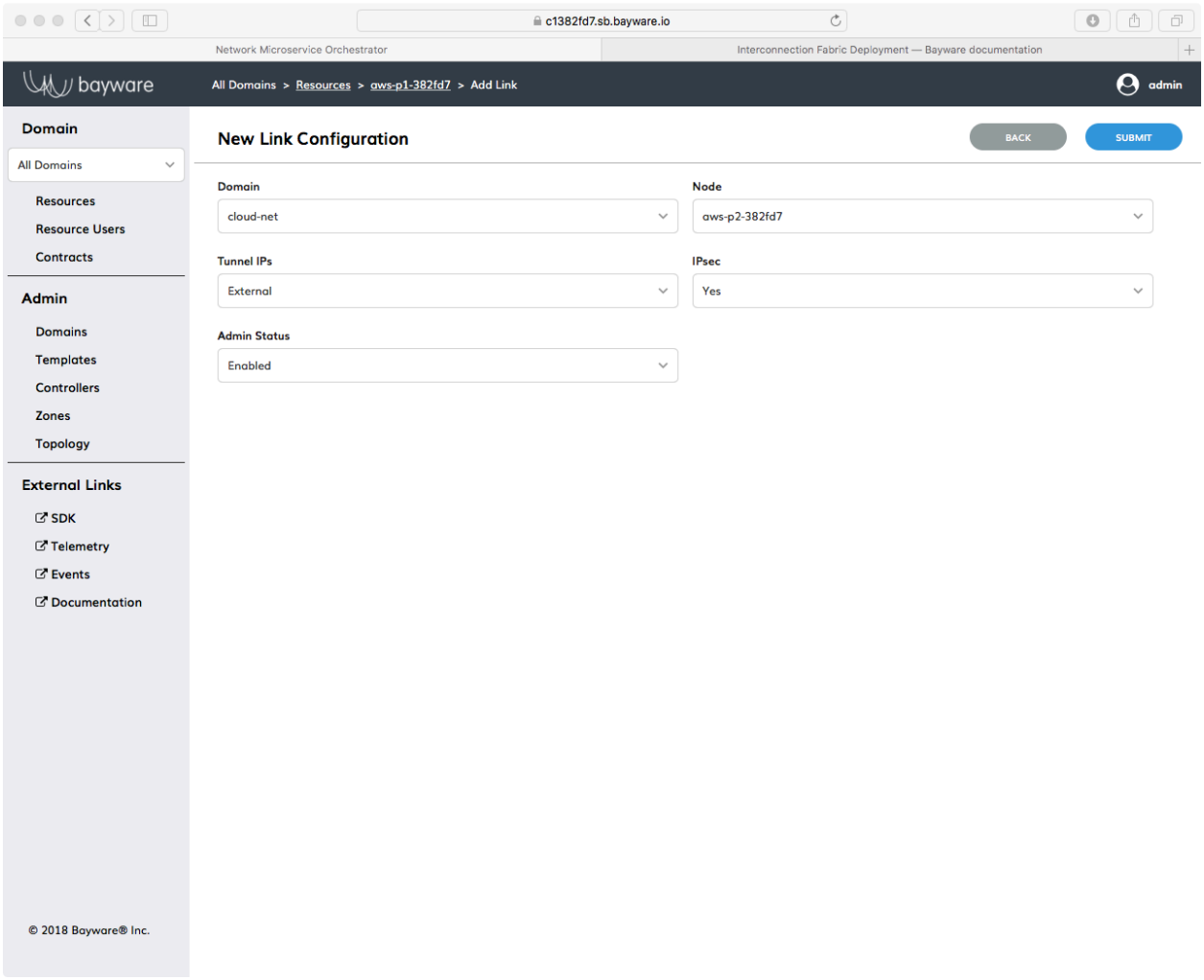


Fig. 12.11: Link Configuration Page For aws-p1-382fd7

Now that three of the six links have been added, there are three more to add.

Click again on *Resources* in the left-side navigation bar. Find `aws-p2-382fd7` and select it. Scroll to the bottom as before and add two links: one between `aws-p2-382fd7` and `azr-p1-382fd7` and another between `aws-p2-382fd7` and `gcp-p1-382fd7`.

When you've finished, you should see three entries under *Link Configuration* for `aws-p2-382fd7`: the link originally created from the `aws-p1-382fd7` *Resources* page and the other two links you just created.

Five down, one to go…

Finally, for a full-mesh interconnect, you need to create a link between `gcp-p1-382fd7` and `azr-p1-382fd7`. You can do this from the *Resources* page of either node by following the previous examples.

When you're finished, navigate back to the *Topology* page of the orchestrator. Now you should see your interconnection network with its four nodes and six links. Note that it may take up to a minute for links to appear between the processor nodes.

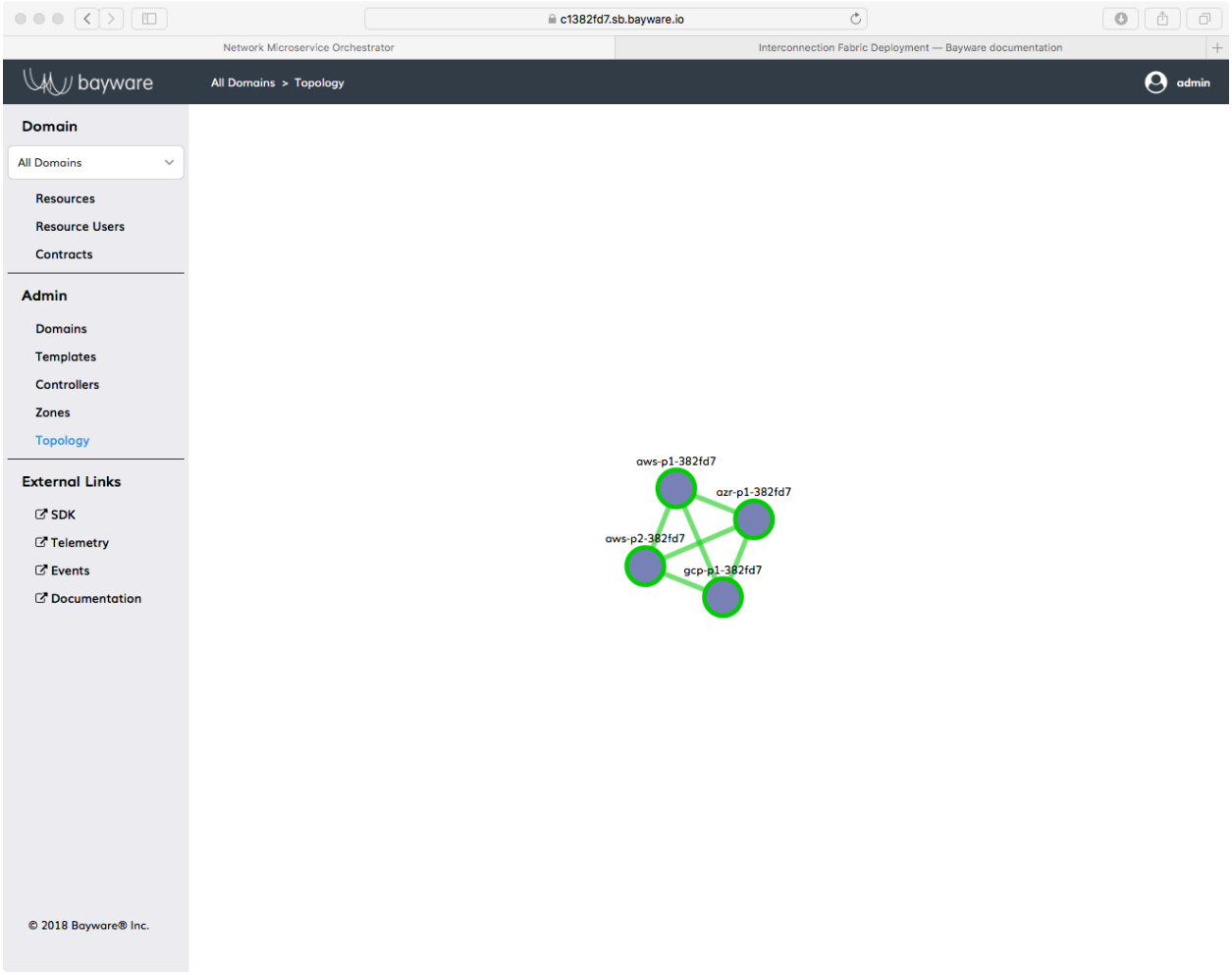


Fig. 12.12: Orchestrator Topology Showing Nodes and Interconnection Links

For yet more proof that something good has happened, go to the *Resources* page for one of the processors using the *Resources* link in the left-side navigation bar.

Tip: You can also find the *Resources* page for a specific processor by clicking on that processor in the

Topology page and then clicking its *Node Name* that appears in the information overlay in the upper-right corner.

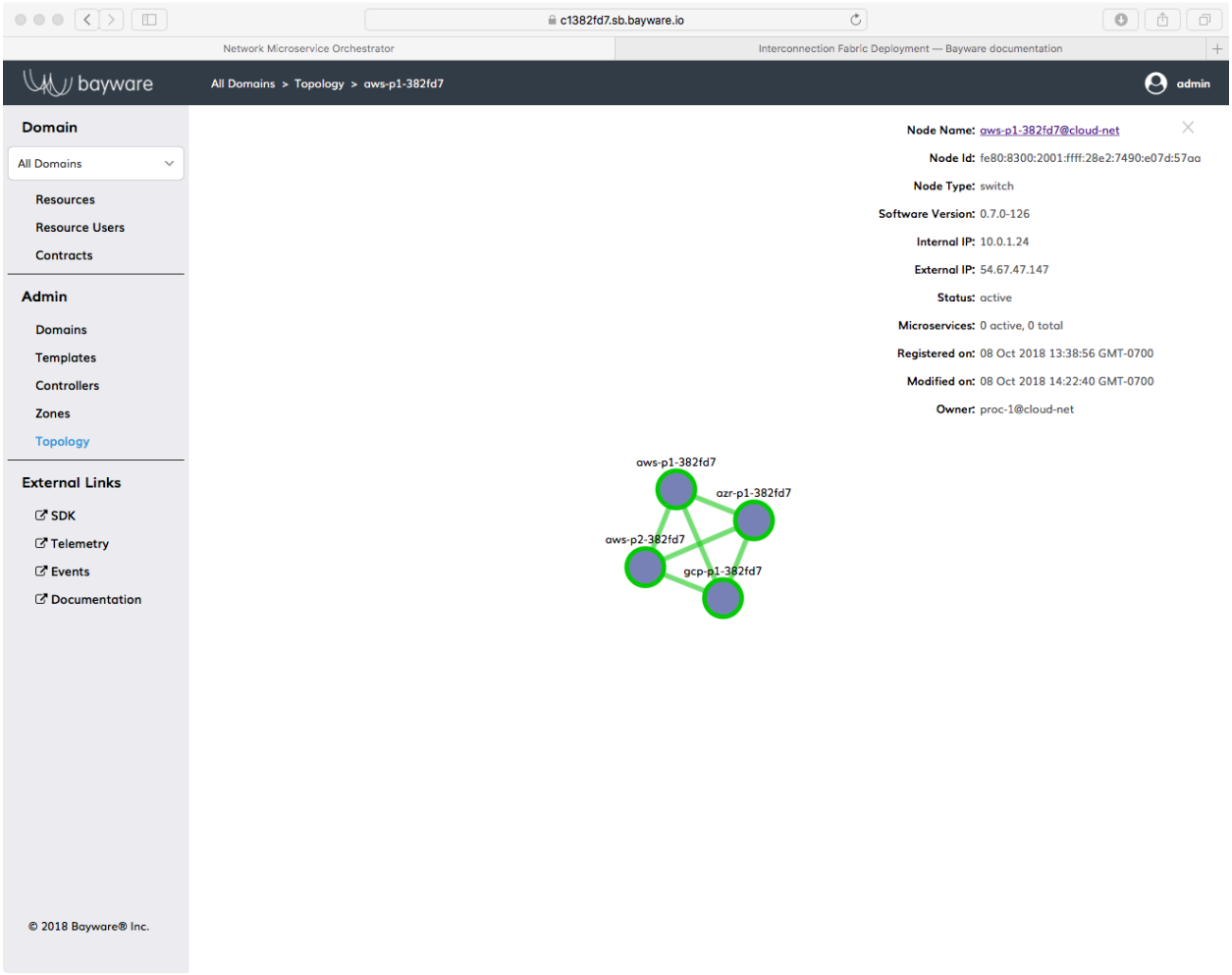


Fig. 12.13: Orchestrator Topology Showing Node Information Overlay

Once back on one of the *Resources* page of one of your processors, you'll note that in addition to the *Link Configuration* information at the bottom, now you should also see three entries each under *Links* and *Connections*.

12.2.6 Summary

In this chapter you installed Bayware processors and Open vSwitch on four VMs in your infrastructure. You used the orchestrator *Topology* and *Resources* buttons extensively to monitor the installation processor. You finally created a full-mesh between all four processor nodes with a few simple clicks on the orchestrator.

Next up: install your first application, Getaway App…

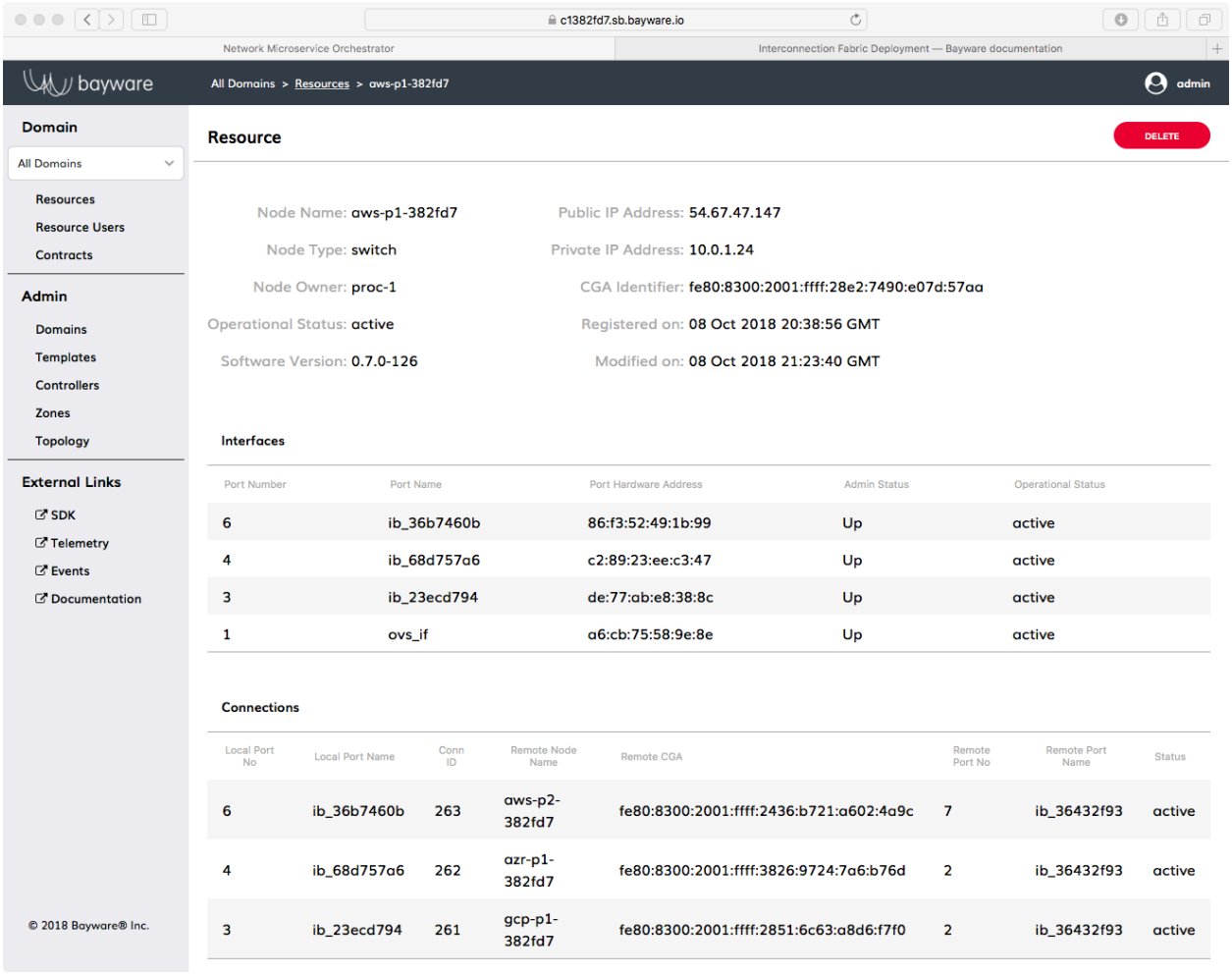


Fig. 12.14: Node Resources Links & Connections

12.3 Application 1 - Getaway App

12.3.1 A Microservice-Based Application & Its Components

Service Graph

Your applications are the lifeblood of your organization. Whether you are supporting internal or external customers, when an app is down your company is losing money. But now that you've deployed Bayware's service interconnection fabric, deploying your application is a snap.

You need to deploy a travel application called Getaway App for your customers. This microservice-based application can be represented with the following service graph.

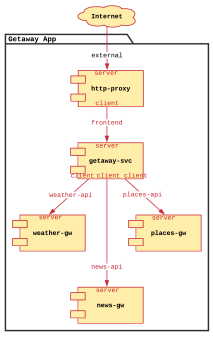


Fig. 12.15: Getaway App Service Graph

Your users, armed with their smart phones and web browsers, first hit the `http-proxy` microservice in front of the firewall. The `http-proxy` requests information from the `getaway-svc` that, in turn, collates data collected from `news-gw`, `places-gw`, and `weather-gw` and returns the information to `http-proxy` for display to the customer.

Nodes & Host Owners

What does this mean for your service interconnection fabric? It's simple. You need to deploy the five Getaway App microservices on five different VMs in your cloud infrastructure. Next to each microservice, you will install a Linux interface and a small daemon called the Bayware agent. The Bayware agent handles registration, authentication, and microcode delivery in conjunction with the orchestrator for your microservice. The agent uses the new Bayware interface to communicate with the service interconnection fabric.

So before we actually login to your VMs and deploy code, we need to tell the orchestrator about your service graph so all the registration, authentication, and microcode delivery is handled properly.

To that end, each service graph node (microservice-agent pair) requires a *Resource User* with a role of *hostOwner*. Go back to the orchestrator tab on your web browser and click *Resource Users* in the left-side navigation bar to see a list of resource users that have been pre-populated in your system as shown in Fig. 12.16.

During Getaway App installation, you will be using the five resource users of type *hostOwner* with the usernames shown here

- `http-proxy`
- `getaway-svc`
- `news-gw`

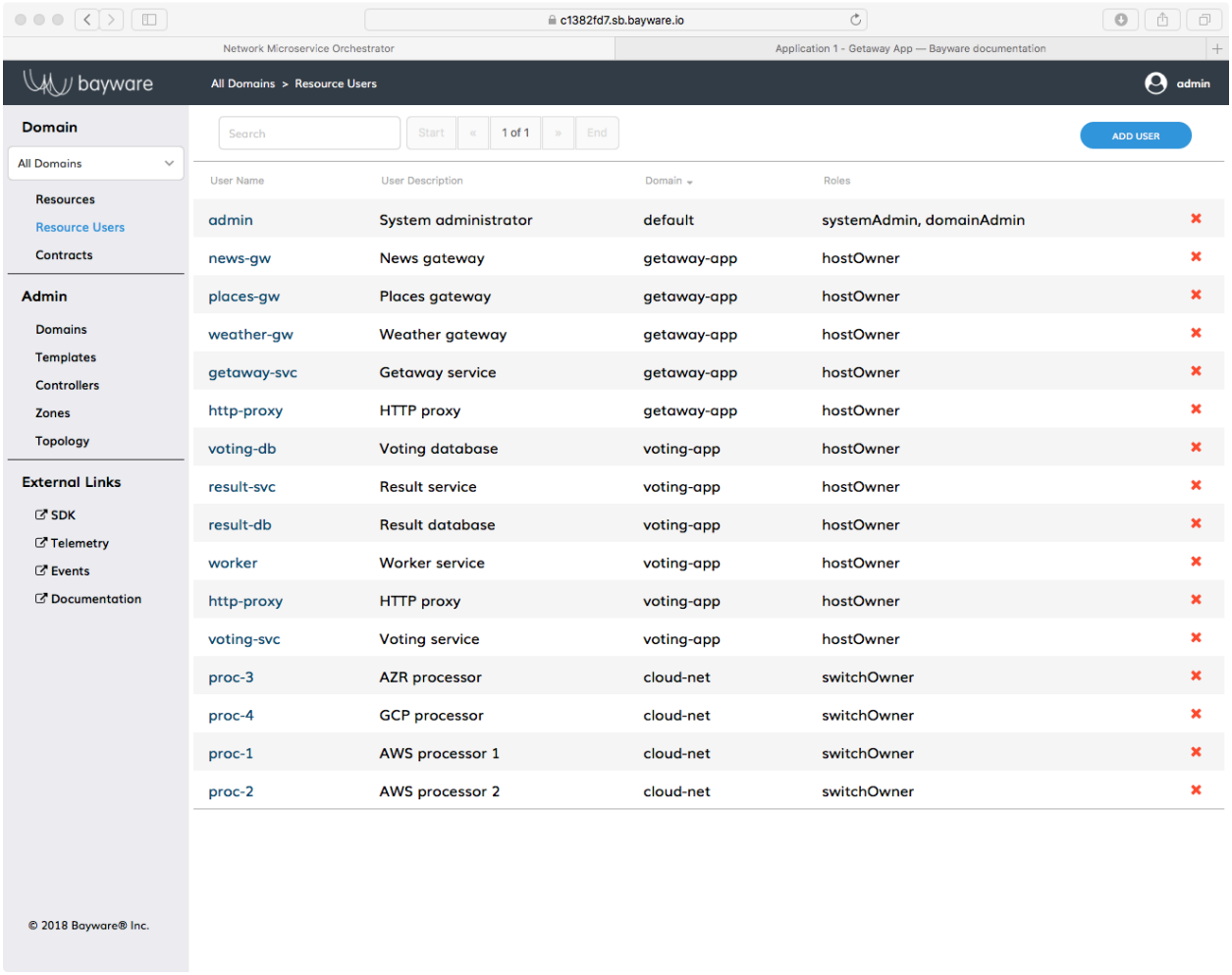


Fig. 12.16: A List of Resource Users Pre-Populated on the Orchestrator

- `places-gw`
- `weather-gw`

and all are in the domain `getaway-app`.

It's no coincidence, then, that these usernames, along with their passwords, are shown again in *rows 23 through 27* of the table in your *SIS*. You'll need this information once we start installing.

Important: The *nodes* in a service graph map to *host owners* on the Bayware orchestrator.

Edges & Contracts

With host owners defined above, your microservices now have a place to live, but they still don't have any way to communicate. A service graph needs edges in addition to nodes.

The edges in your service graph define the communicative relationships between your microservices. At Bayware, we call these communicative relationships *Contracts*.

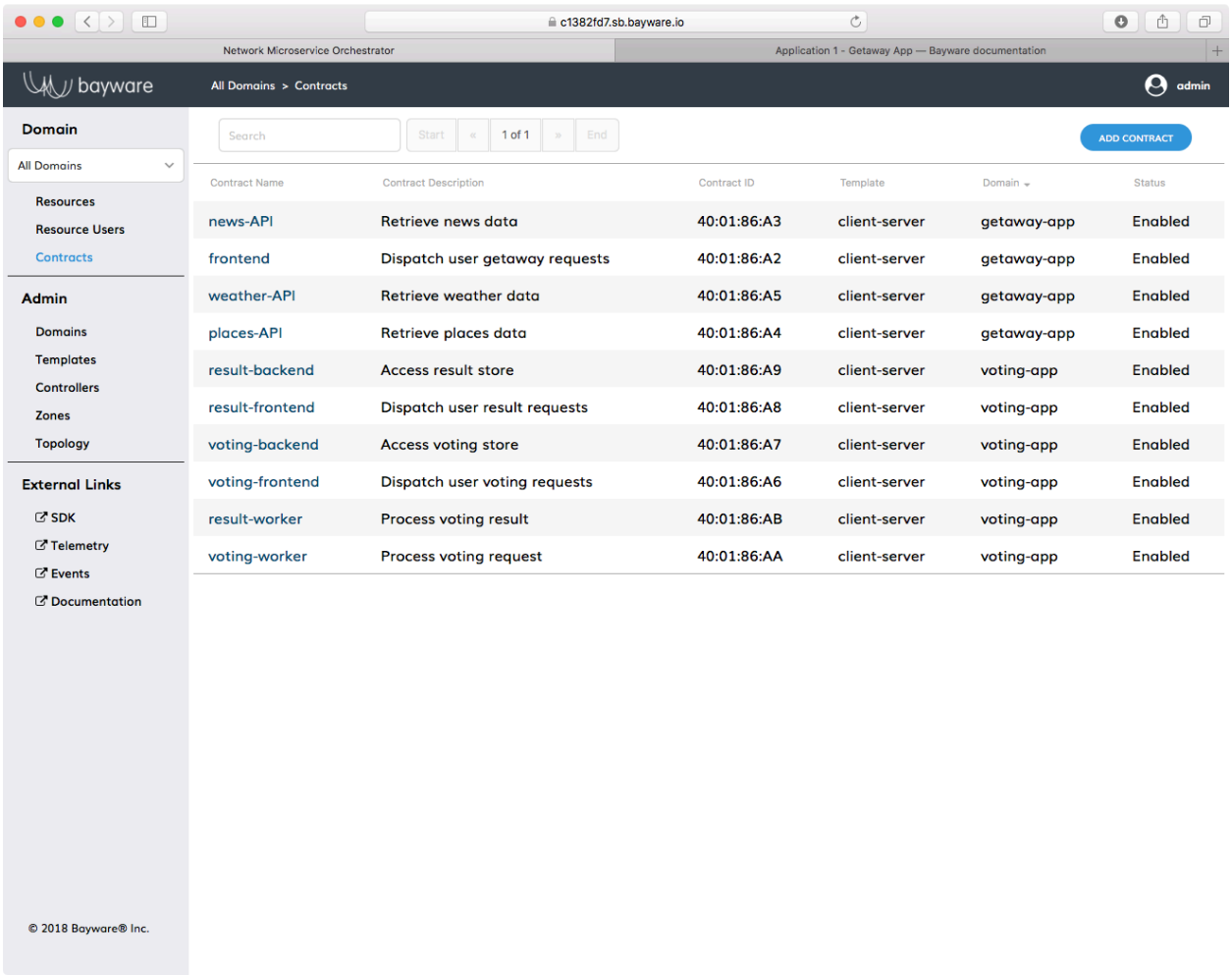


Fig. 12.17: Contracts Used in this Tutorial

As the Getaway App has four edges, your deployment has four contracts. Back on the orchestrator in your web browser, click on *Contracts* in the left-side navigation bar. You'll find four *Contract Names* that reside in the `getaway-app` domain as shown in Fig. 12.17. These are

- `frontend`
- `news-API`
- `places-API`
- `weather-API`

The contracts have been pre-populated for your convenience, but they are easily generated on the fly. We won't do it now, but one would simply click *Add Contract* and fill in a few fields. The chosen *Template* determines the allowed *Roles* in the contract–although, you can add additional *Roles* later. Once you have submitted the contract, you assign *Roles* by clicking on the contract name in the *Contracts* pane and clicking on the *Roles* in the bottom of the window.

Go ahead and do a little exploring now so you can see how the relationships in your service graph have shaped the contracts on the orchestrator. Your contract roles are related to your microservices using the *hostOwner* names described in the previous section. Note that all Getaway App contracts use the *client-server* template, which have role `client` and role `server`.

Table 12.2: Getaway Contracts & Roles

Contract Name	Client Role	Server Role
`frontend`	`http-proxy`	`getaway-svc`
`news-API`	`getaway-svc`	`news_gw`
`places-API`	`getaway-svc`	`places_gw`
`weather-API`	`getaway-svc`	`weather_gw`

Important: The *edges* in a service graph map to *contracts* on the Bayware orchestrator.

Authentication, Registration, & Microcode

Now that you understand host owners and contracts (nodes and edges), let's take a look at what's going to happen behind the scenes.

As described above, after installing a microservice on a VM, an interface and small daemon known as the Bayware interface and agent are installed next to the microservice. During agent configuration, the user supplies the host owner's `domain`, `username`, and `password` previously created on the orchestrator. The agent then reaches out to the orchestrator, but is redirected to the identity service, which authenticates the agent. After successful *authentication*, the agent obtains tokens that enable access to orchestrator services.

The orchestrator now supplies the agent with a net prefix from which the agent generates its own host identifier–a Cryptographically Generated Address (CGA) in the format of an IPv6 address. Using this CGA, the agent requests *registration* at the orchestrator, which kicks off the node discovery protocol that ultimately attaches the VM to the service interconnection fabric through the Bayware interface.

The orchestrator, at this point, has also pushed *microcode* associated with contract roles described above to the agent. The agent, sitting between the microservice and the Bayware interface, embeds the microcode into the data stream to create self-steering flows.

12.3.2 Installation

Fig. 12.18 shows how you will deploy Getaway App microservices. As shown, `aws-11` and `aws-12` will attach to the processor running on `aws-p1` and run the `http-proxy` and `getaway-svc` microservices respectively. The `places-gw`, `weather-gw`, and `news-gw` will be deployed on `aws-21`, `aws-31`, and `aws-41` respectively. These three nodes are all attached to the processor running on `aws-p2`.

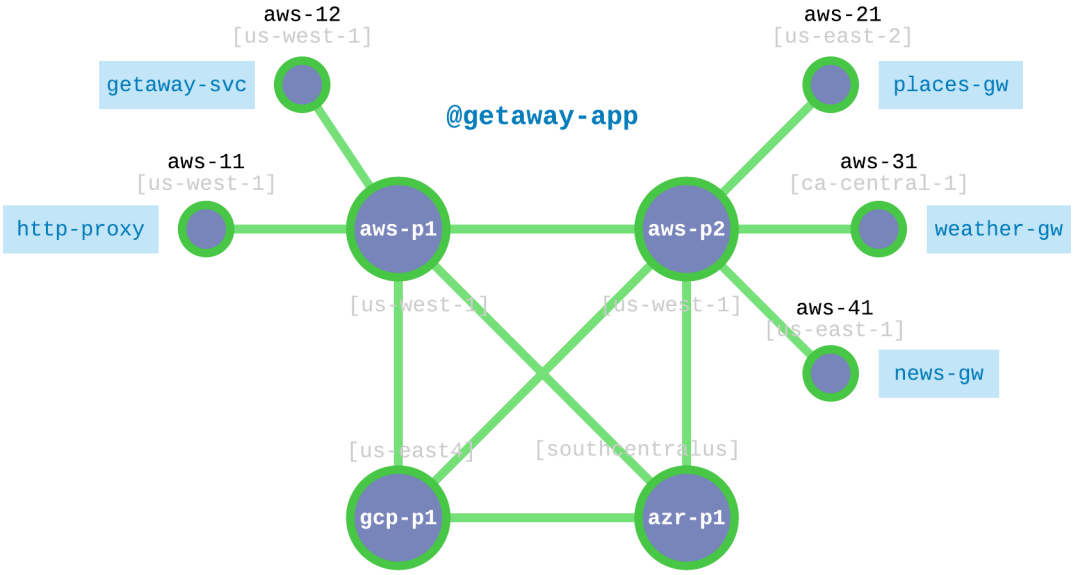


Fig. 12.18: Getaway Microservices Nodes and the Interconnection Fabric

Each of the five microservices runs alongside an instance of the Bayware agent, which connects to the service interconnection fabric through the Bayware interface. Each microservice runs as a service unit under the RHEL 7 **`systemd`** system and service manager. Installation steps are provided in detail below.

But wait. Here's a detail that will become interesting later once you get to *Hybrid Cloud - Moving Microservices*. Fig. 12.18 and Table 12.3 reveal that you will be running all your microservices in AWS. Make a mental note of that.

Table 12.3: Getaway Microservices VM Mapping

Microservice	VM	Service Unit
`http-proxy`	`aws-11`	getaway-proxy
`getaway-svc`	`aws-12`	getaway-service
`news-gw`	`aws-41`	getaway-news
`places-gw`	`aws-21`	getaway-places
`weather-gw`	`aws-31`	getaway-weather

To deliver the microservices to their VMs, you'll perform the following steps

1. Login to a workload VM's OS
2. Install, configure, & start agent
3. Install and start service unit
4. Repeat 1 - 3 for remaining microservices

5. Interact with Getaway App

Before you begin, take a look at the *Resources* page again on the orchestrator. You should only see four nodes all of type *switch*. Once you add workloads, they will also appear under *Resources*, but as type *host*.

Step 1: SSH to VM

Let's start with the `http-proxy` microservice, which needs to be installed on `aws-11`. Starting from the prompt on your *Command Center*, login to `aws-11`.

```
]$ ssh centos@aws-11
```

You should now see a prompt on `aws-11` similar to

```
[centos@aws-11-382fd7 ~]$
```

The following commands all require super-user credentials, so become `root`

```
[centos@aws-11-382fd7 ~]$ sudo su -
```

You should now see a prompt on `aws-11`

```
[root@aws-11-382fd7 ~]#
```

(The `root` prompt above is abbreviated simply as `]#` in the commands that follow.)

Step 2: Install, configure, & start agent and interface

Installation of the agent and interface requires access to the `bayware-repo` and the `epel-release` repository. These have been preinstalled on your virtual machines.

Execute the following command at the `root` prompt to install the agent on `aws-11`:

```
]# yum install -y ib_agent
```

Similar to configuring the processors in your interconnection fabric in the *SIF Deployment* chapter, you will now run the configuration script interactively in order to configure the agent that will run next to the microservice and its new Linux interface.

First, to save some typing, let's `cd` to the directory that contains the script

```
]# cd /opt/ib_agent/bin
```

The script is called **`ib_configure`**. You can display the usage instructions by typing

```
]# ./ib_configure -h
usage: ib_configure [-h] [-i] [-o OUTFILE] [-s] [-c ORCHESTRATOR] [-d DOMAIN]
                    [-l USERNAME] [-p PASSWORD] [-a IP_ADDR] [-t TCP_PORT]

Configure the Bayware Agent.

optional arguments:
  -h, --help            show this help message and exit
  -i, --interactive     set up configuration parameters interactively
  -o OUTFILE, -f OUTFILE, --outfile OUTFILE
```

(continues on next page)

(continued from previous page)

```
                      create text file listing all configuration parameters
 -s, --ipsec          set up IPsec for this Agent
 -c ORCHESTRATOR, --orchestrator ORCHESTRATOR
                      orchestrator FQDN or IP address
 -d DOMAIN, --domain DOMAIN
                      domain in which host owner of this node was created on
                      the orchestrator
 -l USERNAME, --username USERNAME
                      username for the host owner of this node that was
                      configured on the orchestrator
 -p PASSWORD, --password PASSWORD
                      password associated with the username of the host
                      owner of this node that was configured on the
                      orchestrator
 -a IP_ADDR           IPv4 address of REST interface on which Agent listens
 -t TCP_PORT          TCP port of REST interface on which Agent listens
```

When you run the script in interactive mode, you will be prompted for orchestrator FQDN, and the `domain`, `username`, and `password` assigned to the `http-proxy` microservice.

- orchestrator IP or FQDN: use `c1382fd7.sb.bayware.io` as shown in the *URL table* at the top of the *SIS*. Your FQDN prefix will be different than `c1382fd7` shown here. Do *not* include the `https://` that is present in the URL.
- node `domain`: use the `http-proxy domain` from *agent login credentials*, row 23
- node `username`: use the `http-proxy username` from *agent login credentials*, row 23
- node `password`: use the `http-proxy password` from *agent login credentials*, row 23
- configure IPsec: answer `YES`

Note that, for convenience, the `username` is the same as the name of the microservice.

With that information in hand, execute the following command and follow the prompts.

```
]# ./ib_configure -i
```

Now start and enable the agent

```
]# systemctl start ib_agent
]# systemctl enable ib_agent
```

If all went well, you should see a small, green circle appear on the *Topology* page of the orchestrator as shown in Fig. 12.19. The agent should also automatically form a link with your processor, `aws-p1-382fd7`, in your service interconnection fabric. As you install agents on your workload nodes and their small, green circles appear in the *Topology* window, clicking on a circle brings up overlay information related to the node. The *Owner* information at the bottom of the overlay shows the host owner and domain used on the given resource.

Step 3: Install and start service unit

As shown in Table 12.3, you need to install the microservice `http-proxy` on `aws-11`. The package (service unit) associated with this microservice is called getaway-proxy. While you're still signed in as `root` execute the following three commands

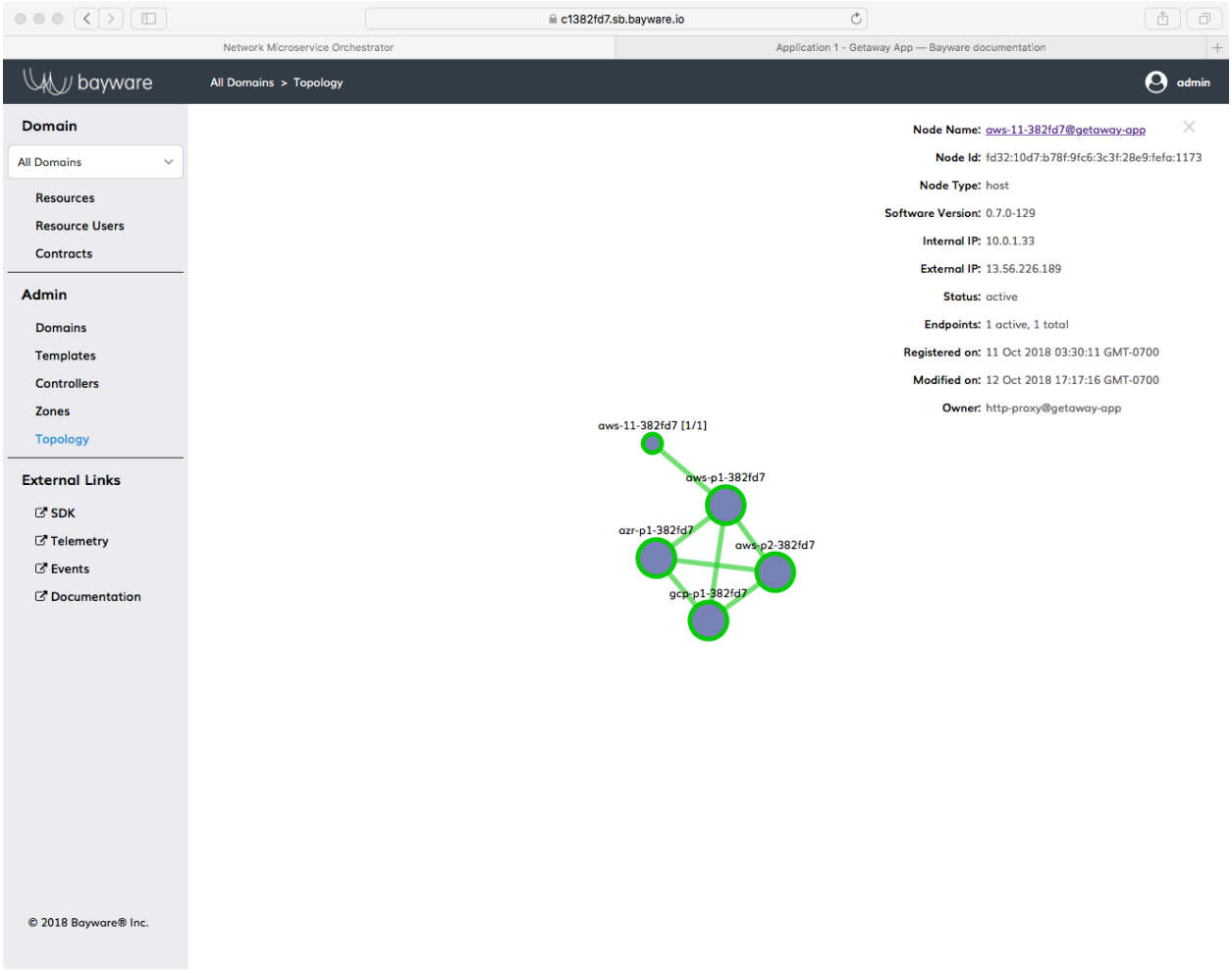


Fig. 12.19: Agent Installed on `aws-11` in Getaway App

```
]# yum install -y getaway-proxy
]# systemctl start getaway-proxy
]# systemctl enable getaway-proxy
```

Step 4: Repeat 1 - 3 for remaining microservices

Easy, right?

One down, four to go... finish installing the remaining components of your application by following steps 1 through 3 above for the microservices `getaway-svc`, `news-gw`, `places-gw`, and `weather-gw`. Remember to follow the microservice-to-VM mapping shown in *Getaway Microservices VM Mapping* above.

Hint: It's like a test, right? Even if it's open-book and you can scroll back through the detailed descriptions above (and you probably should if you want to recall *why* you're typing something), we've created a *CHEAT SHEET* to help you install the remaining four microservices without scrolling. Just be sure to have your **personal** SIS open for the `ib_configure` part as you'll need to lookup the *login credentials* for the host owner user names hinted at. The *CHEAT SHEET* assumes you are starting from your CCC.

```
CHEAT SHEET - GETAWAY APP INSTALL

hint: use elements in the lists [] below for each iteration

]$ ssh centos@aws-12              hint: [ aws-12, aws-41, aws-21, aws-31 ]
]$ sudo su -
]# yum install -y ib_agent
]# cd /opt/ib_agent/bin
]# ./ib_configure -i              hint: [ getaway-svc, news-gw, places-gw, weather-gw ]
]# systemctl start ib_agent
]# systemctl enable ib_agent
]# yum install -y getaway-service     hint: [ getaway-service, getaway-news, getaway-
↪places, getaway-weather ]
]# systemctl start getaway-service    hint: [ getaway-service, getaway-news, getaway-
↪places, getaway-weather ]
]# systemctl enable getaway-service   hint: [ getaway-service, getaway-news, getaway-
↪places, getaway-weather ]
]# exit
]$ exit
```

When you're finished installing all microservice components, take a look again at the *Topology* page and ensure that there are five microservices and that each is connected to one of the processors as shown in Fig. 12.20. (You can get overlay information about a green circle on the *Topology* page by clicking it.)

Step 5: Interact with the Getaway App

Getaway App

Now your Getaway App–and all its microservices–is up and running on Bayware technology. In the next section, you will have an opportunity to play with some of the innovative features this brings to your application.

But first, let's spend some time with the Getaway App itself since, in the end, that is the whole point of application deployment.

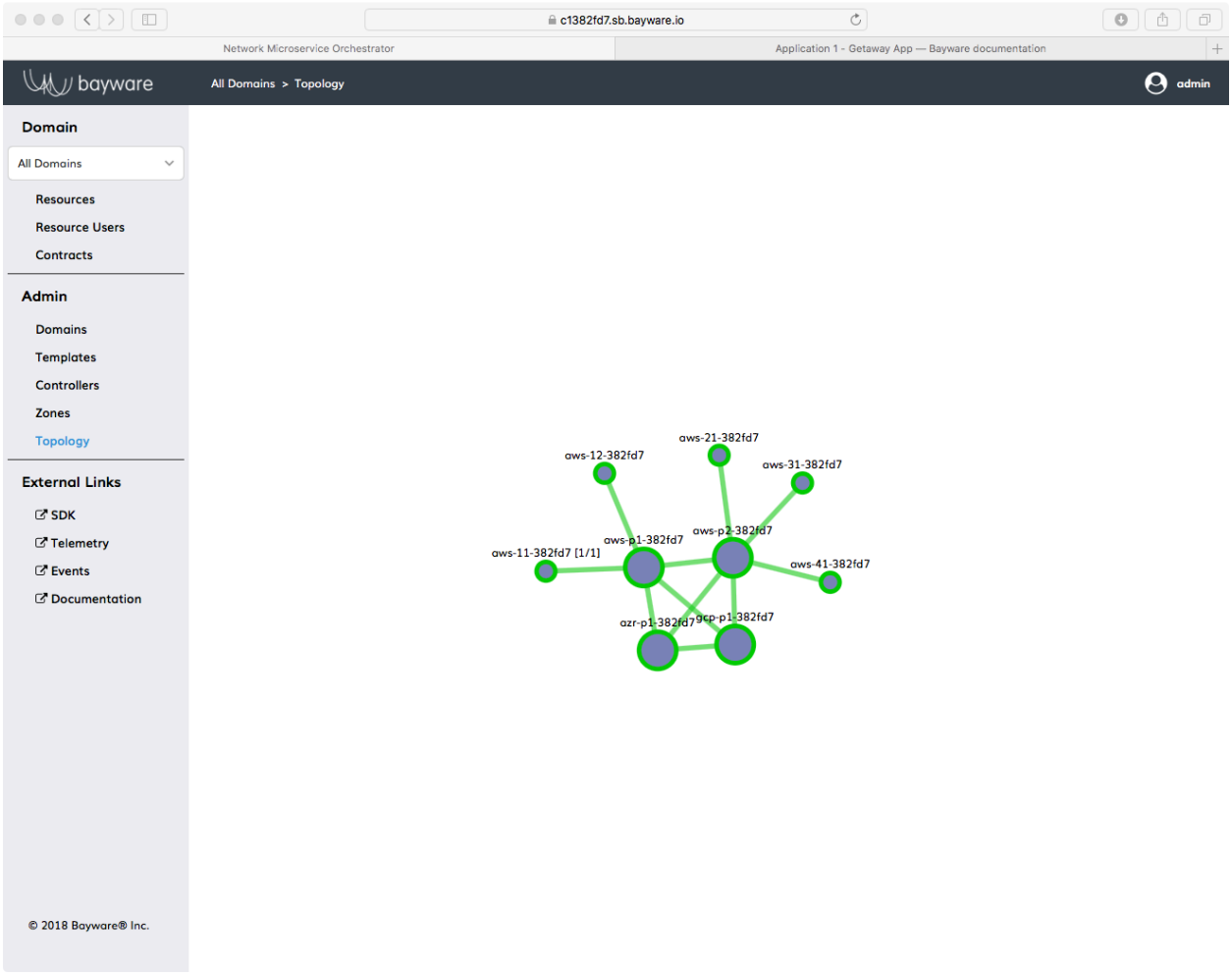


Fig. 12.20: Orchestrator *Topology* for Getaway App

Open a new tab in your browser and type in the web address for the Getaway App Entry Point shown in the *URL table* at the top of your *SIS*, similar to `https://ap382fd7.sb.bayware.io/getaway`.

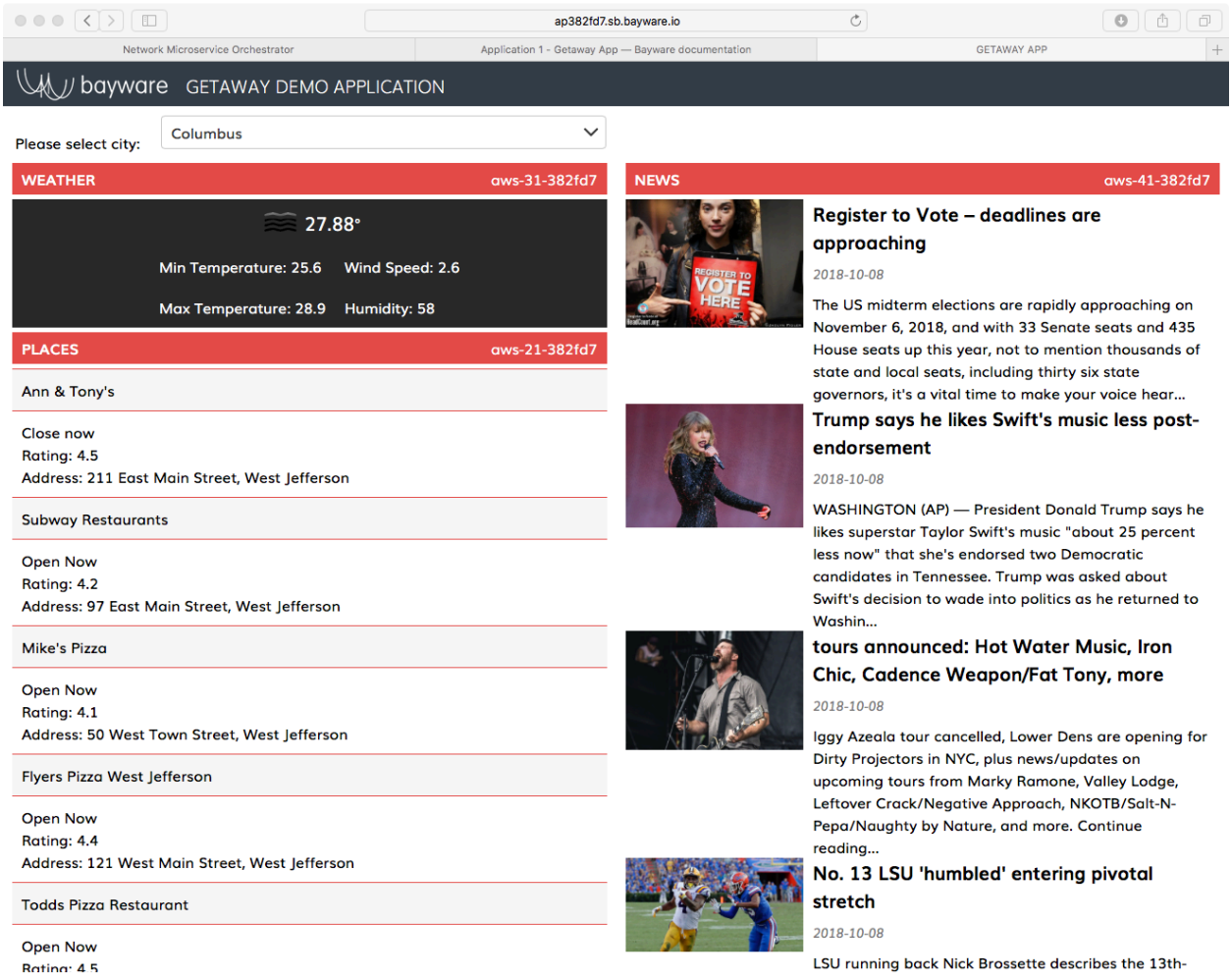


Fig. 12.21: Getaway App Running in a Browser

As shown in Fig. 12.21, three panes show weather, places, and news (operated by the microservices `weather-gw`, `places-gw`, and `news-gw` respectively) for a particular city. You can choose the city using the drop-down selector box at the top of the page.

Of primary interest to the devOps professional–and irrelevant to a real end user–is the notation on the right end of each title bar for *WEATHER*, *PLACES*, and *NEWS* that shows the VM on which the given microservice is running. If you followed the tutorial to this point, you should see *NEWS* running on `gcp-11-382fd7`, *PLACES* running on `aws-21-382fd7`, and *WEATHER* running on `aws-31-382fd7`. In the next section, we'll show how easy it is to direct microservice requests to a different VM no matter the public cloud provider.

Service Graph Revisited

With all the security inherent in running Getaway App over a Bayware service interconnection fabric, the orchestrator knows which microservices (*host owners*) are installed and their communicative relationships (*contracts*). As described in *Service Graph* and *Nodes & Host Owners* and *Edges & Contracts*, one starts by adding *hostOwners* and *Contracts* to the orchestrator that correlate to the application service graph nodes and edges.

Now that you've installed all your Getaway App components, you can do a quick sanity check to ensure that you correctly entered host owners and contracts into the orchestrator. Back on the orchestrator, click on *Domains* in the left-side navigation menu. Of the four domain names that are displayed, now click on `getaway-app`. At the bottom of the `getaway-app` domain window, click on *Domain Topology*. You should see Getaway App service graph recreated by the orchestrator as shown in Fig. 12.22.

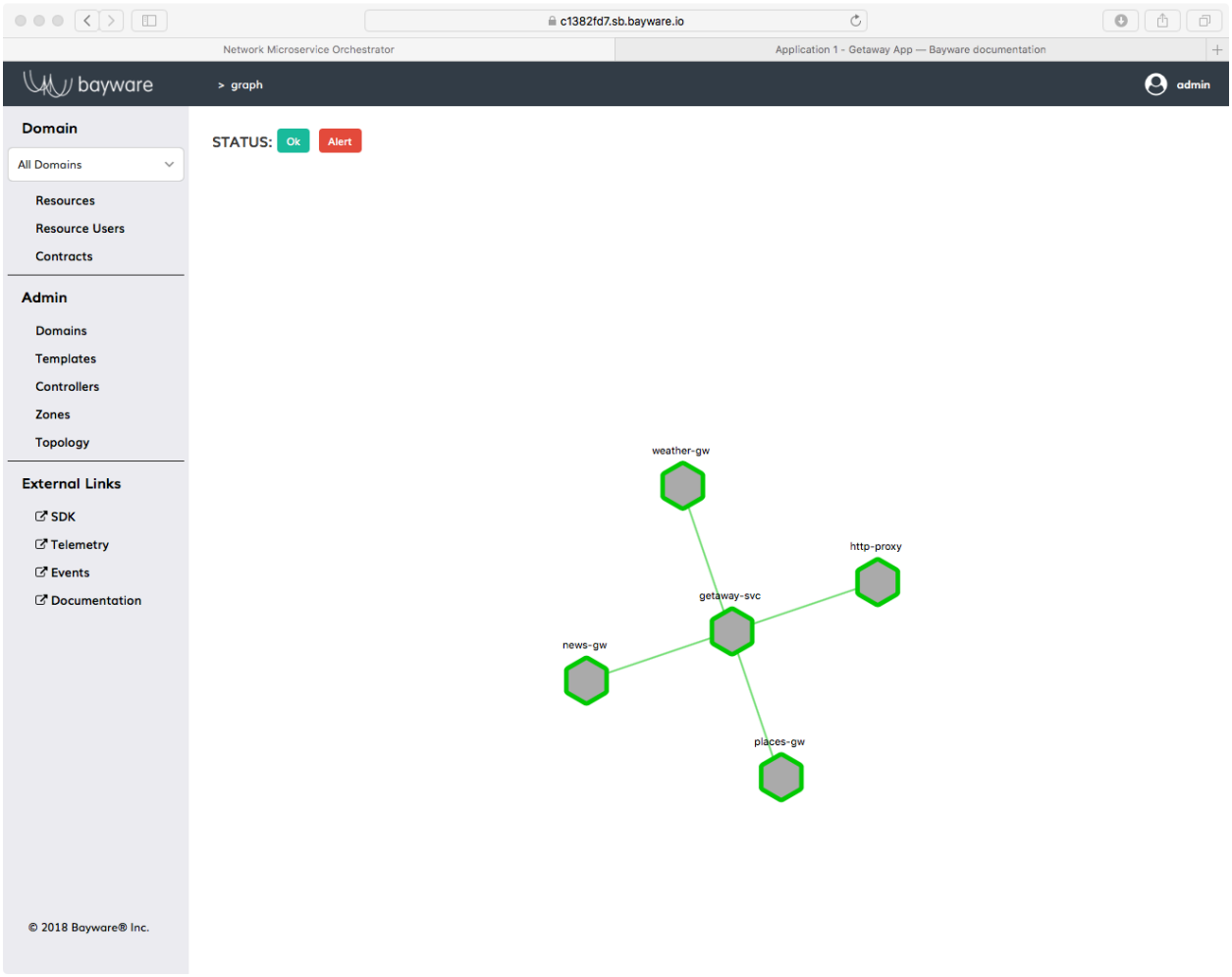


Fig. 12.22: Getaway App Service Graph Generated By Orchestrator

Telemetry

Now might be the right time to revisit telemetry. We introduced Grafana in the *first section* of this tutorial to confirm that 17 virtual machines were really up and running. Now they're actually doing something.

Go to the Grafana page by clicking on *Telemetry* in the left-side navigation menu on the orchestrator. Grafana will open in a new tab in your browser. Start by checking out server `aws-11-382fd7`, which is running microservice `http-proxy`, as shown in Fig. 12.23. Scroll down and you will be able to see both *System* and *Network* dashboards.

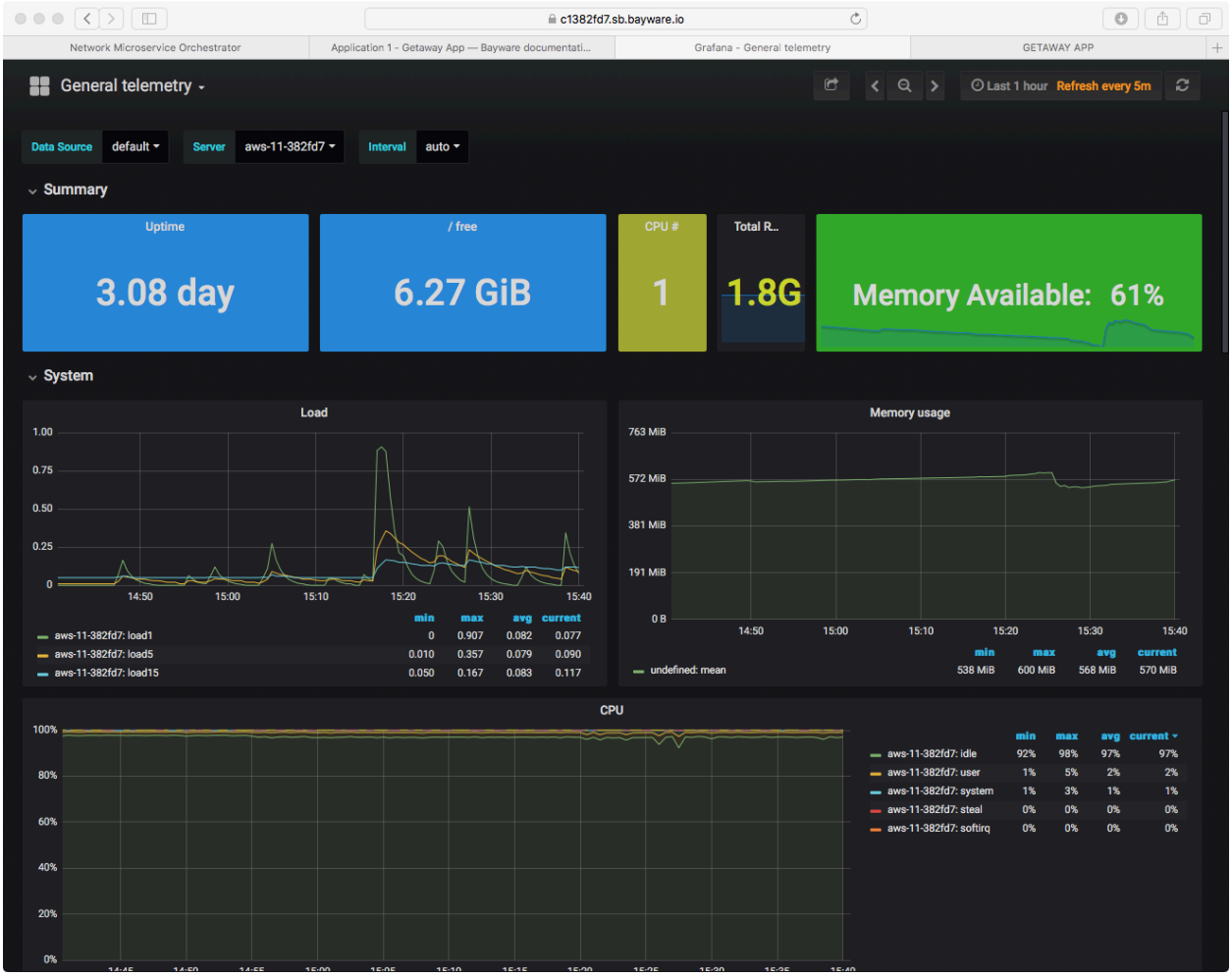


Fig. 12.23: Telemetry: Getaway App `http-proxy` Load

12.3.3 Hybrid Cloud - Moving Microservices

So everything's up and running along smoothly. All your Getaway microservices are deployed alongside an instance of the Bayware agent and interface, you can see your processor nodes and your workload nodes in the orchestrator *Topology* and you can be confident that security is in place and your policies are being enforced.

But you are still at the mercy of your public cloud provider. Maybe the cost will go way up or the level of service will go way down. It would be nice to know that you can easily have a backup microservice at the ready in another cloud provider or two and quickly direct traffic from the current provider to the new provider.

With Bayware you can.

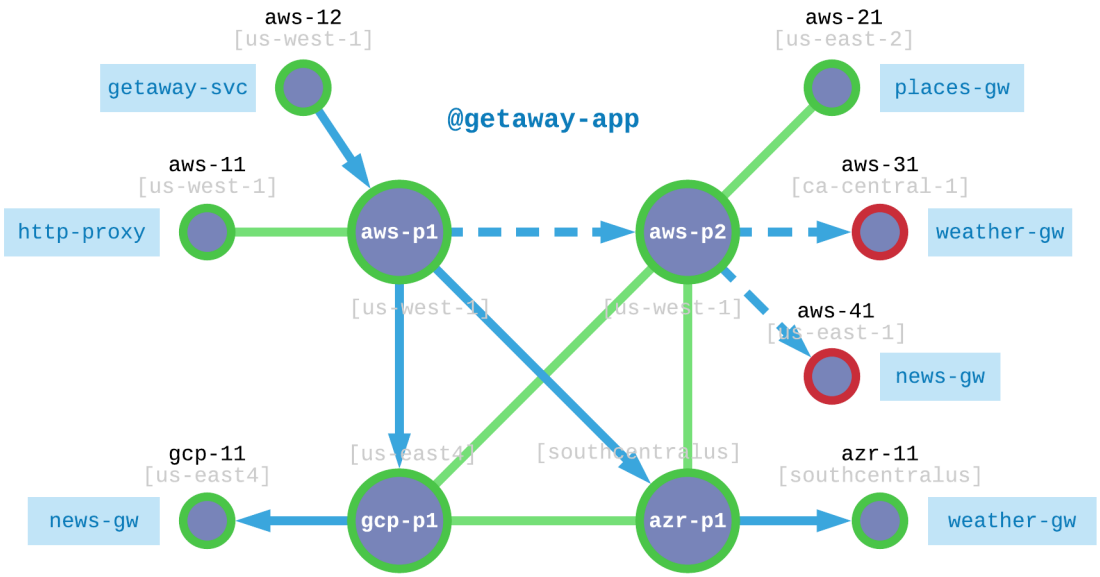


Fig. 12.24: Getaway App Utilizes Redundent, Hybrid-Cloud Microservices

Fig. 12.24 shows where we are and where we're going. *Recall* that all Getaway App microservices were deployed in AWS. Let's focus on `weather-gw` and `news-gw` specifically.

The dashed lines in Fig. 12.24 show how requests from `http-proxy` are currently routed to `aws-31` (for `weather-gw`) and to `aws-41` (for `news-gw`). Again, both of these VMs are operating in AWS.

And then it happens. Some catastrophe befalls `aws-31` and `aws-41` and they go offline. Time is money and the prepared devOps engineer already has backup `news-gw` and `weather-gw` microservices operating in GCP and Azure, respectively. Your customers won't notice a thing.

The Setup

Here's how will simulate such an event.

1. Install `news-gw` and agent on `gcp-11`
2. Install `weather-gw` and agent on `azr-11`
3. Simulate failed `aws-31` by shutting down agent

4. Simulate failed `aws-41` by shutting down agent

Let's get started.

Redundant Installation

Table 12.4 shows the microservices you need to set up on redundant VMs, the name of the new VMs, and the name of the application serivce unit.

Table 12.4: Redundant Microservices VM Mapping

Microservice	VM	Service Unit
`news-gw`	`gcp-11`	getaway-news
`weather-gw`	`azr-11`	getaway-weather

You're likely a pro at this by now. As such, here's another *CHEAT SHEET* to help you install agents and service units in GCP and AZR. If anything seems unclear, go back to the *installation steps* for review. Remember to find the credentials for *news* and *weather* on your SIS, *rows 25 and 27*.

```
CHEAT SHEET - GETAWAY APP MULTI-CLOUD

hint: use elements in the lists [] below for each iteration

]$ ssh centos@gcp-11                  hint: [ gcp-11, azr-11 ]
]$ sudo su -
]# yum install -y ib_agent
]# cd /opt/ib_agent/bin
]# ./ib_configure -i                   hint: [ news-gw, weather-gw ]
]# systemctl start ib_agent
]# systemctl enable ib_agent
]# yum install -y getaway-service      hint: [ getaway-news, getaway-weather ]
]# systemctl start getaway-service     hint: [ getaway-news, getaway-weather ]
]# systemctl enable getaway-service    hint: [ getaway-news, getaway-weather ]
]# exit
]$ exit
```

Go back to the *Topology* page in the orchestrator. You should see new small green circles representing `gcp-11` and `azr-11` as shown in Fig. 12.25.

Stop Systems

Now you're going to take `aws-31` and `aws-41` offline. You'll do this by stopping the `ib_agent` service running on each of these virtual machines. The service interconnection fabric will detect that `news-gw` and `weather-gw` microservices operating on `aws-31` and `aws-41` are no longer registered and traffic will automatically be re-routed to the new microservice instances you just created.

Starting from your *Command Center*, login to `aws-31` and become `root` *just as you did above*.

Now execute the following

```
]# systemctl stop ib_agent
```

You will repeat these steps now on `aws-41`. First, get back to your CCC by exiting out of the two shells you've opened on `aws-31`

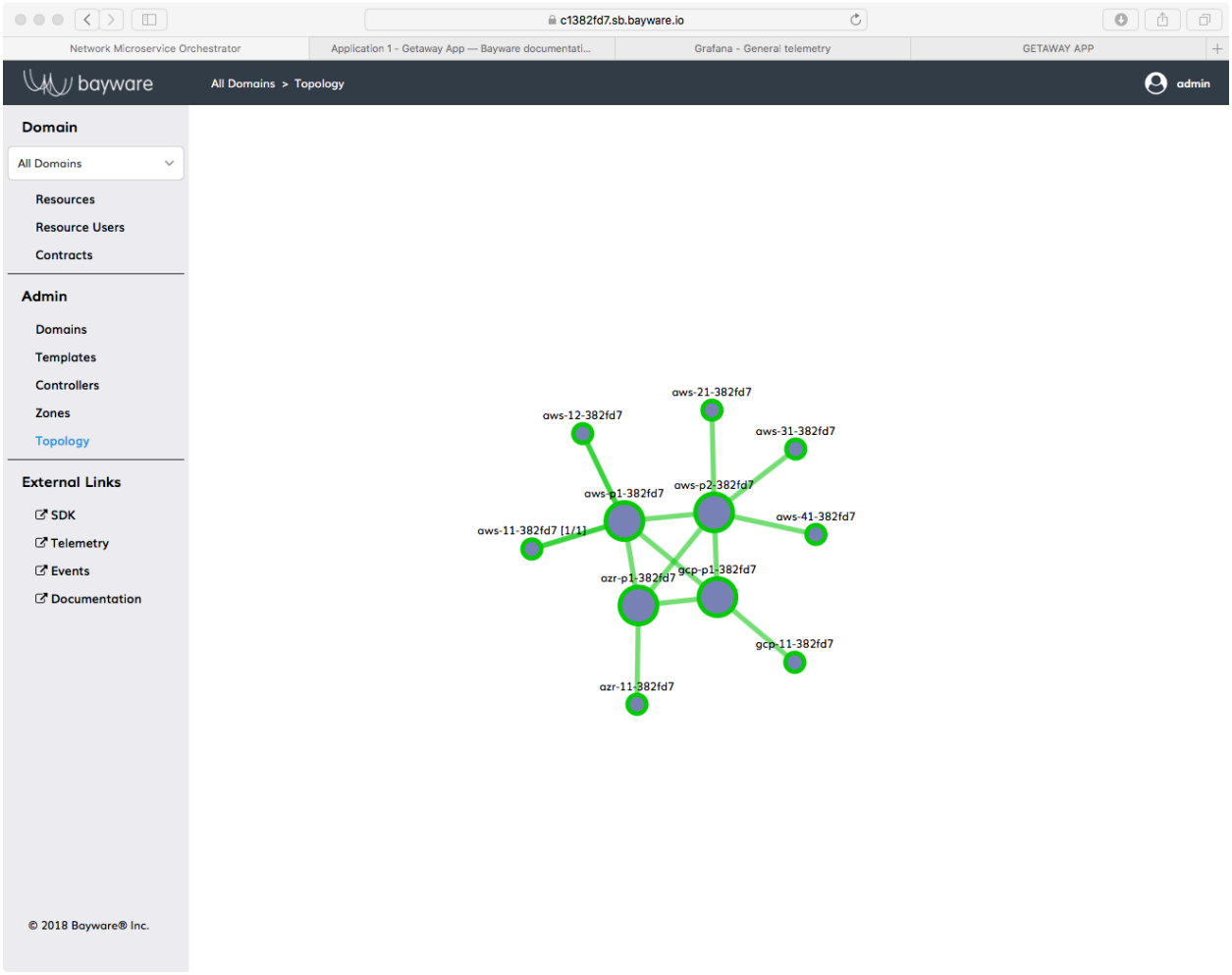


Fig. 12.25: Getaway App *Topology* with Seven Agents

```
]# exit
]$ exit
```

After you're back at your CCC, **ssh** into **aws-41**, become **root**, and stop the ib_agent running on that system.

Did it Work?

At the Orchestrator

Now go back to your browser tab that has the orchestrator open. Click on the *Topology* link in the left-side navigation menu. You should see your service interconnection fabric, but now two of the workload nodes are red, **aws-31** and **aws-41**, as shown in Fig. 12.26.

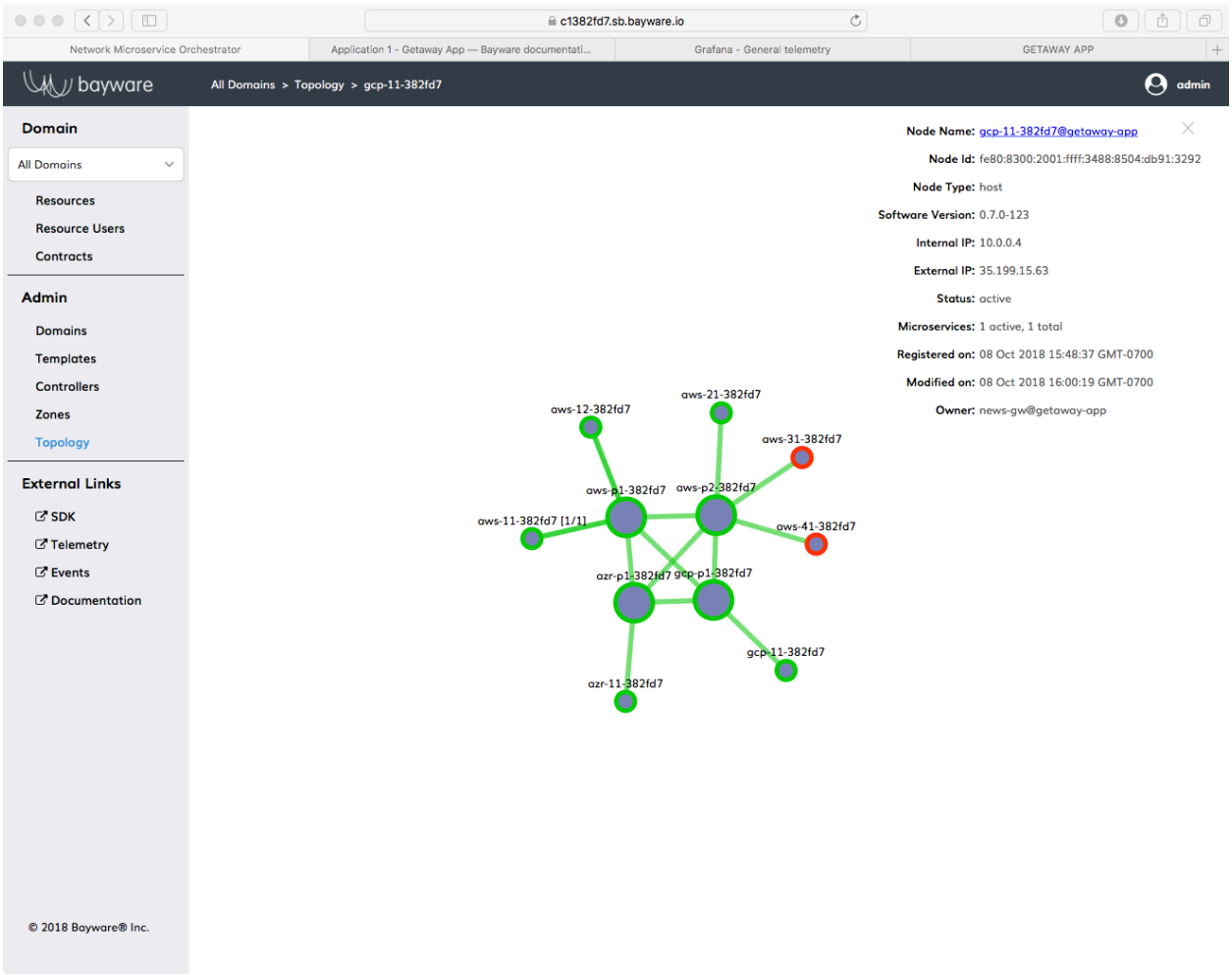


Fig. 12.26: Getaway App Topology with Two Red Workload Nodes

The orchestrator detected that the agents running on these two VMs were down and turned their circles red.

Now find the new workload node for **news-gsw**, **gcp-11**. Click on its circle in the *Topology* window to show the overlay information as shown in Fig. 12.26.

Note that the *owner*, shown at the bottom of the overlay information, is now **news-gw**.

Verify that `weather-gw` now owns `azr-11` by clicking on the appropriate circle in the *Topology* and noting its *owner*.

On the App

If you still have Getaway App open in your browser, go back to that tab. Recall that you can find the URL for Getaway App at the *top of your SIS*.

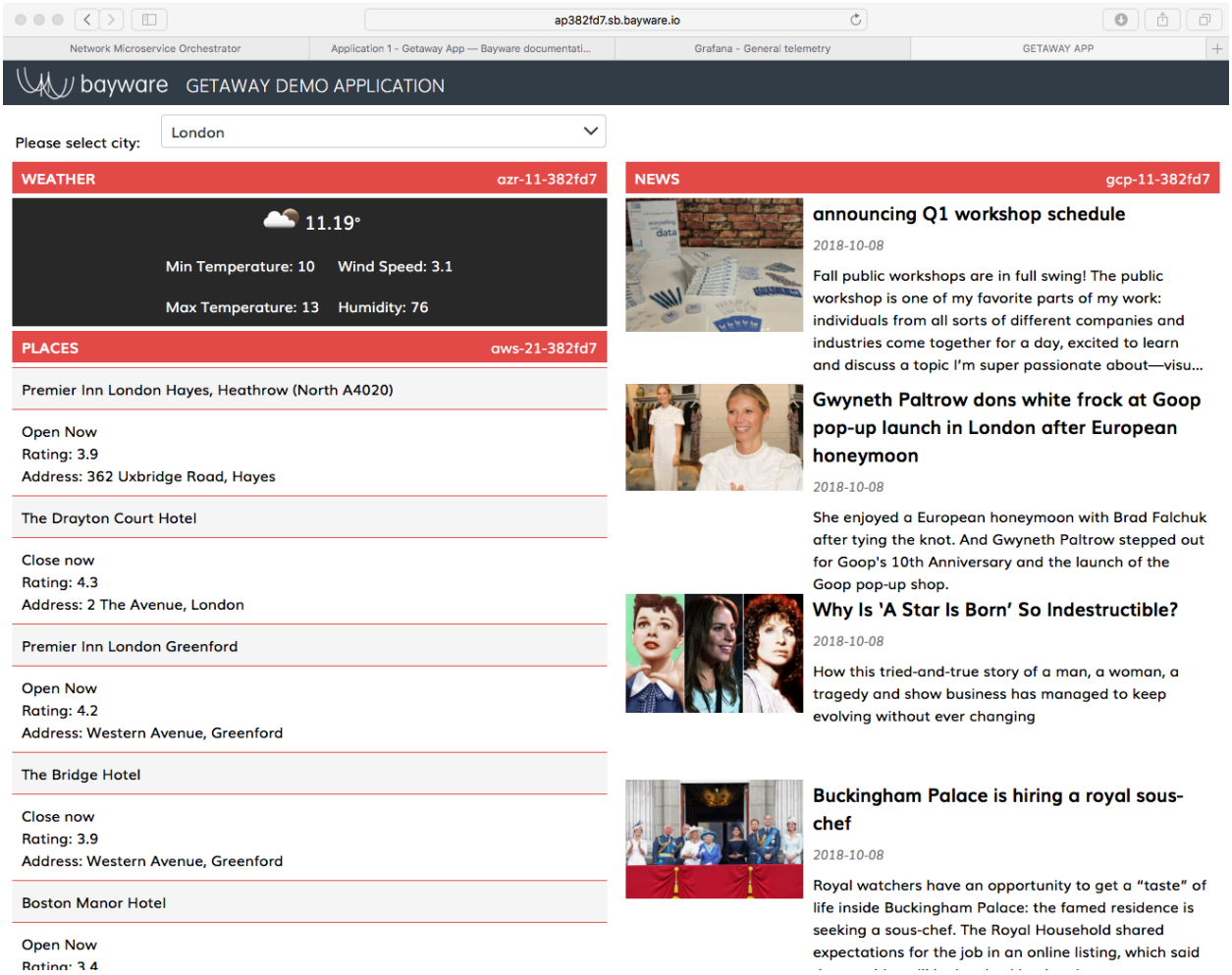


Fig. 12.27: Getaway App News & Weather On New VMs

The right end of each title bar shows the VM on which the particular microservice is running. If all went well in the preceding steps, you should see that *NEWS* running on `gcp-11` and *WEATHER* running on `azr-11` as shown in Fig. 12.27.

12.3.4 Good (Secure) Housekeeping

There's more work ahead and your CFO definitely isn't going to approve a whole new set of virtual machines. So once you (and *they*) are done with Getaway App, let's clean it up by removing agents and microservices. (Hold on to your service interconnection fabric processors, though. Those work quite nicely with any app.)

There's a script for that.

Back on your *Command Center*, be sure you are logged in as `centos` and not `root`. If you're at your `root` prompt, simply type

```
]# exit
```

Ensure you are in your `homedir` by typing `cd`. You should now be at a prompt that looks like

```
[centos@aws-bastion-382fd7 ~]$
```

Now type

```
]$ ./purge-apps.sh
```

But that's not quite the end. All the workload nodes are now cleaned up i.e., ib_agent services and microservices have been stopped and deleted. But the orchestrator doesn't forget so easily. If someone were to try to use one of your workload nodes for another microservice–even if he or she already had a new, valid set of orchestrator host owner credentials–the orchestrator would not recognize its credentials until the previous resource is deleted explicitly from the system.

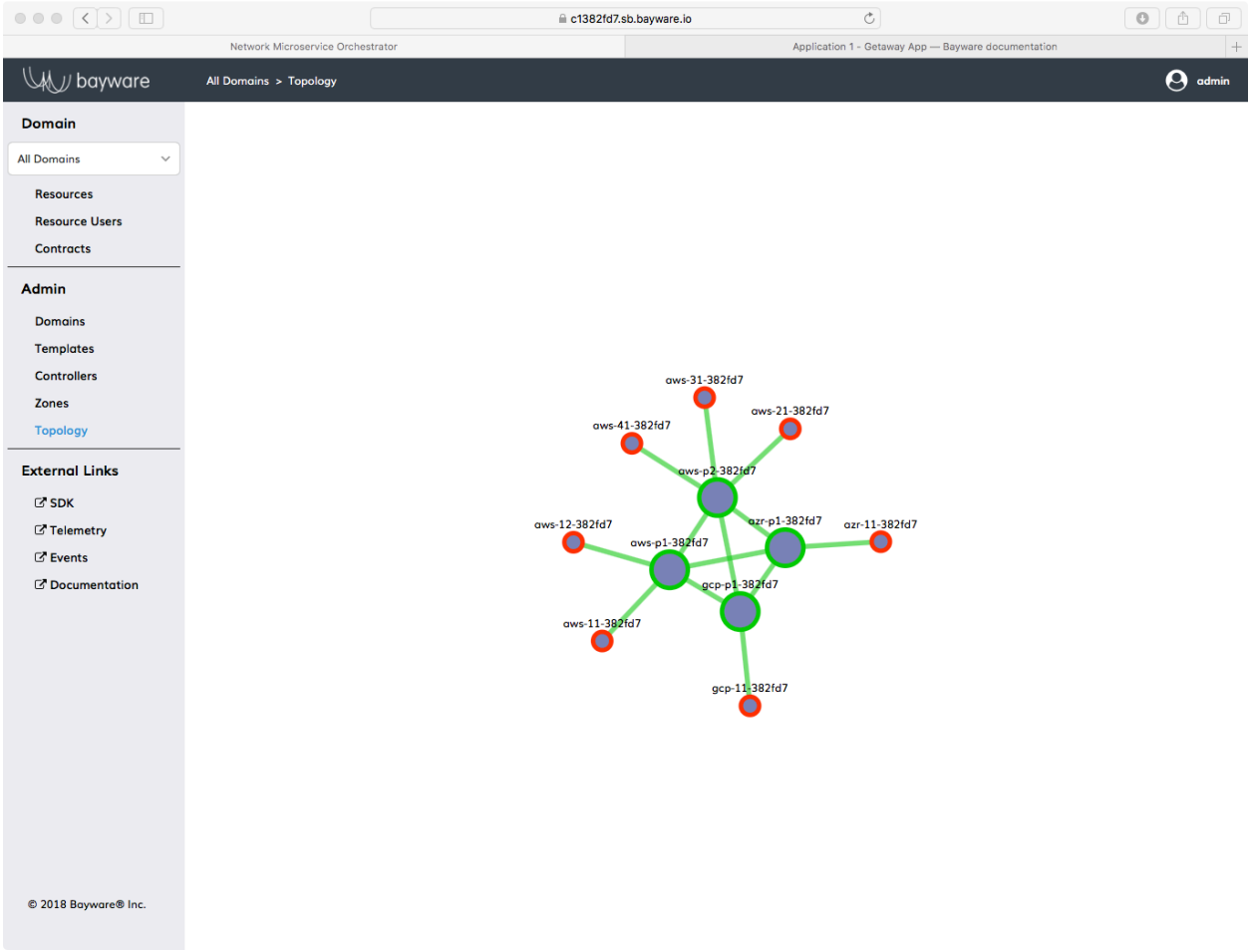


Fig. 12.28: Orchestrator Topology Showing Stopped Getaway App Workloads

Now go back to the orchestrator and click on *Topology*. You should see your service interconnection fabric with a bunch of red circles attached to it. Those are your stopped (deleted) agents from the Getaway App workloads as shown in Fig. 12.28.

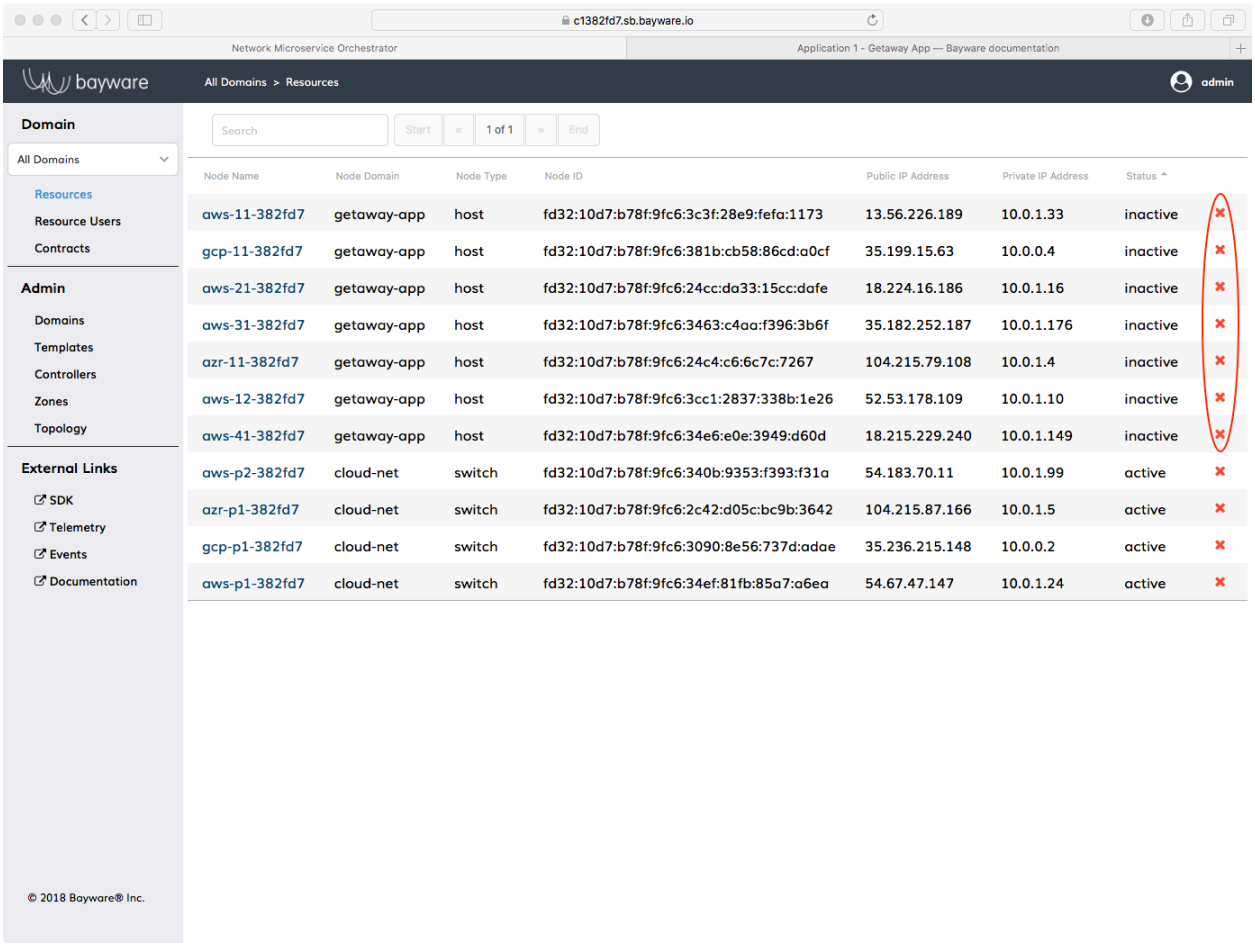


Fig. 12.29: Orchestrator Resources

You need to delete those red circles from the orchestrator. Do that by clicking on *Resources* in the left-side navigation menu. Click on the column header, *Node Domain*, to sort the rows and then click the red x at the right end of each row that is part of the `getaway-app` domain as shown in Fig. 12.29.

What you're *not* deleting...

You are not going to delete resources of type `switch`. These refer to the four processors you installed in *SIF Deployment* and can be used with any app, including the Voting App coming up next.

You also will not delete any of the pre-populated configuration such as those under *Resource Users* and *Contracts* on the orchestrator. Recall that resource users and contracts simply describe your application's service graph nodes and edges. If you decide to install your app again, that configuration will still be in place.

So as a final sanity check, navigate to the *Topology* page. It should look clean with only the original four processors in place.

12.3.5 Summary

In this section you learned how your application's service graph maps to components in Bayware's technology: service graph nodes map to host owners and service graph edges map to contracts. You went through the process of installing and configuring the microservices and agents required for Getaway App. With Getaway App up and running, you explored the Grafana telemetry UI. Finally, you created redundant nodes for *news* and *weather* and saw how Bayware technology automatically re-routed traffic when the original nodes became unavailable.

Next up: deploy Voting App using Ansible and insert a transit node into a contract...

12.4 Application 2 - Voting App

12.4.1 Containerized Microservices

Setting the Scene

So you're developing a reputation within you're company as the go-to DevOps engineer who can ensure quick and hiccup-free deployment of the company's most important applications (the devs love this) and, yet, can respond quickly to changing pricing structures among the major public cloud providers (the CFO loves this). That super easy Getaway App deployment put you on the map.

But now the powers-that-be are upping the ante. The devs have written a quickie application to see if end users prefer working with Mesos or Kubernetes in their containerized environments. They call it Voting App. And true to its raison d'être, it's containerized.

After the accolades of your last deployment, you decide Bayware is the way to go, containers and all. And the Voting App being a bit more involved, you decide to save yourself some typing by using Ansible for working with much of your hybrid cloud infrastructure.

A Word about Containers

Everybody's doing it. And, *they say*, if they're not doing it now, they will be doing it. Bayware knows this and that's why our technology is designed with container support from the bottom up.

Just as a microservice running directly on a virtual machine may be composed of one or more packages (RPMs, say, in an RHEL environment), a containerized microservice may be composed of one or more container images. As long as the containers require the same roles and policy from the network, a simple, single Bayware daemon (the agent) attaches the containers to the Bayware service interconnection fabric through the Bayware interface just as it does for RPMs on a virtual machine.

Service Graph

Your devs have given you the following service graph so that you can get started setting up your host owners and contracts in the Bayware orchestrator.

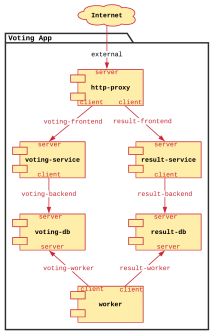


Fig. 12.30: Voting App Service Graph

First, here's how it works. There are two web pages that provide a GUI interface to the Voting App. The first web page allows end users to vote for either Mesos or Kubernetes as their preferred container management system. Once the application is installed, it will be available on a URL similar to `https://ap`*`382fd7`*`.bayware.io/voting/vote`. Recall that the URLs for your sandbox are listed at the *top* of your *SIS*.

The voting web page is shown in Fig. 12.31.

The second web page allows end users to see the aggregate result of all previous voting. Once the application is installed, it will be available on a URL similar to (`https://ap`*`382fd7`*`.bayware.io/voting/result`). Again, check your personal *SIS* for the precise URL.

The result web page is shown in Fig. 12.32.

Behind the scenes, the `http-proxy` microservice queries the `voting-svc` and `result-svc` microservices to serve up the two aforementioned web pages respectively.

On the back end, the `voting-svc` microservice writes cast ballots into the in-memory database microservice called `voting-db` while the `result-svc` reads tabulated results out of the postgres database microservice called `result-db`. The `worker` microservice, meanwhile, reads the cast ballots from `voting-db`, crunches the numbers, and writes the tabulated results to `result-db`.

Nodes & Host Owners

Recall from the *Nodes & Host Owners* discussion in Getaway App that service graph nodes map to host owners in the Bayware service interconnection fabric.

Go back to the orchestrator tab in your browser and click *Resource Users* in the left-side navigation bar. As shown in Fig. 12.33, you should find six *hostOwner* resource users pre-populated that are part of the `voting-app` domain. (Note that *Resource Users* from other `domains` are also visible.) They are

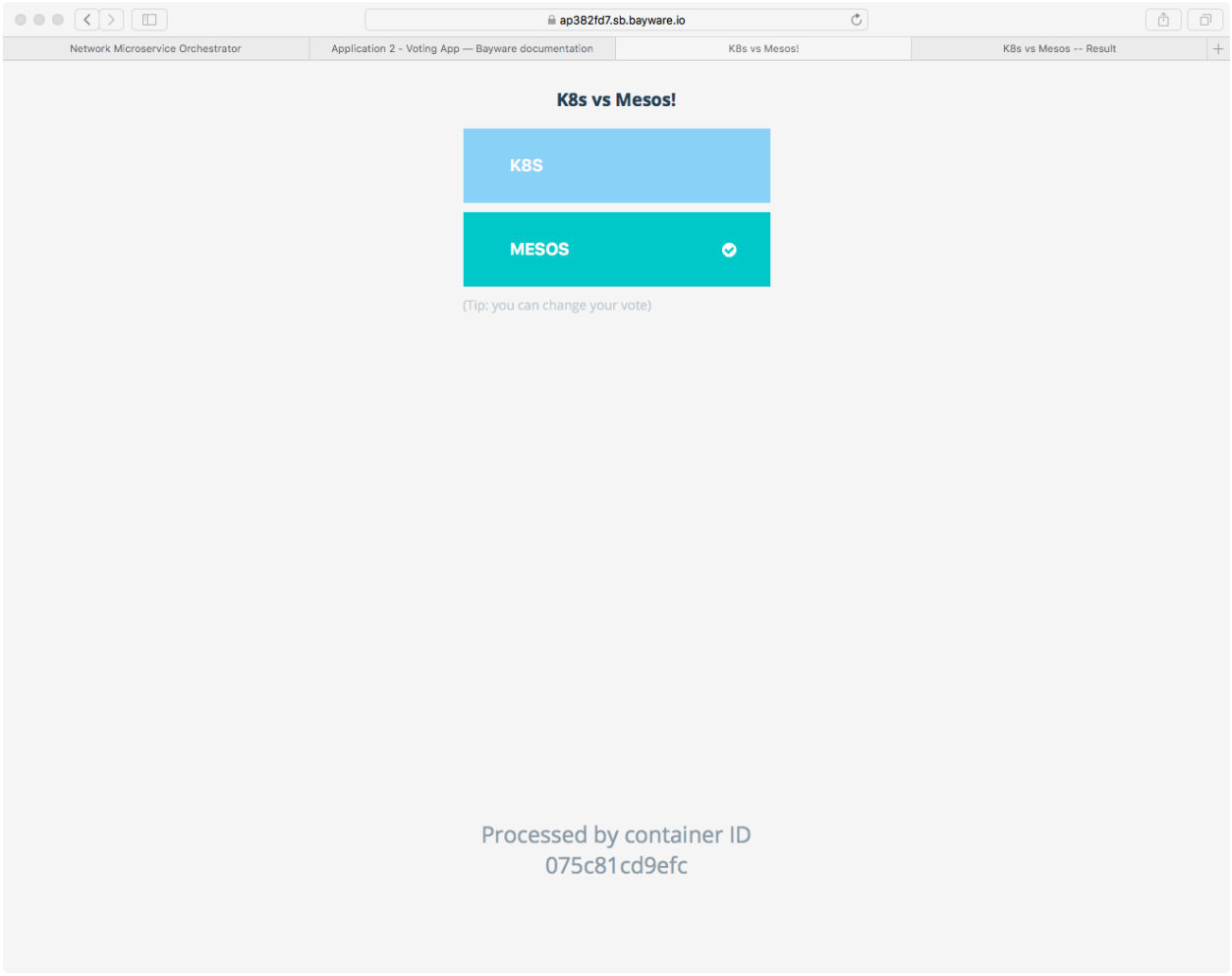


Fig. 12.31: Voting App Vote Interface in Browser

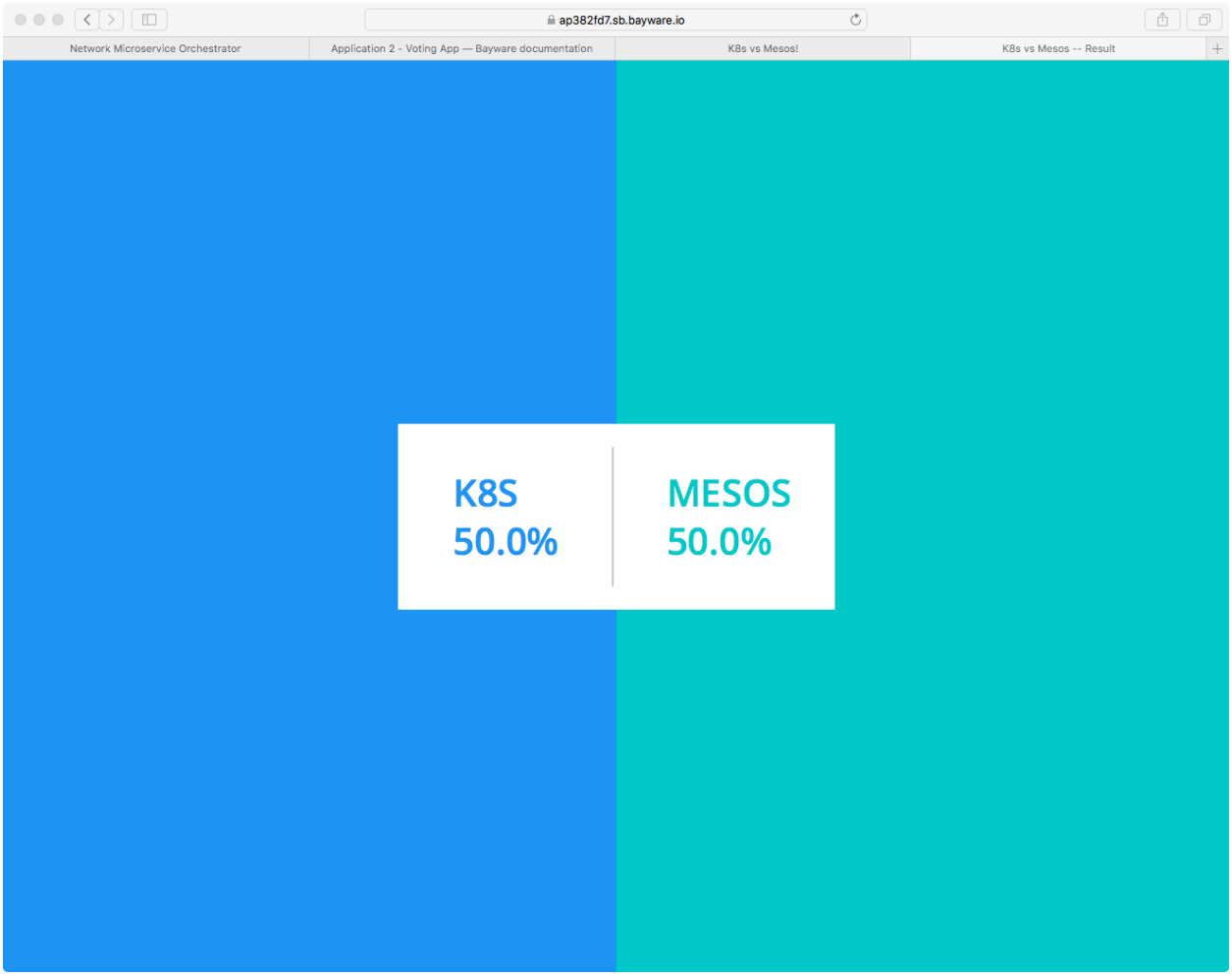


Fig. 12.32: Voting App Result Interface in Browser

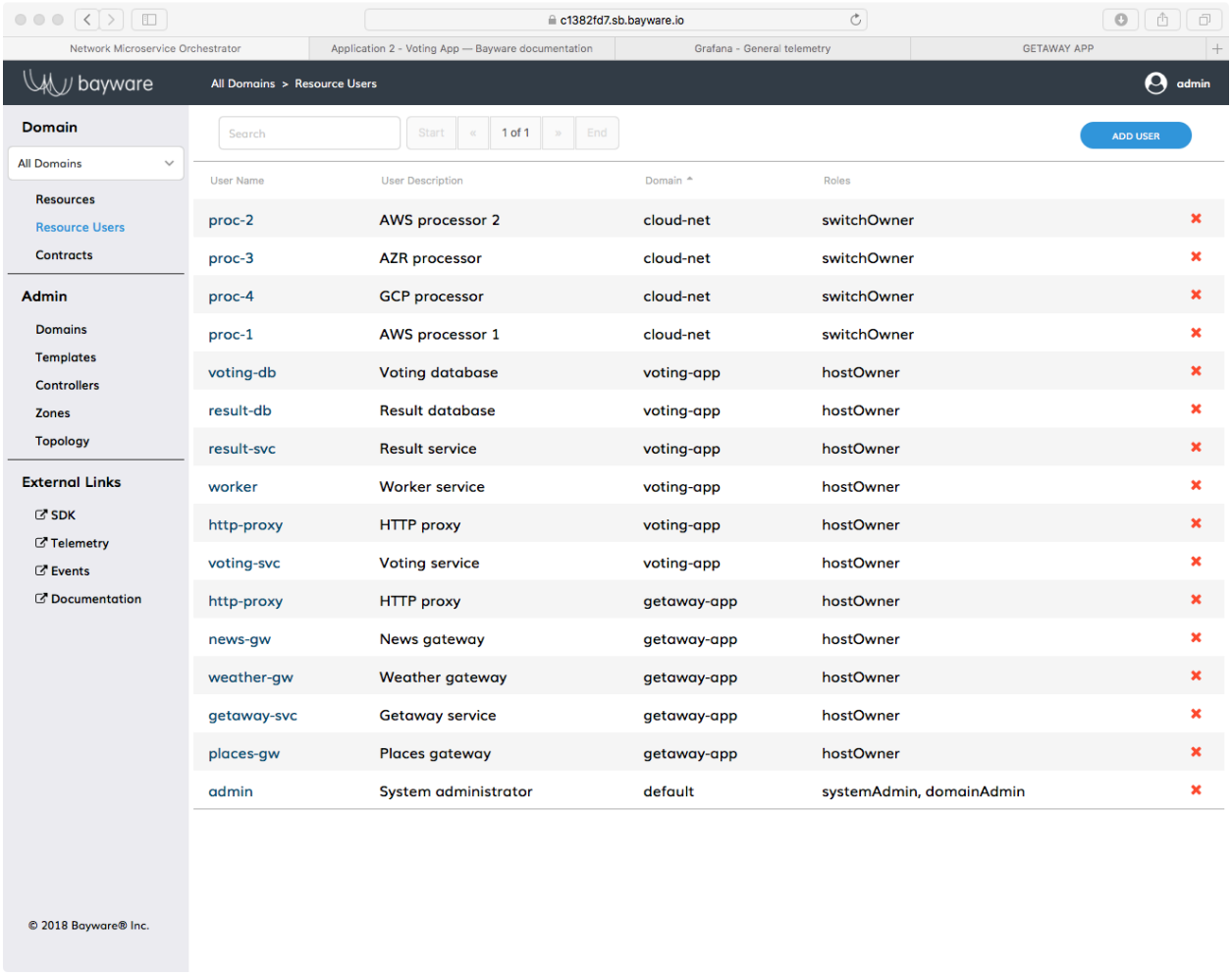


Fig. 12.33: Resource Users in the Orchestrator

- `http-proxy`
- `voting-svc`
- `voting-db`
- `result-svc`
- `result-db`
- `worker`

The six host owner names map one-to-one with the six microservices shown in Fig. 12.30. During the installation process described in the next section, consult *rows 28 through 33* in the table in your *SIS* for the passwords associated with the host owner usernames shown above.

