Kubernetes Service Mesh: How To Set Up Istio

Kubernetes Networking

By Mike Mackrory

• Published on February 12, 2021 • Last updated December 7, 2021

In this article, you will learn how to set up Istio as a Kubernetes service mesh using a free Platform9 Kubernetes account.

Overview

This article will cover:

Quick Introduction to Kubernetes Service Mesh and Istio

One of the challenges with a highly dynamic microservices architecture is creating and maintaining connections. As pods are updated, added, and removed, you need a mechanism to identify each pod and enable communication between them and other pods in the cluster.

A service mesh is one way of managing the communications within your cluster. Network proxies are attached to each application container using the Sidecar pattern. Together these proxies operate and monitor communication within the Kubernetes cluster, and this is the service mesh. The service mesh manages security, works to optimize network performance within the cluster, and reports on the state of networking within the Kubernetes cluster. In a previous article, we provided a detailed comparison of various Kubernetes service mesh options, specifically Istio, Linkerd, and Consul.

Istio is an open-source, platform-independent service mesh started by teams from Google and IBM in partnership with the Envoy team from Lyft. Istio leverages the powerful and proven Envoy proxy to provide a stable and secure service mesh for your Kubernetes cluster. This article will walk through how to deploy, and configure work with Istio using a free Platform 9 Kubernetes account.

Getting Started

If you want to experience Istio for yourself and work along as we go through this tutorial, there are a couple of things you'll need to get started. The first is a free account with Platform9. The Platform9 Managed Kubernetes Free Tier makes it easy to get started. The site will guide you through verifying your account, and you should be ready to go in a couple of minutes.

Get started by creating your free Platform9 Managed Kubernetes Account

Email Address

Create Free Account

I'll be using a virtual machine running Ubuntu 18.04 on my local workstation, and I used the BareOS instructions from the Quick Setup Guide to set it up and connect it to my account. The virtual machine needs to be well provisioned. We suggest a minimum of 4 VCPUS, 16 GB of RAM, and 30GB of HDD.

The Quick Setup Guide has instructions for configuring instances with Amazon Web Services (AWS) and Microsoft Azure. You can also follow the BareOS instructions to connect to instances running in the Google Cloud (GCP) or other public cloud providers.

Once I had the node created and prepared, I initiated the creation of a single master cluster. The Platform9 interface is very intuitive and guides you through the process; however, if you get stuck, the Get Your First Container Running on Kubernetes tutorial has some useful tips and walks you through the process as well.

Installing Istio as a Kubernetes Service Mesh using the CLI

The first step is to log into the node and download Istio. We do that with the following command.

```
$ curl -L https://istio.io/downloadIstio | sh -
```

The command above will download the latest version of Istio to the current directory. At the time of writing, the newest version was 1.8.1. Navigate into the newly-created directory.

```
$ cd istio-1.8.1
```

You'll see the LICENSE file, bin directory, and a samples directory among the contents. We need to add the bin directory to our path.

```
$ export PATH=$PWD/bin:$PATH
```

When you install Istio, you're able to choose from several configuration profiles. We'll be using the demo profile because it showcases the abilities of Istio with moderate resource usage. Other profiles are more appropriate for production deployments and custom configurations.

We'll use the Istio Command-line Tool (istioctl) to specify the profile and install the service.

```
$ istioctl install --set profile=demo -y
```

The installation may take a few minutes to complete, but ultimately you'll see output similar to that shown below.

```
✔ Istio core installed
✔ Istiod installed
✔ Egress gateways installed
✔ Ingress gateways installed
✔ Installation complete
```

The final step is to set up a namespace label. When we add this label, we'll enable Istio Injection. With this label in place, Istio will automatically inject Envoy sidecar proxies to newly deployed workloads.

