Oracle Cloud Native Environment

Service Mesh for Release 1.6

F75588-02
June 2023

Oracle Cloud Native Environment Service Mesh for Release 1.6,

F75588-02

Contents

Preface

This document contains information about the Istio module for Oracle Cloud Native Environment to set up a service mesh. It describes the differences from the upstream version, and includes information on installing and using the Istio module.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry

standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Introduction to the Service Mesh

This chapter provides introductory information about the Istio module for Oracle Cloud Native Environment, which is used to set up a service mesh in Oracle Cloud Native Environment.

What is a Service Mesh?

A *service mesh* is a configurable, low-latency infrastructure layer that controls the interaction between a network of microservices. A service mesh makes sure communication among containerized application infrastructure services is fast, reliable, and secure. The service mesh provides critical capabilities including service discovery, load balancing, encryption, observability, traceability, and authentication and authorization.

A service mesh provides the ability to monitor the microservices in the Kubernetes cluster. Istio can support most of the popular current deployment patterns for deploying microservices. This is transparent to a developer.

What is Istio?

Istio is a type of service mesh designed to manage the interaction and operation of services in a microservices architecture. Istio is an open source project that coordinates communication between services, providing service discovery, load balancing, security, recovery, telemetry, and policy enforcement capabilities. Istio uses a *sidecar* service mesh model. This means that network communication proxy capabilities are implemented in a separate container for every service or application container that is deployed. Envoy is the product that implements this proxy capability and these special containers run alongside every other container. The Istio sidecar service mesh frees developers from having to program these types of capabilities into application code and makes development and enhancement of applications in a microservice architecture much more efficient and rapid.

Istio's control plane provides an abstraction layer over the underlying cluster management platform, Kubernetes.

Istio contains the following components:

- **Envoy**: Sidecar proxies per microservice to handle ingress/egress traffic between services in the cluster and from a service to external services. The proxies form a secure microservice mesh providing a rich set of functions like discovery, rich layer-7 routing, policy enforcement and telemetry recording/reporting functions.
- **Istiod**: A component responsible for service discovery, configuration and certificate management.

For more information on the Istio deployment architecture, see the upstream documentation at:

https://istio.io/latest/docs/ops/deployment/architecture/

About the Istio Module

The Istio module is based on a stable release of the upstream Istio project. Differences between Oracle versions of the software and upstream releases are limited to Oracle provided configuration profiles and patches for specific bugs.

For upstream Istio documentation, see https://istio.io/latest/docs/.

For more information about Istio, see https://istio.io/.

Istio Module Components

The upstream Istio installation has a number of configuration profiles you can choose from. The Istio module components are based on the upstream installation configuration profiles, and includes components curated for Oracle Cloud Native Environment. You can see the upstream installation configuration profiles at:

https://istio.io/latest/docs/setup/additional-setup/config-profiles/

The core Istio components installed with their corresponding container name prefix are:

- Egress gateway (`istio-egressgateway`)
- Ingress gateway (`istio-ingressgateway`)
- Istiod (`istiod`)

Two additional modules are installed as supporting modules for monitoring and visualization of the Kubernetes cluster. These are:

- Grafana (`grafana`)
- Prometheus (`prometheus-server`)

2

Setting up a Service Mesh

This chapter discusses how to install the Istio module to set up a service mesh, and the components deployed when you do this.

The Istio module is installed using a mostly empty profile. The default Istio module profile contains the profile name, the container image hub, and container image tags. If you want to customize an Istio module installation, you can use a custom Istio profile. This allows you to set Kubernetes resource settings, enable or disable individual Istio components, and configure their settings.

To customize these components, you write a YAML configuration file for these settings, and use it when you deploy an Istio module. You can deploy multiple Istio modules, with different configurations, all using the same Istio control plane.

You can deploy a single custom Istio module using a configuration file, or you can deploy multiple Istio modules. If you want to deploy multiple Istio modules, you should have an initial module set up as the default module, which acts as the Istio control plane. This default module is considered the *parent* Istio module. The parent Istio module is installed using the default profile.