Edges & Contracts

In the *Edges & Contracts* discussion in Getaway App, we defined the communicative relationships (edges) between microservices (nodes) as Contracts.

From Voting App service graph shown in Fig. 12.30, note that there are six communicative relationships between microservices (six edges).

You can check out the contracts relevant to Voting App back on the orchestrator (see Fig. 12.34). Return to the tab in your browser that has the orchestrator GUI open and click on *Contracts* in the left-side navigation bar. Look for the *Contract Names* that reside in the `voting-app` domain. They are

- `voting-frontend`
- `result-frontend`
- `voting-backend`
- `result-backend`
- `voting-worker`
- `result-worker`

As with the Getaway App, you can explore the relationships between contracts and microservices (host owners) by clicking on a contract name in your browser and then on a *Role Name* in the list at the bottom of the *Contract* page that comes up. Among other things, the *Role* page's *User Access List* shows connected microservices.

The contract roles in the Voting App are assigned as follows

Contract Name	Client Role	Server Role
voting-frontend	http-proxy	voting-service
result-frontend	http-proxy	result-service
voting-backend	voting-service	voting-db
result-backend	result-service	result-db
voting-worker	worker	voting-db
result-worker	worker	result-db

Authentication, Registration, & Microcode

One of the great things about Bayware technology is that you, the devOps engineer, do not have to fret the details. You simply set up your host owners and contracts in the orchestrator based off the service graph you get from your devs and then you're ready for app deployment.

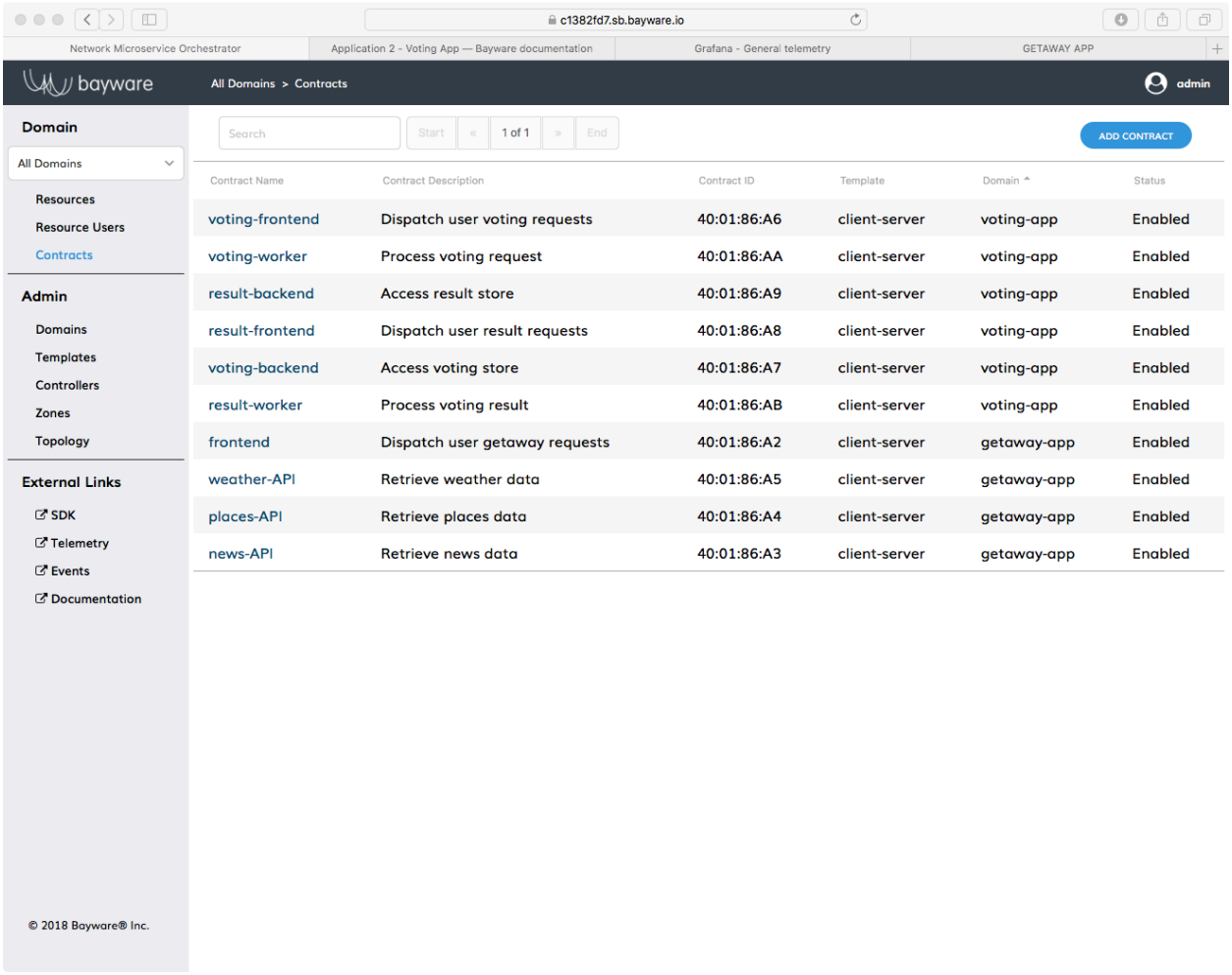


Fig. 12.34: Contracts in the Orchestrator

That is, all the behind-the-scenes *details* discussed during the Getaway App installation, like **authentication**, **registration**, and **microcode** applies equally here to the Voting App container environment. So feel free to go back and review and, when you're ready, onward to installation.

12.4.2 Installation with Ansible

The Script

Fig. 12.35 shows a familiar picture of your service interconnection fabric with attached workload nodes. Microservices for Voting App are highlighted in red.

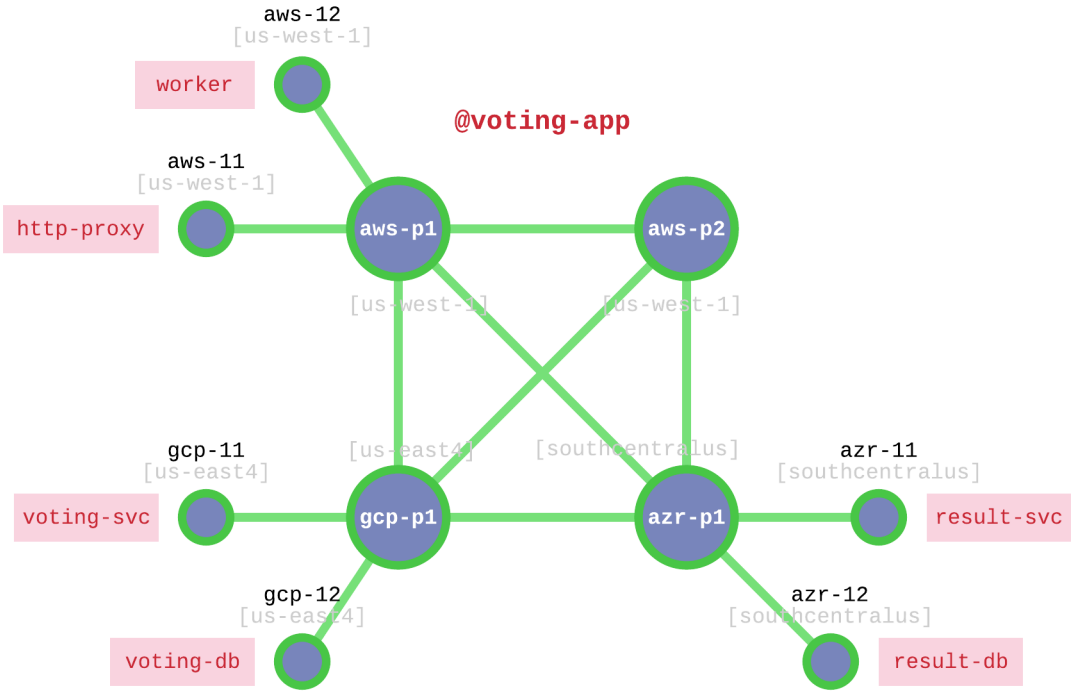


Fig. 12.35: Voting App Deployment on the Interconnection Fabric

Table 12.5 shows the VM on which each microservice will be deployed. Note that this table does not specify the service unit (microservice executable) as in Getaway App. The Ansible script knows those details so none of that typing is required.

Table 12.5: Voting Microservices VM Mapping

Microservice	VM
http-proxy	aws-11
worker	aws-12
voting-svc	gcp-11
voting-db	gcp-12
result-svc	azr-11
result-db	azr-12

Microservice deployment and agent configuration all happen using a script that you can find on your *Com-*

mand Center. So start by opening a terminal window on your local computer and use SSH to log in to your *Command Center*.

You won't need super-user privileges to run the Ansible script so you can do everything from

```
[centos@aws-bastion-382fd7 ~]$
```

Since you just logged in, you should be in the user **centos** home directory. Ensure the script is present by typing

```
[centos@aws-bastion-382fd7 ~]$ ls -alt deploy-voting-app.sh
```

This should give you output similar to

```
-rwxr-xr-x. 1 centos centos 2343 Oct  4 20:46 deploy-voting-app.sh
```

Of course, the time stamp may be different. If you're in the wrong spot or the script has disappeared from your system, the **ls** command above will respond with something that says **No such file or directory**. If that happens, ensure you're in your home directory by simply typing **cd** at the prompt and then try the **ls** again.

Now that you're on your control center and you've confirmed the presence of the script, you only need to repeat two steps for each Voting App microservice to get full application deployment.

1. Run script specifying host owner and VM for one microservice
2. Repeat 1 and 2 for remaining microservices

Step 1: Run Script

Let's start by deploying **http-proxy**. Table 12.5 indicates you should put this on **aws-11**. Enter the following at your prompt

```
]$ ./deploy-voting-app.sh --hostowner=http-proxy --virtualmachine=aws-11
```

After you press **return**, the script will determine the correct domain and password for **http-proxy**, install the correct container image on **aws-11**, and register the agent with orchestrator.

Step 2: Repeat

Well, that definitely saved some typing. So give it a shot with the remaining five microservices and the corresponding virtual machines listed in Table 12.5. After you execute **deploy-voting-app.sh** five more times, everything should be deployed.

```
CHEAT SHEET - VOTING APP INSTALL

hint:  use elements in the list [] below for each iteration

]$ ./deploy-voting-app.sh --hostowner=worker --virtualmachine=aws-12
    hint: [ ( worker, aws-12 ), ( voting-svc, gcp-11 ), ( voting-db, gcp-12 ), ( result-
↪svc, azr-11 ), ( result-db, azr-12 ) ]
```

Back at the Orchestrator

Now go back to your browser window that has the orchestrator open. Click on the *Topology* button on the left-side navigation menu. You should see the four large, green circles that represent Bayware processors as well as smaller circles labeled with the host names of the workload nodes.

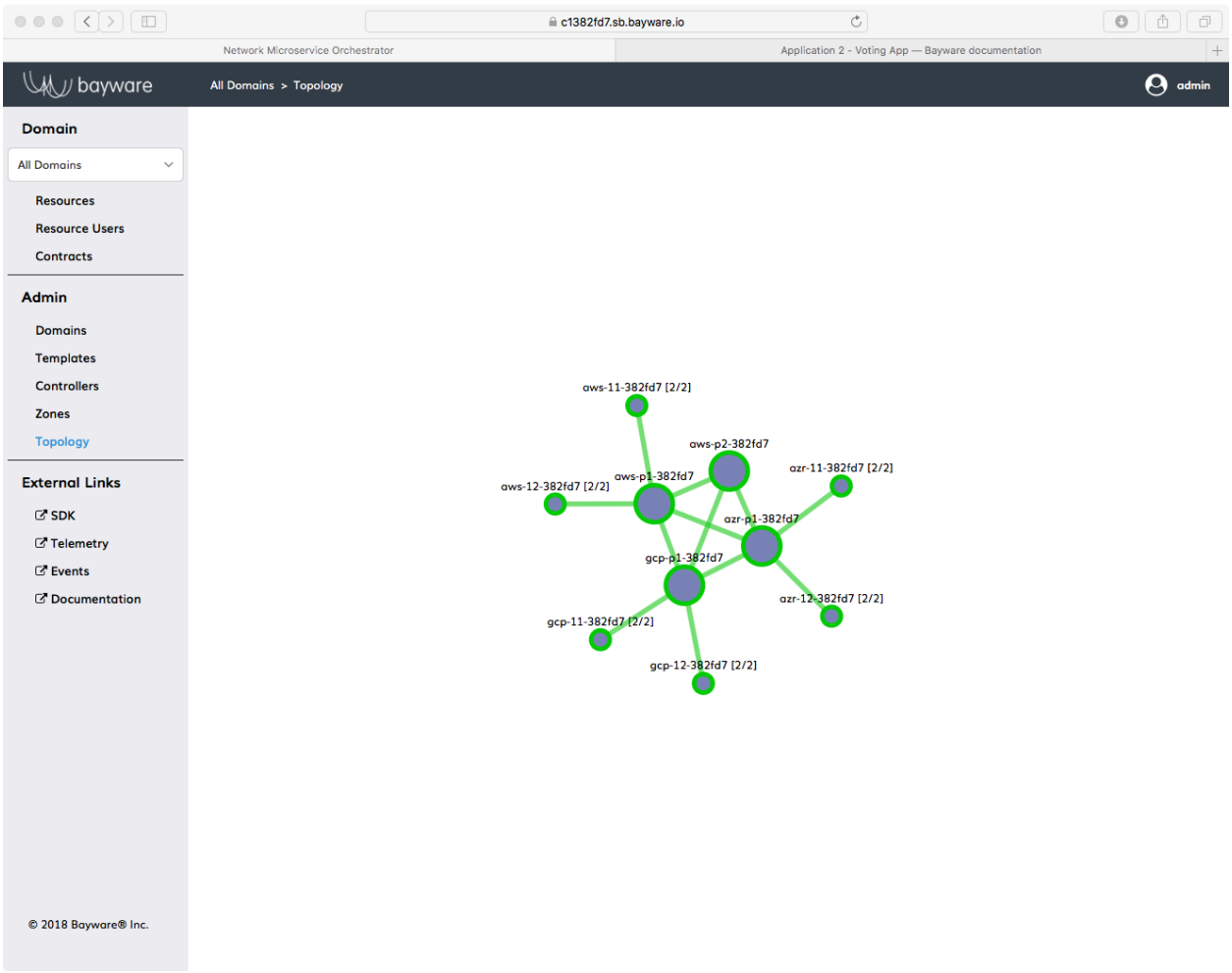


Fig. 12.36: Orchestrator Topology After Voting App Deployment

Find the host names used during Voting App deployment. If you click on a host name, you'll see overlay information about the node. The last item in the overlay information is *Owner*. This should correspond to the host owner of each microservice–which, in this setup is the same name as the microservice listed in Table 12.5–and the `voting-app` domain.

Service Graph Revisited

Just as you saw with Getaway App, the orchestrator generates a service graph based off the host owners and contracts that have been added to the system. That way you can sanity check that the information you entered matches the service graph given to you by your devs.

Back on the orchestrator, click on *Domains* in the left-side navigation menu. Of the four domain names that are displayed, now click on `voting-app`. At the bottom of the `voting-app` domain window, click on *Domain Topology*. You should see Voting App service graph recreated by the orchestrator as shown in Fig. 12.37.

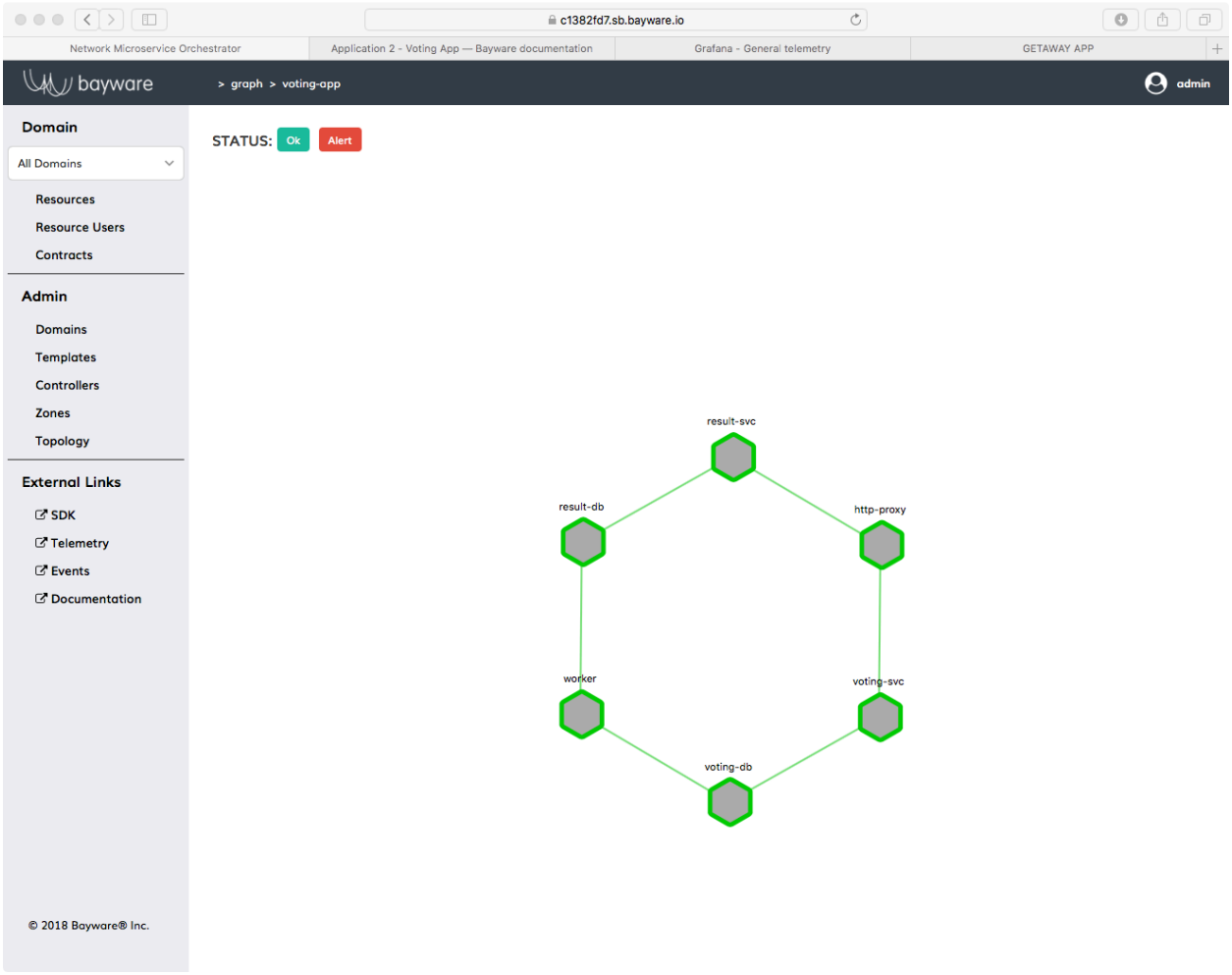


Fig. 12.37: Voting App Service Graph Generated By Orchestrator

Vote

Before you go back to your devs and report on the successful deployment of Voting App, you should… VOTE!

You can find the URLs for the voting application at the *top of your SIS*. Open a new browser tab and type in the URL that ends in `/voting/vote`. You should see a window similar to Fig. 12.38.

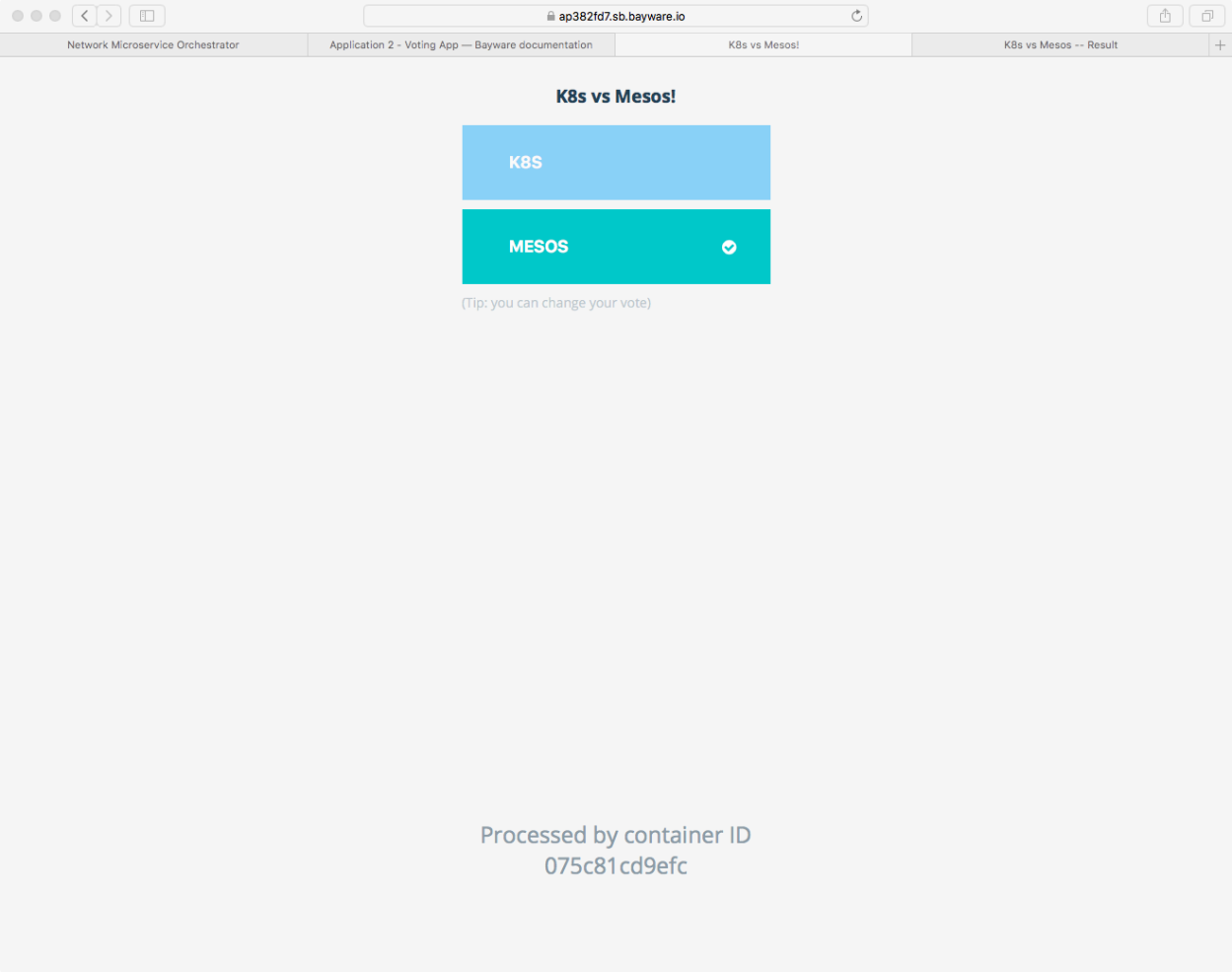


Fig. 12.38: Voting App - The Vote Web Page

Choose your favorite container management system and then open another browser window and type in the URL from your *SIS* that ends in `/voting/result`. You should see the results of the voting similar to Fig. 12.39.

If you don't like the result, just go back to the vote page and cast your ballot again. The new tally is reflected on the result page.

Congratulations! You voted and–if you voted often enough–your favorite management system won.

That's it!

Go back to your devs, tell them that Voting App is deployed in a hybrid cloud environment, and buttoned up with Bayware's innovative security and policy enforcement tools. Just don't tell them how easy it was. Anyway, rumor has it that they're cooking up some new intrusion detection system to ensure *one-person one-vote* and they're going to ask you to deploy it.

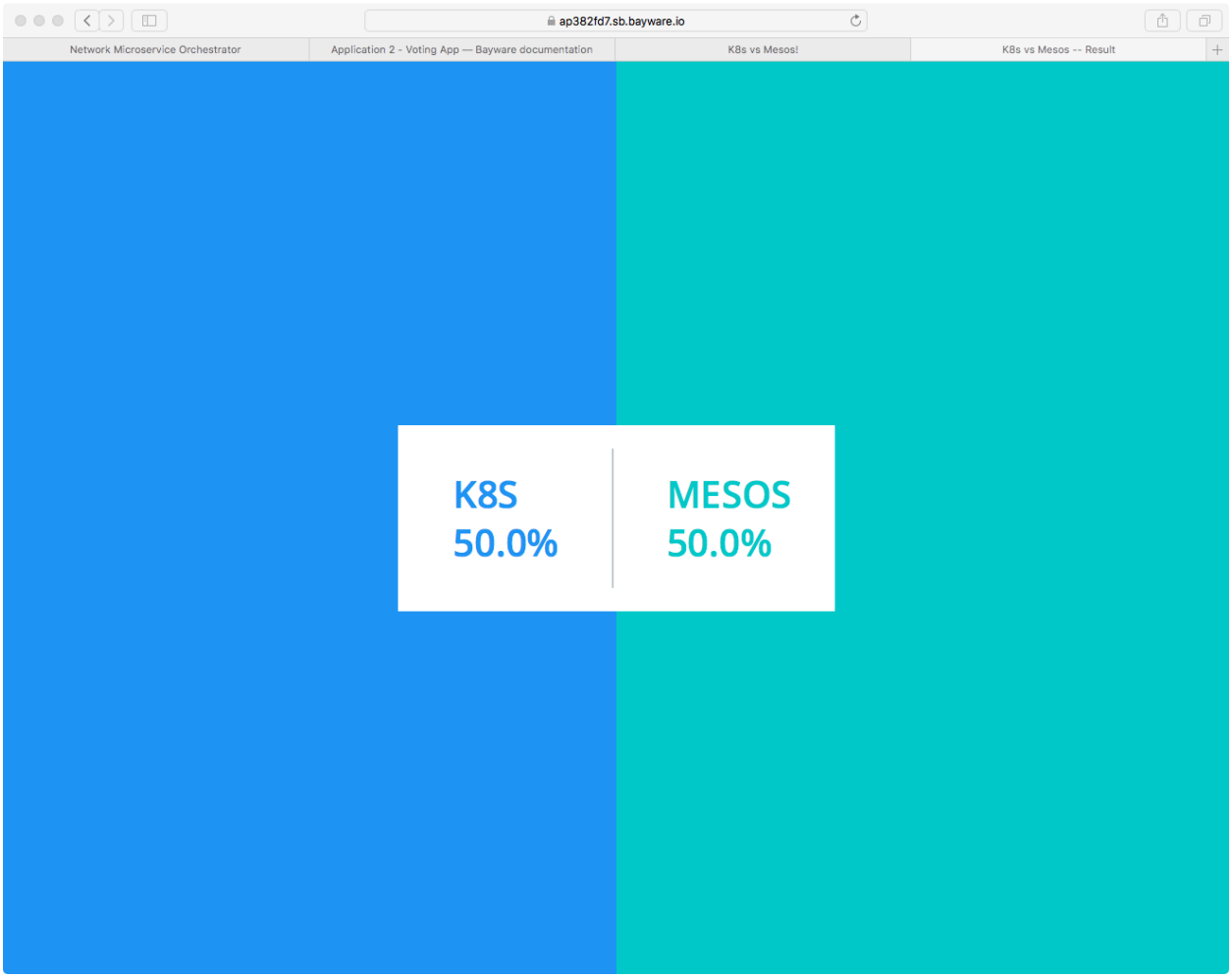


Fig. 12.39: Voting App - The Result Web Page

Carry on.

12.4.3 Use Case - Transit Nodes

Once the higher-ups got the secOps team involved, everything went top secret. But word came down that they had *someone* install *something* on `aws-p2` and they needed you to ensure that all traffic moving in either direction between `http-proxy` on `aws-11` and `voting-svc` on `gcp-11` now go through `aws-p2` as a transit point.

Bayware's got you covered.

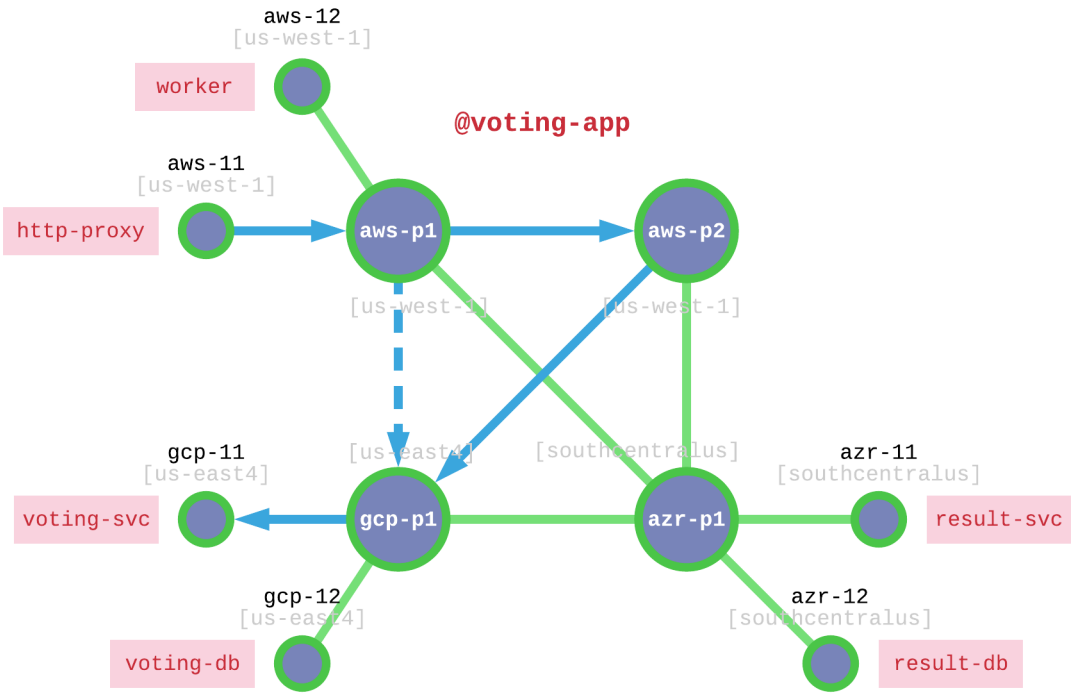


Fig. 12.40: Voting App Transit Node

Fig. 12.40 shows a dotted line between `aws-p1` and `gcp-p1` that indicates the shortest path to move traffic from `http-proxy` to `voting-svc`. Without any other restrictions, Bayware's technology finds shortest paths between agents using Dijkstra's algorithm. Sending traffic through `aws-p2` as a transit node between these two point requires a slight modification to the communicative relationship between `http-proxy` and `voting-svc`. Recall that in Bayware's technology, communicative relationships between nodes are represented by the edges in a service graph. Edges are described with Bayware contracts. If you refer back to *Voting App Service Graph*, you will note that the contract between `http-proxy` and `voting-svc` is called `voting-frontend`. You will need to add a transit point into this contract.

But first, we'll make sure that `aws-p2` isn't doing much right now. Without jumping too far into the weeds, let's take a look at a couple stats in Open vSwitch (OVS) on `aws-p2`. Recall that a Bayware processor is composed of two pieces: the Bayware engine (control plane) and OVS (data plane). When the engine processes control packets, it updates rules inside OVS. OVS stores rules in tables.

Two tables are relevant to this exercise

- Table 30: aggregates all non-control packets

- Table 50: filters all packets at the flow level

As shown in Fig. 12.40, your current Voting App installation does not utilize `aws-p2` at all.

Go back to your terminal window and be sure you're logged into *Command Center*. Now log into `aws-p2` and become `root`:

```
[centos@aws-bastion-382fd7 ~]$ ssh centos@aws-p2
[centos@aws-p2-382fd7 ~]$ sudo su -
[root@aws-p2-382fd7 ~]#
```

Read OVS table 50

```
]# ovs-ofctl dump-flows ib-br table=50
```

You should see an empty result that looks like

```
NXST_FLOW reply (xid=0x4):
```

Table 50 is updated dynamically by the engine. When a new flow is introduced at this processor node, new rules are created in Table 50.

Now read OVS table 30

```
]# ovs-ofctl dump-flows ib-br table=30
```

You should see two entries similar to

```
NXST_FLOW reply (xid=0x4):
cookie=0x3066303030303537, duration=87124.231s, table=30, n_packets=0, n_bytes=0, idle_
↪age=65534, hard_age=65534, priority=1500,ipv6,ipv6_dst=fd32:10d7:b78f:9fc6::/64␣
↪actions=resubmit(,40)
cookie=0x3066303030303537, duration=87124.231s, table=30, n_packets=57, n_bytes=11202,␣
↪idle_age=3660, hard_age=65534, priority=1200,ipv6,ipv6_dst=ff3c::/32 actions=resubmit(,
↪50)
```

Table 30 entries are created during node registration and act as a catch-all for all data packets. The second table row is particularly important as it counts all IPv6 SSM packets, which just happens to be how Bayware packets move around a network. In the example above, remember that `n_packets=57`. Your *number of packets* will differ, but just remember whatever baseline is.

To convince yourself that nothing is happening on `aws-p2`, read Table 30 multiple times. The value in the `n_packets` field should not change.

Now let's bring `aws-p2` into the equation. Modification of the `voting-frontend` contract is all done at the orchestrator.

Start by clicking the *Contracts* button on the orchestrator and then click on the `voting-frontend` contract as shown in Fig. 12.41.

First you'll edit the contract's client role–but pay attention because you'll need to modify the server role later. Click on *client* under *Role Name* near the bottom of the window.

Now click on *EDIT* in the *Path Params* row. You'll be inserting the transit node into the path parameters.

Your browser window should now be split between an editor on the left and some examples and explanation on the right. Modify the JSON description of the path parameters as shown in Fig. 12.44. Careful to use your personal sandbox name when describing `aws-p2-`*`382fd7`*. The contents of the JSON description in its entirety should read

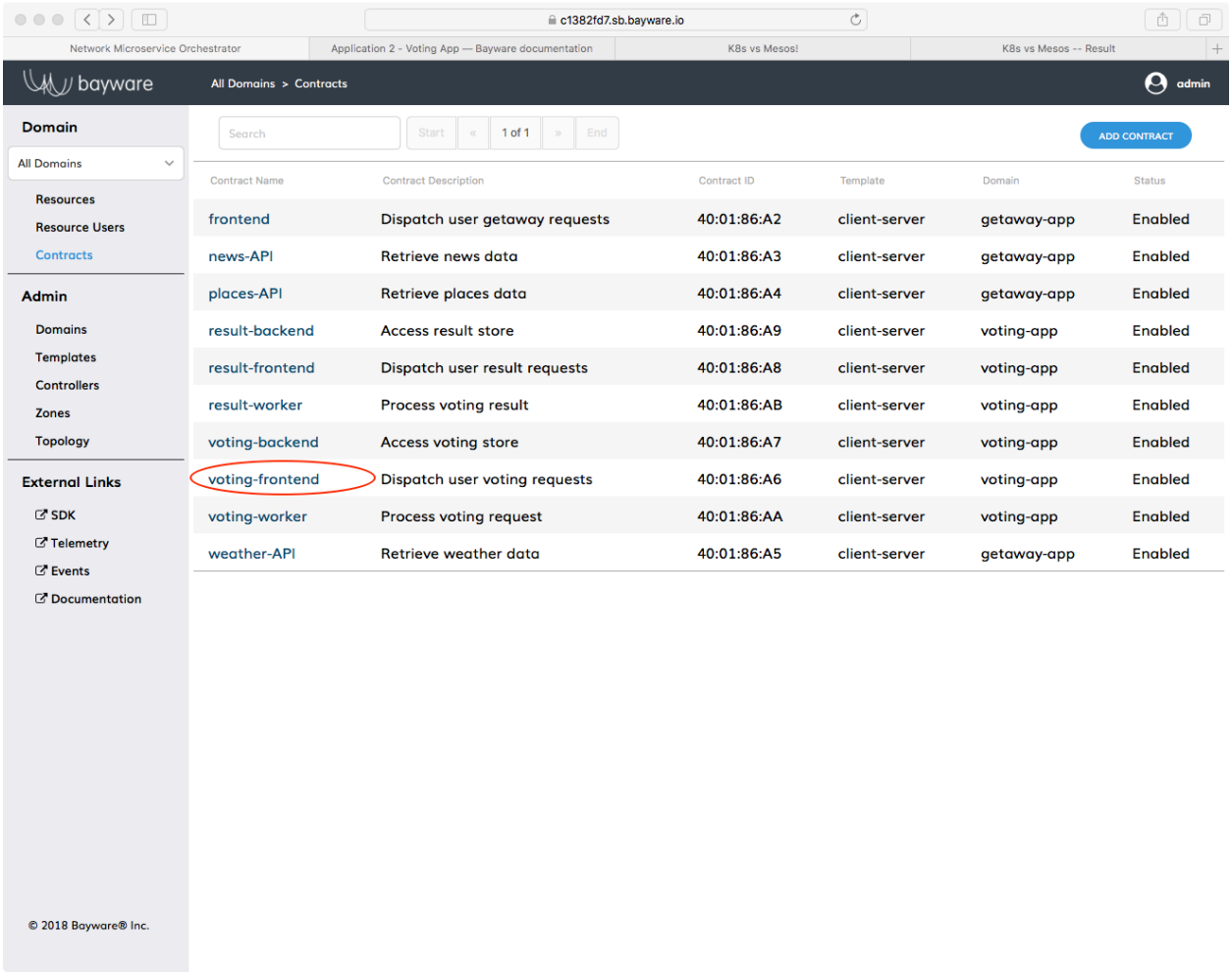


Fig. 12.41: Click on voting-fronted Contract

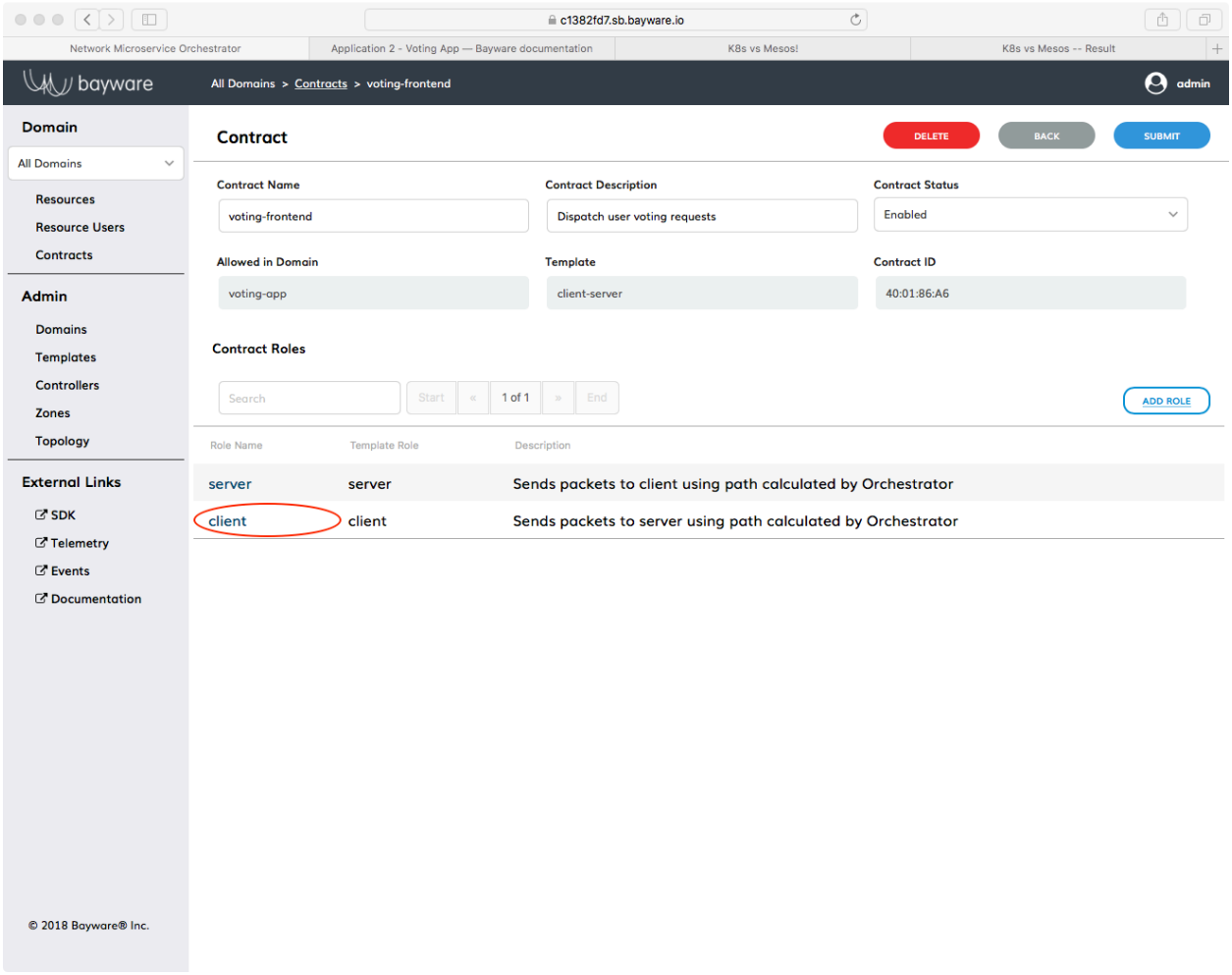


Fig. 12.42: Click on Client Role

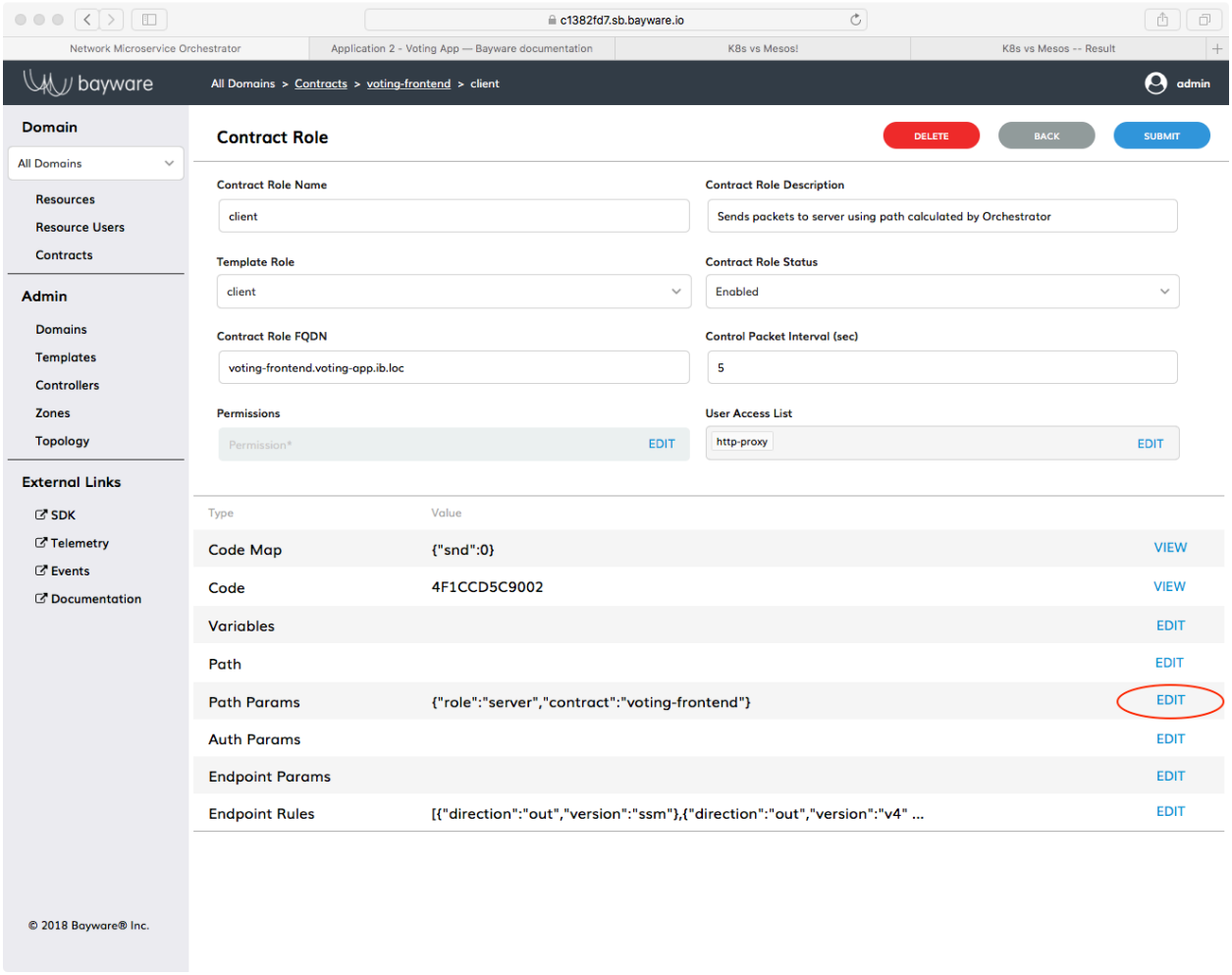


Fig. 12.43: Click on Edit Next to Path Params

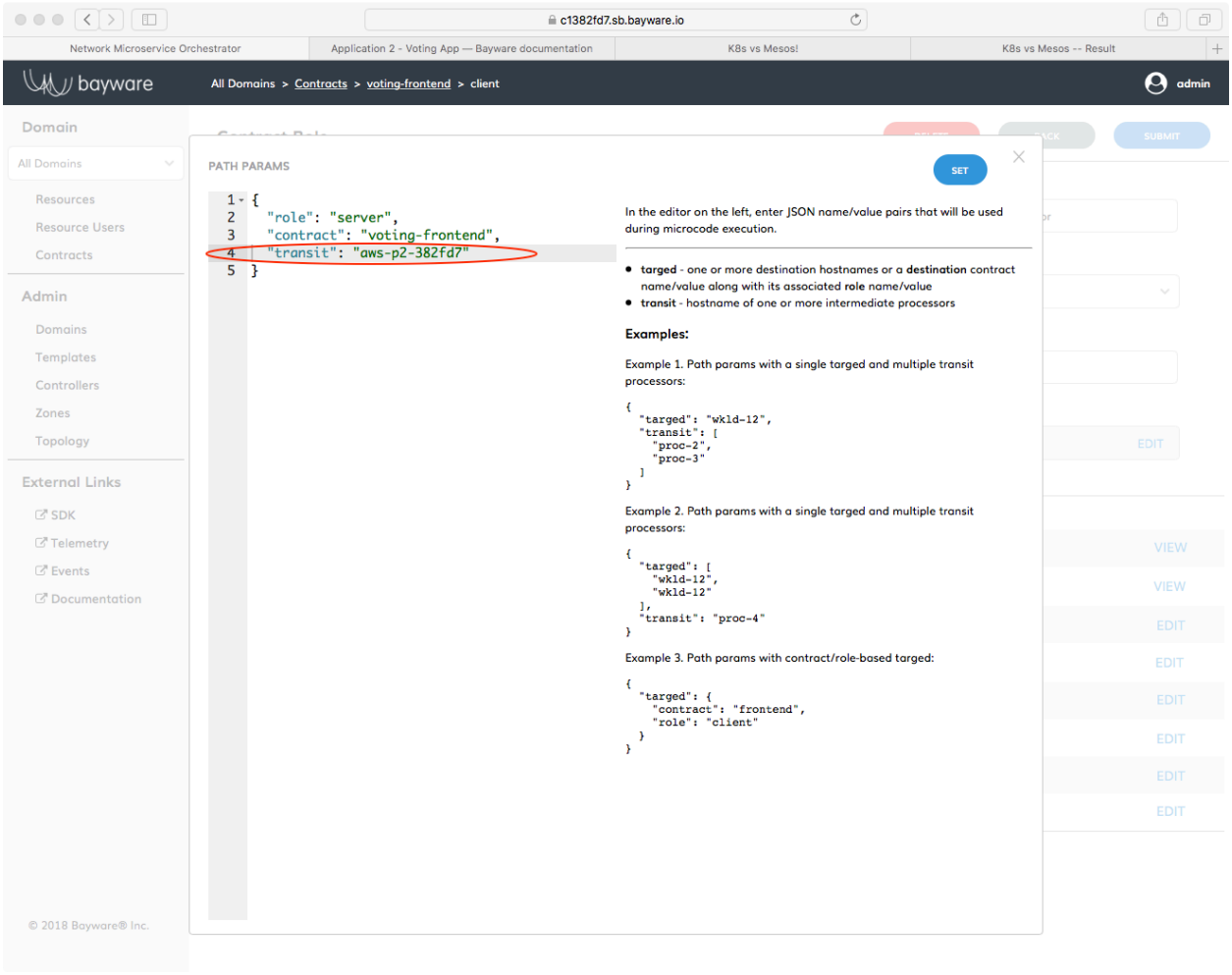


Fig. 12.44: Edit Path Params JSON Description

```
{
    "role": "server",
    "contract": "voting-frontend",
    "transit": "aws-p2-382fd7"
}
```

Don't miss the comma at the end of the *contract* line, either. Click *SET* in the upper-right corner. Then be sure to click *SUBMIT* after you are returned back to the *Contract Role* page.

Important: You need to click both *SET* and *SUBMIT* to get the new Path Params to stick.

Since the client role in `frontend-contract` is assigned to `http-proxy`, traffic from `http-proxy` to the server will now go through `aws-p2`.

Now repeat the changes you made to the path parameters on the client role to those on the server role. That will force traffic coming from `voting-svc` to go to `aws-p2` before going to `http-proxy`.

Once you have edited path parameters on both client and server roles of `frontend-contract`, you can check if everything is working. Go back to the browser tab that has the Voting App *vote* window open. The URL is similar to `https://ap`*`382fd7`*`.sb.bayware.io/voting/vote`. Now vote.

Check out the OVS tables again to ensure traffic is routed through the transit point, `aws-p2`.

Go back to your terminal window where you're logged in as `root` on `aws-p2`. Read Table 30.

```
[root@aws-p2-382fd7 ~]# ovs-ofctl dump-flows ib-br table=30
NXST_FLOW reply (xid=0x4):
cookie=0x3066303030303537, duration=88250.285s, table=30, n_packets=0, n_bytes=0, idle_
↪age=65534, hard_age=65534, priority=1500,ipv6,ipv6_dst=fd32:10d7:b78f:9fc6::/64␣
↪actions=resubmit(,40)
cookie=0x3066303030303537, duration=88250.285s, table=30, n_packets=72, n_bytes=15301,␣
↪idle_age=3, hard_age=65534, priority=1200,ipv6,ipv6_dst=ff3c::/32 actions=resubmit(,50)
```

See in this example that `n_packets=72`. So 72-57=15 packets have been sent through `aws-p2` after modifying both client and server roles and voting one time.

Similarly, take a look at Table 50.

```
[root@aws-p2-382fd7 ~]# ovs-ofctl dump-flows ib-br table=50
NXST_FLOW reply (xid=0x4):
cookie=0x3066303130343564, duration=2.857s, table=50, n_packets=0, n_bytes=0, idle_
↪timeout=60, hard_timeout=60, idle_age=33, priority=1000,ipv6,in_port=3,ipv6_
↪src=fd32:10d7:b78f:9fc6:240d:8cb4:1db0:4082,ipv6_dst=ff3c::4001:86a6,ipv6_
↪label=0x63b71 actions=output:2
cookie=0x3066303130343631, duration=2.641s, table=50, n_packets=0, n_bytes=0, idle_
↪timeout=60, hard_timeout=60, idle_age=18, priority=1000,ipv6,in_port=2,ipv6_
↪src=fd32:10d7:b78f:9fc6:28a3:eea2:e22e:2103,ipv6_dst=ff3c::4001:86a6,ipv6_
↪label=0x0c894 actions=output:3
```

Whereas the table was previously empty, now it contains two entries. Note that `n_packets=0` simply because the entry refresh rate makes it difficult to catch a non-zero value. However, you can verify the relationship between the table entires and your service interconnection fabric. Find the `ipv6_src` address in each entry above and compare it with the *Node ID* column on the *Resources* page in the orchestrator. One entry should map to `aws-11` and the other to `gcp-11`: the workload nodes running the microservices between which the

transit node `aws-p2` was injected. Further, the lower four bytes of the `ipv6_dst` address in each entry above indicate the *Contract* used. Find this in the *Contract ID* column on the *Contracts* page in the orchestrator.

12.4.4 Good (Secure) Housekeeping

If you've been following along, you know things change quickly around here: Getaway App was here and now it's gone. Likewise, when the votes have been counted, you'll want to scrub your infrastructure so you have nothing left but a pristine Bayware service interconnection fabric. That way, if you need a vacation after all the voting brouhaha, re-install Getaway App and start planning a week in Columbus or Frankfurt!

Back on your *Command Center*, be sure you are logged in as `centos` and not `root`. If you're at your `root` prompt, simply type

```
]# exit
```

Ensure you are in your `homedir` by typing `cd`. You should now be at a prompt that looks like

```
[centos@aws-bastion-382fd7 ~]$
```

Now type

```
]$ ./purge-apps.sh
```

As with the purge of Getaway App, all workload nodes in your infrastructure are now disconnected from the orchestrator. They should appear red in the *Topology* window.

For security reasons described in Getaway App, those red workload nodes now need to be deleted from the orchestrator by clicking *Resources* and then clicking the red *x* next to the hosts that are part of the `voting-app` domain. Once you have deleted those, you should only see your four processor nodes back on the *Topology* page.

12.4.5 Summary

In this chapter you learned that Bayware's innovative technology works natively with containers. You deployed Voting App, a containerized application using Ansible. After interacting with the Voting App in your browser, you easily modified `voting-frontend` contract's path parameters to insert a transit node between `http-proxy` and `voting-svc`.

12.5 SIS - Example

The following *Sandbox Installation Summary* (SIS) serves as an example so that tutorial readers may follow along with the text before receiving their personal SIS. Only the SB number itself, *`382fd7`*, will be different in a generated SIS.

Note: since this is a replica of the SIS you will receive from Bayware, some of the tables below may exceed the width of your visible browser area. You can simply scroll horizontally to see the obstrutcted text. Other literal blocks in this tutorial–showing code or a command-line interface–function similarly.

```
BAYWARE SANDBOX INSTALLATION SUMMARY:  382fd7

Welcome to Bayware!

```

(continues on next page)

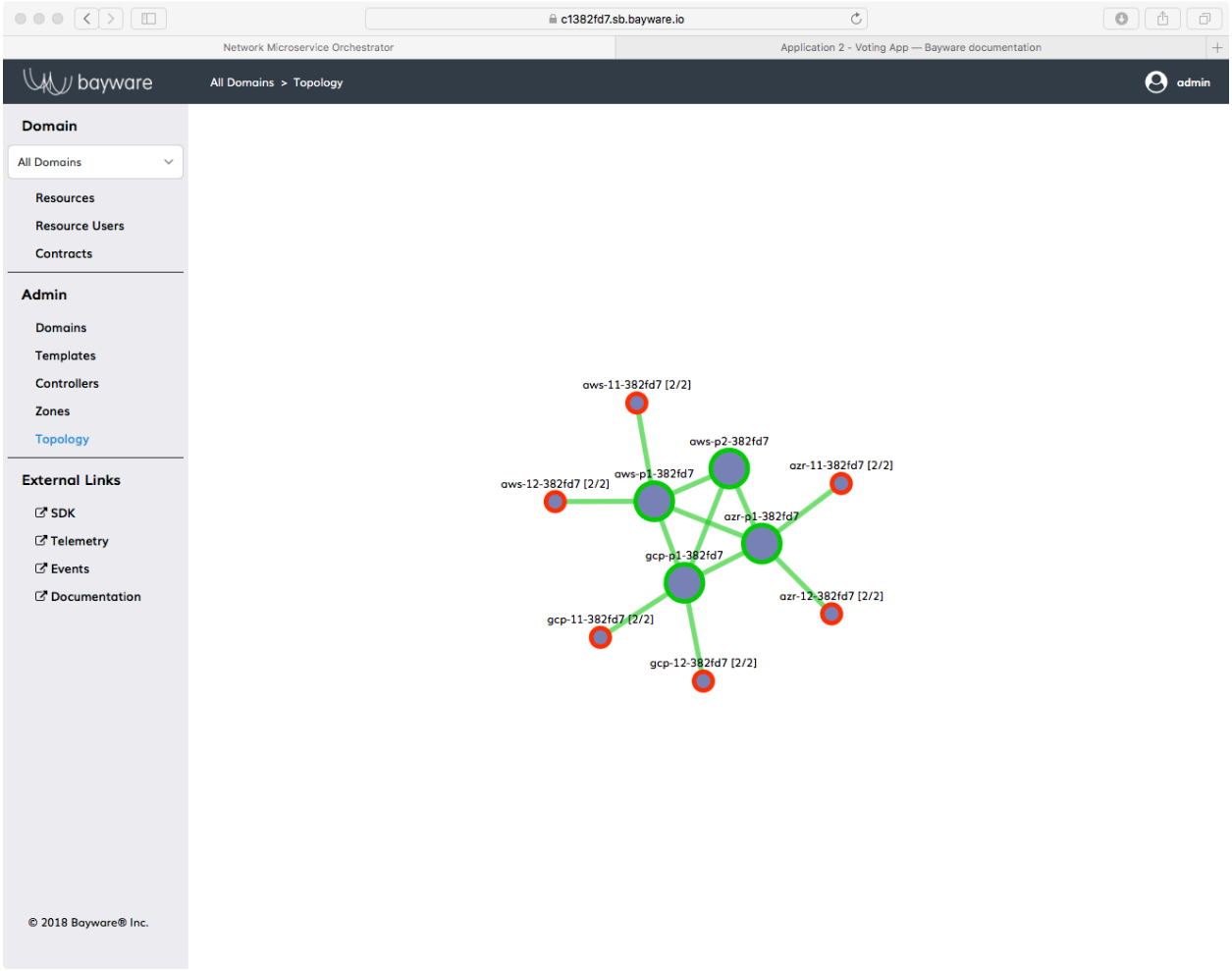
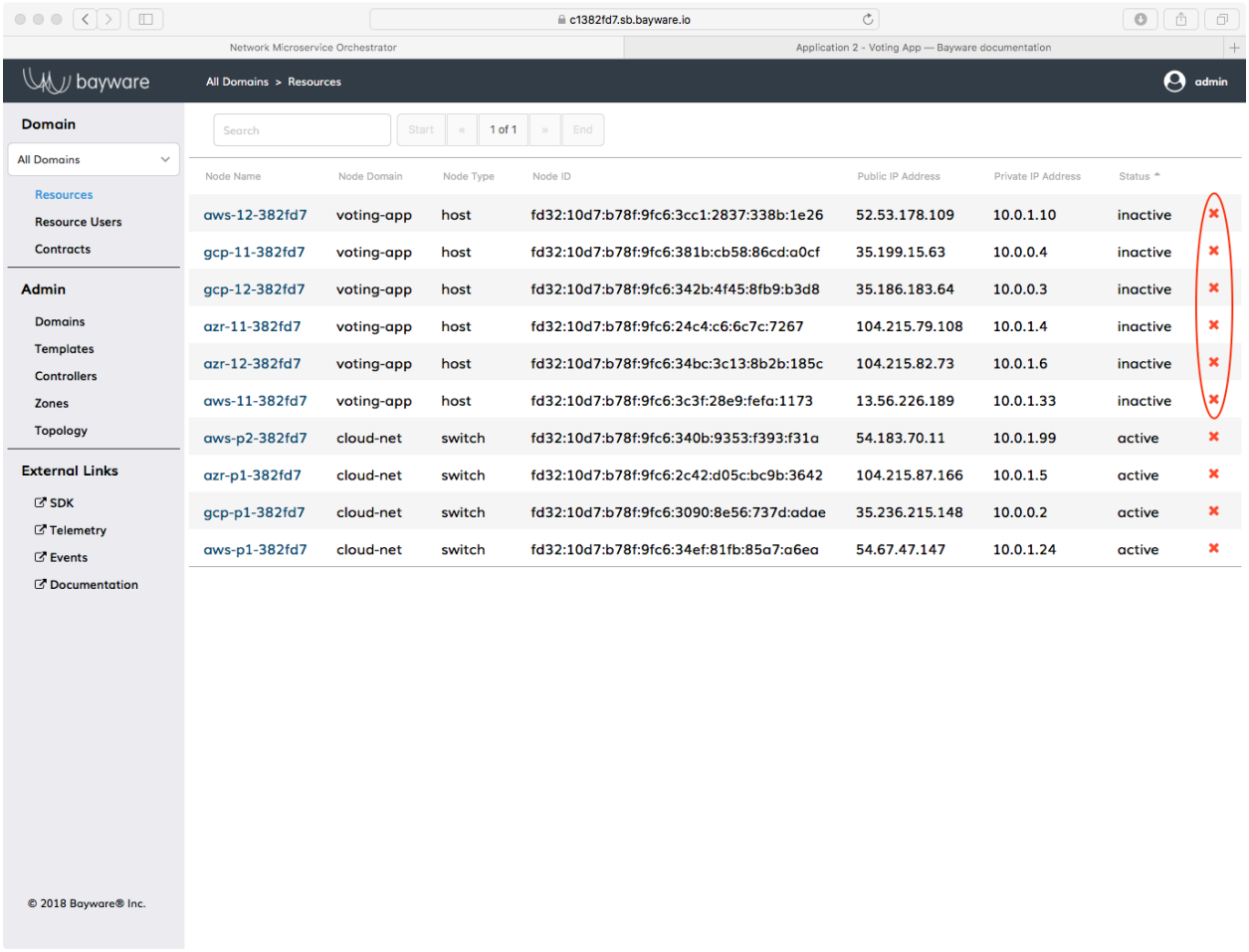


Fig. 12.45: Voting App Topology After Purge


c1382fd7.sb.bayware.io
Network Microservice Orchestrator
Application 2 - Voting App — Bayware documentation
bayware
All Domains > Resources
admin
Domain
All Domains
Resources
Resource Users
Contracts
Admin
Domains
Templates
Controllers
Zones
Topology
External Links
SDK
Telemetry
Events
Documentation
© 2018 Bayware® Inc.
Search
Start
1 of 1
End
Node Name
Node Domain
Node Type
Node ID
Public IP Address
Private IP Address
Status
aws-12-382fd7 voting-app host fd32:10d7:b78f:9fc6:3cc1:2837:338b:1e26 52.53.178.109 10.0.1.10 inactive
gcp-11-382fd7 voting-app host fd32:10d7:b78f:9fc6:381b:cb58:86cd:a0cf 35.199.15.63 10.0.0.4 inactive
gcp-12-382fd7 voting-app host fd32:10d7:b78f:9fc6:342b:4f45:8fb9:b3d8 35.186.183.64 10.0.0.3 inactive
azr-11-382fd7 voting-app host fd32:10d7:b78f:9fc6:24c4:c6:6c7c:7267 104.215.79.108 10.0.1.4 inactive
azr-12-382fd7 voting-app host fd32:10d7:b78f:9fc6:34bc:3c13:8b2b:185c 104.215.82.73 10.0.1.6 inactive
aws-11-382fd7 voting-app host fd32:10d7:b78f:9fc6:3c3f:28e9:fefa:1173 13.56.226.189 10.0.1.33 inactive
aws-p2-382fd7 cloud-net switch fd32:10d7:b78f:9fc6:340b:9353:f393:f31a 54.183.70.11 10.0.1.99 active
azr-p1-382fd7 cloud-net switch fd32:10d7:b78f:9fc6:2c42:d05c:bc9b:3642 104.215.87.166 10.0.1.5 active
gcp-p1-382fd7 cloud-net switch fd32:10d7:b78f:9fc6:3090:8e56:737d:adae 35.236.215.148 10.0.0.2 active
aws-p1-382fd7 cloud-net switch fd32:10d7:b78f:9fc6:34ef:81fb:85a7:a6ea 54.67.47.147 10.0.1.24 active

(continued from previous page)

```
We are happy that you have taken the opportunity to become more familiar with
Bayware's Network Microservices.

While you are working through Bayware's tutorial, you will be utilizing a suite
a virtual machines dedicated to you.  The machines come from three public cloud
providers:  Amazon Web Services (AWS), Microsoft Azure, and Google Cloud
Platform (GCP).

You have DNS access to the following services via your web browser.  You will
be asked to navigate to these pages during the tutorial.
```

```
+-------------------------------------------+-------------------------------+---------
↪-------------------------------------------+
| HTTP Address                              | Service Name                  |␣
↪Description                                          |
+===========================================+===============================+=========
| https://ap382fd7.sb.bayware.io/getaway    | Getaway App Entry Point       | Web␣
↪address for App1, the Getaway App                 |
+-------------------------------------------+-------------------------------+---------
↪-------------------------------------------+
| https://ap382fd7.sb.bayware.io/voting/vote | Voting App Input Entry Point  | Web␣
↪address for App2 input, the Voting App vote.      |
+-------------------------------------------+-------------------------------+---------
↪-------------------------------------------+
| https://ap382fd7.sb.bayware.io/voting/result | Voting App Output Entry Point | Web␣
↪address for App 2 output, the Voting App result.  |
+-------------------------------------------+-------------------------------+---------
↪-------------------------------------------+
| https://c1382fd7.sb.bayware.io            | Orchestrator                  | Bayware␣
↪orchestrator GUI interface application.       |
+-------------------------------------------+-------------------------------+---------
↪-------------------------------------------+

One VM in AWS functions as your Customer Command Center (aws-bastion-382fd7).
You will execute scripts on aws-bastion-382fd7 and you will SSH into other
VMs in your network from aws-bastion-382fd7. In the tutorial, this node
is generally referred to as your CCC.

You have password access to aws-bastion-382fd7.
```

```
bastion FQDN:  ap382fd7.sb.bayware.io
+-----+----------------------+-------------------+-------+-----------+-----------+-------
↪-------+--------------+----------+----------+
| No. | Host Name            | Description       | Cloud | Region    | Geography |␣
↪Public IP    | Private IP   | Username | Password |
+=====+======================+===================+=======+===========+===========+=======
| 1   | aws-bastion-382fd7   | customer cmd cntr | AWS   | us-west-1 | N. Calif  | 18.
↪219.1.181 | 172.18.89.89 | centos   | ik77J349 |
+-----+----------------------+-------------------+-------+-----------+-----------+-------
↪-------+--------------+----------+----------+
```

(continues on next page)

(continued from previous page)

```
The tutorial will provide you with detailed instructions for logging into your
command center.

The remaining VMs in your network are listed below.  Each VM contains a public
key that allows SSH access from your command center VM without a password.

+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| No. | Host Name     | Description      | Cloud | Region     | Geography   | Public IP ␣
↪  | Private IP    | Username | Password |
+=====+===============+==================+=======+============+=============+============
| 2   | aws-c1-382fd7 | Orch - Main      | AWS   | us-west-1  | N. Calif    | 18.219.1.
↪182 | 172.18.22.10  | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 3   | aws-c2-382fd7 | Orch - Telemetry | AWS   | us-west-1  | N. Calif    | 18.214.28.
↪16 | 172.18.18.3   | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 4   | aws-c3-382fd7 | Orch - Events    | AWS   | us-west-1  | N. Calif    | 18.214.28.
↪17 | 172.18.52.9   | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 5   | aws-p1-382fd7 | Proc - Pub Cld1  | AWS   | us-west-1  | N. Calif    | 18.214.28.
↪18 | 172.18.231.63 | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 6   | aws-p2-382fd7 | Proc - Pub Cld1  | AWS   | us-west-1  | N. Calif    | 18.214.28.
↪19 | 172.18.99.29  | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 7   | aws-11-382fd7 | Wkld - Pub Cld1  | AWS   | us-west-1  | N. Calif    | 18.214.28.
↪20 | 172.18.103.200 | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 8   | aws-12-382fd7 | Wkld - Pub Cld1  | AWS   | us-west-1  | N. Calif    | 18.214.28.
↪21 | 172.18.103.201 | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 9   | azr-p1-382fd7 | Proc - Pub Cld2  | Azure | S Cntrl US | Texas       | 44.22.81.4␣
↪  | 10.1.8.8      | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 10  | azr-11-382fd7 | Wkld - Pub Cld2  | Azure | S Cntrl US | Texas       | 44.22.81.5␣
↪  | 10.1.8.9      | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 11  | azr-12-382fd7 | Wkld - Pub Cld2  | Azure | S Cntrl US | Texas       | 44.22.81.6␣
↪  | 10.1.8.10     | centos   | n/a      |
+-----+---------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 12  | gcp-p1-382fd7 | Proc - Pub Cld3  | GCP   | us-east-4  | N. Virginia | 18.9.9.3  ␣
↪  | 10.87.1.11    | centos   | n/a      |
```

(continues on next page)

(continued from previous page)

```
+-----+--------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 13  | gcp-11-382fd7 | Wkld - Pub Cld3  | GCP   | us-east-4  | N. Virginia | 18.9.9.4  ␣
↪  | 10.87.1.2     | centos   | n/a      |
+-----+--------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 14  | gcp-12-382fd7 | Wkld - Pub Cld3  | GCP   | us-east-4  | N. Virginia | 18.9.9.5  ␣
↪  | 10.87.1.3     | centos   | n/a      |
+-----+--------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 15  | aws-21-382fd7 | Wkld - Pub Cld1  | AWS   | us-east-2  | Ohio        | 18.214.28.
↪22 | 172.18.103.201 | centos   | n/a      |
+-----+--------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 16  | aws-31-382fd7 | Wkld - Pub Cld1  | AWS   | us-east-2  | Ohio        | 18.214.28.
↪23 | 172.18.103.202 | centos   | n/a      |
+-----+--------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
| 17  | aws-41-382fd7 | Wkld - Pub Cld1  | AWS   | us-east-1  | N. Virginia | 18.214.28.
↪24 | 172.18.103.203 | centos   | n/a      |
+-----+--------------+------------------+-------+------------+-------------+------------
↪--+---------------+----------+----------+
```

```
Your Bayware Orchestrator login credentials

+-----+--------------+---------+----------+------------------+
| No. | VM Function  | Domain  | Username | Password         |
+=====+==============+=========+==========+==================+
| 18  | orchestrator | default | admin    | greencowsdrive13 |
+-----+--------------+---------+----------+------------------+
```

```
Your Bayware Processor login credentials

+-----+-------------+------------------------+-----------+----------+--------------+
| No. | VM Function | Orch FQDN              | Domain    | Username | Password     |
+=====+=============+========================+===========+==========+==============+
| 19  | processor   | c1382fd7.sb.bayware.io | cloud-net | proc-1   | olivesoda74  |
+-----+-------------+------------------------+-----------+----------+--------------+
| 20  | processor   | c1382fd7.sb.bayware.io | cloud-net | proc-2   | itchysugar32 |
+-----+-------------+------------------------+-----------+----------+--------------+
| 21  | processor   | c1382fd7.sb.bayware.io | cloud-net | proc-3   | greatmoose33 |
+-----+-------------+------------------------+-----------+----------+--------------+
| 22  | processor   | c1382fd7.sb.bayware.io | cloud-net | proc-4   | lumpywish36  |
+-----+-------------+------------------------+-----------+----------+--------------+
```

```
Your Bayware Agent login credentials Application1:  Getaway App

+-----+-------------+------------------------+-------------+-------------+---------------
↪+
| No. | VM Function | Orch FQDN              | Domain      | Username    | Password      ␣
↪|
```

(continues on next page)

(continued from previous page)

```
+=====+=============+=======================+=============+=============+===============+
| 23  | workload    | c1382fd7.sb.bayware.io | getaway-app | http-proxy  | curlycanary14 |
+-----+-------------+-----------------------+-------------+-------------+---------------+
| 24  | workload    | c1382fd7.sb.bayware.io | getaway-app | getaway-svc | busyparrot65  |
+-----+-------------+-----------------------+-------------+-------------+---------------+
| 25  | workload    | c1382fd7.sb.bayware.io | getaway-app | news-gw     | silkycat20    |
+-----+-------------+-----------------------+-------------+-------------+---------------+
| 26  | workload    | c1382fd7.sb.bayware.io | getaway-app | places-gw   | giantstop52   |
+-----+-------------+-----------------------+-------------+-------------+---------------+
| 27  | workload    | c1382fd7.sb.bayware.io | getaway-app | weather-gw  | fuzzylamb20   |
+-----+-------------+-----------------------+-------------+-------------+---------------+
```

```
Your Bayware Workload login credentials Application2:  Voting App

+-----+-------------+-----------------------+------------+------------+--------------+
| No. | VM Function | Orch FQDN             | Domain     | Username   | Password     |
+=====+=============+=======================+============+============+==============+
| 28  | workload    | c1382fd7.sb.bayware.io | voting-app | http-proxy | swiftstar33  |
+-----+-------------+-----------------------+------------+------------+--------------+
| 29  | workload    | c1382fd7.sb.bayware.io | voting-app | voting-svc | messycard58  |
+-----+-------------+-----------------------+------------+------------+--------------+
| 30  | workload    | c1382fd7.sb.bayware.io | voting-app | voting-db  | emptypet53   |
+-----+-------------+-----------------------+------------+------------+--------------+
| 31  | workload    | c1382fd7.sb.bayware.io | voting-app | result-svc | poorhelp59   |
+-----+-------------+-----------------------+------------+------------+--------------+
| 32  | workload    | c1382fd7.sb.bayware.io | voting-app | result-db  | wildsummer93 |
+-----+-------------+-----------------------+------------+------------+--------------+
| 33  | workload    | c1382fd7.sb.bayware.io | voting-app | worker     | tallfeet16   |
+-----+-------------+-----------------------+------------+------------+--------------+
```

12.6 Troubleshooting

12.6.1 Bayware Engine Diagnostics

1. I have just installed and configured the Bayware processor (proc-1, proc-2, proc-3, proc-4), but it does not show up in *Topology* on the orchestrator.

   - Ensure that the service is running. As `root` on the processor node, type the following

     ```
     ]# systemctl status ib_engine
     ```

The status should indicate both *loaded* and *active* (along with either *exited* or *running*). If you do not see this status, restart the service

```
]# systemctl restart ib_engine
```

- Check login credentials used to attach node to orchestrator. You can verify the orchestrator FQDN, domain, username, and password used during engine configuration. As `root`, view the following

```
]# more /opt/ib_engine/releases/1/sys.config
```

Find `keystone_token` near the top. This shows the FQDN of the orchestrator (ignore the trailing path), for example

```
{keystone_token,"https://c1382fd7.sb.bayware.io/api/v1"}
```

You would ensure that `c1382fd7.sb.bayware.io` matches the FQDN for the orchestrator shown on the SIS. You can find the SIS FQDN in the *URL section* (everything that comes after `https://` for the orchestrator row).

Search further for `login`, `password`, and `domain` and ensure that these match *processor login credentials* on your SIS.

If credentials do not match, simply re-run the `ib_engine` configuration script again

```
]# /opt/ib_engine/bin/ib_configure -i
```

2. The Bayware processor shows up on the orchestrator, but it doesn't make connections with any other processor.
   - Be patient. It can take up to one minute to form the link.
   - Click *Resources* and then click on the processor *Node Name* on the orchestrator. Scroll to the bottom and ensure you configured the expected link.
   - As `root` on the processor node, ensure the IPsec client, strongSwan, has been configured.

```
]# systemctl status strongswan
```

If strongSwan is not active, **`restart`** the service

```
]# systemctl restart strongswan
```

Once strongSwan is active, ensure that it has security associations set up with other nodes. There should be one security association established for each green link shown on the *Topology* page.

```
]# strongswan status
```

If there are no security associations or if **`systemctl`** indicated that the strongswan service is not running, then it may not have been configured. Re-run engine configuration bullet point *above* and be sure to answer `yes` to IPsec.

12.6.2 Bayware Agent Diagnostics

1. I have just installed and configured the Bayware agent, but it does not show up in *Topology* on the orchestrator.
   - Ensure that the service is running. As `root` on the workload node, type the following

```
]# systemctl status ib_agent
```

The status should indicate both *loaded* and *active (running)* If you do not see this status, restart the service

```
]# systemctl restart ib_agent
```

- Check login credentials used to attach node to orchestrator. You can verify the orchestrator FQDN, domain, username, and password used during agent configuration. As `root`, view the following

```
]# more /etc/ib_agent.conf
```

Ensure correct `controller_ip` by cross-checking the IP address with that for `aws-c1` on your SIS. Ensure correct `login`, `password`, and `domain` with that expected from the SIS.

If credentials do not match, simply re-run the `ib_agent` configuration script again

```
]# /opt/ib_agent/bin/ib_configure -i
```

- Check `ib_agent` status to ensure that it is properly registered with the orchestrator. To do this, you need the IP address and port used for the REST interface. Look for the `[rest]` section near the bottom of the following file

```
]# more /etc/ib_agent.conf
```

It should look like

```
...

[rest]
rest_ip = 192.168.250.1
rest_port = 5500
log_file = /var/log/ib_agent_rest.log
log_level = DEBUG
```

Note the `rest_ip` and `rest_port` and use them in the following `curl` command. For example,

```
[root@aws-11-382fd7 ~]# curl 192.168.250.1:5500/api/v1/status
```

The `ready`, `registered`, and `success` keys should all be assigned a value of `true`. You can also verify login credentials as well as orchestrator IP address (which is called `controller` in this context.)

2. The Bayware agent shows up on the orchestrator, but it doesn't make connections with any other processor.

- Be patient. It can take up to one minute to form the link.
- As `root` on the workload node, ensure the IPsec client, strongSwan, has been configured.

```
]# systemctl status strongswan
```

If strongSwan is not active, **`restart`** the service

```
]# systemctl restart strongswan
```

Once strongSwan is active, ensure that it has security associations set up with other nodes. There should be one security association established for each green link shown on the *Topology* page.

```
]# strongswan status
```

If there are no security associations or if **systemctl** indicates that the strongswan service is not running, then it may not have been configured. Re-run agent configuration bullet point *above* and be sure to answer `yes` to IPsec.

12.6.3 Getaway App & Voting App Diagnostics

It's best to ensure that the App is running on a properly configured Bayware interconnection fabric before checking individual microservices.

Table 12.6: Getaway App Connectivity

Host	Host Owner	URL
`aws-11`	`http-proxy`	frontend.getaway-app.ib.loc
`aws-12`	`getaway-svc`	news-api.getaway-app.ib.loc
`aws-12`	`getaway-svc`	weather-api.getaway-app.ib.loc
`aws-12`	`getaway-svc`	places-api.getaway-app.ib.loc

Table 12.7: Voting App Connectivity

Host	Host Owner	URL
`aws-11`	`http-proxy`	result-frontend.voting-app.ib.loc
`aws-11`	`http-proxy`	voting-frontend.voting-app.ib.loc
`aws-12`	`worker`	result-worker.voting-app.ib.loc
`aws-12`	`worker`	voting-worker.voting-app.ib.loc
`gcp-11`	`voting-svc`	voting-backend.voting-app.ib.loc
`azr-11`	`result-svc`	result-backend.voting-app.ib.loc

Do this by logging in to one of the workload hosts listed in Table 12.6 and Table 12.7. From the workload host, issue a **ping** command to the URL listed. For example,

```
[centos@aws-11-382fd7]$ ping frontend.getaway-app.ib.loc
```

If connectivity exists over the Bayware interconnection fabric, then you should see ongoing responses indicating `64 bytes from ...`. If you do *not* see response packets, then resume troubleshooting `ib_agent` and `ib_engine` in the sections above.

If you do see ping response packets as indicated, then ensure the application service units are installed and running on the proper VMs. This is performed differently for Getaway App and Voting App.

With Getaway App for instance, as indicated in *Getaway Microservices VM Mapping*, the `http-proxy` microservice running on `aws-11` relies on a service unit called `getaway-proxy`. `getaway-proxy` should be installed and started on `aws-11`. Login to `aws-11` as `root` and ensure it is installed

```
]# yum list installed | grep getaway-proxy
```

If you get a positive response, then ensure that the service unit is running under `systemd`

```
]# systemctl status getaway-proxy
```

You should see a response of `active (running)`. If the service unit is not installed or it is not running, you can follow the tutorial installation instructions to reinstall and start or restart it (`systemctl start getaway-proxy` or `systemctl restart getaway-proxy`).

Also ensure that only a single getaway service is running i.e., there should be only a single getaway-* listed among running services. Show all running services with

```
]# systemctl list-units --type service
```

If an unexpected getaway-* service appears in the list, stop the service. For example, to stop the getaway-service service

```
]# systemctl stop getaway-service
```

With Voting App, you should find a container image running on each VM where you expect a microservice. Login to a workload node as `root` and execute the following

```
]# systemctl list-units --type service --all | grep _container
```

A positive response should show a container service recognizable as being part of Voting App, for instance, `http-proxy_container`. It should be `loaded`, `active`, and `running` with an output similar to

```
http-proxy_container.service       loaded    active   running "http-proxy_container"
```

Re-run the `deploy-voting-app.sh` script as described in *Installation with Ansible* if any service is missing or its status is incorrect.

CHAPTER 13

Deploying a Geo-Redundant App

13.1 Introduction

This tutorial demonstrates how to use Service Interconnection Fabric to quickly and securely deploy a three-tier, geo-redudant application in a multi-cloud or a hybrid-cloud environment using only a description of the application topology. The application topology can be depicted in an application service-type graph, which we will refer to here simply as the service graph.

After describing the service graph in the Service Interconnection Fabric orchestrator, you will see how easy it is to move service instances from private data centers to public clouds or between public clouds: all network address translation, encryption, protocol filtering, and routing will be established automatically.

13.1.1 The Scenario

You will install Getaway, a three-tier application spread across three VPCs located in three public clouds. The first VPC, located in AWS, contains a virtual machine (VM) running microservice getaway-proxy that functions as the presentation tier while another VM running microservice getaway-service functions as the application tier.

The AWS VPC and virtual machines come pre-installed in this tutorial. An additional AWS VPC contains three virtual machines that comprise the orchestrator components: a controller, a telemetry node, and an events node. These are also pre-installed.

The next VPC, in Google Cloud (GCP), functions as the data tier by responding to requests from the application tier for weather, places, and news data for a given city. You will install this VPC and its three VMs using the Fabric Manager (FM) node.

The final VPC, located in Microsoft's Azure, duplicates the weather, places, and news data VMs from the GCP VPC to create a geographically-redundant data center for disaster recovery or data migration from one cloud to another. Again, you will also install these resources using the Fabric Manager.

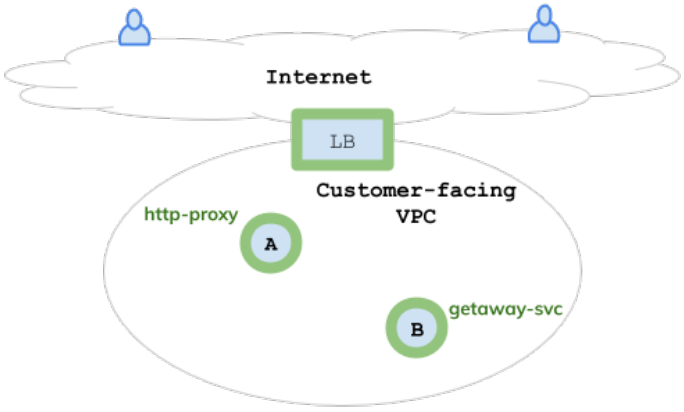


Fig. 13.1: AWS VPC: Proxy + Application

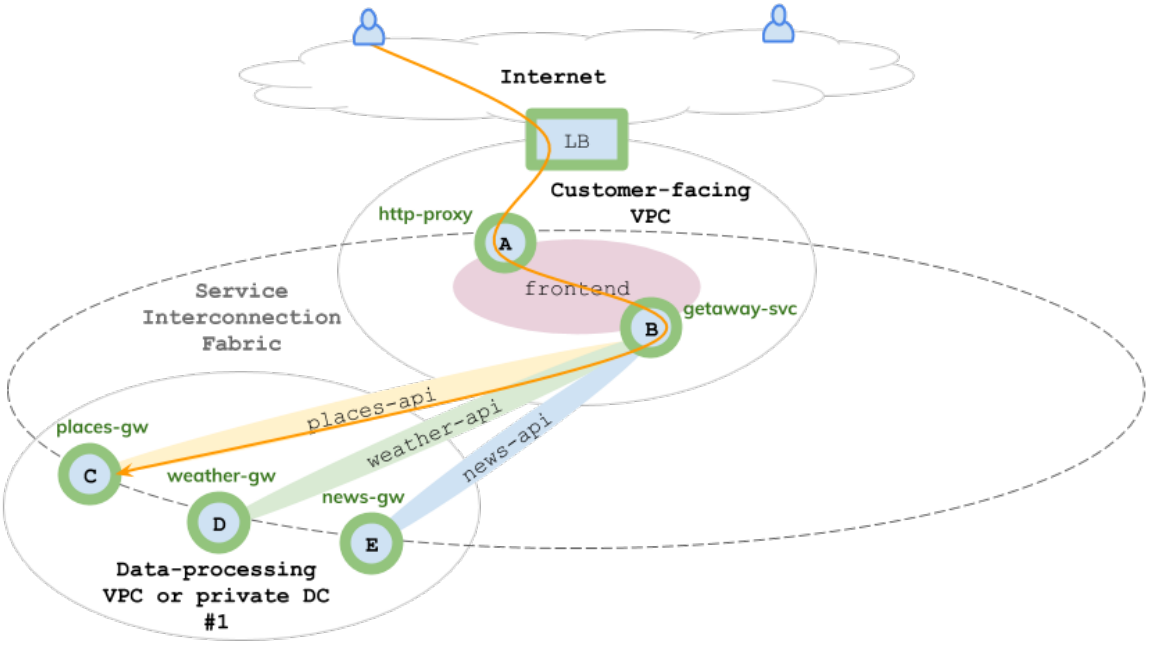


Fig. 13.2: GCP VPC: Weather, Places & News Data

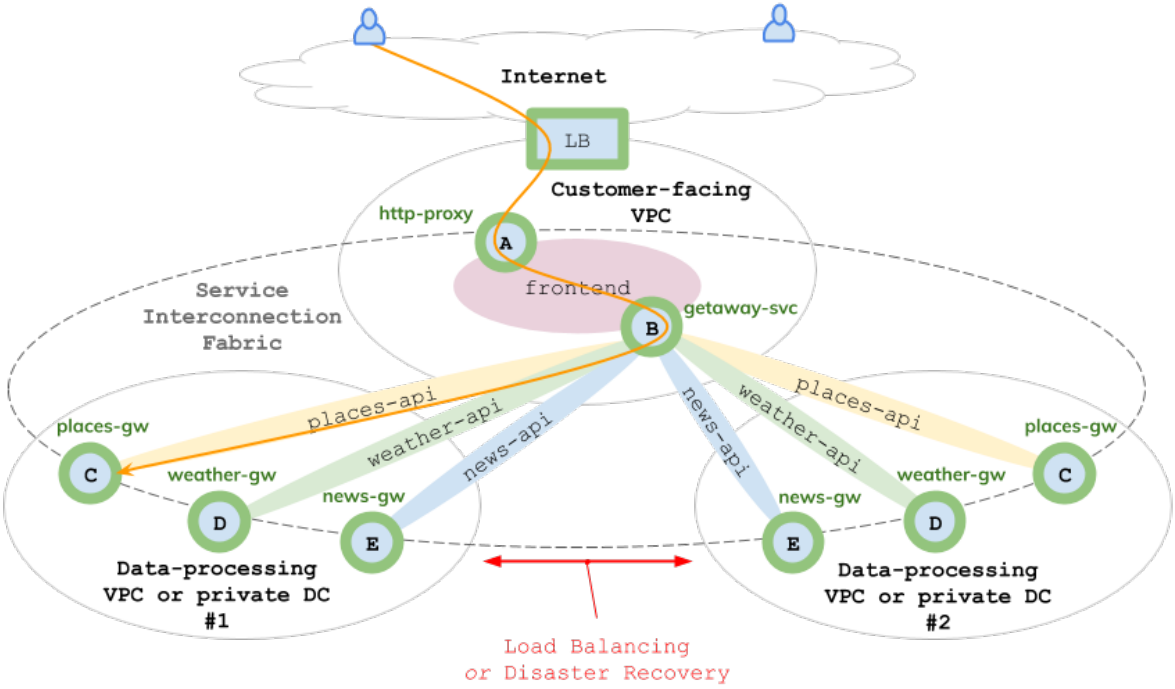


Fig. 13.3: Azure VPC: Disaster Recovery

13.1.2 Personalized Installation

Bayware creates a customized tutorial sandbox for each user. Your welcome email has an attachment called *Sandbox Installation Summary (SIS)*, which looks similar to the following

The *Orchestrator* and *Getaway* buttons are linked to your dedicated sandbox orchestrator and the example application you are building, respectively. The former requires login credentials, which are given in the SIS; the latter won't resolve until you finish deploying the application.

Use these buttons whenever you need to open the orchestrator or Getaway in a browser window. You may also copy and paste the underlying links into your browser, if that suits your needs better.

13.1.3 Fabric Manager

You will interact with the application's infrastructure through the Fabric Manager (FM). The FM has two command-line tools: **`bwctl`** and **`bwctl-api`** that allow you to manage cloud-based infrastructure resources and application policy, respectively.

To get started, open a terminal window on your computer as outlined in *Requirements* for your operating system. At the prompt, use the user name and FQDN for your fabric manager that is in the welcome email SIS attachment and type (using your FM FQDN, not the example FQDN for *jsmith* shown here):

```
]$ ssh ubuntu@fm.jsmithinc.poc.bayware.io
```

Be aware that the first time you log in to a new machine, you might see a warning message that looks like

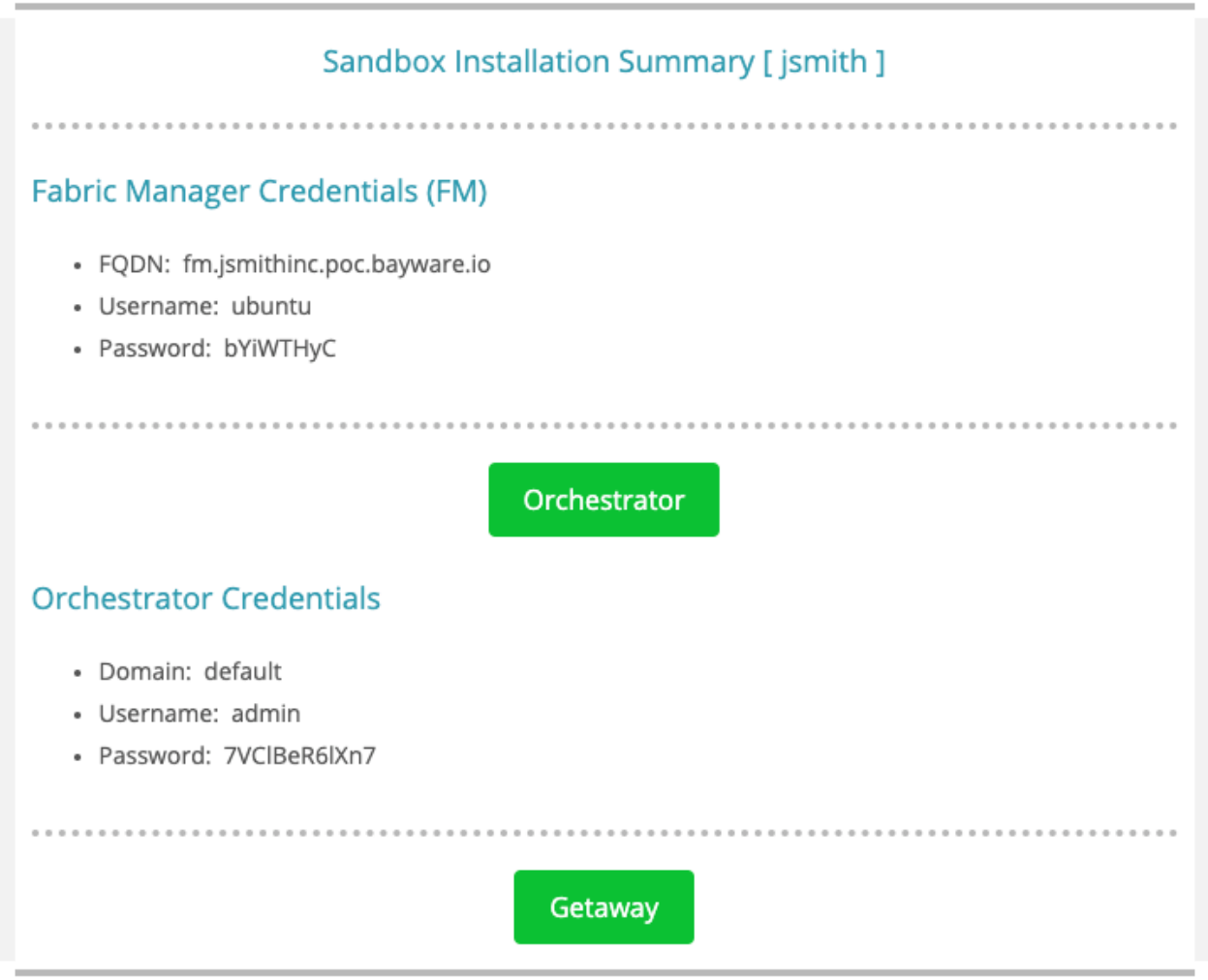


Fig. 13.4: Example Sandbox Installation Summary

```
]$ ssh ubuntu@fm.jsmithinc.poc.bayware.io
The authenticity of host 'jsmithinc.poc.bayware.io (13.56.241.123)' can't be established.
ECDSA key fingerprint is SHA256:6LLVP+3QvrIb8FjRGNleLQRy7zL2eXeNCdOoYRbbxqw.
ECDSA key fingerprint is MD5:7b:fd:15:4c:35:d3:1d:20:fd:3e:3d:b7:1b:14:6a:1b.
```

Where it asks if you wish to continue, just type `yes`.

```
Are you sure you want to continue connecting (yes/no)? yes
```

You will be prompted for your password with the following query

```
ubuntu@fm.jsmithinc.poc.bayware.io password:
```

Type in the password for your fabric manager. You should now be logged in and ready to go. Your Linux command-line prompt should look similar to

```
ubuntu@jsmith-c0:~$
```

Throughout the remainder of the tutorial, the Linux command-line prompt will be abbreviated as

```
]$
```

If it's not entirely clear, keep in mind that you can open up as many SSH sessions with your fabric manager as you require. Simply fire up another terminal window on your computer and log in as just described.

13.1.4 Orchestrator: Controller, Telemetry, & Events

Access the orchestrator from a browser window using the button and credentials shown in your welcome email (see *this* example above). You will need `Domain`, `User Name`, and `Password` as shown in Fig. 13.5. From the orchestrator's controller window you will be able to access windows for telemetry and events using the sidebar navigation menu. Be sure to keep the orchestrator browser window open throughout this tutorial since you will be referring to it frequently.

13.1.5 Summary

In this section you were introduced to Getaway, the application you will work with throughout this tutorial. You prepared for the following sections by learning how to log into your Fabric Manager node and bring up the SIF Orchestrator in your browser.

Next up: build out the infrastructure needed to deploy Getaway services in Google Cloud and Azure.

13.2 Application Infrastructure

13.2.1 Requirements

The required infrastructure must be geographically-distributed, preferably among different public cloud providers. It must have enough VMs to function as workload nodes to run five unique microservices, with three of the five duplicated, for a total of eight.

The Service Interconnection Fabric fulfills these requirements by creating VPCs and VMs, as described in the next section.