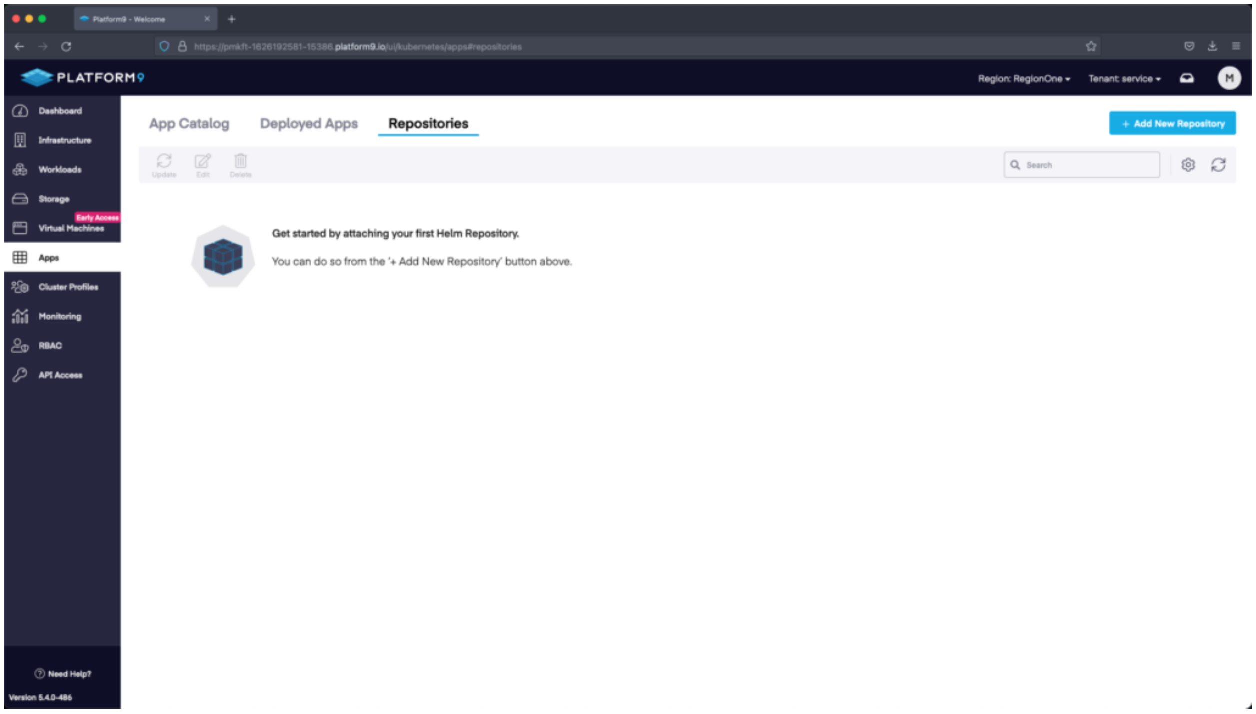
```
$ kubectl label namespace default istio-injection=enabled
```

We're now ready to deploy a sample application and see Istio in action.

Installing Istio using the PMK App Catalog

Another option is to use the PMK App Catalog and the officially supported Helm Chart: https://artifacthub.io/packages/helm/istio-official/istiod.