Creating a Configuration File

To install a customized Istio module, you need to write a YAML configuration file to specify the configuration options. You use the `spec` section of an IstioOperator resource file to set the configuration. For information on the options available to use in the configuration file, see the upstream documentation for the IstioOperator resource, at:

https://istio.io/latest/docs/reference/config/istio.operator.v1alpha1/

Do not include a full IstioOperator file in the configuration file, only use the options available below the `spec` section. That is, do **not** include the following lines in your configuration file:

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
```

The configuration file should include the IstioOperator configuration you want to set, for example:

```
profile: myistio
hub: myhub.example
components:
  egressGateways:
  - name: istio-egressgateway
    enabled: true
```

Or you could provide just the IstioOperator `components` you want to customize, for example:

```
components:
  egressGateways:
  - name: istio-egressgateway
    enabled: true
```

The YAML configuration file is used with the `olcnectl module create` command when you create the Istio module(s).

Example 2-1 Simple configuration file to set up a load balancer for the Istio ingress gateway

This example Istio configuration file uses the Oracle Cloud Infrastructure Cloud Controller Manager module to provision an Oracle Cloud Infrastructure load balancer for the Istio ingress gateway by applying the appropriate annotations to the `istio-ingressgateway` service to set this up.

> **Note:**
>
> To try this example, you must have the Oracle Cloud Infrastructure Cloud Controller Manager module installed.

The YAML configuration file contains:

```
components:
  ingressGateways:
  - name: istio-ingressgateway
    k8s:
      serviceAnnotations:
        service.beta.kubernetes.io/oci-load-balancer-security-list-
management-mode: "None"
        service.beta.kubernetes.io/oci-load-balancer-internal: "true"
        service.beta.kubernetes.io/oci-load-balancer-shape: "flexible"
        service.beta.kubernetes.io/oci-load-balancer-shape-flex-min:
"10"
        service.beta.kubernetes.io/oci-load-balancer-shape-flex-max:
"10"
```

For the full list of Oracle Cloud Infrastructure Cloud Controller Manager annotations you can include, see the upstream documentation at:

https://github.com/oracle/oci-cloud-controller-manager/blob/master/docs/load-balancer-annotations.md

After you deploy the Istio module using this configuration file, you would see the following Kubernetes services deployed to the `istio-system` namespace. On a control plane node, show the services in the `istio-system` namespace.

```
kubectl --namespace istio-system get svc
```

The output should look similar to:

```
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)             ...
grafana                 ClusterIP      10.97.12.24      <none>
3000/TCP            ...
istio-egressgateway     ClusterIP      10.106.217.129   <none>            80/
TCP,443/TCP,1544...
istio-ingressgateway    LoadBalancer   10.103.9.119     100.102.106.171
15021:30762/TCP,80:...
istiod                  ClusterIP      10.106.101.205   <none>
15010/TCP,15012/TCP...
prometheus-server       ClusterIP      10.107.228.56    <none>
9090/TCP            ...
```

You can see the `istio-ingressgateway` service is of type LoadBalancer and has an externalIP associated with it.

Example 2-2 Detailed configuration file to set up a load balancer for the Istio ingress gateway

This example configuration file creates an Istio module with a profile that creates an Istio ingress gateway named `my-istio-ingressgateway` in a namespace named `myistio`. This example also uses the Oracle Cloud Infrastructure Cloud Controller Manager module to provision an Oracle Cloud Infrastructure load balancer for the Istio ingress gateway, and includes more detail on how to configure the gateway.

> **Note:**
>
> To try this example, you must have the Oracle Cloud Infrastructure Cloud Controller Manager module installed.

The YAML configuration file contains:

```
components:
  ingressGateways:
  - enabled: true
    k8s:
      hpaSpec:
        maxReplicas: 5
        minReplicas: 2
        scaleTargetRef:
          apiVersion: apps/v1
          kind: Deployment
          name: my-istio-ingressgateway
      resources:
        limits:
          cpu: 2000m
          memory: 1024Mi
        requests:
          cpu: 100m
          memory: 128Mi
```

```
      serviceAnnotations:
        service.beta.kubernetes.io/oci-load-balancer-security-list-
management-mode: "None"
        service.beta.kubernetes.io/oci-load-balancer-internal: "true"
        service.beta.kubernetes.io/oci-load-balancer-shape: "flexible"
        service.beta.kubernetes.io/oci-load-balancer-shape-flex-min:
"10"
        service.beta.kubernetes.io/oci-load-balancer-shape-flex-max:
"10"
      service:
        ports:
        - name: status-port
          port: 15021
          protocol: TCP
          targetPort: 15021
        - name: http2
          port: 80
          protocol: TCP
          targetPort: 8080
        - name: https
          port: 443
          protocol: TCP
          targetPort: 8443
        - name: tcp-istiod
          port: 15012
          protocol: TCP
          targetPort: 15012
        - name: tls
          port: 15443
          protocol: TCP
          targetPort: 15443
      strategy:
        rollingUpdate:
          maxSurge: 100%
          maxUnavailable: 25%
    name: my-istio-ingressgateway
    namespace: myistio
values:
  gateways:
    istio-ingressgateway:
      autoscaleEnabled: true
      env: {}
      name: istio-ingressgateway
      secretVolumes:
      - mountPath: /etc/istio/ingressgateway-certs
        name: ingressgateway-certs
        secretName: istio-ingressgateway-certs
      - mountPath: /etc/istio/ingressgateway-ca-certs
        name: ingressgateway-ca-certs
        secretName: istio-ingressgateway-ca-certs
      type: LoadBalancer
```

In this example, the Istio ingress gateway named `my-istio-ingressgateway` is in a namespace named `myistio`. This namespace is not the default Istio namespace of

`istio-system`. If you are installing the gateway service into a non-default namespace, as shown in this example, you must first create the namespace. Create the new namespace with the `kubectl create namespace` command on a control plane node. For example, on a control plane node:

```
kubectl create namespace myistio
```

After you deploy the Istio module using this configuration file, you would see the following Kubernetes services deployed to the default `istio-system` namespace. On a control plane node, show the services in the `istio-system` namespace.

```
kubectl --namespace istio-system get svc
```

The output should look similar to:

```
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)                ...
grafana                ClusterIP      10.103.216.188   <none>
3000/TCP               ...
istio-egressgateway    ClusterIP      10.111.113.121   <none>        80/
TCP,443/TCP,15443/TC...
istio-ingressgateway   LoadBalancer   10.106.116.57    <pending>
15021:30210/TCP,80:3193...
istiod                 ClusterIP      10.99.54.66      <none>        15010/
TCP,15012/TCP,443...
prometheus-server      ClusterIP      10.110.20.110    <none>
9090/TCP               ...
```

The `istio-ingressgateway` service is of type LoadBalancer and has no externalIP associated with it (it is in the `pending` state). This is the default service that is deployed, and is set up using the default Istio configuration.

To show the ingress gateway service named `my-isitio-ingressgateway`, get the services running in the `myistio` namespace. On a control plane node, show the services in the `myistio` namespace.

```
kubectl --namespace myistio get svc
```

The output should look similar to:

```
NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP
PORT(S)              ...
my-istio-ingressgateway   LoadBalancer   10.96.174.73   100.102.107.82
15021:30292/TCP,80:3...
```

You can see the `my-istio-ingressgateway` service is of type LoadBalancer and has an externalIP associated with it. This is the service created using the configuration file.

Deploying the Istio Module

You can deploy all the modules required to create a service mesh and a Kubernetes cluster using a single `olcnectl module create` command. This method might be useful if you want to deploy a service mesh at the same time as deploying a Kubernetes cluster.

If you have an existing deployment of the Kubernetes module, you can specify that instance when deploying a service mesh.

You can create a custom Istio module using a configuration file. This allows you to set Kubernetes resource settings, enable or disable individual Istio components, and configure their settings. For information on creating custom profiles for Istio modules, see Creating a Configuration File.

If you are installing multiple custom Istio modules using configuration files, see Deploying Multiple Custom Istio Modules.

For the full list of the options available when creating modules, see the `olcnectl module create` command in Platform Command-Line Interface.

> **Note:**
>
> The Istio module installs two other modules, the Prometheus module and the Grafana module. These two modules install Prometheus and Grafana respectively to enable monitoring and visualization of the Kubernetes cluster. You do not need to provide any information for these modules when you install the Istio module.

To deploy an Istio module:

1. If you do not already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Getting Started. The name of the environment in this example is `myenvironment`.
2. If you do not already have a Kubernetes module set up or deployed, set one up. For information on adding a Kubernetes module to an environment, see Container Orchestration. The name of the Kubernetes module in this example is `mycluster`.
3. Create an Istio module and associate it with the Kubernetes module named `mycluster` using the `--istio-kubernetes-module` option. In this example, the Istio module is named `myistio`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module istio \
   --name myistio \
   --istio-kubernetes-module mycluster
   ```

   The `--module` option sets the module type to create, which is `istio`. You define the name of the Istio module using the `--name` option, which in this case is `myistio`.

As the Istio module requires Kubernetes, you must also provide the option for that module.

The `--istio-kubernetes-module` option sets the name of the Kubernetes module to use. The Kubernetes module should already be set up or deployed. If you have an existing Kubernetes module deployed, you can specify the name of the module using this option. If no Kubernetes module is deployed with the name you provide, a new module is deployed which allows you to deploy Kubernetes at the same time as a service mesh.

If you are installing an Istio module using a custom profile, include the `--istio-profile` option to specify the location of the YAML configuration file. The Platform API Server configures the Istio module using the settings in the configuration file.

If you do not include all the required options when adding the modules you are prompted to provide them.