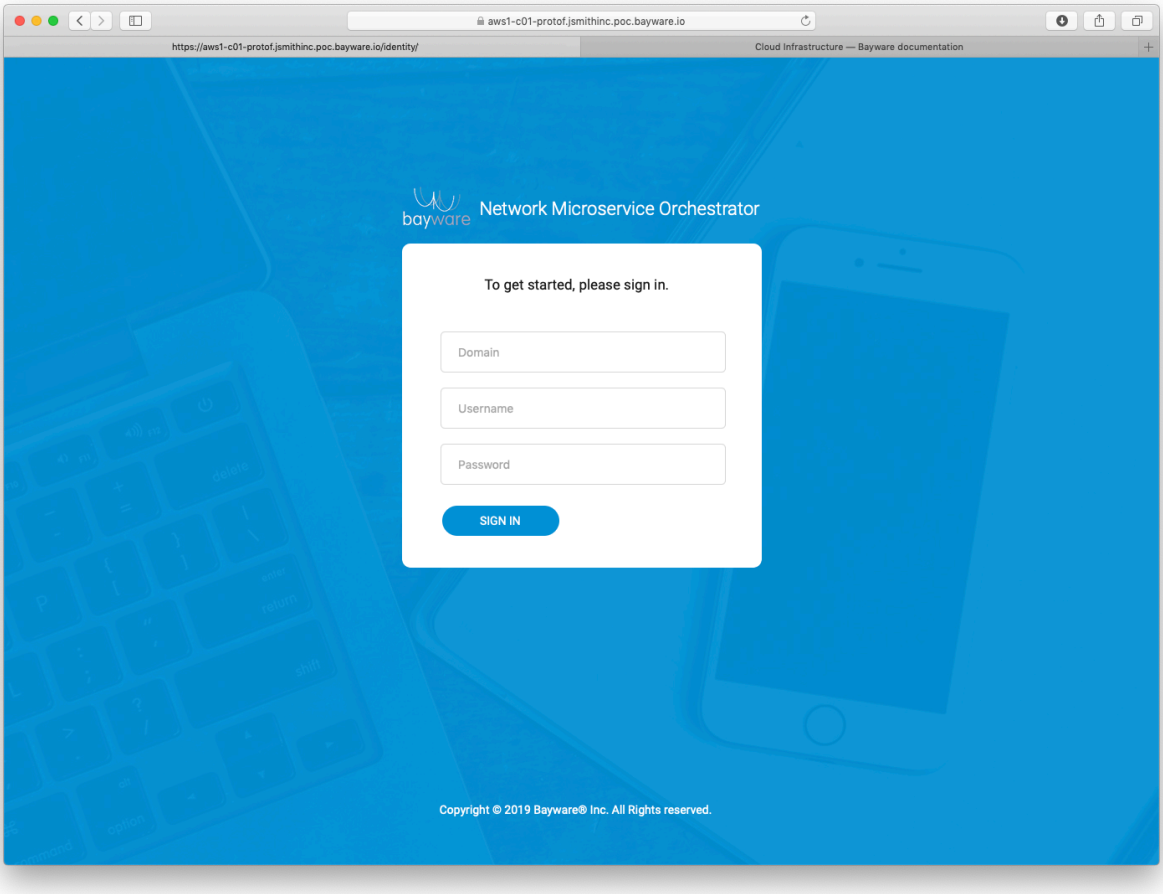


Fig. 13.5: Orchestrator Login Page

13.2.2 Service Interconnection Fabric

One can operate in isolated networks in public clouds by creating Virtual Private Clouds (VPCs). Within a VPC, one can create virtual machines (VMs) giving a virtual computing instance running its own operating system.

A Service Interconnection Fabric (SIF) is comprised of a set of VPCs (in a single public cloud, in multiple public clouds, or in public clouds and private data centers) whose VMs inter-communicate only through SIF processors. Generally, one SIF processor is deployed on one VM in each VPC.

We collectively refer to VMs used as SIF processors and VMs used to run application microservices as nodes.

In the remainder of this section, you will create VPCs and nodes.

13.2.3 Fabric Manager Tool: bwctl

This tutorial uses infrastructure components in AWS, Google Cloud, and Azure. The components in AWS are pre-installed; you will create the components in GCP and Azure using your fabric manager's tool called **bwctl**.

AWS Infrastructure

...on the Orchestrator

The AWS VPC contains two workload nodes and a processor node to connect it to the service interconnection fabric. You can see this on the orchestrator.

Return to your browser tab with the open orchestrator GUI. (Recall that you can find the orchestrator button and credentials on the SIS attachment from your welcome email.) Click *Resource Graph* in the sidebar navigation menu. You should see one large green circle representing the processor node and two smaller green circles representing the workload nodes as shown in Fig. 13.6.

...and from the FM CLI

Fabric Manager's command-line interface, **bwctl**, was used to create the AWS infrastructure just as you will be using it to create additional infrastructure in GCP and Azure in the following sections. You can use **bwctl** both interactively and from the Linux prompt. Let's take a look at the AWS infrastructure from within **bwctl** by issuing a **show** command.

To do this, return to your terminal window where you are logged into FM and type `bwctl` at the Linux prompt to start an interactive session.

```
]$ bwctl
```

You should be at the **bwctl** prompt

```
(none) bwctl>
```

This prompt shows you are operating outside of a fabric (or namespace) with the word `none`. Determine the name of the fabric pre-installed for you by typing

```
(none) bwctl> show fabric --list-all
```

You will see output similar to

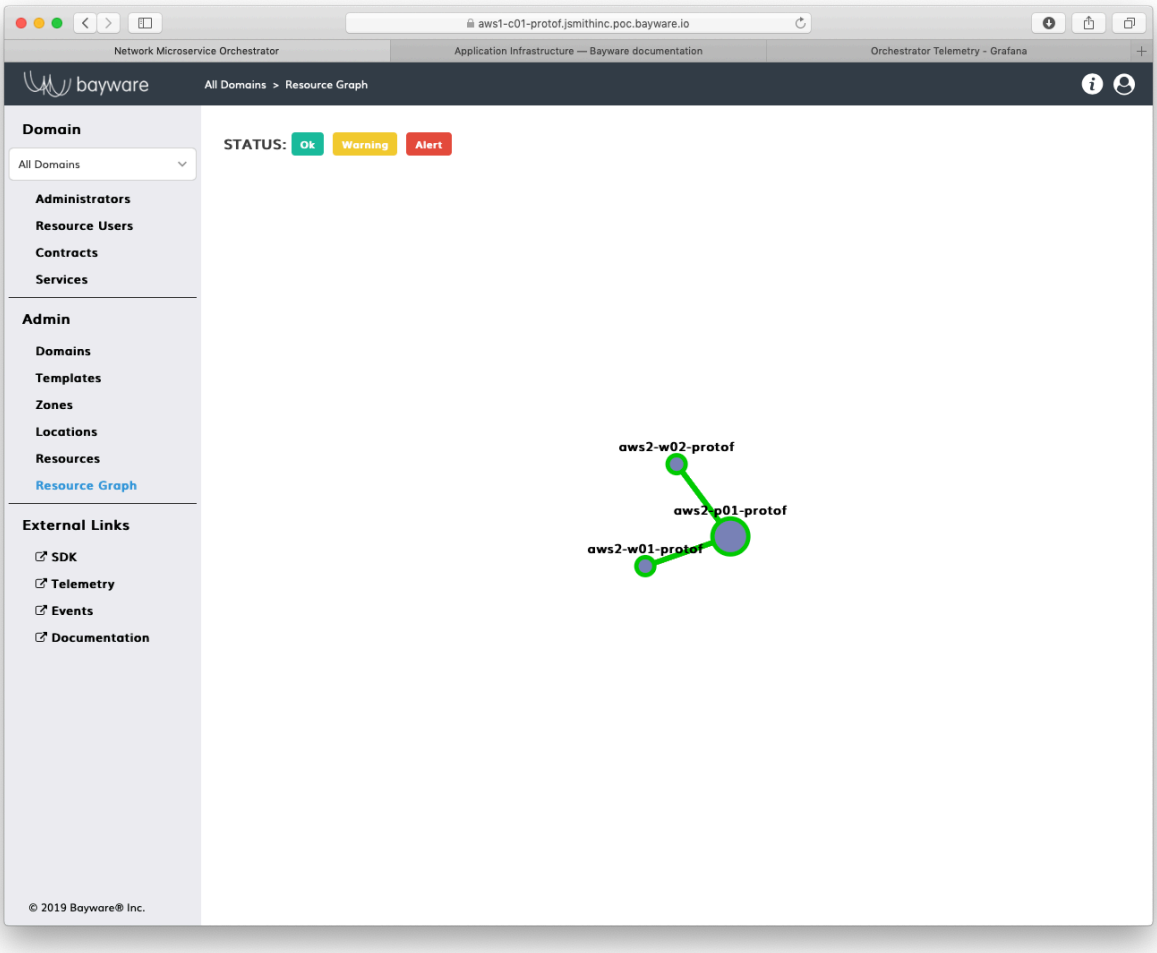


Fig. 13.6: AWS Infrastructure on the Orchestrator

```
FABRIC
protof
```

In this example, the name of the pre-installed fabric is **protof**. Now enter the namespace of your fabric by typing the following (using your own fabric name) at the **bwctl** prompt

```
(none) bwctl> set fabric protof
```

You should now see output similar to

```
[2019-05-10 00:04:19.624]  Active fabric: 'protof'
```

Notice that your **bwctl** prompt has changed, now showing the active fabric

```
(protof) bwctl>
```

Now show the components of your fabric by entering

```
(protof) bwctl> show fabric
```

Take some time to scroll up and scan through the components in your fabric. At the bottom you'll see the AWS VPCs. A little further up you'll see six nodes: three of these comprise the orchestrator, two are used as workloads, and another is used as a processor node.

In the next section you'll examine a YAML file that is used as input to the **bwctl** tool in order to create infrastructure.

For now, simply quit out of the interactive **bwctl** session by typing

```
(protof) bwctl> quit
```

You should now be back at the Linux prompt.

```
ubuntu@jsmith-c0:~$
```

GCP Infrastructure

In this section you will add components from GCP: one VPC, one processor node, and three workload nodes. The tool will add a small piece of software to the processor node to create an engine. The engine provides communication and policy enforcement among VPCs and workloads. The tool will also add a small piece of software to the workload nodes called the agent. The agent acts as a daemon for communicating with engines and the orchestrator.

BWCTL Batch Processing

While you could issue commands interactively at the **bwctl** prompt to add each component to the fabric, it will be more expedient to use a **bwctl batch** command and have the tool do everything in one shot.

bwctl batch commands are issued at the Linux prompt and operate on an input file in YAML format that describes the desired infrastructure. As an example, consider the following

```
---

apiVersion: fabric.bayware.io/v1
kind: Batch
metadata:
  name: backend-infra-and-config-template
  description: 'Creates VPC, processor, and three workloads'
spec:
  - kind: Fabric
    metadata:
      description: 'optional description'
      name: 'protof'
    spec:
      companyName: acmeinc
      credentialsFile:
        aws: /home/ubuntu/credentials/aws-cred.yaml
        azr: /home/ubuntu/credentials/azr-cred.yaml
        gcp: /home/ubuntu/credentials/gcp-credentials.json
        ssh: {}
        s3: /home/ubuntu/credentials/s3-cred.yaml
  - kind: Node
    metadata:
      description: 'optional description'
      fabric: 'protof'
      name: 'gcp1-p01-protof'
    spec:
      properties:
      type: 'processor'
      vpc: 'gcp1-vpc-protof'
  - kind: Vpc
    metadata:
      description: 'optional description'
      fabric: 'protof'
      name: 'gcp1-vpc-protof'
    spec:
      cloud: 'gcp'
      properties:
      zone: 'us-east4'
```

This YAML file specifies three components: a fabric (or namespace) called `protof`, a node for a processor, and a VPC in GCP. When executed, the tool determines which components already exist and adds the remaining components to bring the current state of the fabric in line with the YAML description.

GCP Infrastructure YAML Description

To see the components that you will add in GCP, you can view the YAML file description using the popular Linux **cat** command. The file is located in the **~ubuntu** home directory, which is where you should be in the terminal session open on your FM node if you've been following along. Type the following at the Linux prompt

```
]$ cat gcp-infra-batch.yml
```

The contents of the file will be displayed on the screen. You can use the scroll feature of your terminal

window to see the entire file if it doesn't fit on your screen.

GCP Infrastructure: create batch

Now you'll actually execute the command to add the GCP infrastructure. Again, in the `~ubuntu` home directory, type the following

```
]$ bwctl create batch gcp-infra-batch.yml
```

This command kicks off a series of Terraform and Ansible instructions that interact with Google Cloud and with the newly-created virtual machines. The whole process takes approximately 10 minutes.

Continue with the tutorial once the command completes.

GCP Infrastructure: on the Orchestrator

Now return to the browser tab that has the open orchestrator GUI. Click on the *Resource Graph* button in the navigation menu. If everything went well, you should now see an additional processor node (a large green circle) and three additional workload nodes (small green circles). The new processor node should have a connection to the processor node in AWS and the three new workload nodes should be connected to the GCP processor node, as shown in Fig. 13.7.

Azure Infrastructure

Similar to GCP, you will now create one VPC, one processor node, and three workload nodes in Microsoft's Azure. You will do this in a similar manner, using a **`bwctl batch`** command with a YAML description of the resources.

Azure Infrastructure YAML Description

Creating infrastructure in Azure uses the same type of YAML description that you used in GCP. In fact, the two files are nearly identical. Take a look at the Azure YAML infrastructure description using the **`cat`** command

```
]$ cat azr-infra-batch.yml
```

As you scan through the file, notice that only the name of the cloud provider has changed and the region-specific information.

Azure Infrastructure: create batch

Go ahead and execute the command

```
]$ bwctl create batch azr-infra-batch.yml
```

As before, wait approximately 12 minutes for this to complete before continuing.

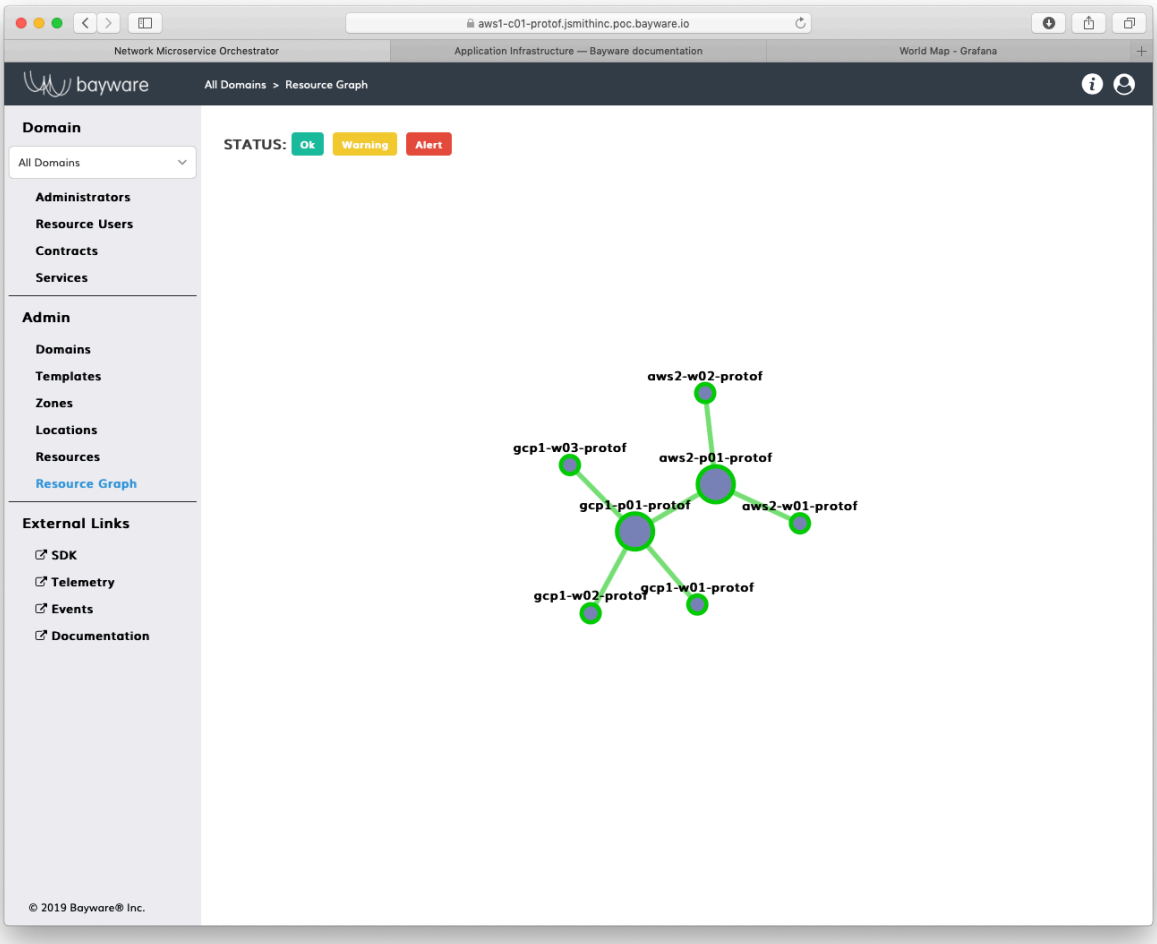


Fig. 13.7: AWS & GCP Infrastructure Resource Graph

Azure Infrastructure: on the Orchestrator

Now that the tool has completed adding the Azure components, return to the orchestrator browser window's *Resource Graph* page. The additional Azure processor should be present and connected to both the GCP and AWS processors and the new Azure workload nodes should be connected to the Azure processor, as shown in Fig. 13.8.

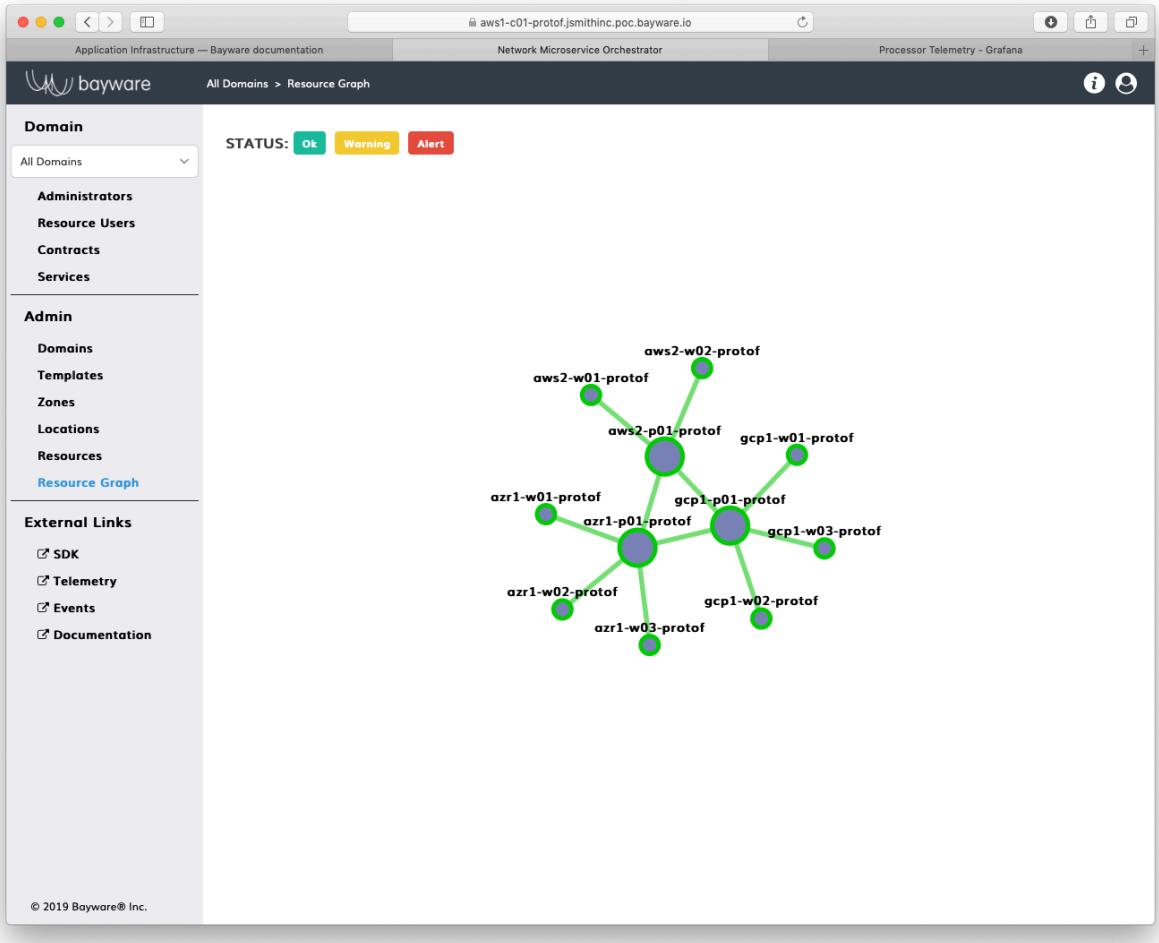


Fig. 13.8: AWS & GCP & Azure Infrastructure Resource Graph

Completed Infrastructure

Before moving on, use **bwctl** interactively once more to show all the components in your fabric as you did in *AWS Infrastructure* above. That is, first type **bwctl** at the Linux prompt

```
]$ bwctl
```

which should put you at the **bwctl** command line. This should look similar to

```
(protof) bwctl>
```

but your fabric name will be different. Then type `show fabric`

```
(protof) bwctl> show fabric
```

As you scroll through the output, notice the newly-created components in GCP and Azure.

Don't forget to quit out of the interactive **`bwctl`** session by typing `quit` at the `bwctl` prompt:

```
(protof) bwctl> quit
```

You should now be back at the Linux prompt.

```
ubuntu@jsmith-c0:~$
```

13.2.4 Summary

In this section you used your fabric manager's infrastructure command-line tool, **`bwctl`**, both interactively, to show fabric infrastructure, and in batch mode, to create new infrastructure. The YAML batch file descriptions demonstrated how creating the same components in different cloud providers is nearly as simple as doing a search-and-replace. You also used the orchestrator GUI to track components as they registered themselves with the controller.

Next up: use fabric manager's **`bwctl-api`** command-line tool to interact with the orchestrator's controller via an API.

13.3 Application Policy

13.3.1 Requirements

Your application service graph defines the required communicative relationships between the application microservices. The service graph for Getaway is shown in Fig. 13.9.

Service Interconnection Fabric allows the application developer to set policy based on the relationships between the nodes in the service graph. To do this, Bayware introduces two new concepts: services and contracts.

13.3.2 Service Graph

Nodes & Edges

The nodes in the service graph map to SIF services. Edges between nodes in the service graph map to SIF contracts. One service may communicate with another service only through a contract. The contract defines the policy that allows the services to communicate.

Contracts in the Service Interconnection Fabric

A contract, which defines policy that allows one service to communicate with another service, may be used by more than two services within a service graph. It may describe, for instance, policy for point-to-multipoint communication and not simply point-to-point communcation. A contract is essentially a waypoint on the path between services that describes the policy that allows those services to communicate.

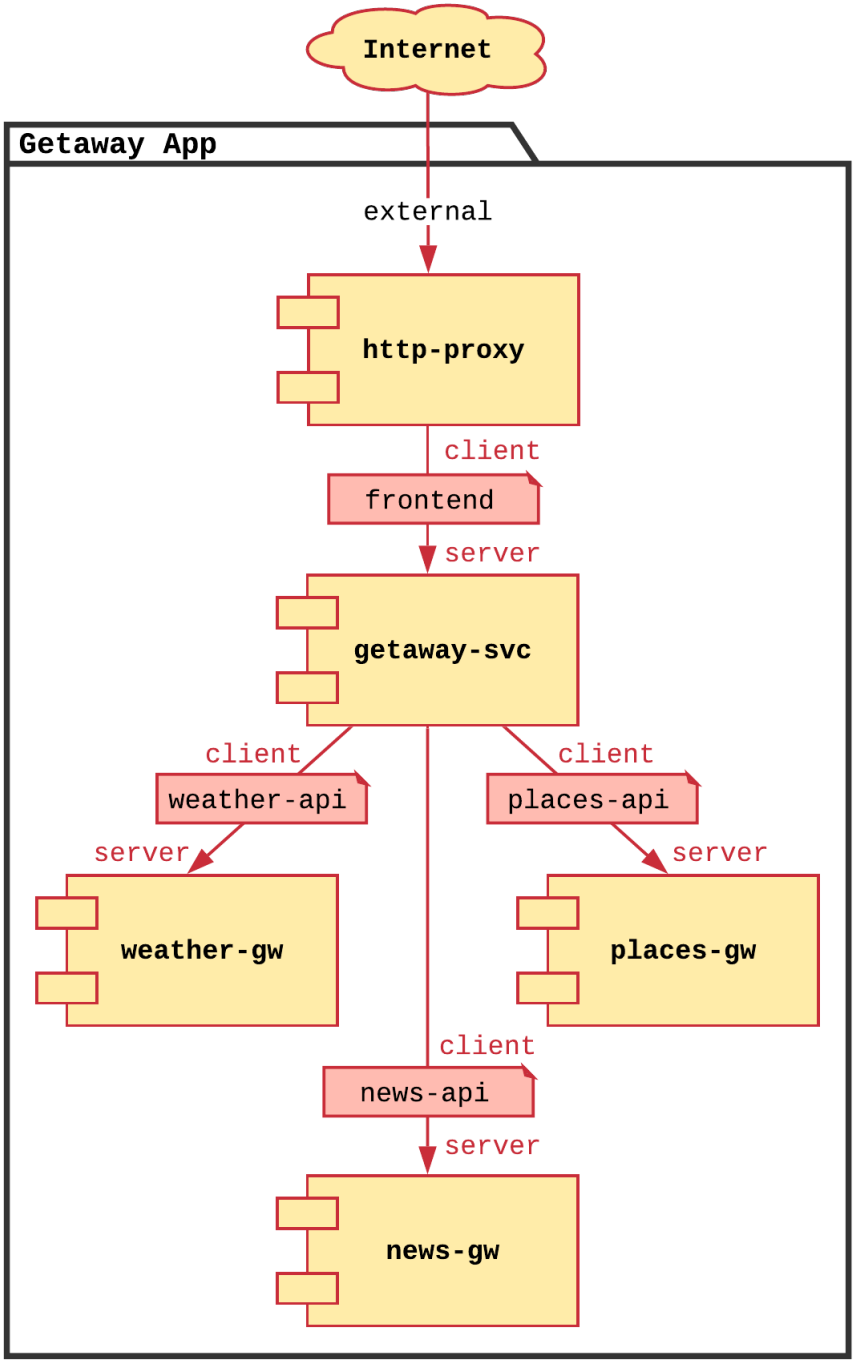


Fig. 13.9: Getaway Application Service Graph

A contract has pre-defined roles: you can think of these as being an endpoint of an edge eminating from the contract waypoint. In its simplest form, a role might be client or server. So the policy inherently embedded within the contract might allow, simply, that the client can initiate communication with the server, but not the other way around.

Services in the Service Interconnection Fabric

A developer's application microservice maps to an SIF service (the node in the application's service graph). When creating an SIF service on the orchestrator, the developer may assign a role from a contract to the service. This assignment defines how the service may communicate.

The developer permits a service in a container or on a VM to communicate with other services by creating a service token on the SIF orchestrator and then installing that service token directly on the VM or via a container orchestration platform such as Kubernetes. That token authenticates the service and authorizes the communication defined by the contract role assigned to that service.

Essentially, defining the policy for your application, then, only requires you to define the service graph: the microservices (SIF services) and how they communicate (SIF contracts).

13.3.3 Fabric Manager Tool: bwctl-api

In *Application Infrastructure* you were introduced to the **bwctl** command-line tool that allows you to interact with public cloud providers to create virtual private clouds and virtual machines. In this section, you will use another fabric manager command-line tool, **bwctl-api**, that allows you to interact directly with the SFI orchestrator. Actions that can be performed by clicking around on the GUI can also be performed by scripting them through **bwctl-api**.

First, you will use **bwctl-api** to generate Getaway's service graph (contracts and services) and, second, you will use **bwctl-api** to generate service tokens that authorize Getaway's microservices to operate on the SIF infrastructure.

Generating the Service Graph

Before we begin, let's confirm that there are, indeed, no contracts and services pre-installed on the orchestrator. Back on the orchestrator GUI open in your browser, click on the *Services* button in the sidebar navigation menu. It should look similar to Fig. 13.10.

You may click on the *Contracts* button in the sidebar navigation menu to confirm that there are no contracts, as well.

Just as you used a YAML description of infrastructure components with **bwctl** in the last section, here you will use a YAML description of Getaway's service graph with **bwctl-api**.

Back in the terminal window that has an SSH session open on your FM node, take a look at the service graph's YAML description located in the **~ubuntu** home directory. Recall that you can do this using Linux **cat** command

```
]$ cat getaway-app.yml
```

As you navigate through the text (scrolling up, if required), notice that the YAML spells out requirements for generating a domain (**getaway-app**), four contracts (**frontend**, **weather-api**, **places-api**, and **news-api**), and five services (**http-proxy**, **getaway-svc**, **weather-gw**, **places-gw**, and **news-gw**). If you look carefully, you'll see that contracts and services are defined within a given domain and that services are assigned a particular role within a contract. In fact, the **getaway-svc** service, given its place in Getaway's service graph, takes on four contract roles.

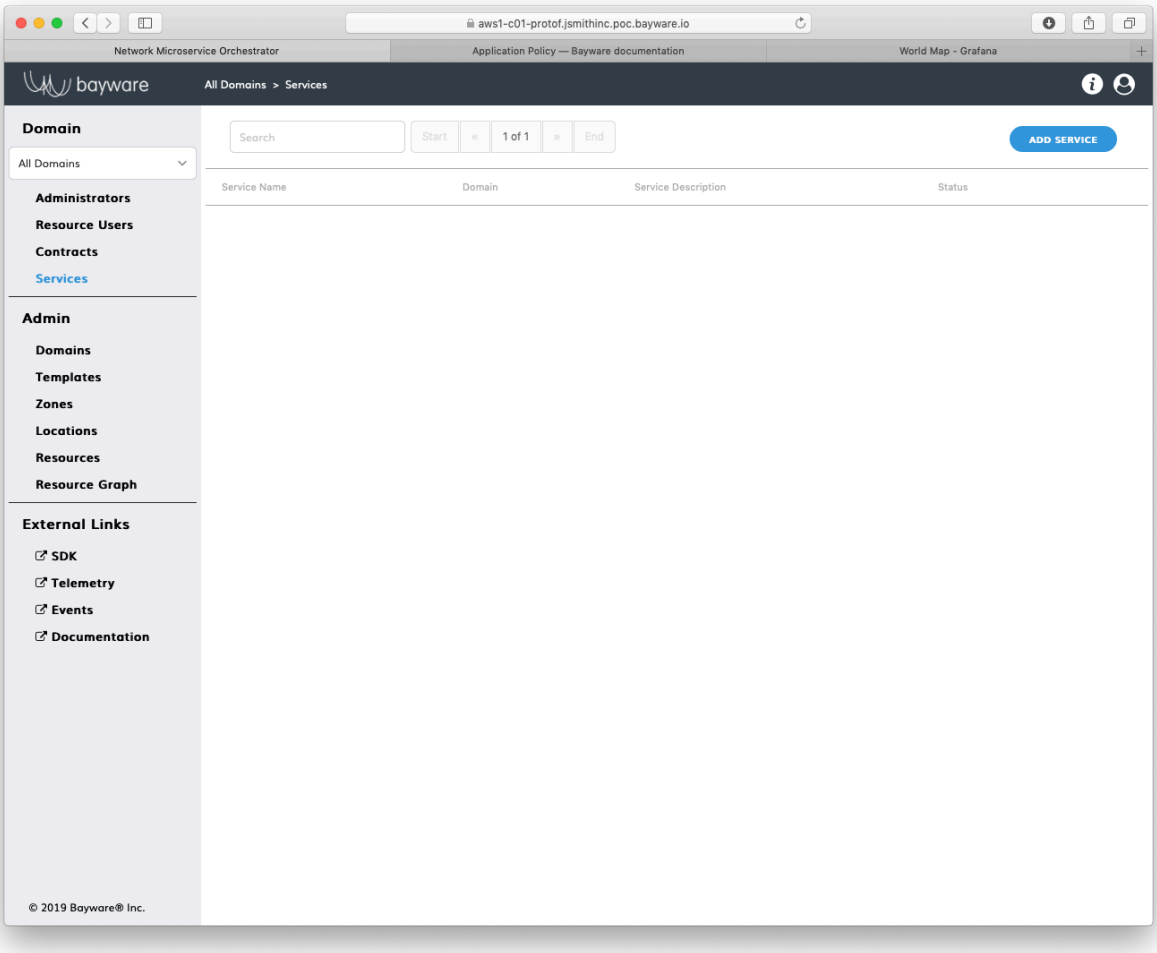


Fig. 13.10: No Services Installed

Now back at the Linux prompt, execute the batch file by typing

```
]$ bwctl-api create batch getaway-app.yml
```

This should complete pretty quickly. Once it does, return to the orchestrator GUI in your browser and reload it. You can do this in most web browsers by clicking the *reload* button in the address bar or by typing **+R** on a Mac keyboard or **CTRL+F5** on a Windows keyboard.

If your orchestrator browser window is still showing the *Services* or *Contracts* page, it should now be populated.

With services and contracts populated, you can now see the service graph. In the navigation menu, click on *Domains* under *Admin*. You should now see a domain called `getaway-app`. Click it.

On the *Domain* page for `getaway-app`, click the *View Graph* button at the bottom. *Voilà!* There is the service graph for Getaway with all its contracts and services depicted (see Fig. 13.11).

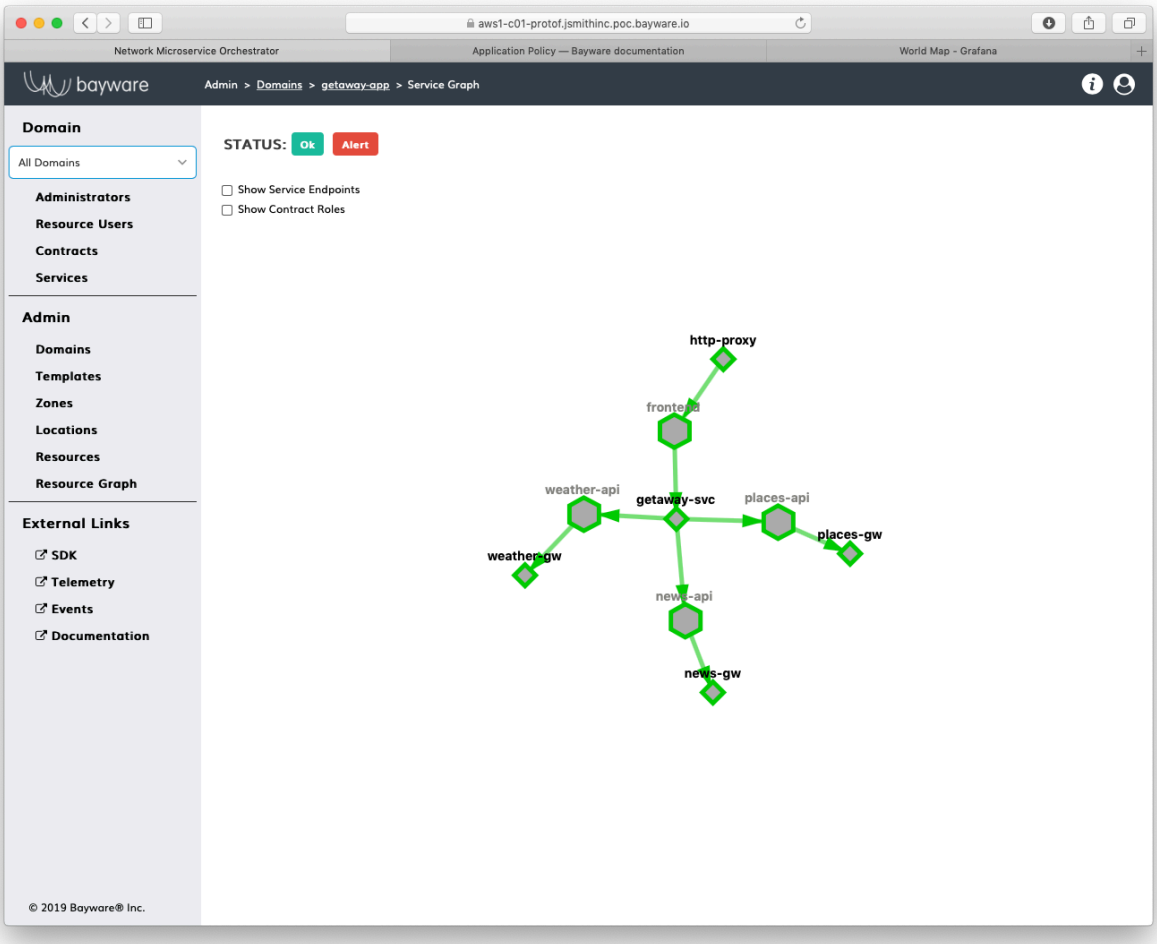


Fig. 13.11: Getaway Service Graph on the Orchestrator

Generating the Service Tokens

Getting back to a (somewhat) more physical world, although the policy is now in place to allow the microservices to communicate (`http-proxy` can request data from `getaway-svc`; `getaway-svc` can request data from the three back-end services, `weather-gw`, `places-gw`, and `news-gw`), they have not yet been authorized to communicate within a workload i.e., a container or a virtual machine.

That's where tokens come in. For a service in a given domain, you can ask the orchestrator to generate a token that can be used to authorize the service to operate. Once you have the token, you, as the devOps engineer, can use it in whichever workload you choose. You will see this in the next section, *Application Microservices*, when you install the tokens. For now, you simply generate the tokens.

As with all things related to the orchestrator, you could generate service tokens using the orchestrator GUI or using the **`bwctl-api`** tool. Here you will use the latter.

A YAML file in the `~ubuntu` home directory describes the service token request. You can explore it by using **`cat`**

```
]$ cat getaway-tokens.yml
```

Notice that the YAML description requests multiple tokens for the three back-end services, `weather-gw`, `places-gw`, and `news-gw`. Each of these services runs both in GCP and in Azure. By creating a unique token for each running service instance, you can control authorization independently in each cloud. This will come in handy later on when you delete a token for `news-gw` in one cloud and see that the same microservice operating in the other cloud automatically takes over.

Now execute the command to get the tokens from the orchestrator. Here you will redirect Linux `stdout` to a file where the tokens will be saved.

```
]$ bwctl-api create batch getaway-tokens.yml > tokens.yml
```

This should be quick. Once it has completed, use **`cat`** to explore the file containing the service tokens returned by the orchestrator

```
]$ cat tokens.yml
```

The orchestrator has returned a YAML sequence of eight tokens, each associated with a particular service within a given domain in a given cloud, as prescribed by the requesting YAML file. You will use these in the next section to create service endpoints that get Getaway's microservices up and running and communicating with each other.

13.3.4 Summary

In this section you learned about the relationship between an application's service graph and SIF services and contracts to inject policy between microservices. Services map to service graph nodes; contracts map to service graph edges. You created the service graph for Getaway and then asked the orchestrator to generate tokens that you will use in the next section to create service endpoints. All this was done using your fabric manager's command-line tool, **`bwctl-api`**, that allows orchestrator GUI operations to be scripted through its API.

Next up: You will install service tokens that authorize Getaway's microservices to operate over the service interconnection fabric.

13.4 Application Microservices

13.4.1 Requirements

Getaway's first two tiers should be deployed in AWS and the back-end data tier should be deployed in a geo-redundant matter, in this case, both in GCP and Azure as shown in Fig. 13.12.

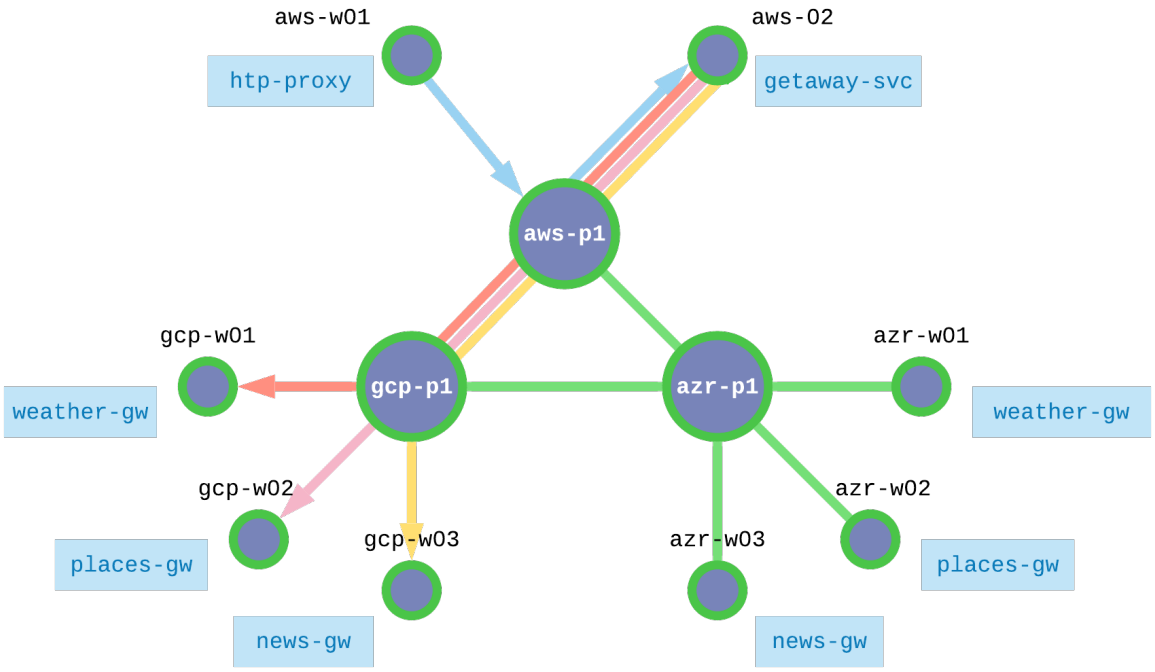


Fig. 13.12: Geo-redundant Application

Each workload node should be authorized to operate within the service interconnection fabric and the microservices should be installed and started.

13.4.2 Authorized Microservices

The service interconnection fabric authorizes workloads (VMs or containers) to operate using tokens. Authorization tokens for a given service may be generated using the orchestrator GUI or the orchestrator API, as you did in the last section. The tokens must then be installed on each workload before that workload can communicate over the fabric.

When operating in a VM environment, the user configures the agent running on a workload with the required token. This can be done manually or using orchestration software, such as Ansible.

When operating in a container environment, a call from kublete to the CNI triggers the SIF plugin to request a token from the Kubernetes server, which has them stored as pod annotations.

13.4.3 Workload Orchestration

Recall that in *Application Infrastructure* you were introduced to the fabric manager's **bwctl** command-line tool in order to interact with public cloud services and create the infrasturcture for Getaway. In *Application Policy* you were introduced to FM's **bwctl-api** command-line tool in order to interact with the orchestrator to create policy and authorization for Getaway.

The success in this section, however, is up to the devOps engineer. It's up to him or her to decide how and where to deploy microservices and then use the service tokens to authorize the chosen containers or virtual machines to communicate over the fabric. This would typically be done with a workload orchestrator, such as Kubernetes for containers or Ansible for VMs.

Our example application, Getaway, runs on virtual machines. You will use an Ansible playbook to interact with the workloads.

The Ansible playbook needs to do two things

1. configure the workload Agent to use a service token
2. install and start Getaway's microservices

These functions have been wrapped into a single playbook.

Back in your terminal session on FM, ensure that you are in th **~ubuntu** home directory

```
]$ cd
```

Now change into the directory that contains the Ansible playbook

```
]$ cd application
```

Notice that your Linux prompt, which contains the current path, has changed. Previously, it looked similar to

```
]$ ubuntu@jsmith-c0:~$
```

but now it should look more like

```
]$ ubuntu@jsmith-c0:~/application$
```

Execute the following playbook from this directory by typing

```
]$ ansible-playbook deploy-app.yml
```

This playbook shouldn't take longer than a minute or so to complete since the tasks are relatively quick, although it does touch eight workload nodes.

Once the playbook completes, you can continue with the tutorial.

Back at the Ochestrator

Let's ensure that your workloads have been properly authorized at the orchestrator. To do that, we will introduce one more term, *network endpoint*. Fig. 13.13 shows the relationship between network endpoint, service endpoint, and your workloads.

When you used the Ansible playbook above to install a token on a given workload, a network endpoint and a service endpoint were added to that workload. Each service has its own service endpoint. If multiple services exist on a workload, they all communicate through the same network endpoint.

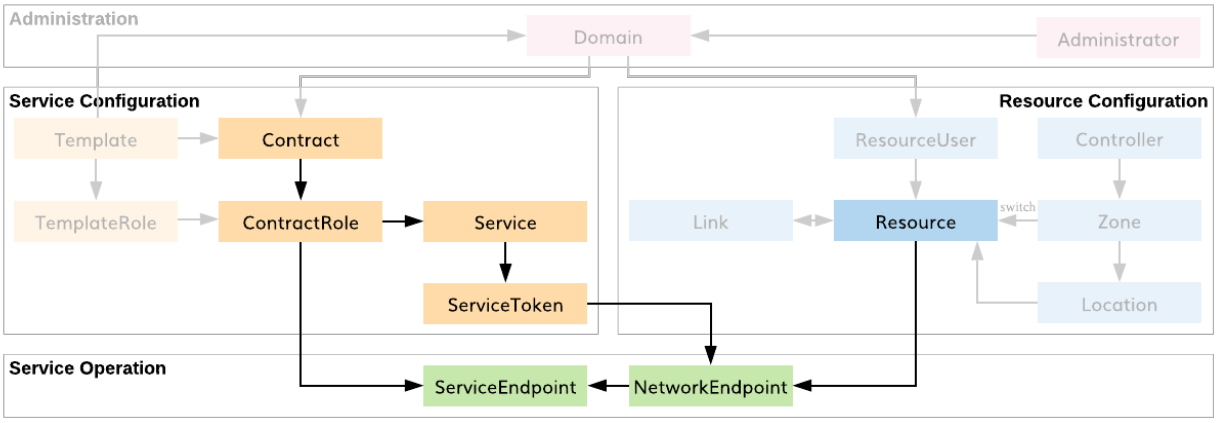


Fig. 13.13: SEs, NEs, & Workloads

Return to the orchestrator GUI open in your browser. Click on *Services* in the sidebar navigation menu and then click on `weather-gw`. Scroll to the bottom. Here you should see two network endpoints, `gcp1-w01-protof` and `azr1-w01-protof`: the *weather* service has been installed and authorized on one node in GCP and one node in Azure. This is highlighted in Fig. 13.14.

Recall from the Getaway *service graph* that `getaway-svc` service communicates with four other services: `http-proxy`, `weather-gw`, `news-gw`, and `places-gw`. Click on *Services* again, but this time select `getaway-svc`. At the bottom of the page you should see a single network endpoint: `getaway-svc` is running on `aws2-w02-protof` workload. Click *NE Name*. A pop-up dialog box shows the service endpoints communicating through this network endpoint. There are four.

Getaway Application

Finally, click on the *Getaway* button on the SIS attachment that was included in your welcome email to launch Getaway in a browser window. You should see weather, places, and news for the city selected at the top of the app. Each pane in the browser displays the node on which it is running. The geo-redundant microservices should all be running in Azure at this point in the tutorial. See Fig. 13.16 below.

13.4.4 Summary

In this section you learned how application microservices are authorized to use the service interconnection fabric to communicate with policy dictated by the user. Tokens created in the last section are installed on VMs or containers with the help of commonly-used workload orchestration software, such as Ansible and Kubernetes. You learned how SIF network and service endpoints are related to authorization tokens and where these are displayed in the orchestartor. Finally, you saw and interacted with Getaway running in your browser.

Next up: With Getaway installed over a service interrconnection fabric, you will discover some cool features that help ensure a resilient application.

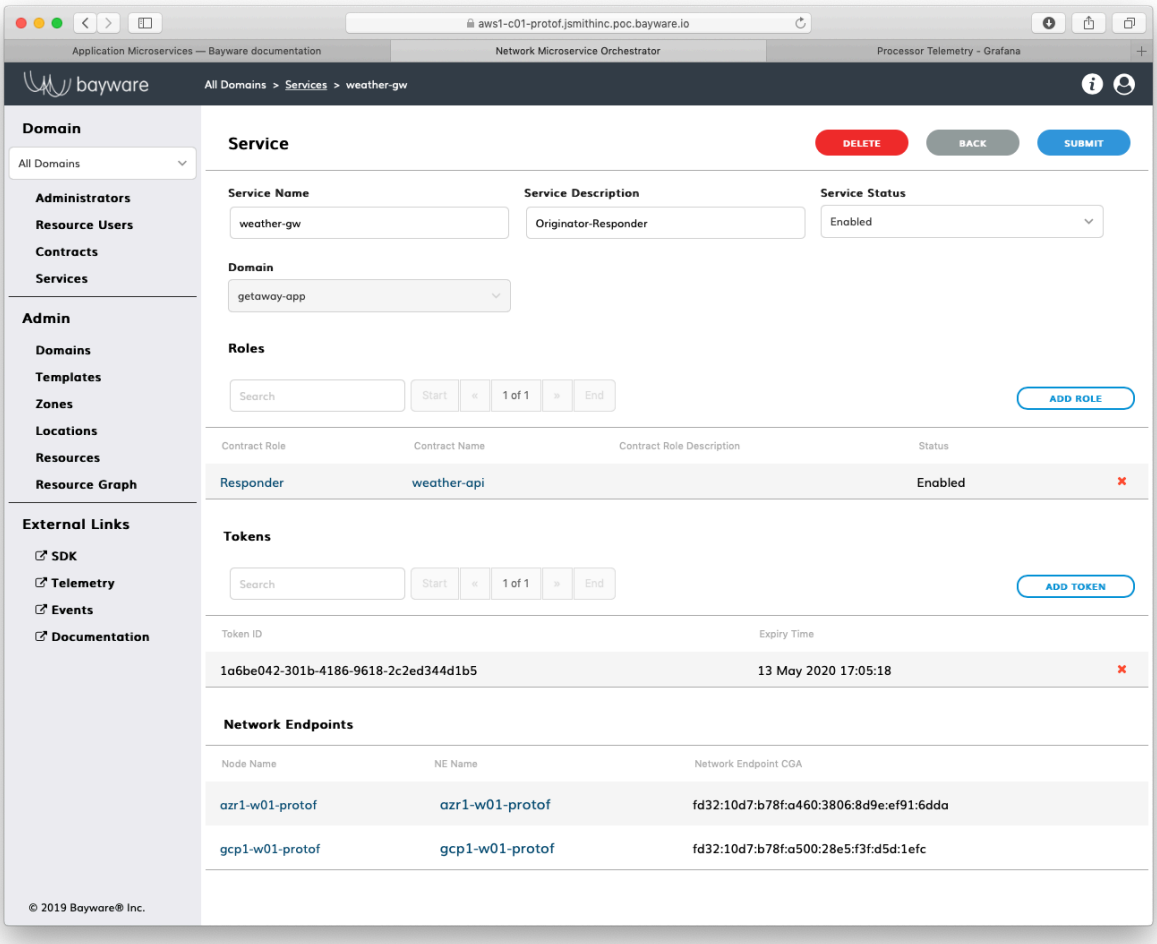


Fig. 13.14: Network Endpoints for weather-gw Service

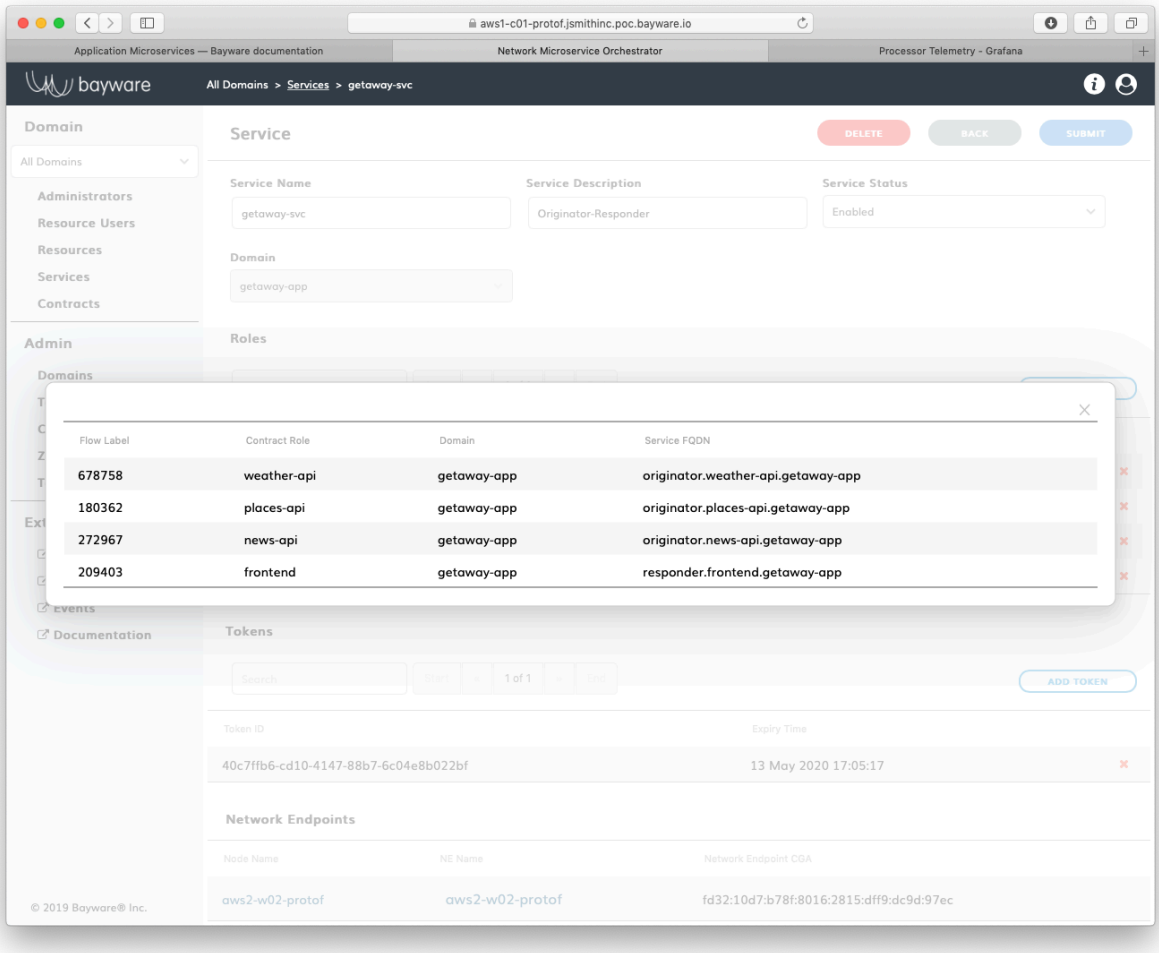


Fig. 13.15: Service Endpoints for getaway-svc

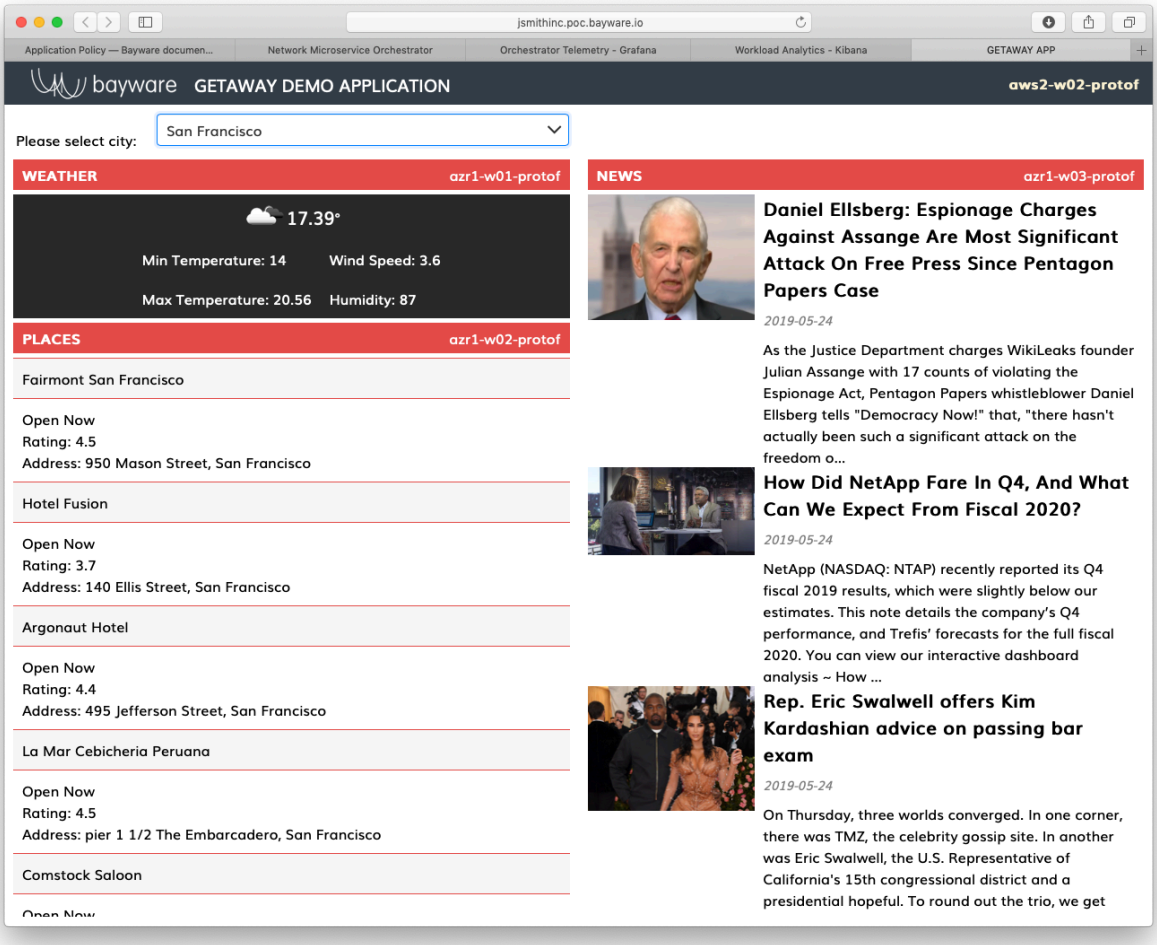


Fig. 13.16: Getaway App

13.5 Feature Showcase

13.5.1 Getaway - A Complete Picture

Now that Getaway is up and running with three back-end services running redundantly in both GCP and Azure, you will walk through a few scenarios that demonstrate how the service interconnection fabric reacts to changing infrastructure conditions.

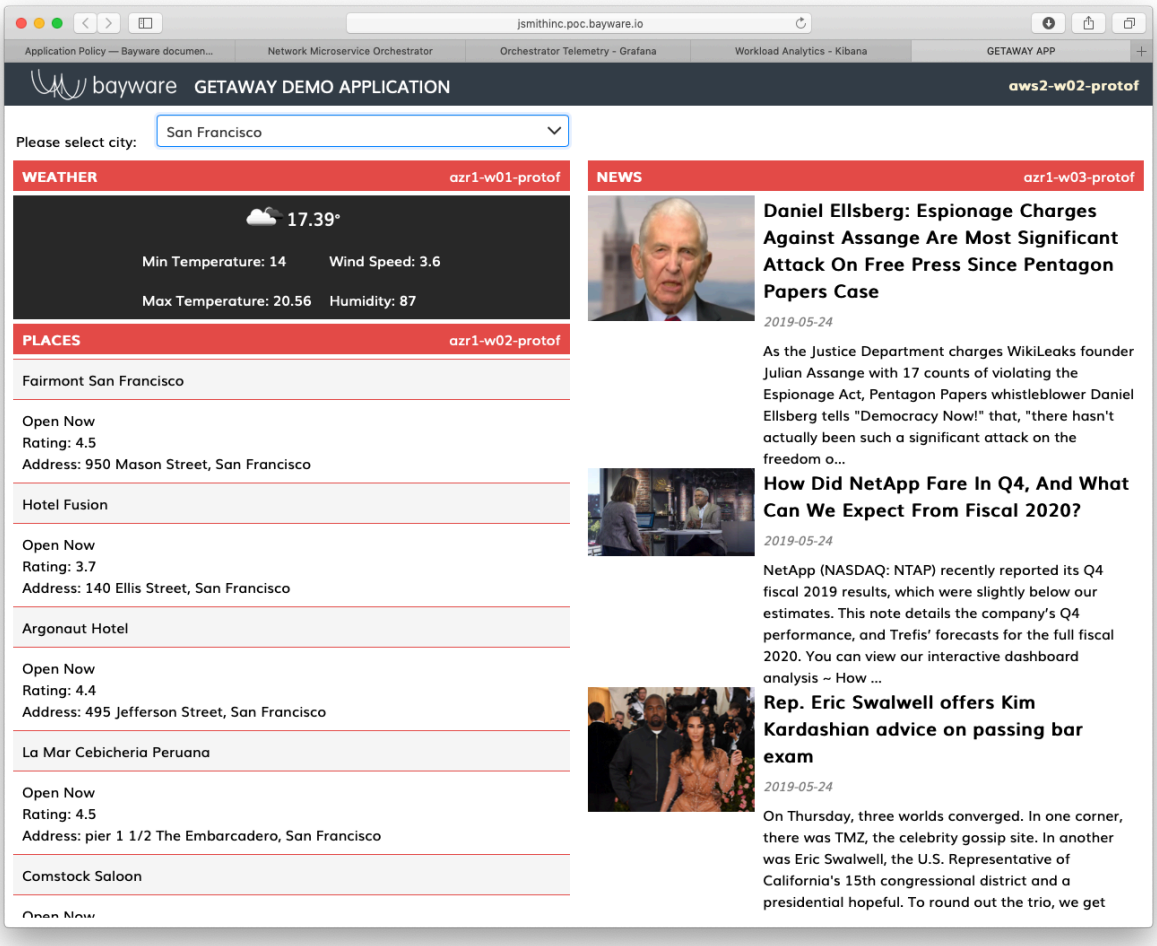


Fig. 13.17: Getaway Application

As you can see from Fig. 13.18, `weather-gw`, `places-gw`, and `news-gw` are each running on a workload node in a VPC called gcp1 and another VPC called azr1: the former represents Google Cloud and the latter represents Azure.

Remember, you can find your service graph by clicking on *Domains* in the sidebar navigation menu and then clicking on *Show Graph* for `getaway-app`.

Going back to your *Resource Graph* panel (choose this from sidebar navigation menu), you should note once again how your three VPCs are interconnected via three SIF processor nodes, one in each VPC.

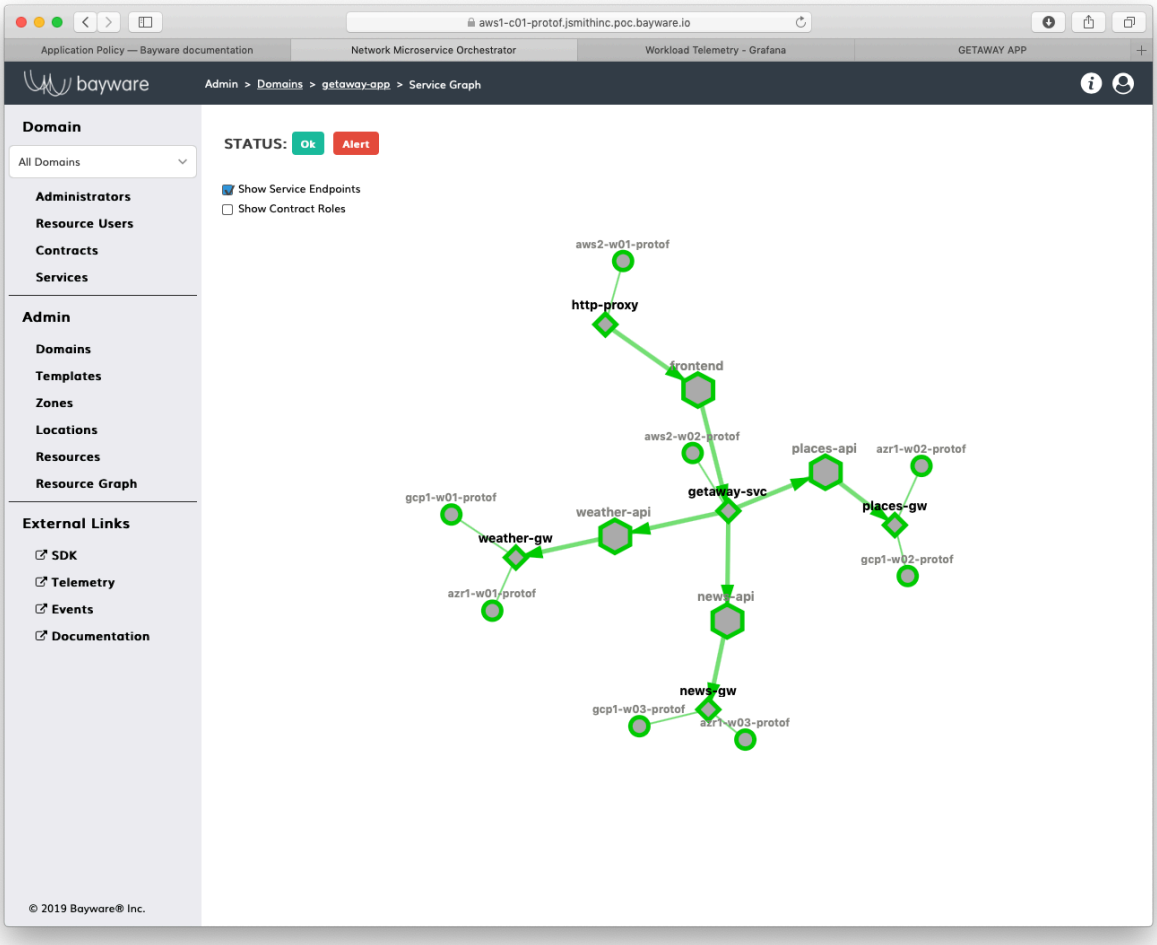


Fig. 13.18: Getaway Service Graph

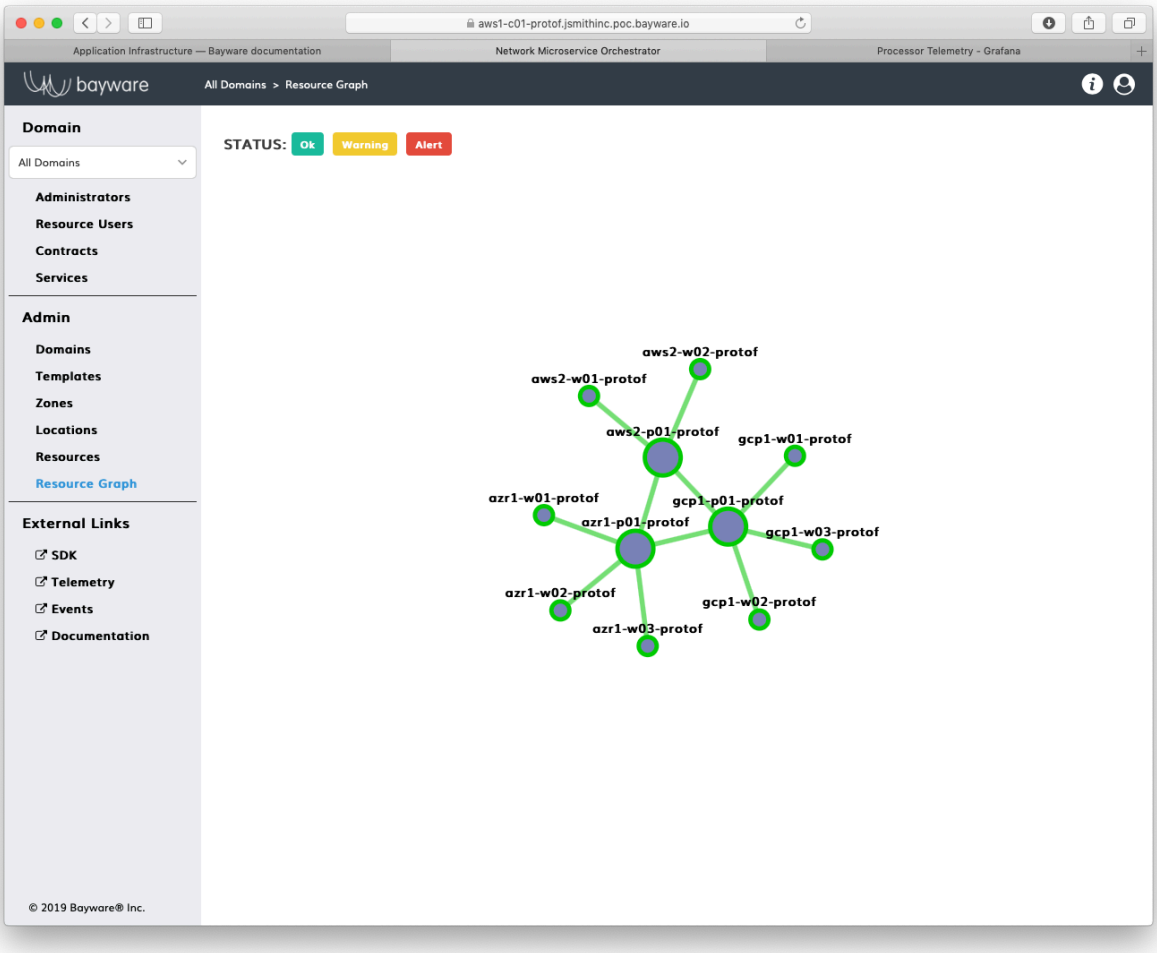


Fig. 13.19: Getaway Resource Graph

13.5.2 Rising Infrastructure Cost

If the cost to service your clients from one VPC becomes too expensive, you might want to switch to a different VPC. In this section you'll simulate what happens if the cost of getting to Azure rises.

To start, notice in Fig. 13.20 that all backend data is currently coming from Azure. You can see the `azr1-` names in the upper-right corner of each microservice panel.

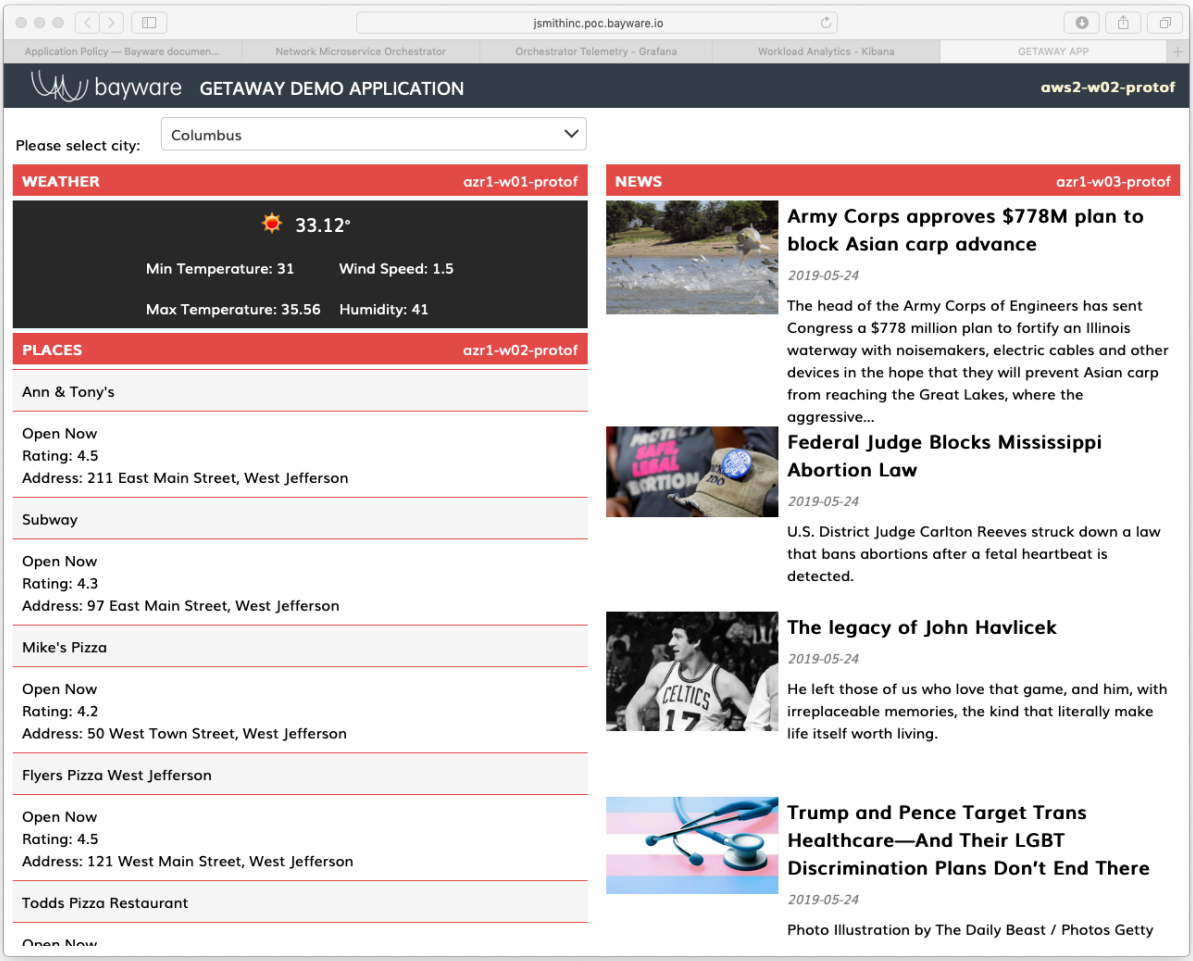


Fig. 13.20: Services From Azure

Back on the *Resource Graph* page in the orchestrator, click on the link between `aws2-p01-protof` and `azr1-p01-protof` i.e., the link that connects the AWS VPC to the Azure VPC. As shown in Fig. 13.21, you can see that the cost is set to 1.

In the overlay panel in the upper-right corner of the *Resource Graph* page, click on the hyperlink to the `azr1-` node. You should see a figure similar to Fig. 13.22. Under the *Links* section at the bottom of the page, find the row associated with `aws2-p01-protof` (remembering that the suffix on your node name will differ) and click on the settings icon (the gear) as shown in Fig. 13.22.

Change the cost to 10, as shown in Fig. 13.23.

Return to your application open in another browser window. You'll see that all Getaway back-end services are now coming from GCP rather than Azure because the cost of getting to Azure became too expensive.

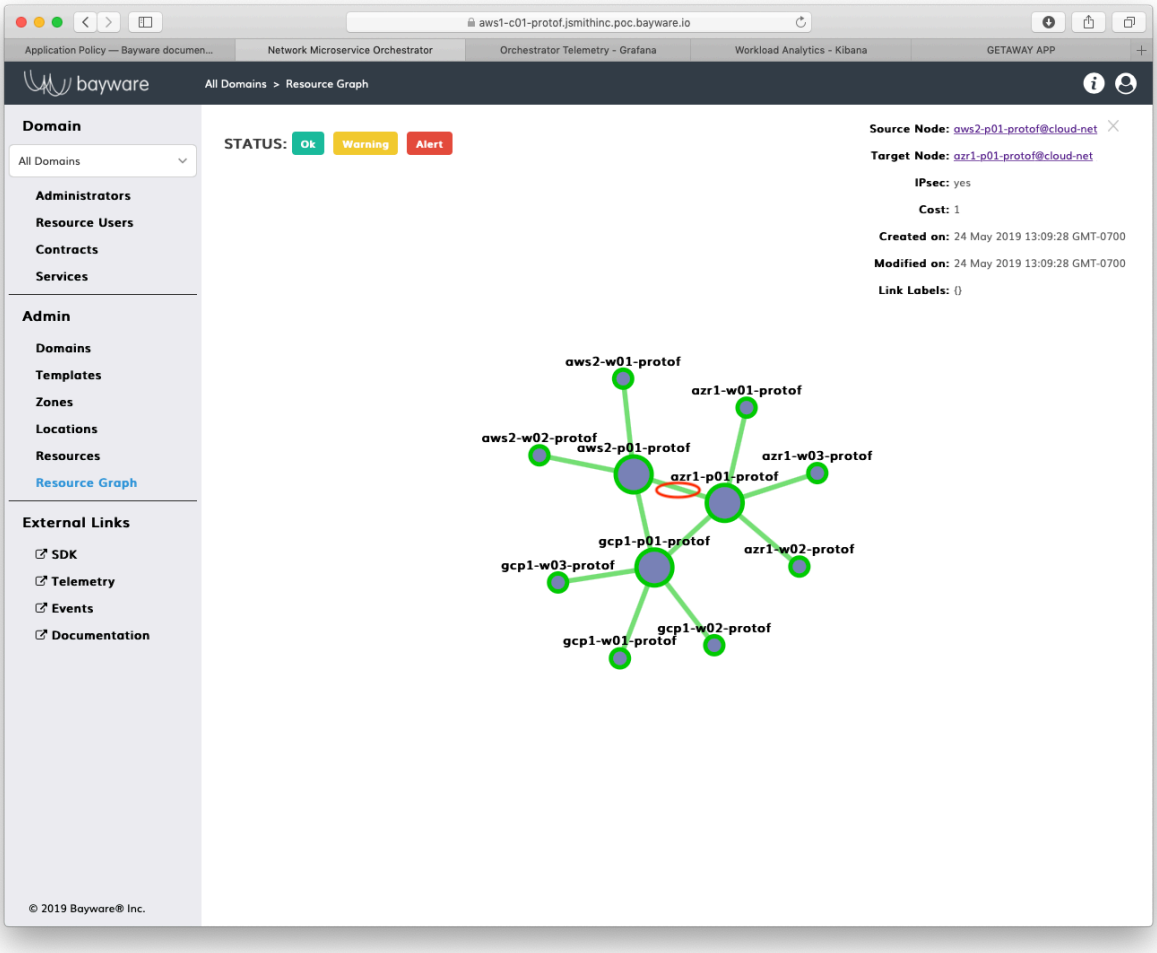


Fig. 13.21: Resource Graph - Link Cost

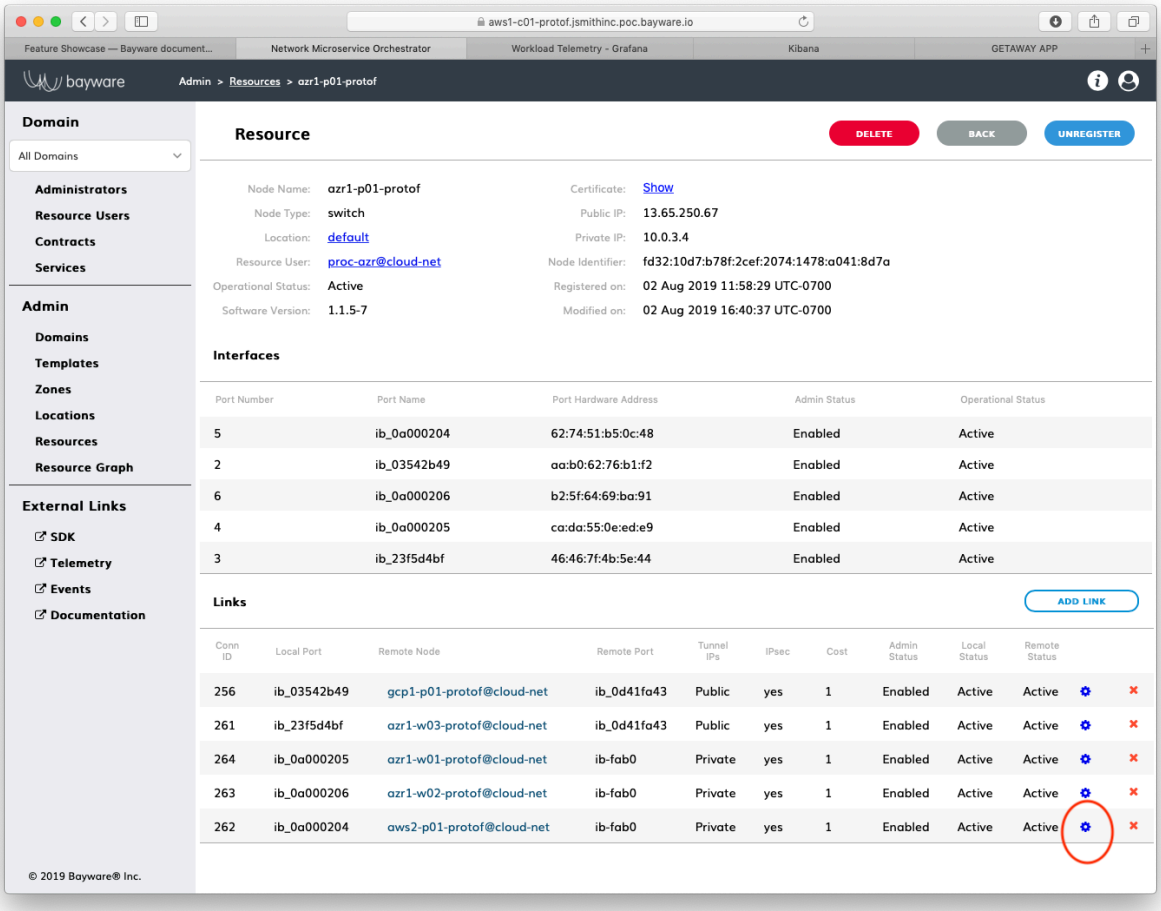


Fig. 13.22: Link Configuration

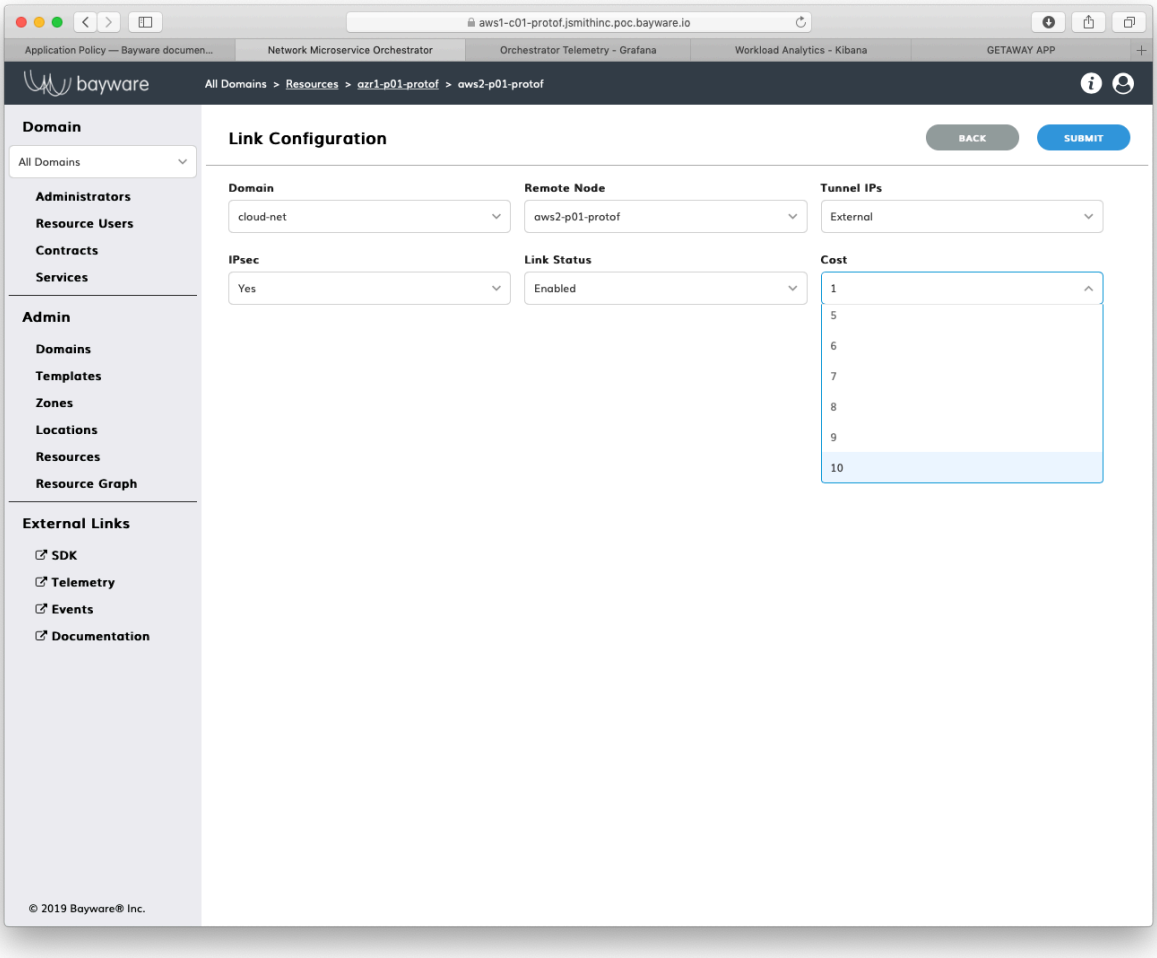


Fig. 13.23: Cost 10

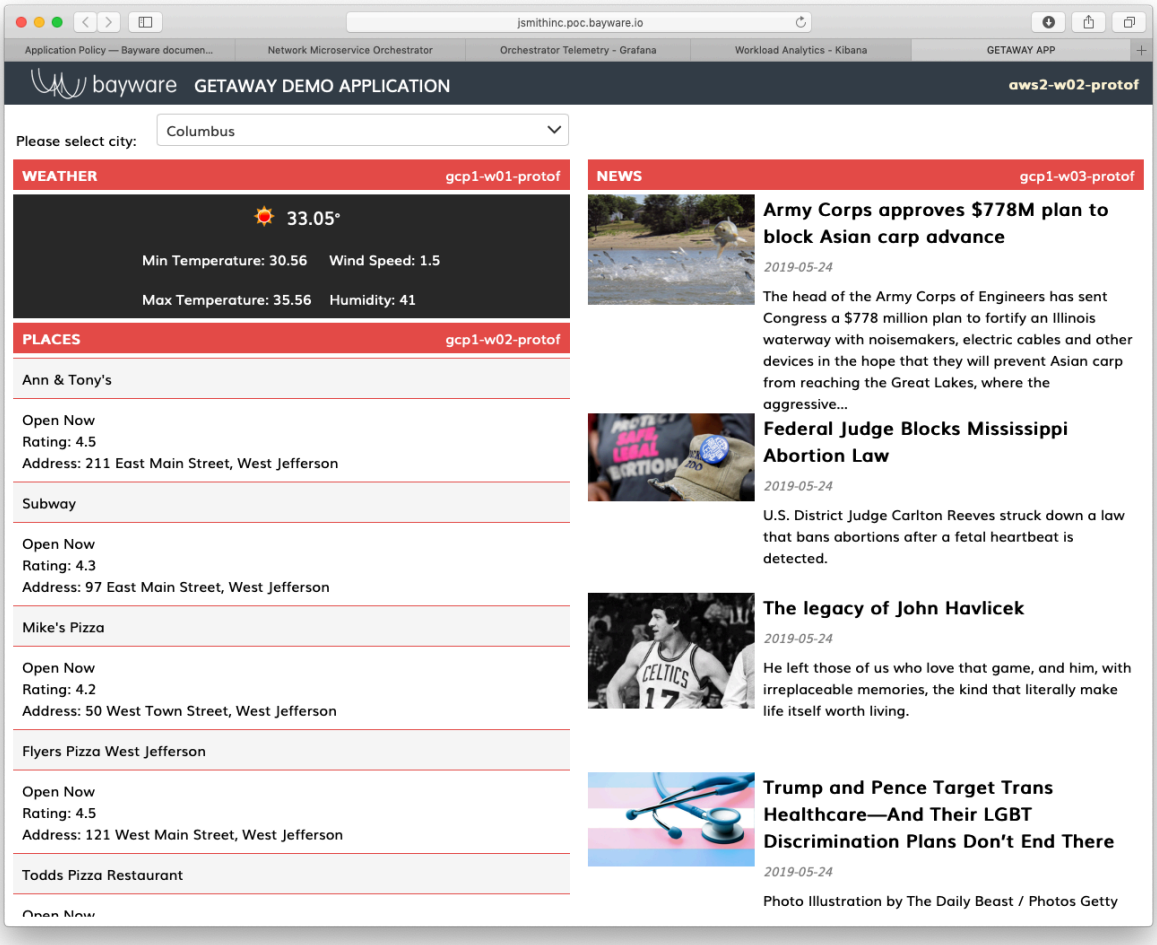


Fig. 13.24: Services From GCP

13.5.3 Revoking Authorization

Recall that you used tokens to authorize your microservices to operate on each workload node. Let's see what happens when you become uneasy about the security of one of your nodes and you want to de-authorize it from operating in your application.

Back in the orchestrator GUI, click on *Services* in the sidebar navigation menu. Then click on `weather-gw`.

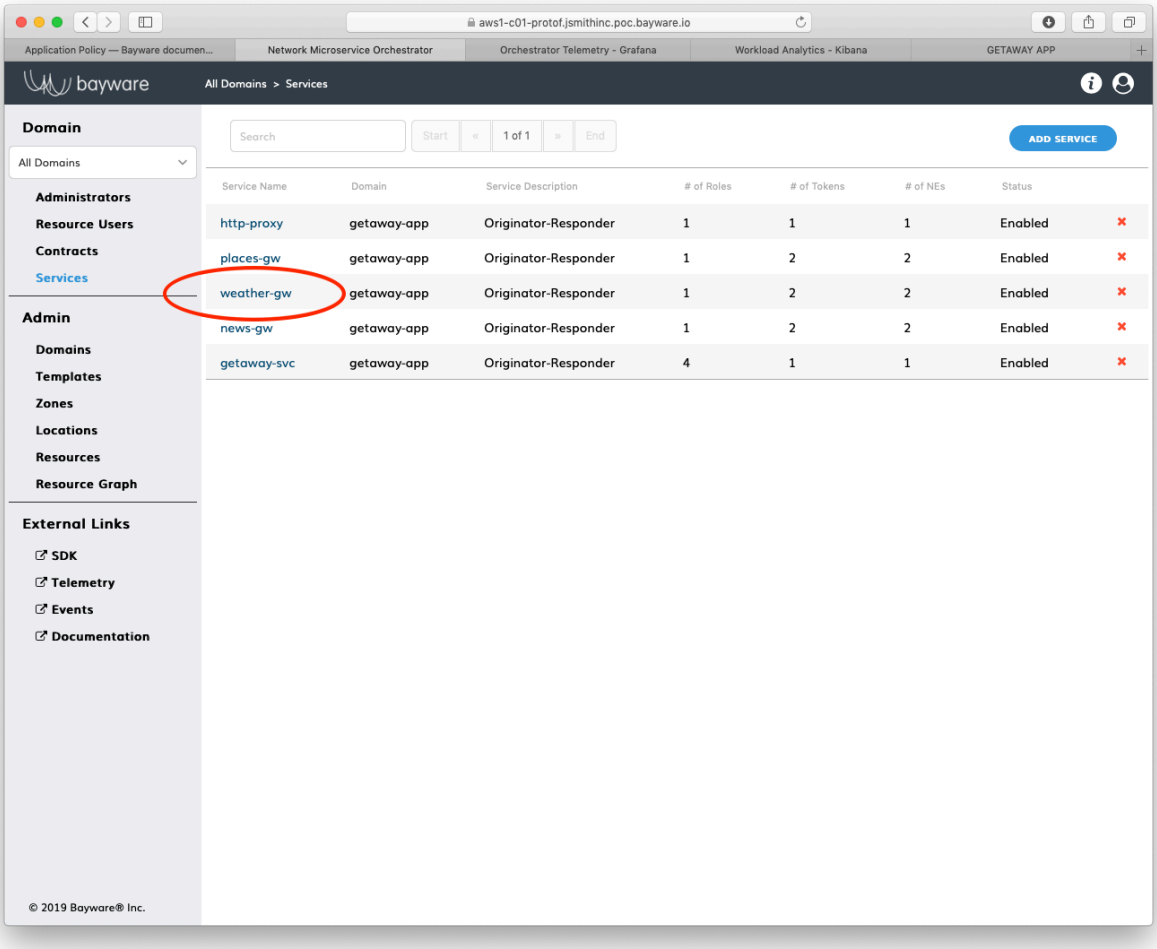


Fig. 13.25: Services

Scrolling down slightly, you should see two entries under the *Tokens* section. (Recall that the `weather-gw` microservice is deployed in both Azure and GCP and that each uses a distinct token.) This should look similar to Fig. 13.26.

Inspect each token by clicking on the hyperlinks under *# of SEs* as shown in Fig. 13.27. Note which token has been utilized in Azure and which has been utilized in GCP.

After you have finished inspecting the tokens, delete the token used to authorize the endpoint in GCP by clicking on the red x on the right as shown in Fig. 13.28.

Return to your application running in your browser and refresh the window. Since you de-authorized `weather-gw` in GCP, it is now being serviced by Azure. (Recall that the upper right corner of each microservice window in the application indicates the node supplying the data.)

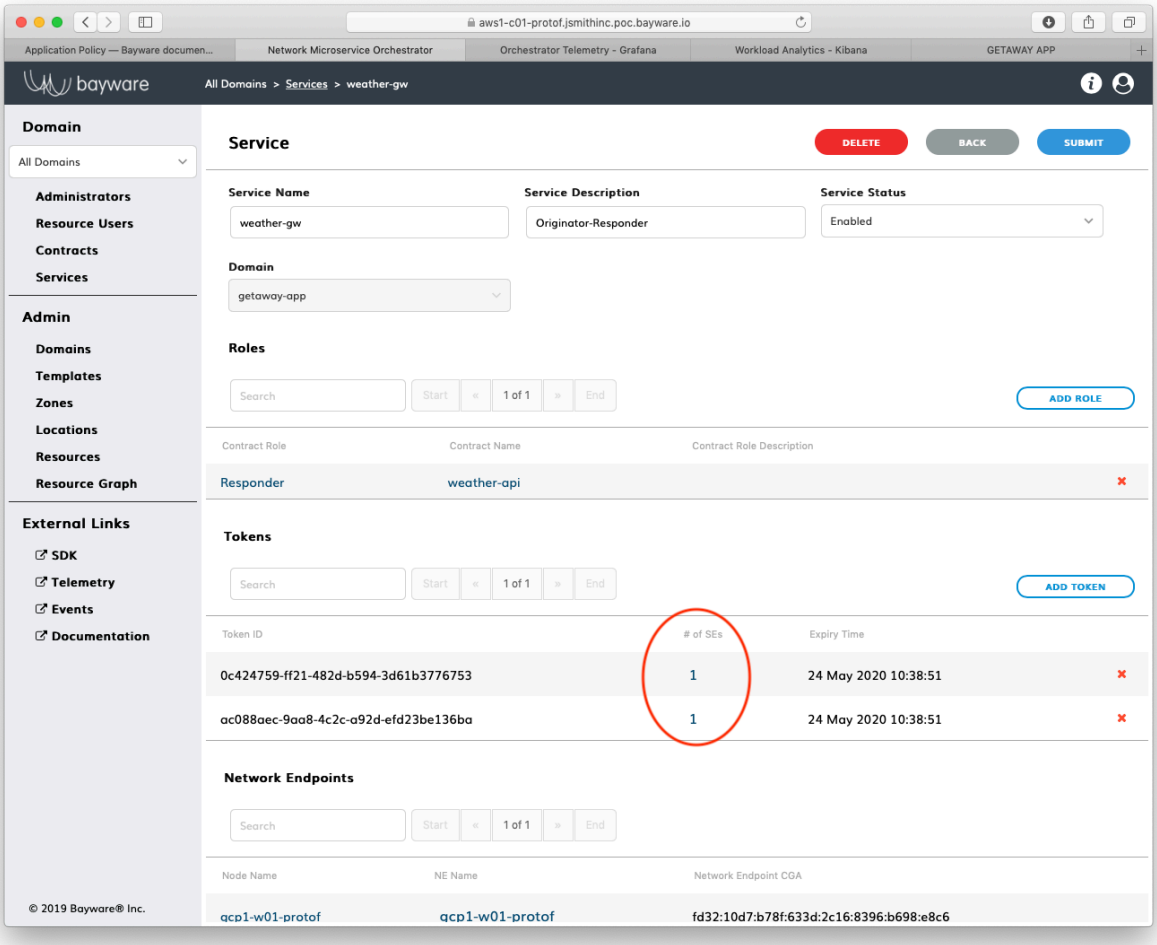


Fig. 13.26: weather-gw Service

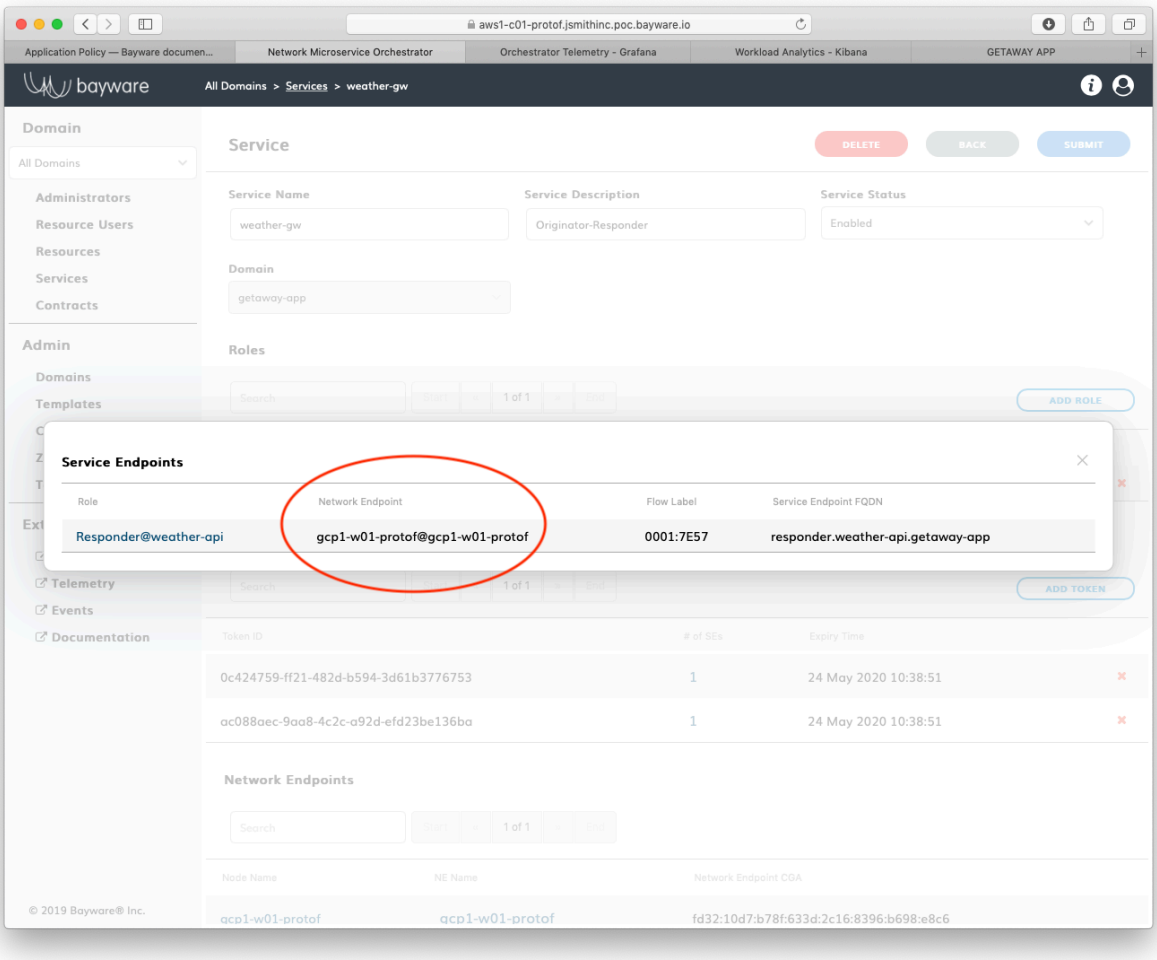


Fig. 13.27: Tokens

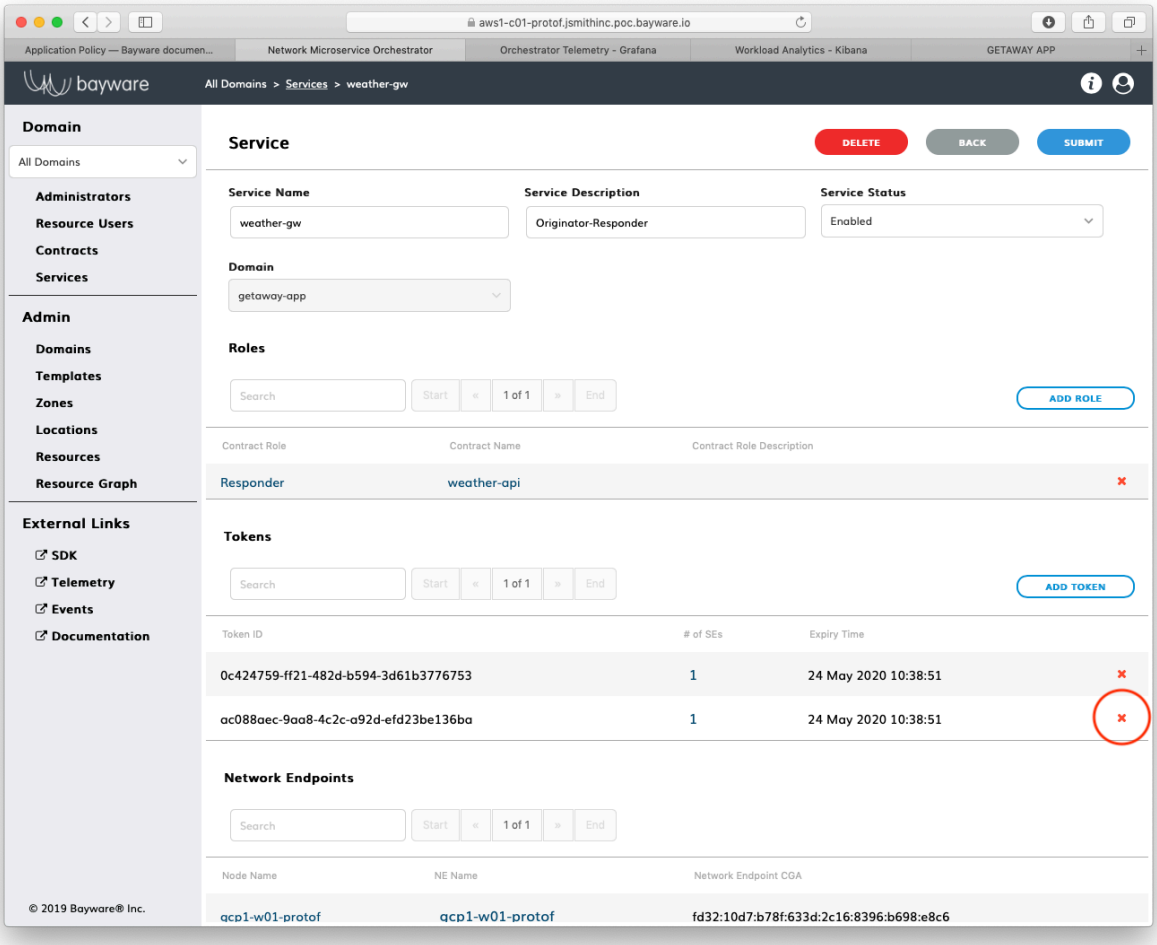


Fig. 13.28: Delete Token

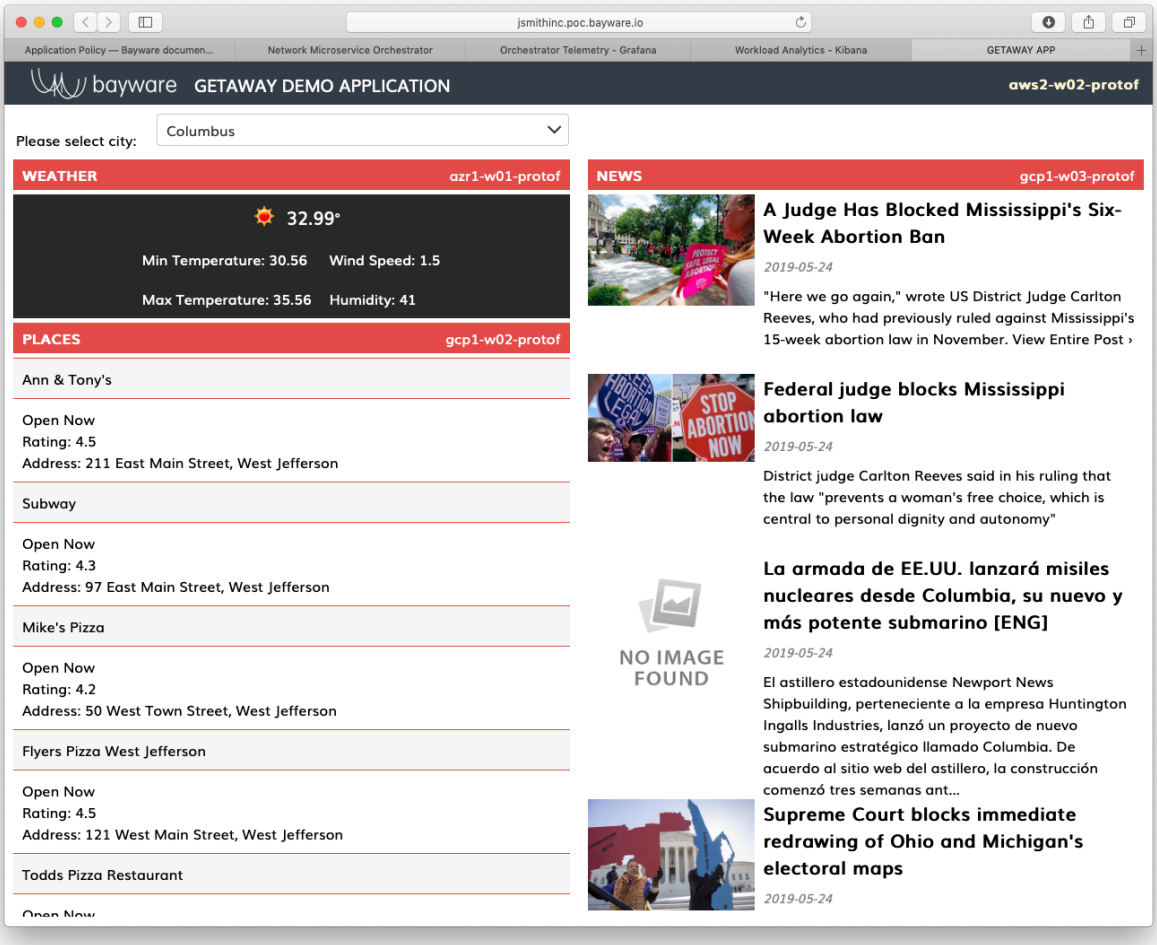


Fig. 13.29: Weather Service From Azure

13.5.4 Failing Workload

Now let's see what happens when you've received one of those familiar emails from your public cloud service provider that explains that one of your VMs has become unreachable because of an underlying problem with the hardware. How does your application respond?