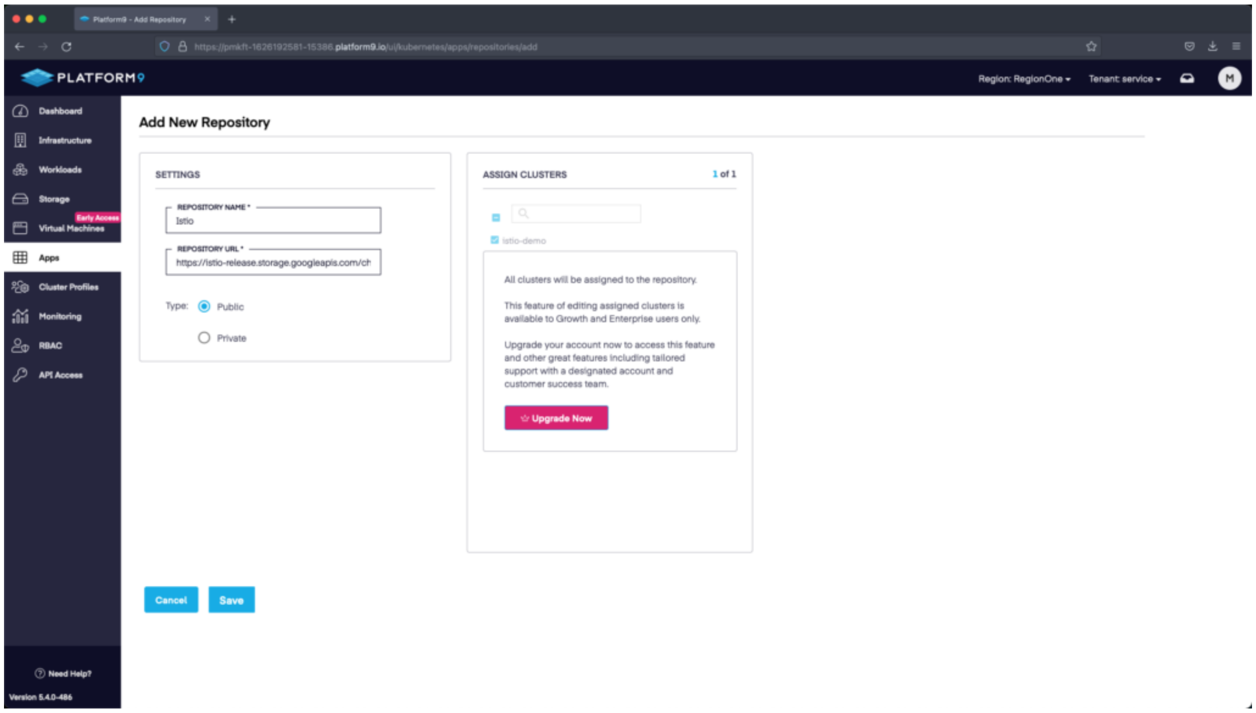
We will start out in the Apps section and open Repositories. From here we can add a repository and attach it to a specific cluster or set of clusters.



Get started by attaching your first Helm Repository.

You can do so from the '+ Add New Repository' button above.

In this section we will name the repository and add the URL where the charts are hosted. In our example we are naming the repository Istio and we are using the official charts repository URL: https://istio-release.storage.googleapis.com/charts.