4. Use the `olcnectl module install` command to install the Istio module. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name myistio
```

The Istio software packages are installed on the control plane nodes, and the Istio module is deployed into the Kubernetes cluster.

Deploying Multiple Custom Istio Modules

If you want to deploy multiple Istio modules, you should create an Istio module as the *parent* module with a default profile. This creates a single Istio control plane to manage the custom Istio modules. You can do this by deploying an Istio module without a profile configuration file. You then deploy any further Istio modules with their respective profile configuration files and set the parent module using the `--istio-parent` option.

To deploy multiple Istio modules:

1. Follow the steps in Deploying the Istio Module to set up a default Istio module to act as the parent module. Do not include a custom profile configuration file when you create the Istio module.
2. Create a second Istio module with a YAML configuration file. Use the `olcnectl module create` command to create the module.

```
olcnectl module create \
--environment-name myenvironment \
--module istio \
--name mycustomistio \
--istio-kubernetes-module mycluster \
--istio-parent myistio \
--istio-profile mycustomistio.yaml
```

The `--name` option sets the name of this second Istio module. In this example it is set to `mycustomistio`.

The `--istio-parent` option sets the name of the parent Istio module. In this example, the parent Istio module is named `myistio`, which is also the name of the Istio module used in the example in Deploying the Istio Module.

The `--istio-profile` option sets the location of the YAML configuration file.

3. Install the Istio module, using the `olcnectl module install` command. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name mycustomistio
```

4. To add more custom Istio modules to the parent Istio control plane, create more Istio modules, using different module names, configuration files, and specify the parent module.

Verifying the Istio Module Deployment

You can verify the Istio module is deployed and the required containers are running in the `istio-system` namespace. To verify the containers are deployed, you need to use the `kubectl` command. For information on setting up the `kubectl` command, see Container Orchestration.

To verify the required containers are running, on a control plan node, list the containers running in the `istio-system` namespace. You should see similar results to those shown here:

```
kubectl get deployment -n istio-system
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
grafana                2/2     2            2           2m44s
istio-egressgateway    2/2     2            2           2m48s
istio-ingressgateway   2/2     2            2           2m48s
istiod                 2/2     2            2           3m2s
prometheus-server      2/2     2            2           2m44s
```

You can also review information about the Istio module and its properties.

On the operator node, use the `olcnectl module report` command to review information about the module. For example, use the following command to review the Istio module named `myistio` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name myistio \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

Removing the Istio Module

You can remove a deployment of a service mesh and leave the Kubernetes cluster in place. To do this, you remove the Istio module from the environment.

Use the `olcnectl module uninstall` command to remove the Istio module. For example, to uninstall the Istio module named `myistio` in the environment named `myenvironment`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name myistio
```

The Istio module and its supporting Prometheus and Grafana modules are removed from the environment.

You can confirm the Istio, Prometheus and Grafana modules are removed using the `olcnectl module instances` command. These three modules are no longer listed as modules in the environment.

You can also confirm the Istio components are removed using the `kubectl` command on a control plane node to query all deployments running in the `istio-system` namespace. You should see there are no deployments returned.

```
kubectl get deployment -n istio-system
No resources found in istio-system namespace.
```

3
Using a Service Mesh

Istio automatically populates its service registry with all services you create in the service mesh, so it knows all possible service endpoints. By default, the Envoy proxy sidecars manage traffic by sending requests to each service instance in turn in a round-robin fashion. You can configure the management of this traffic to suit your own application requirements using the Istio traffic management APIs. The APIs are accessed using Kubernetes custom resource definitions (CRDs), which you set up and deploy using YAML files.

The Istio API traffic management features available are:

- **Virtual services**: Configure request routing to services within the service mesh. Each virtual service can contain a series of routing rules, that are evaluated in order.
- **Destination rules**: Configures the destination of routing rules within a virtual service. Destination rules are evaluated and actioned after the virtual service routing rules. For example, routing traffic to a particular version of a service.
- **Gateways**: Configure inbound and outbound traffic for services in the mesh. Gateways are configured as standalone Envoy proxies, running at the edge of the mesh. An ingress and an egress gateway are deployed automatically when you install the Istio module.
- **Service entries**: Configure services outside the service mesh in the Istio service registry. Allows you to manage the traffic to services as if they are in the service mesh. Services in the mesh are automatically added to the service registry, and service entries allow you to bring in outside services.
- **Sidecars**: Configure sidecar proxies to set the ports, protocols and services to which a microservice can connect.

These Istio traffic management APIs are well documented in the upstream documentation at:

https://istio.io/latest/docs/concepts/traffic-management/

Enabling Proxy Sidecars

Istio enables network communication between services to be abstracted from the services themselves and to instead be handled by proxies. Istio uses a sidecar design, which means that communication proxies run in their own containers alongside every service container.

To enable the use of a service mesh in your Kubernetes applications, you need to enable automatic proxy sidecar injection. This injects proxy sidecar containers into pods you create.

To put automatic sidecar injection into effect, the namespace to be used by an application must be labeled with `istio-injection=enabled`. For example, to enable automatic sidecar injection for the `default` namespace:

```
kubectl label namespace default istio-injection=enabled
namespace/default labeled

kubectl get namespace -L istio-injection
NAME                          STATUS   AGE   ISTIO-INJECTION
default                       Active   29h   enabled
externalip-validation-system  Active   29h
```

```
istio-system                   Active   29h
kube-node-lease                Active   29h
kube-public                    Active   29h
kube-system                    Active   29h
kubernetes-dashboard           Active   29h
```

Any applications deployed into the default namespace have automatic sidecar injection enabled and the sidecar runs alongside the pod. For example, create a simple NGINX deployment:

```
kubectl create deployment --image nginx hello-world
deployment.apps/hello-world created
```

Show the details of the pod to see that an `istio-proxy` container is also deployed with the application:

```
kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-world-5fcdb6bc85-wph7h   2/2     Running   0          7m40s

kubectl describe pods hello-world-5fcdb6bc85-wph7h
...
  Normal  Started    13s   kubelet, worker1.example.com  Started container nginx
  Normal  Started    12s   kubelet, worker1.example.com  Started container istio-
proxy
```

Setting up a Load Balancer for an Ingress Gateway

If you are deploying the Istio module, you may also want to set up a load balancer to handle the Istio ingress gateway traffic. The information in this section shows you how to set up a load balancer to manage access to services from outside the cluster using the Istio ingress gateway.

The load balancer port mapping in this section sets ports for HTTP and HTTPS. That is, the load balancer listens for HTTP traffic on port `80` and redirects it to the Istio ingress gateway NodePort number for `http2`. You query the port number to set for `http2` by entering the following on a control plane node:

```
kubectl describe svc istio-ingressgateway -n istio-system |grep http2
Port:                     http2  80/TCP
NodePort:                 http2  32681/TCP
```

In this example, the NodePort is `32681`. So the load balancer must be configured to listen for HTTP traffic on port `80` and redirect it to the `istio-ingressgateway` service on port `32681`.

For HTTPS traffic, the load balancer listens on port `443` and redirects it to the Istio ingress gateway NodePort number for `https`. To find the port numbers to set for `https`, enter:

```
kubectl describe svc istio-ingressgateway -n istio-system |grep https
Port:                     https  443/TCP
NodePort:                 https  31941/TCP
```

In this example, the NodePort is `31941`. So the load balancer must be configured to listen for HTTPS traffic on port `443` and redirect it to the `istio-ingressgateway` service on port `31941`.

The load balancer should be set up with the following configuration for HTTP traffic:

- The listener listening on TCP port `80`.
- The distribution set to round robin.
- The target set to the TCP port for `http2` on the worker nodes. In this example it is `32681`.
- The health check set to TCP.

For HTTPS traffic:

- The listener listening on TCP port `443`.
- The distribution set to round robin.
- The target set to the TCP port for `https` on the worker nodes. In this example it is `31941`.
- The health check set to TCP.

For more information on setting up your own load balancer, see Oracle® Linux 8: Setting Up Load Balancing, or Oracle® Linux 7: Administrator's Guide.

If you are deploying to Oracle Cloud Infrastructure, you can either set up a new load balancer or, if you have one, use the load balancer you set up for the Kubernetes module.

To set up a load balancer on Oracle Cloud Infrastructure for HTTP traffic:

1. Add a backend set to the load balancer using weighted round robin.
2. Add the worker nodes to the backend set. Set the port for the worker nodes to the TCP port for `http2`. In this example it is `32681`.
3. Create a listener for the backend set using TCP port `80`.

To set up a load balancer on Oracle Cloud Infrastructure for HTTPS traffic:

1. Add a backend set to the load balancer using weighted round robin.
2. Add the worker nodes to the backend set. Set the port for the worker nodes to the TCP port for `https`. In this example it is `31941`.
3. Create a listener for the backend set using TCP port `443`.

For more information on setting up a load balancer in Oracle Cloud Infrastructure, see the Oracle Cloud Infrastructure documentation.

Setting up an Ingress Gateway

An Istio ingress gateway allows you to define entry points into the service mesh through which all incoming traffic flows. A ingress gateway allows you to manage access to services from outside the cluster. You can monitor and set route rules for the traffic entering the cluster.

This section contains a simple example to configure the automatically created ingress gateway to an NGINX web server application. The example assumes you have a load balancer available at `lb.example.com` and is connecting to the `istio-ingressgateway` service on `TCP` port `32681`. The load balancer listener is set to listen on `HTTP` port `80`, which is the port for the NGINX web server application used in the virtual service in this example.

To set up an ingress gateway:

1. Create the deployment file to create the NGINX web server application. Create a file named `my-nginx.yml`, containing:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-webserver
  name: my-nginx
  namespace: my-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-webserver
  template:
    metadata:
      labels:
        app: my-webserver
    spec:
      containers:
      - image: nginx
        name: my-nginx
        ports:
        - containerPort: 80
```