Bring up the *Resource Graph* page in the orchestrator GUI and note that all the circles are green.

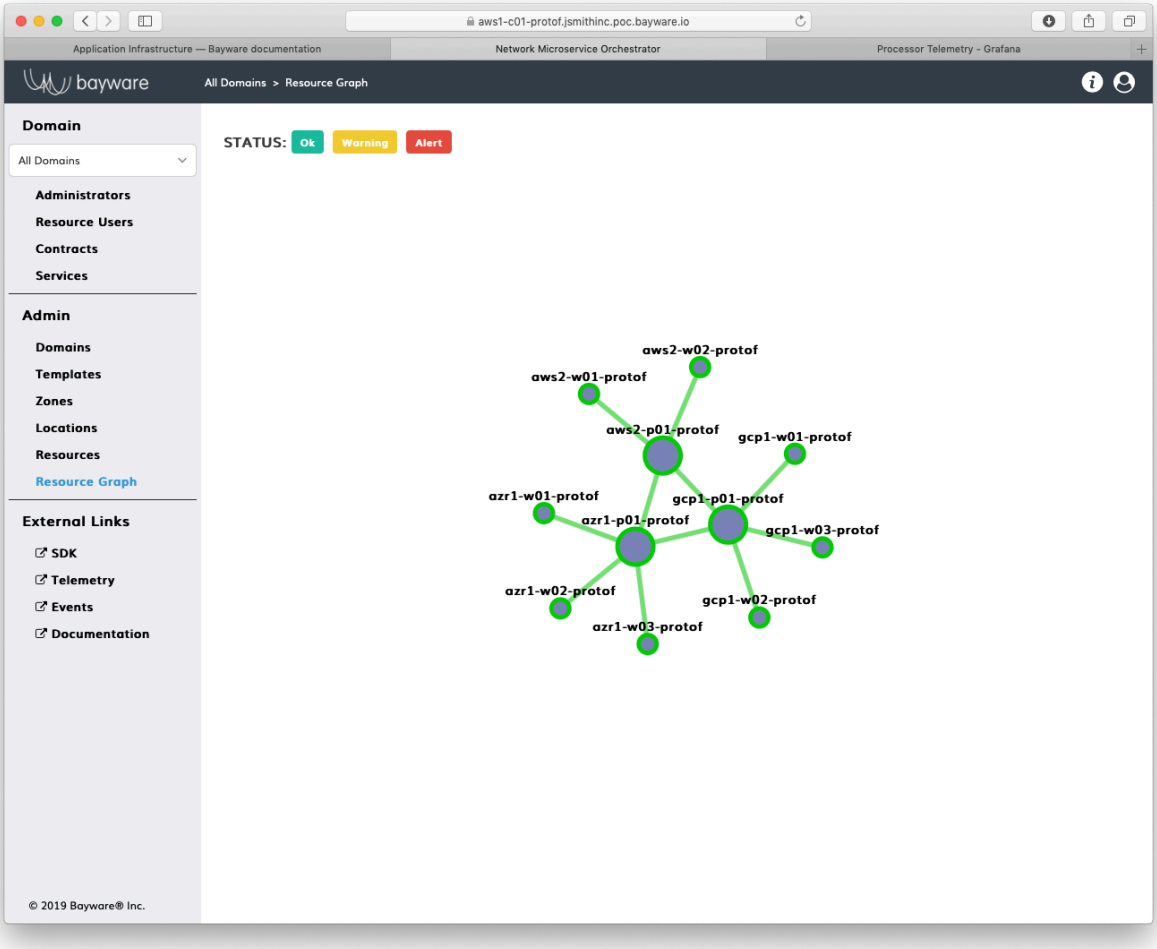


Fig. 13.30: Resource Graph - Everything's Green

Now, some typing. Return to your terminal window with the SSH connection to your fabric manager. Ensure you are in the `~ubuntu` home directory by typing `cd`

```
]$ cd
```

To simulate a node failure, you will stop the policy agent running on `gcp1-w02-`*`protof`*. Type the following at your Linux prompt, ensuring you use your own fabric name, which is the suffix to all the node names displayed in the *Resource Graph* page:

```
]$ bwctl stop workload gcp1-w02-protof
```

After a little chugging, you should see a message similar to

```
Workloads [`gcp1-w02-protof1] stopped successfully
```

Return to the orchestrator *Resource Graph* page where you should see that one of the workload nodes has turned red, as shown in Fig. 13.31.

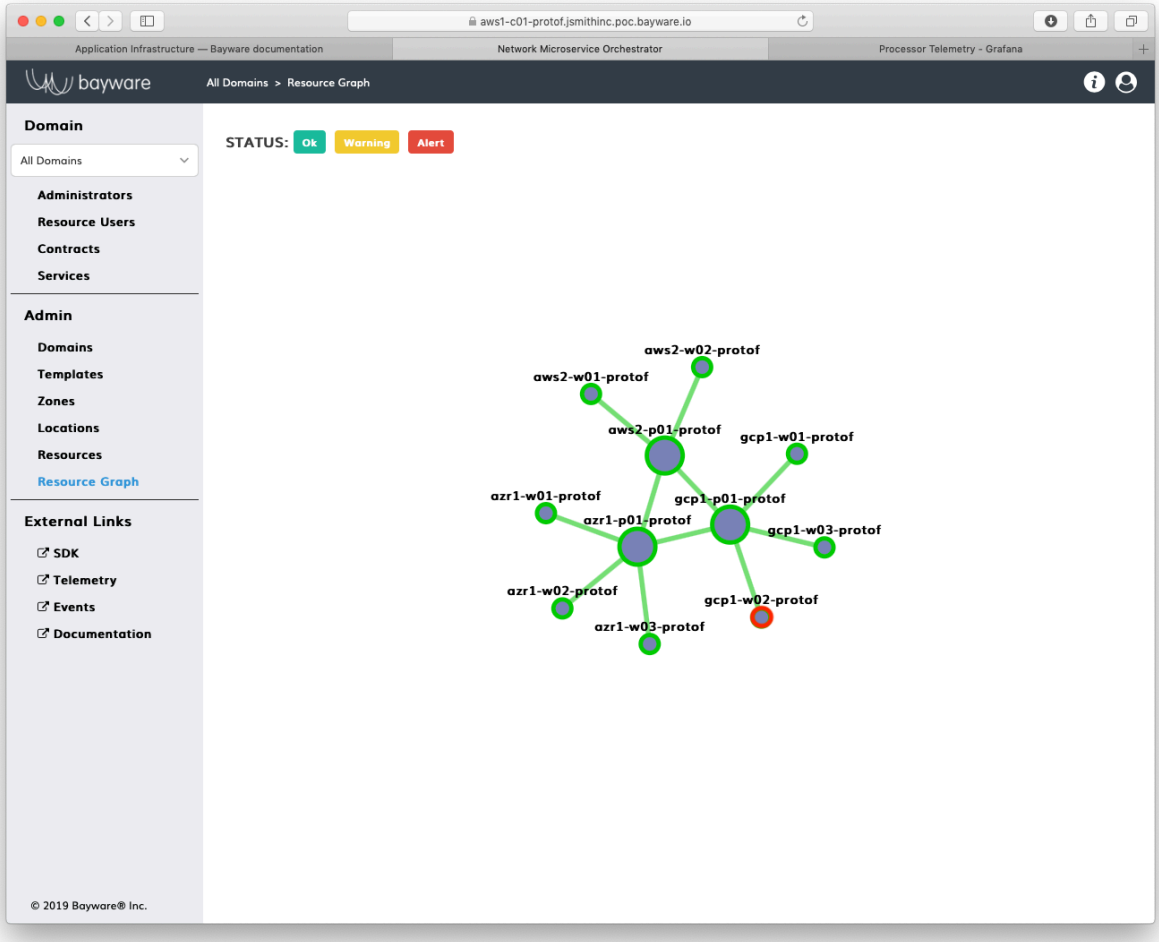


Fig. 13.31: gcp1-w02 Stopped

Back in the Getaway application running in your browser, note that `places-gw` is now serviced from Azure because the corresponding node in GCP went off-line.

13.5.5 Selecting Target

Let's recap where we are at with Getaway. Recall that you started with all back-end microservice data coming from Azure, but after you bumped up the cost of the link between `aws2` and `azr1`, all data started coming from `gcp1`.

You then revoked the token authorizing `weather-gw` in `gcp1` so weather data started coming from `azr1`.

Then you simulated node failure by stopping the policy agent running `places-gw` in `gcp1` so places data started coming from `azr1`.

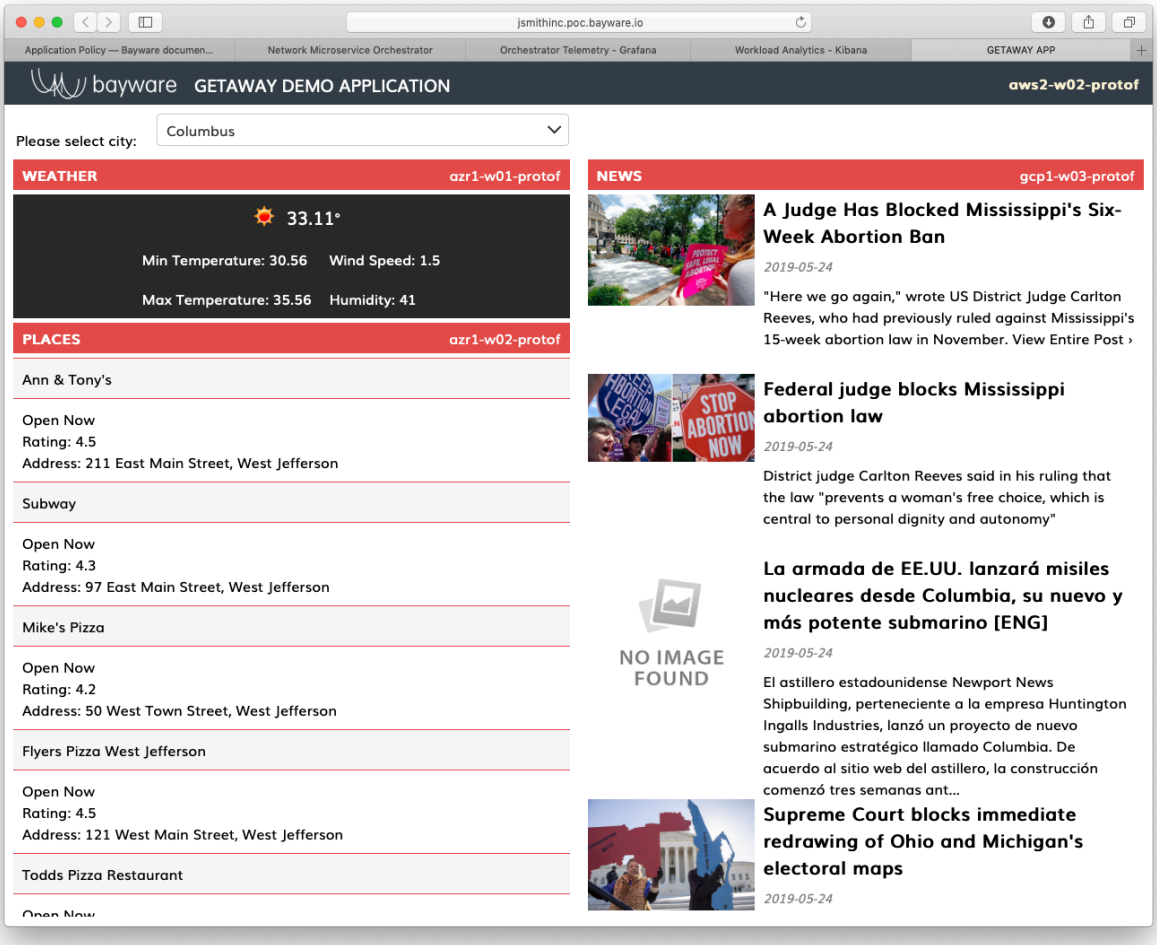


Fig. 13.32: Places Service From Azure

With only `news-gw` supplying data from `gcp1`, suppose the application developer decides he or she wants to prioritize all data coming from Azure without regard to the cost. To do this, the client microservice calls a URL with a more specific FQDN.

Right now, the getaway-service microservice uses

```
http://responder.news-api.getaway-app.ib.loc:8080/
```

You will run a script that changes this to

```
http://azr1.responder.news-api.getaway-app.ib.loc:8080/
```

The more-specific URL tells the DNS resolver to give priority to servers running in Azure when they act as responders in the `news-api` contract.

Back in your terminal window on the fabric manager, ensure you are in the `~ubuntu` home directory by typing

```
]$ cd
```

Now go back into the Ansible application directory by typing

```
]$ cd application
```

Your command-line prompt should now look similar to

```
ubuntu@jsmith-c0:~/application$
```

Execute the playbook that adds the `azr1` prefix to the URL for `news-api` on `aws2-w02-protof` (the VM that runs the getaway-service microservice) by typing

```
]$ ansible-playbook update-app.yml
```

After the playbook completes, refresh Getaway in your browser. You should see that News is now running in `azr1` as show in Fig. 13.33.

13.5.6 Summary

If you find your service graph again by clicking on *Domains* and then *Show Graph* for `getaway-app` and then enable the *Show Service Endpoints* box in the upper-left corner, you'll see how the picture has changed. `weather-gw` and `places-gw` have a single service endpoint each: the GCP endpoints are gone and only the Azure endpoints remain. The `news-gw` service, however, still has two endpoints since Azure was simply given user preference over GCP, rather than revoking authorization or shutting down the node as in the other cases.

13.6 Telemetry

This section provides an overview of some of the telemetry features available using the policy orchestrator. A small description accompanies each of the following figures. After reading through this section, you are encouraged to view the telemetry application in your own sandbox.

From the orchestrator GUI open in your browser, click on *Telemetry* in the sidebar navigation menu. A new window will open in your browser similar to Fig. 13.35.

You can see from the figure that there are four VPCs running in this sandbox:

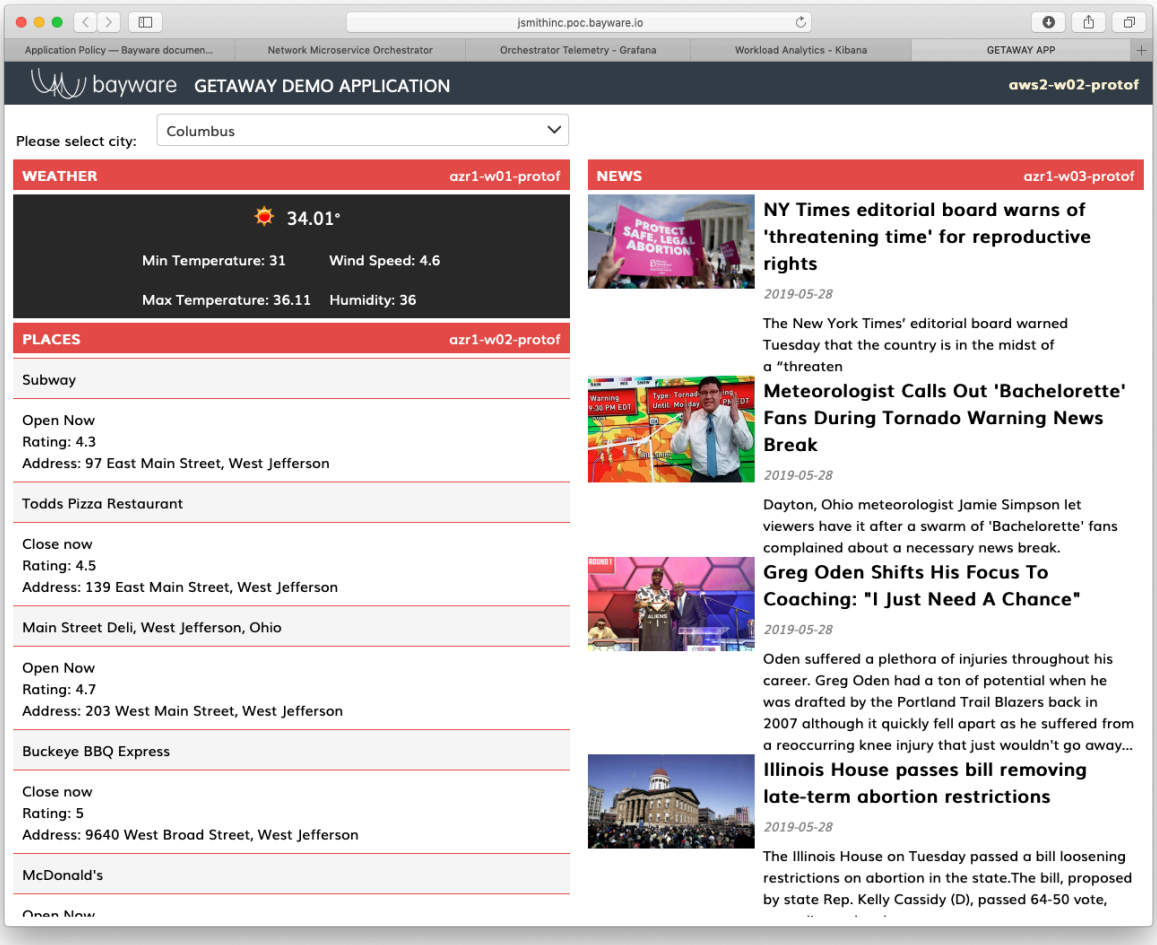


Fig. 13.33: News Service From Azure

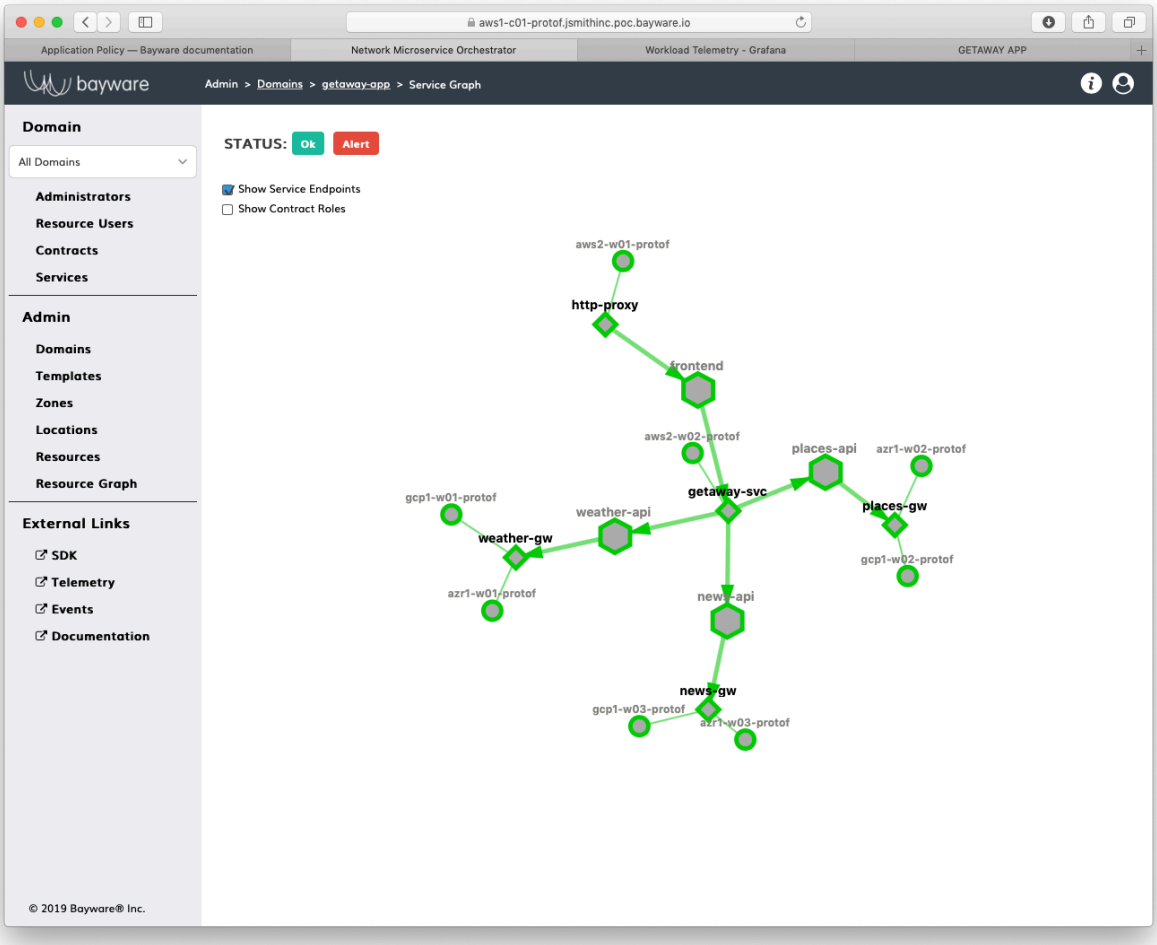


Fig. 13.34: Service Graph Endpoints

- AWS1: the orchestrator nodes operate in an AWS VPC in Northern California
- AWS2: a processor node and two workload nodes operate in a VPC in Northern Virginia
- GCP1: a processor node and three workload nodes operate in a VPC in Northern Virginia
- AZR1: a processor node and three workload nodes operate in a VPC in Texas

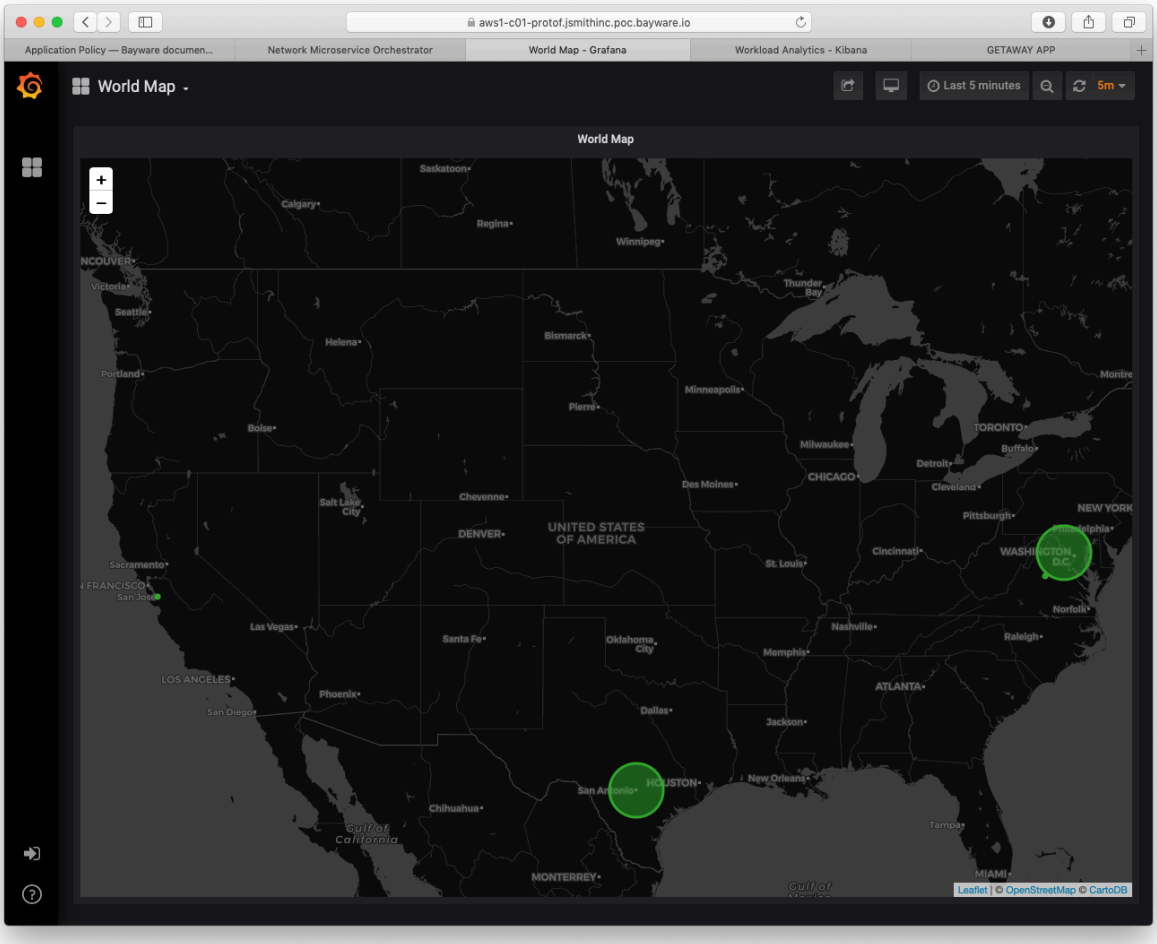


Fig. 13.35: VPC Locations

To navigate between dashboards on the telemetry page, click on *World Map* in the top-left corner. A list of available dashboards appears. Clicking on *Orchestrator Telemetry* brings up a window similar to the one in Fig. 13.36.

You may also find the following telemetry features useful when viewing your dashboards:

- *Server* - view statistics for a given server by clicking on this drop-down menu near the top
- *Last 1 hour* - change the displayed time interval
- **panels** - most panels are initially collapsed and can be expanded by clicking on the panel title

Fig. 13.36 show *Orchestrator Telemetry* for server `aws1-c01-protof`. This server, `c01`, runs the policy controller software. There are two additional servers in the orchestrator. `c02` runs the Telegraf, InfluxDB,

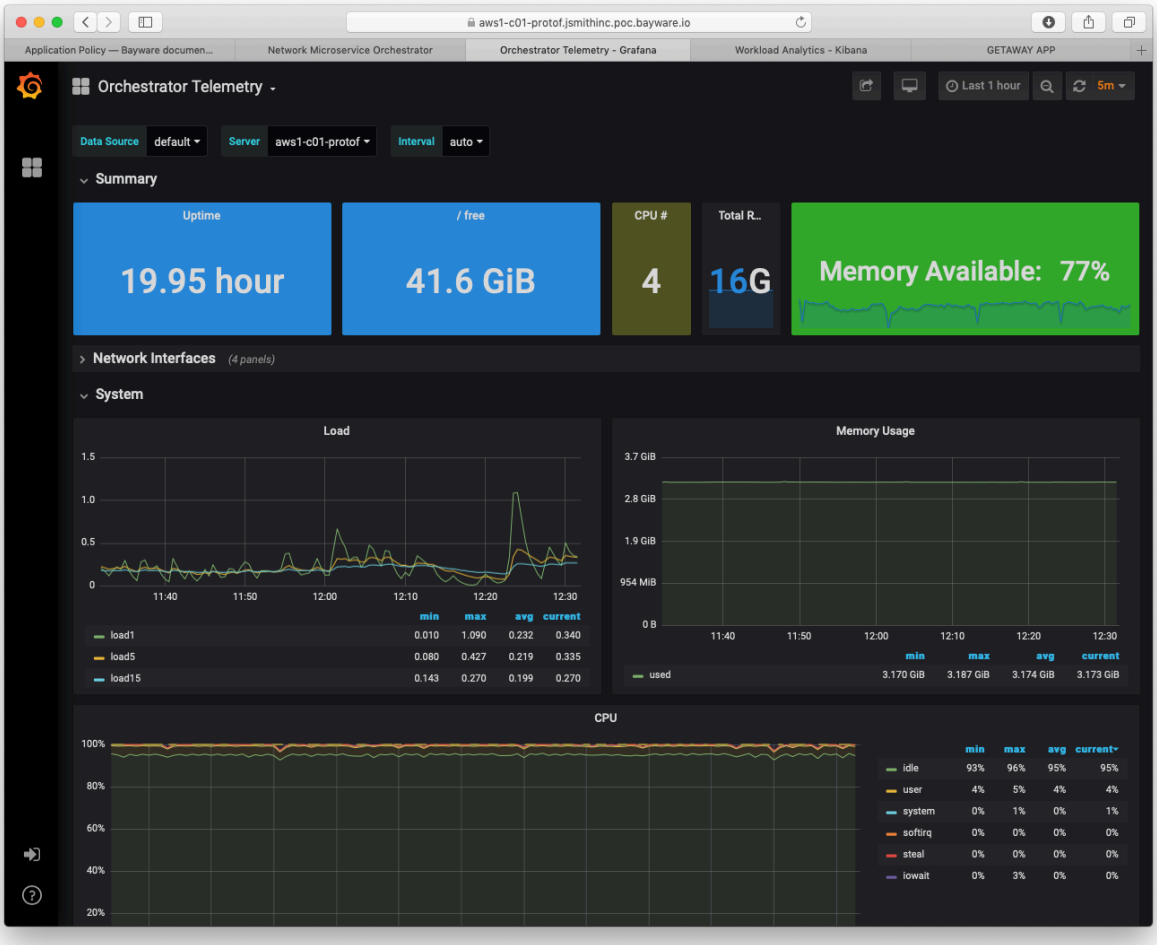


Fig. 13.36: Orchestrator Telemetry

and Grafana stack for telemetry (what you are currently viewing); and `c03` runs the Elasticsearch, Logstash, Kibana (ELK) stack for logging. Both are available as open source.

Now switch to the *Processor Telemetry* dashboard. Do this by clicking on the down arrow next to *Orchestrator Telemetry* located in the upper left corner of the window.

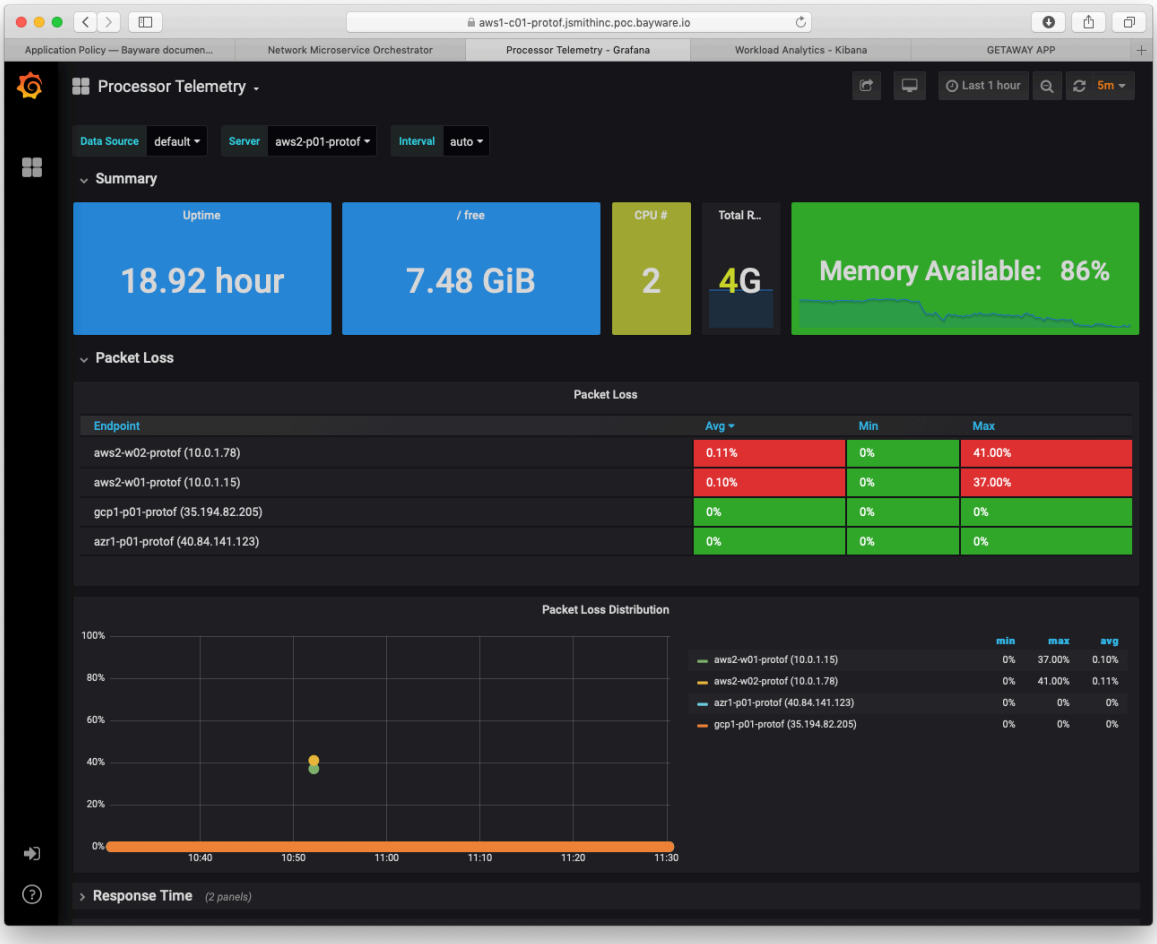


Fig. 13.37: Processor Telemetry - Packet Loss

Fig. 13.37 shows packet-loss statistics for server `aws2-p01-protof`. (Recall that one uses the *Server* drop-down menu in the upper part of the window to select the current server.) It shows brief packet loss within the `aws2` VPC itself, but no packet loss between this server (running in `aws2`) and the ones running in `gcp1` and `azr1`.

Fig. 13.38 shows average response time and response time deviation between `aws2-p01-protof` and each workload in `aws2` as well as between this processor and the processors in each of the other two cloud provider VPCs.

On the right, note that average response time within `aws2` and with `gcp1` is very low. All are < 1.5ms. Response time to Azure is a bit higher at 34.76ms, although, still quite acceptable.

Further, response time deviation values are low enough that this configuration could be used to carry real-time traffic.

Fig. 13.39 shifts to the *Workload Telemetry* dashboard showing server `aws2-w01-protof`. This server

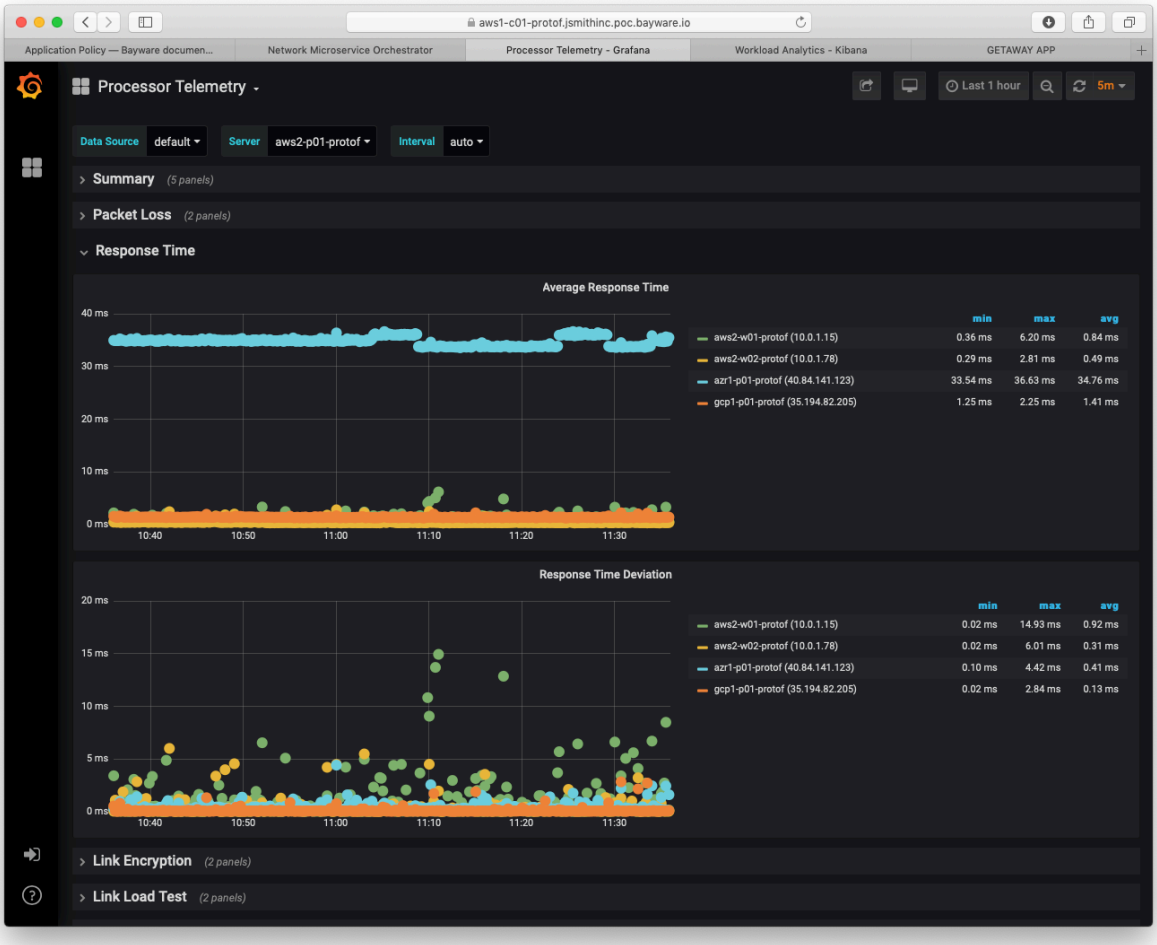


Fig. 13.38: Processor Telemetry - Response Time

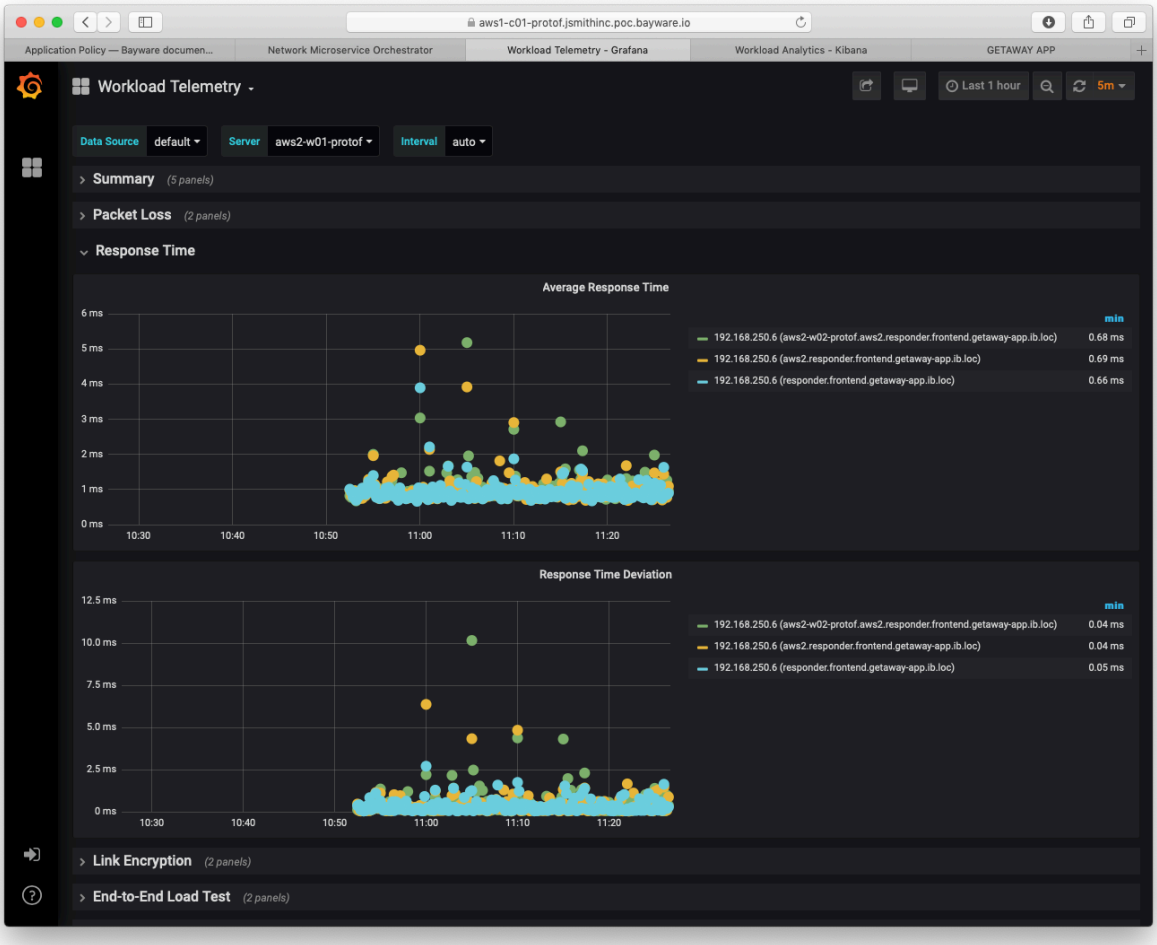


Fig. 13.39: Workload Telemetry - Response Time

runs the getaway-proxy microservice and shows packet loss statistics with getaway-service, which runs on `aws2-w02-protof`. Statistics measured are end-to-end i.e., from application microservice to application microservice.

Having end-to-end statistics between microservices becomes even more powerful if you consider Fig. 13.40 below. Average response time and response time deviation are shown between getaway-proxy microservice and getaway-service microservice. Both are operating in `aws2` VPC at less than 1ms delay: this delay includes transiting through the policy engine (`aws2-p01-protof`) as well as encryption/decryption and the policy agent.

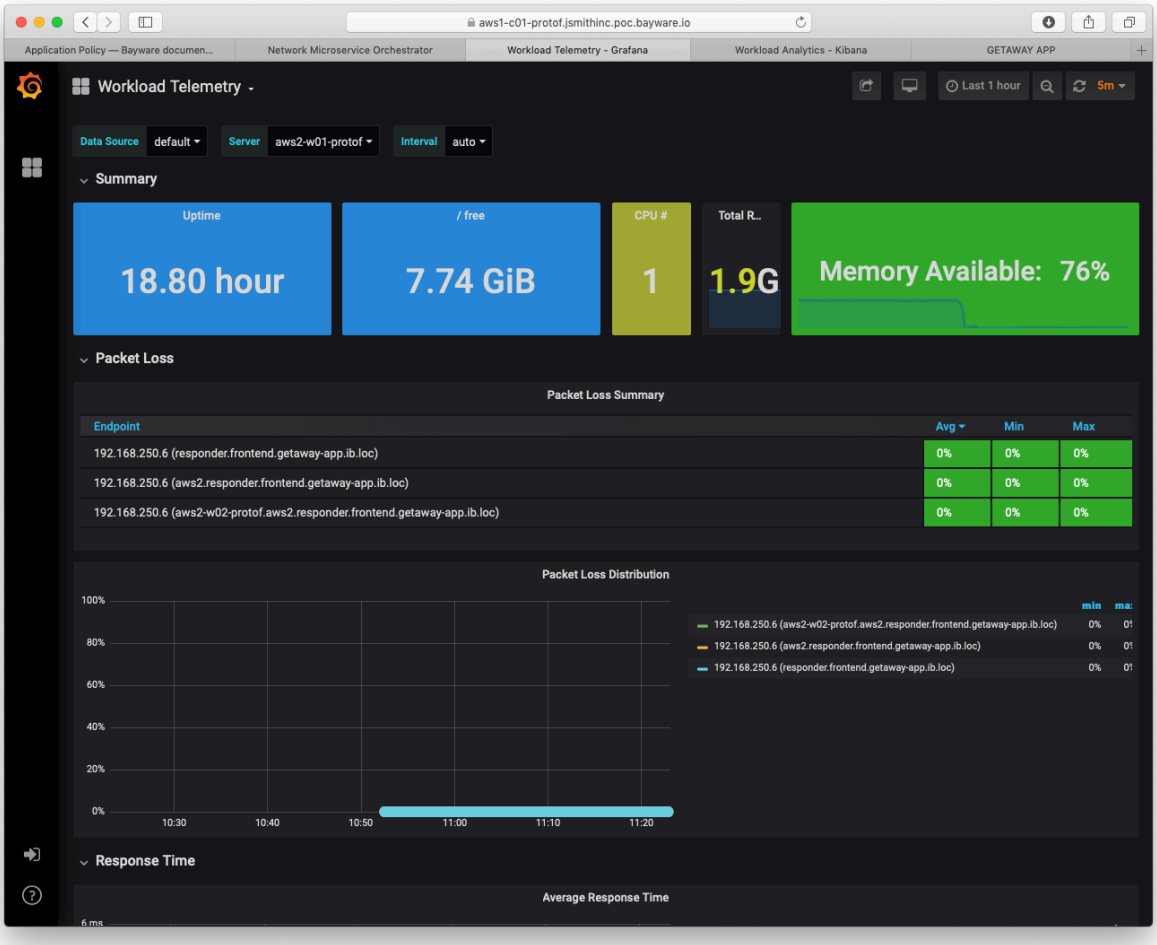


Fig. 13.40: Workload Telemetry - Packet Loss

Finally, Fig. 13.41 shows the *Flow Telemetry* dashboard. The *sFlow agent* drop-down menu is set to `azr1-p01-protof`, which serves the `azr1` VPC with three microservices.

The main panel lists all SIF flows that leave and enter this VPC. The three microservices that communicate with getaway-service are getaway-news, getaway-places, and getaway-weather. Each direction appears in the panel.

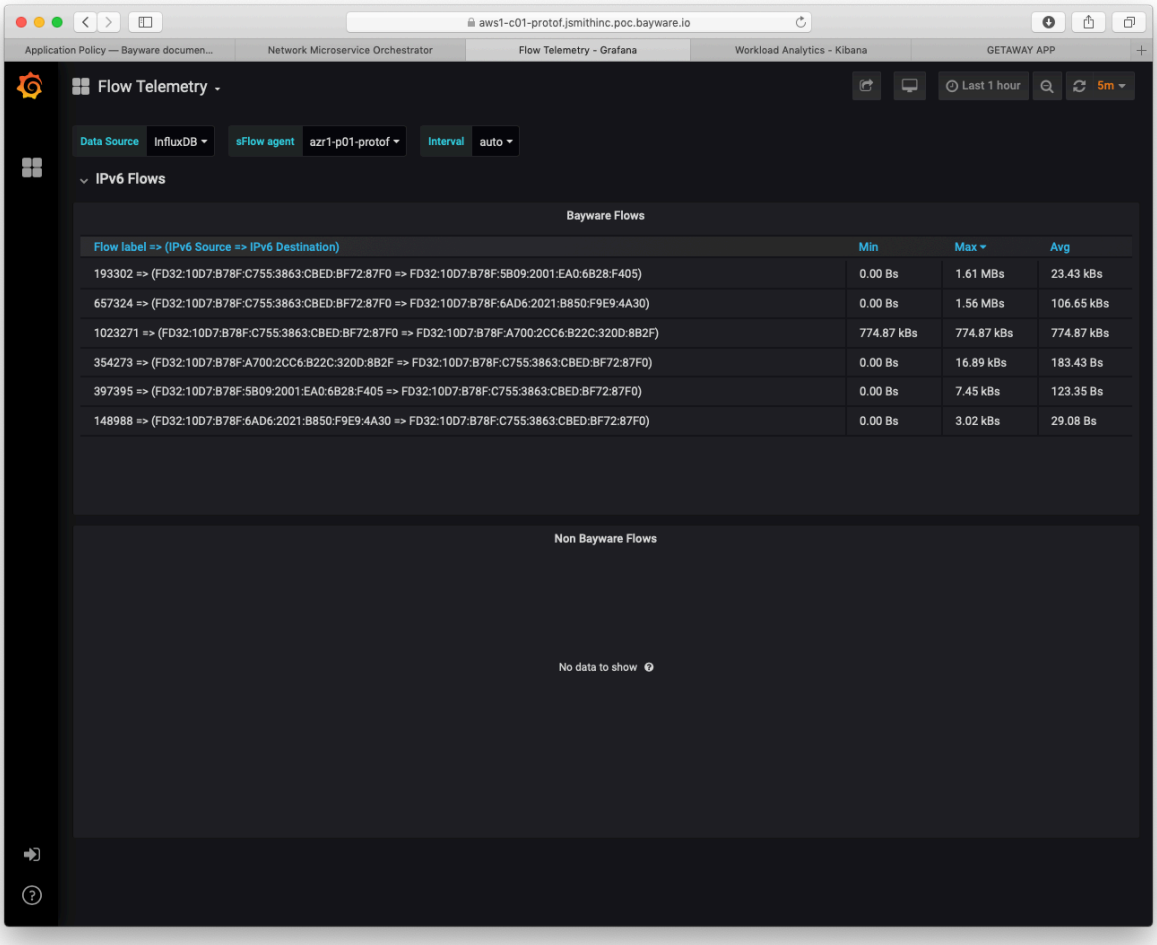


Fig. 13.41: Flow Telemetry

13.6.1 Summary

This section demonstarted how to use the *Telemetry* tools from the policy orchestrator. Statistics were shown for orchestrator, processor, and workload nodes as well as end-to-end telemetry for each flow from client to server.

Next up: now that you have completed the *Deploying a Geo-Redundant App* tutorial, contact Bayware (contact@bayware.io) if you would like to see how a Service Interconnection Fabric can simplify policy setup when deploying your own application.

13.7 What You Need

13.7.1 Contact Us

To work through this tutorial, you will need a personalized set of infrastructure nodes supplied by Bayware. If you haven't already done so, please contact us at info@bayware.io or through the form on our website at the bottom of the **Company** page: https://www.bayware.io/company/.

Once we have created your personalized infrastructure, you will receive a welcome email with the subject *Tutorial Sandbox 2* that has your credentials.

13.7.2 Requirements

You will need a computer with internet access, a web browser, and a terminal window with an SSH client. Regarding the latter:

- on MacOS: use Terminal application with built-in SSH client
- on Linux: use your favorite terminal window with built-in SSH client
- on Windows 10: we recommend installing PuTTY to use as both terminal and SSH client; look here for a good installation & usage video.

13.7.3 Documentation

You will need access to this documentation. Since you are reading this, it means you have already received authorization via a link in the welcome email you received from Bayware. Keep in mind that the link in the email has an embedded token. If your authorization ever times out, you may be asked for a user name and password by our documentation site. When this happens, simply return to your welcome email and click the documentation link with the embedded token.

13.8 What To Expect

In this tutorial, you will be installing an example application called *Getaway*. We want this tutorial to be educational for you. To that end, we have included details of how to use the Service Interconnection Fabric and general concepts about the Service Interconnection Fabric that can be applied to other applications. We will have succeeded when you are able to envision how to run your app with the benefits of this technology.

As you follow through the tutorial, you will be working at the command line on your Fabric Manager and in a browser on your Orchestrator GUI. Command-line text boxes are shown both in light green, which indicates you need to type, and light gray, which displays output you should see on your terminal.

Keep in mind that tutorial screen shots, command-line text boxes, and URLs are shown customized for user *Jane Smith* using an interconnection fabric called *protof*. Your own infrastructure components will vary according to your welcome email.

13.9 Tutorial Outline

1. Introduction *(10 minutes to complete)*
    a. The Scenario
    b. Personalized Installation
    c. Fabric Manager
    d. Orchestrator
2. Application Infrastructure *(20 minutes to complete)*
    a. Requirements
    b. Service Interconnection Fabric
    c. Fabric Manager Tool: bwctl
3. Application Policy *(15 minutes to complete)*
    a. Requirements
    b. Service Graph
    c. Fabric Manager Tool: bwctl-api
4. Application Services *(10 minutest to complete)*
    a. Requirements
    b. Authorized Microservices
    c. Workload Orchestration

CHAPTER 14

Fabric

14.1 Bayware Solution

14.1.1 Application Centricity

Accelerating digital transformation and rapidly changing customer requirements compel enterprises to continuously add and enhance applications. Enterprises are transforming how they develop and deploy applications; leveraging cloud-native principles and microservice architectures to enhance agility, improve time to market, and get lean.

As they modernize, enterprises want to exploit the benefits of multicloud deployments, control communication flows wherever an application runs, and maintain private-cloud levels of security. Doing this is limited by today's complex networking solutions. Enterprises need an *application-centric communication environment* that delivers programmability, observability and security; while underlying infrastructure remains general purpose.

14.1.2 Intent Based Networking

Bayware is a connectivity-as-code architecture that gives every application its own secure, overlay network, all in software. This fit-for-purpose solution introduces the programmable service graph where each application service initiates and controls its own programmable communication flows; enabling the long-promised *intent-based networking*.

Bayware radically simplifies the interface between an application and underlying networks. This accelerates continuous deployment of application services by eliminating constraints imposed by managing network configurations, and securely serves each application's data communications needs as they change and move across any cloud.

14.1.3 Service Interconnection Fabric

Bayware's connectivity-as-code approach uniquely enables direct programming of application data flows in software. This service interconnection fabric (SIF) is the first secure, programmable application service graph.

A service graph represents application services as nodes and network connectivities as edges. While the nodes are software by definition, Bayware extends that to the edges with programmable connectivities. As secure, simple-to-program, lightweight communication contracts, these programmable connectivities deploy with application workloads and provides an unprecedented level of control and agility in heterogeneous hybrid and multi-clouds.

Bayware service interconnection fabric is a suite of distributed software components that run on standard x86 Linux machines and operate securely on top of any virtual or physical infrastructure.

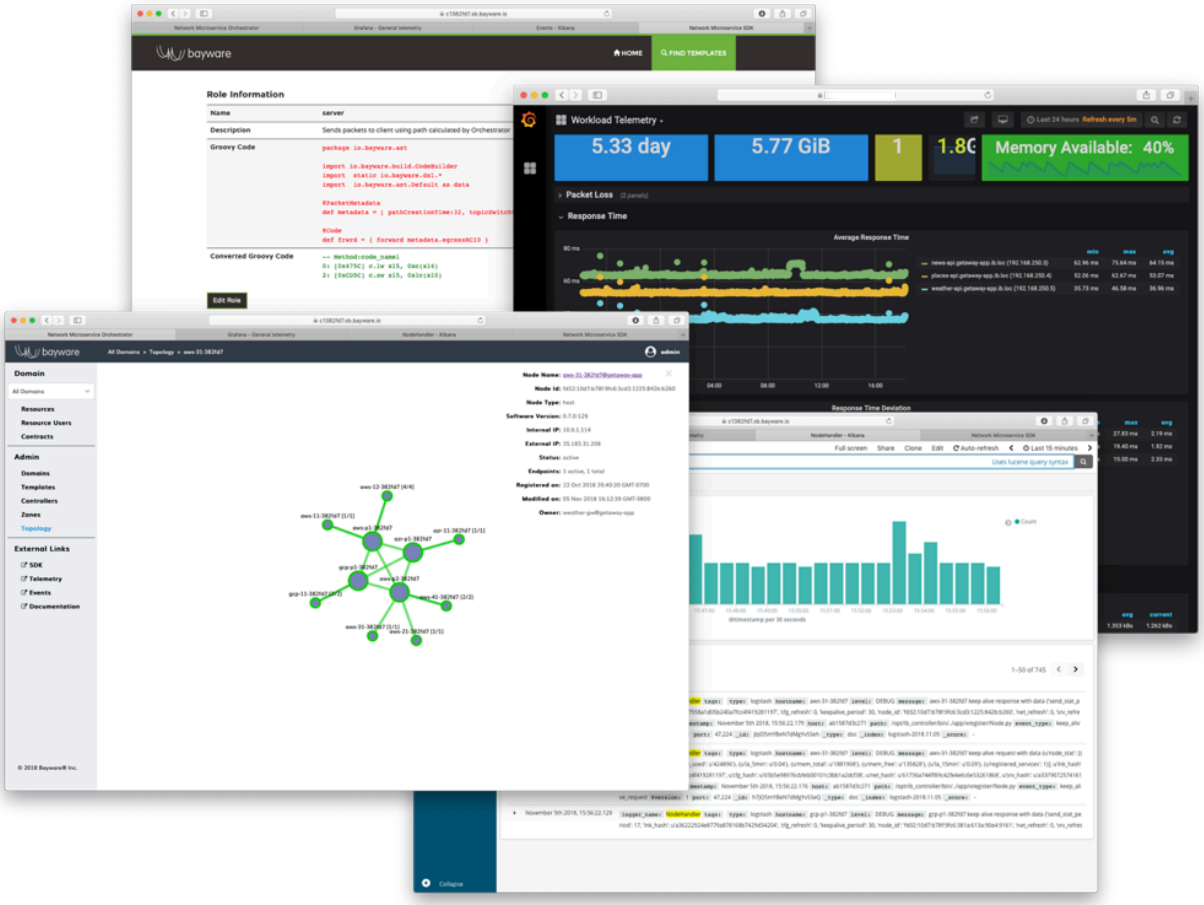


Fig. 14.1: SIF: Programmability, Observability and Security

14.2 How Bayware Works

14.2.1 Basic Principles

Bayware created and patented connectivity-as-code architecture with microcode communication contracts. Each is programmable and can be designed and approved by networking and security professionals. Microcode is carried from the workloads in the packet headers, using standard IPv6 header extensions. The execution of the contract by software processors then creates the desired steering of packets through the overlay network.

With Bayware, provisioning of service-to-service communication is easy:

1. Based on the service graph, program application intent and network policy as microcode communication contracts; simply by adding application labels to the desired flow pattern from a library of contract types.
2. Deploy lightweight Linux daemons (agents) on workload hosts that retrieve authorized contract roles to insert as highly compact microcode into IPv6 packet extension headers in response to applications.
3. Provision a fabric of processor software (on Linux x86 machines) in target public and private clouds to securely execute service-to-service connectivity only as authorized by received microcode.

14.2.2 Three-Part Solution

Bayware's service interconnection fabric is a three-part solution:

- Bayware introduces new technology that captures **connectivity policy** based only on data available from an application and produces executable microcode.
- This executable code is utilized within a new **service discovery** framework to create network microsegments in accordance with connectivity policy.
- Bayware implements datapath where **packet forwarding** decisions are based on the authentication and authorization of application flows over network microsegments.

Bayware's solution, in a nutshell, works in the following steps:

1. **Connectivity Policy**. Bayware's patented technology converts application connectivity policy (*step 1a in diagram*) into short, executable programs that carry application names, their roles, and rules. The Policy Controller stores these programs and allows service instances to request them (*step 1b*). As such, connectivity policy is inherently infrastructure-agnostic, multicloud ready, portable, and easily implemented from within application deployment code.
2. **Service Discovery**. The Policy Agent, installed on compute resources, requests application connectivity policy from the Policy Controller on behalf of application services (*step 2a*). The Policy Agent sends the connectivity policy, in the form of executable microcode marked with a policy identifier, into the network in special packets used by the Policy Engines to create packet processing rules (*step 2b*). In this way, while traditional service discovery simply returns IP addresses to application services, Bayware's solution additionally establishes end-to-end network microsegments between communicating application services.
3. **Policy-Based Forwarding**. When the application service sends data packets, the Policy Agent marks the packets with policy identifiers (*step 3a*). So as packets arrive, the Policy Engine authenticates and authorizes them over a set of installed processing rules dropping packets that fail and forwarding the others (*step 3b*). By doing this, Bayware's solution ensures no connectivity exists between application services that was neither specified in step 1 nor requested in step 2.

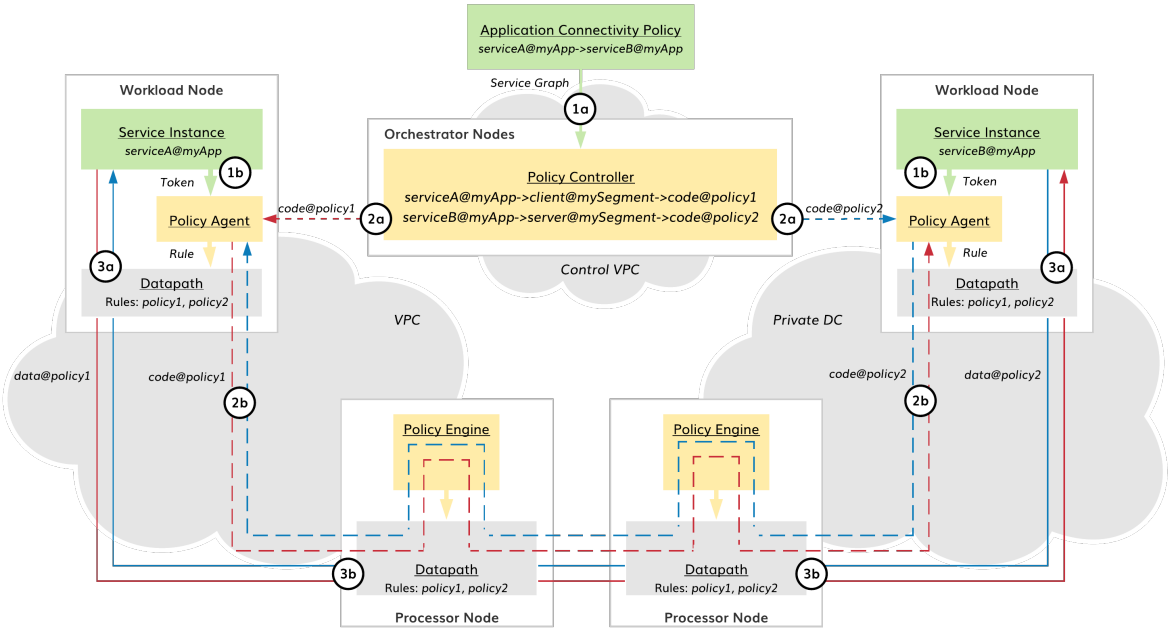


Fig. 14.2: Connectivity Policy, Service Discovery, Policy-Based Forwarding

14.2.3 Deployment Options

Bayware breaks from Software Defined Networking (SDN) models that push complex reconfigurations into underlying networks, which were not built for continuous change.

Enterprises can run Bayware standalone using the underlying infrastructure of cloud providers' VPCs. Bayware also runs in concert with application service orchestration systems and SDNs that provision lower layer data center and branch networking.

Bayware reduces acquisition and operation costs by running over the top of brownfield underlay networks, eliminating the need to install and configure any additional specialized networking appliances or controllers. Bayware provides an all-in-one solution for service-to-service communications.

14.3 Why Bayware

Bayware brings NetOps and SecOps into the DevOps model of continuous, application-centric deployment. It is all code: enterprises get the same development and deployment agility and the same cloud-scaling benefits for networking functions as they are getting from cloud-native applications.

Bayware has a unique solution for enabling application service to communicate across the diverse hybrid-cloud and multi-cloud infrastructures: (1) enables the network to respond to frequent additions, updates and changes in scale and location; (2) ensures that security and network policy meet compliance and corporate standards.

Today's solutions do one, or the other well, but not both. Bayware's pervasive security automatically encrypts flows and is hyper-micro-segmented by default. This improves on security, observability and accountability for network usage rather than requiring compromises as the newest service mesh solutions do.

Service to service communication in a hybrid or multi- cloud/cluster/VPC use case	Standard Service Mesh Stack	Client-side Proxy + Bayware Stack
Client-side load balancer	DevOps	DevOps
Service registry and discovery	DevOps	bayware DevOps
Edge proxies	DevOps + SecOps	
Container Network Interface (CNI)	DevOps + NetOps	
Firewall, ACL, IPAM, vRouter, VPNGW (SDN)	SecOps + NetOps	
Cloud VPCs: Subnets, Security Groups	Cloud specialists + SecOps	
DevOps self-sufficiency	**No**	**Yes**

Fig. 14.3: DevOps Self-Sufficiency

CHAPTER 15

Orchestrator

15.1 Architecture

The SIF Orchestrator makes application service connectivity policy available to VMs and containers for immediate download anytime and anywhere they request it. Simple software programming gives the user the ability to create custom connectivity policies, allowing application services to instantiate network microsegments in strict accordance with security rules. The SIF Orchestrator enforces these policies with flow-level granularity and provides full observability for real-time policy auditing across the fabric.

The diagram below shows the architecture of the SIF Orchestrator in default deployment mode.

The SIF Orchestrator is implemented as a container cluster and typically, consists of three nodes:

- **Controller** – connectivity policy management
- **Telemetry** – compute and network statistics
- **Events** – log processing

The Orchestrator RESTful northbound interface (NBI) allows third-party automation systems to manage all resource and service connectivity policies in the fabric. The Fabric Manager itself utilizes this same interface for policy management. The NBI also facilitates communication with the orchestrator when using a web browser to control policy or access telemetry and log data. The northbound interface is secured with a Let's Encrypt certificate, OAuth2.0 authorization, and OpenID Connect authentication.

The Orchestrator's southbound interface (SBI) enforces the resource and service connectivity policies on all workload and processor nodes in the fabric. The SBI supplies workload Policy Agents with service connectivity policy and processor Policy Engines with resource connectivity policy. Additionally, telemetry and log agents ship data to the Orchestrator from fabric nodes using this interface. All SBI operations are fully automated and secured with mTLS.

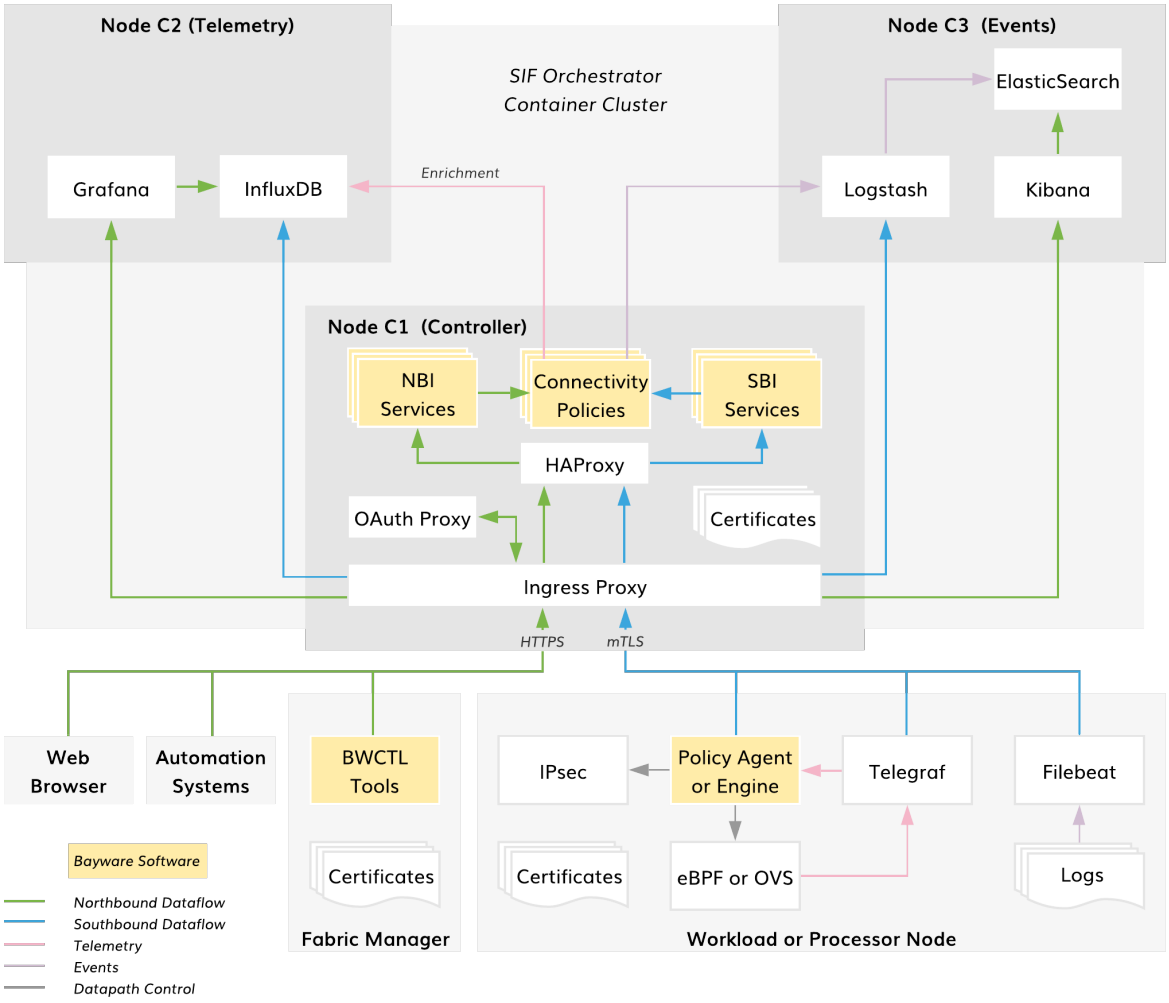


Fig. 15.1: SIF Orchestrator Architecture

15.2 Controller

The controller services are the only mandatory components of the orchestrator. They comprise four functional sets:

- **Proxy** services – ingress, OAuth, and high-availability proxies
- **NBI** services – northbound interface support
- **SBI** services – southbound interface support
- **Connectivity Policy** services – resource and service policy management

As shown in the diagram below, connectivity policy management logically consists of the following functional blocks: administration, service configuration, resource configuration, and operation.

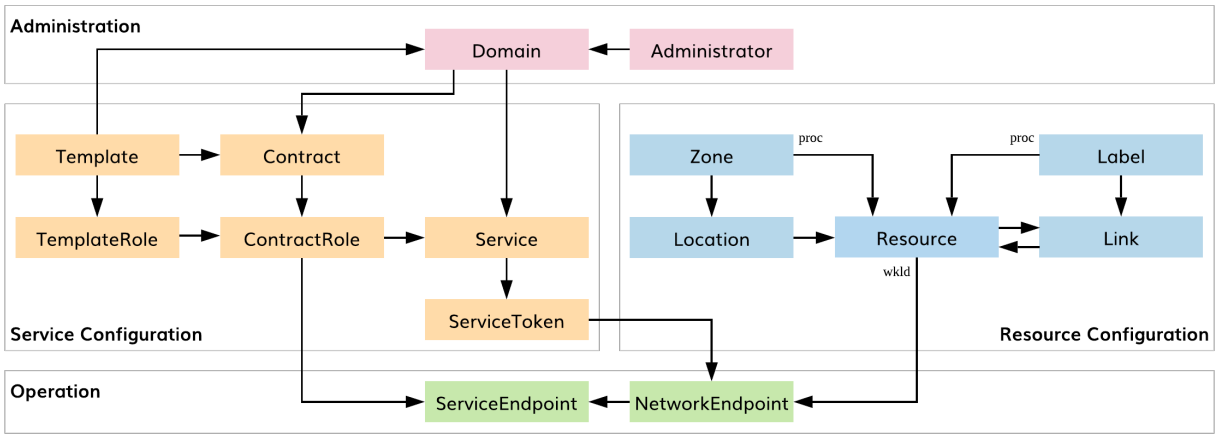


Fig. 15.2: SIF Connectivity Policy Entities

The Administration block allows the user to manage *administrators* and namespaces for application deployment (i.e., *domains*). Service and Resource Configuration blocks provide service and resource graph management, respectively. The Operation block automatically enforces connectivity policies on service and network endpoints.

A service graph depicts service policy configuration on the controller. The screenshot below shows an example.

The small circles on the graph represent an application *service*, while the large circles on the graph represent an application *contract*. The latter defines the relationship between application services. A contract inherits all essential communication rules from a *template* but allows for customization. A service acquires a *contract role* and can communicate with opposite-role services within the same contract. To set up a *service endpoint*, each service must possess a *service token*. The service applies the service token to a *network endpoint* on a VM or Kubernetes worker node (i.e., *workload node*) during service instance deployment. During initialization, the service endpoint automatically retrieves settings from the corresponding contract role specification on the controller and activates a dedicated service discovery process in the fabric.

A resource graph depicts resource policy configuration on the controller. The screenshot below shows an example.

The small circles on the graph represent workload nodes, while the large circles show processor nodes. Workloads and processors are called *resources* in the SIF Policy Entities data model. Workloads provide application services with *network endpoints*. Processors work as secure transit gateways for workloads. A security *zone* abstraction allows the user to assign a processor to serve a set of workloads in a given *location*.

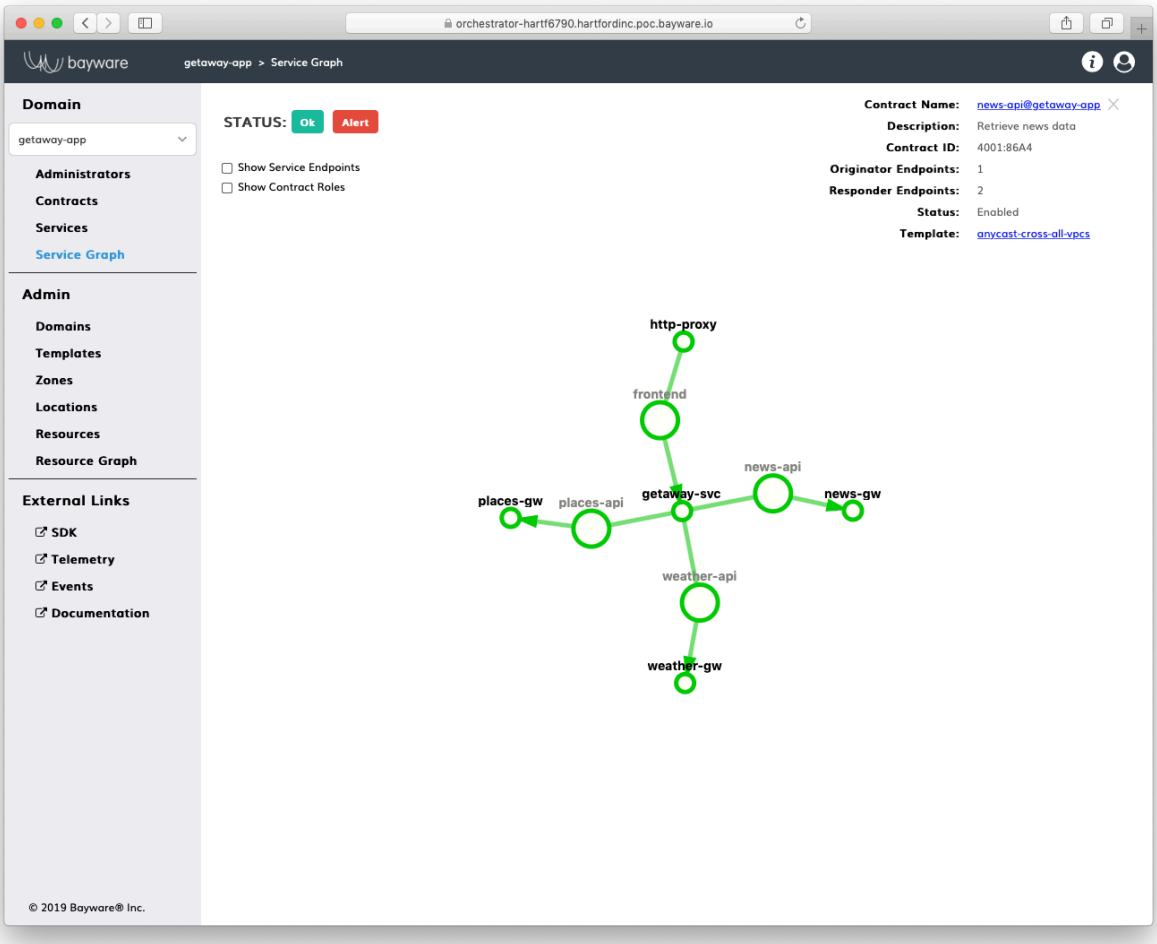


Fig. 15.3: Service Graph

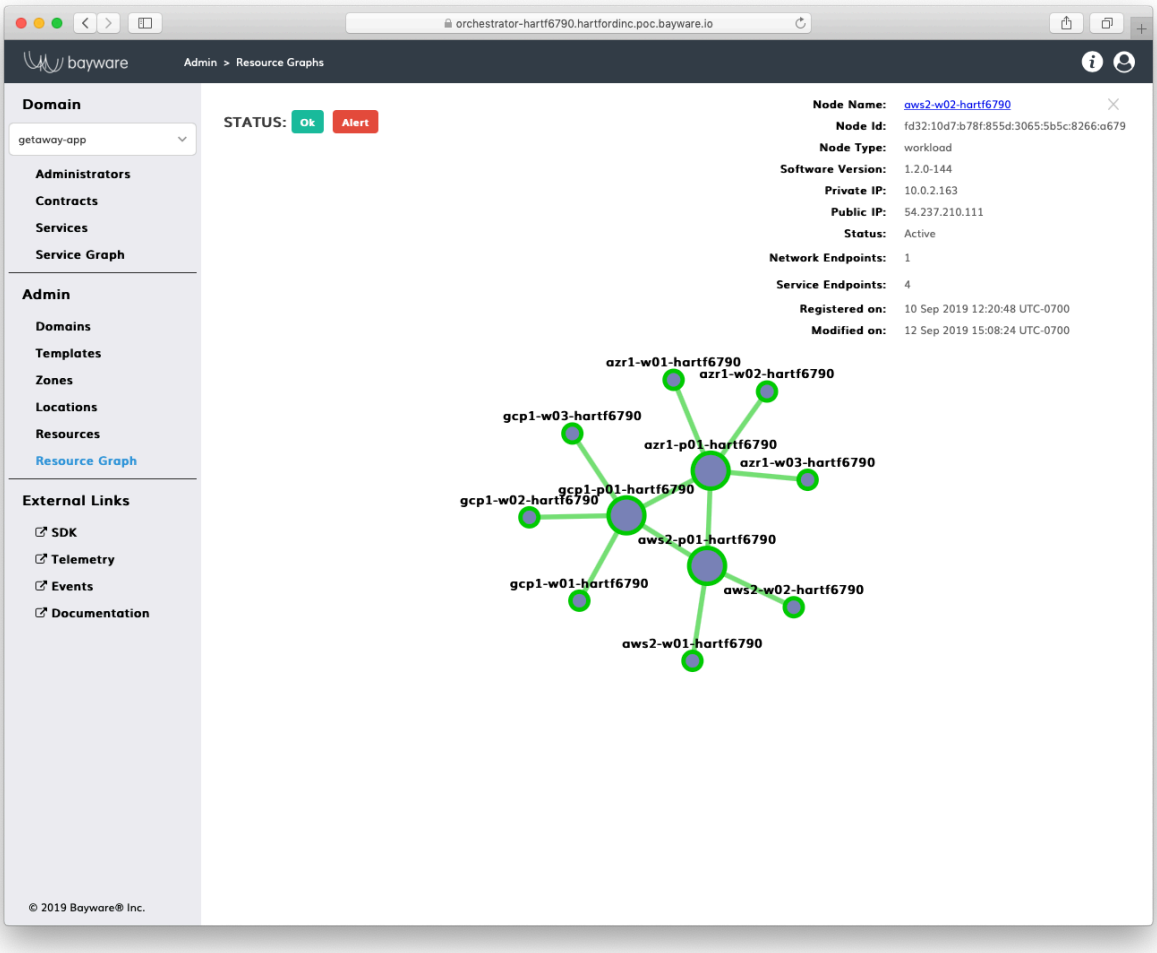


Fig. 15.4: Resource Graph

When a workload in a given location comes up, a logical *link* automatically attaches the workload to a zone processor. Link and processor *labels* allow the user to mark pathways for application service discovery packets in the fabric. By processing labels, these packets may instantly change routes in order to lower infrastructure cost, balance compute load, and facilitate application scaling.

15.3 Telemetry

Telemetry services are optional components of the orchestrator. When installed, they automatically collect and process the following statistics:

- **Orchestrator telemetry**
  - System: CPU, memory, disk, network interface
- **Processor telemetry**
  - System: CPU, memory, disk, network interface
  - Link: packet loss, response time, encryption, link utilization
- **Workload telemetry**
  - System: CPU, memory, disk, and network interface
  - Service: packet loss, response time, endpoint utilization
- **Flow telemetry**
  - Packets: contract, source, destination, transit points, protocol, port, total size

The screenshot below illustrates a processor telemetry dashboard (more examples are in the Telemetry section of the Deploying a Geo-Redundant App tutorial).

A telemetry agent on each node collects statistics from all local sources and sends them to the orchestrator in an encrypted mTLS channel. Tightly integrated with the processor and workload software, the agent is able to automatically discover new links, service endpoints, and flows and process them as telemetry sources. Near-real-time background telemetry enrichment makes the statistics, provided by the agent, easy to interpret and correlate on the orchestrator.

In summary, the orchestrator telemetry services are available out-of-the-box. They provide the user with unique data on application connectivity health from multiple perspectives. The service setup and operation require neither manual provisioning nor configuration. Moreover, automatic authentication and total encryption allow telemetry data to be securely routed in a multicloud environment.

15.4 Events

Events services are also optional components of the orchestrator. When installed, they automatically collect and process the following logs:

- System;
- Encryption (IPsec);
- Datapath (eBPF, OVS);
- Policy Agent and Engine;
- Policy Controller.

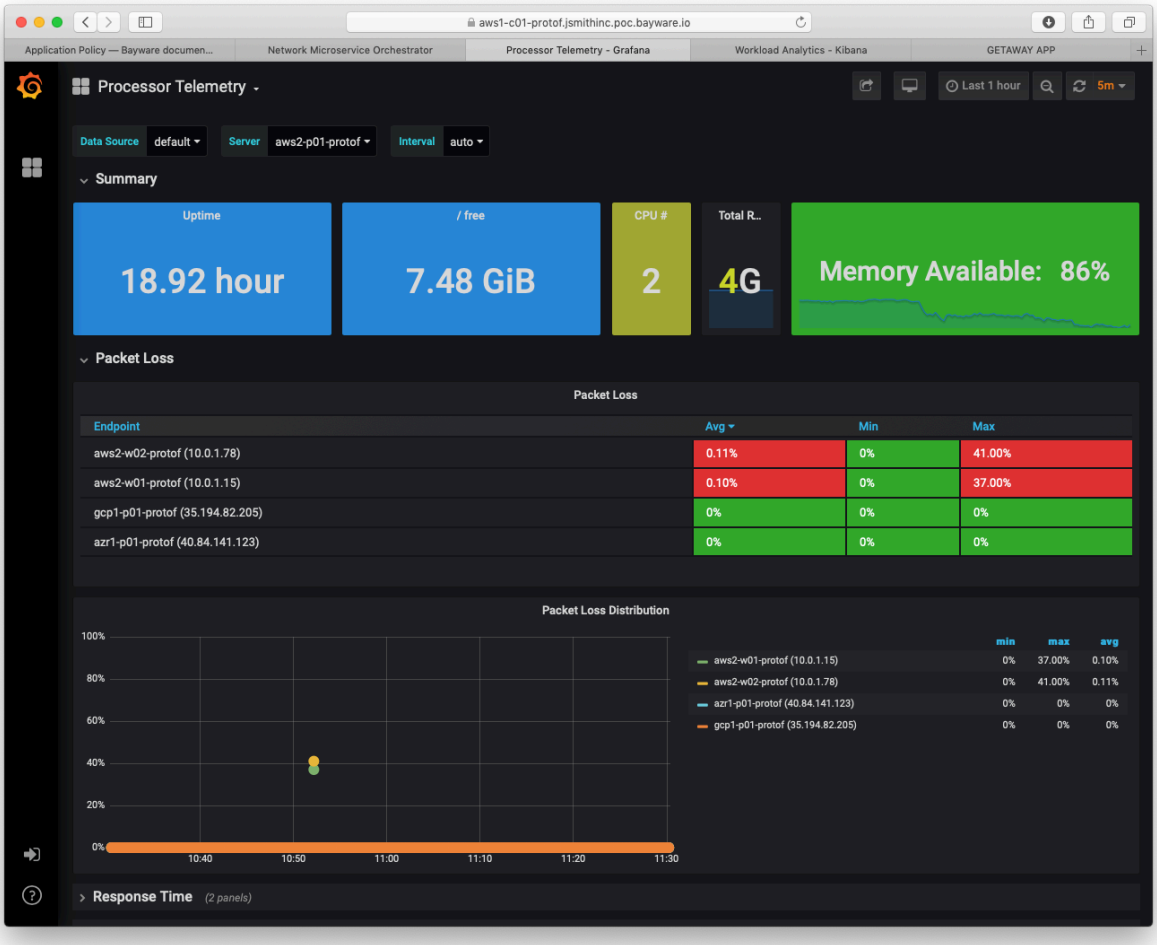


Fig. 15.5: Processor Telemetry – Packet Loss

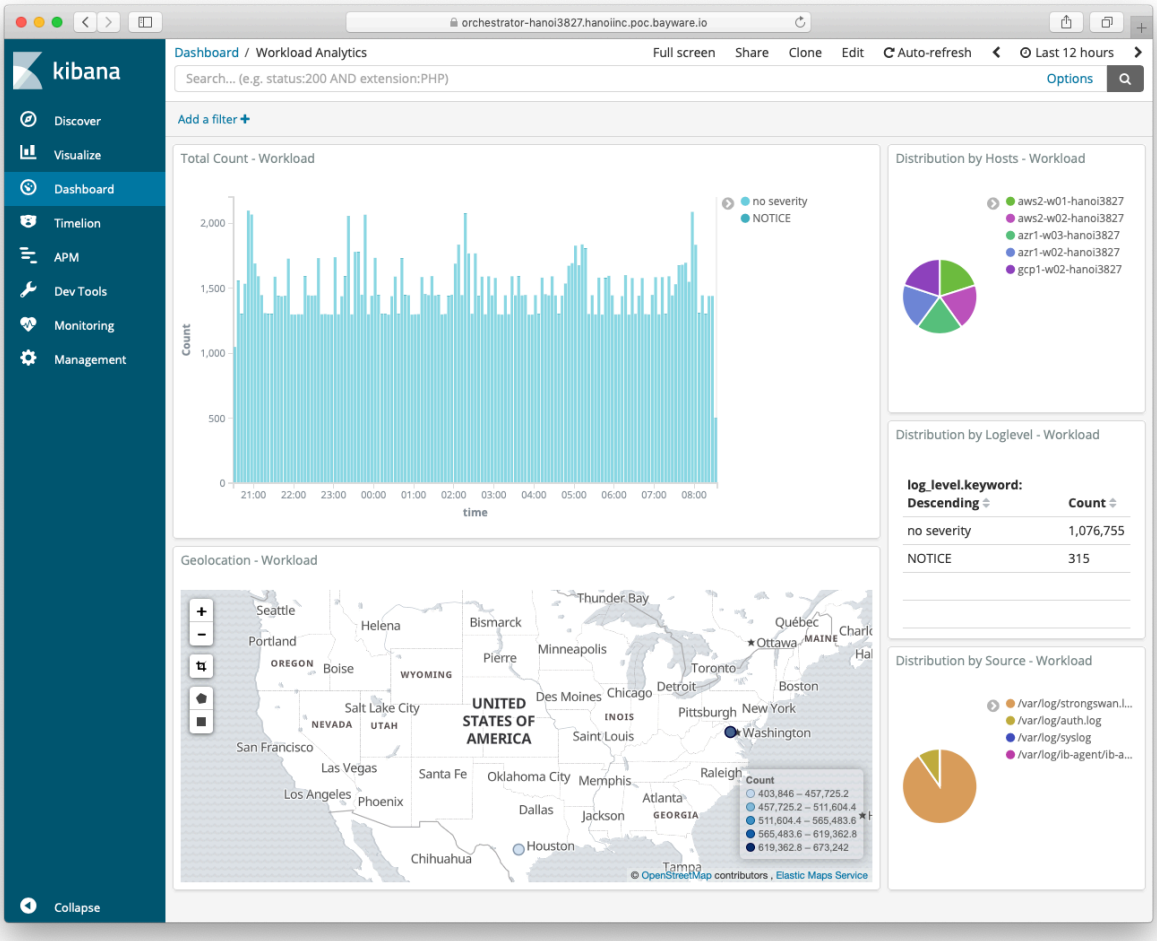


Fig. 15.6: Events – Workload Analytics

The screenshot below illustrates a default workload analytics dashboard.

A log shipper on each workload and processor node sends all local log data to the orchestrator in an encrypted mTLS channel. As shown on the SIF Orchestrator Architecture diagram at the beginning of this chapter, the log shipper is part of the fabric logging pipeline that includes Logstash, ElasticSearch, and Kibana. While deploying a new node, the Fabric Manager automatically sets up all stages in this pipeline to collect, transform, store, and visualize events. Also, by default, the Policy Controller pushes all connectivity policy changes to the Events node. As such, every action made either by a fabric administrator or a workload/processor node is documented and available for auditing on the Events node.

Again, the orchestrator events services are available out-of-the-box. Their zero-touch configuration doesn't require any fabric administrator involvement. Automatic authentication and encryption make the orchestrator events services immediately multicloud-ready. The seamless integration of the orchestrator events services with other fabric components greatly simplifies log data consumption and provides additional level of visibility in application policy, security, and operations.

CHAPTER 16

Processor

16.1 Introduction

In the service interconnection fabric (SIF), each processor node, or simply *processor*, is a security checkpoint for application control and data flows. Processors facilitate secure application service discovery, enact service connectivity policies with flow-level granularity, and forward encrypted application data between clouds, clusters, and other trusted domains.

The SIF processor is a virtual network appliance available both as an image in Azure, AWS, and GCP and as a package for installation on a Linux machine. As shown in the diagram below, processors are deployed as gateways to trusted domains. Each processor secures a set of workload nodes–physical servers, VMs, Kubernetes worker nodes–in application control (i.e., service discovery) and data planes.

The SIF **resource connectivity policy** defines processor reachability by workloads and other processors. Each processor enforces resource policy that the fabric orchestrator requests. Policy enforcement starts with automatic processor registration after its installation. The orchestrator checks the processor identity and adds the processor as a new fabric resource. The processor receives link configuration from the orchestrator and automatically sets up secure connections with workloads and other processors. Additionally, the orchestrator assigns labels to the processor and its links to mark the pathways in the fabric for application service discovery requests.

The SIF **service connectivity policy** defines end-to-end application service reachability. A workload node, hosting an application service instance, sends a service discovery request to an adjacent processor node in order to establish a secure network microsegment in the fabric. After validation, the processor executes the request, configures the microsegment in its datapath using the execution outcome, and forwards the request to other processors or workloads when required. Once the microsegment is established by all the processors en route between the originator and one or multiple responder workloads, service instances on these workloads can immediately start data exchange.

Processors make the SIF a **zero-trust communication environment with zero-touch configuration**.

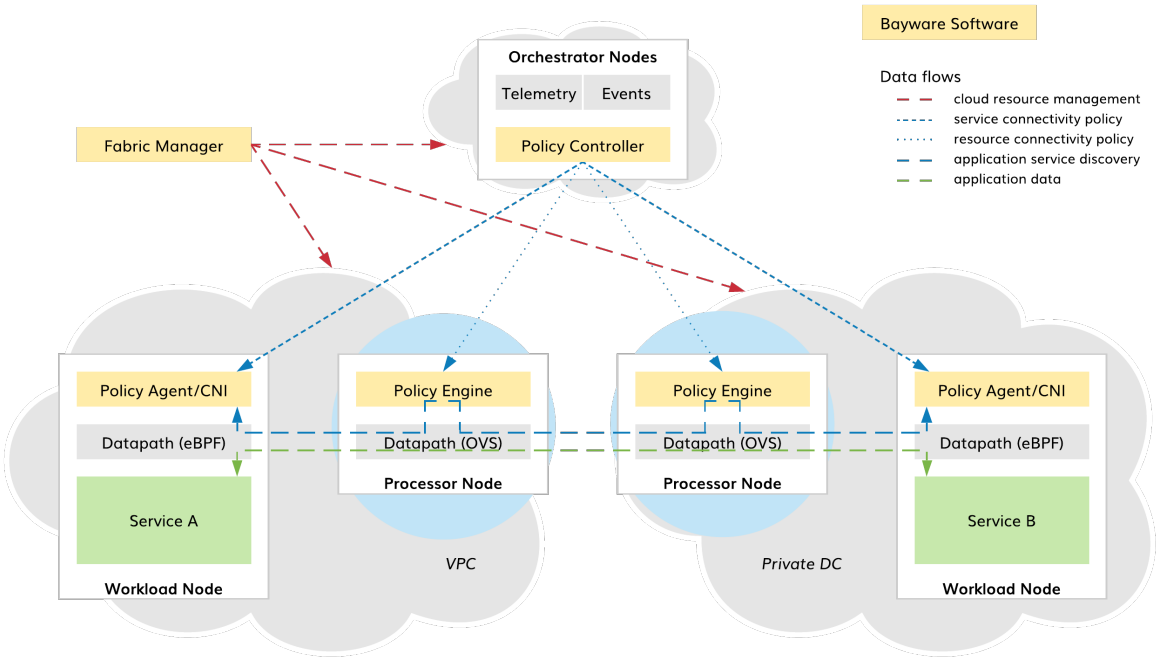


Fig. 16.1: Processors in SIF

16.2 Capabilities

16.2.1 Overview

One or more processors can be assigned to secure a trusted domain, called *zone* in the SIF policy model. When an SIF administrator adds a workload location to a zone, all the workloads in this location automatically connect with one or several processors serving the zone. If one processor has higher priority than the others, all workloads connect to this processor. Otherwise, connections will be evenly distributed among processors with the same priority.

Note: A processor can be assigned to more than one zone, and in each zone the administrator can select a different priority for the processor.

As shown in the diagram below, each processor plays the following roles:

- SSH jump host,
- IPsec gateway,
- Application policy engine,
- Policy defined datapath.