Add New Repository

SETTINGS

REPOSITORY NAME * Istio

REPOSITORY URL * https://istio-release.storage.googleapis.com/ch

Type: [✓ Public] Private

ASSIGN CLUSTERS 1 of 1

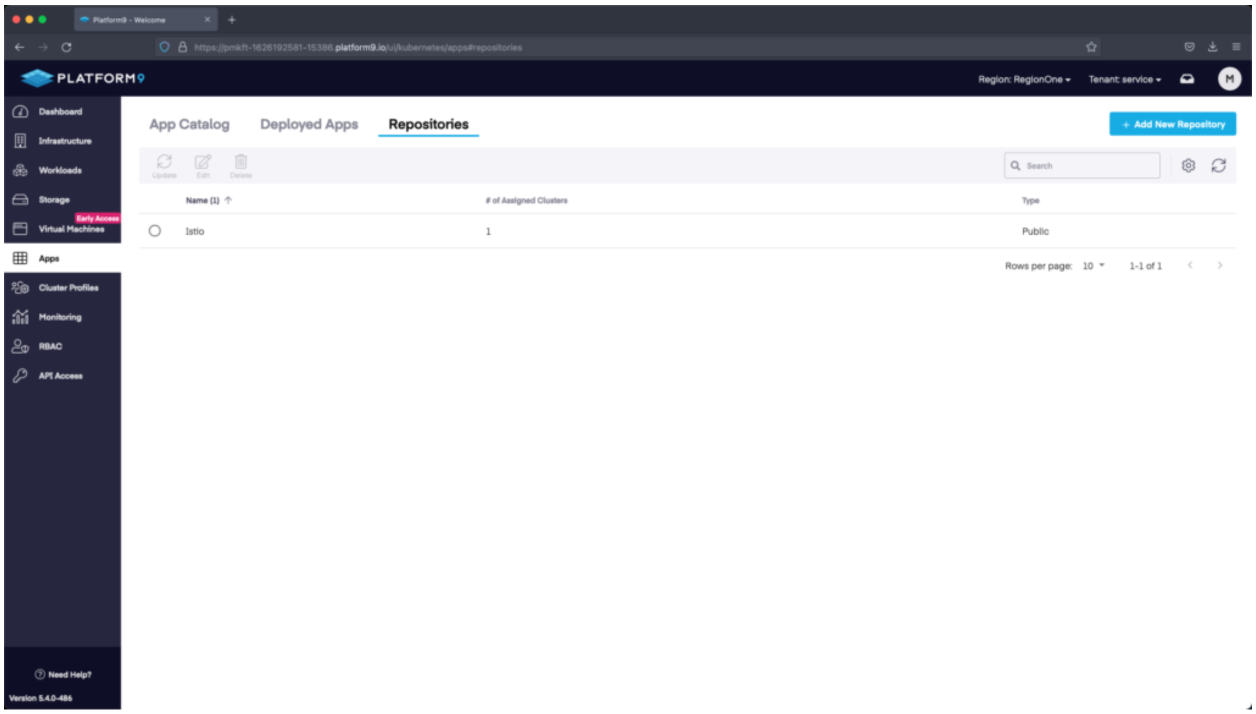
istio-demo

All clusters will be assigned to the repository.

This feature of editing assigned clusters is available to Growth and Enterprise users only.

Upgrade your account now to access this feature and other great features including tailored support with a designated account and customer success team.

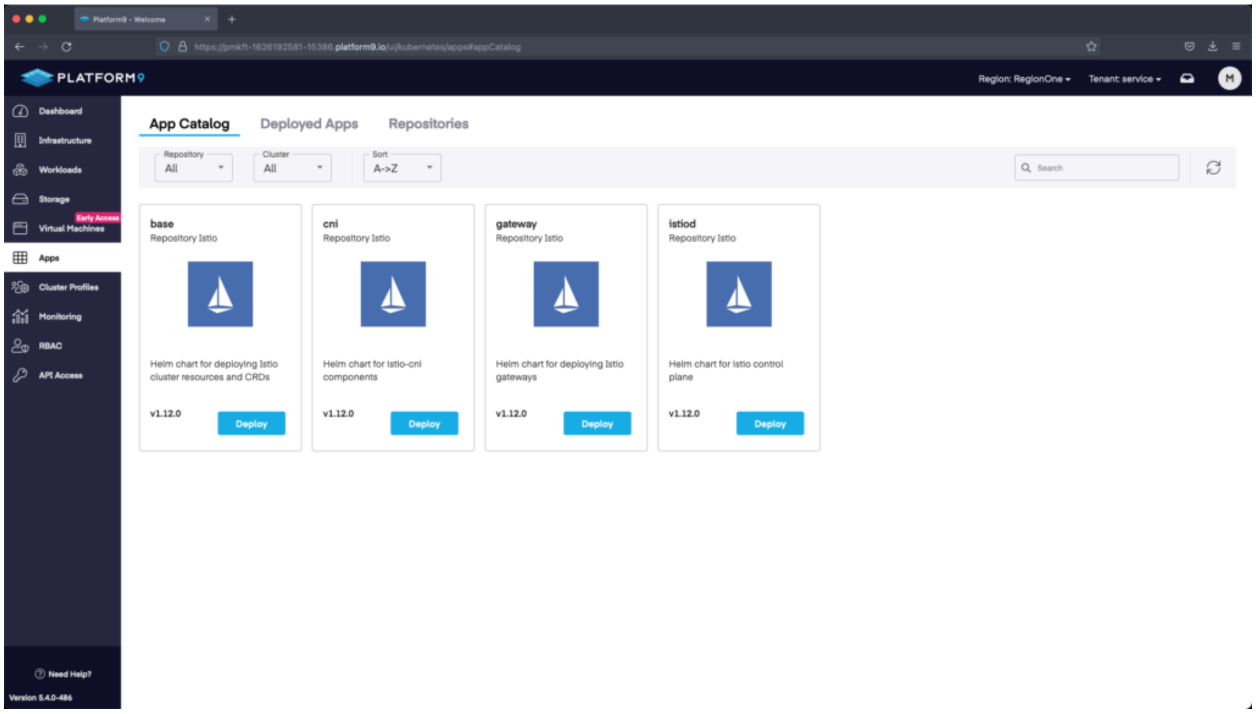
Now that we have added a repository we can view it in the Repositories section. This will show us how many clusters have been assigned to the repository and the type.