2. Create a service for the deployment. Create a file named `my-nginx-service.yml` containing:

```
apiVersion: v1
kind: Service
metadata:
  name: my-http-ingress-service
  namespace: my-namespace
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: my-webserver
  type: ClusterIP
```

3. Create an ingress gateway for the service. Create a file named `my-nginx-gateway.yml` containing:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: my-nginx-gateway
  namespace: my-namespace
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
      - "mynginx.example.com"
```

4. Create a virtual service for the ingress gateway. Create a file named `my-nginx-virtualservice.yml` containing:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: my-nginx-virtualservice
  namespace: my-namespace
spec:
  hosts:
  - "mynginx.example.com"
  gateways:
  - my-nginx-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 80
        host: my-http-ingress-service
```

5. Set up a namespace for the application named `my-namespace` and enable automatic proxy sidecar injection.

```
kubectl create namespace my-namespace
kubectl label namespaces my-namespace istio-injection=enabled
```

6. Run the deployment, service, ingress gateway and virtual service:

```
kubectl apply -f my-nginx.yml
kubectl apply -f my-nginx-service.yml
kubectl apply -f my-nginx-gateway.yml
kubectl apply -f my-nginx-virtualservice.yml
```

7. You can see the ingress gateway is running using:

```
kubectl get gateways.networking.istio.io -n my-namespace
NAME               AGE
my-nginx-gateway   33s
```

8. You can see the virtual service is running using:

```
kubectl get virtualservices.networking.istio.io -n my-namespace
NAME                      GATEWAYS             HOSTS                    AGE
my-nginx-virtualservice   [my-nginx-gateway]   [mynginx.example.com]    107s
```

9. To confirm the ingress gateway is serving the application to the load balancer, use:

```
curl -I -HHost:mynginx.example.com lb.example.com:80/
HTTP/1.1 200 OK
Date: Fri, 06 Mar 2020 00:39:16 GMT
Content-Type: text/html
Content-Length: 612
Connection: keep-alive
last-modified: Tue, 03 Mar 2020 14:32:47 GMT
etag: "5e5e6a8f-264"
accept-ranges: bytes
x-envoy-upstream-service-time: 15
```

Setting up an Egress Gateway

The Istio egress gateway allows you to set up access to external HTTP and HTTPS services from applications inside the service mesh. External services are called using the sidecar container.

The Istio egress gateway is deployed automatically. You do not need to manually deploy it. You can confirm the Istio egress gateway service is running using:

```
kubectl get svc istio-egressgateway -n istio-system
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)                   AGE
istio-egressgateway   ClusterIP   10.111.233.121   <none>        80/TCP,443/
TCP,15443/TCP   9m26s
```

The upstream documentation provides an example to show you how to set up use an Istio egress gateway.

https://istio.io/latest/docs/tasks/traffic-management/egress/egress-gateway/

Testing Network Resilience

Istio network resilience and testing features allow you to set up and test failure recovery and to inject faults to test resilience. You set up these features dynamically at runtime to improve the reliability of your applications in the service mesh. The network resilience and testing features available in this release are:

- **Timeouts**: The amount of time that a sidecar proxy should wait for replies from a service. You can set up a virtual service to configure specific timeouts for a service. The default timeout for HTTP requests is 15 seconds.
- **Retries**: The number of retries allowed by the sidecar proxy to connect to a service after an initial connection failure. You can set up a virtual service to enable and configure the number of retries for a service. By default, no retries are allowed.
- **Fault injection**: Set up fault injection mechanisms to test failure recovery of applications. You can set up a virtual service to set up and inject faults into a service. You can set delays to mimic network latency or an overloaded upstream service. You can also set aborts to mimic crashes in an upstream service.

4

Introduction to Monitoring and Visualization

This chapter provides information about the service mesh visualization and monitoring components installed when you deploy the Istio module. The components are:

- **Prometheus**: Prometheus is the time-series database that monitors and gathers metrics about the Kubernetes cluster.
- **Grafana**: Grafana can be used to monitor and visualize the time-series data stored in Prometheus. Grafana enables you to visually query and monitor the network traffic and services in your Kubernetes cluster. Grafana includes browser-based dashboards to visualize the cluster metrics gathered from Prometheus.