16.2.2 SSH Jumphost

The fabric manager utilizes each processor as a single point of entry into a given security zone. To reach workloads in a given zone, the fabric manager uses one of the zone processors as a Secure Shell (SSH)

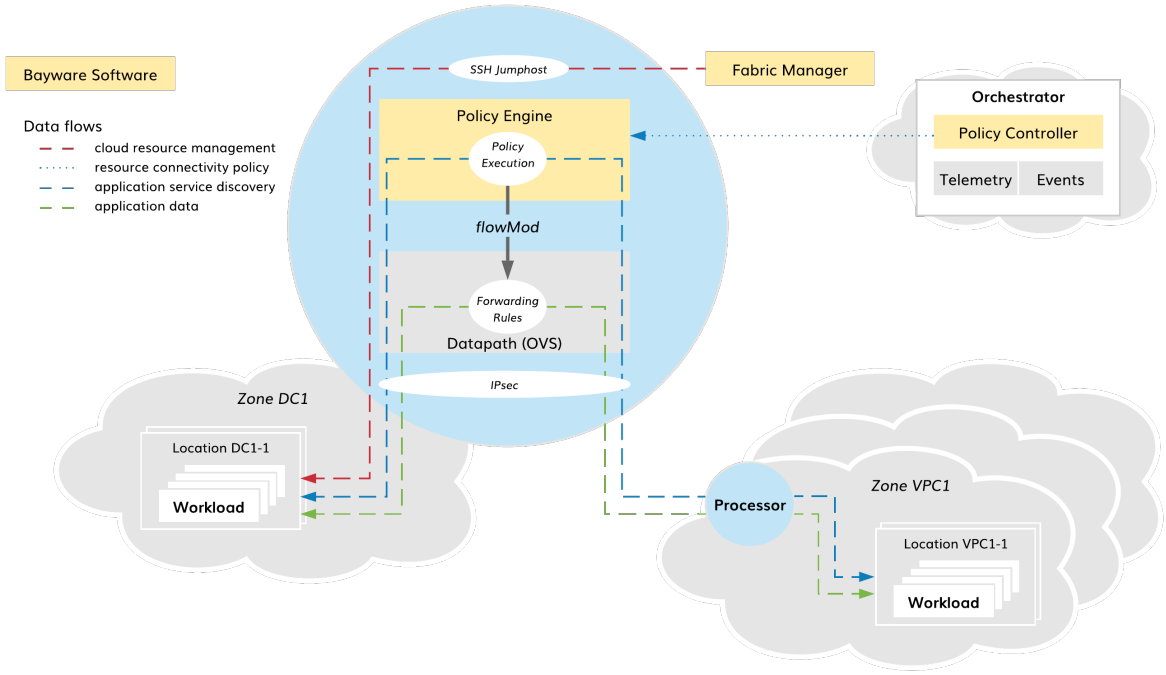


Fig. 16.2: Processor Capabilities

intermediate hop. This allows the fabric manager to transparently manage workloads in multiple trusted domains without exposing those workloads to public networks, even though they might be in overlapping private IP address spaces.

Using processors as SSH jump hosts enables additional security measurements in the multicloud infrastructure. At the network level, only SSH connections from the fabric manager to zone processors and from zone processors to workloads are permitted. At the SSH level, processors, in this case, perform additional authorization on fabric manager-workload connections.

16.2.3 IPsec Gateway

When application data leave the trusted domain, processors automatically encrypt all packets. All processors in a given fabric form a site-to-cloud or cloud-to-cloud VPN, walled off from other fabrics and the outside world. Resource connectivity policy defines a desired VPN topology abstractly as a resource graph with processors playing transit node roles.

As part of resource policy enforcement, the fabric orchestrator imposes link configuration on each processor. Processors use certificate-based mutual authorization to set up secure connections with the prescribed nodes. Then, a standard Linux kernel subsystem performs packet encryption and decryption using hardware acceleration whenever available. Additionally, the fabric manager sets up packet filters to ensure that only IPsec traffic originated or terminated on processors can leave and enter security zones.

16.2.4 Application Policy Engine

To be able to communicate with other workloads in the fabric, a workload requests that processors establish one or more secure network microsegments. Processors always work in a default-deny mode. A packet cannot

traverse a processor until that processor executes an application connectivity request for the data flow to which the packet belongs.

The connectivity request arrives at the processor as executable code assigned to the flow originator endpoint by the orchestrator. The processor validates the code signature and executes the instructions. The result of code execution may request that the processor: (1) *connect* the flow endpoint to a given network microsegment and (2) *accept* data coming to the flow endpoint from a given network microsegment. Using this outcome, the processor sets up local forwarding rules for a period of time specified in the request. Additionally, the application connectivity request may *subscribe* to requests from other workloads and *publish* itself to already subscribed workloads.

With this new approach, various connectivity policies can be easily developed or customized. For example, one policy can restrict a microsegment to a subset of trusted domains or even a single domain. Another policy can establish a microsegment with cost-based or load-sharing target selection. Because all these policies are just code, processors will immediately enact them upon workload request across the fabric.

16.2.5 Policy Defined Datapath

Each processor includes a standard Linux datapath, running in a default-deny mode. As described above, only workloads can change datapath forwarding behavior. A local policy engine installs rules in the datapath after execution of workload instructions.

Each rule in the datapath processes application data packets in two steps. Firstly, the rule checks whether the packet comes from a flow endpoint already attached to a given microsegment. Secondly, it ensures that the destination node accepts packets from this microsegment. If both true, the datapath forwards the packet to the next hop associated with the destination node.

Such an application-defined datapath enables unprecedented workload mobility plus security with flow-level granularity. The fabric forwarding behavior instantly adapts to new deployments of application service instances as well as application connectivity policy changes.

16.3 Internals

The processor consists of an application policy engine and a datapath. The policy engine executes application instructions delivered in discovery packets and uses execution outcome to change datapath forwarding behavior.

16.3.1 Policy Engine

From a high level, the policy engine functionality breaks down into five modules as presented in the following diagram.

Ingress Parsing and Authorization

Discovery packets are parsed, classified, rate limited, and processed by security blocks to ensure that only authorized packets proceed to instruction execution.

All discovery-specific information is contained in the IPv6 header. The parser sanity checks the IPv6 header and calls out relevant fields and extensions. An orchestrator ECDSA signature in an IPv6 header extension covers packet authorization information, control data, instructions, and program data. So, security blocks ensure packet legitimacy before instruction execution. In the next step, the classifier performs the necessary lookups in flow and contract tables to initialize an isolated execution environment for the packet instructions. The rate limiter protects security blocks and execution environment from overloading.

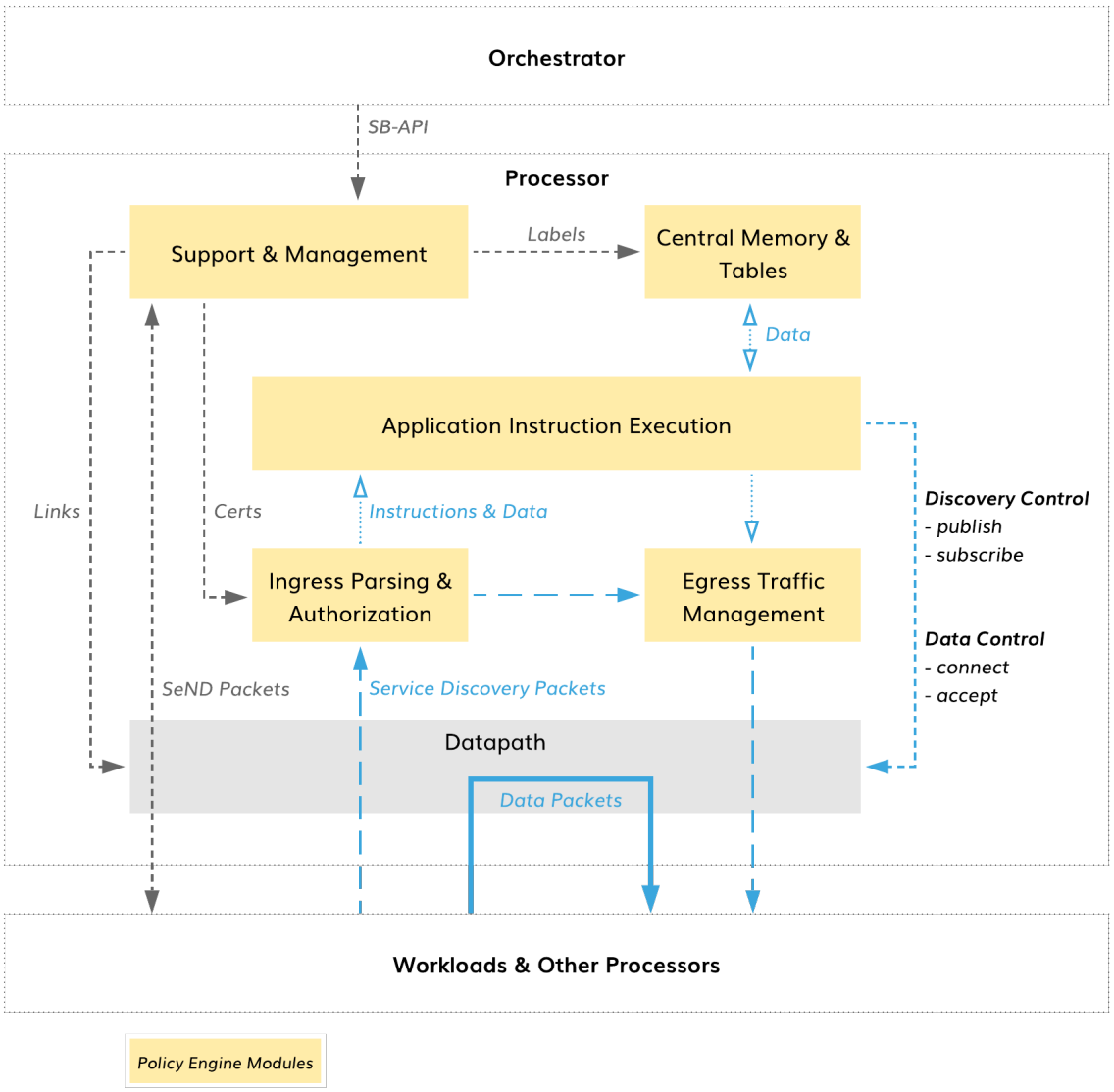


Fig. 16.3: Policy Engine

Application Instruction Execution

A virtual processing unit executes the instructions in the discovery packet using packet program data, stored program data and other shared, processor-level information. Following instruction execution, special logic filters generate control commands for the datapath through a set of allowed actions determined by the security model.

To process instructions, each discovery packet receives a virtual processing unit (PU) fully isolated from the outside world. The PU and instruction set are based on the RISC-V open source ISA. The RV32IAC variant with vector extensions offers support for the base integer instructions, atomic instructions, compressed 16-bit instructions, and vector-processing instructions. A PU uses a virtual 16-bit address space broken into pages. Each page contains address space for a particular control data.

The packet instructions ultimately communicate the result of execution to indicate on which connection(s) the discovery packet should be *published* and whether to *subscribe* to incoming discovery packets, *connect* flow originator to a given network micoregment, or *accept* data packets coming to the flow endpoint from a given network microsegment.

Central Memory and Tables

Each processor contains a set of tables with control data that are central to instruction processing. During execution, the instructions can read and/or write in these tables using the memory pages. The tables contain isolated flow data and shared processor-level information.

Egress Traffic Management

The discovery packet content can be modified upon instruction request and sent to workloads or other processors after execution. Egress discovery packets may contain modified program packet data and path pointers in addition to standard IPv6 tweaks, such as hop count.

Support and Management

Using Southbound API (SB-API) and Secure Neighbor Discovery (SeND), the policy engine communicates with the fabric orchestrator, workloads, and other processors to support all processor functions including label, certificate and link management.

16.3.2 Datapath

The processor uses the standard Open vSwitch (OVS) as a datapath. Only discovery packets can establish packet processing rules for application flows in this datapath. Each rule is ephemeral and its expiry time is derived from an ECDSA signature TTL of the associated discovery packet. A unique authorization and forwarding logic employs two opposite-role rules to process each application packet in the datapath. The datapath executes the logic in a regular, superfast way–by matching packet headers and defining actions for them.

Policy Tables

From a high-level, the datapath comprises three types of tables with packet processing rules: Input, Output, and Group. Two global tables, Input and Output, process all unicast data packets. A set of Group tables serves discovery packets and is used instead of the Output table for multicast data packets (not shown on the diagram).

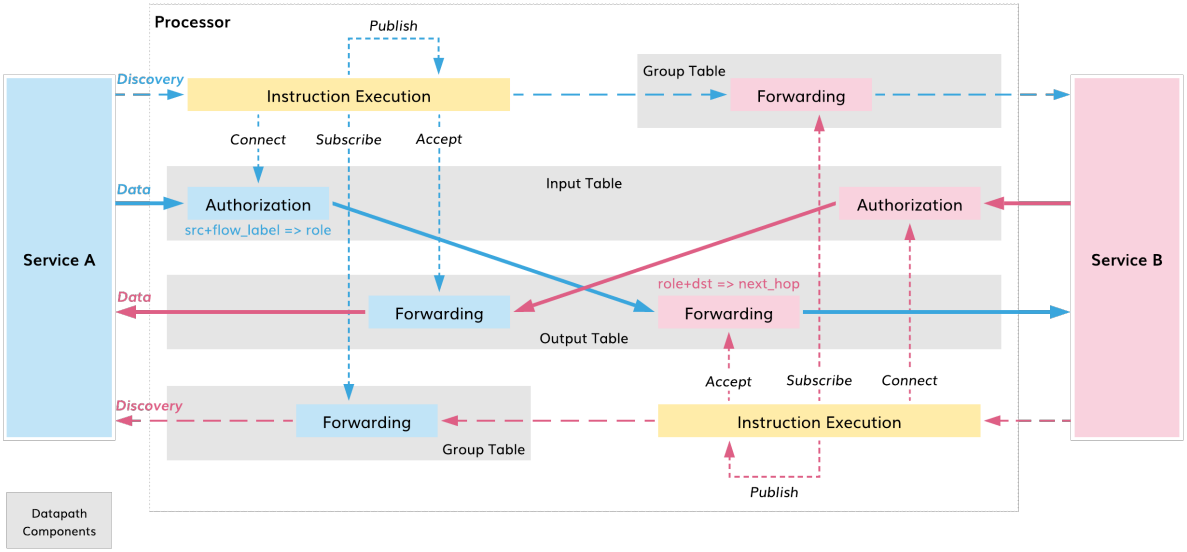


Fig. 16.4: Processor Datapath

Note: Each network microsegment has always two corresponding Group tables in the datapath.

A discovery packet can instruct the datapath to add records to any or all three tables. The Input table always receives an authorization record. The Group table associated with the discovery packet and the Output table may each conditionally receive a forwarding record.

Required Actions

Discovery packet requests are agnostic to the datapath implementation. The instruction execution outcome calls out actions in a highly abstract manner:

- *connect* the flow originator to a given network microsegment,
- *accept* data packets destined to the flow originator from a given network microsegment,
- *subscribe* to discovery packets in a given network microsegment,
- *publish* the discovery packet on a particular group of ports.

Note: A discovery packet can't pass any argument while requesting actions *connect*, *accept*, and *subscribe*. Only the action *publish()* allows the packet to specify egress ports. Upon packet request, special logic filters securely generate rules for the datapath using only controller-signed information from the discovery packet and a vector with egress ports if requested.

The action *connect* creates an authorization record in the Input table, *accept* sets up a forwarding record in the Output table, and *subscribe* installs a forwarding record in the Group table.

Authorized to Forward

As data packets arrive, the datapath authorizes them over a set of installed processing rules dropping packets that fail and forwarding the others.

The Input table matches the packet source address and flow label against a set of authorization records in order to associate the packet with a flow endpoint role in a given microsegment. The packet is dropped if the association not found.

In the next step, the Output table matches the packet originator role and destination address against a set of forwarding records. The packet is dropped if the destination is neither found nor accepting data packets with the given role.

Note: In case of a multicast data packet, the datapath sends the packet from the Input table to the Group table associated with the packet originator role. The packet is dropped if the association is not found.

This unique *authorize-to-forward* approach ensures that the datapath responds to application policy changes in real time and forwards application data packets at line-rate speed.

CHAPTER 17

Workload

17.1 Overview

From a host operating system's point of view, the Bayware Agent works on the layer 3 of OSI Model. The Agent receives IP packets on one IP interface, processes them and sends to another. Those all happen on data level. On control level, the Agent accesses controller's RESTful interface to the applications located on the host.

The Bayware Agent is responsible for:

- node initialization
- node and link registration
- service endpoint registration
- service endpoint operation

The Bayware Agent comprises of:

- control plane module
- data plane module

Control Plane Module carries out such the functions: node initialization, node and link registration, service endpoint registration. Data Plane Module performs the service endpoint operation function.

17.2 Control Plane Module

17.2.1 Module Structure

Control Plane Module comprises of:

- RSA Key Generator
- CGA Address Generator and Validator

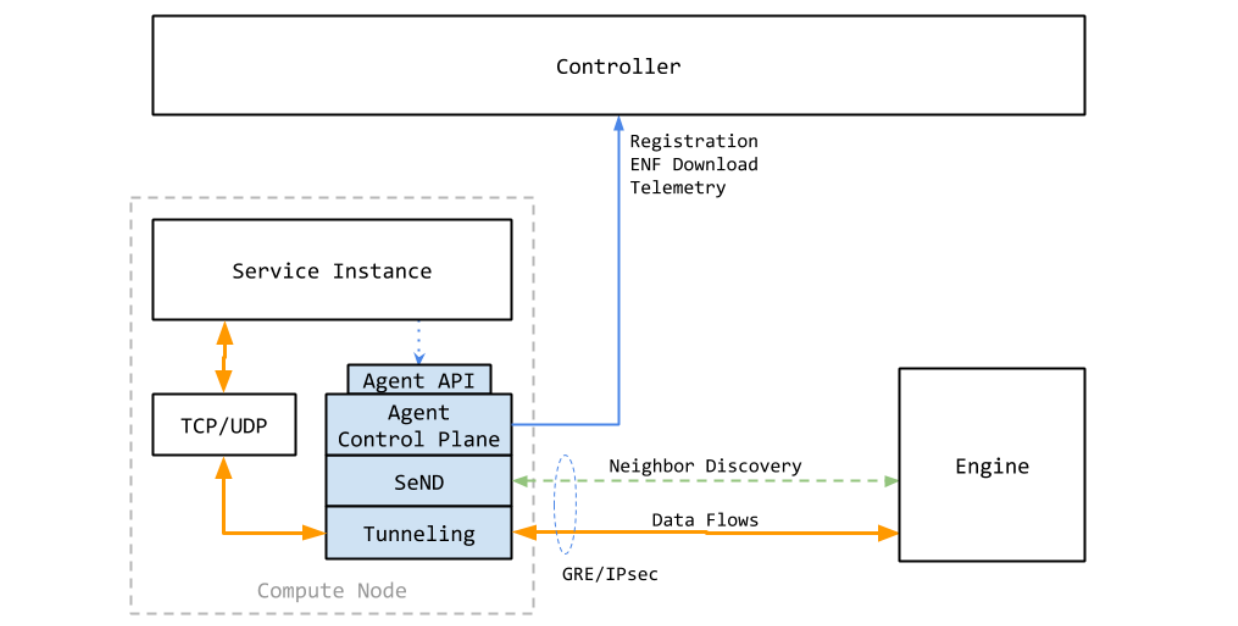


Fig. 17.1: Figure. Bayware Agent

- Neighbor Discovery block
- Control Bayware Traffic Sender/Receiver
- In/Out ACL Management block
- Host and Link Registration/Keep-alive block
- HTTP RESTful Service block
- Service Endpoint Registration block
- Host Control Data
- Service Control Data

Control Plane Module implements such the interfaces:

- two-directional Bayware Data Plane interface
- client-side Controller SB interface
- server-side Agent RESTful interface

17.2.2 Module Logic

On its start the Bayware Agent loads its configuration and sets up the logging of its activity. As configuration parameters, the Agent accepts:

- host name (Full Qualified Domain Name)
- user name
- user password
- user domain
- controller name

At this stage the Agent verifies the self-signed certificate with the local host name in the certificate subject field. If needed, the Agent generates new keys and certificate as described later.

Next, the Agent starts REST server threads. From this point the agent is ready to process application requests.

In parallel, the agent requests a controller API gateway to provide initial configuration. The API gateway responses by redirecting the agent to the controller identity service for authentication. The Agent provides the identity service with the user name, password, and domain. As a result of successful authentication the agent obtains the token that enables agent access to the controller services. Using the token, the agent requests its own user profile with the scope description.

As a part of the scope the agent receives the netprefix. The Agent uses the netprefix to check or generate the host identifier–Cryptographically Generated Address (CGA)–as described later.

When the host identifier is ready, Control Plane module checks App and Net interfaces on operating system level. Thereafter, the Control Plane module starts Data Plane module. Now, Control Plane module is able to process SeND messages in its data plane thread, as well as Type I/II packets.

After start of Data Plane module, Control Plane module requests the controller to register this host.

Upon successful host registration, Control Plane module begin dispatching, via Data Plane module, the SeND advertisement packets using the previously generated RSA keys and CGA. At the same time, Control Module receives SeND advertisement packets from neighbors, via Data Plane module as well. When a new neighbor is discovered, Control Module initiates the connection establishment algorithm execution involving SeND exchange and controller's API calls.

In parallel, Control Module monitors the host status and serves the application service requests. The module synchronizes its operational data with the controller, sending keep-alive messages periodically.

17.2.3 RSA and CGA Management

Control Plane Module is responsible for management of both RSA keys and CGA node identifier.

The RSA/CGA Management algorithm executions begins with hostname validation. The agent matches the actual host name against the name stored in the agent's configuration file. If the names don't match, the agent wipes out both RSA keys and CGA from its configuration file and generates a new RSA pair. The agent also generates a new RSA pair when the names match but RSA keys are not present in the agent configuration file.

When the pair of RSA keys are ready for use, the agent checks whether a CGA for this pair exists. If it doesn't exist or CGA verification fails (i.e. CGA is not a derivative of public key and auxiliary parameters), the agent generates a new CGA. The CGA generation is performed as per RFC3972.

17.3 Data Plane Module

Data Plane Module can be logically divided into the following functional blocks:

- policy control block
- ingress and egress packet processing blocks
- control plane send and receive blocks
- application and network interface blocks

17.3.1 Policy Control Functions

Policy Control block allows to add or remove rules of packet processing. The block includes:

- Policy Control methods
- Egress Policy database
- Ingress Policy database
- Control Data database

17.3.2 Ingress and Egress Packet Processing Functions

Ingress and Egress Packet Processing blocks process packets in accordance with the policy. The blocks receive and transmit packets on the interfaces such as:

- application interface
- network interface
- interface with control module

Packets arriving from both local applications and the control module undergo the following processing steps:

1. Egress policy is applied to the packet
2. The packet payload is encrypted

3. Optionally, IPv6 header is created or edited
4. Optionally, TCP/UDP checksum is recalculated
5. Ethernet header is created or edited

Packets arriving from both network and control module undergo the following processing steps:

1. Ethernet header is removed
2. Ingress policy is applied to the packet
3. The packet payload is decrypted
4. Optionally, IPv6 header is replaced or edited
5. TCP/UDP checksum is recalculated

IPv6 ICMP packets proceed between control module and network interfaces without being processed inside data plane module.

17.3.3 ontrol Plane send and receive functions

Data Plane module forwards packets between Control Plane module and network interface.

The packets from Control Module are sent to:

- Egress Policy applicator, if IPv6 SSM
- Net Egress Queue, if IPv6 ICMP

The packets to Control Module are received by Control Egress Queue from:

- Net Interface Listener, if IPv6 ICMP
- Ingress Policy applicator, if IPv6 SSM

17.3.4 Application and network interface functions

Data Plane module connects to Virtual TUN Interface for receiving and sending packets from/to local applications.

Data Plane module connects to Virtual GRE-TAP Interface for receiving and sending packets from/to network.

CHAPTER 18

System Requirements

With the service interconnection fabric, you can easily stretch your compute environment from a private data center to a public cloud, over different public clouds, or across application clusters in the same cloud. This approach doesn't require changes to your application code nor a long learning curve of public cloud APIs. Once you have deployed a fabric, you can easily move application services from one workload node to another, scale workload nodes in and out, or redirect traffic between clouds for failover and cost optimization.

To make the cloud migration process agile, secure and flexible, the service interconnection fabric deployment itself is designed to be highly automated and simple. You can deploy all service interconnection fabric components–VPCs and nodes (i.e., orchestrators, processors, and workloads)–either using the fabric manager software or your own automation tools.

It is common to use the fabric manager for the deployment of all fabric components in the public clouds. In this case, you utilize public cloud automation in full and significantly simplify your deployment. All integration with cloud services and between fabric components is done automatically by the fabric manager.

In private data centers, the installation of processor and workload software components on already existing VMs or physical servers–without access from fabric manager–might be the only available option. The good news is that the installation and configuration of each component takes about a minute and can be easily added to your existing provisioning pipeline. However with this approach, the certificate, telemetry, events, and software management of the components–covered by the fabric manager in public clouds–should be added to your private data center automation tools.

Alternatively, you can get the best of both worlds with a deployment approach that combines the fabric manager automation power with the flexibility of your own VM management tools. If you are interested in bringing your own VMs to the fabric manager, we can do that also.

The following guides will discuss currently available options and lead you through all the steps of deploying:

- fabric manager in a public cloud,
- orchestrator in a public cloud,
- processor in a public cloud or a private data center,
- workload in a public cloud or a private data center.

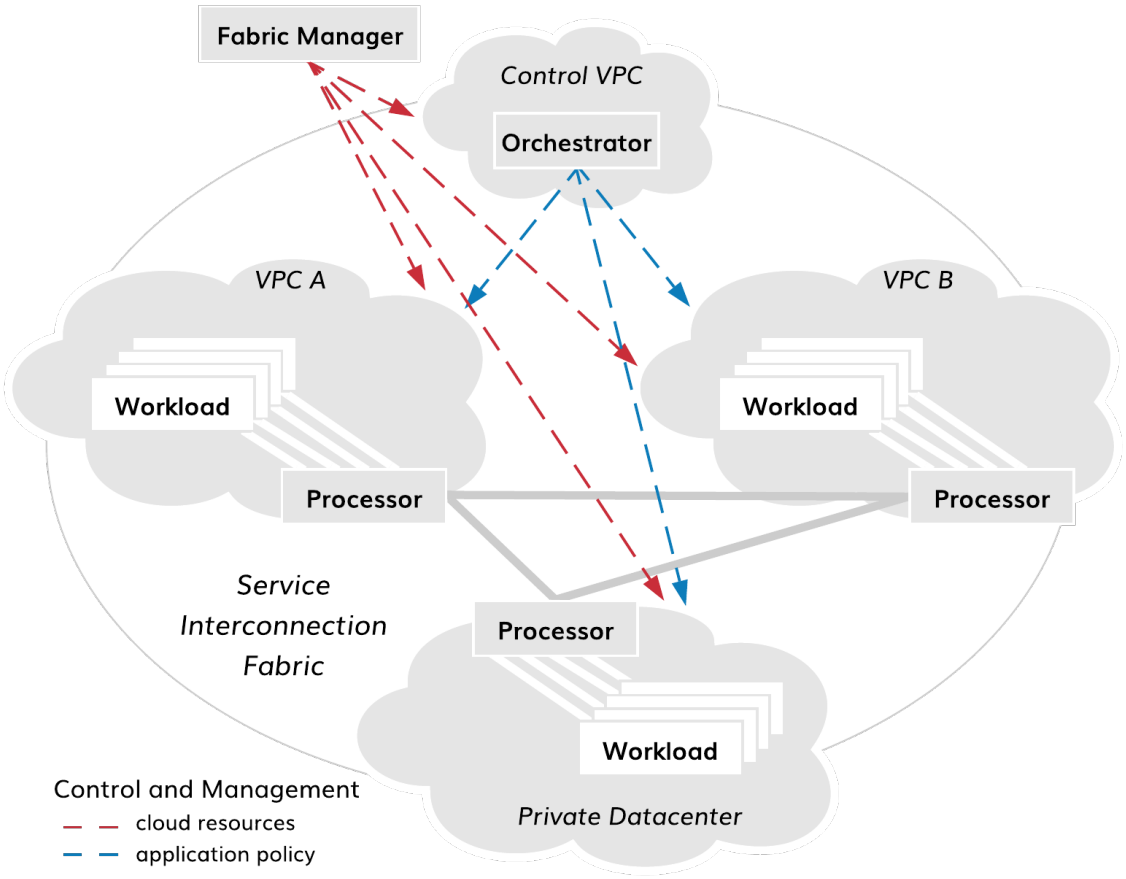


Fig. 18.1: Fabric Deployment Diagram

18.1 Server Requirements

The fabric manager is supplied with default resource templates which specify types of virtual machines for each cloud based on their roles in the service interconnection fabric.

The following table lists minimum requirements for installing and operating service interconnection fabric components on virtual machines or physical servers.

Component	OS	Proc Cores	RAM	Storage
Fabric Manager	Ubuntu 18.04 LTS or later	2	4GB	10GB
Orchestrator	Ubuntu 18.04 LTS or later[1]	4	16GB	1TB
Processor	Ubuntu 18.04 LTS or later RHEL 8 or later[2]	2	4GB	60GB
Workload	Ubuntu 18.04 LTS or later RHEL 8 or later[3]	2	4GB	10GB

18.2 Firewall Settings

When you create a VPC and a VM in it with the fabric manager, all security groups set up automatically based on the VM role. In case you install service interconnection fabric components using your own automation tool, the tool must provision particular security rules to allow the components communicate within the service interconnection fabric.

There are five distinctive sets of security rules in the service interconnection fabric:

- fabric manager,
- orchestrator ingress proxy,
- orchestrator nodes,
- processor nodes,
- workload nodes.

The purpose of fabric manager and node groups is obvious. The ingress proxy security group contains the subset of ports open to the internet that are associated with the orchestrator. Today, ingress proxy is part of orchestrator's controller node, in which case the ingress proxy security group and the orchestrator security group can both be applied to the controller node. The ingress proxy security group opens tcp/80, tcp/443, and tcp/5044 to the internet (as well as ICMP and tcp/22 from the fabric manager). The orchestrator security group should only open ports to the local subnet as required for Docker, telemetry, events, etc. (as well as icmp and tcp/22 from the fabric manager). In the future, you may deploy the ingress proxy on a separate VM within the orchestrator VPC/subnet, in which case the ingress proxy security would apply only to the ingress proxy node only.

The security group rules are summarized in table below.

Security Group Rules

[1] Please note that the orchestrator is deployed in Docker containers and may run across one or more nodes.
[2] Deploying the processor on RHEL 8 is supported starting with the fabric family version 1.4.
[3] Deploying the processor on RHEL 8 is supported starting with the fabric family version 1.3.

Protocol	Fabric Manager	Orchestrator Ingress Proxy	Orchestrator Nodes	Processor Nodes	Workload Nodes
SSH	from Internet	YES - from FM only	YES - from FM only	YES - from FM only	YES - from processors in the same VPC
ICMP	from Internet	YES - from FM only	YES - from FM only	YES - from FM only	YES - from processors in the same VPC
IPsec (udp/500; udp/4500)	NO	NO	NO	YES - from workloads in the same VPC and all processors	YES - from processors in the same VPC
HTTPS	NO	YES - from Internet	NO	NO	NO
HTTP	NO	YES - from Internet[4]	NO	NO	NO
Logs (tcp/5044)	NO	YES - from Internet	NO	NO	NO

Note: When you deploy service interconnection fabric components in your private data center, use the same security rules to provision your data center firewalls.

18.3 Public Cloud VM Setup

While the fabric manager sets up VMs fully automatically, there are a few details to keep in mind when configuring a VM in a public cloud for hosting processor and workload software components with your own automation tools. This next section comments on those points generally and subsequent sections reference public cloud providers more specifically.

This guide does not replace public cloud provider documentation. You should follow the steps provided by your cloud provider when spinning up a VM while ensuring VMs meet service interconnection fabric requirements.

18.3.1 Engine Installation Requirements

Policy engine, or simply engine, is a software component that makes your VM a **processor node**. The processor node secures one or multiple sets of workload nodes. All communication between workloads in different clouds happen through the processor nodes using IPsec with NAT traversal by default. The processor node communicates with the fabric orchestrator using HTTPS with mTLS to enforce resource policy. As such, the VM hosting the policy engine software must satisfy particular requirements.

Public IP Address

The orchestrator must be reachable by the policy engine via IP. If this is possible with a private IP address, a public IP address is not required.

[4] Required for the Let's Encrypt certificate setup only.

Also, other policy engines must be reachable by the policy engine via IP. If this is possible with a private IP address, a public IP address is not required.

A policy engine VM's public IP address must be static.

Private IP Address

The policy engine must be reachable by policy agents located in its security zones via IP. If this is possible with a private IP address, use it as a preferable method.

A policy engine VM's private IP address must be static.

IP Forwarding

IP Forwarding must be enabled on all policy engine VMs.

Protocol Filtering

Ingress ports udp/4500 and udp/500 must be opened up on firewalls for all VMs hosting policy engines.

VM Name and Certificate

All VMs in the service interconnection fabric are identified by VM names specified in x509 node certificates (see Certificate Requirements). Modifying a cloud VM name–whenever it's possible–requires processor node certificate to be regenerated and the node to be re-registered in the fabric.

Tips for Engine Installation

Amazon Web Services

Follow AWS instructions for using public and private IP addresses and opening firewall ports.

To enable IP Forwarding, select the VM instance then navigate to `Actions->Networking->Change Source/Dst`. Check. `Disable source/destination` Check.

Microsoft Azure

Follow Azure instructions for using public and private IP addresses and opening firewall ports.

Enable IP Forwarding after instance creation by selecting the interface and going to its IP Configuration menu. Turn `IP Forwarding` ON.

Google Cloud Platform

GCP requires all recommended settings to be configured during instance creation.

18.3.2 Agent Installation Requirements

Policy agent, or simply agent, is a software component that makes your VM a **workload node**. The workload node secures communication for hosted application services. The workload node communicates with other workload nodes through processor nodes using IPsec with NAT traversal by default. Also, the workload node communicates with the fabric orchestrator using HTTPS with mTLS to enforce resource and application policy. As such, the VM hosting the policy agent software must satisfy particular requirements.

Public IP Address

The orchestrator must be reachable by the policy agent via IP. If this is possible with a private IP address, a public IP address is not required.

Any required public IP address may be dynamic.

Private IP Address

At least one policy engine must be reachable by the policy agent via IP. If this is possible with a private IP address, use it as a preferable method.

A policy agent VM's private IP address must be static.

IP Forwarding

IP Forwarding is generally disabled by default on new images. VMs with policy agents do not require IP Forwarding so it should be disabled.

Protocol Filtering

Ingress ports udp/4500 and udp/500 must be opened up on firewalls for all VMs hosting policy agents.

VM Name

All VMs in the service interconnection fabric are identified by VM names specified in x509 node certificates (see Certificate Requirements). Modifying a cloud VM name–whenever it's possible–requires workload node certificate to be regenerated and the node to be re-registered in the fabric.

Tips for Agent Installation

Amazon Web Services

Follow AWS instructions for using public and private IP addresses and opening firewall ports.

Microsoft Azure

Follow Azure instructions for using public and private IP addresses and opening firewall ports.

Google Cloud Platform

GCP requires all recommended settings to be configured during instance creation.

18.4 Private Datacenter VM Setup

You can install processor and workload software in your private data center on any machine that meets minimum requirements. The VM setup in a private data center is identical to the public VM setup for both processor and workload nodes.

A VM in private data center that hosts either policy engine or agent requires:

- a valid x509 node certificate (see Certificate Requirements);
- a static public IP if the machine communicates with service interconnection fabric nodes–processors or workloads–reachable via public IPs only;
- a static private IP if the machine communicates with service interconnection fabric nodes–processors or workloads–reachable via private IPs;
- ingress ports `udp/4500` and `udp/500` opened on VM and network firewalls;
- egress port `tcp/443` opened on VM and network firewalls.

18.5 Certificate Requirements

All communication in the service interconnection fabric is based on service identity and resource identity. While service authorization tokens provide application-level security, x509 certificates secure communications at resource level.

By default, the fabric manager plays Root CA role. When you deploy the fabric interconnection components with the fabric manager, all required certificates are generated and provisioned automatically, including a Let's Encrypt certificate for the orchestrator northbound interface.

If you want to use your own PKI system or to deploy policy engine or agent on a VM without access from the fabric manager, you can generate certificates by following these instructions.

18.5.1 Using Certificates

The service interconnection fabric relies on the certificates as follows:

- orchestrator NB-API certificate,
- orchestrator SB-API certificate,
- orchestrator Flow-Sign certificate,
- node certificate.

The orchestrator in the service interconnection fabric terminates TLS and mTLS sessions for the northbound and southbound interface respectively. Each interface requires a certificate associated with its FQDN:

- NBI – `orchestrator-<fabric>.<fm-name>.<company-domain>`
- SBI – `controller-<fabric>.<fm-name>.<company-domain>`

In order to authorize data flows between application services, the orchestrator requires a Flow-Sign certificate. All processor and workload nodes use this Flow-Sign certificate to validate flow signature.

Each processor and workload node must possess a node certificate to operate in the service interconnection fabric. The node certificate is used to:

- identify node name and type,
- set up mTLS channel with orchestrator,
- set up IPsec link between nodes,
- discover adjacent nodes using Secure Neighbor Discovery (SeND),
- create node Cryptographically Generated Address (CGA).

The certificates can be provisioned using an existing PKI system or the fabric manager.

Note: When used as Root CA, the fabric manager generates an RSA key and an associated certificate signing request (CSR) on each node. Next, the fabric manager retrieves the CSR and returns a signed certificate back to the node. Private keys never leave the nodes, on which they have been generated.

18.5.2 Generating Node Certificate

Generating Key and CSR

Go to your processor or workload node and ensure you are in the **certificate directory** as specified in your agent or engine configuration. By default, it's `~/opt/bayware/certs/`:

```
]$ cd ~/opt/bayware/certs/
```

To generate an RSA key, run this command on your processor or workload node:

```
]$ openssl genrsa -out node.key 2048
```

To generate a certificate signing request, run this command on your processor or workload node with the company name, fabric name, node role, and hostname as a subject, in this example - `'/O=myorg2/DC=myfab2/DC=processor/CN=aws-p01-myfab2'`:

```
]$ openssl req -new -key node.key -out node.csr -subj '/O=myorg2/DC=myfab2/DC=processor/
↪CN=aws1-p01-myfab2'
```

Warning: As a reminder, be sure to cross-check the input values for each subject field.

- O = company name from your fabric manager configuration;
- DC appears twice -
  - DC = fabric name,
  - DC = processor | workload | orchestrator;
- CN = hostname of the VM that issues the CSR.

Copy the CSR you have just generated–in this example `node.csr` –to your fabric manager.

Note: On the fabric manager, place the CSR in the **fabric directory**, in this example: `~/.bwctl/myfab2`

Issuing Certificate

Go to your fabric manager and ensure you are in the correct **fabric directory**, in this example `~/.bwctl/myfab2`:

```
]$ cd ~/.bwctl/myfab2
```

Note: The current working directory–in this example `~/.bwctl/myfab2` –should now contain `node.csr` file among others.

Run the following openssl command on your fabric manager node to generate the x509 certificate from the CSR:

```
]$ openssl x509 -req -in node.csr -sha256 -days 3650 -out node.crt -CAkey rootca.key -CA␣
↪rootca.crt -CAcreateserial -CAserial node.srl -extfile usr_crt_extensions.ext
```

Example of expected output after running the command:

```
Signature ok
subject=O = myorg2, DC = myfab2, DC = processor, CN = aws-p01-myfab2
Getting CA Private Key
```

Deploying Key and Certificates

In the last step, copy the node and root CA certificates (in this example, `node.crt` and `rootca.crt`) to the node–from which you received the CSR–in the folder you have specified in the policy engine or agent configuration. Also, move into this folder the RSA ke file (in this example, `node.key`) if you happened to create it elsewhere.

Note: By default, the policy engine and agent works with the certificate and the private key located at `~/opt/bayware/certs/`

This is the content of the folder on your processor or workload node you should see at this step:

```
-r-------- 1 ubuntu   ubuntu 1956 Oct  4 15:28 rootca.crt
-rw-rw-r-- 1 ubuntu   ubuntu 1696 Oct  4 15:28 node.crt
-rw------- 1 ubuntu   ubuntu 1675 Oct  4 15:28 node.key
```

18.5.3 node.crt (example)

Here is an example of a processor node certificate:

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number:
          a7:13:91:c4:dc:ec:e7:86
   Signature Algorithm: sha256WithRSAEncryption
       Issuer: O = "myorg2", DC = myfab2, CN = green-c0
       Validity
           Not Before: Mar 15 22:56:53 2019 GMT
           Not After : Mar 12 22:56:53 2029 GMT
       Subject: O = "myorg2", DC = myfab2, DC = processor, CN = aws1-p01-fab2
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
               Public-Key: (2048 bit)
               Modulus:
                   00:de:fd:58:db:8b:4e:7e:fa:59:e3:0e:b1:21:1a:
                   a6:3f:f9:73:12:df:e4:46:e9:d7:f0:d4:88:f5:ad:
                   f2:81:16:18:98:53:87:b9:ae:37:e9:70:75:f2:24:
                   94:46:cd:56:db:29:4d:c4:a8:d8:93:77:d0:ac:2e:
                   fa:e1:43:11:c1:73:d6:1e:56:50:4e:15:03:ae:9e:
                   2f:fe:df:40:1e:da:aa:5e:e4:25:a0:29:1b:3f:87:
                   2c:81:48:cd:0b:40:78:9a:d4:f0:a5:4a:45:b8:50:
                   b2:7b:a5:43:a7:b9:10:40:d7:94:cd:fa:15:43:d2:
                   dd:54:bf:29:f3:a4:bf:9d:6d:56:2e:ca:3b:c3:82:
                   d3:c8:90:5a:4d:51:52:86:97:d9:85:51:44:62:55:
                   5b:06:dc:5c:2b:54:e3:a9:64:00:65:71:3d:8e:c3:
                   75:2a:9d:f0:94:47:7b:7b:e6:83:4a:6b:e5:09:59:
                   d2:8d:3f:46:32:cc:91:28:35:c5:4f:ae:bc:54:fb:
                   fe:7e:63:c7:d9:69:a6:ff:5b:d9:3a:32:9c:51:25:
                   15:61:a1:5c:95:bf:57:3a:62:f4:03:c1:f3:fc:bd:
                   ad:79:cd:e9:d9:62:ea:dd:c6:ad:65:d8:d8:73:46:
                   3e:38:e0:3e:23:62:b8:19:b2:44:e8:c4:ae:39:3c:
                   46:4d
               Exponent: 65537 (0x10001)
       X509v3 extensions:
           X509v3 Key Usage: critical
               Digital Signature, Key Encipherment
           X509v3 Extended Key Usage:
               TLS Web Server Authentication, TLS Web Client Authentication
           X509v3 Basic Constraints: critical
               CA:FALSE
           X509v3 Subject Key Identifier:
               DB:A2:5E:C8:32:4B:53:5B:D1:A0:49:78:18:B9:E7:71:FB:9D:6F:0B
           X509v3 Authority Key Identifier:
               keyid:50:C9:A4:B9:8E:8A:68:98:D6:AD:AB:6C:99:AB:72:29:C4:3B:98:CC
   Signature Algorithm: sha256WithRSAEncryption
        29:fc:07:b2:00:14:05:ea:22:22:c5:e7:6d:8e:8a:5e:8c:59:
        a7:c4:5d:54:94:a6:24:9e:02:1a:d9:58:25:c9:fa:69:77:bd:
        db:91:17:93:49:70:08:ae:af:c2:c1:9c:8a:35:b4:ce:ca:65:
        91:77:22:cf:89:42:17:ce:f3:f0:29:7a:38:3c:b6:f1:06:a0:
        b4:ae:5c:de:60:08:2a:e9:84:dd:5e:84:70:bf:5c:9e:1f:f7:
        5d:62:85:17:ba:10:2e:6d:34:75:3f:9f:70:e4:10:46:59:60:
```

(continues on next page)

(continued from previous page)

```
        ff:93:b7:c7:22:6e:d2:3c:58:68:75:68:b7:fe:9b:7c:f2:69:
        64:83:af:15:80:80:04:35:c1:05:80:f9:a2:bd:1c:67:93:5d:
        3d:fc:1d:cd:86:fd:ae:6e:9b:3a:22:7a:ad:1d:c6:dc:b4:ee:
        ae:5c:69:0b:1a:1f:5b:e3:58:20:b8:bd:bf:ab:a7:bd:cb:e6:
        38:ee:12:ad:96:83:96:c8:2a:e7:55:47:68:b6:25:7a:be:1b:
        36:48:0d:da:4c:8f:79:7e:ef:4f:bf:fc:05:f7:01:7f:9c:e7:
        b1:13:f2:6e:c9:d1:6f:6a:85:16:f8:d1:5c:83:ff:f1:ba:70:
        89:9d:02:e6:54:e1:7f:5c:0e:a4:ef:7e:3d:9d:03:c3:6a:80:
        34:5c:6a:f6:52:0c:19:ba:98:08:b6:47:b5:91:7e:fd:98:d5:
        8e:9a:ba:b7:bf:39:11:52:4a:26:cb:26:56:65:a3:e0:ca:05:
        04:29:24:e4:86:88:3a:15:e6:d1:dd:48:e7:f1:f6:31:68:3e:
        2d:81:8a:05:a1:1f:31:12:a8:6d:a0:38:ed:af:9e:d2:a4:c0:
        40:bf:49:d1:e5:d5:ee:28:c7:8d:4d:23:27:ee:74:d5:ca:4b:
        ae:ff:61:22:21:07:75:6d:db:de:b3:6c:46:f3:fb:11:6f:28:
        e4:98:fa:f7:b0:6e:64:a1:be:0d:ee:e6:64:73:90:e9:bc:b6:
        4d:3b:94:e6:c7:71:5c:9c:1b:67:c2:d0:19:89:1d:f1:76:16:
        ca:f0:b9:39:81:e2:96:7d:fa:7a:cd:9f:90:7a:1b:f7:3e:7d:
        db:43:bb:c4:79:d0:d9:0c:0c:f5:39:93:63:46:ba:5b:af:8e:
        5d:32:f4:1d:b1:84:cc:bc:45:5c:57:4c:c9:15:e0:fa:f2:37:
        23:fb:f8:3a:de:83:1c:c9:0b:8f:80:b0:10:b3:fc:03:e4:e6:
        f3:e1:8a:77:73:44:c2:71:7f:52:d9:c1:a6:b2:a6:63:f6:97:
        f2:c1:de:c1:06:d8:ae:de:1f:2b:4d:c1:c0:2f:88:2d:c4:1b:
        37:bd:c2:2f:2e:2e:9a:2f
```

CHAPTER 19

Deploying Fabric Manager

19.1 Spin up Fabric Manager

The fabric manager software allows you to manage resources in three public clouds:

- Microsoft Azure,
- Amazon Web Services (AWS),
- Google Cloud.

You can spin up a virtual machine with the fabric manager software in either Azure or AWS. By default, the fabric manager uses AWS S3 service to store its backup files and AWS Route 53 service for the hosting of orchestrator domain names.

Note: Setting up the fabric manager in either Azure or AWS provides you with the same set of capabilities for resource management in three clouds.

19.1.1 Microsoft Azure

To spin up the fabric manager from Azure Marketplace, simply search for Bayware and click on the "Get It Now" button to begin the process. As you fill out the required Azure forms, keep in mind that Bayware recommends using B2s machine type.

19.1.2 Amazon Web Services (AWS)

To spin up the fabric manager in AWS, sign into the AWS Console, select `Services > EC2` on the left side of the page and ensure you are in your desired AWS region shown in the upper right.

Now click the `Launch Instance` button. On the subsequent page, select `Community AMIs` and type the search box `bayware-c0` to find the latest fabric manager image.

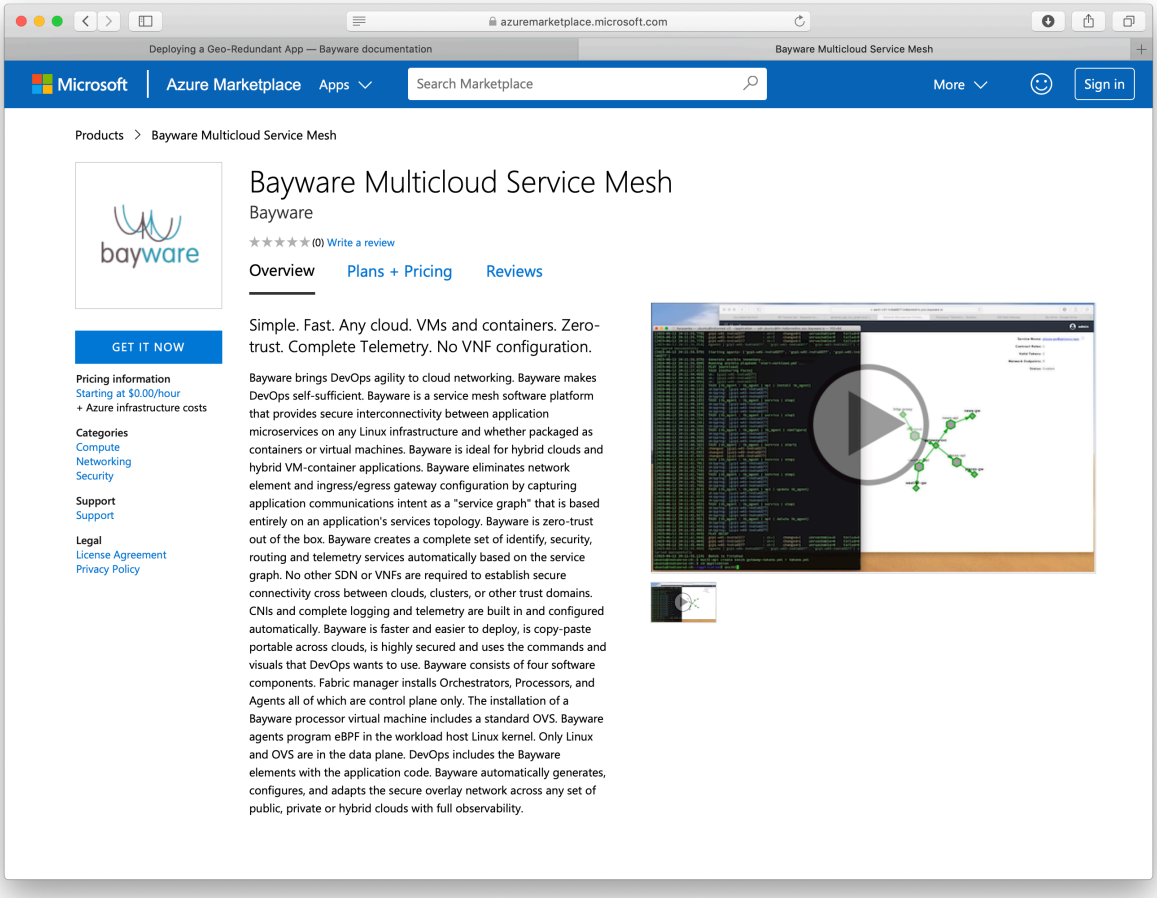


Fig. 19.1: Azure Fabric Manager marketplace offering

Note: The fabric manager image has always the name built as follows: `bayware-c0-<version>`,where the version comprises three parts – two with a family number and one image version within the family, for example `bayware-c0-v1-2-8`.

Select the image and continue through the rest of the AWS launch process. We recommend using t2.medium as machine type.

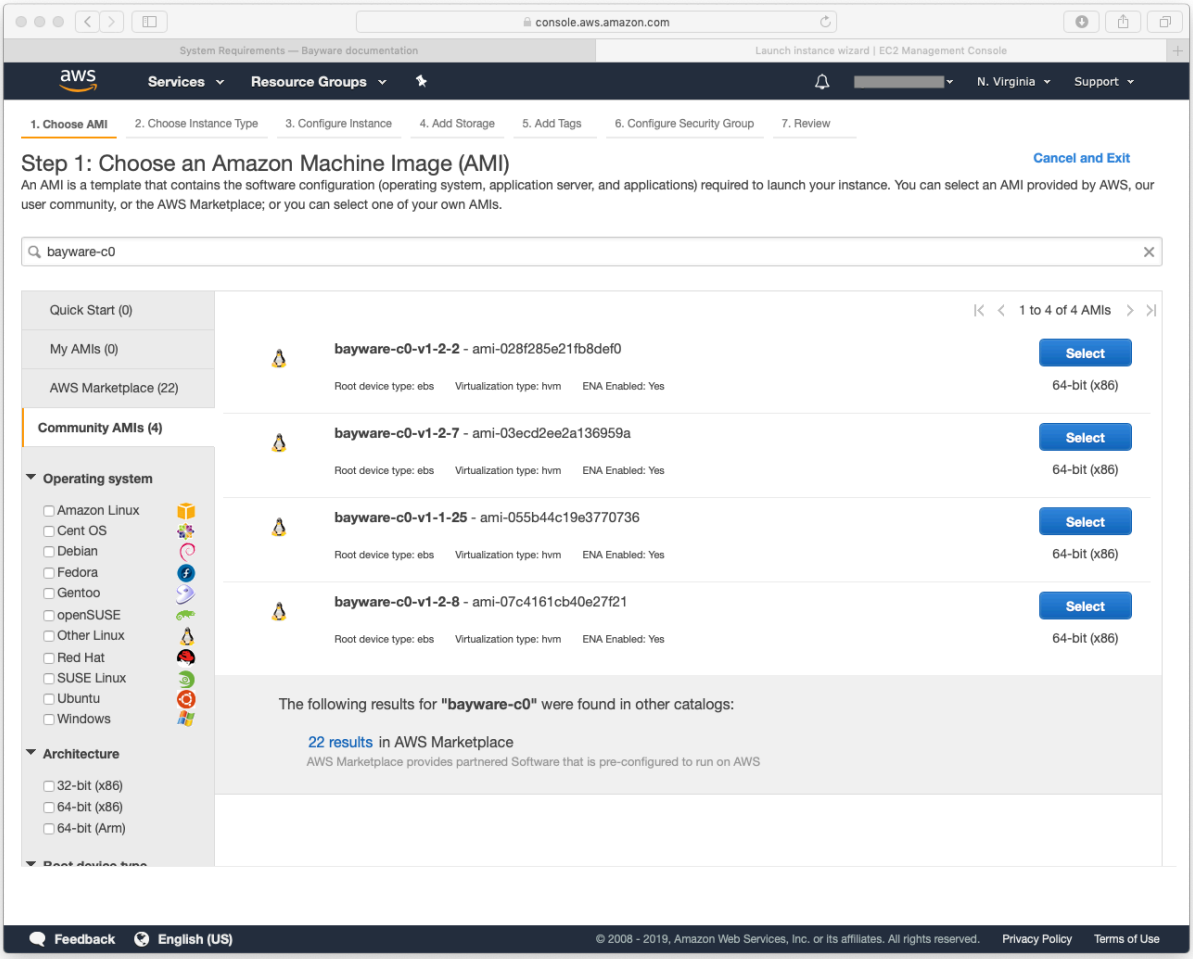


Fig. 19.2: AWS Fabric Manager Community AMI offering

19.2 Update BWCTL CLI Tool

Upon successfully completing the creation of the new VM image, it is time to update all necessary packages and dependencies for BWCTL. To do this, you will need to `SSH` into your newly created VM and switch to root level access to update all packages as such:

```
]$ sudo su -
```

Next, to update BWCTL, run the command:

```
]# pip3 install --upgrade bwctl
```

To update the BWCTL-resources package, run the command:

```
]# pip3 install --upgrade bwctl-resources
```

To exit from the current command prompt once you have completed updating, run the command:

```
]# exit
```

19.3 Configure BWCTL

Next, it's time to create the BWCTL environment in the home directory of the current user (`ubuntu`).

First, start BWCTL running the command:

```
]$ bwctl init
```

You should see this output:

```
[2019-09-25 17:30:12.156]  Welcome to bwctl initialization
[2019-09-25 17:30:12.156]  Fabric manager
[2019-09-25 17:30:12.156]   Company name (value is required):
```

In interactive mode, provide all required values when prompted.

Note: Press `<Enter>` to accept the default values.

After the initialization you should have a configuration similar to:

```
[2019-09-25 17:30:12.156]  Welcome to bwctl initialization
[2019-09-25 17:30:12.156]  Fabric manager
[2019-09-25 17:30:12.156]   Company name (value is required): myorg3
[2019-09-25 17:30:30.113]  Global
[2019-09-25 17:30:30.113]   Cloud providers credentials file [~/.bwctl/credentials.yml]:
[2019-09-25 17:30:34.004]   DNS hosted zone (value is required): poc.bayware.io
[2019-09-25 17:30:37.325]   Debug enabled [true]:
[2019-09-25 17:30:42.062]   Production mode enabled [true]:
[2019-09-25 17:30:44.548]   Marketplace images to be used [false]:
[2019-09-25 17:30:48.624]  Components
[2019-09-25 17:30:48.624]   Family version [1.2]:
[2019-09-25 17:30:51.959]  Cloud storage
[2019-09-25 17:30:51.959]   Store bwctl state on AWS S3 [false]:
[2019-09-25 17:30:58.786]   Store terraform state on AWS S3 [true]:
[2019-09-25 17:31:05.633]    AWS S3 bucket name [terraform-states-sandboxes]:
[2019-09-25 17:31:12.933]    AWS region [us-west-1]:
[2019-09-25 17:31:15.876]  SSH keys
[2019-09-25 17:31:15.876]   SSH Private key file []:
[2019-09-25 17:31:21.268]  Configuration is done
```

To view the file with your cloud provider credentials, cat to where the cloud `credentials.yml` file was specified during the initialization by running the command with the path to the file–in this example `/home/ubuntu/.bwctl/credentials.yml` –as argument:

```
]$ cd /home/ubuntu/.bwctl/credentials.yml
```

You should see this output:

```
---

# Add cloud-provider credentials that will be used when creating
# infrastructure and accessing repositories.

aws:
  # In the AWS console, select the IAM service for managing users and keys.
  # Select Users, and then Add User.  Type in a user name and check
  # programmatic access.  Users require access to EC2, S3, and Route53.
  # Copy and paste the secret access key and key ID here.
  aws_secret_access_key:
  aws_access_key_id:
azr:
  # Azure provides detailed steps for generating required credentials
  # on the command line, which you can find at this URL:
  # https://docs.microsoft.com/en-us/azure/virtual-machines/linux/terraform-install-
↪configure#set-up-terraform-access-to-azure
  azr_client_id:
  azr_client_secret:
  azr_resource_group_name:
  azr_subscription_id:
  azr_tennant_id:
gcp:
  # Google uses a GCP Service Account that is granted a limited set of
  # IAM permissions for generating infrastructure.  From the IAM & Admin
  # page, select the service account to use and then click "create key"
  # in the drop-down menu on the right.  The JSON file will be downloaded
  # to your computer.  Put the path to that file here.
  google_cloud_keyfile_json:
```

Use your editor of choice (ex: vim, nano) to add your public cloud credentials to `credentials.yml`.

19.4 Create Fabric

The next step is to create a fabric. The fabric acts as a namespace into which your infrastructure components will be deployed.

Note: The fabric manager allows you to create multiple fabrics to isolate various applications or different environments.

To get started, SSH into your Fabric Manager VM and enter the BWCTL command prompt:

```
]$ bwctl
```

You should be at the `bwctl` prompt:

```
(None) bwctl>
```

Now, to create a new fabric, run the command with your fabric name–in this example `myfab2` –as the argument:

```
(None) bwctl> create fabric myfab2
```

You should see output similar to:

```
[2019-09-25 17:33:24.563]  Creating fabric: myfab2...
...
[2019-09-25 17:33:29.901]  Fabric 'myfab21' created successfully
```

To configure the fabric, run the command with your organization name–in this example `myorg2` –as the argument:

```
(None) bwctl> configure fabric myfab2
```

You should see output similar to:

```
[2019-09-25 17:34:29.730]  Install CA for fabric 'myfab2'
...
[2019-09-25 17:34:36.859  Fabric 'myfab2' configured successfully
```

To verify the new fabric has been created with the argument provided, run the command:

```
(None) bwctl> show fabric
```

You should see output similar to:

```
[2019-09-25 17:35:50.356]  Available fabrics listed.  Use "bwctl set fabric FABRIC_NAME"␣
↪to select fabric.
  FABRIC
  myfab2
```

Now, set BWCTL to the new fabric by running this command:

```
(None) bwctl> set fabric myfab2
```

You should see output similar to:

```
[2019-09-25 17:36:22.476]  Active fabric: 'myfab2'
```

Notice that your `bwctl` prompt has changed, now showing the active fabric:

```
(myfab2) bwctl>
```

CHAPTER 20

Deploying Orchestrator

You can deploy the orchestrator in any cloud using your fabric manager. From the same fabric manager you can set up multiple fabrics–e.g. one for test environment and another for production–and place each fabric orchestrator in a different cloud or cloud region.

Note: Each service interconnection fabric requires its own orchestrator due to security and availability reasons.

The orchestrator is a microservice-based application itself that runs in Docker containers on one or multiple VMs. When set up in a three node configuration, each orchestrator node plays a role as follows:

- **controller** node (NB-API, SB-API, resource and application policy);
- **telemetry** node (InfluxDB and Grafana);
- **events** node (ElasticSearch, Logstash, and Kibana).

The controller node is the only mandatory component of the orchestrator deployment.

To begin, SSH to your fabric manager.

Set BWCTL to the fabric, in which you need to deploy the orchestrator, by running this command with the fabric name–in this example `myfab2` –as argument:

```
]$ bwctl set fabric myfab2
```

20.1 Create VPC

After the fabric set, you can create a VPC for hosting of your orchestrator nodes in this fabric.

Note: It is recommended to use a dedicated VPC for the orchestrator deployment only.

Once you are in the BWCTL command prompt, show a list of available VPC regions by running this command:

```
(myfab2) bwctl> show vpc --regions
```

You should see the list of the regions, in which you can create your VPC, similar to:

```
aws:
 ap-east-1
 ap-northeast-1
 ap-northeast-2
 ap-south-1
 ap-southeast-1
 ap-southeast-2
 ca-central-1
 eu-central-1
 eu-north-1
 eu-west-1
 eu-west-2
 eu-west-3
 sa-east-1
 us-east-1
 us-east-2
 us-west-1
 us-west-2
azr:
 australiaeast
 australiasoutheast
 brazilsouth
 canadacentral
 centralindia
 centralus
 eastasia
 eastus
 eastus2
 japaneast
 northcentralus
 northeurope
 southcentralus
 southeastasia
 southindia
 westcentralus
 westeurope
 westus
 westus2
gcp:
 asia-east1
 asia-east2
 asia-northeast1
 asia-northeast2
 asia-south1
 asia-southeast1
 australia-southeast1
 europe-north1
```

(continues on next page)

(continued from previous page)

```
europe-west1
europe-west2
europe-west3
europe-west4
europe-west6
northamerica-northeast1
southamerica-east1
us-central1
us-east1
us-east4
us-west1
us-west2
```

Now, to create a new VPC for orchestrator nodes, run the command with the cloud and region names–in this example azr and westus, respectively, as an argument:

```
]$ bwctl> create vpc azr westus
```

You should see output similar to:

```
[2019-09-25 17:36:58.649]  Creating VPC: azr1-vpc-myfab2...
...
[2019-09-25 17:38:26.089]  VPCs ['azr1-vpc-myfab2'] created successfully
```

Note: The VPC name has been autogenerated. Use this name from the command output at the next step.

20.2 Create Controller Node

To create a controller node for the orchestrator, run this command with the orchestrator VPC name–in this example `azr1-vpc-myfab2` –as argument:

```
]$ bwctl> create orchestrator controller azr1-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 17:39:48.091]  Creating new orchestrator 'azr1-c01-myfab2'...
...
[2019-09-25 17:43:56.811]  ['azr1-c01-myfab2'] created successfully
[2019-09-25 17:43:56.840]  Generating SSH config...
```

Note: The orchestrator node name has been autogenerated. Use this name at the next step.

Next, configure the orchestrator node by running this command with the orchestrator node name–in this example `azr1-c01-myfab2` –as argument:

```
]$ bwctl> configure orchestrator azr1-c01-myfab2
```

You should see output similar to:

```
[2019-09-25 17:44:38.177]  Setup/check swarm manager on orchestrator 'azr1-c01-myfab2'
...
[2019-09-25 17:50:14.166]  Orchestrators: ['azr1-c01-myfab2'] configured successfully
[2019-09-25 17:50:14.166]  IMPORTANT: Here is administrator's password that was used to␣
↪initialize controller. Please change it after first login
[2019-09-25 17:50:14.166]  Password: RWpoi5RkMDBi
```

Warning: Be sure to write down the **PASSWORD** as it appears on your screen, it will be needed later.

To login to the orchestrator, you will use the FQDN of orchestrator northbound interface (NBI).

The FQDN of orchestrator NBI has been auto-generated on the prior step and in this example has the structure as follows:

```
orchestrator-myfab2.myorg2.poc.bayware.io
```

Note: The FQDN of orchestrator NBI is always defined in the following manner: `orchestrator-<fabric>.<company>.<DNS hosted zone>` wherein `company` and `DNS hosted zone` are from the fabric management configuration and same for all fabrics.

Authenticate into the orchestrator via a web browser and use the following information:

- Orchestrator URL - **FQDN of orchestrator NBI**,
- Domain - **default**,
- Username - **admin**,
- Password - **PASSWORD** from the prior step.

20.3 Create Telemetry Node

To create a telemetry node for the orchestrator, run this command with the orchestrator VPC name–in this example `azr1-vpc-myfab2` –as argument:

```
]$ bwctl> create orchestrator telemetry azr1-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 22:01:58.323]  Creating new orchestrator 'azr1-c02-myfab2'...
...
[2019-09-25 22:03:55.862]  ['azr1-c02-myfab2'] created successfully
[2019-09-25 22:03:55.905]  Generating SSH config...
```

Note: The orchestrator node name has been autogenerated. Use this name at the next step.

Next, configure the orchestrator node by running this command with the orchestrator node name–in this example `azr1-c02-myfab2` –as argument:

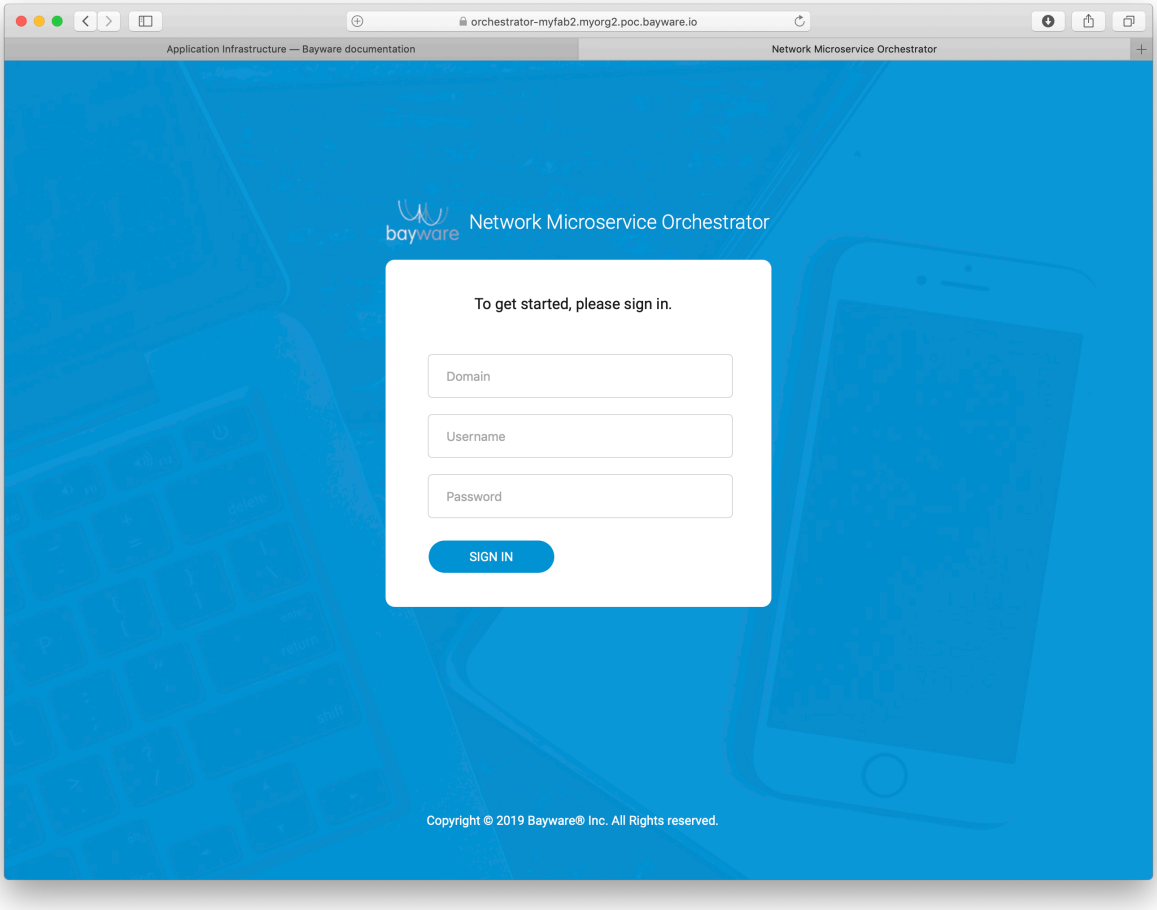


Fig. 20.1: Fig. Orchestrator Login Page

```
]$ bwctl> configure orchestrator azr1-c02-myfab2
```

You should see output similar to:

```
[2019-09-25 22:04:55.433]  Setup/check swarm manager on orchestrator 'azr1-c01-myfab2'
...
[2019-09-25 22:07:48.390] Orchestrators: ['azr1-c02-myfab2'] configured successfully
```

Use your browser to verify the telemetry node is up and running. From the orchestrator GUI open in your browser, click on `Telemetry` in the sidebar navigation menu. A new window will open in your browser similar to the one shown below.

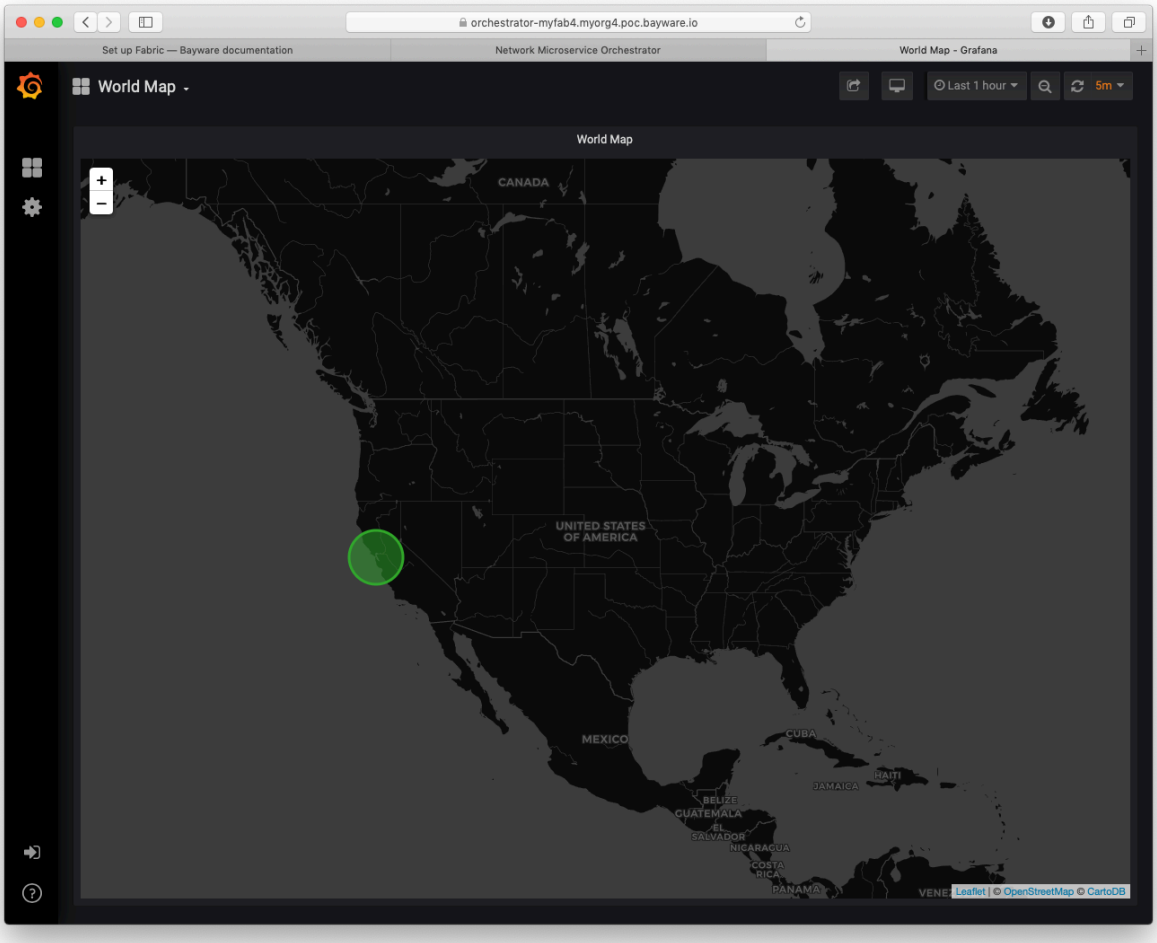


Fig. 20.2: Telemetry Home Page

20.4 Create Events Node

To create an events node for the orchestrator, run this command with the orchestrator VPC name–in this example `azr1-vpc-myfab2` –as argument:

```
]$ bwctl> create orchestrator events azr1-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 22:50:35.536]  Creating new orchestrator 'aws1-c03-myfab2'...
...
[2019-09-25 22:52:34.133]  ['aws1-c03-myfab2'] created successfully
[2019-09-25 22:52:34.178]  Generating SSH config...
```

Note: The orchestrator node name has been autogenerated. Use this name at the next step.

Next, configure the orchestrator node by running this command with the orchestrator node name–in this example azr1-c03-myfab2–as argument:

```
]$ bwctl> configure orchestrator azr1-c03-myfab2
```

You should see output similar to:

```
[2019-09-25 23:00:04.972]  Setup/check swarm manager on orchestrator 'aws1-c01-myfab2'
...
[2019-09-25 23:02:51.605]  Orchestrators: ['azr1-c03-myfab2'] configured successfully
```

Use your browser to verify the events node is up and running. From the orchestrator GUI open in your browser, click on `Events` in the sidebar navigation menu. A new window will open in your browser similar to the one shown below.

20.5 Delete Orchestrator Node

You can delete Telemetry or Events node at any time, without interruption of your application functionality.

To delete the orchestrator node, run this command with the orchestrator node name–in this example `azr1-c03-myfab2` –as the argument:

```
]$ bwctl> delete orchestrator events azr1-c03-myfab2
```

You should see output similar to:

```
[2019-09-26 22:39:00.134]  Deleting orchestrator 'aws1-c03-manil7109'...
...
[2019-09-26 22:41:31.939]  Orchestrator 'aws1-c03-manil7109' deleted successfully
[2019-09-26 22:41:31.963]  Generating SSH config...
```

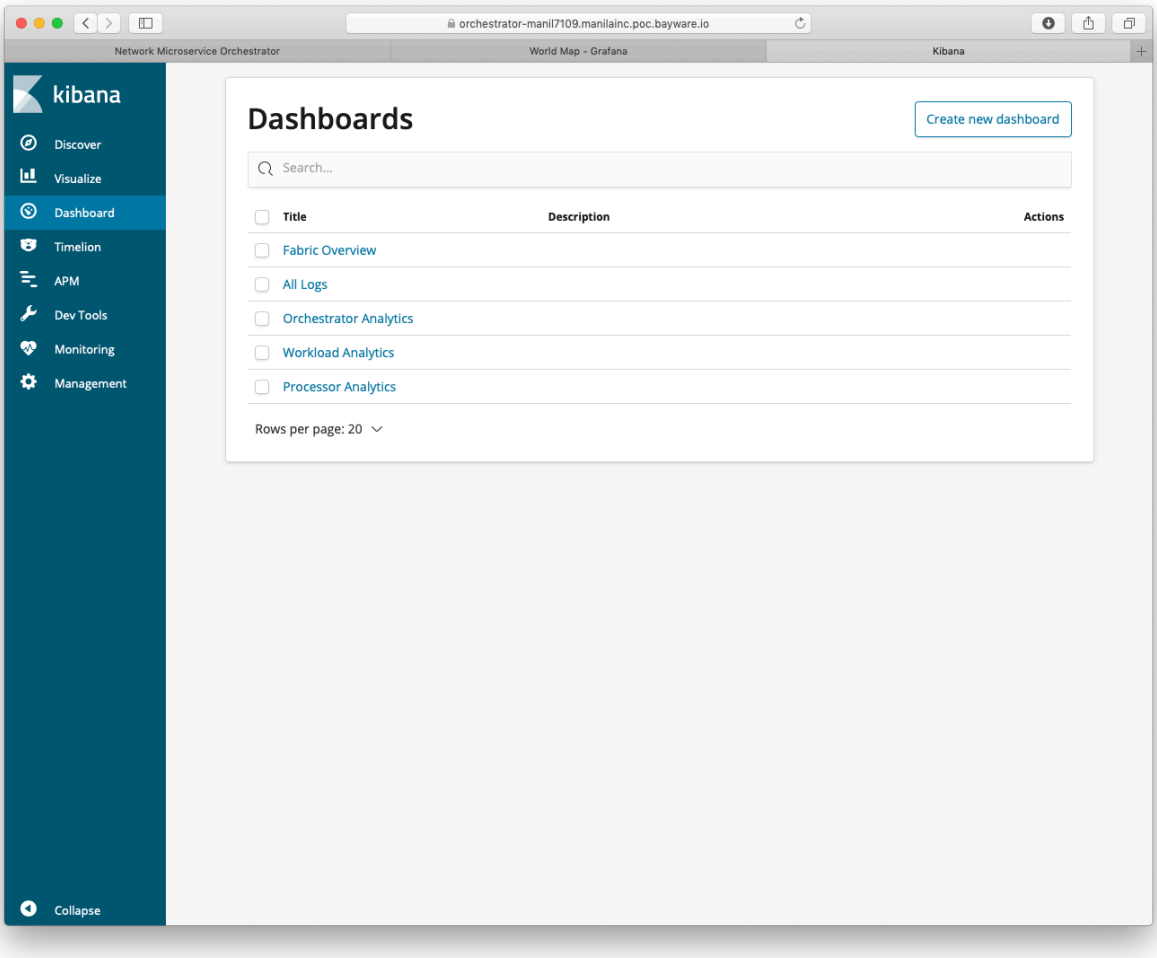


Fig. 20.3: Events Home Page

CHAPTER 21

Deploying Processor

21.1 Public Cloud Deployment

You can add a processor node with policy engine to an existing application VPC or create a new VPC.

Note: Deploying several processors in the same VPC allows you to improve application availability and share the load among the processor nodes.

21.1.1 Create VPC

To create a new VPC for application deployment, with the cloud and region names in this example `azr` and `westus` –as an argument:

```
(myfab2) bwctl> create vpc azr westus
```

You should see output similar to:

```
[2019-09-25 17:51:51.688]  Creating VPC: azr2-vpc-myfab2...
...
[2019-09-25 17:52:50.803]  VPCs ['azr2-vpc-myfab2'] created successfully
```

21.1.2 Create Processor Node

Next, to create a processor, run the command with the target VPC name as an argument:

```
(myfab2) bwctl> create processor azr2-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 17:53:22.613]  Creating new processor 'azr2-p01-myfab2'...
...
[2019-09-25 17:57:27.735]  ['azr2-p01-myfab2'] created successfully
[2019-09-25 17:57:27.763]  Generating SSH config...
```

To configure the processor, you will use the FQDN of orchestrator southbound interface (SBI).

The FQDN of orchestrator SBI has been auto-generated on the prior step and in this example has the structure as follows:

```
controller-myfab2.myorg2.poc.bayware.io
```

Note: The FQDN of orchestrator SBI is always defined in the following manner: `controller-<fabric>.<company>.<DNS hosted zone>`

To configure the processor, run the command with the FQDN of orchestrator SBI–in this example `controller-myfab2.myorg2.poc.bayware.io` –as an argument:

```
(myfab2) bwctl> configure processor azr2-p01-myfab2 --orchestrator-fqdn controller-
↪myfab2.mayorg2.poc.bayware.io
```

You should see output similar to:

```
[2019-09-25 17:58:58.573]  Generate ansible inventory...
...
[2019-09-25 18:00:18.506]  Processors ['azr2-p01-myfab2'] configured successfully
```

To start the processor, run the command:

```
(myfab2) bwctl> start processor azr2-p01-myfab2
```

You should see output similar to:

```
[2019-09-25 18:00:44.719]  Processors to be started: ['azr2-p01-myfab2']
...
[2019-09-25 18:00:47.537]  Processors ['azr2-p01-myfab2'] started successfully
```

21.2 Private Datacenter Deployment

You can install the policy engine on a Linux machine in your private data center. The policy engine has been fully integrated and tested with the operating systems as follows:

- Ubuntu 18.04 LTS,
- RHEL 8 (available starting with the fabric family version 1.4).

You need root access to the Linux machine–thereafter called **processor node** –in order to install the policy engine.

21.2.1 Ubuntu

Add Repository

First, switch to root level access by running this command:

```
]$ sudo su -
```

To add the Bayware repository key to the processor node, run this command:

```
]# wget -qO - https://s3-us-west-1.amazonaws.com/bayware-repo/public/ubuntu/Bayware-
↪public.key | sudo apt-key add -
```

Now, add the Bayware repository to the processor node by running this command:

```
]# echo "deb https://s3-us-west-1.amazonaws.com/bayware-repo/public/1.2/ubuntu bionic␣
↪main" > /etc/apt/sources.list.d/bayware-s3-pub.list
```

Update the package cache on the processor node by running this command:

```
]# apt update
```

Install Engine

To install the policy engine on the processor node, run this command:

```
]# apt install -y ib-engine
```

Note: The policy engine package depends on: `strongswan` (>=5), `openvswitch-common` (>=2.9), `openvswitch-switch` (>=2.9). All dependencies are installed automatically if not found on processor node.

Configure Engine

The policy engine requires for its operations the following configuration:

- paths to root CA certificate, processor certificate, processor private key;
- FQDN of orchestrator southbound API;
- processor location name (optional).

By default, the policy engine works with the certificates and the private key located at `~/opt/bayware/certs/`

To view folder content, run this command:

```
]# ll /opt/bayware/certs/
```

If you have the certificates and the key already installed on the processor node, you should see output similar to this:

```
total 32
drwxr-xr-x 2 root   root 4096 Oct  4 23:54 ./
drwxr-xr-x 4 root   root 4096 Oct  4 23:56 ../
-rw-r--r-- 1 root   root 1956 Oct  4 23:54 ca.crt
```

(continues on next page)

(continued from previous page)

```
-rw-r--r-- 1 root   root 1696 Oct  4 23:54 node.crt
-rw-r--r-- 1 root   root 1005 Oct  4 23:54 node.csr
-r-------- 1 ubuntu root 1675 Oct  4 23:54 node.key
-r-------- 1 ubuntu root 1704 Oct  4 23:54 node.p8
-r-------- 1 ubuntu root 3371 Oct  4 23:54 node.pem
```

Note: You can find requirements to the processor node certificate in a separate guide under the section *Certificate Requirements*.

If you want to change the path to the certificates and the key, use options offered by the policy engine configuration script. To find the available options, run this command:

```
]# /opt/bayware/ib-engine/bin/ib-configure -h
```

To configure orchestrator and location names, run the command with FQDN of orchestrator southbound API and location name as its arguments. Use the option `-s` if you want to set up IPsec configuration for this engine:

```
]# /opt/bayware/ib-engine/bin/ib-configure -s -c <FQDN of Orchestrator SBI> -l <location>
```

You should see this output:

```
engine configuration completed successfully
```

Note: All configuration settings can be changed directly in the config file located at `~/opt/conf/sys.config`

Start Engine

To add the policy engine to processor node autostart, run this command:

```
]# systemctl enable ib-engine
```

To start the policy engine, run this command:

```
]# systemctl start ib-engine
```

Uninstall Engine

To uninstall the policy engine, run this command:

```
]# apt remove -y ib-engine
```

21.2.2 RHEL

In progress...

CHAPTER 22

Deploying Workload

22.1 Public Cloud Deployment

You can create a workload node in the VPC with a processor node already installed.

Note: The processor node secures workload data and control communication including the fabric manager and workload interaction. So, it is mandatory to have a processor node installed in the VPC before the workload deployment.

22.1.1 Create Workload Node

To create a new workload in the VPC, run the command:

```
(myfab2) bwctl> create workload azr2-vpc-myfab2
```

You should see output similar to:

```
[2019-09-25 18:03:26.462]  Creating new workload 'azr2-w01-myfab2'...
...
[2019-09-25 18:06:24.269]  ['azr2-w01-myfab2'] created successfully
[2019-09-25 18:06:24.297]  Generating SSH config...
```

To configure the workload, run the command with the FQDN of orchestrator SBI–in this example `controller-myfab2.myorg2.poc.bayware.io` –as an argument:

```
(myfab2) bwctl> configure workload azr2-w01-myfab2 --orchestrator-fqdn controller-myfab2.
↪myorg2.poc.bayware.io
```

You should see output similar to:

```
[2019-09-25 18:07:17.658]  Generate ansible inventory...
...
[2019-09-25 18:08:25.858]  Workloads ['azr2-w01-myfab2'] configured successfully
```

To start the workload, run the command:

```
(myfab2) bwctl> start workload azr2-w01-myfab2
```

You should see output similar to:

```
[2019-09-25 18:09:18.375]  Workloads to be started: ['azr2-w01-myfab2']
...
[2019-09-25 18:09:21.495]  Workloads ['azr2-w01-myfab2'] started successfully
```

22.2 Private Datacenter Deployment

You can install the policy agent in your private data center on a Linux machine with kernel version 4.15 and up. The policy agent has been fully integrated and tested with the operating systems as follows:

- Ubuntu 18.04 LTS,
- RHEL 8 (available starting with the fabric family version 1.3).

You need root access to the Linux machine–thereafter called **workload node** –in order to install the policy agent.

22.2.1 Ubuntu

Add Repository

First, switch to root level access by running this command:

```
]$ sudo su -
```

To add the Bayware repository key to the workload node, run this command:

```
]# wget -qO - https://s3-us-west-1.amazonaws.com/bayware-repo/public/ubuntu/Bayware-
↪public.key | sudo apt-key add -
```

Now, add the Bayware repository to the workload node by running this command:

```
]# echo "deb https://s3-us-west-1.amazonaws.com/bayware-repo/public/1.2/ubuntu bionic␣
↪main" > /etc/apt/sources.list.d/bayware-s3-pub.list
```

Update the package cache on the workload node by running this command:

```
]# apt update
```

Install Agent

To install the policy agent on the workload node, run this command:

```
]# apt install -y ib-agent
```

Note: The policy agent package depends on: `strongswan` (>=5), `python3` (>=3.6), `python3-iniparse`, `python3-openssl`, `haveged`, `libjansson4`, `libini-config5`. All dependencies are installed automatically if not found on workload node.

Configure Agent

The policy agent requires for its operations the following configuration:

- paths to root CA certificate, workload certificate, workload private key;
- FQDN of orchestrator southbound API;
- workload location name.

By default, the policy agent works with the certificates and the private key located at `~/opt/bayware/certs/`

To view folder content, run this command:

```
]# ll /opt/bayware/certs/
```

If you have the certificates and the key already installed on the workload node, you should see output similar to this:

```
total 32
drwxr-xr-x 2 root   root 4096 Oct  4 15:28 ./
drwxr-xr-x 4 root   root 4096 Oct  4 15:38 ../
-rw-r--r-- 1 root   root 1956 Oct  4 15:28 ca.crt
-rw-r--r-- 1 root   root 1696 Oct  4 15:28 node.crt
-rw-r--r-- 1 root   root 1001 Oct  4 15:28 node.csr
-r-------- 1 ubuntu root 1675 Oct  4 15:28 node.key
-r-------- 1 ubuntu root 1704 Oct  4 15:28 node.p8
-r-------- 1 ubuntu root 3371 Oct  4 15:28 node.pem
```

Note: You can find requirements to the workload node certificate in a separate guide under the section *Certificate Requirements*.

If you want to change the path to the certificates and the key, use options offered by the policy agent configuration script. To find the available options, run this command:

```
]# /opt/bayware/ib-agent/bin/ib-configure -h
```

To configure orchestrator and location names, run the command with FQDN of orchestrator southbound API and location name as its arguments. Use the option `-s` if you want to set up IPsec configuration for this agent:

```
]# /opt/bayware/ib-agent/bin/ib-configure -s -c <FQDN of Orchestrator SBI> -l <location>
```

You should see this output:

```
agent configuration completed successfully
```

Note: All configuration settings can be changed directly in the config file located at `~/etc/ib-agent.conf`

To check the current policy agent configuration, run this command:

```
]# cat /etc/ib-agent.conf
```

You should see output similar to this:

```
[agent]
controller = <FQDN of Orchestrator SBI>
location = <location>
local_domain = ib.loc
token_file = /opt/bayware/ib-agent/conf/tokens.dat
log_file = /var/log/ib-agent/ib-agent.log
log_level = INFO
log_count = 5

[net_iface]
name = ib-fab0
address = 192.168.250.0/24

[ctl_iface]
name = ib-ctl0

[mirror_iface]
name = ib-mon0

[cert]
ca_cert = /opt/bayware/certs/ca.crt
node_cert = /opt/bayware/certs/node.crt
node_key = /opt/bayware/certs/node.key

[rest]
rest_ip = 127.0.0.1
rest_port = 5500
log_file = /var/log/ib-agent/ib-agent-rest.log
log_level = WARNING

[resolver]
log_file = /var/log/ib-agent/ib-agent-resolver.log
log_level = WARNING
file_size = 100000
backup_count = 5
dns_port = 5053
```

Start Agent

To add the policy agent to workload node autostart, run this command:

```
]# systemctl enable ib-agent
```

To start the policy agent, run this command:

```
]# systemctl start ib-agent
```

Uninstall Agent

To uninstall the policy agent, run this command:

```
]# apt remove -y ib-agent
```

22.2.2 RHEL

In progress...

CHAPTER 23

Working with Batches

While you can deploy service interconnection fabric components one by one, using the batch deployments accelerates your migration to clouds even more.

With batch deployments, you can easily extend your existing fabric or automatically create a new fabric. The easiest and error-proven way to do that is to perform three steps as follows:

- export your existing fabric to a yml-file with one `bwctl` command,
- edit and save the yml-file,
- run the batch deployment from the yml-file with another `bwctl` command.

23.1 Extend Existing Fabric

23.1.1 Create Batch File Template

First, set `bwctl` to an existing fabric by running this command with the fabric name–in this example `myfab5` –as the argument:

```
(None) bwctl> set fabric myfab5
```

You should see output similar to:

```
[2019-10-14 16:03:56.255]  Active fabric: 'myfab5'
```

Notice that your `bwctl` prompt has changed, now showing the active fabric:

```
(myfab5) bwctl>
```

Now, export the current fabric state by running this command with the file name–in this example `myfab5.yml` –as the argument:

```
(myfab5) bwctl> export fabric myfab5.yml
```

You should see output similar to:

```
[2019-10-14 16:08:44.299]  Exporting to 'myfab5.yml'
[2019-10-14 16:08:44.403]  Fabric configuration exported successfully
```

In this example, the fabric comprises:

- two VPCs in Azure,
- orchestrator node in one VPC,
- one processor and one workload in another VPC.

The fabric resource graph is shown below.

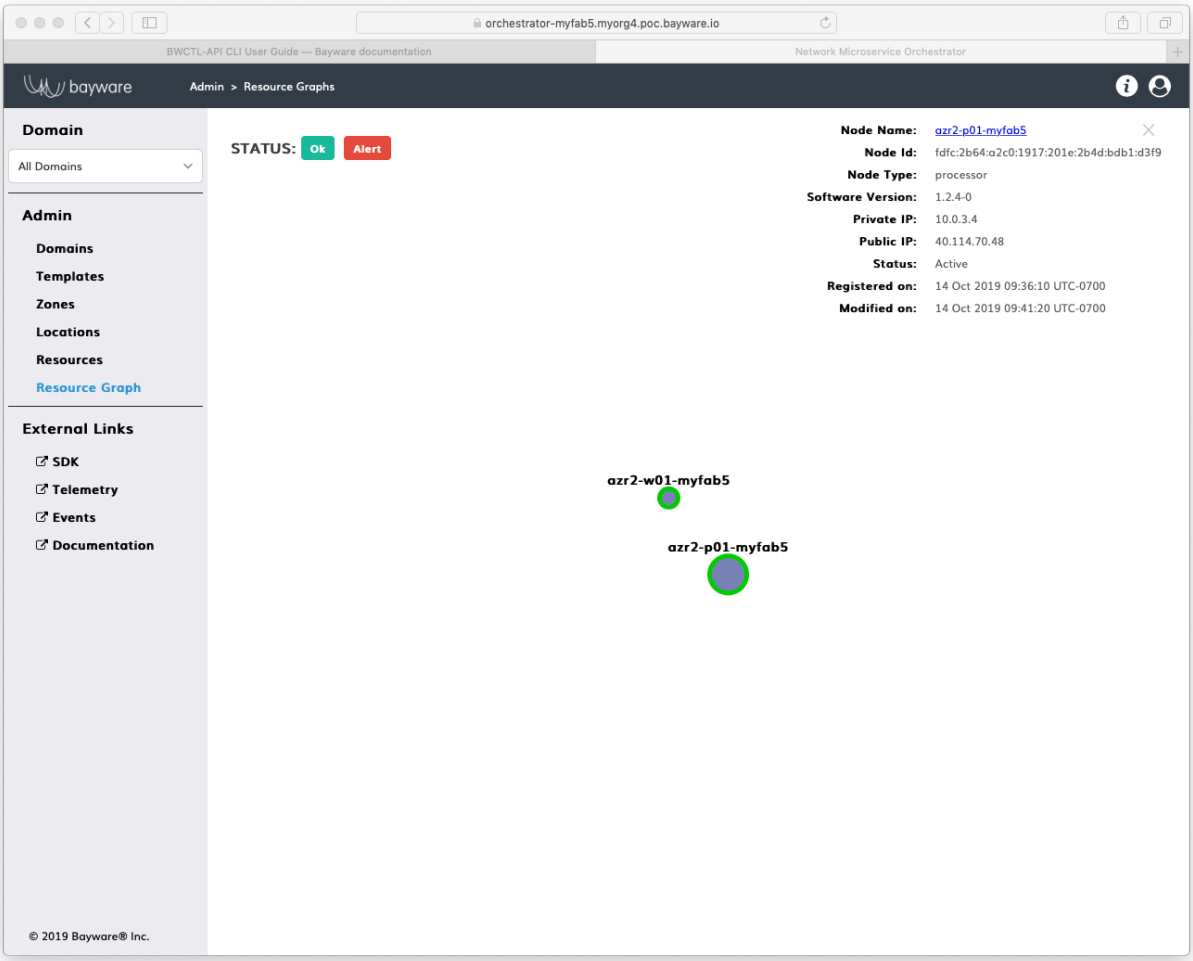


Fig. 23.1: Initial Resource Graph (Azure)

The file with the current fabric state contains:

```
---
apiVersion: fabric.bayware.io/v2
kind: Batch
metadata:
  name: 'myfab5'
  description: 'Fabric "myfab5" export at Mon Oct 14 16:50:38 2019'
spec:
  - kind: Fabric
    metadata:
      description: 'optional description'
      name: 'myfab5'
    spec:
      companyName: myorg4
      credentialsFile: {}
      sshKeys:
        privateKey: {}
  - kind: Orchestrator
    metadata:
      description: 'optional description>'
      fabric: 'myfab5'
      name: 'azr1-c01-myfab5'
    spec:
      role: 'manager'
      type: 'controller'
      properties:
        marketplace: False
      vpc: 'azr1-vpc-myfab5'
    state: 'configured'
  - kind: Processor
    metadata:
      description: 'optional description'
      fabric: 'myfab5'
      name: 'azr2-p01-myfab5'
    spec:
      config:
        orchestrator: 'controller-myfab5.myorg4.poc.bayware.io'
      properties:
        marketplace: False
      vpc: 'azr2-vpc-myfab5'
    state: 'started'
  - kind: Vpc
    metadata:
      description: 'optional description'
      fabric: 'myfab5'
      name: 'azr1-vpc-myfab5'
    spec:
      cloud: 'azr'
      properties:
      region: 'westus'
  - kind: Vpc
    metadata:
      description: 'optional description'
```

(continues on next page)

(continued from previous page)

```
      fabric: 'myfab5'
      name: 'azr2-vpc-myfab5'
    spec:
      cloud: 'azr'
      properties:
      region: 'eastus'
  - kind: Workload
    metadata:
      description: 'optional description'
      fabric: 'myfab5'
      name: 'azr2-w01-myfab5'
    spec:
      config:
        orchestrator: 'controller-myfab5.myorg4.poc.bayware.io'
      properties:
        marketplace: False
      vpc: 'azr2-vpc-myfab5'
    state: 'started'
```

23.1.2 Edit Batch File

To amend the current fabric state with a desired amount of new VPCs, processors and workloads, use your favorite text editor, e.g. `vim` or `nano`, to describe a desired state:

```
]$ nano myfab5.yml
```

For example, to add more workloads to the existing VPC, duplicate the existing workload specification as many times as needed while providing a unique name for each new workload–in this example `azr2-w02-myfab5` and `azr2-w03-myfab5`.

Note: You can either keep or remove the existing components from the batch file. While running the batch deployment, bwctl will apply only a difference between the current and desired state.

After editing, the batch file–in this example `myfab5.yml` –contains:

```
---
apiVersion: fabric.bayware.io/v2
kind: Batch
metadata:
  name: 'myfab5'
  description: 'Fabric "myfab5" export at Mon Oct 14 16:50:38 2019'
spec:
  - kind: Workload
    metadata:
      description: 'optional description'
      fabric: 'myfab5'
      name: 'azr2-w02-myfab5'
    spec:
      config:
        orchestrator: 'controller-myfab5.myorg4.poc.bayware.io'
```

(continues on next page)

(continued from previous page)

```
     properties:
       marketplace: False
      vpc: 'azr2-vpc-myfab5'
    state: 'started'
  - kind: Workload
    metadata:
      description: 'optional description'
      fabric: 'myfab5'
      name: 'azr2-w03-myfab5'
    spec:
      config:
        orchestrator: 'controller-myfab5.myorg4.poc.bayware.io'
      properties:
        marketplace: False
      vpc: 'azr2-vpc-myfab5'
    state: 'started'
```

23.1.3 Run Batch Deployment

To deploy new fabric components from the batch file, run this command with the batch file name–in this example `myfab5.yml` –as the argument:

```
(myfab5) bwctl> create batch myfab5.yml
```

You will see output similar to:

```
[2019-10-14 18:25:42.565]  Create batch: file='myfab5.yml', input=format='yaml', dry-
↪run=False
...
[2019-10-14 18:33:29.640]  Batch is finished
```

Check your resource graph at this point to see that the fabric has two more workload nodes now.

23.2 Create New Fabric

23.2.1 Create Batch File Template

You can use the state of your existing fabric, to create a new fabric. The new fabric might be completely identical to the existing one or have the same set of components but deployed in a different cloud.

First, export the current fabric configuration by running this command with the file name–in this example `myfab6.yml` –as the argument:

```
(myfab5) bwctl> export fabric myfab6.yml
```

You should see output similar to:

```
[2019-10-14 18:41:47.936]  Exporting to 'myfab6.yml'
[2019-10-14 18:41:47.955]  Fabric configuration exported successfully
```

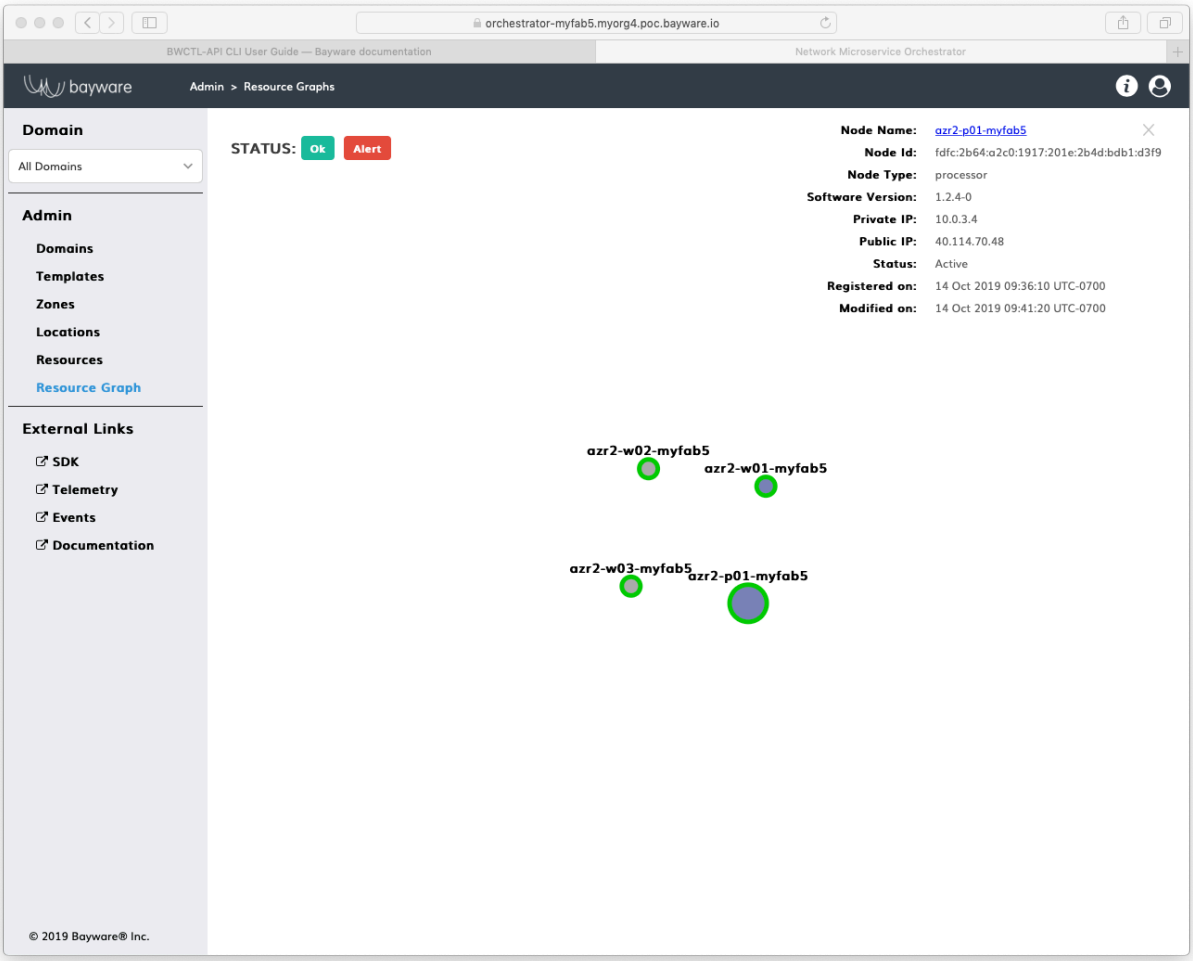


Fig. 23.2: Resource Graph after the Batch Deployment (Azure)

Note: In this example, the state of the fabric, extended in the prior step, is exported. So, the file `myfab6.yml` now describes two Azure VPCs, one orchestrator node, one processor node and three workload nodes.

23.2.2 Edit Batch File

To create a desired state for the new fabric, edit the file created in the prior step.

In this example, the new fabric will fully replicate in AWS the fabric existing in Azure. So, use find-and-replace option in your favorite text editor to make these changes to the batch file–in this example `myfab6.yml`:

Specification	Before	After
Fabric Manager	`myfab5`	`myfab6`
Cloud	`azr`	`aws`
Control VPC Region	`westus`	`us-west-1`
Workload VPC Region	`eastus`	`us-east-1`

Save the file after editing.

23.2.3 Run Batch Deployment

To deploy the new fabric from the batch file you have created in the prior step, run this command with the batch file name–in this example `myfab6.yml` –as the argument:

```
]$ bwctl create batch myfab6.yml
```

You will see output similar to:

```
[2019-10-14 18:53:56.377]  Create batch: file='myfab6.yml', input=format='yaml', dry-
↪run=False
...
[2019-10-14 19:08:57.007]  Batch is finished
[2019-10-14 19:08:57.007]  IMPORTANT: Here is administrator's password that was used to␣
↪initialize controller. Please change it after first login
[2019-10-14 19:08:57.007]  Password: 0Y417IqAMa6h
```

The new fabric with two VPCs, orchestrator, processor and three workload nodes have been created in AWS.

Warning: When a controller node of orchestrator is created during a batch deployment, a password is always shown in the last line of the command output. Write down the **PASSWORD** as it appears on your screen, as it will be needed later.

To check the current state of the new fabric, set `bwctl` to the new fabric by running this command with the fabric name-in this example `myfab6` -as the argument:

```
]$ bwctl set fabric myfab6
```

You should see output similar to:

```
[2019-10-14 20:29:47.921]  Active fabric: 'myfab6'
```

Now, export the current fabric state by running this command with the file name–in this example `myfab6-current.yml` –as the argument:

```
]$ bwctl export fabric myfab6-current.yml
```

You should see output similar to:

```
[2019-10-14 20:30:09.084]  Exporting to 'myfab6-current.yml'
[2019-10-14 20:30:09.188]  Fabric configuration exported successfully
```

The file with the current fabric state contains:

```
---
apiVersion: fabric.bayware.io/v2
kind: Batch
metadata:
  name: 'myfab6'
  description: 'Fabric "myfab6" export at Tue Oct 15 15:30:09 2019'
spec:
  - kind: Fabric
    metadata:
      description: 'optional description'
      name: 'myfab6'
    spec:
      companyName: myorg4
      credentialsFile: {}
      sshKeys:
        privateKey: {}
  - kind: Orchestrator
    metadata:
      description: 'optional description>'
      fabric: 'myfab6'
      name: 'aws1-c01-myfab6'
    spec:
      role: 'manager'
      type: 'controller'
      properties:
        marketplace: False
      vpc: 'aws1-vpc-myfab6'
    state: 'configured'
  - kind: Processor
    metadata:
      description: 'optional description'
      fabric: 'myfab6'
      name: 'aws2-p01-myfab6'
    spec:
      config:
```

(continues on next page)

(continued from previous page)

```
      orchestrator: 'controller-myfab6.myorg4.poc.bayware.io'
    properties:
      marketplace: False
    vpc: 'aws2-vpc-myfab6'
  state: 'started'
- kind: Vpc
  metadata:
    description: 'optional description'
    fabric: 'myfab6'
    name: 'aws1-vpc-myfab6'
  spec:
    cloud: 'aws'
    properties:
    region: 'us-west-1'
- kind: Vpc
  metadata:
    description: 'optional description'
    fabric: 'myfab6'
    name: 'aws2-vpc-myfab6'
  spec:
    cloud: 'aws'
    properties:
    region: 'us-east-1'
- kind: Workload
  metadata:
    description: 'optional description'
    fabric: 'myfab6'
    name: 'aws2-w01-myfab6'
  spec:
    config:
      orchestrator: 'controller-myfab6.myorg4.poc.bayware.io'
    properties:
      marketplace: False
    vpc: 'aws2-vpc-myfab6'
  state: 'started'
- kind: Workload
  metadata:
    description: 'optional description'
    fabric: 'myfab6'
    name: 'aws2-w02-myfab6'
  spec:
    config:
      orchestrator: 'controller-myfab6.myorg4.poc.bayware.io'
    properties:
      marketplace: False
    vpc: 'aws2-vpc-myfab6'
  state: 'started'
- kind: Workload
  metadata:
    description: 'optional description'
    fabric: 'myfab6'
    name: 'aws2-w03-myfab6'
```

(continues on next page)

(continued from previous page)

```
  spec:
    config:
      orchestrator: 'controller-myfab6.myorg4.poc.bayware.io'
    properties:
      marketplace: False
    vpc: 'aws2-vpc-myfab6'
  state: 'started'
```

Login to the new orchestrator and check the resource graph of your new fabric.

At this point, you will see that the new fabric in AWS completely replicates the Azure fabric.

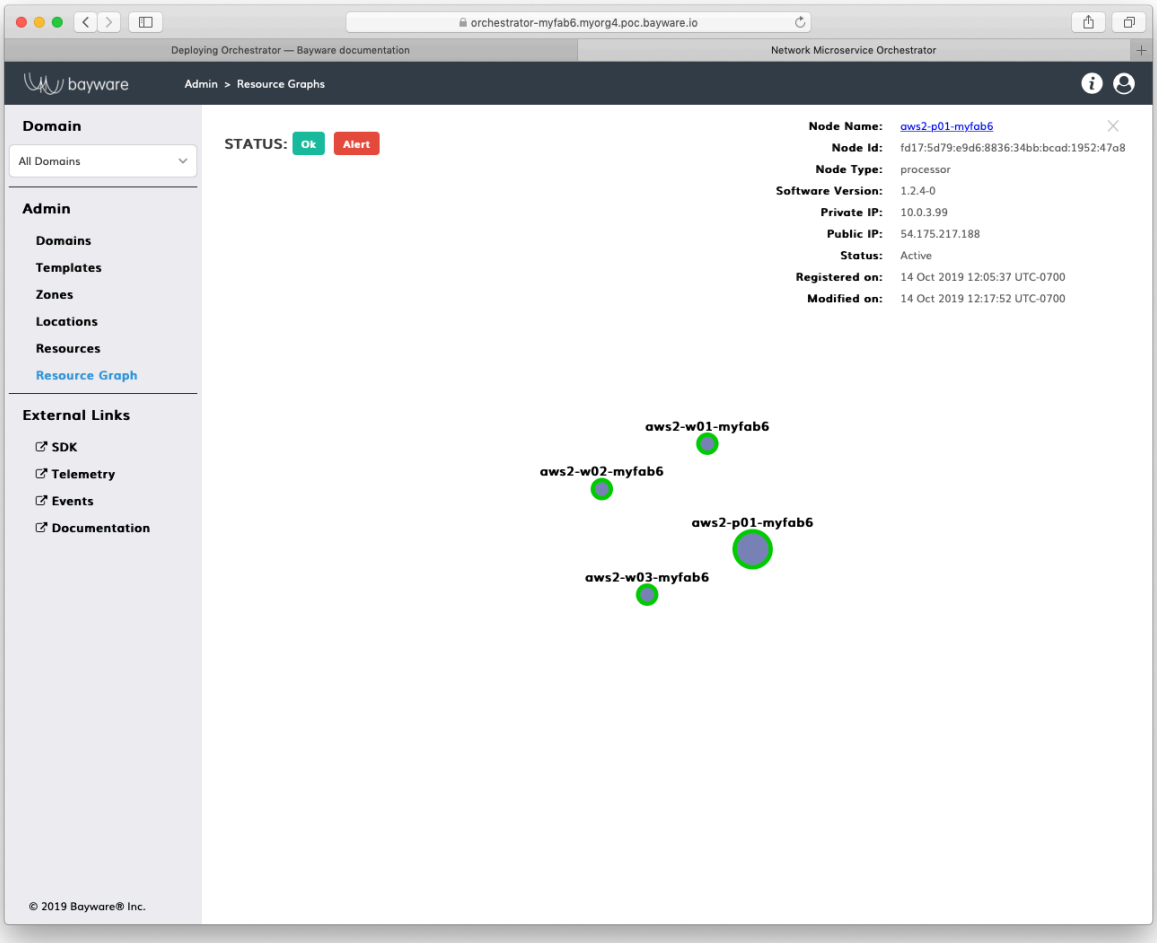


Fig. 23.3: Resource Graph of New Fabric (AWS)

23.3 Summary

Batch deployment is a powerful tool when you need to:

- add new VPCs or scale out processors and workloads in your existing VPCs;

- copy your existing environment, e.g. `test`, and automatically create a new identical environment, e.g. `production`;
- copy your entire infrastructure in one cloud and paste it in another.

CHAPTER 24

BWCTL CLI Manual

24.1 About BWCTL

BWCTL is a command line interface (CLI) tool that enables you to interact with public clouds using commands from your shell. The tool offers all the functionality required for SIF component management. With BWCTL you can build your fabric from scratch by setting up VPCs, orchestrator nodes, processor nodes, and workload nodes.

BWCTL tool comes preinstalled on your fabric manager node. To use BWCTL tool, access your fabric manager node from any Linux, macOS, or Windows machine.

To run the commands, you will need a terminal window with an SSH client:

- MacOS – Use Terminal application with built-in SSH client.
- Linux – Use your favorite terminal window with built-in SSH client.
- Windows 10 – If you haven't already enabled an SSH client to use with PowerShell, PuTTY is an easy alternative. PuTTY can act as both your terminal window and your SSH client.

When creating the fabric manager in a public cloud, you are able to specify how you would like to access the virtual machine using SSH. That is, you may specify a username and password or a public key. Use these credentials to log into your VM.

However, BWCTL tools run under the username ubuntu. If you have created a different username during VM creation, simply `su` into ubuntu before proceeding:

```
jsmith@my-fabric-manager:~$ sudo su - ubuntu
```

BWCTL enables you to install fabric components and configure them. You can `show`, `create`, `update`, `start`, `stop`, and `delete` components of the service interconnection fabric: `fabric`, `vpc`, `orchestrator`, `processor`, `workload`. Also, the tool allows you to perform the same operations in batch mode i.e., on a grouping of fabric components.

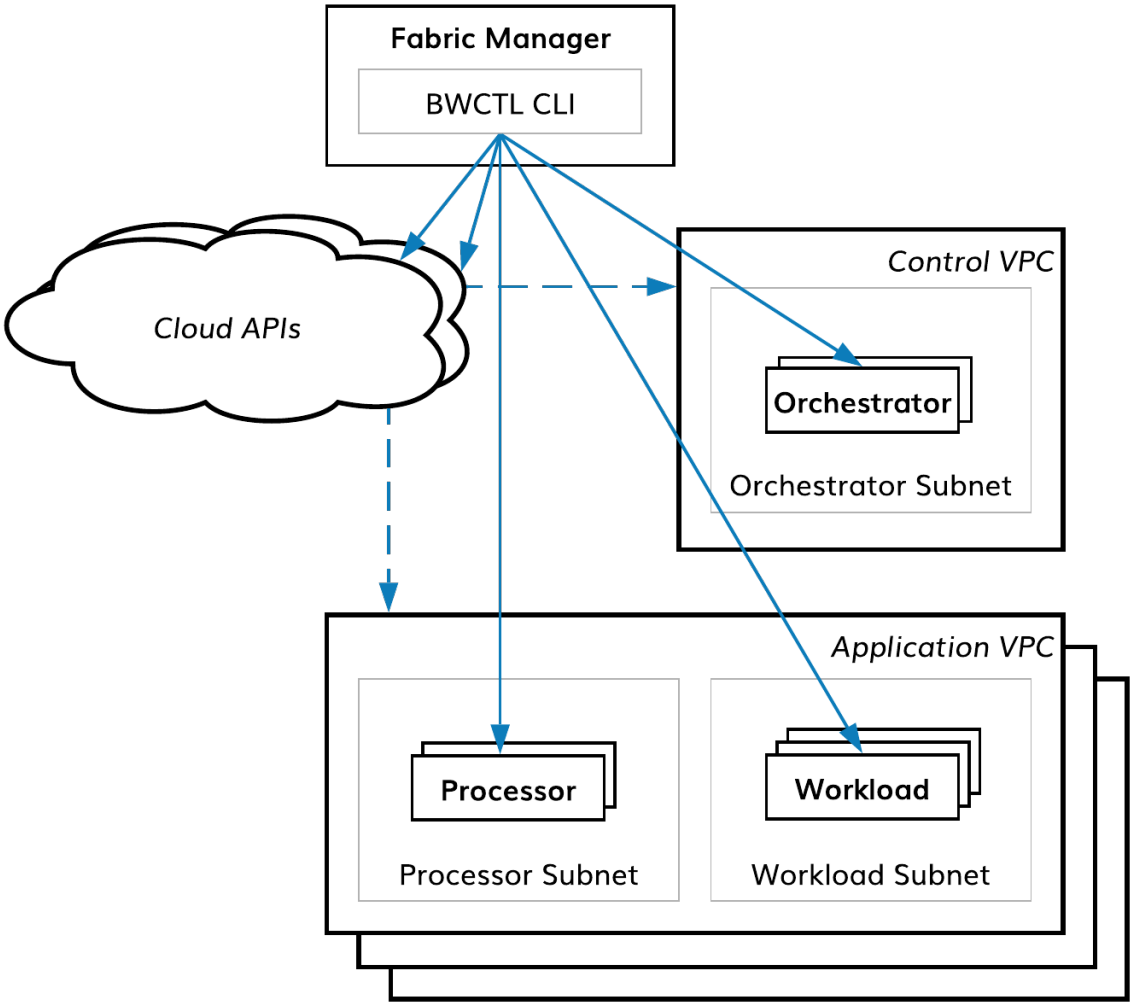


Fig. 24.1: BWCTL CLI for SIF component management

24.2 Upgrading BWCTL

BWCTL tool comprises two packages: one with business logic, called `bwctl`, and another with resource templates, called `bwctl-resources`.

You can upgrade the `bwctl` package already installed on your fabric manager to the latest version in the family by running the command:

```
]$ pip3 install --upgrade bwctl
```

Verify that the `bwctl` package installed correctly by running the command:

```
]$ bwctl --version
bwctl/0.7.1
```

To upgrade the `bwctl-resources` package, run the command:

```
]$ pip3 install --upgrade bwctl-resources
```

Verify that the `bwctl-resources` package installed correctly by running the command:

```
]$ bwctl-resources --version
bwctl-resources/0.7.1
```

24.3 Configuring BWCTL

24.3.1 Configuring BWCTL after installation

Before you can start using BWCTL for fabric deployment, you must configure the tool with your cloud credentials.

First, start the BWXTL initalization process by running the command:

```
]$ bwctl init
```

You should see this output:

```
[2019-09-25 17:30:12.156] Welcome to bwctl initialization
[2019-09-25 17:30:12.156]  Fabric manager
[2019-09-25 17:30:12.156]   Company name (value is required):
```

In interactive mode, provide all required values when prompted.

Note: Press `<Enter>` to accept the default values.

After the initialization you should have a configuration similar to:

```
[2019-09-25 17:30:12.156]  Welcome to bwctl initialization
[2019-09-25 17:30:12.156]  Fabric manager
[2019-09-25 17:30:12.156]   Company name (value is required): myorg3
[2019-09-25 17:30:30.113]  Global
```

(continues on next page)

(continued from previous page)

```
[2019-09-25 17:30:30.113]   Cloud providers credentials file [~/.bwctl/credentials.yml]:
[2019-09-25 17:30:34.004]   DNS hosted zone (value is required): poc.bayware.io
[2019-09-25 17:30:37.325]   Debug enabled [true]:
[2019-09-25 17:30:42.062]   Production mode enabled [true]:
[2019-09-25 17:30:44.548]   Marketplace images to be used [false]:
[2019-09-25 17:30:48.624]  Components
[2019-09-25 17:30:48.624]   Family version [1.2]:
[2019-09-25 17:30:51.959]  Cloud storage
[2019-09-25 17:30:51.959]   Store bwctl state on AWS S3 [false]:
[2019-09-25 17:30:58.786]   Store terraform state on AWS S3 [true]:
[2019-09-25 17:31:05.633]    AWS S3 bucket name [terraform-states-sandboxes]:
[2019-09-25 17:31:12.933]    AWS region [us-west-1]:
[2019-09-25 17:31:15.876]  SSH keys
[2019-09-25 17:31:15.876]   SSH Private key file []:
[2019-09-25 17:31:21.268]  Configuration is done
```

To view the file with your cloud provider credentials, `cat` to where the `cloud credentials.yml` file was specified during the initialization by running the command with the path to the file–in this example `/home/ubuntu/.bwctl/credentials.yml` –as argument:

```
]$ cd /home/ubuntu/.bwctl/credentials.yml
```

You should see this output:

```
---

# Add cloud-provider credentials that will be used when creating
# infrastructure and accessing repositories.

aws:
  # In the AWS console, select the IAM service for managing users and keys.
  # Select Users, and then Add User.  Type in a user name and check
  # programmatic access.  Users require access to EC2, S3, and Route53.
  # Copy and paste the secret access key and key ID here.
  aws_secret_access_key:
  aws_access_key_id:
azr:
  # Azure provides detailed steps for generating required credentials
  # on the command line, which you can find at this URL:
  # https://docs.microsoft.com/en-us/azure/virtual-machines/linux/terraform-install-
↪configure#set-up-terraform-access-to-azure
  azr_client_id:
  azr_client_secret:
  azr_resource_group_name:
  azr_subscription_id:
  azr_tennant_id:
gcp:
  # Google uses a GCP Service Account that is granted a limited set of
  # IAM permissions for generating infrastructure.  From the IAM & Admin
  # page, select the service account to use and then click "create key"
  # in the drop-down menu on the right.  The JSON file will be downloaded
  # to your computer.  Put the path to that file here.
  google_cloud_keyfile_json:
```

Use your editor of choice (ex: `vim`, `nano`) to add your public cloud credentials to `credentials.yml`.

24.3.2 Changing BWCTL configuration

If you need to change BWCTL configuration, run `bwctl init` again or update its configuration file stored locally at `/home/ubuntu/.bwctl/config`.

To check your current configuration, run this command:

```
$ cat .bwctl/config
cloud_storage:
  state:
    bucket: terraform-states-sandboxes
    enabled: false
    region: us-west-1
  terraform:
    bucket: terraform-states-sandboxes
    enabled: true
    region: us-west-1
components:
  branch: release
  family: '1.2'
credentials_file: ~/.bwctl/credentials.yml
current_fabric: myfab6
debug: true
fabric_manager:
  company_name: myorg4
  id: bangkok-c0_54.200.219.211_ubuntu
  ip: 54.200.219.211
  production: bangkokinc
hosted_zone: poc.bayware.io
marketplace: false
production: true
ssh_keys:
  private_key: ''
```

24.4 Getting started with BWCTL

24.4.1 Typing the first command

Using Linux command line

To give a command in BWCTL using the Linux command line, you will type `bwctl` along with the required input and press the `<return>` key.

To start using BWCTL tool, run the command:

```
]$ bwctl --help
Usage: bwctl-api [OPTIONS] COMMAND [ARGS]...

  Bayware CLI
```

(continues on next page)

(continued from previous page)

```
Options:
 -v, --version  Print version and exit.
 -h, --help     Show this message and exit.

Commands:
  configure  Configure commands
  create     Create commands
  delete     Delete commands
  export     Export fabric specification to file
  leave      Leave commands
  restart    Restart commands
  set        Set commands
  show       Show commands
  start      Start commands
  stop       Stop commands
  update     Update commands
```

Using BWCTL command line

To switch from the Linux command line to the BWCTL command line, you will type `bwctl` and press the `<return>` key:

```
]$ bwctl
(None) bwctl>
```

The BWCTL command line begins with the fabric name in parentheses. If no fabric is set, the parentheses contain the word `None`.

To get help in the BWCTL command line, run the command:

```
(None) bwctl> help
```

To quit the BWCTL command line, run the command:

```
(None) bwctl> quit
```

24.4.2 Command structure

The BWCTL command line is comprised of several components:

- bwctl
- any options required by bwctl to execute the command
- the command and, in most cases, subcommand
- any arguments required by the command

```
]$ bwctl --help
Usage: bwctl [OPTIONS] COMMAND [ARGS]...
```

24.4.3 Command line options

You can use the following command line options by typing them on the command line immediately after `bwctl`:

–**version, -v** A boolean switch that displays the current version of BWCTL-API tool.

–**help, -h** A boolean switch that displays the commands available for execution.

You can finish the command line with the `--help` option following either command or subcommand. The output will always give you a hint about what else you need to type.

To see the help for the command, type the command only followed by `--help` and press `<return>`:

```
]$ bwctl show --help
Usage: bwctl show [OPTIONS] COMMAND [ARGS]...

  Show commands

Options:
  -h, --help  Show this message and exit.

Commands:
  fabric        Show fabric information
  orchestrator  Show orchestrator information
  processor     Show processor information
  vpc           Show VPC information
  workload      Show workload information
```

To see the help for the subcommand, type the command followed by the subcommand and `--help` and press `<return>`:

```
]$ bwctl show workload --help
Usage: bwctl show workload [OPTIONS]

  Show workload information

Options:
  --name TEXT                 Show full information on a given workload.
  --cloud [azr|aws|gcp|all]  List workloads or show full information on them.
  --full                      Show full information on workloads.
  -h, --help                  Show this message and exit.
```

Different commands support different options. Detailed information on the available options can be found in the documentation section *Using commands*.

24.4.4 Commands

With BWCTL you can manage all fabric components in your service interconnection fabric. Each command includes the entity kind as a subcommand and the entity name as an argument. Some commands have an entity specification file as a mandatory argument.

BWCTL supports the following commands:

configure KIND NAME [OPTIONS] The command configures one or multiple entities. The specification file is optional for this command.

create KIND NAME [OPTIONS] The command creates one or multiple entities. The specification file is mandatory for the `batch` kind only.

delete KIND NAME [OPTIONS] The command deletes one or multiple entities. The specification file is mandatory for the `batch` kind only.

export KIND FILE [OPTIONS] The command exports the fabric specification to a file.

leave KIND The command causes the tool to exit from the current fabric (namespace).

restart KIND NAME [OPTIONS] The command restarts one or multiple entities. You can use options instead of the name to specify entities in the command.

set KIND NAME The command causes the tool to enter the specified fabric (namespace).

show KIND NAME [OPTIONS] The command shows one or multiple entities. You can use options instead of the name to specify entities in the command.

start KIND NAME [OPTIONS] The command starts one or multiple entities. You can use options instead of the name to specify entities in the command.

stop KIND NAME [OPTIONS] The command updates one or multiple entities. You can use options instead of the name to specify entities in the command.

update KIND NAME [OPTIONS] The command updates one or multiple entities. You can use options instead of the name to specify entities in the command.

24.4.5 Kinds

The diagram below depicts the service interconnection fabric components and relationships between them.

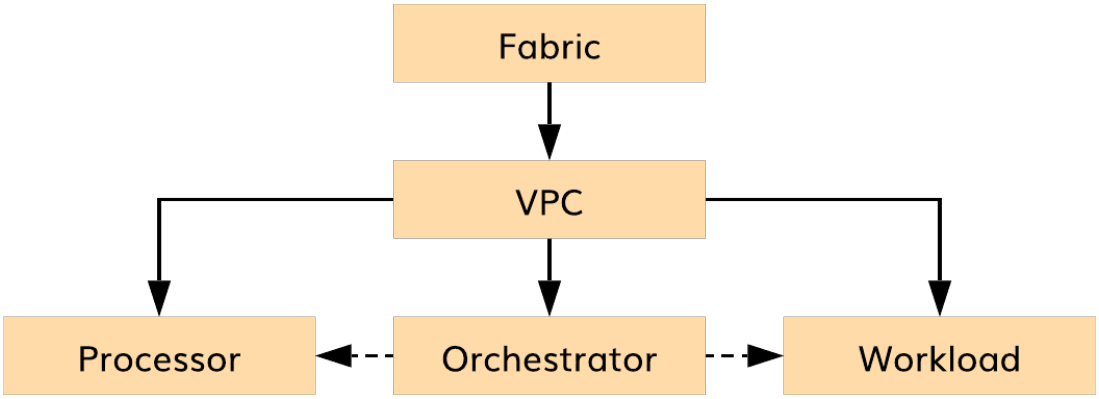


Fig. 24.2: Fabric components

To see the entity types, you can run the show command without a subcommand:

```
]$ bwctl show
Usage: bwctl show [OPTIONS] COMMAND [ARGS]...

  Show commands
```

(continues on next page)

(continued from previous page)

```
Options:
  -h, --help  Show this message and exit.

Commands:
  fabric        Show fabric information
  orchestrator  Show orchestrator information
  processor     Show processor information
  vpc           Show VPC information
  workload      Show workload information
```

BWCTL manages the following entity types:

fabric NAME The `fabric` entity represents a fabric itself.

orchestrator NAME The `orchestrator` entity represents a VM playing orchestrator node role.

processor NAME The `processor` entity represents a VM playing processor node role.

vpc NAME The `vpc` entity represents a cloud VPC.

workload NAME The `workload` entity represents a VM playing workload node role.

24.4.6 Batch

With BWCTL CLI, you can use a single batch command to manage a set of entities of the same or different types. Below is an example of the command.

```
]$ bwctl create batch azr-infra-batch.yml
```

24.5 Using commands

24.5.1 Supported commands for each entity type

There are five groups of entities, each of which has its own set of commands.

configure, create, delete, export, leave, set, show This set of commands is applicable to the following types of entities:

- FABRIC

create, delete, show This set of commands is applicable to the following types of entities:

- VPC

configure, create, delete, show, update This set of commands is applicable to the following types of entities:

- ORCHESTRATOR

configure, create, delete, show, start, stop, update This set of commands is applicable to the following types of entities:

- PROCESSOR
- WORKLOAD

create, delete This set of commands is applicable to the following types of entities:

- BATCH

24.5.2 Managing fabrics

You can manage fabrics using the following commands:

configure fabric NAME [OPTIONS] The command configures the fabric. You can use the options in this command as follows:

`--credentials-file <ceredentials-file>` path to file with credentials for public clouds.

`--ssh-private-key <ssh-private-key>` fabric manager's SSH private key. **The SSH private key is optional. If not specified, the tool will create an SSH key dedicated to the fabric.**

create fabric NAME The command creates the fabric.

delete fabric NAME The command deletes the fabric.

leave fabric The command leaves the current fabric.

set fabric NAME The command sets the fabric as current.

show fabric [OPTIONS] The command shows the current fabric components if the fabric is set. If not set, the command outputs the list of all available fabrics. Also, you can always use this option:

`--list-all` to show the list of all available fabrics.

export fabric FILE The command exports all fabric components and their specifications into a file.

An example of the fabric specification file is shown below.

```
]$ cat fabric-spec.yml
---
apiVersion: fabric.bayware.io/v2
kind: Fabric
metadata:
  description: 'Fabric ohio5784'
  name: 'ohio5784'
spec:
  companyName: ohioinc
  credentialsFile: /home/ubuntu/credentials/credentials.yml
  sshKeys:
    privateKey: {}
```

24.5.3 Managing VPCs

You can manage VPCs using the following commands:

create vpc aws|azr|gcp REGION [OPTIONS] The command creates a VPC in the current fabric. The VPC is created in the specified region of the selected public cloud provider. You can use the options in this command as follows:

`--file <filename>` file with VPC configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

Note: You can see a list of regions by running the command `bwctl show vpc --regions`.

delete vpc NAME The command deletes the vpc in the current fabric. You can use the options in this command as follows:

`--dry-run`

running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

show vpc [OPTIONS] The command shows the list of all VPCs in the current fabric if run without any option. You can use the options in this command as follows:

`--name <vpc-name>` shows information on a given VPC.

`--cloud <aws|azr|gcp|all>` lists all VPCs in a given cloud.

`--full` instead of list, provides full information on vpc.

`--zones` outputs the list of zones in which VPC can be created.

An example of the VPC specification file is shown below.

```
]$ cat vpc-spec.yml
---
apiVersion: fabric.bayware.io/v2
kind: Vpc
metadata:
  description: 'Azure VPC'
  fabric: 'ohio5784'
  name: 'azr1-vpc-ohio5784'
spec:
  cloud: 'azr'
  properties:
    region: 'southcentralus'
```

24.5.4 Managing orchestrators

You can manage orchestrators using the following commands:

configure orchestrator NAME [OPTIONS] The command configures the orchestrator node in the current fabric. You can use the options in this command as follows:

`--all` configures all orchestrator nodes.

`--file <filename>` file with orchestrator configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

create orchestrator controller|telemetry|events VPC [OPTIONS] The command creates the vpc in the current fabric. You can use the options in this command as follows:

`--file <filename>` file with vpc configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

delete orchestrator NAME The command deletes the orchestrator node in the current fabric. You can use the options in this command as follows:

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

show orchestrator [OPTIONS] The command shows the list of all orchestrators in the current fabric if run without any option. You can use the options in this command as follows:

`--name <node-name>` shows information on a given orchestrator node.

`--cloud <aws|azr|gcp|all>` lists all VPCs in a given cloud.

`--full` instead of list, provides full information on orchestrator nodes.

update orchestrator NAME The command updates the orchestrator node in the current fabric. You can use the options in this command as follows:

`--all` updates all orchestrator nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

An example of the orchestrator specification file is shown below.

```
]$ cat orchestrator-spec.yml
---
apiVersion: fabric.bayware.io/v2
kind: Orchestrator
metadata:
  description: 'Policy controller'
  fabric: 'ohio5784'
  name: 'aws1-c01-ohio5784'
spec:
  role: 'manager'
  type: 'controller'
  properties:
     vpc: 'aws1-vpc-ohio5784'
state: 'configured'
```

24.5.5 Managing processors

You can manage processors using the following commands:

configure processor NAME [OPTIONS] The command configures the processor node in the current fabric. You can use the options in this command as follows:

`--all` configures all processor nodes.

`--orchestrator <FQDN>` orchestrator FQDN.

`--location <name>` location name.

`--file <filename>` file with processor configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

create processor VPC [OPTIONS] The command creates the processor node in the current fabric. You can use the options in this command as follows:

`--file <filename>` file with processor configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

delete processor NAME The command deletes the processor node in the current fabric. You can use the options in this command as follows:

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

show processor [OPTIONS] The command shows the list of all processors in the current fabric if run without any option. You can use the options in this command as follows:

`--name <node-name>` shows information on a given processor node.

`--cloud <aws|azr|gcp|all>` lists all processors in a given cloud.

`--full` instead of list, provides full information on processor nodes.

start processor [NAME] [OPTIONS] The command starts the given processor node in the current fabric if the node's name is specified. You can use the options in this command as follows:

`--all` starts all processor nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

stops processor [NAME] [OPTIONS] The command stops the given processor node in the current fabric if the node's name is specified. You can use the options in this command as follows:

`--all` stops all processor nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

update processor [NAME] [OPTIONS] The command updates the given processor node in the current fabric if the node's name is specified. You can use the options in this command as follows:

`--all` updates all processor nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

An example of the processor specification file is shown below.

```
]$ cat processor-spec.yml
---
apiVersion: fabric.bayware.io/v2
kind: Processor
metadata:
  description: 'Azure processor'
  fabric: 'ohio5784'
  name: 'azr1-p01-ohio5784'
spec:
  config:
    orchestrator: 'controller-ohio5784.ohioinc.poc.bayware.io'
  properties:
    vpc: 'azr1-vpc-ohio5784'
state: 'started'
```

24.5.6 Managing workloads

You can manage workloads using the following commands:

configure workload NAME [OPTIONS] The command configures the workload node in the current fabric. You can use the options in this command as follows:

`--all` configures all workload nodes.

`--orchestrator <FQDN>` orchestrator FQDN.

`--location <name>` location name.

`--file <filename>` file with workload configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

create workload VPC [OPTIONS] The command creates the workload node in the current fabric. You can use the options in this command as follows:

`--file <filename>` file with processor configuration.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

delete workload NAME The command deletes the workload node in the current fabric. You can use the options in this command as follows:

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

show workload [OPTIONS] The command shows the list of all workloads in the current fabric if run without any option. You can use the options in this command as follows:

`--name <node-name>` shows information on a given workload node.

`--cloud <aws|azr|gcp|all>` lists all workloads in a given cloud.

`--full` instead of list, provides full information on workload nodes.

start workload [NAME] [OPTIONS] The command starts the given workload node in the current fabric if the node's name is specified. You can use the options in this command as follows:

`--all` starts all workload nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

stops workload [NAME] [OPTIONS] The command stops the given workload node in the current fabric if the node's name is specified. You can use the options in this command as follows:

`--all` stops all workload nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

update workload [NAME] [OPTIONS] The command updates the given workload node in the current fabric if the node's name is specified. You can use the options in this command as follows:

`--all` updates all workload nodes.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

An example of the workload specification file is shown below.

```
]$ cat workload-spec.yml
---
apiVersion: fabric.bayware.io/v2
kind: Workload
metadata:
  description: 'Azure workload'
  fabric: 'ohio5784'
  name: 'azr1-w01-ohio5784'
spec:
  config:
    orchestrator: 'controller-ohio5784.ohioinc.poc.bayware.io'
  properties:
    vpc: 'azr1-vpc-ohio5784'
state: 'started'
```

24.5.7 Working with batches

You can manage batch file execution using the following commands:

create batch FILE [OPTIONS] The command creates the batch. The specification file is mandatory for this command.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

delete batch FILE [OPTIONS] The command deletes the batch. The specification file is mandatory for this command.

`--dry-run` running the command with this option doesn't make any changes but shows which changes will be made if you run the command without the `--dry-run` option.

An example of the batch specification file is shown below.

```
---
apiVersion: fabric.bayware.io/v2
kind: Batch
metadata:
  name: backend-infra-and-config-template
  description: 'Creates VPC, processor, and three workloads'
spec:
  - kind: Workload
    metadata:
      description: 'optional description'
      fabric: 'texas2270'
      name: 'azr1-w01-texas2270'
    spec:
      config:
        domain: 'cloud-net'
        orchestrator: 'controller-texas2270.texasinc.poc.bayware.io'
        password: 'messycard58'
        username: 'wkld-azr'
      properties:
      vpc: 'azr1-vpc-texas2270'
    state: 'started'
```

(continues on next page)

(continued from previous page)

```
- kind: Workload
  metadata:
    description: 'optional description'
    fabric: 'texas2270'
    name: 'azr1-w02-texas2270'
  spec:
    config:
      orchestrator: 'controller-texas2270.texasinc.poc.bayware.io'
    properties:
    vpc: 'azr1-vpc-texas2270'
  state: 'started'
- kind: Processor
  metadata:
    description: 'optional description'
    fabric: 'texas2270'
    name: 'azr1-p01-texas2270'
  spec:
    config:
      orchestrator: 'controller-texas2270.texasinc.poc.bayware.io'
    properties:
    vpc: 'azr1-vpc-texas2270'
   state: 'started'
- kind: Vpc
  metadata:
    description: 'optional description'
    fabric: 'texas2270'
    name: 'azr1-vpc-texas2270'
  spec:
    cloud: 'azr'
    properties:
    zone: 'southcentralus'
```

24.6 BWCTL cheat sheet

BWCTL Commands

	Action	Object	Required	Required	Optional	Optional
Common options						
bwctl						-v, --version -h, --help
Batch						
bwctl	create	batch	<filename>			--dry-run
bwctl	delete	batch	<filename>			--dry-run
Fabric						
bwctl	configure	fabric	<name>		--company-name <company-name> --credentials-file <credentials-file> --ssh-private-key <ssh-private-key> --file <filename>	
bwctl	create	fabric	<name>			
bwctl	delete	fabric	<name>			
bwctl	leave	fabric				
bwctl	set	fabric	<name>			
bwctl	show	fabric			--list-all	
bwctl	export	fabric	<filename>			
Orchestrator						
bwctl	configure	orchestrator	<name>		--file <filename>	--all --dry-run
bwctl	delete	orchestrator	<name>			--dry-run
bwctl	create	orchestrator	<orch-type>	<vpc>	--file <filename>	--dry-run
bwctl	show	orchestrator			--name --cloud <aws|azr|gcp|all> --full	
bwctl	update	orchestrator	<name>			--all --dry-run
Processor						
bwctl	configure	processor	<name>		--orchestrator <FQDN> --location <name> --file <filename>	--all --dry-run
bwctl	delete	processor	<name>			--dry-run
bwctl	create	processor	<vpc>		--file <filename>	--dry-run
bwctl	show	processor			--name --cloud <aws|azr|gcp|all> --full	
bwctl	start	processor	<name>			--all --dry-run
bwctl	stop	processor	<name>			--all --dry-run
bwctl	update	processor	<name>			--all --dry-run
VPC						
bwctl	create	vpc	<aws|azr|gcp>	<zone>	--file <filename>	--dry-run
bwctl	delete	vpc	<name>			--dry-run
bwctl	show	vpc			--regions --name --cloud <aws|azr|gcp|all> --full	
Workload						
bwctl	configure	workload	<name>		--orchestrator <FQDN> --location <name> --file <filename>	--all --dry-run
bwctl	delete	workload	<name>			--dry-run
bwctl	create	workload	<vpc>		--file <filename>	--dry-run
bwctl	show	workload			--name --cloud <aws|azr|gcp|all> --full	
bwctl	start	workload	<name>			--all --dry-run
bwctl	stop	workload	<name>			--all --dry-run
bwctl	update	workload	<name>			--all --dry-run

Fig. 24.3: BWCTL-CLI Cheat Sheet

CHAPTER 25

System Administration

This document describes the system administration functions necessary for configuring SIF policy with the `bwctl-api` command-line tool or via a web interface. The steps below will guide you through the creation of domains and administrators.

Note: Both the `bwctl-api` command-line tool and the web interface utilize the same orchestrator northbound interface (NBI).

25.1 Login to Orchestrator

25.1.1 Default Credentials

As part of orchestrator configuration process performed with the fabric manager, a FQDN of orchestrator NBI and default administrator credentials were automatically generated.

Note: The FQDN of orchestrator NBI is always defined in the following manner: `orchestrator-<fabric>.<company>.<DNS hosted zone>` wherein `company` and `DNS hosted zone` are from the fabric management configuration and same for all fabrics.

The default administrator credentials are always as follows:

- Orchestrator URL - **FQDN of orchestrator NBI**
- Domain - **default**
- Username - **admin**
- Password - **PASSWORD** from the configuration step.

25.1.2 Using Web Interface

Go to the orchestrator Login page using the FQDN of orchestrator northbound interface–in this example orchestrator- `myfab5.myorg4.poc.bayware.io`.

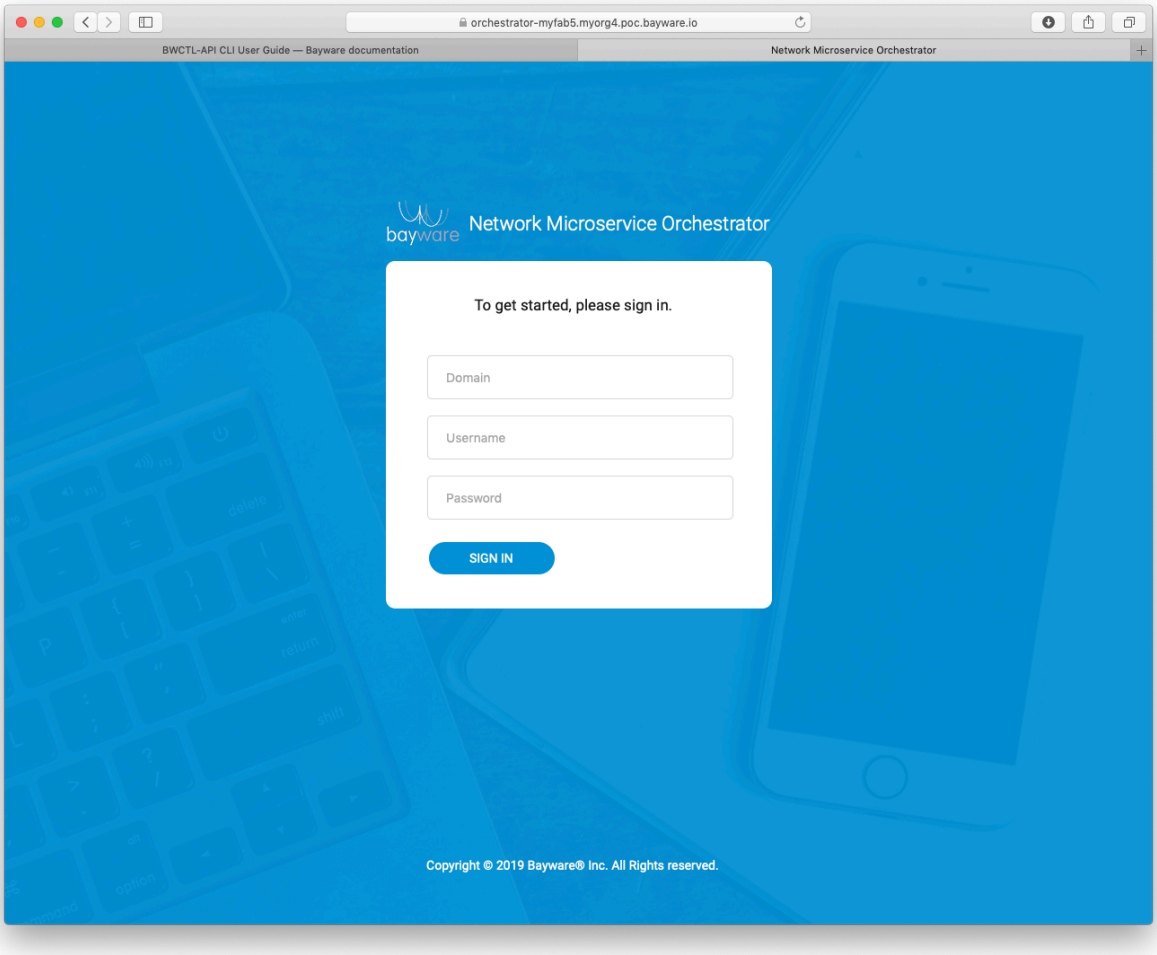


Fig. 25.1: Orchestrator Login Page

Authenticate into the orchestrator and you will be redirected to the Resource Graph page.

Note: Use the default administrator credentials when login to the orchestrator for the first time.

25.1.3 Using BWCTL-API

You can install the BWCTL-API CLI tool on your workstation and work with the orchestrator using a command-line interface.

Note: The BWCTL-API CLI tool comes preinstalled on all fabric manager nodes.

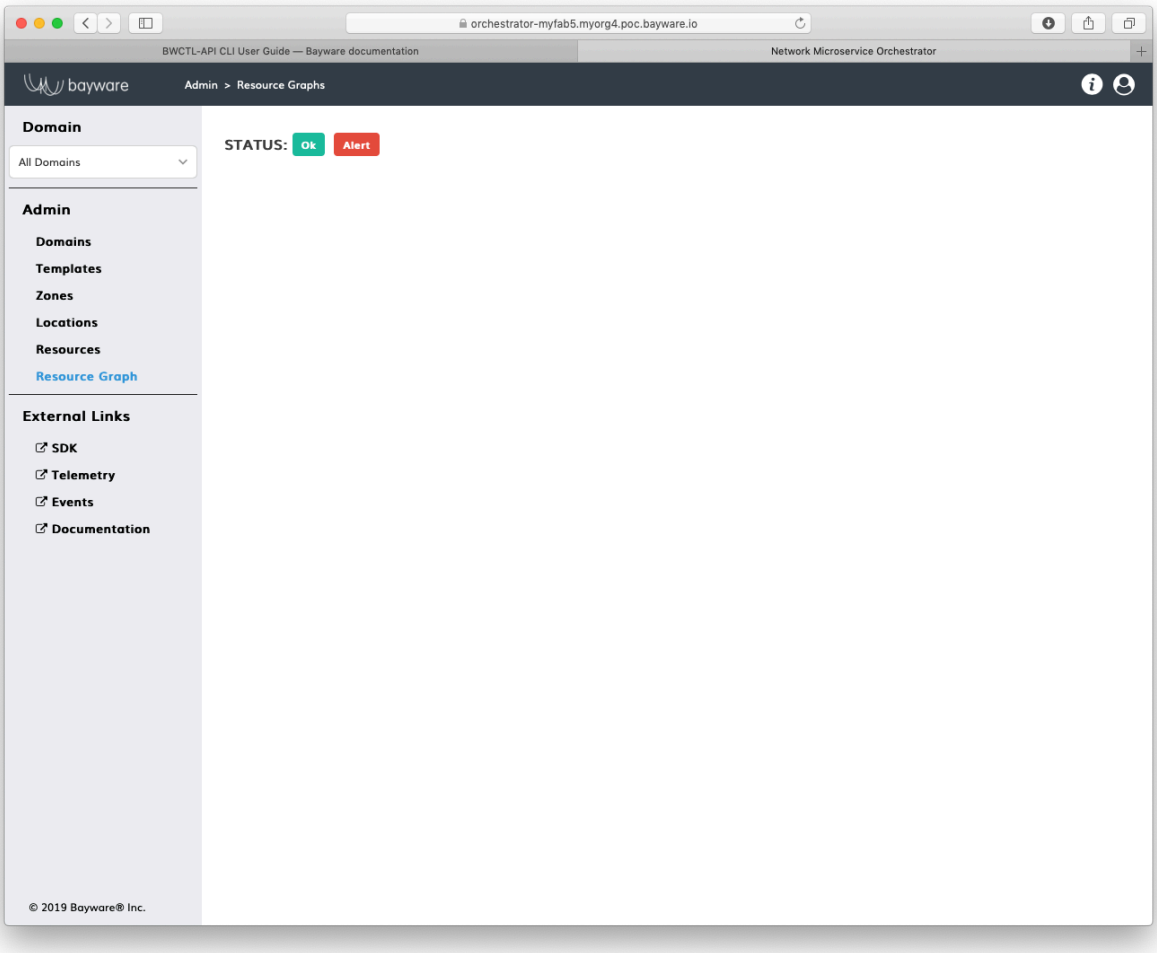


Fig. 25.2: Example of Resource Graph Page

Configure the tool with the administrator credentials.

Note: Again, Use the default administrator credentials when login to the orchestrator for the first time.

To set up the credentials using the BWCTL-API default configuration file, run this command:

```
]$ nano .bwctl-api/config.yml
```

After editing, your credential file will look similar to:

```
hostname: 'orchestrator-myfab5.myorg4.poc.bayware.io'
domain: 'default'
login:  'admin'
password: 'aEPbj6AMa2Yz'
```

To check whether you are able to authenticate into the orchestrator, run this command:

```
]$ bwctl-api show domain
```

You should see the default domain specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Batch
metadata:
  name: List of Domains
spec:
- kind: Domain
  metadata:
    domain: default
    domain_description: System default administrative domain
  spec:
    auth_method:
    - LocalAuth
    - LDAP
    domain_type: Administrative
```

25.2 Create Administrative Domain

25.2.1 Default Domain

After the orchestrator installation, only the `default` domain exists for administrative purposes.

Note: You can keep using this domain for resource and application policy management only if you don't need to reduce administrative scope.

25.2.2 Using Web Interface

To add a new domain, in the `Admin > Domains` section, click `Add Domain`.

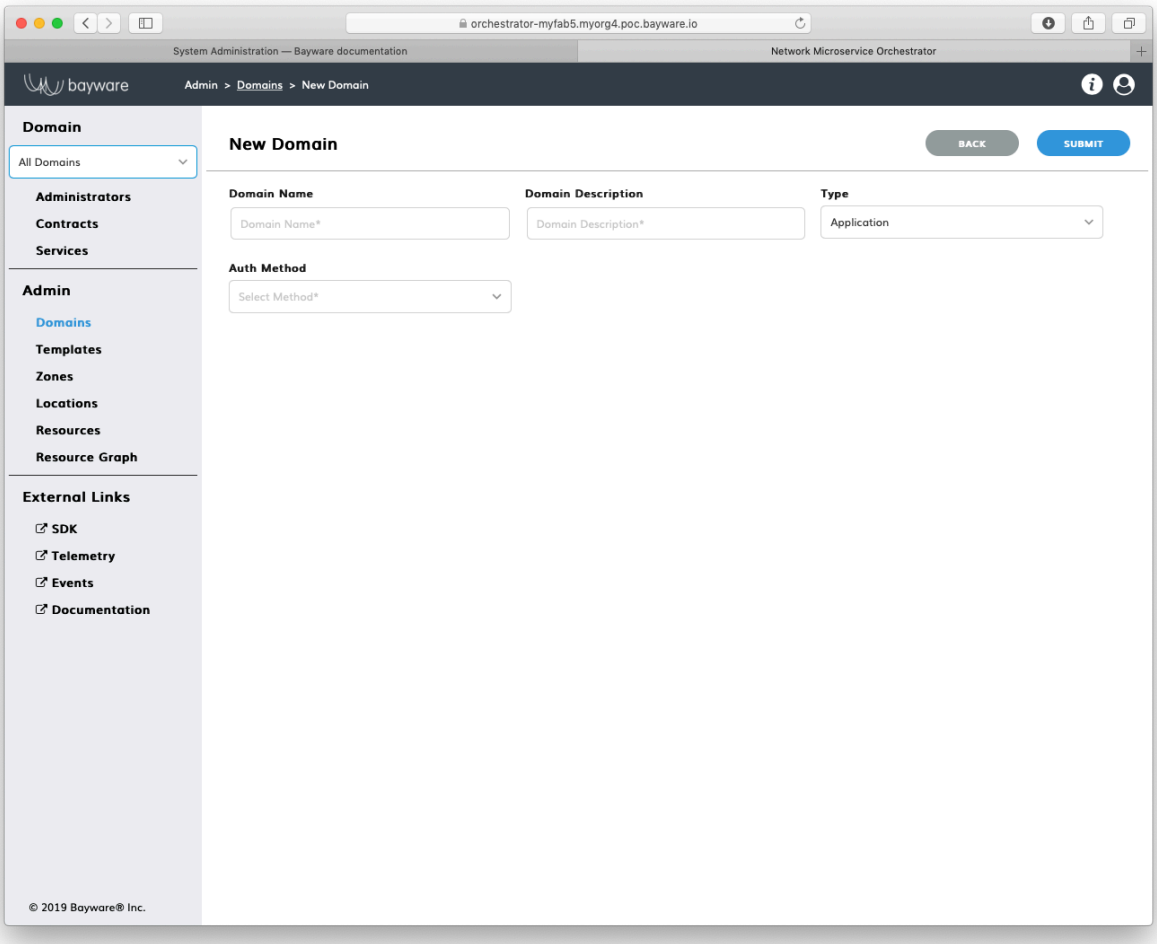


Fig. 25.3: New Domain Page

Fill out the fields on the `New Domain` page:

domain name desired domain name;

type `Administrative` and `Application` – an administrative domain is used to manage application and/or resource policy, while an application domain is used to manage application policy only;

description add description for domain;

authorization method `LocalAuth` in local orchestrator database or `LDAPAuth` at directory server.

Submit the new domain configuration and you should see the domain appears in the list on the `Admin > Domains` page.

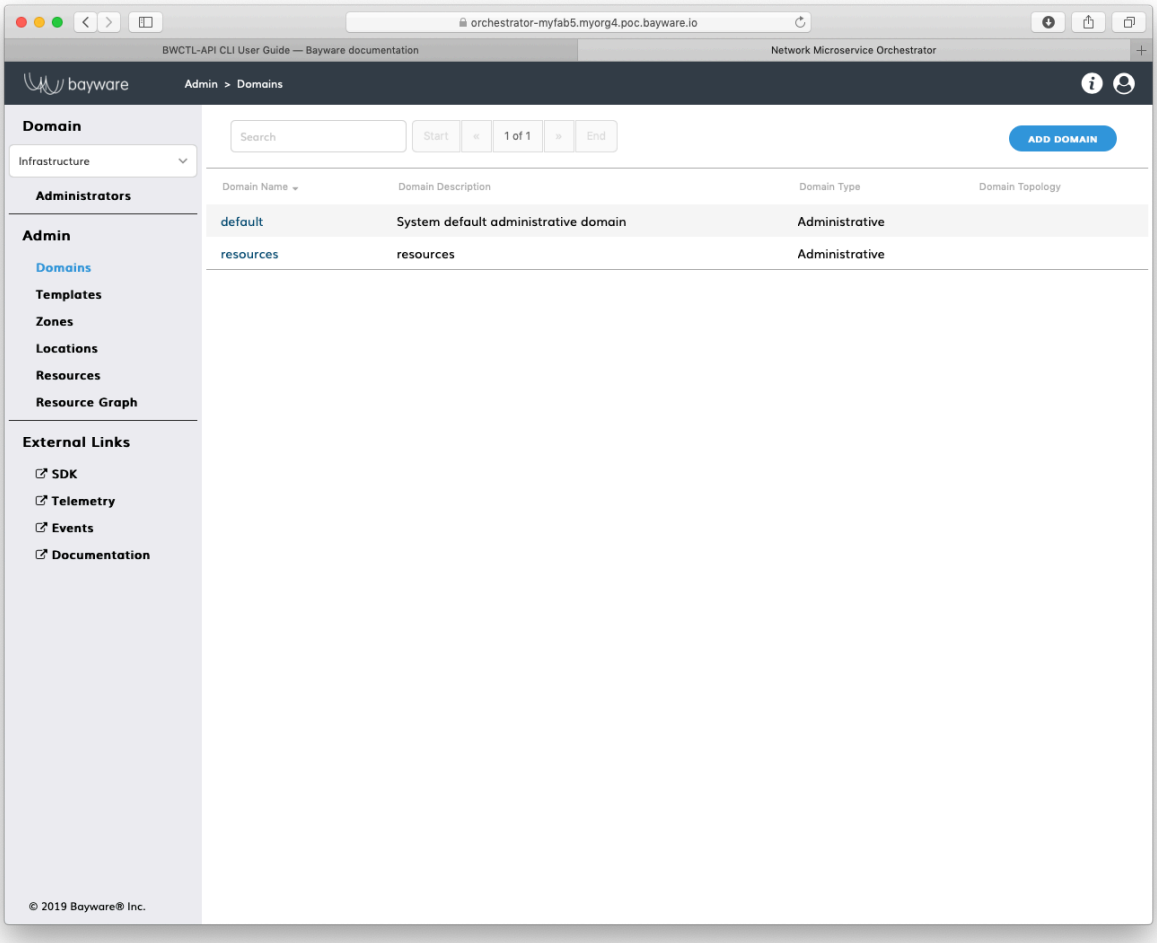


Fig. 25.4: List of Domains

25.2.3 Using BWCTL-API

To create a new domain from `bwctl-api`, run this command with the desired domain name and type–in this example `resources` and `Administrative` respectively–as the arguments:

```
]$ bwctl-api create domain resources -type Administrative
```

You should see output similar to this:

```
[2019-10-15 18:28:49.711]  Domain 'resources' created successfully
```

Now, check again the list of existing domains by running this command:

```
]$ bwctl-api show domain
```

You should see the new domain specification among others:

```
---
apiVersion: policy.bayware.io/v1
kind: Batch
metadata:
  name: List of Domains
spec:
- kind: Domain
  metadata:
    domain: default
    domain_description: System default administrative domain
  spec:
    auth_method:
    - LocalAuth
    - LDAP
    domain_type: Administrative
- kind: Domain
  metadata:
    domain: resources
    domain_description: resources
  spec:
    auth_method:
    - LocalAuth
    domain_type: Administrative
```

Note: When options are not specified, the `bwctl-api` tool applies default configuration settings. See BWCTL-API CLI Manual for specific details.

25.3 Create Administrator

25.3.1 Default Administrator

The new orchestrator is always set up with a default administrator `admin` placed in the administrative domain `default`.

25.3.2 Using Web Interface

If needed, to create a named administrator account click `Add Admin` on the `All Domains > Administrators` page.

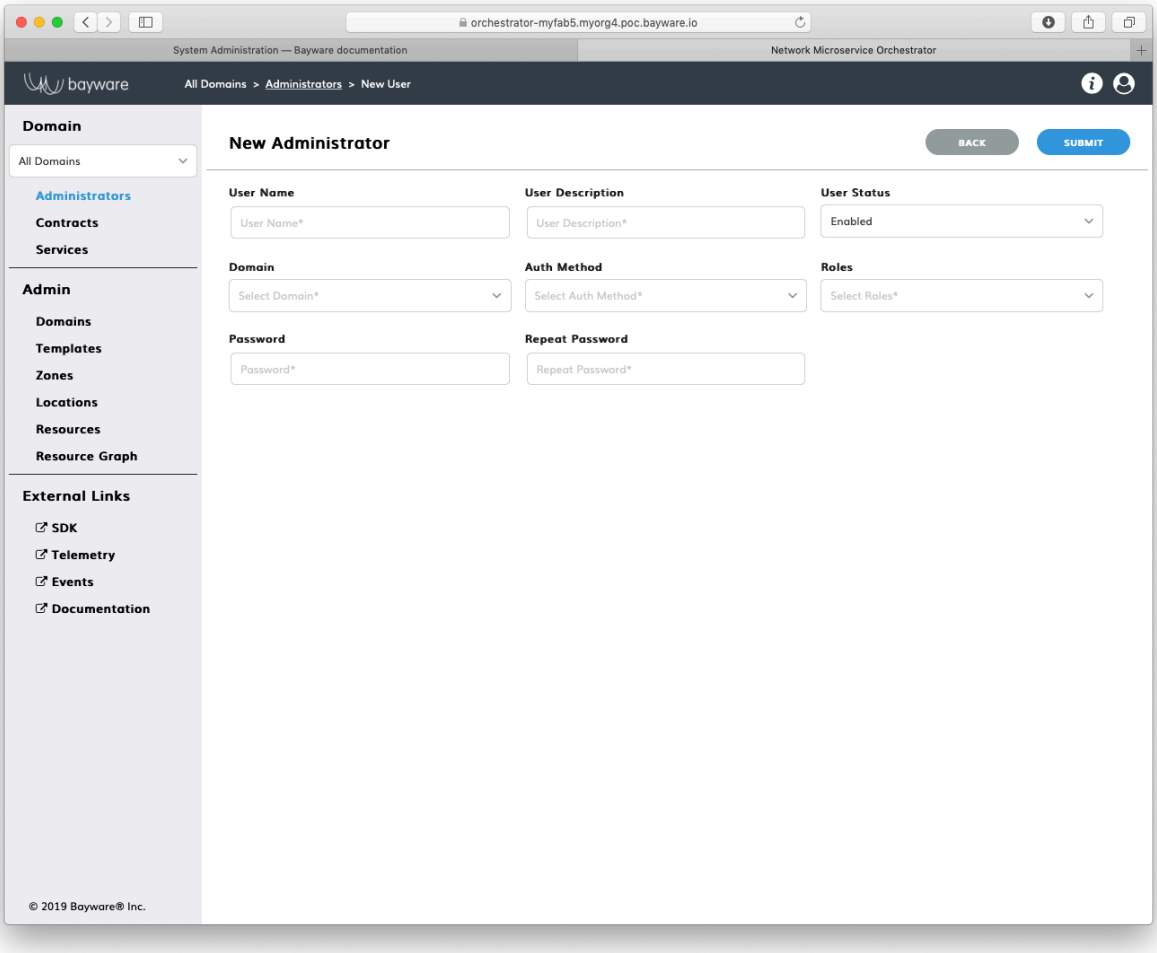


Fig. 25.5: New Administrator Page

Fill out the fields on the `New Administrator` page:

User Name desired administrator name;

User Description administrator description;

User status choose between `Enabled` and `Disabled`;

Domain select domain where administrator will operate;

Auth Method `LocalAuth` in local database or `LDAPAuth` at directory server (available options are inherited from domain authentication type);

Roles administrator permissions– `systemAdmin` or `domainAdmin`;

Password, Repeat password administrator password.

Submit the new administrator configuration and you should see the new administrator appears in the list on the `All Domains > Administrators` page.

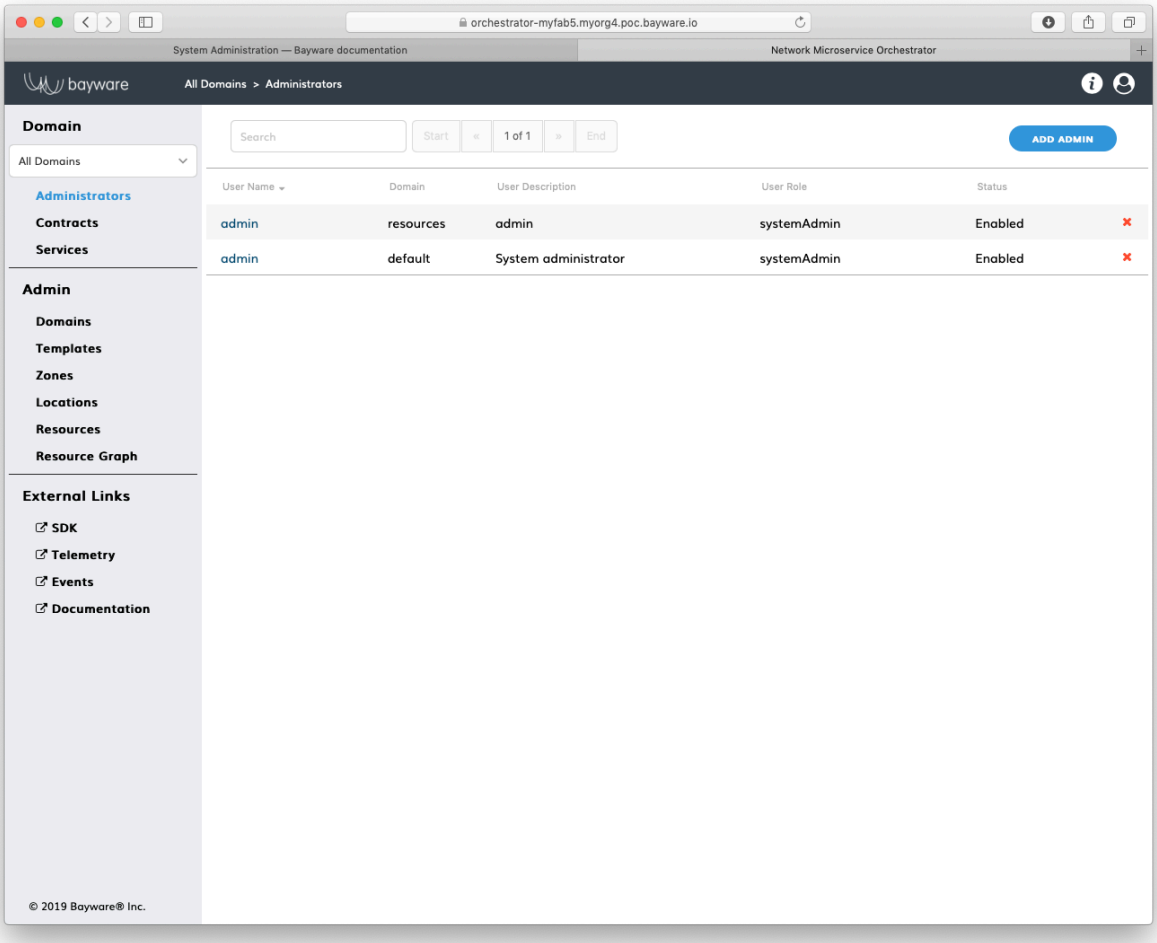


Fig. 25.6: List of Administrators

25.3.3 Using BWCTL-API

To create a new administrator from `bwctl-api`, run this command with the desired administrator name in given domain and role–in this example `admin@resources` and `systemAdmin` respectively–as the arguments:

```
]$ bwctl-api create administrator admin@resources --roles systemAdmin
```

You will be prompted to enter and repeat password:

```
Password:
Repeat for confirmation:
[2019-10-15 20:57:55.891]  Administrator 'admin' created successfully
```

Now, check the list of existing administrators by running this command:

```
]$ bwctl-api show domain
```

You should see output similar to:

```
---
apiVersion: policy.bayware.io/v1
kind: Batch
metadata:
  name: List of Administrators
spec:
- kind: Administrator
  metadata:
    user_domain: resources
    username: admin
  spec:
    is_active: true
    roles:
    - systemAdmin
    user_auth_method: LocalAuth
- kind: Administrator
  metadata:
    user_domain: default
    username: admin
  spec:
    is_active: true
    roles:
    - systemAdmin
    user_auth_method: LocalAuth
```

Note: When options are not specified, the `bwctl-api` tool applies default configuration settings. See BWCTL-API CLI Manual for specific details.

CHAPTER 26

Resource Connectivity Management

This document describes the management functions necessary for configuring resource connectivity policy with the BWCTL-API command-line tool or via a web interface.

To set up a connectivity policy for the processor and workload nodes, all you need to do is put nodes in security zones and connect zones when needed.

The steps below will guide you through the creation of zones and links between them.

26.1 Declare Location

A group of workloads is assigned to a security zone via a workload location. It allows the workloads to automatically build links with the processors assigned to the same zone.

The fabric manager automatically assigns a workload to a location at the workload configuration step.

Note: By default, the fabric manager uses the prefix of the workload VPC name as its location name, for example: **vpc-name:** `azr2-vpc-myfab5` ==> **location-name:** `azr2`

After configuration, the workload registers with the orchestrator. A workload always provides its location to the orchestrator during the registration step. The orchestrator automatically adds a newly received location name to its resource database.

To set up a zone policy before your workloads are registered, you need to declare a location.

26.1.1 Using Web Interface

To declare a location, click `Add Location` in the `Admin > Locations` section.

Fill out the fields on the `New Location` page:

location name desired location name;

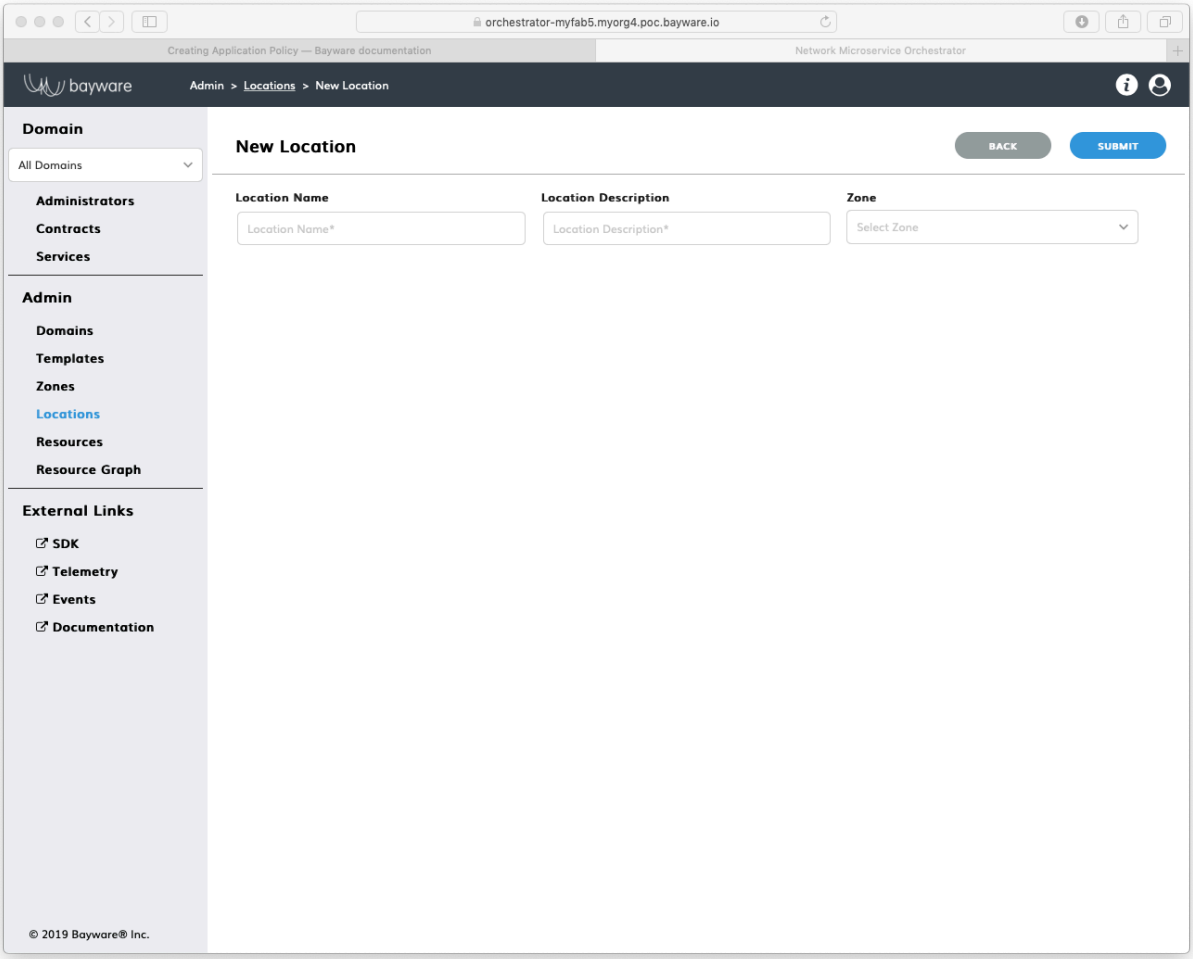


Fig. 26.1: Add New Location

description add description for location;

zone select zone for location–leave None to make decision later.

Submit the new location configuration. You should see the location appear in the list on the `Admin > Locations` page.

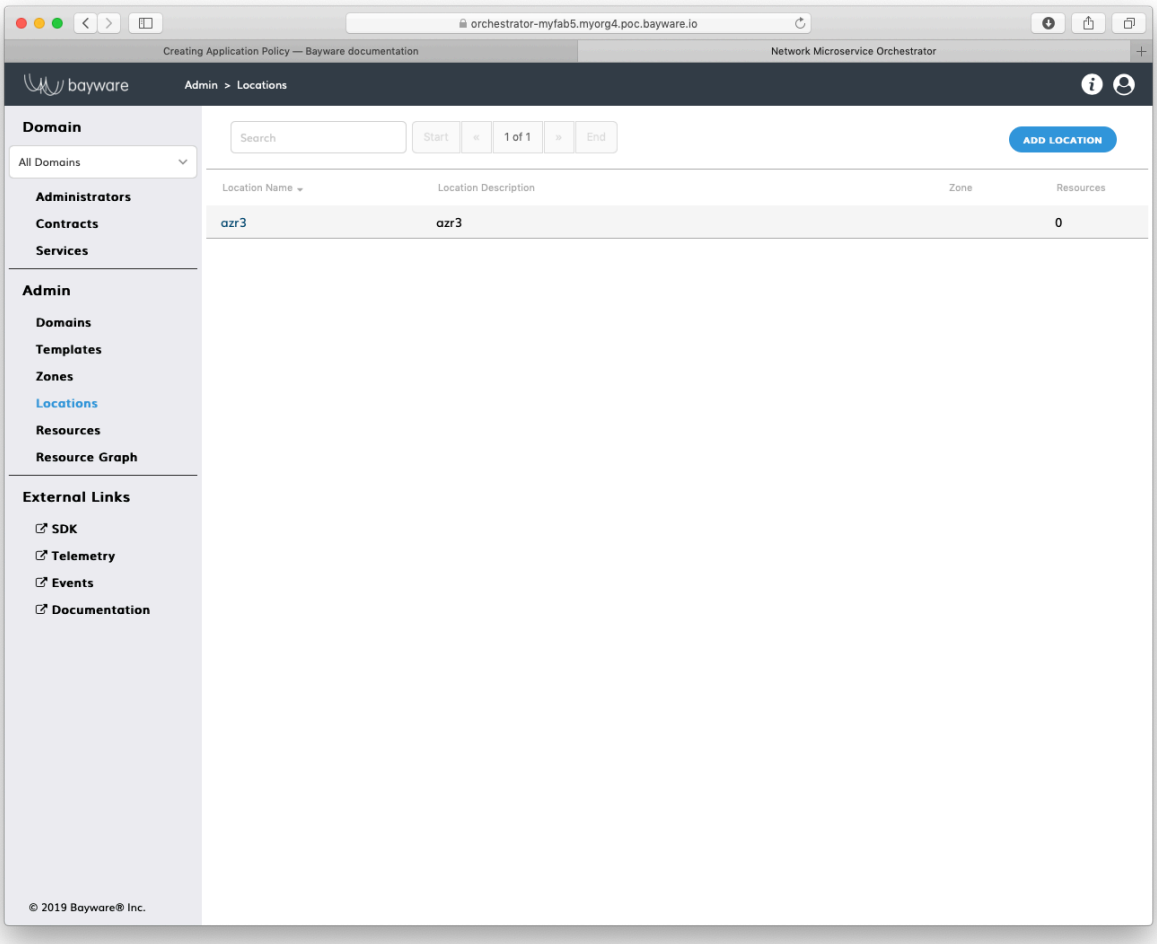


Fig. 26.2: List of Locations

26.1.2 Using BWCTL-API

To declare a location, run this command with the desired location name–in this example `azr3` –as an argument:

```
]$ bwctl-api create location azr3
```

You should see output similar to this:

```
[2019-10-17 22:48:34.362]  Location 'azr3' created successfully
```

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

To check the location configuration, run this command with the location name–in this example `azr3` –as an argument:

```
]$ bwctl-api show location azr3
```

You should see a new location specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Location
metadata:
  description: azr3
  name: azr3
spec: {}
```

26.2 Create Zone

Once you have deployed resources–in this example a VPC with one processor and three workload nodes–open the resource graph page and verify that the workload nodes are not connected to the processor node.

You need to set up a resource policy that permits the workload nodes to connect to the processor.

Note: To set up a resource policy for workloads, you need to create a zone and assign the location with workload nodes and at least one processor to this zone.

26.2.1 Set Up Zone

Using Web Interface

To add a new zone, click `Add Zone` in the `Admin > Zones` section.

Fill out the fields on the `New Zone` page:

zone name desired zone name;

description add description for zone.

Submit the new zone configuration. You should see the zone appear in the list on the `Admin > Zones` page.

Using BWCTL-API

To create a new zone, run this command with a desired zone name (any string without spaces)–in this example `azure-eastus` –as an argument:

```
]$ bwctl-api create zone azure-eastus
```

You should see output similar to this:

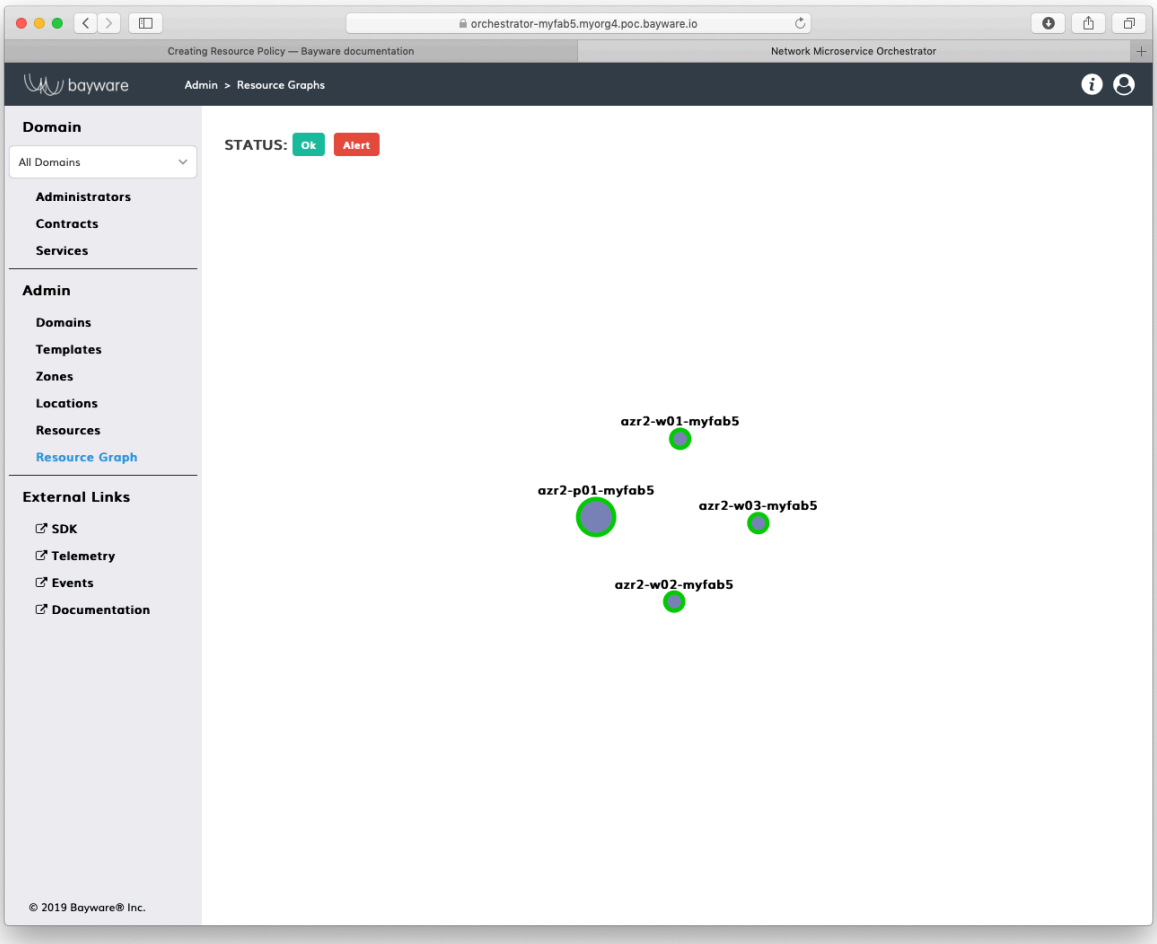


Fig. 26.3: Resource Graph before Policy Setup

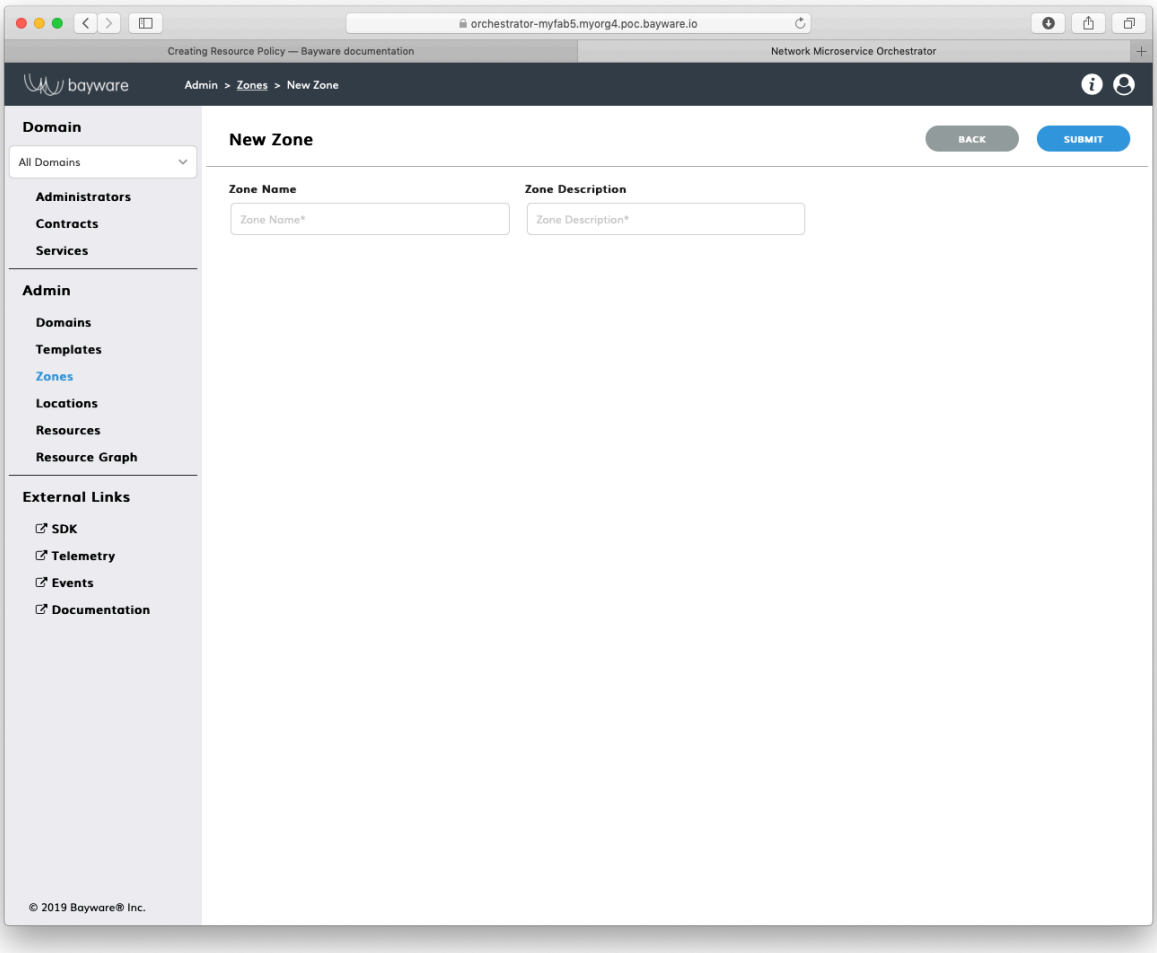


Fig. 26.4: Add New Zone

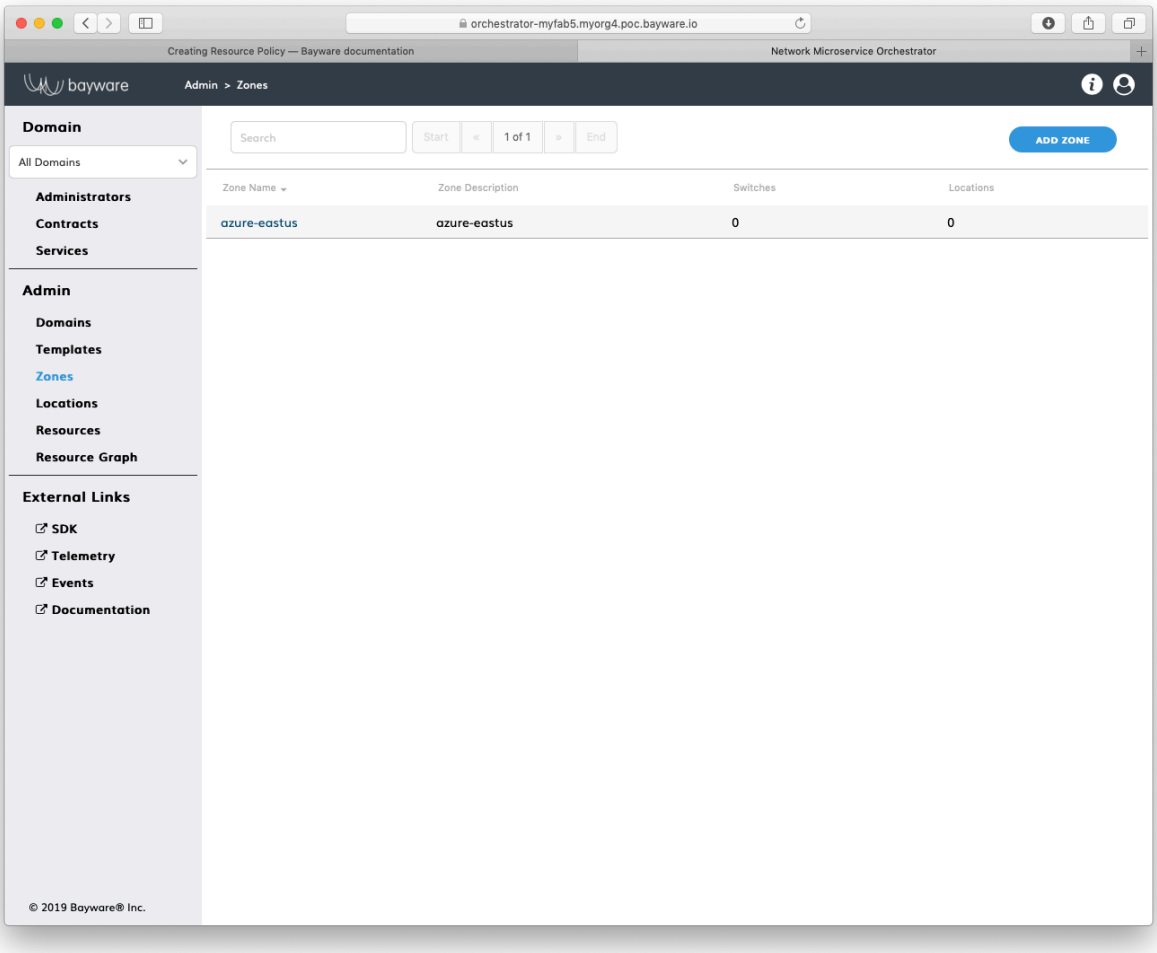


Fig. 26.5: List of Zones

```
[2019-10-17 22:58:33.609]  Zone 'azure-eastus' created successfully
```

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

Check a new zone configuration by running this command with the zone name–in this example `azure-eastus` –as an argument:

```
]$ bwctl-api show zone azure-eastus
```

You should see a new zone specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Zone
metadata:
  description: azure-eastus
  name: azure-eastus
spec:
  locations: []
  processors: []
```

26.2.2 Add Processor to Zone

Using Web Interface

To add a processor to the zone, click on the zone name in the `Admin > Zones` section–in this example `azure-eastus`. On the zone page, click `Add Processor`.

Fill out the fields on the `New Processor` page:

processor name name of the processor that will secure workloads in the zone;

tunnel IPs type of IP addresses– `Private` or `Public` –the processor will use to communicate with workloads in the zone;

IPse to encrypt communication– `yes` or `no` –between the processor and workloads in the zone;

priority processor usage priority– `High` or `Low` –for workloads in the zone.

Submit the configuration. You should see the processor appear in the list of zone processors on the `Admin > Zones > azure-eastus` page.

Using BWCTL-API

To assign a processor to the zone, run this command with the processor name–in this example `azr2-p01-myfab5` –as an argument:

```
]$ bwctl-api update zone azure-eastus -a azr2-p01-myfab5
```

You should see output similar to this:

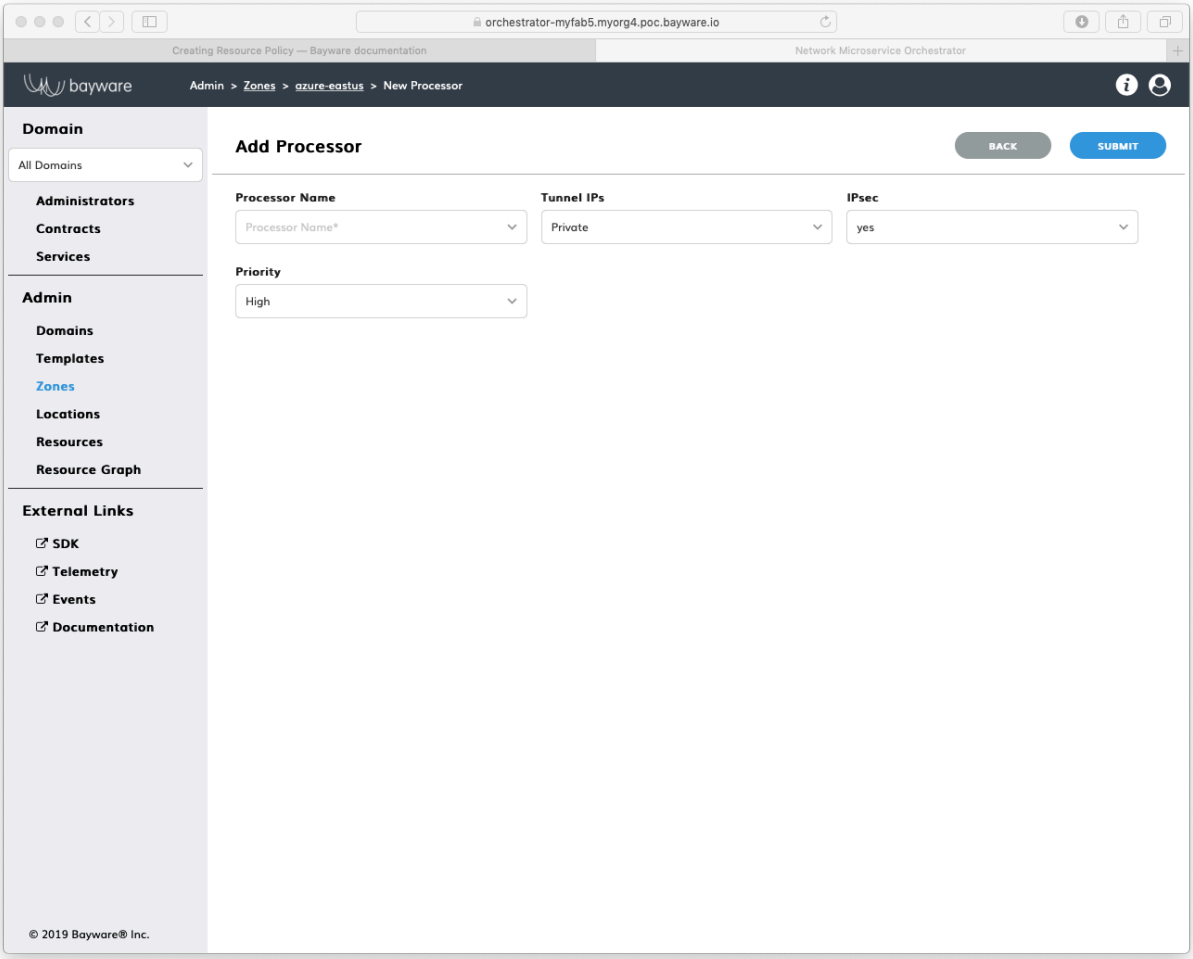


Fig. 26.6: Add Processor to Zone

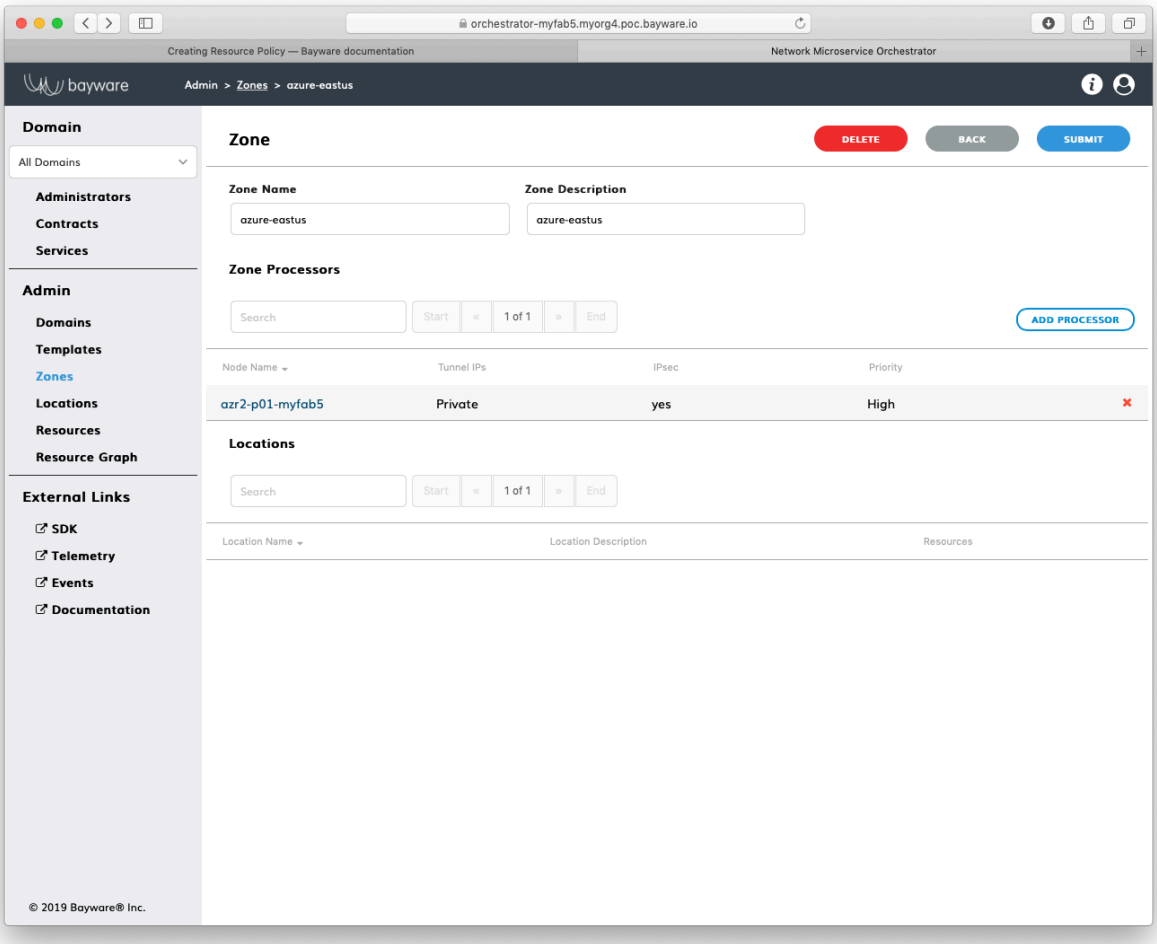


Fig. 26.7: List of Zone Processors

```
[2019-10-17 23:05:25.307]  Processor 'azr2-p01-myfab5' assigned to zone 'azure-eastus'
[2019-10-17 23:05:25.307]  Zone 'azure-eastus' updated successfully
```

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

Check a new zone configuration by running this command with the zone name–in this example `azure-eastus` –as an argument:

```
]$ bwctl-api show zone azure-eastus
```

You should see that the zone specification now includes the processor:

```
---
apiVersion: policy.bayware.io/v1
kind: Zone
metadata:
  description: azure-eastus
  name: azure-eastus
spec:
  locations: []
  processors:
  - ipsec_enable: true
    name: azr2-p01-myfab5
    tunnel_ip_type: private
```

26.2.3 Add Workload to Zone

Using Web Interface

You will use a location to add a workload to a zone.

To add a location with your workload nodes to the zone, click on the location name in the `Admin > Locations` section–in this example `azr2`. On the location page, click on the dropdown menu titled `Zone`.

Select the zone–in this example `azure-eastus` –and submit the configuration.

To verify, go to the `Admin > Zones > azure-eastus` page and find the location name in the list of zone locations.