> **Note:**
>
> Prometheus and Grafana are automatically configured to provide standard metrics and dashboards for the Istio module. Persisting custom or manual configuration of these components is not possible.

About Grafana and Prometheus

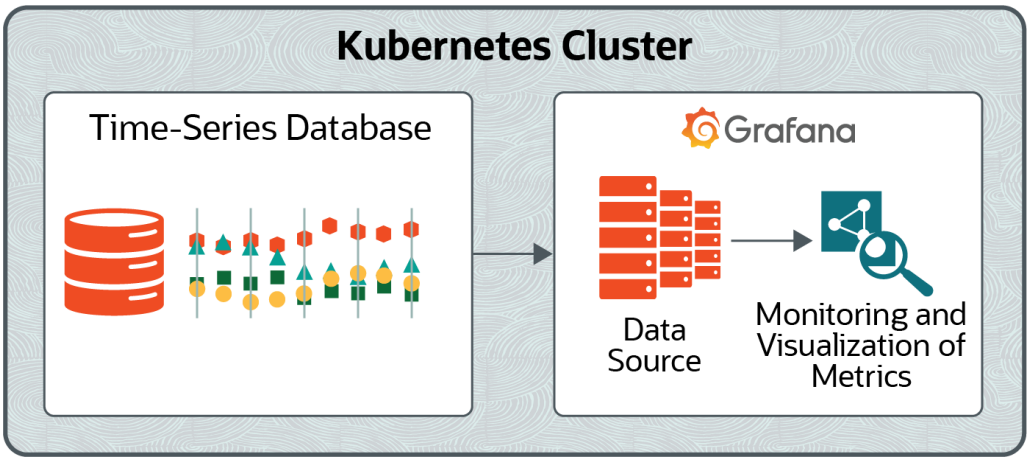
Grafana is an open-source platform for monitoring the performance of your Kubernetes cluster. It contains dashboards that allow you to visualize the real-time metrics of the cluster which are stored in the Prometheus time-series database.

Time-series data is a sequence of values ordered by time. It measures how these values change over time. Examples of time-series data include self-driving cars collecting information about their local environments continually, homes monitoring themselves to regulate temperature or identify intruders, or a police department tracking its vehicles for efficiency purposes.

Grafana defines a data source to integrate with Prometheus. A data source is Grafana's link to this type of database. By using a data source for Prometheus in Grafana, Grafana can retrieve, analyze, monitor, and visualize the metrics that are stored in a Prometheus time-series database.

The following architectural diagram illustrates how Grafana uses the data source to integrate with Prometheus. As a result, Grafana can monitor and visualize the metrics that are stored in the database.

Figure 4-1 Architectural Diagram for Integrating a Time-Series Database (Prometheus) with Grafana



Grafana is based on a stable release of the upstream Grafana project. Differences between Oracle versions of the software and upstream releases are limited to Oracle specific fixes and patches for specific bugs.

For upstream Grafana documentation, see the upstream documentation at:

https://grafana.com/docs/

For more information about Grafana, see the upstream documentation at:

https://grafana.com

Grafana Components

This section contains information about the components in Grafana that are used to monitor and visualize the metrics that are stored in a Prometheus database.

Data Source

The data source is Grafana's link to Prometheus. Grafana contains out-of-the-box support to connect to Prometheus that you can use to monitor and visualize the metrics in your Oracle Cloud Native Environment. Grafana refers to a connection to this type of database as a data source.

Query Editor

Grafana has a query editor that exposes the capabilities of your data source and allows you to query the metrics that it contains. Grafana provides a custom query editor for each data source, including the data source used to integrate Grafana with Prometheus. You can use the query editor to structure queries that allow you to visualize the metrics that are stored in the Prometheus database.

Panel

The panel is the main element in Grafana used to visualize metrics from Prometheus. Each type of panel has its own query editor that allows you to fine-tune the data that you want to visualize.

There are several panel types, including Graph, Singlestat, Table, Text, and Dashboard List. In this book, you use the Graph panel. This is the main panel type in Grafana and it provides a rich set of graphing options.

For more information about the other panel types, see the upstream documentation at:

https://grafana.com/docs/grafana/latest/panels-visualizations/

Dashboard

A dashboard is a grouping of panels prearranged into rows. A row is the divider between panels.

User

A user has an account in Grafana. A user is granted permissions in Grafana based on the following roles:

- **Admin:** An admin has superuser permissions in Grafana, and can do everything, including adding and editing data sources, generating queries for the data sources, and creating and modifying dashboards.
- **Editor:** An editor has limited permissions in Grafana. Although an editor can create and modify dashboards, they cannot create or edit data sources, or generate queries for the data sources.
- **Viewer:** A viewer has read-only access to the components in Grafana. For example, a viewer can view data sources and dashboards, but cannot modify them.