Using BWCTL-API

To assign a location with your workload nodes to the zone, run this command with the location name–in this example `azr2` –as an argument:

```
]$ bwctl-api update location azr2 -z azure-eastus
```

You should see output similar to this:

```
[2019-10-17 23:32:54.982]  Location 'azr2' updated successfully
```

Check the zone configuration by running this command:

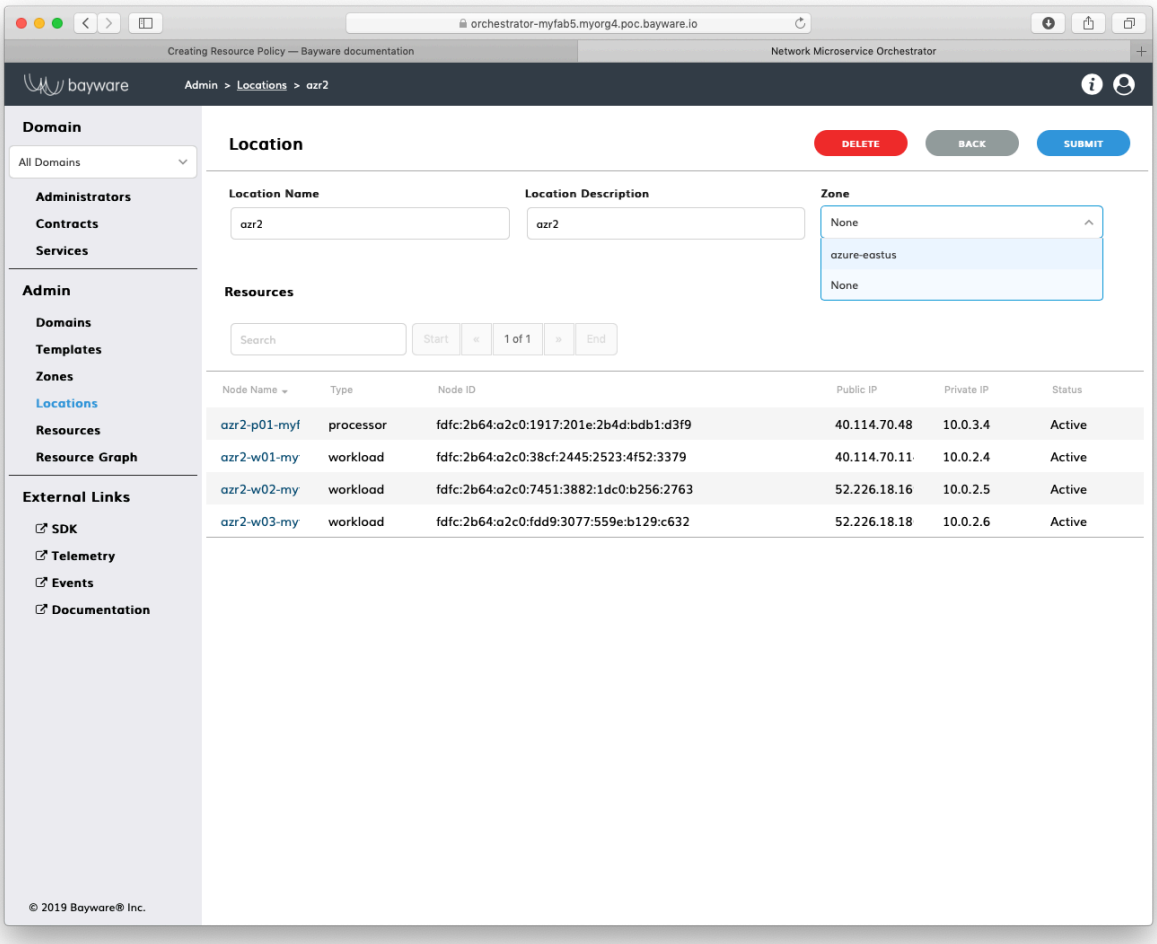


Fig. 26.8: Add Location to Zone

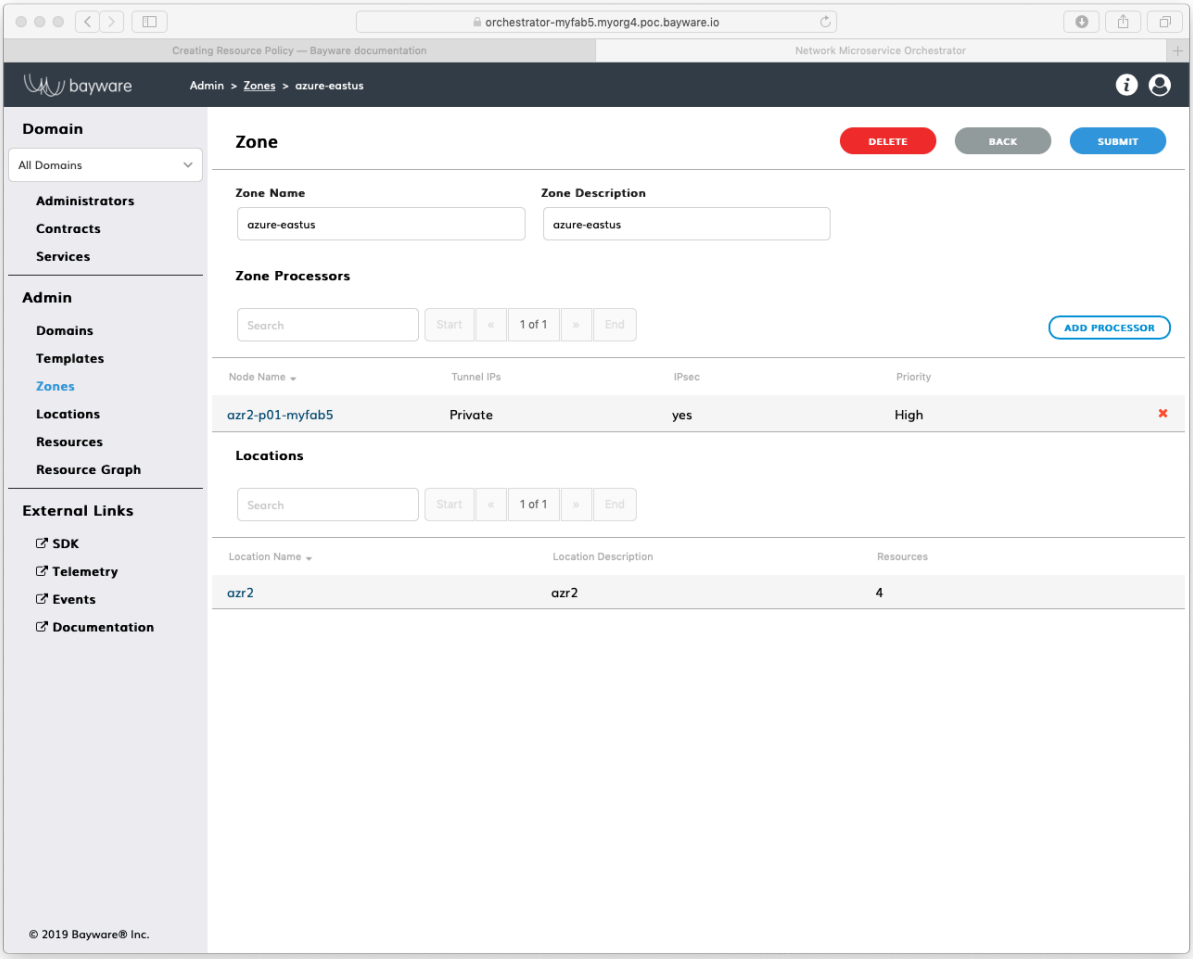


Fig. 26.9: List of Zone Locations

```
]$ bwctl-api show zone azure-eastus
```

You should see that the zone specification now includes the location:

```
---
apiVersion: policy.bayware.io/v1
kind: Zone
metadata:
  description: azure-eastus
  name: azure-eastus
spec:
  locations:
  - name: azr2
  processors:
  - ipsec_enable: true
    name: azr2-p01-myfab5
    tunnel_ip_type: private
```

At this point, you can open the resource graph page and see that the workloads now are connected to the processor.

26.3 Connect Zones

26.3.1 Declare Processor

To connect two zones, you need to set up a link between the processors serving these zones.

You can describe a link between existing processors or processors you are planning to spin up, but haven't yet created. If a processor doesn't exist yet, you need to declare it before configuring the link.

Using Web Interface

To declare a processor, click `Add Resource` in the `Admin > Resources` section.

Fill out the fields on the **New Resource** page:

node name desired name of node;

node type type of node– `processor` or `workload`;

location expected node location.

Submit the configuration. You should see the processor appear on the `Admin > Resources` page with the status `Init`.

Using BWCTL-API

To declare a processor, run this command with the expected node name and its location–in this example `gcp1-p01-myfab2` and `azr3` respectively–as arguments:

```
]$ bwctl-api create resource azr3-p01-myfab5 -type processor -l azr3
```

You should see output similar to this:

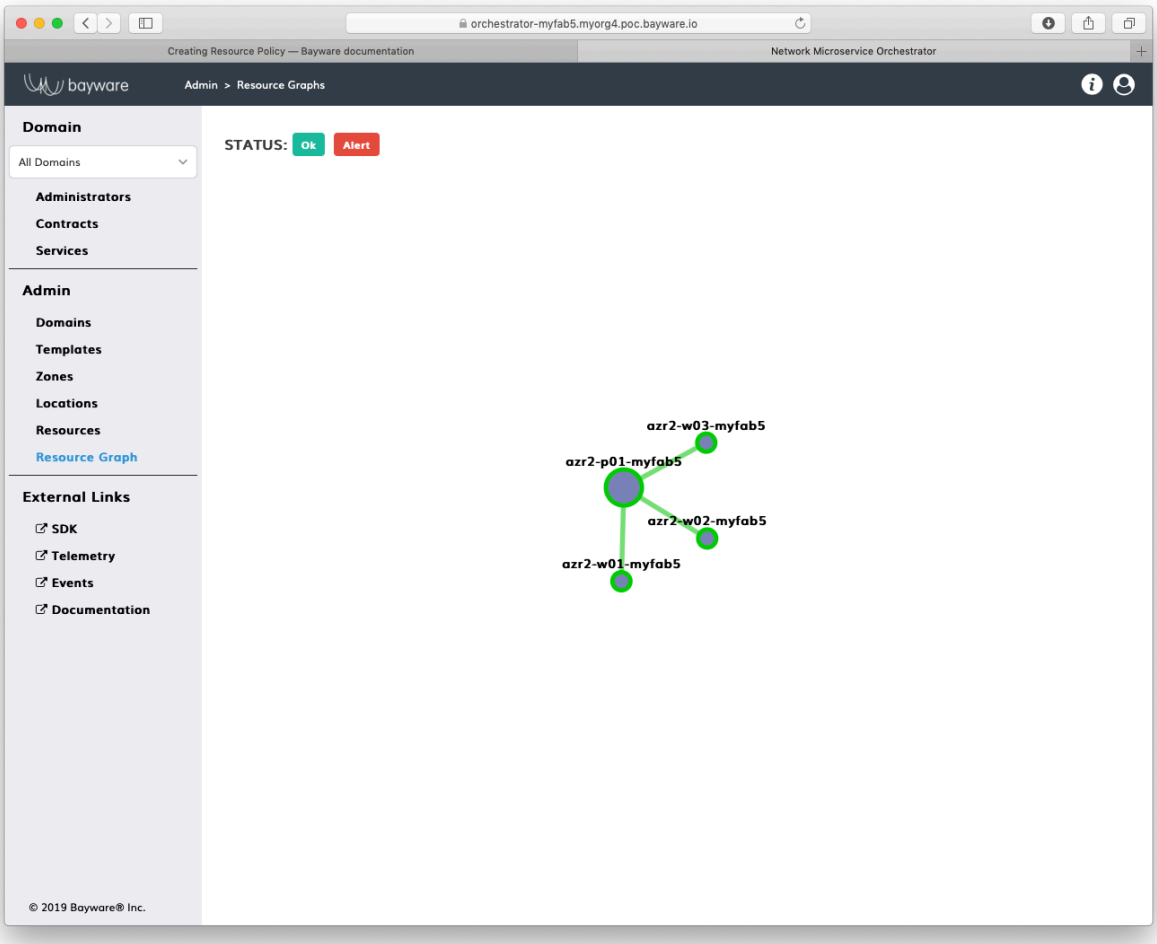


Fig. 26.10: Resource Graph after Zone Configured

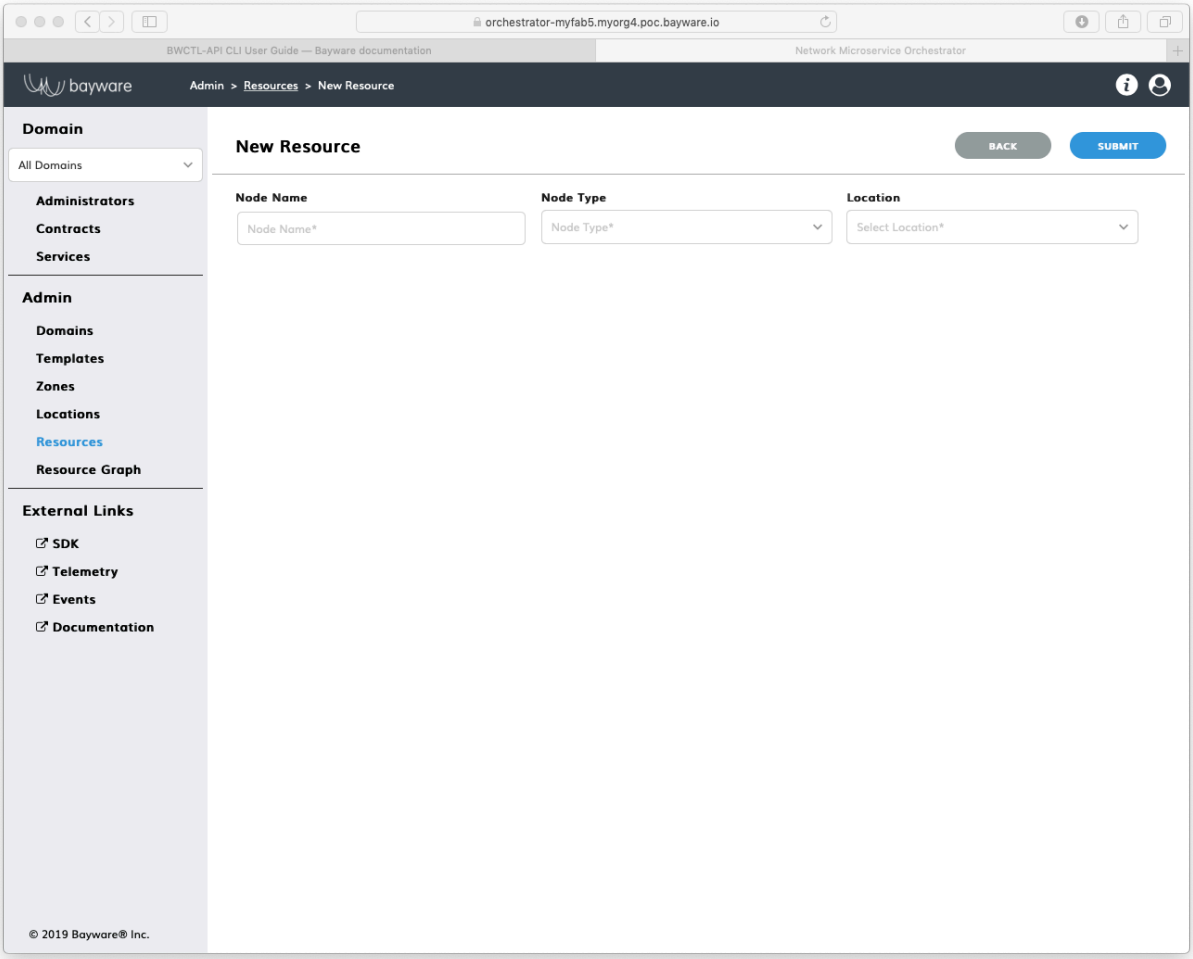


Fig. 26.11: Add New Resource

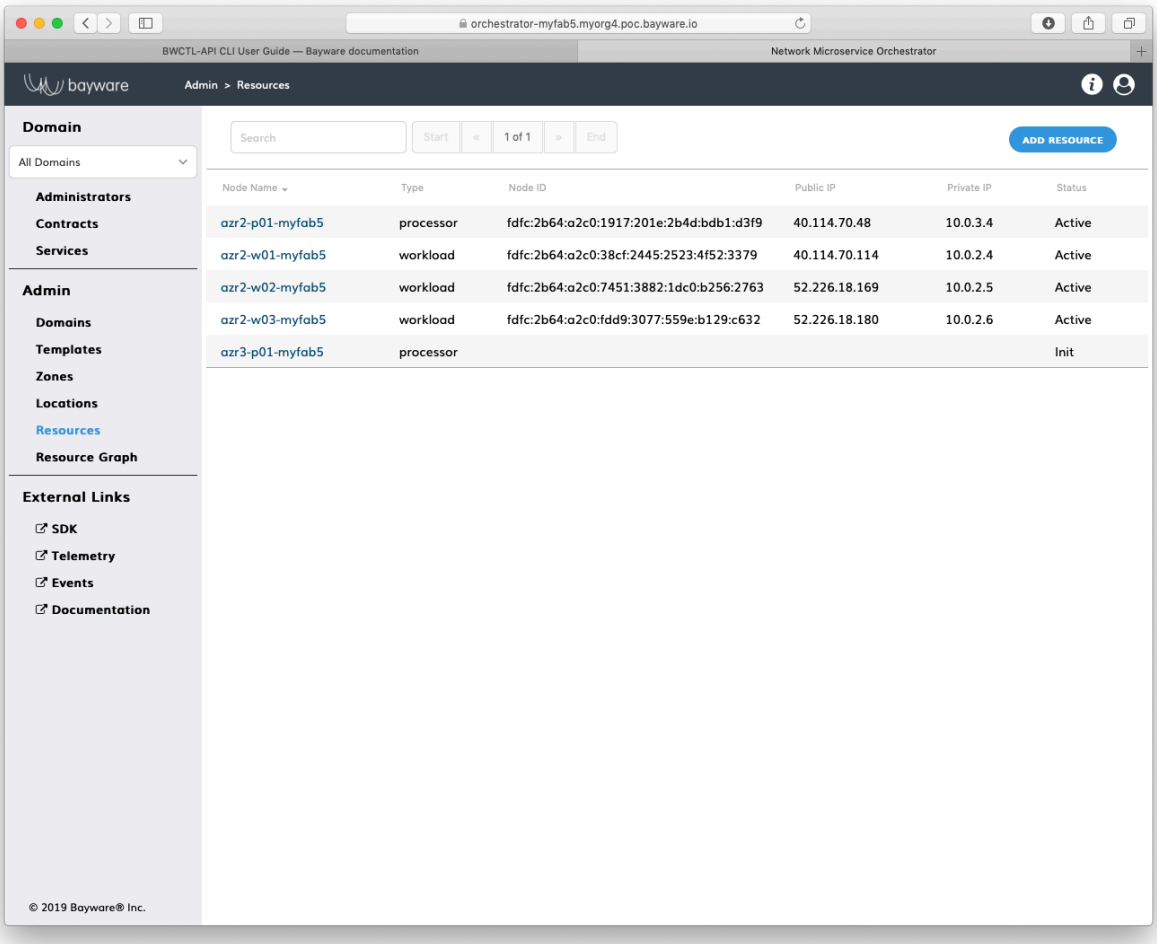


Fig. 26.12: List of Resources

```
[2019-10-18 17:03:00.261]  Resource 'gcp1-p01-myfab2' created successfully
```

Check the resource configuration by running this command:

```
]$ bwctl-api show resource azr3-p01-myfab5
```

You should see that the zone specification now includes the location:

```
---
apiVersion: policy.bayware.io/v1
kind: Resource
metadata:
  name: azr3-p01-myfab5
spec:
  location: azr3
  type: processor
  status: Init
```

26.3.2 Specify Link

Using Web Interface

To specify a link between processors, click `Add Link` in the `Admin > Resources > azr2-p01-myfab5` section.

Fill out the fields on the `New Link` page:

link name will be auto generated after you click Submit;

link description add description to link;

link status administrative status of link– `Enabled` or `Disabled`;

remote node name name of remote processor;

tunnel IPs type of IP addresses– `Private` or `Public` –the processor will use to communicate with another processor;

IPse to encrypt communication– `yes` or `no` –between the processors;

cost link cost from 1 to 10.

Submit the configuration. You should see the link appear on the `Admin > Resources > azr2-p01-myfab5` page.

Using BWCTL-API

To specify a link between processors, run this command with the source and target processor node names–in this example `azr2-p01-myfab5` and `azr3-p01-myfab5` –as arguments:

```
]$ bwctl-api create link -s azr2-p01-myfab5 -t azr3-p01-myfab5
```

You should see output similar to this:

```
[2019-09-26 19:30:52.559]  Link 'azr2-p01-myfab5_azr3-p01-myfab5' created successfully
```

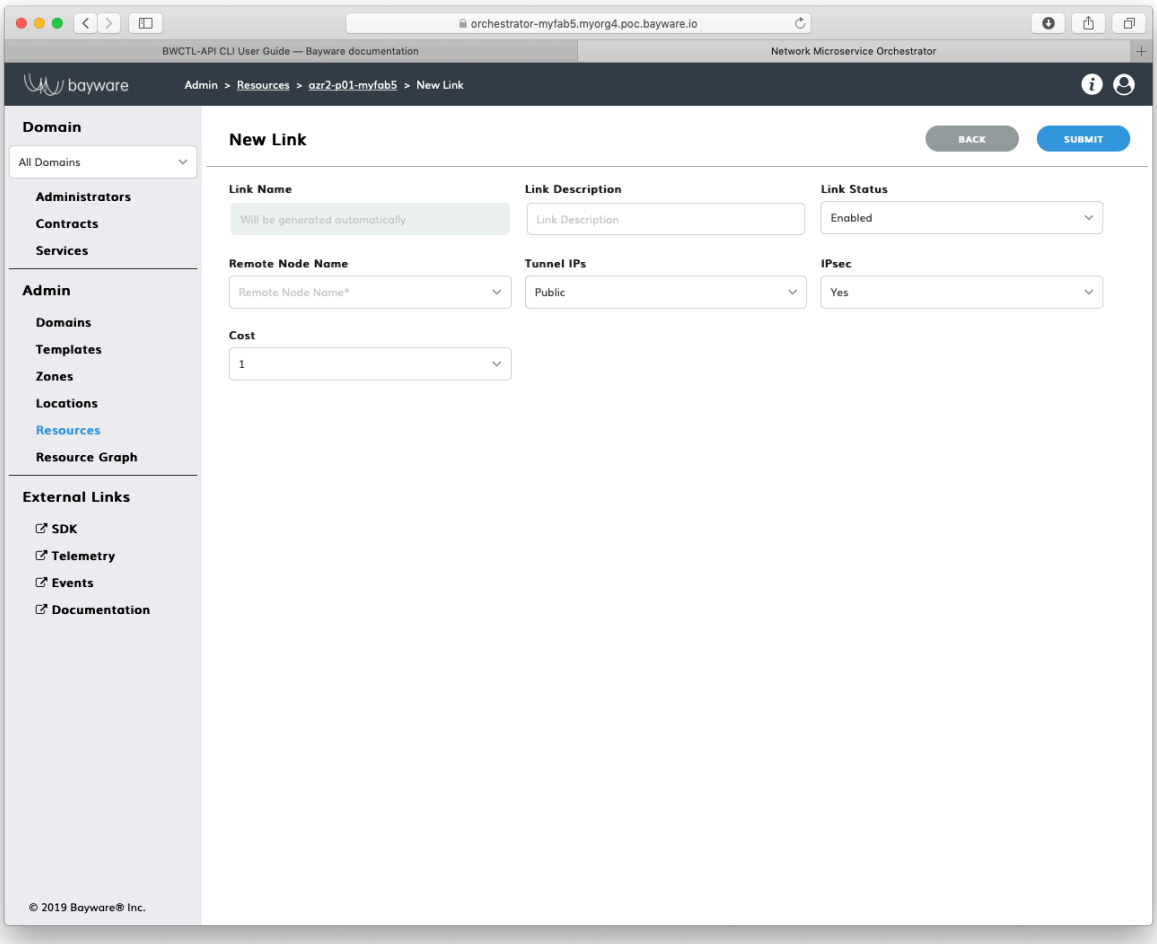


Fig. 26.13: Add New Link

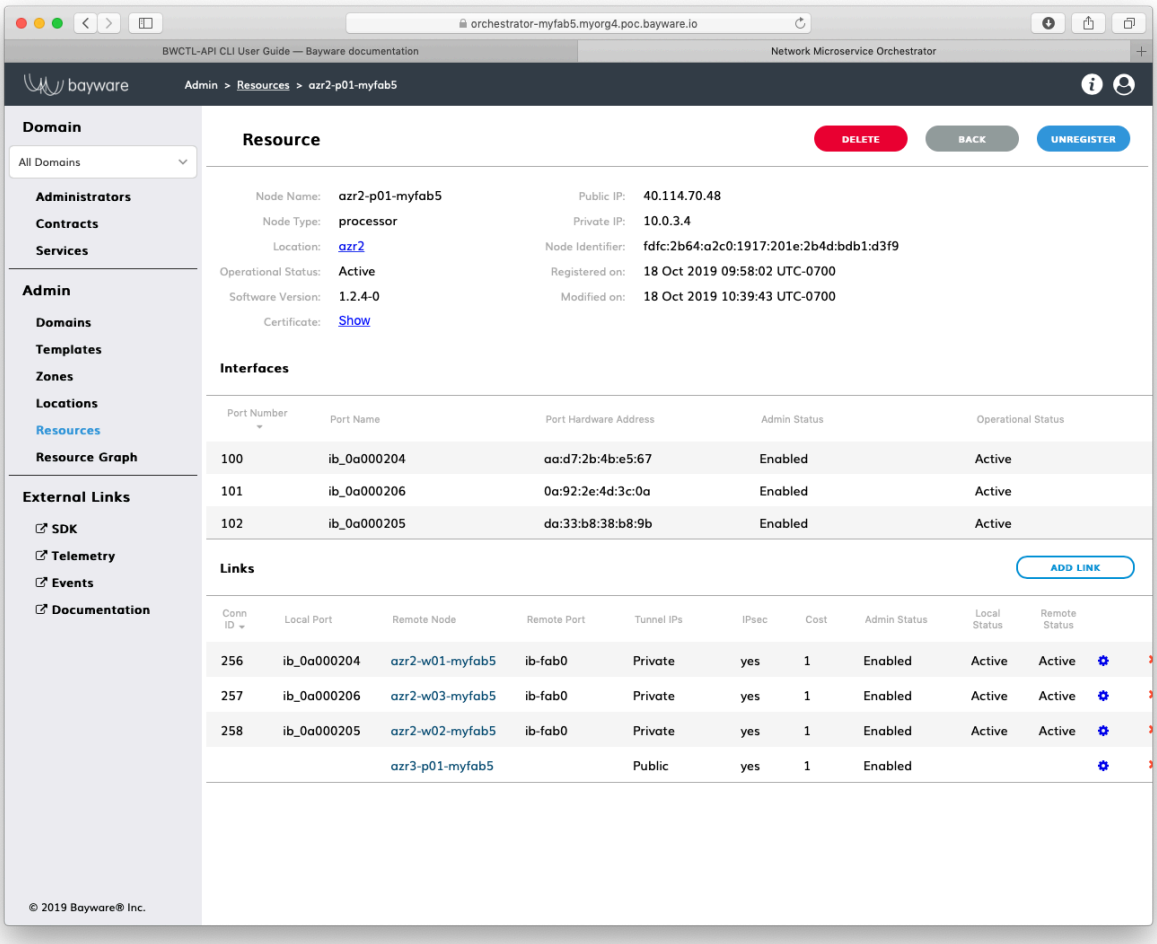


Fig. 26.14: List of Links

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

Note: The auto-generated link name is always built as follows: **node-name1_node-name2**, wherein node names in the string are placed in alphabetical order.

Check the link configuration by running this command with the link auto-generated name–in this example `azr2-p01-myfab5_azr3-p01-myfab5` –as an argument:

```
]$ bwctl-api show link azr2-p01-myfab5_azr3-p01-myfab5
```

You should see a new link specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Link
metadata:
  name: azr2-p01-myfab5_azr3-p01-myfab5
spec:
  admin_status: true
  cost: 1
  ipsec_enable: true
  source_node: azr2-p01-myfab5
  status: active
  target_node: azr3-p01-myfab5
  tunnel_ip_type: public
```

26.4 Working with Batches

To set up a resource policy, you can also use batch files.

Export the resource policy from an existing zone and replace the existing entity names with the names allocated for the new zone.

Export the existing zone policy by running this command with the zone and output file names–in this example `azr2` and `new-resource-policy` respectively–as arguments:

```
]$ bwctl-api show zone azure-eastus > new-resource-policy.yml
```

Open the file in your favorite editor, e.g. `nano`:

```
]$ nano new-resource-policy.yml
```

Add location, resource, and link specifications to the new zone specification.

Note: While editing, you need to provide new zone, location, and processor names.

After editing, your file should have content similar to:

```
---
apiVersion: policy.bayware.io/v1
kind: Batch
metadata:
  name: New Resource Policy
spec:
- kind: Location
  metadata:
    description: azr3
    name: azr3
  spec:
    count_resources: 0
- kind: Resource
  metadata:
    name: azr3-p01-myfab5
  spec:
    location: azr3
    type: processor
    status: Init
- kind: Zone
  metadata:
    description: azure-westus
    name: azure-westus
  spec:
    locations:
    - name: azr3
    processors:
    - ipsec_enable: true
      name: azr3-p01-myfab5
      tunnel_ip_type: private
- kind: Link
  metadata:
    name: azr2-p01-myfab5_azr3-p01-myfab5
  spec:
    admin_status: true
    cost: 1
    ipsec_enable: true
    source_node: azr2-p01-myfab5
    status: active
    target_node: azr3-p01-myfab5
    tunnel_ip_type: public
```

Now, run the policy deployment using the batch file name–in this example `new-resource-policy.yml` –as an argument:

```
]$ bwctl-api create batch new-resource-policy.yml
```

You should see output similar to:

```
[2019-10-18 19:18:13.212]  Location 'azr3' created successfully
[2019-10-18 19:18:13.405]  Resource 'azr3-p01-myfab5' created successfully
[2019-10-18 19:18:13.745]  Zone 'azure-westus' created successfully
[2019-10-18 19:18:13.745]  Location 'azr3' updated in zone 'azure-westus'
```

(continues on next page)

(continued from previous page)

```
[2019-10-18 19:18:13.745]  Processor 'azr3-p01-myfab5' assigned to zone 'azure-westus'
[2019-10-18 19:18:14.076]  Link from  'azr2-p01-myfab5' to 'azr3-p01-myfab5' created␣
↪successfully
```

Note: At this point, you can deploy resources in the new zone. Each workload will automatically connect to a zone processor. Also, the new processor will automatically build a link with the existing processor.

CHAPTER 27

Service Connectivity Management

This document describes the management functions necessary for configuring service connectivity policy with the BWCTL-API command-line tool or via a web interface.

To set up an application policy, all you need to do is upload a communication rule template and describe an application service graph.

The steps below will guide you through the uploading of a template and the creation of a service graph.

27.1 Upload Template

27.1.1 Using Web-interface

To create a new template, lick `Add Template` in the `Admin > Templates` section.

Fill out the fields on the `New Template` page:

template name desired template name;

description add description for template;

status select template administrative status– `Enabled` or `Disabled`;

orientation select orientation– `Directed` or `Undirected` –to describe the relationships between two template roles, `Directed` will be represented as an arrow on a service graph;

multicast is multicast– `False` or `True`.

Submit the new template. You should see the template appear in the list on the `Admin > Templates` page.

Note: At this point, you would need to configure two template roles. Click on the template name and set up each role. See the SDK documentation for specific details.

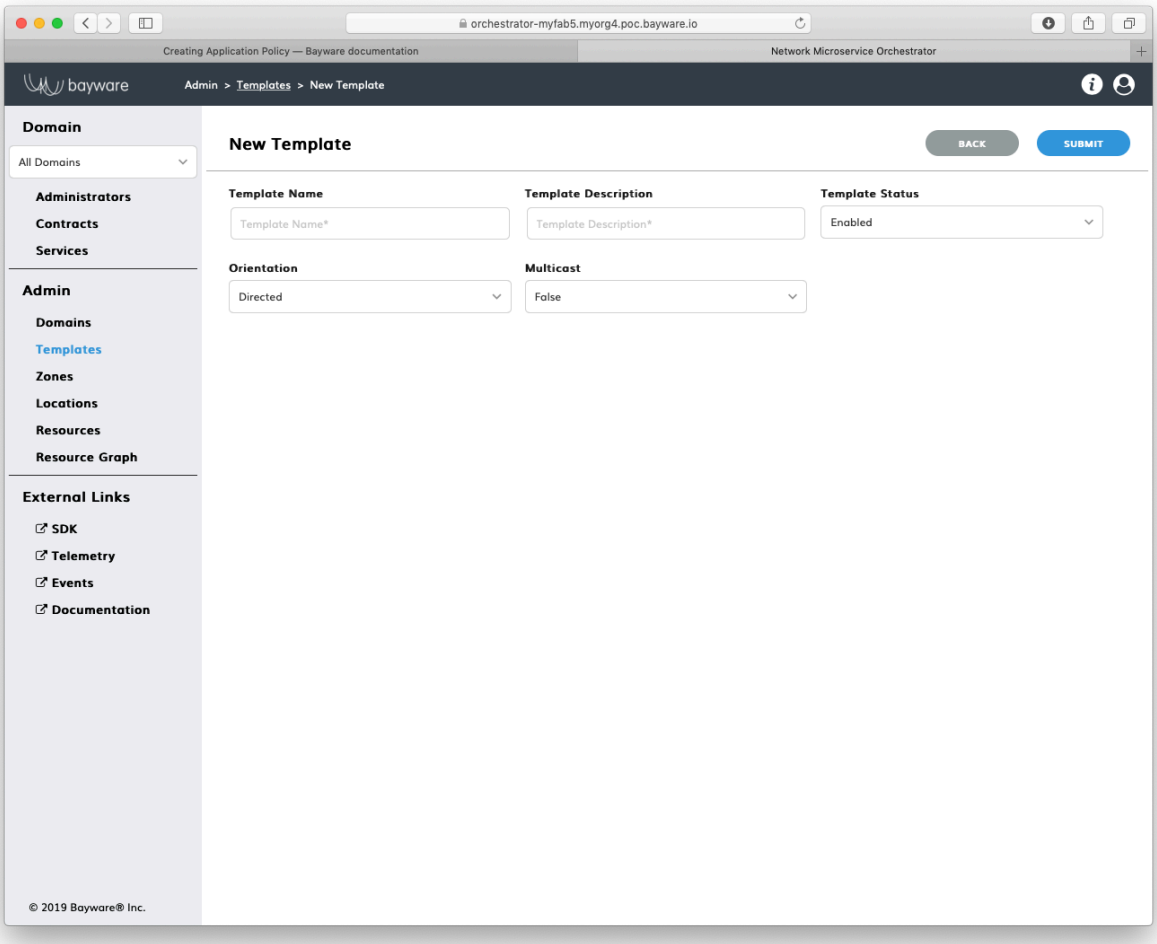


Fig. 27.1: Add New Template

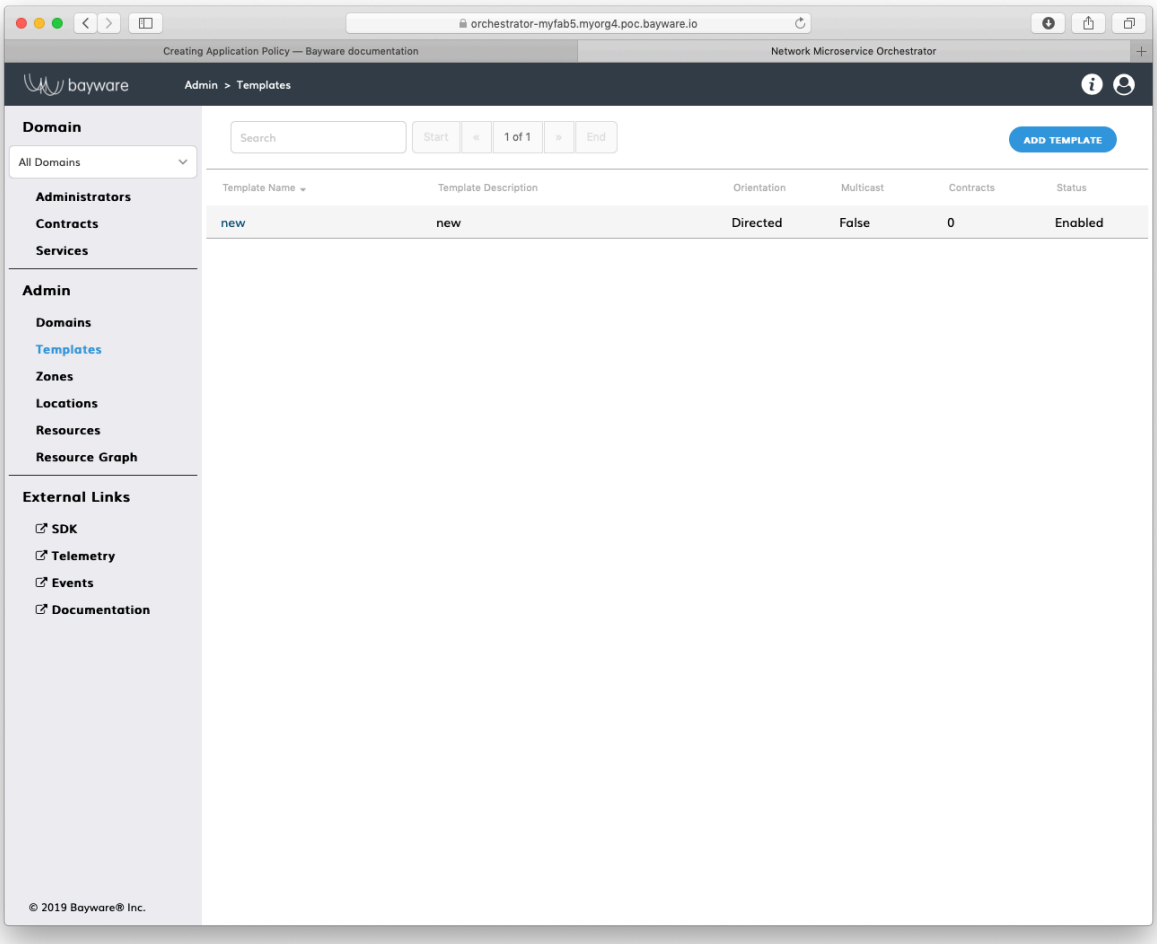


Fig. 27.2: List of Templates

27.1.2 Using BWCTL-API

To upload a default template that comes with BWCTL-API, run this command:

```
]$ bwctl-api create template default
```

You should see this output:

```
[2019-10-18 22:01:02.939]  Template 'default' created successfully
```

Note: To find more templates available for upload, go to the SDK section of the orchestrator.

Check the template specification by running this command with the template name–in this example `default` –as an argument:

```
]$ bwctl-api show template default
```

You should see the default template specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Template
metadata:
  description: Exchange data between originators and responders from any VPCs
  name: default
spec:
  domains: []
  enabled: true
  is_multicast: false
  orientation: 1
  roles:
  - code_binary:␣
↪409C470100E7846300E000EF0A700793C11C004000EF409C470500E7846300C000EF579DC11C004000EF409C00178713C0989
    code_map:
      originator: 0
    description: null
    id: 3
    ingress_rules_default:
    - {}
    name: originator
    path_binary: '000000000001'
    path_params_default: {}
    program_data_default:
      params:
      - name: hopsCount
        value: 0
      ppl: 0
    propagation_interval_default: 5
    role_index: 0
  - code_binary:␣
↪409C470100E7846300E000EF0A700793C11C004000EF409C470500E7846300C000EF579DC11C004000EF409C00178713C0989
    code_map:
```

(continues on next page)

(continued from previous page)

```
    responder: 0
  description: null
  id: 4
  ingress_rules_default:
  - {}
  name: responder
  path_binary: '000000000001'
  path_params_default: {}
  program_data_default:
    params:
    - name: hopsCount
      value: 0
    ppl: 0
  propagation_interval_default: 5
  role_index: 1
```

27.2 Create Service Graph

27.2.1 Create Domain

Using Web-interface

To create a namespace for your application policy, lick `Add Domain` in the `Admin > Domains` section.

Fill out the fields on the `New Domain` page:

domain name desired domain name;

domain description add description for domain;

type select domain type– `Application` or `Administrative`;

auth method select authentication method for domain administrators– `LocalAuth` or `LDAP`.

Submit the new domain configuration. You should see the domain appear in the list on the `Admin > Domains` page.

Using BWCTL-API

To create a namespace for your application policy, run this command with the desired domain name (any string without spaces)–in this example `myapp` –as an argument:

```
]$ bwctl-api create domain myapp
```

You should see output similar to this:

```
[2019-10-19 00:34:45.616]  Domain 'myapp' created successfully
```

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

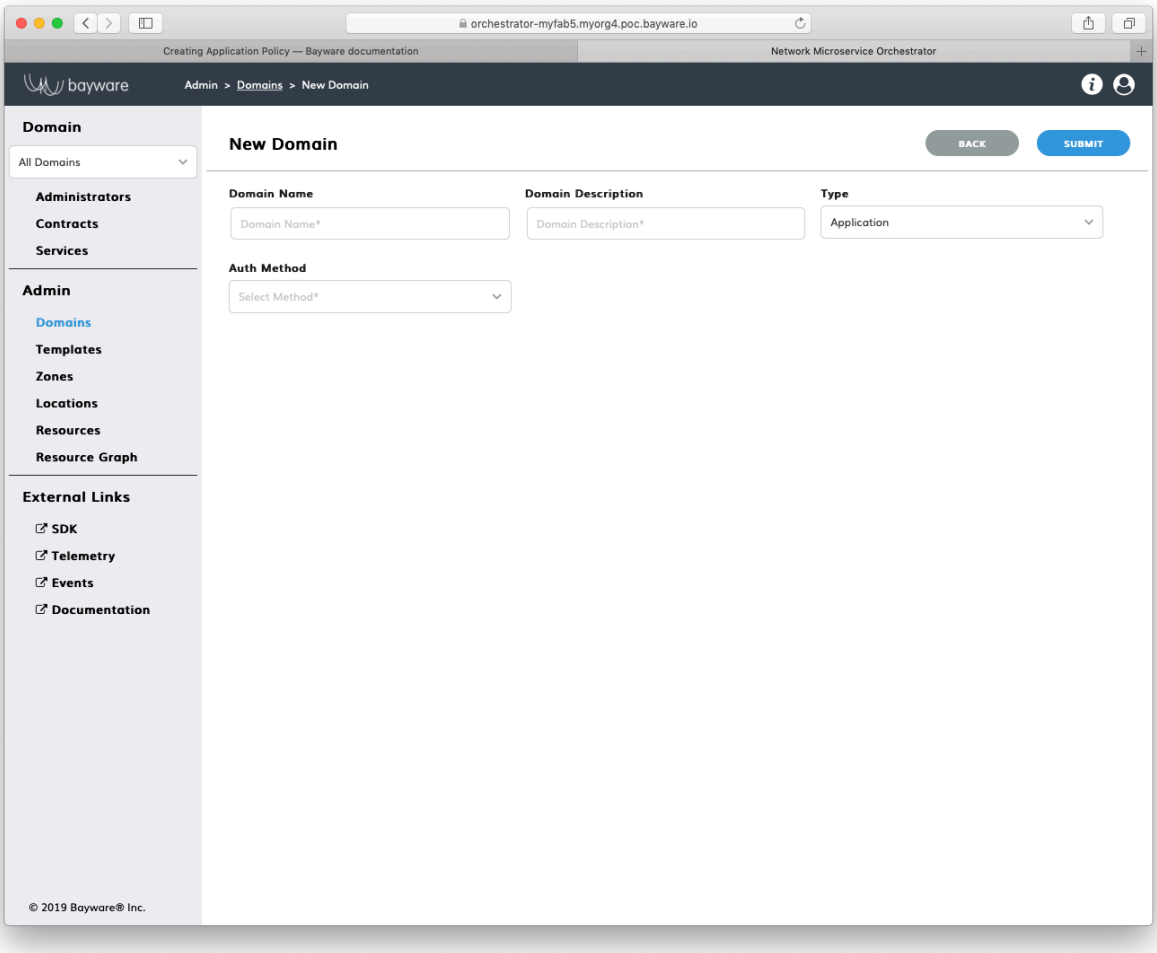


Fig. 27.3: Add New Domain

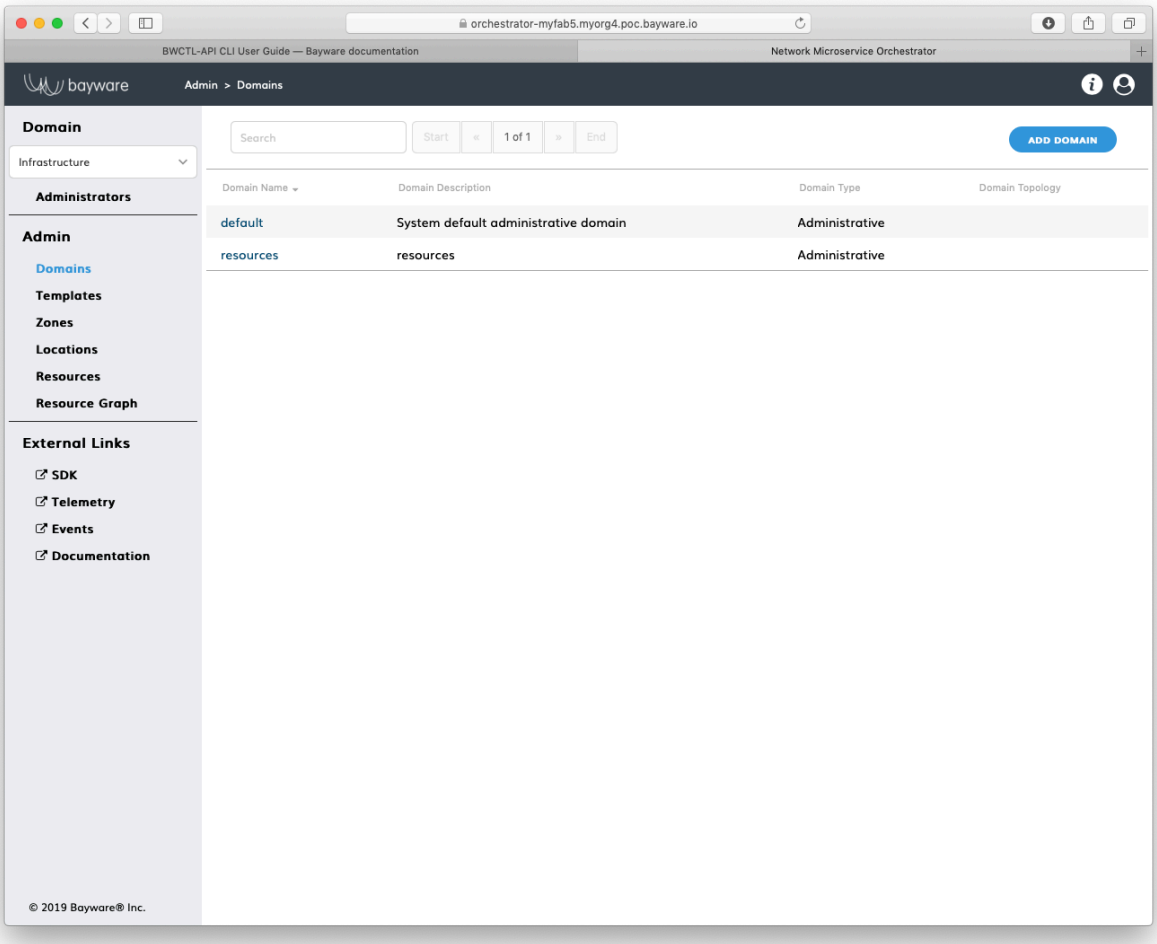


Fig. 27.4: List of Domains

To check the domain configuration, run this command with the domain name–in this example `myapp` –as an argument:

```
]$ bwctl-api show domain myapp
```

You should see a new domain specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Domain
metadata:
  domain: myapp
  domain_description: myapp
spec:
  auth_method:
  - LocalAuth
  domain_type: Application
```

27.2.2 Specify Contract

Using Web-interface

To specify a security segment for your application, lick `Add Contract` in the `myApp > Contracts` section.

Fill out the fields on the `New Contract` page:

contract name desired contract name;

contract description add description for contract;

contract status in which status contract to be created– `Enabled` or `Disabled`;

domain select domain for contract;

template select template for contract.

Submit the new contract configuration. You should see the contract appear in the list on the `myApp > Contracts` page.

Using BWCTL-API

To specify a security segment for your application, run this command with a desired contract name (any string without spaces) preceding the domain name–in this example `frontend@myapp` –as an argument:

```
]$ bwctl-api create contract frontend@myapp
```

You should see output similar to this:

```
[2019-10-19 00:36:51.590]  Contract 'frontend@myapp' created successfully
```

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

To check the contract configuration, run this command with the `contract@domain` –in this example `frontend@myapp` –as an argument:

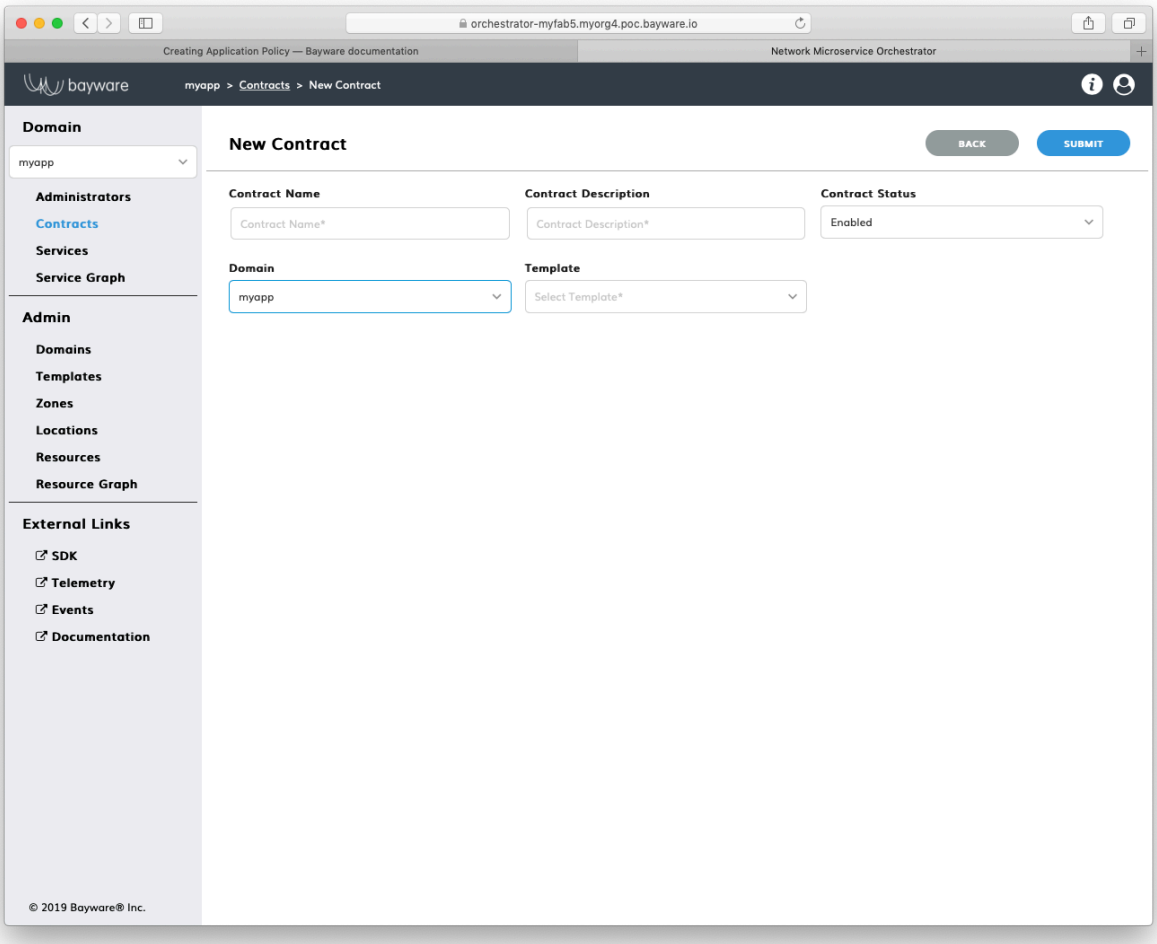


Fig. 27.5: Add New Contract

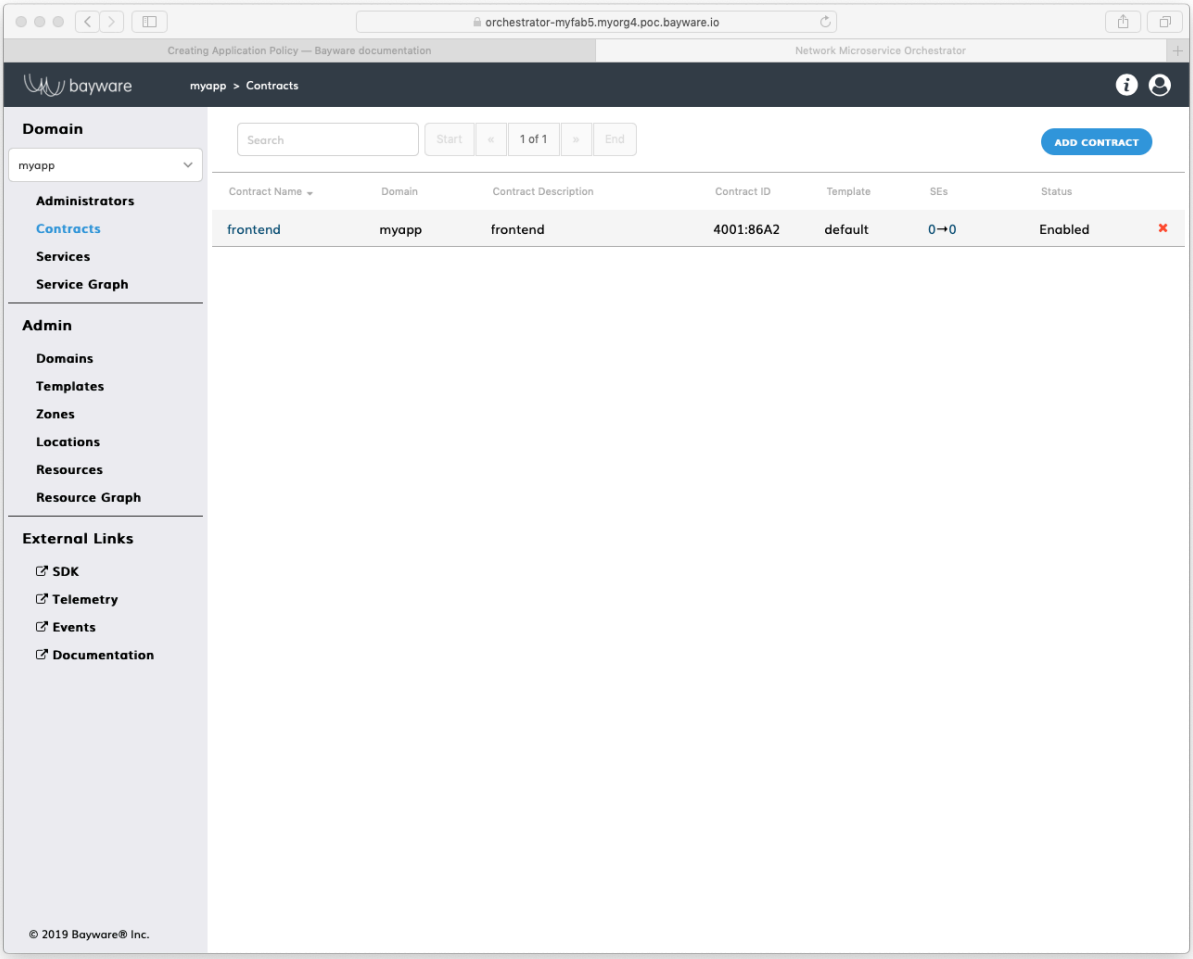


Fig. 27.6: List of Contracts

```
]$ bwctl-api show contract frontend@myapp
```

You should see a new contract specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Contract
metadata:
  description: frontend
  domain: myapp
  name: frontend
spec:
  contract_roles:
  - cfg_hash: 69141fa83039b5ee8d18adf364dd2835
    description: null
    id: 1
    ingress_rules:
    - {}
    name: originator
    path_params: {}
    port_mirror_enabled: false
    program_data:
      params:
      - name: hopsCount
        value: 0
      ppl: 0
    propagation_interval: 5
    role_index: 0
    service_rdn: originator.frontend.myapp
    stat_enabled: false
  - cfg_hash: ff5f3105716821fdbdfb2a6260d6d274
    description: null
    id: 2
    ingress_rules:
    - {}
    name: responder
    path_params: {}
    port_mirror_enabled: false
    program_data:
      params:
      - name: hopsCount
        value: 0
      ppl: 0
    propagation_interval: 5
    role_index: 1
    service_rdn: responder.frontend.myapp
    stat_enabled: false
  enabled: true
  template: default
```

27.2.3 Name Service

Using Web-interface

To specify a new application service, lick `Add Service` in the `myApp > Services` section.

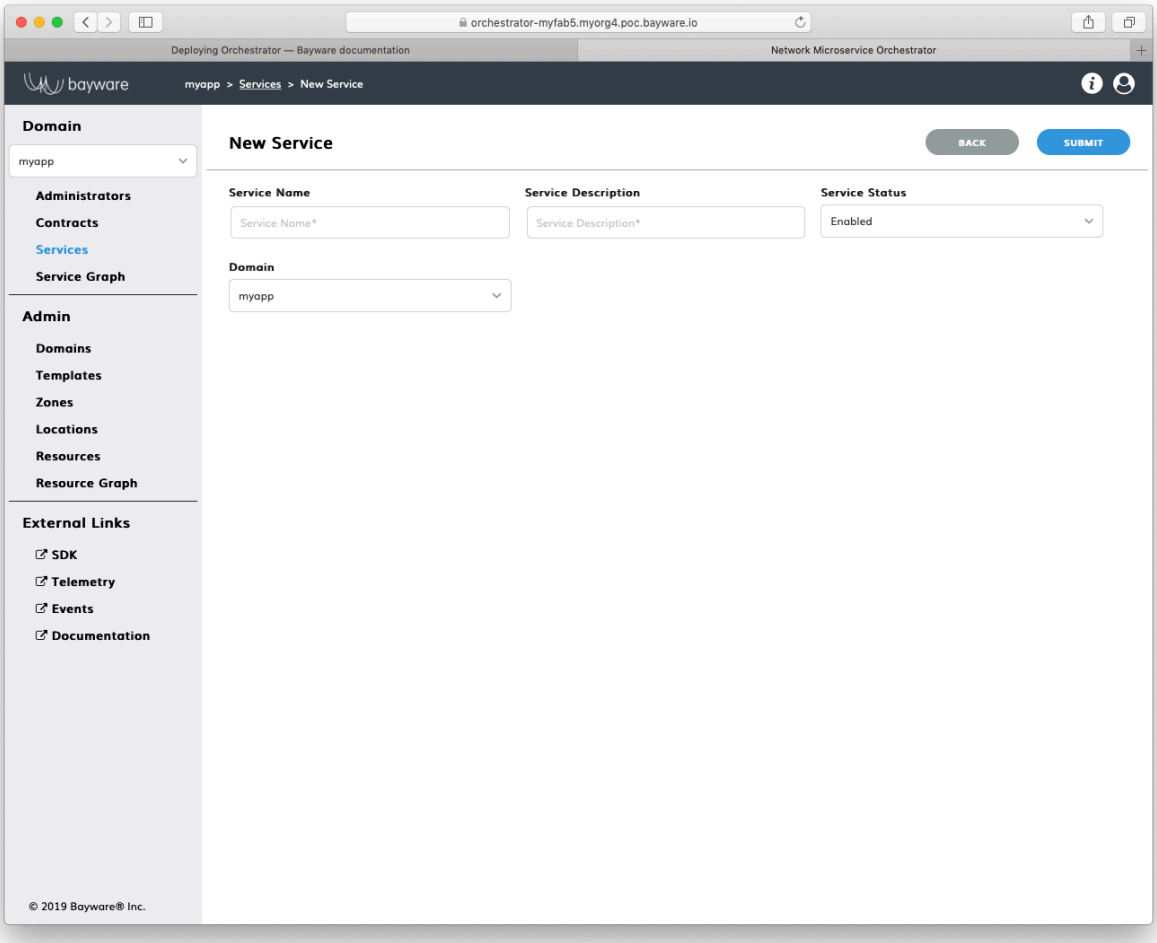


Fig. 27.7: Add New Service

Fill out the fields on the `New Service` page:

service name desired service name;

service description add description for service;

service status in which status service to be created– `Enabled` or `Disabled`;

domain select domain for service.

Submit the new service configuration. You should see the service appear in the list on the `myApp > Services` page.

Using BWCTL-API

To specify a new application service, run this command with a desired service name (any string without spaces) preceding the domain name–in this example `http-proxy@myapp` –as an argument:

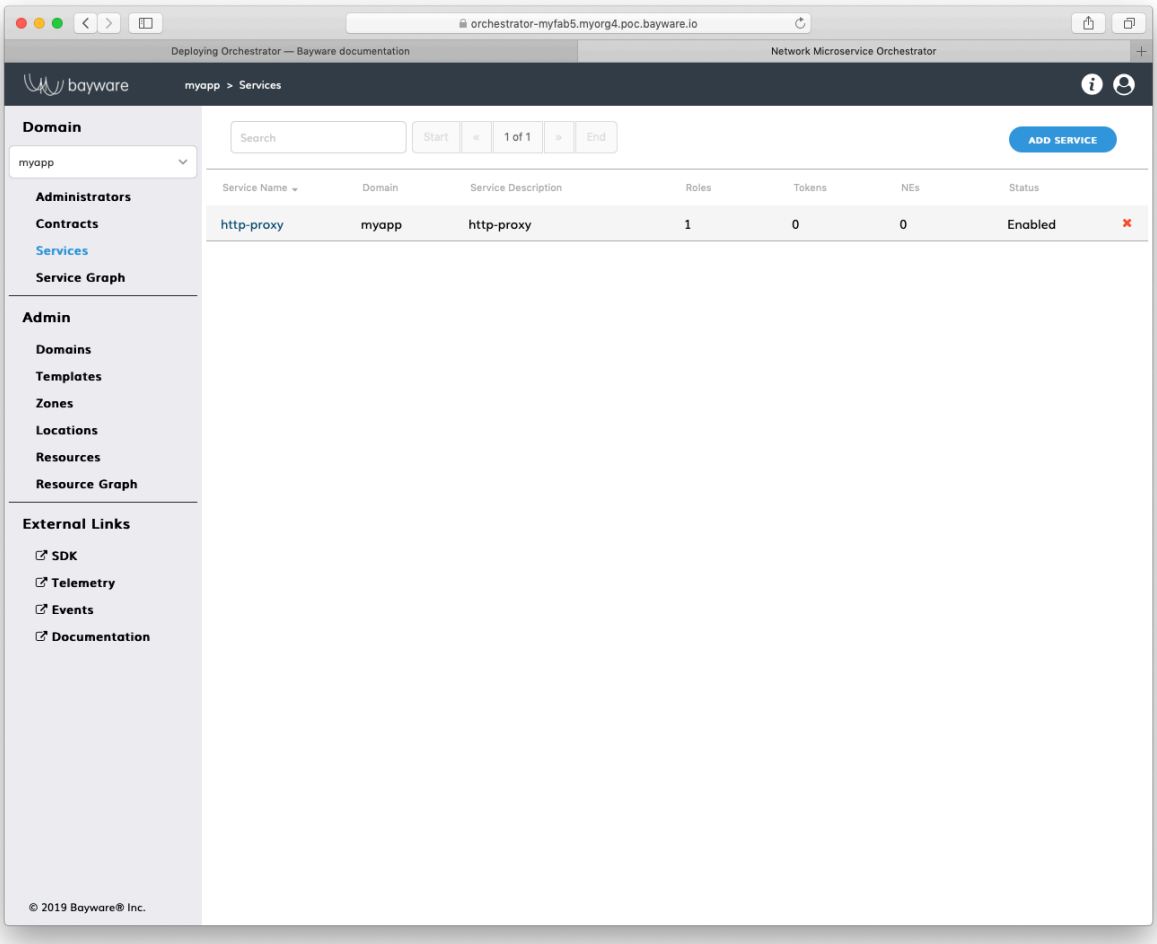


Fig. 27.8: List of Services

```
]$ bwctl-api create service http-proxy@myapp
```

You should see this output:

```
[2019-10-19 00:37:19.873]  Service 'http-proxy@myapp' created successfully
```

Note: When options are not specified on the command line, BWCTL-API applies default configuration settings. See BWCTL-API CLI Manual for specific details.

To check the service configuration, run this command with the service@domain–in this example `http-proxy@myapp` –as an argument:

```
]$ bwctl-api show service http-proxy@myapp
```

You should see a new service specification:

```
---
apiVersion: policy.bayware.io/v1
kind: Service
metadata:
  description: http-proxy
  domain: myapp
  name: http-proxy
spec:
  contract_roles: []
  enabled: true
```

27.2.4 Authorize Service

Using Web-interface

To authorize an application service to access the security segment, lick on the service name in the `myApp > Services` section–in this example `http-proxy`. Now, click Add Role on the `myApp > Services > http-proxy` page.

Fill out the fields in the `Add Contract Role` pop-up window:

contract select contract for an application service;

contract role select contract role for an application service.

Submit the new role configuration. You should see the role appear in the list of Roles on the `myApp > Services > http-proxy` page.

Using BWCTL-API

To authorize an application service to access the security segment, you have to assign the service a role in the contract.

To check available roles, run this command with the contract name–in this example `frontend@myapp` –as an argument:

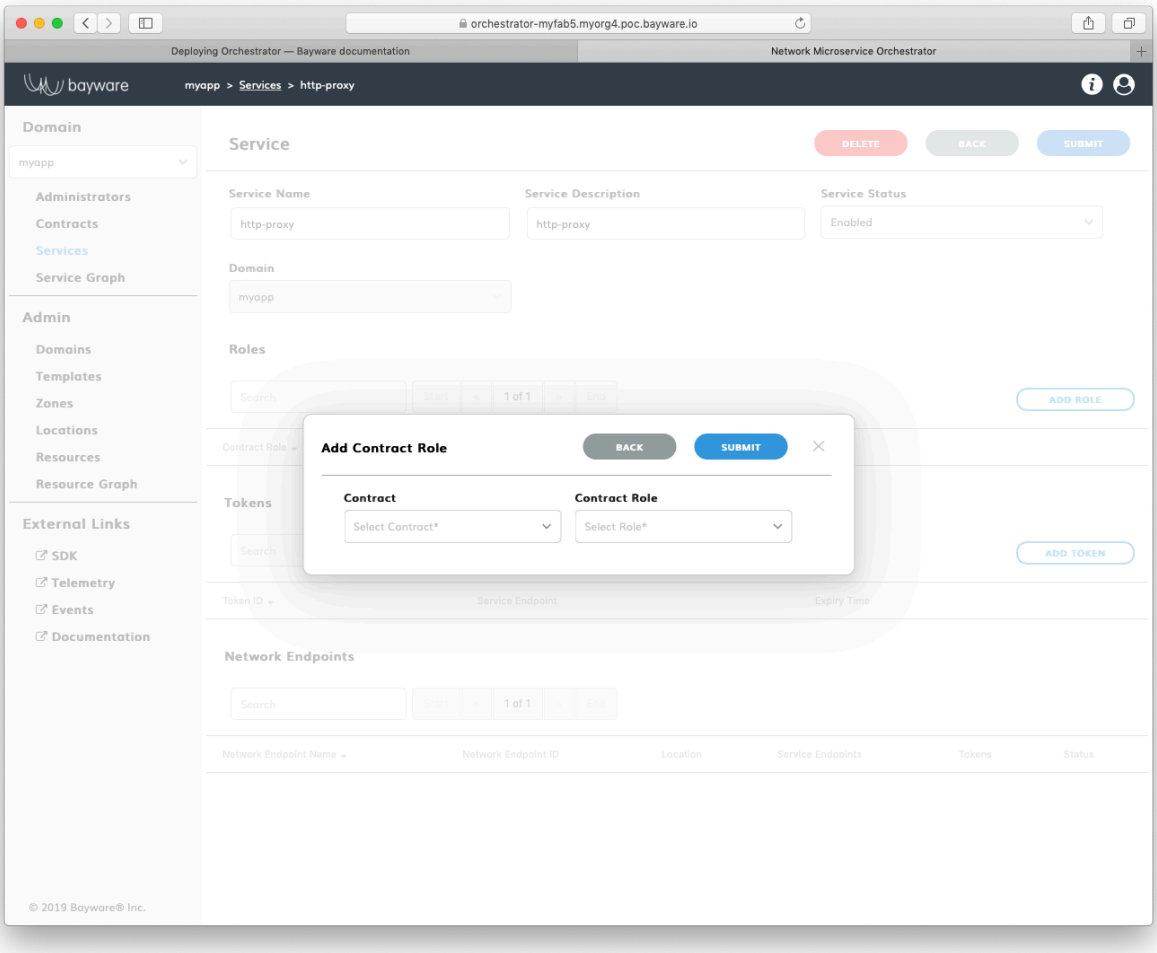


Fig. 27.9: Add New Role

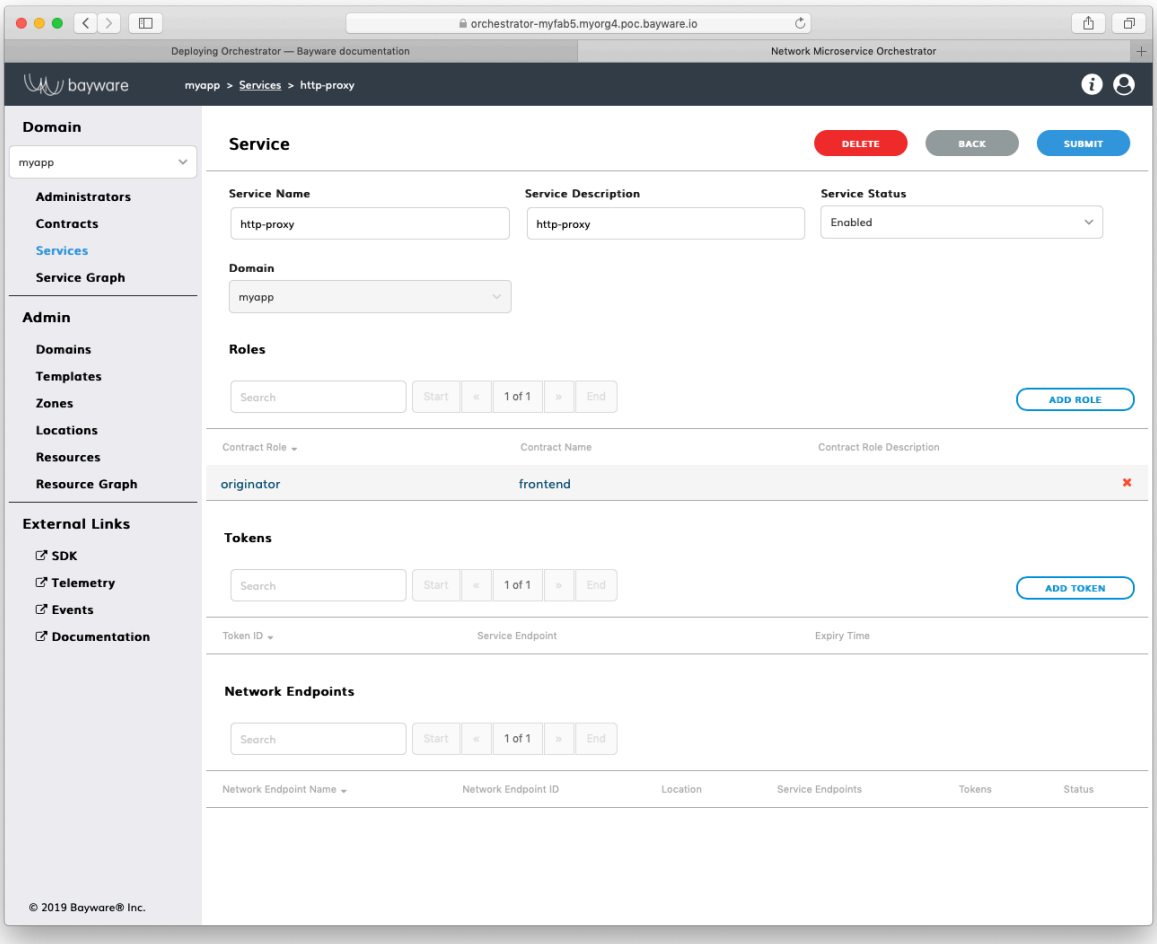


Fig. 27.10: List of Roles

```
]$ bwctl-api show contract frontend@myapp
```

You should see output similar to this:

```
---
apiVersion: policy.bayware.io/v1
kind: Contract
metadata:
  description: frontend
  domain: myapp
  name: frontend
spec:
  contract_roles:
  - cfg_hash: c40f2ddc0843e983a4ea4088e2ea0f8e
    description: null
    id: 1
    ingress_rules:
    - {}
    name: originator
    path_params: {}
    port_mirror_enabled: false
    program_data:
      params:
      - name: hopsCount
        value: 0
      ppl: 0
    propagation_interval: 5
    role_index: 0
    service_rdn: originator.frontend.myapp
    stat_enabled: false
  - cfg_hash: 84dcec61d02bb315a50354e38b1e6a0a
    description: null
    id: 2
    ingress_rules:
    - {}
    name: responder
    path_params: {}
    port_mirror_enabled: false
    program_data:
      params:
      - name: hopsCount
        value: 0
      ppl: 0
    propagation_interval: 5
    role_index: 1
    service_rdn: responder.frontend.myapp
    stat_enabled: false
  enabled: true
  template: default
```

Note: The contract specification always includes two roles. A unique role identifier is built using this notation – **<role_name>:<contract_name>**.

To assign a contract role to the service, run this command with the service name and the contract role–in this example `originator:frontend` –as an argument:

```
]$ bwctl-api update service http-proxy@myapp -a originator:frontend
```

You should see output similar to this:

```
[2019-10-19 00:38:36.246]  Service 'http-proxy@myapp' updated successfully
```

27.3 Working with Batches

To set up an application policy, you can also use batch files.

Create a new application policy file in your favorite editor, e.g. `nano`:

```
]$ nano new-app-policy.yml
```

Add template, domain, contract and service specifications to the file.

After editing, your file should have content similar to:

```
---
apiVersion: policy.bayware.io/v1
kind: Batch
metadata:
  name: New App Policy
spec:
- kind: Template
  metadata:
    name: default
  spec:
    is_multicast: false
    orientation: directed
    roles:
    - name: originator
      code_binary:␣
↪409C470100E7846300E000EF0A700793C11C004000EF409C470500E7846300C000EF579DC11C004000EF409C00178713C0989
      propagation_interval_default: 5
      program_data_default:
        ppl: 0
        params:
        - name: hopsCount
          value: 0
      code_map:
        originator: 0
      path_binary: 000000000001
    - name: responder
      code_binary:␣
↪409C470100E7846300E000EF0A700793C11C004000EF409C470500E7846300C000EF579DC11C004000EF409C00178713C0989
      propagation_interval_default: 5
      program_data_default:
        ppl: 0
        params:
```

(continues on next page)

(continued from previous page)

```
        - name: hopsCount
          value: 0
      code_map:
        responder: 0
      path_binary: 000000000001
- kind: Domain
  metadata:
    domain: myapp
  spec:
   auth_method:
    - LocalAuth
    domain_type: Application
- kind: Contract
  metadata:
    domain: myapp
    name: frontend
  spec:
    template: default
    contract_roles:
    - template_role: originator
    - template_role: responder
- kind: Service
  metadata:
    name: http-proxy
    domain: myapp
  spec:
    contract_roles:
    - contract: frontend
      contract_role: originator
```

Now, run the policy deployment using the batch file name–in this example `new-app-policy.yml` –as an argument:

```
]$ bwctl-api create batch new-app-policy.yml
```

You should see output similar to:

```
[2019-10-19 23:36:15.317]  Template 'default' created successfully
[2019-10-19 23:36:15.376]  Domain 'myapp' created successfully
[2019-10-19 23:36:15.840]  Contract 'frontend@myapp' created successfully
[2019-10-19 23:36:16.201]  Service 'http-proxy@myapp' created successfully
```

To verify that your application policy is now in place, go to orchestrator, select your application domain–in this example `myapp` –and click `Service Graph`.

Note: At this point, you can start deploying application services in the fabric. See the next section for service authorization and deployment details.

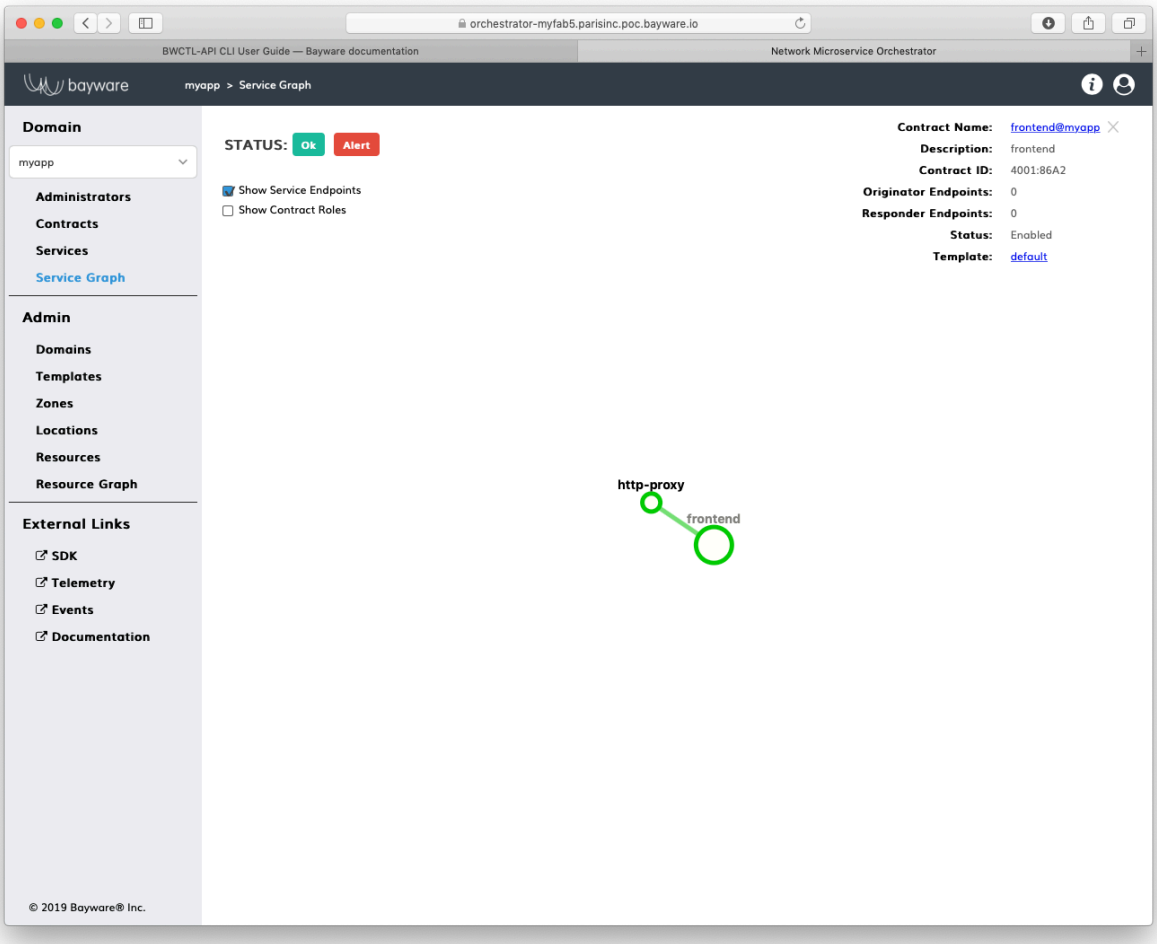


Fig. 27.11: Application Service Graph

CHAPTER 28

Application Deployment

This document describes the management functions necessary for deploying of application services with the BWCTL-API command-line tool or via a web interface.

To deploy application services, all you need to do is generate one or multiple tokens for each application service and place these tokens on workload nodes. The token will automatically enable secure communication for the application service hosted by the node, in strict accordance with the service roles (see the `Create Service Graph > Authorize Service` section for details).

The steps below will guide you through the token generation and deployment process.

28.1 Generate Token

An application service requires at least one token to communicate with other services. You can generate a number of tokens to provide each service instance with its own token.

28.1.1 Using Web Interface

To generate a new token for an application service, click `Add Token` on the service page in the `myApp > Services` section–in this example `http-proxy`.

Fill out the fields in the `Token` pop-up window:

token expire period in days token validity duration.

Note: Starting with family version 1.4, the token scope can be restricted to a given workload or workload location.

Submit the new token configuration. You should see the token identifier and value appear in the next pop-up window.

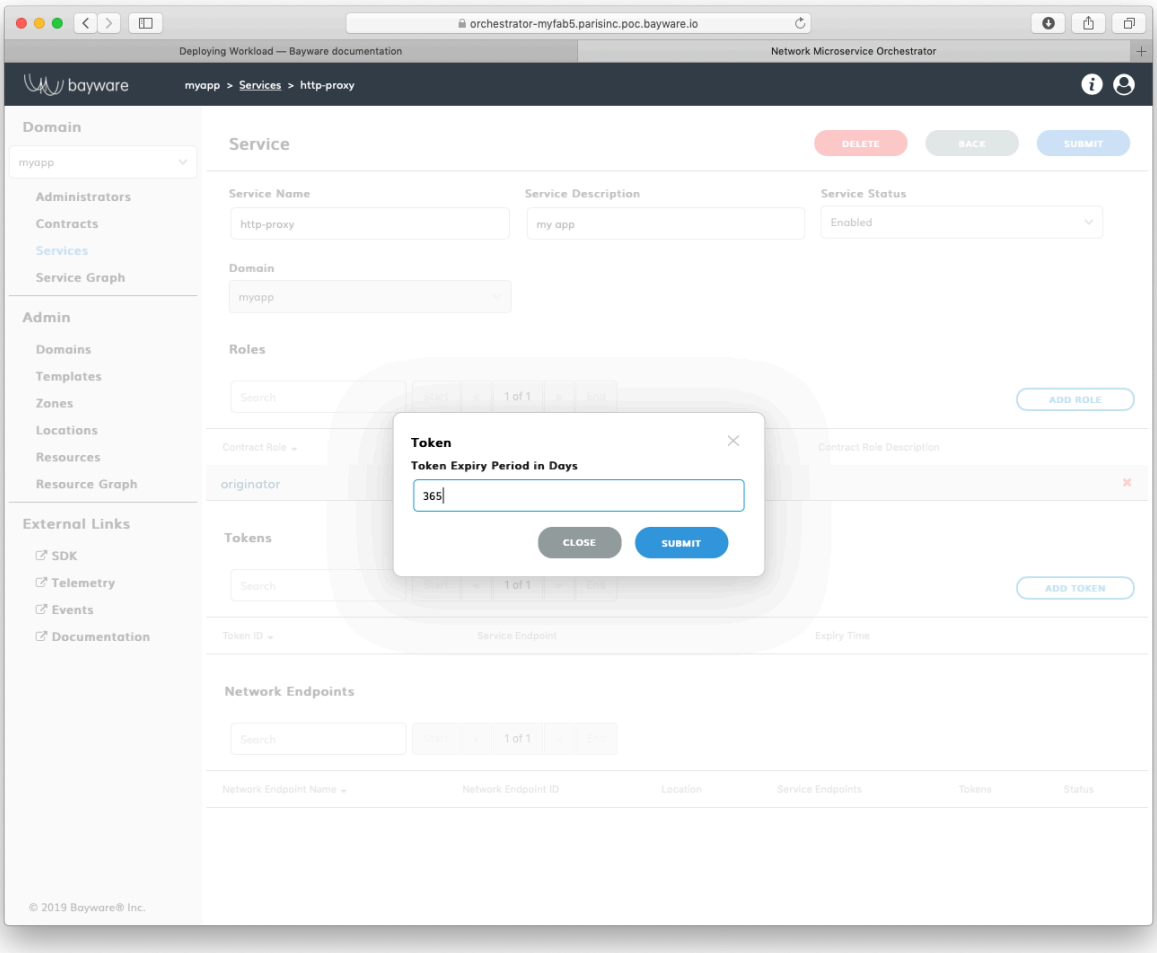


Fig. 28.1: Add New Token

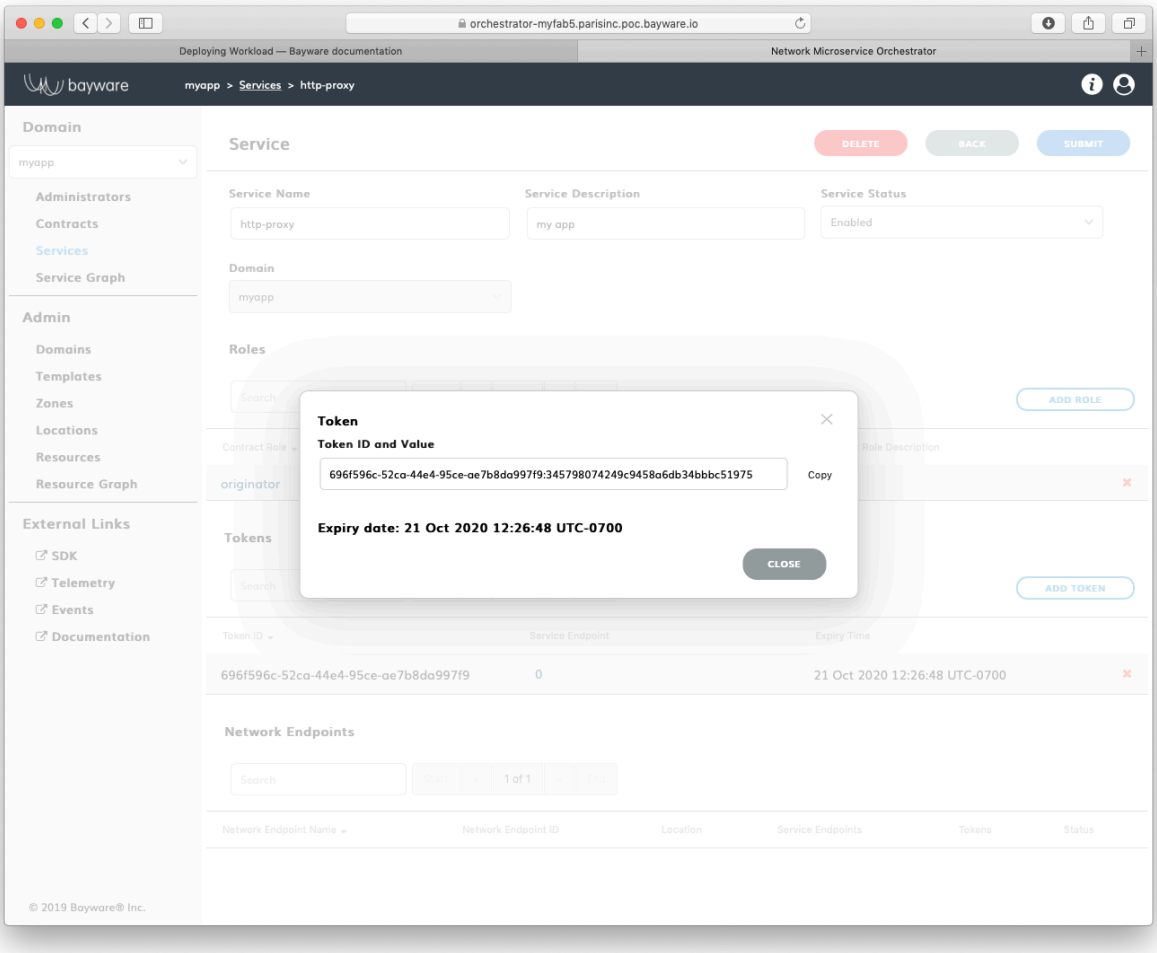


Fig. 28.2: Token ID and Value

Warning: Token comprises two parts–token identity and token secret–separated by a colon. This is the only time you can see the token secret. Be sure to copy the entire **TOKEN** as it appears on your screen, it will be needed later.

After you copied your new token, close the pop-up window. You should see the token identifier and expiry time appear in the `Tokens` list on the application service page–in this example `http-proxy@myApp`.

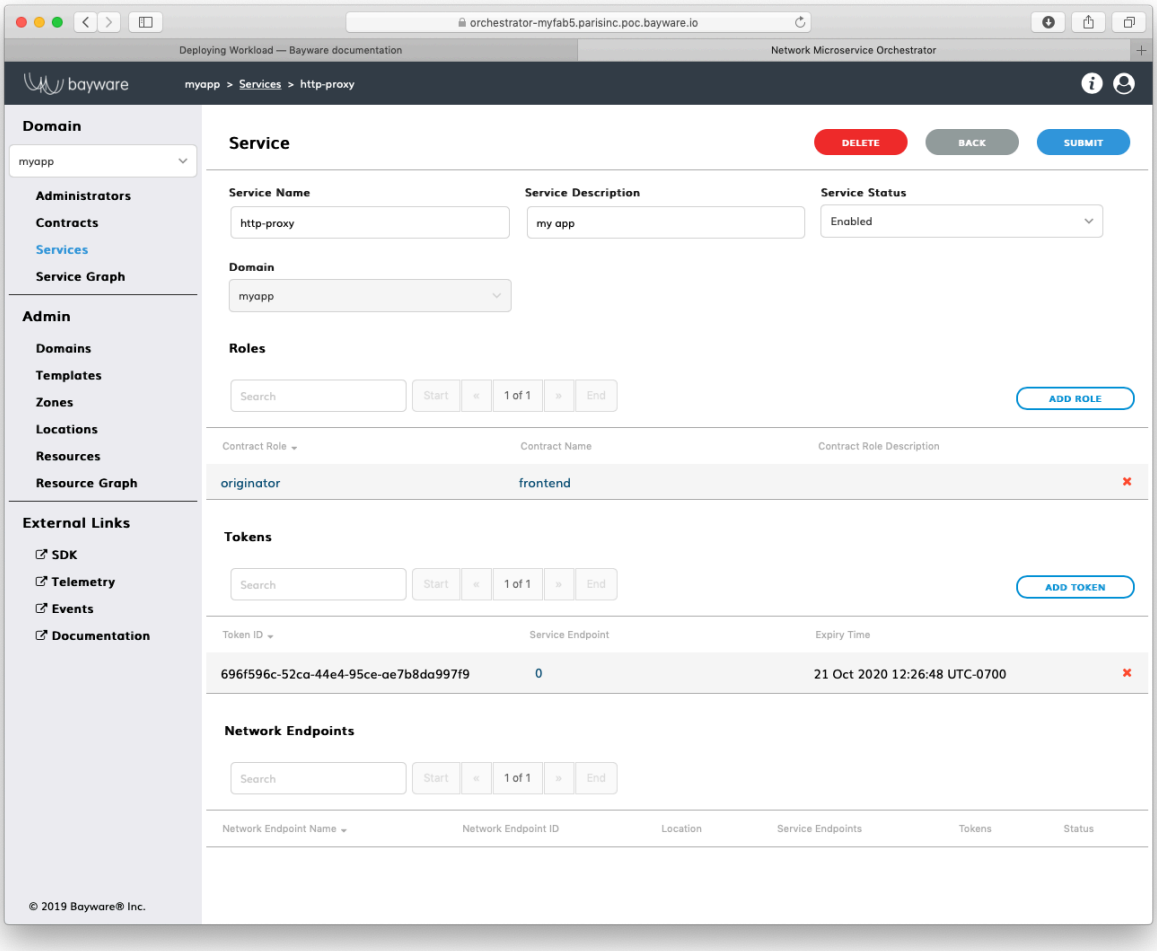


Fig. 28.3: List of Tokens

28.1.2 Using BWCTL-API

To generate a new authorization token for your application service, run this command using `service_name@contract_name` –in this example `http-proxy@myapp` –as an argument:

```
]$ bwctl-api create service_token http-proxy@myapp
```

You should see output similar to this:

```
---
apiVersion: policy.bayware.io/v1
kind: ServiceToken
metadata:
  token_ident: 76686212-f14e-4919-aabc-bcd6b09e28dc:ee0760eb054ffd95e290e6ef2bbf7739
spec:
  domain: myapp
  expiry_time: 22 Oct 2020 19:40:49 GMT
  service: http-proxy
  status: Active
```

Warning: Again, the Token comprises two parts–token identity and token secret–separated by a colon. This is the only time you can see the token secret. Be sure to copy the entire **TOKEN** as it appears on your screen, it will be needed later.

28.2 Deploy Service

To deploy an application service on a workload node, you need to provide the node policy agent with a service authorization token.

Note: To discover a remote service, your application service must use the remote service FQDN from the contract.

The steps below will guide you how to pass a token to the policy agent and set up a remote service FQDN for the application service.

28.2.1 SSH to Workload Node

To start deploying a service on a workload, first ssh to the workload from your fabric manager using the workload name–in this example `azr2-w01-myfab5`:

```
]$ ssh azr2-w01-myfab5
```

Note: The fabric manager SSH service is set up automatically for easy and secure access to any node in your service interconnection fabric.

When you are on the workload node, switch to root level access:

```
[ubuntu@azr2-w01-myfab5]$ sudo su -
```

28.2.2 Add Token

Next, edit the policy agent token file by running this command:

```
[ubuntu@azr2-w01-myfab5]# nano /opt/bayware/ib-agent/conf/tokens.dat
```

Add the token to the `tokens.dat` file and save the file, which in this example will contain after editing:

```
76686212-f14e-4919-aabc-bcd6b09e28dc:ee0760eb054ffd95e290e6ef2bbf7739
```

To apply the token, reload the policy agent by running this command:

```
[ubuntu@azr2-w01-myfab5]# systemctl reload ib-agent
```

At this point, you can visit the policy orchestrator and find a registered endpoint on the `Service Graph` page of your application.

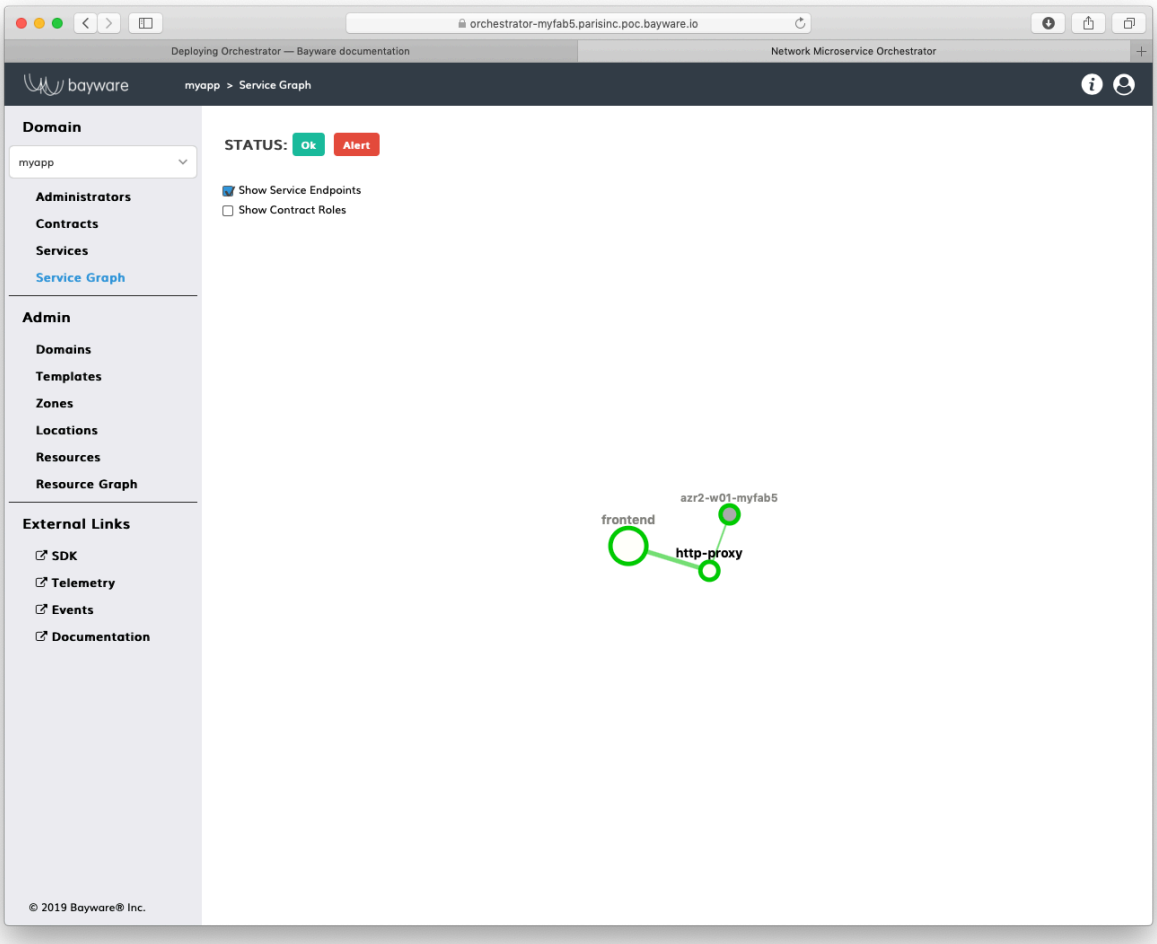


Fig. 28.4: Service Graph with Registered Service Endpoint

28.2.3 Install Service

Now, you can install your application service on the workload node. In this example, a package called `getaway-proxy` installs by running this command:

```
[ubuntu@azr2-w01-myfab5]# apt-get install getaway-proxy
```

The service automatically discovers all remote services sharing the same contract. To enable it, update the application service configuration file with the remote service FQDN by running this command:

```
[ubuntu@azr2-w01-myfab5]# nano /opt/getaway-proxy/conf/app.conf
```

After editing, the service configuration file in this example contains:

```
WS_APP = 'http://responder.frontend.myapp.ib.loc:8080/'
```

Note: The remote service FQDN is automatically built using this notation: **<role>.<contract>.<domain>.<hosted_zone>**, wherein the default hosted zone is `ib.loc`

Note: You can always change the FQDN parts: `hosted_zone` in the policy agent configuration and `role.contract.domain` in contract settings.

To start the application service, run this command:

```
[ubuntu@azr2-w01-myfab5]# systemctl start getaway-proxy
```

Note: As soon as an instance of the remote service `responder.frontend.myapp`. is deployed in the fabric, it will be automatically discovered, and `getaway-proxy` will be able to reach it.

28.3 Working with Batches

You can generate tokens in batches and use the batch command output in your application CI/CD pipeline to deploy a number of new services or their instances or to rotate tokens.

Create a token request file on the fabric manager node–for example `myapp-tokens.yml`:

```
]$ nano myapp-tokens.yml
```

After editing, this example of the batch file contains:

```
---
apiVersion: policy.bayware.io/v1
kind: Batch
metadata:
  name: batch file for generating 6 tokens
spec:
- kind: Token
  metadata:
      name: aws-proxy
  spec:
      service: http-proxy
      domain: myapp
- kind: Token
  metadata:
      name: aws-svc
```

(continues on next page)

(continued from previous page)

```
  spec:
      service: getaway-svc
      domain: myapp
- kind: Token
  metadata:
      name: gcp-news
  spec:
      service: news-gw
      domain: myapp
- kind: Token
  metadata:
      name: gcp-places
  spec:
      service: places-gw
      domain: myapp
- kind: Token
  metadata:
      name: gcp-weather
  spec:
      service: weather-gw
      domain: myapp
- kind: Token
  metadata:
      name: azr-weather
  spec:
      service: weather-gw
      domain: myapp
```

Notice that the batch requests multiple tokens for the five services. One of the services– `weather-gw` –runs both in GCP and in Azure. So the request contains two tokens for each instance of `weather-gw`, totaling in six tokens for the five services.

Note: By creating a unique token for each running service instance, you can control service authorization independently in each cloud or VPC. This will come in handy later on when you need to rotate or revoke service authorization in one cloud independently from others.