For more information about the Admin, Editor, and Viewer roles, see the upstream documentation at:

https://grafana.com/docs/grafana/latest/administration/roles-and-permissions/

5

Visualizing the Service Mesh Using Grafana

When Grafana is deployed, a data source is configured for a Prometheus time-series database. A data source is Grafana's link to the database. Because this data source is configured, Grafana can retrieve and analyze the metrics of a Kubernetes cluster that are gathered and stored in the Prometheus database.

In this chapter, you learn how to:

- Get the IP address of the node on which Grafana is deployed and the port number that is reserved for Grafana. You need this information to access the Grafana console.
- Access the Grafana console.
- Create a dashboard in Grafana to monitor and visualize the metrics that are retrieved and analyzed from Prometheus through the data source.
- Formulate a query for the dashboard to fine-tune how the metrics for Prometheus appear in Grafana.

Getting the Grafana IP Address and Port Number

In this section, you get the IP address of the node on which Grafana is deployed and the port number for Grafana.

1. At the prompt of the machine where you installed the Istio module, enter the following command:

   ```
   kubectl -n istio-system get svc grafana
   ```

   There is no NodePort set up for the port on which the Grafana service is running. In this example, the port that is reserved for Grafana is `9090`.

   ```
   NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
   grafana      ClusterIP   10.104.245.12   <none>        9090/TCP     6d1h
   ```

   A NodePort is an open port on every node of your Kubernetes cluster. By opening a NodePort for Grafana, Kubernetes transparently routes incoming traffic on the NodePort to the Grafana service. The NodePort is assigned from a pool of cluster-configured NodePort ranges (typically 30000–32767).

2. Enter the following command to set up a NodePort for the Grafana service:

   ```
   kubectl patch svc grafana -n istio-system -p '{"spec":{"type":"NodePort"}}'
   ```

3. Verify that you see the `service/grafana patched` status message.

   You can now connect to Grafana via the NodePort.

4. Enter the following command again:

   ```
   kubectl -n istio-system get svc grafana
   ```

   This time, you can see that a NodePort is set up for the Grafana container. You can use this NodePort to connect to the service.

```
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)       AGE
grafana       ClusterIP   10.104.245.12   <none>        9090/30921    6d1h
```

In this example, the NodePort is `30921`, which maps to the service port of `9090`.

5. Enter the following command to get the IP address of either a control plane node or the IP address of the worker node on which the Grafana container is running:

```
ip addr
```

Make a note of the IP address and NodePort that you obtained in this procedure because you need these values to access the Grafana console.

Accessing the Grafana Console

As part of deploying Grafana, a user account is created, and the Admin role is assigned to the account. This account has superuser permissions in Grafana and can create dashboards and generate queries for the dashboards.

In this section, you access the Grafana console.

1. Open a web browser.
2. In the **Address** field, enter `http://[IP_address]:[NodePort]`

   You obtained the IP address and NodePort in Getting the Grafana IP Address and Port Number.

After you provide the Grafana URL, you are taken to the **Home Dashboard** page. This page provides a work flow to help you configure Grafana, including creating a dashboard in Grafana and generating a query for the dashboard.

Creating a Dashboard for Prometheus Metrics

In Grafana, a dashboard is how you monitor and visualize the metrics that are retrieved from the Prometheus data source. The dashboard is a grouping of one or more panels, prearranged into rows.

In this section, you create a dashboard and select the Graph panel for it. The Graph panel is the main panel in Grafana, and has a rich set of graphing options.

As part of creating your dashboard, you use the query editor feature to formulate a query for the dashboard. A query is used to fine-tune how the metrics for Prometheus appear in Grafana. For this procedure, you specify the `prometheus_engine_query_duration_seconds` query, which allows Grafana to graph the metrics that Prometheus collects about itself.

1. In the **Home Dashboard** page, click the **Create your first dashboard** icon.
2. In the **New dashboard** page, click the **Choose Visualization** icon.
3. In the **Visualization** field, enter `Graph`.
4. Click the **Queries** icon. This icon appears to the left of the **Visualization** field and looks like a database.
5. In the **Query** field, select the Prometheus data source that was added as part of deploying Grafana.
6. In the query editor field, which appears to the right of the **Metrics** drop-down menu, enter the `prometheus_engine_query_duration_seconds` query, and then

click the **Disable/enable query** button. This button appears to the right of the query that you entered and looks like an eye.

7. Verify that data associated with the metric appears, and then click **Save dashboard**.
8. In the **Save As** dialog box, give the dashboard a new name, select a folder location for the dashboard, and then click **Save**.
9. Verify that you see the **Dashboard saved** status message and a graph that displays information about the metric that you entered.