Now, execute the command to get the tokens from the orchestrator. Here you will redirect Linux stdout to a file where the tokens will be saved.

```
]$ bwctl-api create batch myapp-tokens.yml > tokens.yml
```

Once it has completed, use **cat** to explore the file containing the service tokens returned by the orchestrator.

```
]$ cat tokens.yml
```

The orchestrator has returned a YAML sequence of six tokens, each associated with a particular service within a given domain in a given cloud, as prescribed by the requesting YAML file.

```
---
- domain: myapp
  expiry_time: 23 Oct 2020 22:38:04 GMT
```

(continues on next page)

(continued from previous page)

```
  name: aws-proxy
  service: http-proxy
  status: Active
  token: 84d93958-5ce3-4a03-97f8-b2783d62a3ca:83f2fcf74684d9bd88c952c4854ba9bf

- domain: myapp
  expiry_time: 23 Oct 2020 22:38:04 GMT
  name: aws-svc
  service: getaway-svc
  status: Active
  token: 4e29b3f0-7592-423f-bbcd-0a745af36cf7:e34122af6ab00c26e839796211b0f249

- domain: myapp
  expiry_time: 23 Oct 2020 22:38:04 GMT
  name: azr-news
  service: news-gw
  status: Active
  token: d5a4d58a-fddc-4e49-b48c-233a8dbc26bc:bf55693b5edf55061bf8637e7d65c3d2

- domain: myapp
  expiry_time: 23 Oct 2020 22:38:04 GMT
  name: azr-places
  service: places-gw
  status: Active
  token: 1ef1b997-144e-49a8-9985-4d04d06953e1:892864e7531c1cd0bcc8cf78232a0851

- domain: myapp
  expiry_time: 23 Oct 2020 22:38:05 GMT
  name: azr-weather
  service: weather-gw
  status: Active
  token: 204e36f4-4cdb-4d7a-b69b-529723d446d6:74427571e07536e43c6e59a650538675

- domain: myapp
  expiry_time: 23 Oct 2020 22:38:05 GMT
  name: gcp-weather
  service: weather-gw
  status: Active
  token: 720f45bf-b4d6-4664-b844-9f9a6fc681ac:90646deaeaf93223e1963c5445fb6c1a
```

Now, you can use this file in your application deployment pipeline to create service endpoints that get your application services up and running and communicating with each other.

CHAPTER 29

BWCTL-API Command Line Interface

29.1 About BWCTL-API

BWCTL-API is a command line interface (CLI) tool that enables you to interact with the SIF Orchestrator using commands in your command-line shell. The tool offers all the functionality provided by the Orchestrator Graphical User Interface (GUI) as they both utilize the same Orchestrator's RESTful Northbound Interface (NBI).

In general, you can manage all policy entities in your service interconnection fabric using interchangeably one of the three tools:

- Browser-based GUI,
- BWCTL-API CLI,
- RESTful NBI.

To use BWCTL-API tool, you can install it on your local Linux machine or access remotely the tool already installed on your fabric manager node from any Linux, macOS, or Windows machine.

When the tool installed locally, use a common shell program, *e.g. bash*, to run BWCTL-API commands. To run the commands remotely, you will need a terminal window with an SSH client:

- MacOS – Use Terminal application with built-in SSH client.
- Linux – Use your favorite terminal window with built-in SSH client.
- Windows 10 – If you haven't already enabled an SSH client to use with PowerShell, PuTTY is an easy alternative. PuTTY can act as both your terminal window and your SSH client.

BWCTL-API enables you to monitor policy entities and configure them. You can `show`, `create`, `update`, `enable`, `disable`, and `delete` policy entities of the service interconnection fabric: `domains`, `administrators`, `contracts`, `templates`, `services`, `service-tokens`, `resources`, `zones`, `locations`, `links`, `label-class-link`, `label-class-node`. Also, the tool allows you to perform the same operation on a batch of different policy entities.

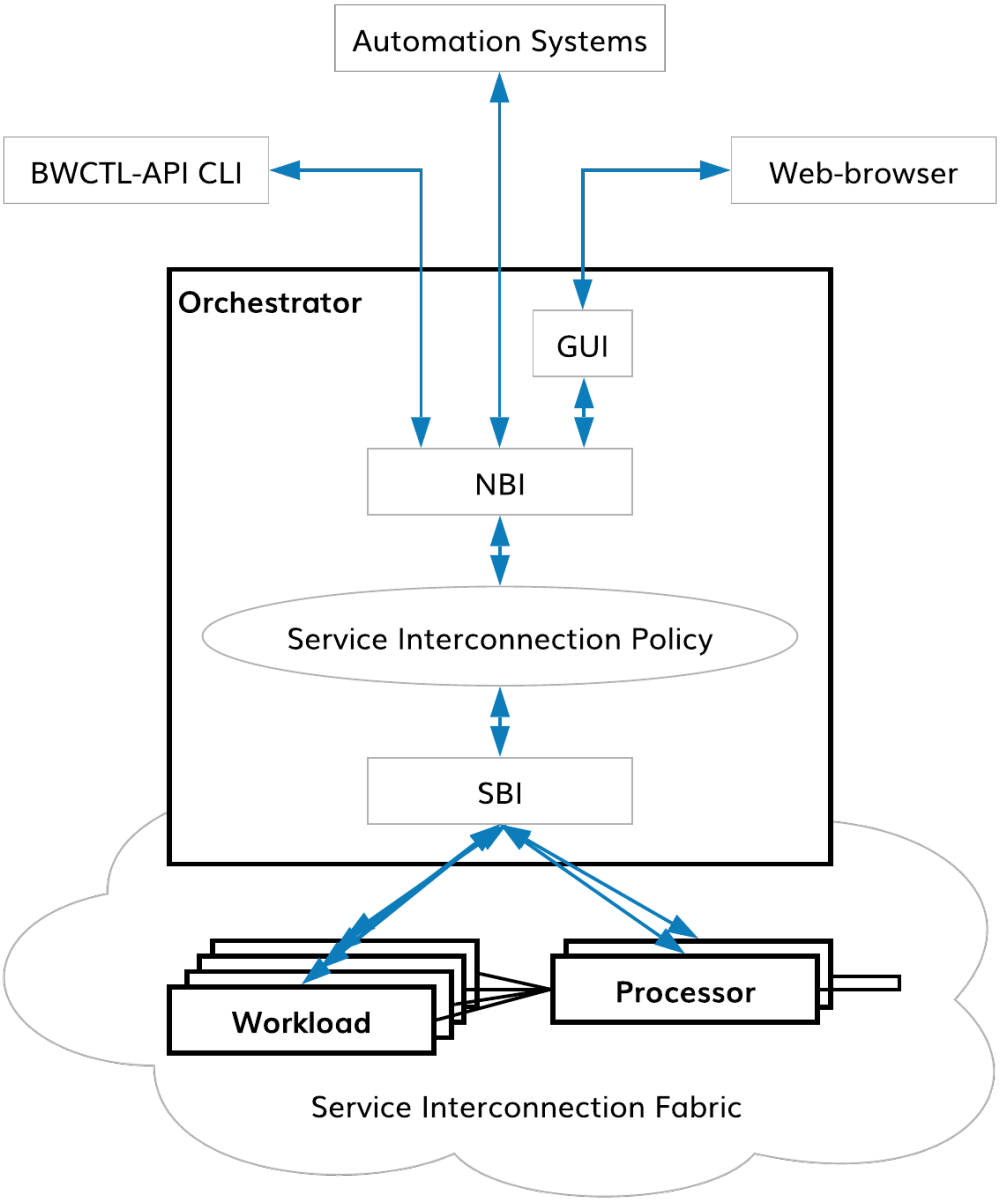


Fig. 29.1: BWCTL-API CLI for SIF policy management

29.2 Installing BWCTL-API

29.2.1 Ways to install

BWCTL-API tool comes already preinstalled on the fabric manager image available in AWS, Azure and GCP clouds.

To install BWCTL-API on your own Linux machine, you have first to install the Bayware repository on the machine and after that you can deploy the most recent version of the tool from the repository.

To add the repo, run the command (the command shown assumes you install the repo on a Debian machine):

```
]$ add-apt-repository 'deb https://s3-us-west-1.amazonaws.com/bayware-repo-devel/public/
↪<specify_family_here>/ubuntu bionic main'
```

Note: BWCTL-API version must be from the same family as the other Bayware components in your service interconnection fabric, so use the right family version when installing repo. If BWCTL-API version is incompatible with the orchestrator, the tool will fail to establish a connection to orchestrator with an error message specifying the required family. The family is specified in the form of platform version and might look like `2.x` or `3.x`.

29.2.2 Installing BWCTL-API with apt on Debian/Ubuntu

Installing BWCTL-API on Debian/Ubuntu with apt provides isolation for the tool and its dependencies. Also, it's easy to upgrade when a new version of BWCTL-API tool is released.

First, switch to root level access to install all packages as such:

```
]$ sudo su -
```

To install BWCTL-API on the machine with the Bayware repository already installed, run the command:

```
]# apt-get install bwctl-api
```

Verify that BWCTL-API installed correctly by running the command:

```
]# bwctl-api --version
bwctl-api/1.3.0
```

29.2.3 Upgrading BWCTL-API to the latest version

You can upgrade BWCTL-API tool already installed on your machine to the latest version in the family by running the command:

```
]# apt-get update
]# apt-get --only-upgrade install bwctl-api
```

29.2.4 Uninstalling BWCTL-API

If you need to uninstall BWCTL-API tool, run the command:

```
]# apt-get --purge remove bwctl-api
```

To exit from the current command prompt once you have completed installing, updating, or deleting BWCTL_API, run the command:

```
]# exit
```

29.3 Configuring BWCTL-API

29.3.1 Configuring BWCTL-API after installation

Before you can run BWCTL-API, you must configure the tool with your orchestrator credentials. You store configuration locally in the file called `config.yaml` located at:

```
~/.bwctl-api/config.yml
```

The file contains BWCTL-API credential details. To verify information in the configuration file, run the commands:

```
]$ cd .bwctl-api
~/.bwctl-api$ more config.yml
---
hostname: orchestrator-fab1.example.com
Domain: EXAMPLEDOMAIN
login: EXAMPLELOGIN
password: EXAMPLEPASSWORD
```

The `hostname` is an FQDN of the fabric orchestrator which you access with BWCTL-API tool. The `domain`, `login`, and `password` are your credentials at the orchestrator that determine what permissions you have for managing the service interconnection fabric.

To run BWCTL-API commands you must have an account on the orchestrator with one of the two administrative roles: `systemAdmin` or `domainAdmin`. See how to create an administrator account in the documentation on orchestrator GUI.

Note: If you are configuring BWCTL-API on the fabric manager node, the credentials can be automatically retrieved when the orchestrator's controller node created. For more information, see BWCTL CLI documentation.

29.3.2 Changing BWCTL-API configuration

If you need to change BWCTL-API configuration, update its configuration file stored locally at `~/.bwctl-api/config.yml`.

29.4 Getting started with BWCTL-API

29.4.1 Typing the first command

To give a command in BWCTL-API, you will type `bwctl-api` along with the required input and press the `<return>` key.

To start using BWCTL-API tool, run the command:

```
]$ bwctl-api
Usage: bwctl-api [OPTIONS] COMMAND [ARGS]...

  Bayware CLI (Policy management)

Options:
  -v, --version  Print version and exit.
  -h, --help     Show this message and exit.

Commands:
  create   Create policy entity
  delete   Delete policy entity
  disable  Disable policy entity
  enable   Enable policy entity
  show     Show policy entity
  update   Update policy entity
```

The output above is the same as from running the command:

```
$ bwctl-api --help
```

29.4.2 Command Structure

The command line is comprised of several components:

- bwctl-api,
- any options required by bwctl-api to execute the command,
- the command and, in most cases, subcommand,
- any arguments required by the command.

```
]$ bwctl-api --help
Usage: bwctl-api [OPTIONS] COMMAND [ARGS]...
```

29.4.3 Command Line Options

You can use the following command line options typing them on the command line immediately after `bwctl-api`:

—**version, -v** A boolean switch that displays the current version of BWCTL-API tool.

—**help, -h** A boolean switch that displays the commands available for execution.

You can finish the command line with the `--help` option following either command or subcommand. The output will always give you a hint about what else you need to type.

To see the help for the command, type the command only followed by `--help` and press `<return>`:

```
]$ bwctl-api show --help
Usage: bwctl-api show COMMAND [ARGS] [OPTIONS]

  To show policy entity, enter <command>.

Options:
  -h, --help  Show this message and exit.

Commands:
  administrator      Show administrator
  contract           Show contract
  domain             Show domain
  label-class-link   Show label class link
  label-class-node   Show label class node
  link               Show links
  location           Show location
  resource           Show resource
  service            Show service
  service-token      Show service token
  template           Show template
  zone               Show zone
```

To see the help for the subcommand, type the command followed by the subcommand and the `--help` and press `<return>`:

```
]$ bwctl-api show contract --help
Usage: bwctl-api show contract [OPTIONS] [CONTRACT@DOMAIN]

  To show contract, enter <contract>@<domain>. To show all contracts within
  a domain, use --domain <domain>.

Options:
  -d, --domain              Domain name.
  -c, --config-file         Path to configuration file.
  -o, --output-format       Output format: json or yaml.
  -h, --help                Show this message and exit.
```

Different commands support different options. Detail information on options find in the documentation section *Using commands*.

29.4.4 Commands

With BWCTL-API you can manage all policy entities in your service interconnection fabric. Each command includes the entity kind, as subcommand, and entity name, as argument. Some commands have the entity specification file as a mandatory argument.

BWCTL-API supports the following commands:

create KIND NAME [OPTIONS]

The command creates one or multiple entities. The specification file is mandatory for this command.

delete KIND NAME [OPTIONS]

The command deletes one or multiple entities. The specification file is mandatory for the `batch` kind.

disable KIND NAME

The command disables a single entity.

enable KIND NAME

The command enables a single entity.

show KIND NAME [OPTIONS]

The command shows one or multiple entities. For some entity types, the entity name is optional in this command.

update KIND NAME [OPTIONS]

The command updates one or multiple entities. The specification file is mandatory for this command.

29.4.5 Kinds

The diagram below depicts the policy entities and relationships between them.

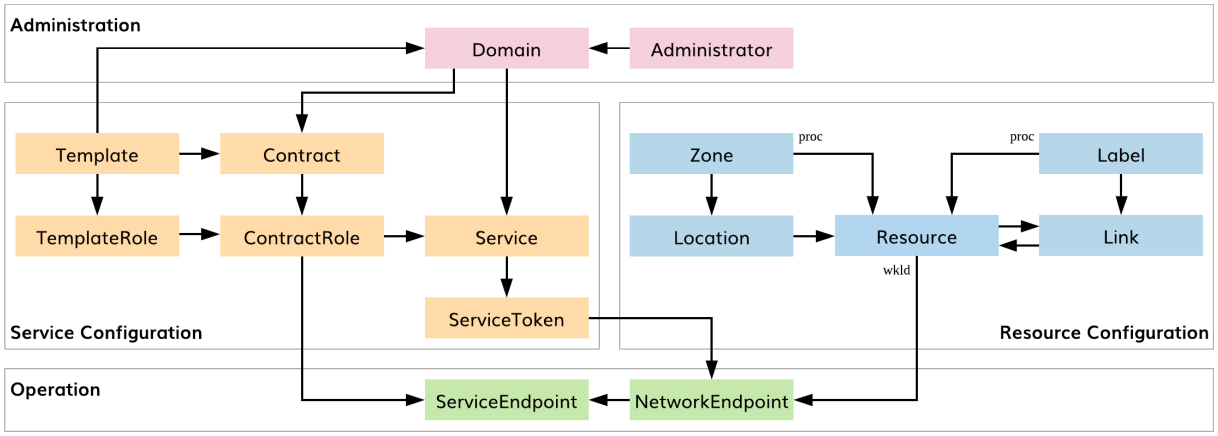


Fig. 29.2: Policy entity

To see the entity types you can run any command without subcommand:

```
]$ bwctl-api show
Usage: bwctl-api show [OPTIONS] COMMAND [ARGS]...

  Show policy entity commands

Options:
  -h, --help  Show this message and exit.
```

(continues on next page)

(continued from previous page)

```
Commands:
  administrator     Show administrator
  contract          Show contract
  domain            Show domain
  label-class-link  Show label class link
  label-class-node  Show label class node
  link              Show link
  location          Show location
  resource          Show resource
  service           Show service
  service-token     Show service token
  template          Show template
  zone              Show zone
```

BWCTL-API manages the following entity types:

administrator NAME@DOMAIN

The `administrator` entity is an account of the service interconnection fabric administrator.

contract NAME@DOMAIN

The `contract` entity represents a communication microsegment for application services.

domain NAME

The `domain` entity serves as an administrative boundary between different portions of the service interconnection fabric.

link NAME

The `link` entity represents a connection between the service interconnection fabric resources: workload node and processor node or between two processor nodes.

location NAME

The `location` entity is an abstraction of the site where the workload nodes are deployed, *e.g. cloud VPC or private datacenter.*

resource NAME

The `resource` entity represents compute and network resources in the service interconnection fabric: a workload node with policy agent or a processor node with policy engine.

service NAME@DOMAIN

A set of applications, an individual application or an application microservice is represented in the policy model as the `service` entity.

service-token NAME@DOMAIN

The `service-token` entity is a service credential that defines the service access permissions to the communication microsegments.

template NAME

The `template` entity represents a predefined set of communication rules that can be used in contracts.

zone NAME

The `zone` entity is a service zone for processors, which bounds the processors to the workload nodes in one or multiple locations.

29.4.6 Batch

With BWCTL-API CLI, you can use a single `batch` command to manage a set of entities of the same or different types. Below is an example of the command.

```
]$ bwctl-api create batch getaway-app.yml
```

29.5 Using Commands

29.5.1 Supported commands for each entity type

There are three groups of entities, each of which has its own set of commands.

show, create, update, delete, enable, disable This set of commands is applicable to the following types of entities:

- ADMINISTRATOR
- CONTRACT
- LINK
- SERVICE
- TEMPLATE

show, create, update, delete This set of commands is applicable to the following types of entities:

- DOMAIN
- LABEL-CLASS-LINK
- LABEL-CLASS-NODE
- LOCATION
- RESOURCE
- ZONE

show, create, delete This set of commands is applicable to the following types of entities:

- SERVICE-TOKEN

create, update, delete This set of commands is applicable to the following types of entities:

- BATCH

29.5.2 Managing Administrators

You can manage administrators using the following commands:

show administrator [OPTIONS] The command shows all administrators. You can use the options in this command as follows: `-d, --domain`

domain name to show administrators within a domain only.

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show administrator NAME@DOMAIN [OPTIONS] The command shows the administrator. You can use the options in this command as follows: `-d, --domain`

domain name to show administrators within a domain only.

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create administrator NAME@DOMAIN [OPTIONS] The command creates the administrator. The specification file is mandatory for this command. `-f, --file`

path to the specification file.

update administrator NAME@DOMAIN [OPTIONS] The command updates the administrator. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`--description`

description.

`-auth`

administrator authentication method, either `local` or `ldap` (if both are allowed in the domain).

`-p, --password`

administrator password.

`--role`

administrator role: systemAdmin or domainAdmin.

`--enabled`

administrator account status: true or false.

`-c, --config-file`

path to configuration file.

delete administrator NAME@DOMAIN `-c, --config-file`

path to configuration file.

enable administrator NAME@DOMAIN `-c, --config-file`

path to configuration file.

disable administrator NAME@DOMAIN `-c, --config-file`

path to configuration file.

An example of the administrator specification file is shown below.

```
]$ cat administrator-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Administrator
metadata:
  username: admin
  user_domain: default
spec:
  is_active: true
  roles:
  - systemAdmin
  user_auth_method: LDAP
```

29.5.3 Managing Contracts

You can manage contracts using the following commands:

show contract [OPTIONS] The command shows all contracts. You can use the options in this command as follows: `-d, --domain`

`-d, --domain`

domain name to show contracts within a domain only.

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show contract NAME@DOMAIN [OPTIONS] The command shows the contract. You can use the options in this command as follows: `-o, --output-format`

`-o, --output-format`

output format for this command, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

`-d, --domain`

domain name to show contracts within a domain only.

create contract NAME@DOMAIN [OPTIONS] The command creates the contract. The specification file is mandatory for this command. `-f, --file`

`-f, --file`

path to the specification file.

`--description`

description.

`--template`

template name.

update contract NAME@DOMAIN [OPTIONS] The command updates the contract. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`--description`

description.

delete contract NAME@DOMAIN

`-c, --config-file`

path to configuration file.

enable contract NAME@DOMAIN

`-c, --config-file`

path to configuration file.

disable contract NAME@DOMAIN

`-c, --config-file`

path to configuration file.

An example of the contract specification file is shown below.

```
]$ cat contract-spec.yml
---
apiVersion: policy.bayware.io/v1
kind: Contract
metadata:
  name: frontend
  domain: getaway-app
spec:
  template: anycast-cross-all-vpcs
  contract_roles:
  - template_role: Originator
  - template_role: Responder
      ingress_rules:
      - protocol: icmp
      - protocol: tcp
        ports:
        - 8080
        - 5201
```

29.5.4 Managing Domains

You can manage domains using the following commands:

show domain [OPTIONS] The command shows all domains. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show domain NAME [OPTIONS] The command shows the domain. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create domain NAME [OPTIONS] The command creates the domain. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`--description`

description.

`--auth`

domain authentication method, `local` or `ldap` (both can be allowed in the domain).

`-c, --config-file`

path to configuration file.

update domain NAME [OPTIONS] The command updates the domain. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`--description`

description.

`--auth`

domain authentication method, `local` or `ldap` (both can be allowed in the domain).

delete domain NAME

`-c, --config-file`

path to configuration file.

An example of the domain specification file is shown below.

```
]$ cat domain-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Domain
metadata:
  domain: getaway-app
spec:
  auth_method:
  - LocalAuth
  domain_type: Application
```

29.5.5 Managing Labels

Two class types of managing labels.

Link Labels

You can manage Link labels using the following commands:

show label-class-link [OPTIONS] The command shows all label links. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

create label-class-link NAME [OPTIONS] The command creates the link label. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`--description`

label class description.

`--class-id <class_identifier>`

label class identifier.

`--label-value-min <min_value>`

minimum value of link label.

`--label-value-max <max_value>`

maximum value of link label.

update label-class-link NAME [OPTIONS] The command creates the label class link. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`--description`

label class description.

`--class-id <class_identifier>`

label class identifier.

`--label-value-min <min_value>`

minimum value of link label.

`--label-value-max <max_value>`

maximum value of link label.

`-a, --append <label_name>`

update an existing link label.

`-d, --delete <label_name>`

delete an existing link label.

`--label-value <value>`

label value.

`--label-description <description>`

label description.

delete label-class-link NAME [OPTIONS]

`-c, --config-file`

path to configuration file.

Node Labels

show label-class-node [OPTIONS] The command shows all label class nodes. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

create label-class-node [OPTIONS] The command creates the label class node.

`-f, --file`

path to specification file.

`--description`

description.

`--class-id`

label class identifier.

`--label-value-min`

minimum value of node.

`--label-value-max`

maximum value of node.

update label-class-node NAME [OPTIONS] The command creates the label class node.

`-f, --file`

path to specification file.

`--description`

description.

`--class-id`

label class identifier.

`--label-value-min`

minimum value of node.

`--label-value-max`

maximum value of node.

`-a, --append`

update an existing node label.

`--label-value`

label value.

`--label-description`

label description.

delete label-class-node NAME [OPTIONS]

`-c, --config-file`

path to configuration file.

29.5.6 Managing Links

You can manage links using the following commands:

show link [OPTIONS] The command shows all links. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show link NAME [OPTIONS] The command shows the link. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create link NAME [OPTIONS] The command creates the link. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

`--enabled`

link administrative status: true or false.

update link NAME [OPTIONS] The command updates the link. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`--description`

description.

`-c, --config-file`

path to configuration file.

`--enabled`

link administrative status: true or false.

delete link NAME

`-c, --config-file`

path to configuration file.

enable link NAME

`-c, --config-file`

path to configuration file.

disable link NAME

`-c, --config-file`

path to configuration file.

An example of the link specification file is shown below.

```
]$ cat link-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Link
metadata:
  name: <autogenerated>
spec:
  source_node: aws2-p01-fab1
  target_node: azr1-p01-fab1
  cost: 1
  admin_status: true
  ipsec_enable: true
  tunnel_ip_type: public
```

29.5.7 Managing Locations

You can manage locations using the following commands:

show location [OPTIONS] The command shows all locations. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show location NAME [OPTIONS] The command shows the location. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create location NAME [OPTIONS] The command creates the location. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

update location NAME [OPTIONS] The command updates the location. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`-desc, --description`

description.

`-c, --config-file`

path to configuration file.

delete location NAME

`-c, --config-file`

path to configuration file.

An example of the location specification file is shown below.

```
]$ cat location-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Location
metadata:
  name: aws2
spec:
```

29.5.8 Managing Resources

You can manage resources using the following commands:

show resource [OPTIONS] The command shows all resources. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show resource NAME [OPTIONS] The command shows the resource. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create resource NAME [OPTIONS] The command creates the resource. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

update resource NAME [OPTIONS] The command updates the resource. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`-desc, --description`

description.

delete resource NAME

`-c, --config-file`

path to configuration file.

An example of the resource specification file is shown below.

```
]$ cat resource-spec.yml
---
kind: Resource
metadata:
  name: aws2-p01-fab1
spec:
  location: aws2
  node_type: processor
```

29.5.9 Managing Services

You can manage services using the following commands:

show service [OPTIONS] The command shows all services. You can use the options in this command as follows:

`-d, --domain`

domain name to show services within a domain only.

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show service NAME@DOMAIN [OPTIONS] The command shows the service. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

`-d, --domain`

domain name to show services within a domain only.

create service NAME@DOMAIN [OPTIONS] The command creates the service. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

`--description`

description.

update service NAME@DOMAIN [OPTIONS] The command updates the service. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`-desc, --description`

description.

`-a, --append`

add contract role: <role_name:contract_name>.

`-d, --delete`

delete contract role: <role_name:contract_name>.

`-c, --config-file`

path to configuration file.

delete service NAME@DOMAIN

`-c, --config-file`

path to configuration file.

enable service NAME@DOMAIN

`-c, --config-file`

path to configuration file.

disable service NAME@DOMAIN

`-c, --config-file`

path to configuration file.

An example of the service specification file is shown below.

```
]$ cat service-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Service
metadata:
  name: http-proxy
  domain: getaway-app
spec:
  contract_roles:
  - contract: frontend
    contract_role: Originator
```

29.5.10 Managing Service Tokens

show service-token SERVICE@DOMAIN [OPTIONS] The command shows the service token. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create service-token SERVICE@DOMAIN [OPTIONS] The command creates the service token. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

delete service-token SERVICE@DOMAIN [OPTIONS] The command deletes the service token. The following option is mandatory for this command:

`--token-id`

token identifier.

`-c, --config-file`

path to configuration file.

An example of the service token specification file is shown below.

```
]$ cat serviceToken-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Token
metadata:
  name: aws-proxy
spec:
  service: http-proxy
  domain: getaway-app
```

29.5.11 Managing Templates

You can manage templates using the following commands:

show template [OPTIONS] The command shows all templates. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show template NAME [OPTIONS] The command shows the template. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create template NAME [OPTIONS] The command creates the template. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

`--description`

description.

update template NAME [OPTIONS] The command updates the template. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`-desc, --description`

description.

`-c, --config-file`

path to configuration file.

delete template NAME

`-c, --config-file`

path to configuration file.

enable template NAME

`-c, --config-file`

path to configuration file.

disable template NAME

`-c, --config-file`

path to configuration file.

An example of the template specification file is shown below.

```
]$ cat template-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Template
metadata:
  name: multicast-cross-all-vpcs
spec:
  is_multicast: true
  orientation: directed
  roles:
  - name: Publisher
    code_binary:
↪"409C470100E7846300E000EF0A500793C11C004000EF409C470500E7846300C000EF5795C11C004000EF409C00178713C098
↪"
    propagation_interval_default: 5
    program_data_default:
      ppl: 0
      params:
      - name: hopCount
        value: 0
    code_map:
      Publisher: 0
    path_binary: "000000000001"
  - name: Subscriber
  code_binary:
↪"409C470100E7846300C000EF5791C11C004000EF409C470500E7846300C000EF5791C11C004000EF409C00178713C0989002
↪"
    propagation_interval_default: 5
    program_data_default:
      ppl: 0
      params:
      - name: hopCount
        value: 0
    code_map:
      Subscriber: 0
    path_binary: "000000000001"
```

29.5.12 Managing Zones

You can manage zones using the following commands:

show zone [OPTIONS] The command shows all zones in the fabric. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

show zone NAME [OPTIONS] The command shows the zone. You can use the options in this command as follows:

`-o, --output-format`

output format, either `json` or `yaml`.

`-c, --config-file`

path to configuration file.

create zone NAME [OPTIONS] The command creates the zone. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

`--description`

description.

update zone NAME [OPTIONS] The command updates the zone. You can use the specification file or the options in this command as follows:

`-f, --file`

path to the specification file.

`-desc, --description`

description.

`--tunnel-ip`

processor tunnel IP address: private or public.

`--ipsec`

processor IPsec status: true or false.

`--priority`

processor priority: high or low.

`-a, --append`

add processor to zone: true or false.

`-d, --delete`

delete processors from zone: processor name.

`-c, --config-file`

path to configuration file.

delete zone NAME

`-c, --config-file`

path to configuration file.

An example of the zone specification file is shown below.

```
]$ cat zone-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Zone
metadata:
   name: AWS Zone
spec:
   locations:
   - name: aws2
     tunnel_ip_type: private
     ipsec_enable: true
```

29.5.13 Working with Batches

You can manage batches using the following commands:

create batch [OPTIONS] The command creates the batch. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

update batch [OPTIONS] The command updates the batch. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

delete batch [OPTIONS] The command deletes the batch. The specification file is mandatory for this command.

`-f, --file`

path to the specification file.

`-c, --config-file`

path to configuration file.

An example of the batch specification file is shown below.

```
]$ cat batch-spec.yml
---
apiVersion: policy.bayware.io/v2
kind: Batch
metadata:
  name: getaway-app
spec:
- kind: Domain
```

(continues on next page)

(continued from previous page)

```
  metadata:
  domain: getaway-app
spec:
  auth_method:
    - LocalAuth
  domain_type: Application
- kind: Contract
  metadata:
    name: frontend
    domain: getaway-app
spec:
    template: anycast-cross-all-vpcs
    contract_roles:
    - template_role: Originator
      endpoint_rules:
        ingress:
          - protocol: icmp
    - template_role: Responder
      endpoint_rules:
        ingress:
          - protocol: icmp
          - protocol: tcp
            ports:
              - 8080
              - 5201
- kind: Service
  metadata:
    name: http-proxy
    domain: getaway-app
spec:
    contract_roles:
    - contract: frontend
      contract_role: Originator
- kind: Service
  metadata:
    name: getaway-svc
    domain: getaway-app
spec:
    contract_roles:
    - contract: frontend
      contract_role: Responder
```

CHAPTER 30

Policy Agent REST API

30.1 About REST API

Policy agent REST API ("REST API") enables you to interact with the policy agent instance using HTTP-requests.

REST API offers a read-only access to the policy agent operational data. Via REST API you can retrieve agent configuration, operational status of agent's interfaces and connections, information on application's network and service endpoints, records stored in agent's DNS resolver database. All responses are returned in json.

Additionally, REST API allows you to add and delete network endpoints. This functionality is used by Kubernetes CNI plugins to dynamically set up network endpoint for each pod. As such, it is highly recommended to keep REST API bound to the `localhost` interface, to which it is attached by default.

Also, policy agent REST API works as an interface between Resolver and Policy Agent database with DNS records.

REST API is set up automatically when the agent starts on the workload node. By default, REST API is exposed on `127.0.0.1:5500`. You can change the interface address and port number in the agent configuration file using a text editor or agent's `ib-configure` utility. To apply the new configuration you need to reload the agent.

30.2 Configuring REST API

30.2.1 Configuration file

REST API configuration is stored in the `ib-agent.conf` file located on the workload node at:

```
~/etc/ib-agent.conf
```

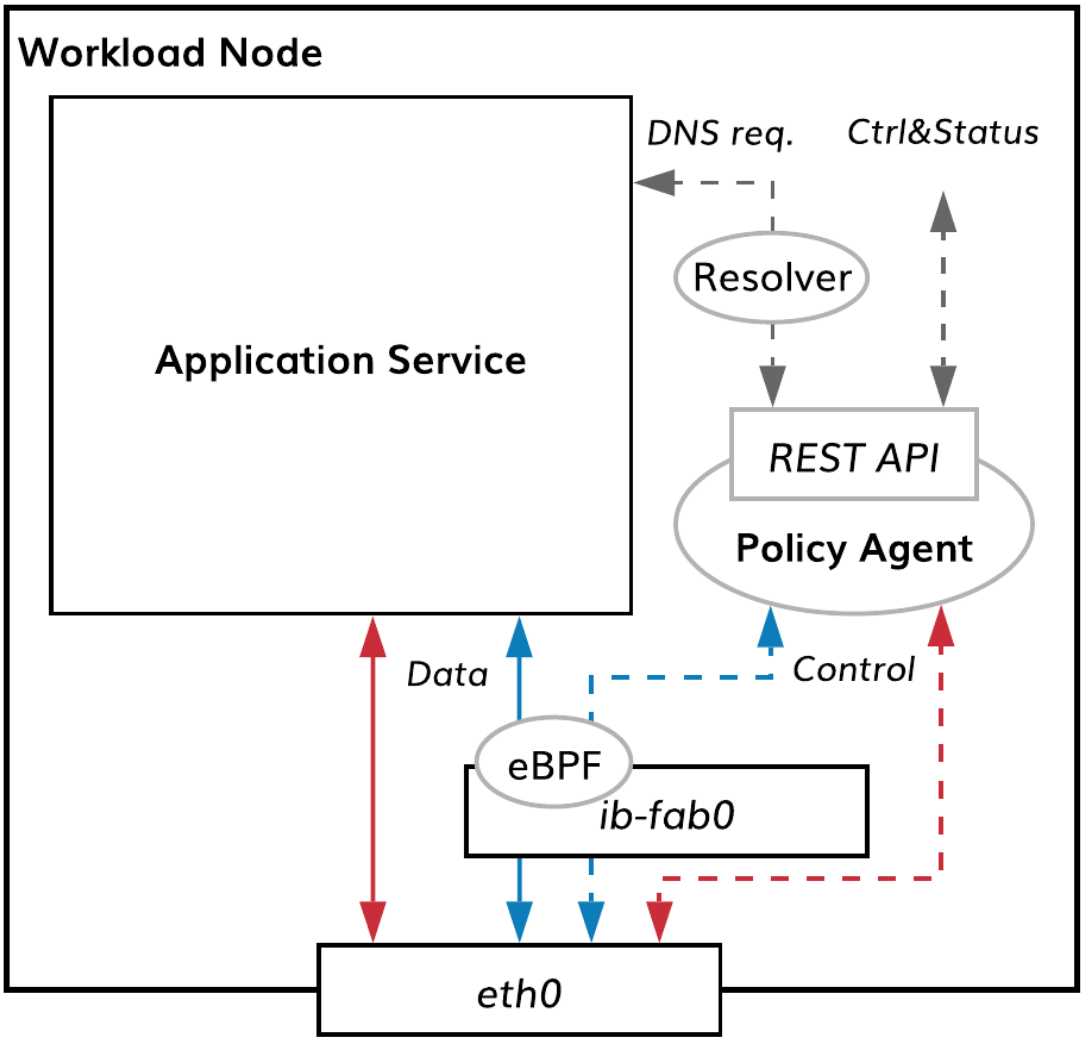


Fig. 30.1: FIG. Workload node with policy agent

The file contains REST API configuration details in the section titled `[rest]`. To verify information in the configuration file, run the command:

```
]$ cat /etc/ib-agent.conf
[agent]
controller = controller-texas2270.texasinc.poc.bayware.io
location = azr1
local_domain = ib.loc
token_file = /opt/bayware/ib-agent/conf/tokens.dat
log_file = /var/log/ib-agent/ib-agent.log
log_level = INFO

[net_iface]
name = ib-fab0
address = 192.168.250.0/24

[ctl_iface]
name = ib-ctl0

[mirror_iface]
name = ib-mon0

[cert]
ca_cert = /opt/bayware/certs/ca.crt
node_cert = /opt/bayware/certs/node.crt
node_key = /opt/bayware/certs/node.key

[rest]
rest_ip = 127.0.0.1
rest_port = 5500
log_file = /var/log/ib-agent/ib-agent-rest.log
log_level = WARNING

[resolver]
log_file = /var/log/ib-agent/ib-agent-resolver.log
log_level = WARNING
file_size = 100000
backup_count = 5
dns_port = 5053
```

30.2.2 Configuration commands

To change REST API configuration, use the policy agent configuration tool called `ib-configure` and located on the workload node at:

```
~/opt/bayware/ib-agent/bin/ib-configure
```

The tool enables you to change the IP address and/or TCP port on which Agent exposes its REST API.

The following commands require super-user privileges, so become root:

```
]$ sudo su -
```

Now, to bind the REST API to a different network interface on the node, run the command:

```
]# /opt/bayware/ib-agent/bin/ib-configure -a <IPv4_address>
```

To bind the REST API to a different TCP port on the same interface, run the command:

```
]# /opt/bayware/ib-agent/bin/ib-configure -r <TCP_port>
```

On a successful command execution, the tool will return the response as shown below:

```
]# /opt/bayware/ib-agent/bin/ib-configure -r 5500
agent configuration completed successfully
```

To apply the configuration changes, you need to reload the agent using the command:

```
]# systemctl reload ib-agent
```

30.3 Getting started with REST API

30.3.1 Making the first request

Here is an example of the REST API request and response (jq in this and other examples is used only for formatting):

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/resolver | jq
{
  "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app.ib.loc": {
    "hop_limit": 253,
    "last_update": "2019-08-23 22:42:48.665033 GMT",
    "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
    "unicast_ip": "192.168.250.7"
  },
  "aws2.originator.weather-api.getaway-app.ib.loc": {
    "hop_limit": 253,
    "last_update": "2019-08-23 22:42:48.665033 GMT",
    "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
    "unicast_ip": "192.168.250.7"
  },
  "originator.weather-api.getaway-app.ib.loc": {
    "hop_limit": 253,
    "last_update": "2019-08-23 22:42:48.665033 GMT",
    "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
    "unicast_ip": "192.168.250.7"
  }
}
```

30.3.2 Available resources

Policy agent REST API supports various categories of information, or various resources, that can be returned, such as:

- CERT

- CONNECTION
- IFACE
- NETWORK_ENDPOINT
- SERVICE
- STATUS

/cert Certificate is used to verify a node certificate employed by the policy agent. You can only fetch data.

/connection Connection is used to verify a current operational status of the logical connection between the workload and processor established by the policy agent. You can only fetch data.

/iface Iface is used to verify a current operational status of the network interfaces managed by the policy agent. You can only fetch data.

/network_endpoint Network endpoint is used to verify a current operational status of the network endpoints managed by the policy agent. You can only fetch data.

/service Service is used to verify a current operational status of the service endpoints managed by the policy agent. You can only fetch data.

/status Status is used to verify a current operational status of the policy agent. You can only fetch data.

30.4 Using REST API

30.4.1 Certificate object

Certificate object has only one endpoint.

GET /cert

Get the certificate.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/cert HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/cert | jq -R 'split("\\n")'
[
  "{\"result\":{\"x509\":\"Certificate:",
  "    Data:",
  "        Version: 3 (0x2)",
  "        Serial Number:",
  "            1e:f6:87:56:6d:7f:97:d7:27:d7:50:70:50:d7:b8:b5:dc:da:13:de",
  "        Signature Algorithm: sha256WithRSAEncryption",
  "        Issuer: O=texasinc, DC=texas2270, CN=texas-c0",
  "        Validity",
  "            Not Before: Aug 14 21:10:35 2019 GMT",
  "            Not After : Aug 11 21:10:35 2029 GMT",
  "        Subject: O=texasinc, DC=texas2270, DC=workload, CN=azr1-w01-texas2270",
  "        Subject Public Key Info:",
  "            Public Key Algorithm: rsaEncryption",
```

(continues on next page)

(continued from previous page)

```
"                RSA Public-Key: (2048 bit)",
"                Modulus:",
"                    00:be:01:be:35:18:b7:85:fc:8e:c8:9d:da:d2:27:",
"                    57:13:6b:8c:ab:cb:cf:39:15:f9:cf:b3:5d:d4:3e:",
"                    b3:9d:82:aa:1d:86:f5:b0:98:58:7f:32:18:50:f8:",
"                    61:ae:60:f6:43:2a:28:3f:99:83:cc:15:dd:ec:aa:",
"                    84:ac:c0:00:df:4d:a8:84:14:0a:94:ba:a8:37:3d:",
"                    84:c6:9c:ad:d5:ac:43:01:d0:86:07:36:c7:b6:5c:",
"                    c5:78:4b:de:ca:a5:d9:83:60:a9:bb:c1:1d:05:b0:",
"                    e8:71:5e:7f:45:98:77:3d:07:58:42:16:f1:0e:79:",
"                    5b:a4:22:95:0e:6c:cb:98:20:b7:d8:75:f6:69:1f:",
"                    88:c3:07:5c:56:96:12:d0:6f:00:60:14:3e:33:cc:",
"                    67:22:26:bf:ba:2e:59:a8:a2:e9:25:97:bc:6c:35:",
"                    54:ee:ef:e7:c3:fd:26:dd:5f:8b:40:71:9a:f0:63:",
"                    61:ac:b1:be:d2:3f:1e:98:50:6f:49:58:c9:12:51:",
"                    1f:48:61:5a:50:9a:45:51:4b:8a:fe:39:01:8e:df:",
"                    33:b3:68:34:da:a5:96:94:c1:16:4f:ae:d4:75:91:",
"                    0b:fc:ca:b6:69:97:a2:e8:ba:98:17:e7:ef:e6:5d:",
"                    1f:96:0c:58:d9:91:13:51:f6:4e:f9:9f:80:1d:c3:",
"                    43:c9",
"                Exponent: 65537 (0x10001)",
"        X509v3 extensions:",
"            X509v3 Key Usage: critical",
"                Digital Signature, Key Encipherment",
"            X509v3 Extended Key Usage: ",
"                TLS Web Server Authentication, TLS Web Client Authentication",
"            X509v3 Basic Constraints: critical",
"                CA:FALSE",
"            X509v3 Subject Key Identifier: ",
"                55:E5:A6:58:32:83:D6:D6:64:3A:E8:87:BC:BE:63:71:BC:72:B4:A6",
"            X509v3 Authority Key Identifier: ",
"                keyid:C1:F6:2F:CD:CF:70:9F:99:8B:2E:F8:B1:54:1E:08:C4:46:73:AA:19",
"",
"    Signature Algorithm: sha256WithRSAEncryption",
"         a1:3e:76:a6:d1:62:a3:c2:73:e4:2a:9d:b6:12:2a:22:48:1f:",
"         63:f5:f1:c4:f6:5f:e7:66:63:51:e4:9e:bc:02:87:a6:90:cd:",
"         e7:39:04:ec:ac:9d:58:42:95:ff:f0:34:72:a2:f1:4a:67:bf:",
"         f7:da:6f:ee:b9:bc:f8:51:27:5d:6e:e7:e9:89:c1:88:e9:f8:",
"         73:fd:b4:1c:fd:f8:41:66:5a:a7:51:bf:c8:dc:92:27:6a:e5:",
"         d4:59:60:70:6e:c2:2b:3d:e5:47:55:67:44:69:5f:0a:61:8a:",
"         4a:03:43:70:67:61:ec:bc:00:e1:80:35:b1:2d:32:bb:ba:0a:",
"         40:e3:b0:f4:c0:fe:fb:23:9d:c3:80:2a:df:23:9a:e5:81:ce:",
"         ea:22:1e:15:78:7b:4e:ab:2c:cd:b9:5e:cd:1e:57:89:07:f6:",
"         be:fd:a1:a0:e3:99:c5:0f:8f:1f:58:d2:e2:6f:e4:e6:1d:05:",
"         d0:1a:98:6e:ba:b5:b7:6e:90:67:c8:85:33:cd:7a:34:31:f6:",
"         e4:17:8f:cf:f4:3a:1b:48:95:56:5f:a0:da:31:23:9e:22:da:",
"         c8:f1:b8:8e:06:c7:23:7b:34:cb:12:a2:ca:42:17:65:12:2c:",
"         9b:a9:d9:6b:1e:e6:86:48:ed:41:4f:07:d8:6c:b5:2f:6d:da:",
"         b7:7d:ee:7a:4e:6f:b4:b4:6b:da:dd:71:cd:6b:90:52:61:d8:",
"         b6:8a:42:43:5c:29:75:fe:b8:e6:ec:73:80:35:66:72:32:e0:",
"         3e:a3:c0:84:bb:71:7e:34:d5:df:b8:de:7d:30:cb:fb:c7:1b:",
"         4d:60:0a:ca:d6:eb:cb:82:0b:5e:53:db:ad:4a:bc:8e:a3:f9:",
```

(continues on next page)

(continued from previous page)

```
  "         b4:de:bb:72:78:8e:b2:ee:75:14:33:08:bf:f4:8d:ab:19:2c:",
  "         f9:a8:cf:1b:e0:79:05:e8:55:da:35:1b:c3:fe:c8:b6:ec:3a:",
  "         37:e8:13:2b:15:90:c5:83:11:ae:38:a2:18:26:fb:50:8a:1c:",
  "         2b:c4:83:54:10:8a:35:05:f9:18:f7:13:e3:a6:13:1d:10:b4:",
  "         ff:27:77:a8:f9:6e:81:f9:1d:d9:c5:b5:3f:78:82:ad:71:6f:",
  "         82:74:89:76:ef:5e:91:8a:f7:fa:b4:ef:7f:a1:20:2f:15:bf:",
  "         27:8a:85:1d:ae:f3:10:26:45:d1:fa:be:e6:69:94:e6:4d:3b:",
  "         5c:53:76:32:8f:11:73:5b:2b:a4:82:45:74:4f:38:29:67:49:",
  "         f6:d2:6a:55:0f:c9:96:42:63:cb:75:3f:cf:93:60:26:96:76:",
  "         59:10:d2:9d:3c:5a:39:3a:50:44:f3:e7:54:15:9b:9c:e2:e8:",
  "         9e:ee:56:79:96:d6:e4:e8",
  "\"}}"
]
```

30.4.2 Connection object

Connection object has only one endpoint.

GET /connection

Get information about the current connection status.

HTTP request for this endpoint is shown below.

```
GET /api/v1/connection HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$ curl -s http://127.0.0.1:5500/api/v1/connection | jq
{
  "result": {
    "304": {
      "keepalive_timestamp": "Fri, 23 Aug 2019 22:29:21 GMT",
      "local_conn": 304,
      "local_port": 1,
      "nonce_timestamp": "Fri, 23 Aug 2019 22:29:19 GMT",
      "remote_address": "fd3210d7b78fea9d20c9c41f59347aed",
      "remote_conn": 258,
      "remote_mac": "16c2b80359c1",
      "remote_node_role": "processor",
      "remote_port": 32,
      "remote_portname": "ib_0a000206",
      "status": "active"
    }
  }
}
```

30.4.3 Iface object

Iface object has three endpoints.

Control interface

GET /iface/ctl_iface

Get information about the current control interface status.

HTTP request for this endpoint is shown below.

```
GET /api/v1/iface/ctl_iface HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$ curl -s http://127.0.0.1:5500/api/v1/iface/ctl_iface | jq -R 'split("\\n")'
[
  "{\"result\":{\"ctl_iface\":\"",
  "",
  "203: ib-ctl0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel
  state DOWN group default qlen 1000",
  "    link/ether 06:55:44:13:ff:35 brd ff:ff:ff:ff:ff:ff",
  "    RX: bytes  packets  errors  dropped overrun mcast   ",
  "    0          0        0       0       0       0       ",
  "    TX: bytes  packets  errors  dropped carrier collsns ",
  "    0          0        0       0       0       0\"}}"
]
```

Mirror interface

GET /iface/mirror_iface

Get information about the current mirror interface status.

Note: The Rx bytes and packets counters will show non-zero values only if you have the port mirroring enabled in the specification of at least one contract role whose service endpoint(s) present on the node.

HTTP request for this endpoint is shown below.

```
GET /api/v1/iface/mirror_iface HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$ curl -s http://127.0.0.1:5500/api/v1/iface/mirror_iface | jq -R 'split("\\n")'
[
  "{\"result\":{\"mirror_iface\":\"",
  "",
  "204: ib-mon0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state
  UNKNOWN group default qlen 1000",
  "    link/ether 06:72:48:6c:8f:6d brd ff:ff:ff:ff:ff:ff",
  "    inet6 fe80::472:48ff:fe6c:8f6d/64 scope link ",
  "       valid_lft forever preferred_lft forever",
  "    RX: bytes  packets  errors  dropped overrun mcast   ",
  "    0          0        0       0       0       0       ",
```

(continues on next page)

(continued from previous page)

```
  "    TX: bytes  packets  errors  dropped carrier collsns ",
  "    12460      178      0       0       0       0\"}}"
]
```

Network interface

GET /iface/net_iface

Get information about the current network interface status.

HTTP request for this endpoint is shown below.

```
GET /api/v1/iface/net_iface HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$ curl -s http://127.0.0.1:5500/api/v1/iface/net_iface | jq -R 'split("\\n")'
[
  "{\"result\":{\"net_iface\":\"",
  "",
  "207: ib-fab0@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1350 qdisc
  fq_codel state UNKNOWN group default qlen 1000",
  "    link/ether 96:9e:eb:10:c7:31 brd ff:ff:ff:ff:ff:ff",
  "    inet 192.168.250.1/30 scope global ib-fab0",
  "       valid_lft forever preferred_lft forever",
  "    RX: bytes  packets  errors  dropped overrun mcast   ",
  "    439537314  2251739  0       0       0       0       ",
  "    TX: bytes  packets  errors  dropped carrier collsns ",
  "    325448824  2091568  0       0       0       0\"}}"
]
```

30.4.4 Network endpoint object

Network endpoint object has multiple endpoints.

Network endpoint status

GET /network_endpoint

Get information about the current network endpoint status.

HTTP request for this endpoint is shown below.

```
GET /api/v1/network_endpoint HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/network_endpoint | jq
{
  "result": [
    {
      "cga": "fd32:10d7:b78f:7699:242e:1b2e:0e61:d882",
      "cga_params": {
        "ccount": 0,
        "ext": "",
        "modifier": 2.850508822951617e+38,
        "prefix": "fd32:10d7:b78f:7699::"
      },
      "ip": "192.168.250.1",
      "mac": "96:9e:eb:10:c7:31",
      "name": "azr1-w01-texas2270",
      "ne_id": 1554,
      "ne_instance": "azr1-w01-texas2270"
    }
  ]
}
```

Create network endpoint

PUT /network_endpoint/{ne_instance}

Create network endpoint. API is used by CNI plugin.

HTTP request for this endpoint is shown below.

```
PUT /api/v1/network_endpoint/instance123 HTTP/1.1
Host: 127.0.0.1:5500
```

Additional parameters must be sent in the request body in JSON format as shown in the example below.

```
{
   "name":"nginx-deployment-77f588df6b-jck2q",
   "ip_address":"10.10.110.82",
   "mac":"6a:49:c9:11:4c:60",
   "tokens":[
     "2646f16e-0dec-4577-9e43-076b7be1b0ab"
   ]
}
```

Delete network endpoint

DELETE /network_endpoint/{ne_instance}

Delete network endpoint. API is used by CNI plugin.

```
DELETE /api/v1/network_endpoint/instance123 HTTP/1.1
Host: 127.0.0.1:5500
```

30.4.5 Service object

Service object has multiple endpoints.

Available local service endpoints

GET /service/available

Get information about the available local service endpoints.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/available HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/available | jq
{
  "azr1-w01-texas2270": {
    "ne_cfg_hash": "0612947c582c4cf105ac3428c3f5b613a4a5",
    "services": [
      {
        "contract": "weather-api",
        "contract_id": 3221325477,
        "contract_role": "Responder",
        "domain": "getaway-app",
        "endpoint_rules": [
          {
            "protocol": "icmp"
          },
          {
            "ports": [
              8080,
              5201
            ],
            "protocol": "tcp"
          }
        ],
        "is_multicast": false,
        "port_mirror_enabled": false,
        "propagation_interval": 5,
        "remote_endpoint_rules": [
          {
            "protocol": "icmp"
          },
          {
            "protocol": "tcp"
          }
        ],
        "role_index": 1,
        "se_cfg_hash": "46eeb901fb5ad0a0027f4c2b9d351b85",
        "service_rdn": "responder.weather-api.getaway-app",
        "stat_enabled": false
```

(continues on next page)

(continued from previous page)

```
      }
    ],
    "success": true,
    "type": "serviceResponse"
  }
}
```

Registered local service endpoints

GET /service/registered

Get information about the registered local service endpoints.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/registered HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/registered | jq
{
  "azr1-w01-texas2270": [
    {
      "contract": "weather-api",
      "contract_role": "Responder",
      "domain": "getaway-app",
      "filtration": {
        "bpf_maps": {
          "MAP_F_IN_PORT": [
            {
              "key": "612a8633-5114-06",
              "val": "00"
            },
            {
              "key": "612a8633-901f-06",
              "val": "00"
            }
          ],
          "MAP_F_IN_PROTO": [
            {
              "key": "612a8633-3a",
              "val": "00"
            }
          ],
          "MAP_F_OUT_PROTO": [
            {
              "key": "612a8633-06",
              "val": "00"
            },
            {
              "key": "612a8633-01",
```

(continues on next page)

(continued from previous page)


```
            "val": "00"
          }
        ],
        "MAP_IN_UNI_SE": [
          {
            "key": "242e1b2e0e61d882-c00186a5",
            "val": "612a8633-b0-c0a8fa0100000000000000000000000-969eeb10c731"
          }
        ],
        "MAP_OUT_V4_SE": [
          {
            "key": "c0a8fa01-400186a5",
            "val": "612a8633-a0-fd3210d7b78f7699242e1b2e0e61d882-0a8633"
          }
        ]
      },
      "endpoint_rules": {
        "egress": [
          {
            "protocol": "icmp"
          },
          {
            "protocol": "tcp"
          }
        ],
        "ingress": [
          {
            "protocol": "icmp"
          },
          {
            "ports": [
              8080,
              5201
            ],
            "protocol": "tcp"
          }
        ]
      }
    },
    "flow_label": 689715,
    "flow_label_hex": "0xa8633",
    "group_id": 3221325477,
    "group_id_hex": "0xc00186a5",
    "role_index": 1,
    "service_resolve": "responder.weather-api.getaway-app"
  }
 ]
}
```

Remote service endpoints

GET /service/remote

Get information about the current remote service endpoints.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/remote HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/remote | jq
{
  "remote_service_endpoints": [
    {
      "bpf_maps": {
        "MAP_IN_UNI": [
          {
            "key": "fd3210d7b78fdb9530e6ae3ff3e72973-069e86",
            "val": "c00186a5-c0a8fa07-00000000000000000000000000000000"
          }
        ],
        "MAP_OUT_V4_DST": [
          {
            "key": "c0a8fa07",
            "val": "400186a5-fd3210d7b78fdb9530e6ae3ff3e72973-16c2b80359c1"
          }
        ]
      },
      "hop_limit": 253,
      "last_update": "2019-08-23 22:40:15.516714 GMT",
      "local_group_id": 3221325477,
      "remote_cga": "fd32:10d7:b78f:db95:30e6:ae3f:f3e7:2973",
      "remote_flow_label": 433798,
      "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app
↪",
      "unicast_ip": "192.168.250.7"
    }
  ]
}
```

Fetch all resolver database records

GET /service/resolver

Get all records from the resolver database.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/resolver HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/resolver | jq
{
  "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app.ib.loc": {
    "hop_limit": 253,
    "last_update": "2019-08-23 22:42:48.665033 GMT",
    "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
    "unicast_ip": "192.168.250.7"
  },
  "aws2.originator.weather-api.getaway-app.ib.loc": {
    "hop_limit": 253,
    "last_update": "2019-08-23 22:42:48.665033 GMT",
    "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
    "unicast_ip": "192.168.250.7"
  },
  "originator.weather-api.getaway-app.ib.loc": {
    "hop_limit": 253,
    "last_update": "2019-08-23 22:42:48.665033 GMT",
    "service_domain_name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
    "unicast_ip": "192.168.250.7"
  }
}
```

Resolve name into IP address

GET /service/resolver/{service_RDN}.{local_domain}

Get IP address for the specified DNS name. API is used by the policy agent resolver.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/resolver/originator.weather-api.getaway-app.ib.loc HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/resolver/originator.weather-api.getaway-
↪app.ib.loc | jq
{
  "host": "192.168.250.7",
  "success": true
}
```

Service endpoint statistics

GET /service/stat

Get the current service endpoint statistics.

Note: To see statistics on a particular service endpoint you need to enable it in the specification of the contract role associated with the endpoint.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/stat HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
{
   "stat": [
       {
           "name": "aws2-w02-texas2270.aws2.originator.weather-api.getaway-app",
           "bytes_in": 2300,
           "bytes_out": 1200,
           "pkts_in": 98,
           "pkts_out": 123,
           "drops_in": 0,
           "drops_out": 8
       }
   ]
}
```

eBPF Maps

GET /service/ebpfmaps

Get content of the eBPF maps.

HTTP request for this endpoint is shown below.

```
GET /api/v1/service/ebpfmaps HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:

```
]$  curl -s http://127.0.0.1:5500/api/v1/service/ebpfmaps | jq
{
  "MAP_CONFIG": [
    {
      "key": "01",
      "val": "fd3210d7b78f7699"
    },
    {
      "key": "06",
      "val": "cc00000000000000"
    },
    {
      "key": "05",
      "val": "969eeb10c7310000"
    },
    {
      "key": "07",
      "val": "38f122bb7af79677"
    },
    {
      "key": "02",
```

(continues on next page)

(continued from previous page)


```
      "val": "cb00000000000000"
    },
    {
      "key": "04",
      "val": "cf00000000000000"
    },
    {
      "key": "03",
      "val": "06554413ff350000"
    }
  ],
  "MAP_F_IN_PORT": [
    {
      "key": "612a8633-5114-06",
      "val": "00"
    },
    {
      "key": "612a8633-901f-06",
      "val": "00"
    }
  ],
  "MAP_F_IN_PROTO": [
    {
      "key": "612a8633-3a",
      "val": "00"
    }
  ],
  "MAP_F_OUT_PORT": [],
  "MAP_F_OUT_PROTO": [
    {
      "key": "612a8633-01",
      "val": "00"
    },
    {
      "key": "612a8633-06",
      "val": "00"
    }
  ],
  "MAP_IN_SSM": [],
  "MAP_IN_SSM_SE": [],
  "MAP_IN_SSM_T1": [],
  "MAP_IN_UNI": [
    {
      "key": "fd3210d7b78fdb9530e6ae3ff3e72973-069e86",
      "val": "c00186a5-c0a8fa07-00000000000000000000000000000000"
    }
  ],
  "MAP_IN_UNI_SE": [
    {
      "key": "242e1b2e0e61d882-c00186a5",
      "val": "612a8633-b0-c0a8fa0100000000000000000000000-969eeb10c731"
    }
```


(continues on next page)

(continued from previous page)


```
  ],
  "MAP_OUT_V4_DST": [
    {
      "key": "c0a8fa07",
      "val": "400186a5-fd3210d7b78fdb9530e6ae3ff3e72973-16c2b80359c1"
    }
  ],
  "MAP_OUT_V4_SE": [
    {
      "key": "c0a8fa01-400186a5",
      "val": "612a8633-a0-fd3210d7b78f7699242e1b2e0e61d882-0a8633"
    }
  ],
  "MAP_OUT_V6_DST": [],
  "MAP_OUT_V6_SE": []
}
```


30.4.6 Status object

Status object has only one endpoint.

GET /status

Get information about the current policy agent status.

HTTP request for this endpoint is shown below.

```
GET /api/v1/status HTTP/1.1
Host: 127.0.0.1:5500
```

Here is an example of the REST API request and response:


```
]$ curl -s http://127.0.0.1:5500/api/v1/status | jq
{
  "controller": "controller-texas2270.texasinc.poc.bayware.io",
  "host_id": "fd3210d7b78f769938f122bb7af79677",
  "hostname": "azr1-w01-texas2270",
  "local_domain": "ib.loc",
  "location": "azr1",
  "ready": true,
  "registered": true,
  "version": "1.2.0"
}
```


30.5 Quick Reference

GET /cert

`http://127.0.0.1:5500/api/v1/certificate`

Get the certificate.

GET /connection

`http://127.0.0.1:5500/api/v1/connection`

Get information about the current connection status.

GET /iface/ctl_iface

`http://127.0.0.1:5500/api/v1/iface/ctl_iface`

Get information about the current control interface status.

GET /iface/mirror_iface

`http://127.0.0.1:5500/api/v1/iface/mirror_iface`

Get information about the current mirror interface status.

GET /iface/net_iface

`http://127.0.0.1:5500/api/v1/iface/net_iface`

Get information about the current network interface status.

GET /network_endpoint

`http://127.0.0.1:5500/api/v1/network_endpoint`

Get information about the current network endpoint status.

PUT /network_endpoint/{ne_instance}

`http://127.0.0.1:5500/api/v1/network_endpoint/instance123`

Create network endpoint.

DELETE /network_endpoint/{ne_instance}

`http://127.0.0.1:5500/api/v1/network_endpoint/instance123`

Delete network endpoint.

GET /service/available

`http://127.0.0.1:5500/api/v1/service/available`

Get information about the available local service endpoints.

GET /service/registered

`http://127.0.0.1:5500/api/v1/service/registered`

Get information about the registered local service endpoints.

GET /service/remote

`http://127.0.0.1:5500/api/v1/service/remote`

Get information about the current remote service endpoints.

GET /service/resolver

`http://127.0.0.1:5500/api/v1/service/resolver`

Get all records from the resolver database.

GET /service/resolver/{service_RDN}.{local_domain}

`http://127.0.0.1:5500/api/v1/service/resolver/originator.weather-api.getaway-app.ib.loc`

Get IP address for the specified DNS name.

GET /service/stat

`http://127.0.0.1:5500/api/v1/service/stat`

Get the current service endpoint statistics.

GET /service/ebpfmaps

`http://127.0.0.1:5500/api/v1/service/ebpfmaps`

Get content of the eBPF maps.

GET /status

`http://127.0.0.1:5500/api/v1/status`

Get information about the current policy agent status.

CHAPTER 31

Network Microservice SDK

31.1 About document

This document describes the Network Microservice Software Development Kit (SDK).

31.2 Overview

31.2.1 What is a network microservice microprogram?

A microprogram is a collection of instructions that performs action set computation when executed by a Network Microservice Processor.

Bayware's approach to microprogram development satisfies the following requirements:

- **extremely fast execution** - microprogram executable code is a collection of RISC-V instructions that are efficiently executed by the network microservice processor
- **excellent expressive power** - any forwarding logic can be described
- **first-grade compactness** - source-based routing implementation comprises three four- or two-byte instructions

31.2.2 What is the network microservice SDK?

The SDK is a tool for microprogram development allowing the generation of a collection of RISC-V instructions executable by a network microservice processor.

31.2.3 What can the network microservice SDK do?

The SDK allows developers to use a high-level language (Java) for microprogram development and automatically generate executable code. This eliminates the necessity to program in RISC-V and dive deep into the Network Microservice Execution Environment implementation.

31.2.4 Who can use the network microservice SDK?

Anyone who needs to introduce new packet forwarding logic and has basic knowledge of Java can use the SDK.

31.3 Getting Started

31.3.1 Development Kit

A service template is generated using the development kit library. The service template is used to form the IBFlow.

The library uses a Java-like syntax.

The development and execution life cycle consists of the following steps

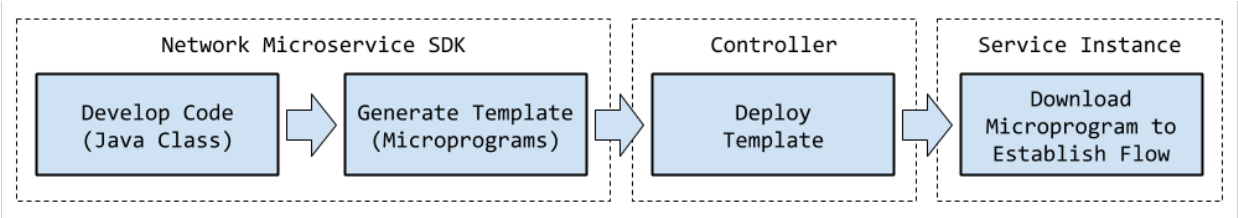


Fig. 31.1: Figure. The ENF Lifecycle

- **Develop** - node execution rules formed (Java)
- **Template Generation** - service template (signed JSON document) formed for created class
- **Deploy** - generated service template deployed at controller
- **IBStream Generation** - application forms IB Flows

31.3.2 Create “Hello world!”

Simple Forward (Source-based Routing)

Step1. Write Source Code

To start microprogram development, create Java Class. The class must be an extension of IceBuilder.

```
1 package org.test;
2
3 import io.bayware.builder.*;
4 import io.bayware.builder.ExecMethod.ExecType;
```

(continues on next page)

(continued from previous page)

```
import io.bayware.type.anno.*;
import io.bayware.type.*;

public class SimpleForward extends IceBuilder {

}
```

Create a method to add a package forwarding logic.

```
@Method(name = "frwrd")
public ExecMethod mForward() {
}
```

The class can contain one or several methods. Executable code will be generated for each method with the annotation `@Method`.

Add a collection of instructions to the method.

```
@Method(name = "frwrd")
public ExecMethod mForward() {
   return new ExecMethod(
           Op.forward(Path.egressRCI0),
           Op.end()
   );
}
```

Wherein:

`Op.forward` – send packet

`Path.egressRCI0` – use RCI0 received after path parsing

`Op.end` – terminate code execution

As a result, the class will be.

```
package org.test;

import io.bayware.builder.*;
import io.bayware.type.anno.*;
import io.bayware.type.*;

public class SimpleForward extends IceBuilder {

   @Method(name = "frwrd")
   public ExecMethod mForward() {
       return new ExecMethod(
               Op.forward(Path.egressRCI0),
               Op.end()
       );
   }
}
```

That's it! The logic of source-based routing is implemented.

Step 2. Generate Executable Code

To generate executable code, use the procedure `ctx.makeHex()`.

Create a context using the class `SimpleForward` defined on the step 1.

```
package org.test;

import io.bayware.builder.*;

public class SimpleForwardTest {


   public static void main(String[] args) {
        Context ctx = Context.create(SimpleForward.class);
        ctx.makeHex();
   }
}
```

Run `ctx.makeHex()`.

The resulting file `SipmleForward.hex` contains a hex-encoded executable code in the first line and a pointer to the first instruction of method in the second line.

```
00859F0301EFA62300000073
frwrd:0
```

Step 3 (optional). Generate Assembly Code

Assembly Code

To generate assembly code, use the procedure `ctx.makeAsm()`.

The resulting file `SipmleForward.s` contains a collection of RISC-V instructions.

```
lh x30, 0x8(x11)
sw x30, 0xc(x31)
ecall
```

This assembly code can be compiled into machine code using any RISC-V compiler.

Dump

To generate dump for assembly code analysis, use the procedure `ctx.makeDump()`.

The resulting file `SipmleForward.dump` contains a collection of RISC-V instructions with additional information.

```
-- Method:frwrd
-------  Op.forward
0x00859F03:lh x30, 0x8(x11)
0x01EFA623:sw x30, 0xc(x31)
-------  Op.end
0x00000073:ecall
```

31.3.3 Using Several Methods for Packet Forwarding

A class can hold several methods. Each method receives its own name defined in the annotation `@Method`.

An example is shown below.

```
package org.test;

import io.bayware.builder.*;
import io.bayware.type.anno.*;
import io.bayware.type.*;

public class SimpleForward extends IceBuilder {

   @Method(name = "frwrd1")
   public ExecMethod mForward1() {
       return new ExecMethod(
               Op.forward(Path.egressRCI0), Op.end()
       );
   }

   @Method(name = "frwrd2")
   public ExecMethod mForward2() {
       return new ExecMethod(
               Op.forward(Path.egressRCI1), Op.end()
       );
   }
}
```

Executing the procedure `ctx.makeHex()` provides the result shown below.

```
00859F0301EFA6230000007300A59F0301EFA62300000073
frwrd1:0
frwrd2:12
```

This code comprises the two methods. An offset for the first instruction of method `frwrd1 is 0. Whereas an offset for the first instruction of method `frwrd2` is 12.

31.4 Variables

31.4.1 Variable Types

A microprogram developer can use the following variable types:

- Word – 4 bytes
- HalfWord – 2 bytes
- Byte – 1 byte
- Bit – flags (from 0 to 31)

31.4.2 Class for Variables

Record Definition

To declare variables, define the class for a record. The record holds a collection of variables with their types.

As an example, the class for the record `SimpleVar<T>` holds the variables `var1`, `var2`, `var3`.

```
package org.test;

import io.bayware.type.record.FieldRecord;
import io.bayware.type.fields.*;
import io.bayware.type.fields.Byte;

public class SimpleVar<T> extends FieldRecord {
   public Field<Word> var1;
   public Field<HalfWord> var2;
   public Field<Byte> var3;
}
```

Declaration of a bit-type variable requires specifying the number of bits to read `@Bit(read = )`. An example is shown below.

```
package org.somebody.model;

import io.bayware.type.record.FieldRecord;
import io.bayware.type.fields.Field;
import io.bayware.type.fields.Byte;
import io.bayware.type.fields.Bit;

public class my_field_rec<T> extends FieldRecord {

   @Bits(read = 4)
   public Field<Bit> var3;

   @Bits(read = 2)
   public Field<Bit> var4;

   @Bits(read = 2)
   public Field<Bit> var5;
}
```

To holds the values of the variables `var3`, `var4`, `var5`, one byte is to be allocated.

If one more variable is to be declared, for example –

```
@Bits(read = 4)
public Field<Bit> var6;
```

an additional byte will be allocated wherein 4 bits to be assigned for the new variable.

Using Several Records

More than one record can be declared. For example, to implement an advanced trace-route logic the micro-program has to collect en route: hop identifiers, timestamps, ingress and egress connections. To hold the values of this variables, the structure shown below is to be defined.

```java
package org.somebody.sample;

import io.bayware.type.record.FieldRecord;
import io.bayware.type.fields.*;
import io.bayware.type.anno.*;

@RecordCount(maxCount = 30)
public class TraceRecord<T> extends FieldRecord {
   public Field<Word> hopIdLo;
   public Field<Word> hopIdHi;
   public Field<Word> timeStamp;
   public Field<HalfWord> inRCI;
   public Field<HalfWord> outRCI;
}
```

Note, that the class annotation defines the maximum number of records - `@RecordCount(maxCount = 30)`.

Record-in-Record

Record can be a field of another record. In this case, `Segment` is specified as a field type.

As an example, `TraceRecord` is a filed of `TraceInfo`.

```java
package org.somebody.sample;

import io.bayware.type.record.GeneralRecord;
import io.bayware.type.fields.Byte;
import io.bayware.type.fields.*;

public class TraceInfo extends GeneralRecord {
   public Field<Word> passedHopCounter;
   public TraceRecord<Segment> hopInfo;
   public Field<Byte> test;
}
```

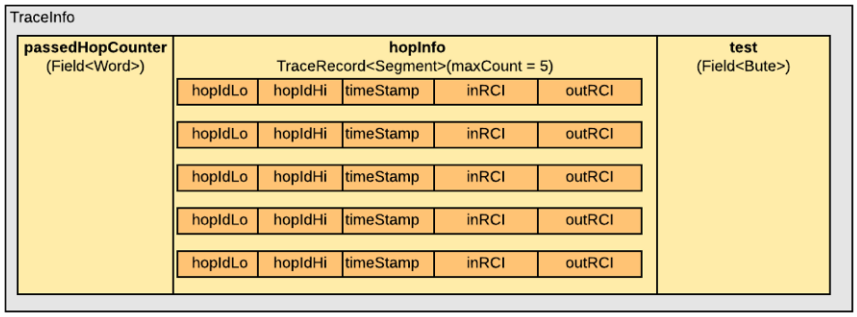


Fig. 31.2: Figure. An example of record-in-record

31.4.3 Using Variables in Microprogram

As a new record class is created, it can be used in a microprogram. An example is shown below.

```
package org.test;

import io.bayware.builder.*;
import io.bayware.type.anno.*;
import io.bayware.type.*;

public class SimpleForward extends IceBuilder {

   @Var(type = VarType.packet)
   SimpleVar packVar;

   @Method(name = "frwrd")
   public ExecMethod method1() {
       return new ExecMethod(Op.forward(Path.egressRCI0),
                             Op.assign(packVar.var1, 23),
                             Op.assign(packVar.var2, 34),
                             Op.assign(packVar.var3, packVar.var1),
                             Op.plus(packVar.var3, 3),
                             Op.end()
       );
   }
}
```

The variable `var1` receives the value of 23.

The variable `var2` receives the value of 34.

The variable `var3` receives the value of the variable `var1`.

Add the value of 3 to `var3` and store the result in `var3`.

31.4.4 Pre-defined Variables

A few variables are already pre-defined. They can be used in every microprogram.

Record: Switch

The record `Node` holds the basic information about the node.

switchId This variable holds the Switch Identifier calculated as a 20-bit MD5 hash of switch's CGA.

switchIPAddr0 This variable holds the most-significant word of the switch's IPv6 CG address. It is the upper four bytes of the netprefix.

switchIPAddr1 This variable holds the second word of the switch's IPv6 CG address. It is the lower four bytes of the netprefix.

switchIPAddr2 This variable holds the third word of the switch's IPv6 CG address. It is the upper four bytes of the cryptographically generated interface identifier.

switchIPAddr3 This variable holds the least-significant word of the switch's IPv6 CG address. It is the lower four bytes of the cryptographically generated interface identifier.

switchType Switch Type

timeStamp Current Time

Record: Path

The record Path holds the path parsing results.

pathCreationTime The Path Creation Time parsed from the Path Chunk.

topicSwitchTag The Topic Switch Tag extracted from the Topic Policy Table.

pktType 8'b0–Type II; 8'b1–Type III; 8'b2–Type I packet without optional PLD_DATA Chunk; 8'b3–Type I packet with optional PLD_DATA Chunk; Else–undefined.

ingressRCI The Segment Identifier of the ingress link, on which the packet received.

egressRCI0 The potential egress RCI parsed from the Path Chunk.

egressRCI1 The potential egress RCI parsed from the Path Chunk.

egressRCI3 The potential egress RCI parsed from the Path Chunk.

egressRCI4 The potential egress RCI parsed from the Path Chunk.

egressRCI5 The potential egress RCI parsed from the Path Chunk.

egressRCI6 The potential egress RCI parsed from the Path Chunk.

egressRCI7 The potential egress RCI parsed from the Path Chunk.

Record: Connection

The record Connection holds descriptions for each egress connection.

connectionCreationTime The timestamp for the connection creation time.

connectionQualityCredits Connection aggregated quality credits. This variable characterizes connection quality.

connectionStatus The connection is up or down.

connectionUtilizationCredits Connection aggregated utilization credits. This variable characterizes connection utilization.

connId This variable contains a 12-bit RCI. Note that values between 0 and 0xF are not valid connection IDs. A few examples: 0x0 – a "dummy" hop; 0x1 – a subnet in a dynamic path description; 0xF – a broadcast request.

remoteNodeId This variable holds the 32-bit identifier of the remote node assigned to the node by the controller.

remoteNodeIPAddr0 This variable holds the most-significant word of the remote node's IPv6 CG address. It is the upper four bytes of the netprefix.

remoteNodeIPAddr1 This variable holds the second word of the remote node's IPv6 CG address. It is the lower four bytes of the netprefix.

remoteNodeIPAddr2 This variable holds the third word of the remote node's IPv6 CG address. It is the upper four bytes of the cryptographically generated interface identifier.

remoteNodeIPAddr3 This variable holds the least-significant word of the remote node's IPv6 CG address. It is the lower four bytes of the cryptographically generated interface identifier.

remoteNodeNetRole The network role (switch or host) of the remote node.

Record: Tag

The record `Tag` holds a set of connections with a given tag.

RCI0 A RCI associated with the tag used during lookup.

RCI1 A RCI associated with the tag used during lookup.

RCI2 A RCI associated with the tag used during lookup.

RCI3 A RCI associated with the tag used during lookup.

RCI4 A RCI associated with the tag used during lookup.

RCI5 A RCI associated with the tag used during lookup.

RCI6 A RCI associated with the tag used during lookup.

RCI7 A RCI associated with the tag used during lookup.

Record: ActionSet

The record `ActionSet` holds information on actions required by the microprogram.

executeActionSetType As defined in the Execution Environment document.

updateFlowDefaultActionSetType As defined in the Execution Environment document.

updatePacketPathPointer As defined in the Execution Environment document.

updatePacketProgramData As defined in the Execution Environment document.

newFlowActionSetConnectionFlags As defined in the Execution Environment document.

newFlowActionSetConnection As defined in the Execution Environment document.

newTopicActionSetConnection As defined in the Execution Environment document.

Record: *

The owner may pre-define its owner record or a plurality of records.

any Variable defined by the owner of Overlay.

31.4.5 Scope of Variables

Variables have the following scopes:

- **local** – the variable is initialized during microprogram execution; after execution the variable is to be destroyed,
- **packet** – the variable is defined for a packet and passed with the packet from node to node,
- **flow** – the variable is defined for a given flow and visible for all packets of the flow,
- **topic** – the variable is defined for a given topic and visible for all packets of the topic.

The scope is specified at the time of defining the variable within a microprogram class.

For example:

```
public class SimpleForward extends IceBuilder {

   @Var(type = VarType.packet)
   SimpleVar packVar;

   @Var(type = VarType.local)
   SimpleVar localVar;

...
```

31.5 Statements

From a developer point of view, the method of microprogram is a collection of operators: `operator1`, `operaror2`, `operator3`.

The class `ExecMethod` holds a set of operators (i.e. statements) that are to be executed.

```
new ExecMethod(IStatement...)
```

Wherein IStatement… - a set of operators for execution.

For example:

```
new ExecMethod(
       Op.plus(tracePacket.passedHopCounter,1),
       Op.novel(tracePacket.hopInfo),
       Op.assign(tracePacket.hopInfo.hopId, Node.switchIPAddr0),
       Op.assign(tracePacket.hopInfo.timeStamp, Node.timeStamp),
       Op.assign(tracePacket.hopInfo.inRCI,Path.ingressRCI),
       Op.assign(tracePacket.hopInfo.outRCI,Path.egressRCI0),
       Op.forward(Path.egressRCI0),
       Op.end());
```

All the operators are defined in the class `io.bayware.builder.Op`.

CHAPTER 32

API Reference

32.1 About Document

These documents describes application programming interfaces (APIs) of the Network Microservice Overlay.

32.2 Overview

Three APIs are used for interaction between Bayware components and third-party systems. These APIs are:

- Controller Northbound RESTful API (NB API)
- Controller Southbound RESTful API (SB API)
- Agent RESTful API (Agent API)

NB API provides third-party software with the access to the **service, user, and node management** functions. The controller GUI utilizes the NB API as well.

SB API is the interface between the controller and IceBreaker Nodes. It provides the nodes with the access to the **node, link, and service endpoint registration** functions, including initial configuration and subsequent configuration support of these entities. Besides, the SB API processes the node's requests for publishing statistics and alerts.

Agent API is used by Bayware-enabled applications to **set up and manage the service endpoints**.

The Engine Connection Request Trigger (CRT) is the special interface used by the controller to trigger the switch to establish connection for **immediate data update on either switch or controller side**, e.g. flow CRL update, topic policy update, re-tagging connections, tunnel endpoint provisioning, providing meter values.

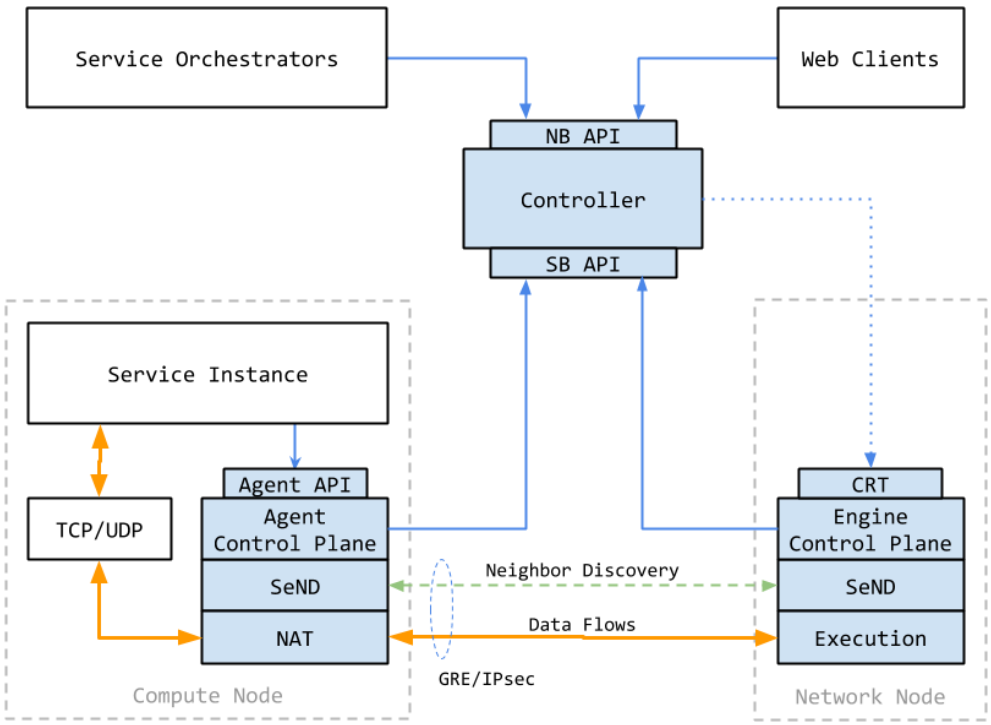


Fig. 32.1: Figure. Network Microservice Protocols and APIs

32.2.1 Controller Northbound RESTful API

32.2.2 Controller Southbound RESTful API

Using API

User Authentication

To access the controller Southbound API, the user must already be authenticated and possess a valid access token. Nodes pass this token to the controller in the HTTP headers of their requests. An example of such a token placed in a HTTP-header is

```
Authentication: b438e13737144ca8a39cad03b8986dcb
```

Additional Parameter Format

Any additional parameters are passed to the controller in the JSON format. The format is specified in HTTP headers of the requests.

```
Content-Type: application/json
```

Version

All requests use the format shown below. The base prefix is the same in all the requests.

```
/api/<version>/<url>
```

The current implementation uses the value v1 in the Version field of the prefix. As such, all requests have the prefix shown below.

```
/api/v1/
```

Node Identifier

The node CGA address is used as a node identifier in all requests. The CGA is passed to the controller as a hexadecimal string 32 bytes long.

Southbound API for Agent

The Southbound API allows the agent to perform the operations shown in the table below.

Operation	URL	Description
Host Management		
GET	/host/0/init	Get initial config from cntlr
PUT	/host/<nodeid>	Register host on cntlr
POST	/host/<nodeid>	Send keepalive msg to cntlr
DELETE	/host/<nodeid>	Unregister host on cntlr
Connection Management		
PUT	</host/<nodeid>/conn	Register connection on cntlr
Endpoint Management		
PUT	/host/<nodeid>/service	Register svc endpoint cntlr

GET /host/0/init

Method returns initial configuration for the node, for example

```
{
    "auth_login_url": "https://1.2.3.4/identity/api/login/",
    "auth_openid_url": "https://1.2.3.4/identity/openid/",
    "auth_params" {
        "nonce": "n-0S6_WzA2Mj",
        "state": "af0ifjsldkj",
        "redirect_uri": "https://1.2.3.4/login_callback/",
        "response_type": "id_token token",
        "client_id": "430213",
        "scope": "openid profile domain roles provider_id"
    }
}
```

After receiving this information, the node sends the registration request to **auth_url** (Identity Service) providing the username, password and domain alongside with the **auth_params**.

PUT /host/<nodeid>

Request Structure

An example of this request is shown below.

```
{
    "cga_params": "<hex_string>",
    "cga_sign": "<hex_string>",
    "cfg": {
        "ip":"8.8.8.10",
        "hostname": "testhostname",
        "noderole": 0,
        "ports": [
            {"number": 1, "name": "eth0", "hwaddr": "aabbccddeeff", "admin_status": 1},
            {"number": 2, "name": "eth1", "hwaddr": "ffeeddccbbaa", "admin_status": 2},
        ]
    },
    "cfg_hash": "abc123456789",
    "net": {},
    "net_hash": "9876654321abc"
}
```

cga_params

Parameter **cga_params** is a hexadecimal string of concatenated octets as shown below.

Name	Description	Length (b)
Modifier	used in CGA address gen	128
Collision Count	used in CGA address gen	8
Public Key	RSA public key in DER format	variable

cga_sign

Controller must validate CGA Sign with Public Key. Signed string has the format

Name	Description	Length (b)
CGA Address		128
CGA Params	includes Public Key	200+

cga_hash

Parameter **cfg_hash** is a MD5 hash calculated by the agent. It is calculated on the serialized string comprising **IP address, hostname, noderole, ports**. For example

```
"ip":"8.8.8.10",
"hostname": "testhostname",
"noderole": 0,
"ports": [
    {"number": 1, "name": "eth0", "hwaddr": "aabbccddeeff", "admin_status": 1},
```

(continues on next page)

(continued from previous page)

```
    {"number": 2, "name": "eth1", "hwaddr": "ffeeddccbbaa", "admin_status": 2}
]
```

net_hash

Parameter **net_hash** is a MD5 hash of the object **net** described later.

Response

The response HTTP codes to this request are the following:

- **HTTP 200** - host registered successfully
- **HTTP 401** - unauthorized (token is not valid)
- **HTTP 403** - forbidden (user has no permissions on the requested node)
- **HTTP 404** - node not found (not registered)

An example of the response on a successful registration (**HTTP 200**) is shown below.

```
{
    "host_id": "123456789abc",
    "lldp_key": "dfasadfadsf",
    "keepalive_period": 600
}
```

POST /host/<nodeid>

Keep-alive Request

This request is used to send keep-alive message or update some parameters.

A request example is shown below.

```
{
    "cfg_hash": "abc123456789",
    "net_hash": "123456789abc"
}
```

Keep-alive Response

The response HTTP codes to this request are the following:

- **HTTP 200** - processed successfully (no command present)
- **HTTP 202** - accepted, but the processing has not been completed (a command present)
- **HTTP 401** - unauthorized (token is not valid)
- **HTTP 403** - forbidden (user has no permissions on the requested node)
- **HTTP 404** - node not found (not registered)

An example of the response on a successful keep-alive update (**HTTP 202**) is shown below.

```
{
    "host_id": "123456789abc",
    "keepalive_period": 600,
    "cfg_refresh": 1,
    "net_refresh": 1
}
```

Configuration Management Request

When the parameter **cfg_refresh** or **net_refresh** has the value of 1 in the response, the host repeats the keep-alive request but, this time, sending the block of requested configuration. For example

```
{
    "cfg": {
        "ip":"8.8.8.10",
        "hostname": "testhostname",
        "ports": [
            {"number": 1, "name": "eth0", "hwaddr": "aabbccddeeff", "admin_status": 1},
            {"number": 2, "name": "eth1", "hwaddr": "ffeeddccbbaa", "admin_status": 2}
        ]
    },
    "cfg_hash": "abc123456789",
    "net": [
        "conn": [
            {"local_conn": 12, "local_port": 8, "remote_cga", "<hex_string>", "status":␣
↪1},
            {...}
        ]
    ],
    "net_hash": "123456789abc"
}
```

Configuration Management Response

An example of the response on a successful keep-alive update (**HTTP 200**), that does not require any new information from the host, is shown below.

```
{
    "host_id": "123456789abc",
    "keepalive_period": 600,
    "cfg_refresh": 0,
    "net_refresh": 0
}
```

DELETE /host/<nodeid>

Request

This request calls on for host unregistration. The request has no body.

Response

The response HTTP codes to this request are the following:

- **HTTP 200** - processed successfully
- **HTTP 401** - unauthorized (token is not valid)
- **HTTP 403** - forbidden (user has no permissions on the requested node)
- **HTTP 404** - node not found (not registered)

PUT /host/<nodeid>/conn

This is a request for link registration.

Content of the request and response to it depends on link registration state: **preauth** or **auth**.

Preauth Request

An example of the request in the Preauth state is shown below.

```
{
    "stage": "preauth",
    "nonce": "112233445566",
    "remote_cga": "<hex_string>",
    "local_port" 8,
    "local_conn": 12
}
```

Where

- **stage** - link establishment phase (can be “preauth” or “auth”)
- **nonce** - Nonce value from the SeND packet (in hex string format)
- **remote_cga** - CGA IP Address of remote node (in hex string format)
- **local_port** - local port identifier (integer)
- **local_conn** - local connection identifier (integer)

Preauth Response

The response in a normal case is **HTTP 200** with empty body.

All the possible responses to this request are:

- **HTTP 200** - processed successfully
- **HTTP 401** - unauthorized (token is not valid)
- **HTTP 403** - forbidden (user has no permissions on the requested node)

- **HTTP 404** - node not found (not registered)
- **HTTP 406** - connection validation failed (e.g., invalid nonce, local port not found, remote_cga not registered)

Auth Request

An example of the request in the Auth state is shown below.

```
{
    "stage": "auth",
    "nonce": "112233445566",
    "remote_cga": "<hex_string>",
    "local_port": 15
}
```

Auth Response

An example of the response on a successful Auth request (**HTTP 200**) is shown below.

```
{
    "success": true,
    "data": {
        "nonce": "112233445566",
        "remote_port": 33,
        "remote_portname" "eth1",
        "remote_conn": 122,
        "remote_node_role": "host",
        "remote_domain": "domainA"
    }
}
```

All the possible responses to this request are:

- **HTTP 200** - processed successfully
- **HTTP 401** - unauthorized (token is not valid)
- **HTTP 403** - forbidden (user has no permissions on the requested node)
- **HTTP 404** - node not found (not registered)
- **HTTP 406** - connection validation failed (e.g., invalid nonce, local port not found, remote_cga not registered)

Southbound API for Engine

The Southbound API allows switches to perform the operations shown in the table below.

Operation	URL	Description
Host Management		
GET	/switch/0/init	Get initial config from cntlr
PUT	/switch/<nodeid>	Register node on cntlr
POST	/switch/<nodeid>	Send node keepalive msg to cntlr
DELETE	/switch/<nodeid>	Unregister node on cntlr
Connection Management		
PUT	/switch/<nodeid>/conn	register connection on cntlr

GET /switch/0/init

TBD

PUT /switch/<nodeid>

Request

An example of switch registration request is shown below.

```
{
    "cfg": {
        "ip":"8.8.8.10",
        "hostname": "testhostname",
        "noderole": 0,
        "ports": [
             {"dp": "<datapath_id>",
              "ports": [
                   {"number": 1, "name": "eth0", "hwaddr": "aabbccddeeff", "admin_
↪status": 1},
                   {"number": 2, "name": "eth1", "hwaddr": "ffeeddccbbaa", "admin_
↪status": 2}
                ]
             }
       ]
    },
    "cfg_hash": "abc123456789",
    "net": {},
    "net_hash": "9876654321abc"
}
```

cfg_hash

Parameter **cfg_hash** is a MD5 hash calculated by the supervisor. It is calculated on the serialized string comprising **IP address, hostname, noderole, ports**. For example

```
"ip":"8.8.8.10",
"switchname": "testswitchname",
"noderole": 0,
```

(continues on next page)

(continued from previous page)

```
"ports": [
    {"dp": "<datapath_id>",
     "ports": [
            {"number": 1, "name": "eth0", "hwaddr": "aabbccddeeff", "admin_status": 1},
            {"number": 2, "name": "eth1", "hwaddr": "ffeeddccbbaa", "admin_status": 2}
        ]
    }
]
```

net_hash

Parameter **net_hash** is a MD5 hash of the object **net** described later.

Response

The response HTTP codes to this request are the following:

- **HTTP 200** - node registered successfully
- **HTTP 401** - unauthorized (token is not valid)
- **HTTP 403** - forbidden (user has no permissions on the requested node)
- **HTTP 404** - node not found (not registered)

An example of the response on a successful registration (**HTTP 200**) is shown below.

CHAPTER 33

Release Notes

33.1 Platform Version 1.3 (Nov, 2019)

33.1.1 Fabric Manager

- Workload images on RHEL 8 added to GCP, AWS, Azure
- Telemetry node deployment simplified
- Statistics per each service endpoint displayed in Grafana
- Public git repository with fabric manager resource templates

33.1.2 Orchestrator

- New service graph management commands in BWCTL-API CLI
- Centralized management of link labels for advanced service connectivity policy
- Multiple processors per zone for redundancy and load distribution
- Contract templates are available for direct upload from SDK
- Web-SDK enhanced with resource graphs and automatic tests of contract templates

33.1.3 Processor

- Reduced network protocol overhead

33.1.4 Workload

- Policy agent resolver interface updated for better support of client-side load balancers

- RHEL 8 support
- Libreswan support
- Policy agent deployed in pod on Kubernetes worker node

33.2 Platform Version 1.2 (Sep, 2019)

33.2.1 Fabric Manager

- Fabric manager access to workload nodes additionally secured by processor nodes
- Single sign-on required to access Grafana and Kibana
- All sflow telemetry enriched with service names
- All sflow data encrypted
- Fabric manager Terraform plans and Ansible playbooks open to the public

33.2.2 Orchestrator

- Egress protocol filtering rules generated automatically from opposite-role ingress rules
- Simplified policy data model for fully automatic resource management
- Certificate-based node authentication by orchestrator is mandatory
- CA-signed Flow-Sign certificate is mandatory
- Status of orchestrator certificates displayed on new info page

33.2.3 Processor

- Improved policy execution performance

33.2.4 Workload

- Stateful firewall functionality added to TCP/UDP protocol filtering in eBPF
- Improved performance of agent lib-nss and DNS resolver

33.3 Platform Version 1.1 (Jul, 2019)

33.3.1 Fabric Manager

- Orchestrator, processor and workload nodes automatically placed in three separate subnets
- New SG rules allowed inter-VPC IPsec traffic between processors only
- New SG rules allowed intra-VPC IPsec workload connection to processor only
- CA-signed certificate for orchestrator southbound interface automatically installed
- IPsec events from workloads and processors pushed to orchestrator

- Compatible versions of platform components isolated within a family

33.3.2 Orchestrator

- Multiple processors and locations per availability zone supported
- Location-based automatic workload attachment replaced address-based link configuration
- Southbound interface decoupled from northbound
- mTLS is mandatory for all agent and engine communication with controller
- CA-signed certificate for each node is mandatory

33.3.3 Processor

- Improved performance of engine-OVS control channel
- Improved IPsec establishment time
- Improved virtual interface management

33.3.4 Workload

- Service authorization tokens stored in Kubernetes secrets
- Bayware CNI-plugin interoperability with Kubernetes bridge, Calico and Cilium CNIs added
- Port mirroring option added to contract role settings
- Policy agent graceful restart introduced

33.4 Platform Version 1.0 (May, 2019)

33.4.1 Fabric Manager

- Fabric Manager introduced
- Basic Root CA functionality for automatic node certificate mgmt added
- The BWCTL command line tool for vpc, VM and component mgmt introduced
- The BWCTL-API command line tool for app's communication policy mgmt introduced
- Images for FM, orchestrator, processor and workload published in AWS, Azure, GCP

33.4.2 Orchestrator

- Service type graph enhanced with service instance representation
- Service endpoint, network endpoint and service token added to the data model
- Unified RESTful API for third-party automation systems, BWCTL-API CLI and GUI introduced
- All orchestrator components containerized

33.4.3 Processor

- Handshake between opposite-role instances required for creating network microsegment
- Packet path cost evaluation added
- sFlow telemetry uploaded

33.4.4 Workload

- Service authorization tokens supported
- Automatic discovery of the opposite-role instances introduced
- Instance affinity option added to name resolution
- Local DNS server for containers and resolver library for VMs supported
- Kubernetes support added
- All data packet processing moved from user space dataplane to eBPF
- Debian/Ubuntu 18.04 LTS required

CHAPTER 34

Further Reading

1. RISC-V is a free and open ISA enabling a new era of processor innovation
2. BPF at Cilium - execute bytecode in the Linux kernel
3. Grafana - an open platform for analytics and monitoring
4. Open vSwitch - production quality, multilayer open virtual switch
5. strongSwan - an open source, IPsec-based VPN solution
6. Kibana - visualize your Elasticsearch data
7. Security Architcture for IP RFC4301
8. IP Encapsulating Security Payload RFC4303
9. Network Service Header (NSH) RFC8300
10. DevOps, meet NetOps and SecOps. Network computing. March 2017.
11. Cluster Networking. Kubernetes concepts. June 2018.
12. The Ideal Network for Containers and NFV Microservices. Mellanox blog. June 2017.
13. What is microsegmentation? How getting granular improves network security. Network World. January 2018.
14. Active networking: one view of the past, present, and future. Jonathan M. Smith, Scott M. Nettles. University of Pennsylvania. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) (Volume: 34, Issue: 1), February 2004.
15. Segment routing, a source-routing architecture.
16. Network service header. IETF RFC 8300. January 2018.
17. NSH and Segment Routing Integration for Service Function Chaining. IETF draft. March 2018.
18. LISP Control plane integration with NSH. IETF draft. March 2018.
19. Apache Groovy: A multi-faceted language for the Java platform.
20. Erlang programming language: Build massively scalable soft real-time systems.

21. Kubernetes blog. Using eBPF in Kubernetes.

CHAPTER 35

Glossary

Network Datapath Orchestration Continuous adjustment of network forwarding behavior in unison with upper layer flux

Ephemeral Network Function (ENF) Usually a short-lived network task that steers or modifies packets thereby implementing a particular communication role. Both the preparation and the execution of the datapath action set, which produces the network forwarding behavior, comprise the ENF.

Network Microservice The combined behavior of one or more ENFs realizing a communication pattern

Network Microservice Orchestrator (NMO) A vertically distributed, highly-available system that arranges and coordinates automated tasks resulting in network microservice establishment, maintenance, and termination. May scale out as demand requires.

Network Microservice Processor (NMP) Functional block responsible for ENF action set preparation at the switch; paired with switch datapath, which provides ENF action set execution

Network Microservice Agent (NMA) Functional block responsible for ENF endpoint management at the host; controls host datapath in Linux kernel

Network Microservice SDK Toolset for creating microcode to implement network microservices in either Java- or Python-like syntax

Domain Domain is a collection of Users, Resources, and Contracts. The visibility of a set of Users, Resources, and Contracts are limited by the domain boundaries. Domain serves as a logical division between different portions of the system.

User User is an entity that receives access to the resources that are isolated in the domain. There are two types of users: admin users and resource users. Admin user is an Orchestrator's northbound API entity for administrator or external system. Resource user is an Orchestrator's southbound API entity for switch or host.

Template Generic implementation of a communication pattern comprised of ENFs; also, Network Microservice Template

Resource Either switch datapath or host datapath; used interchangeably with node;

Contract Customized implementation of a communication pattern comprised of ENFs; also, Network Microservice Instance or topic

Flow Series of packets with a common set of identifiers e.g., host, contract and flow label

Northbound API Orchestrator's externally-facing API for management by service orchestrator

Southbound API Orchestrator's internally-facing API for host and switch operation

Agent API Optional API to business logic running on host for more granular control over datapath

CHAPTER 36

Indices and tables

- genindex
- modindex
- search