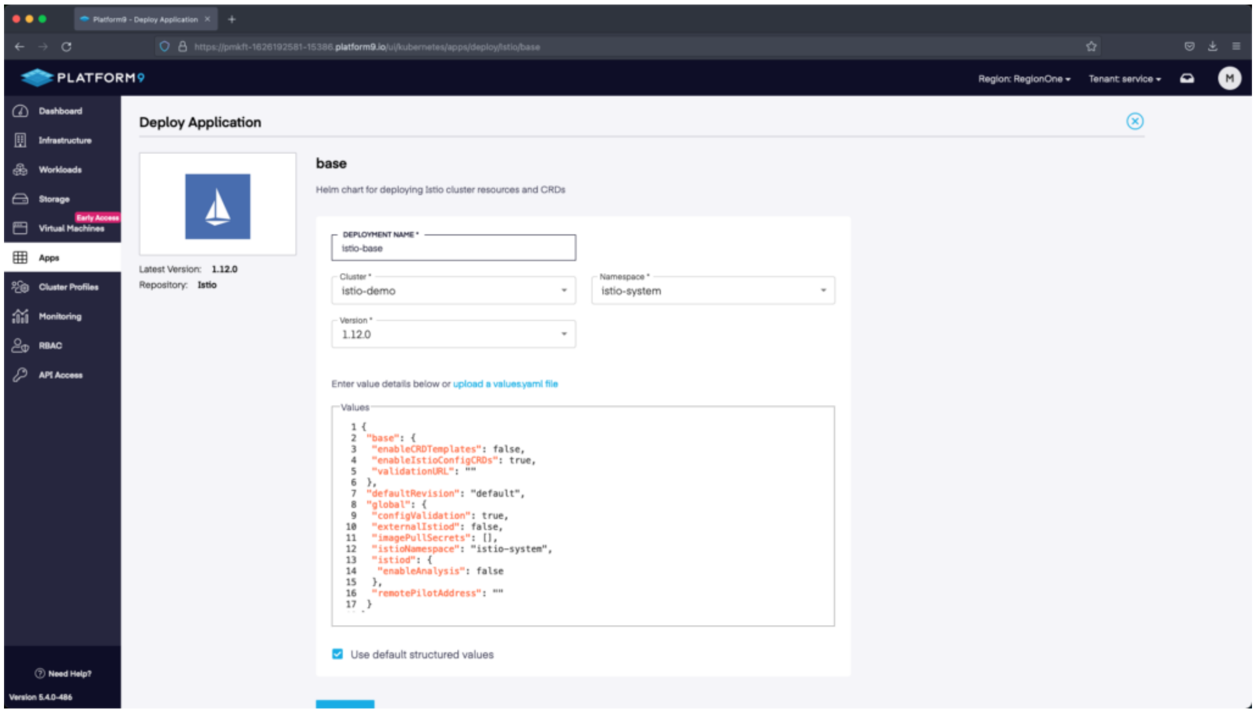
Name (1)	# of Assigned Clusters	Type
Istio	1	Public

Rows per page: 10 1-1 of 1

If we move back over to the App Catalog section we can see the different charts that are available. There are four charts available from the Istio repository: base, cni, gateway, and istiod. We are going to install each.



We will start out with Istio Base. The installation will use the default values, however we need to name the deployment, select the cluster, and select the namespace. The istio-system namespace is not created by default, we will need to add it. To do so select the drop down menu under Namespace and then select “Add new namespace” and name it istio-system. After creating the new namespace we can select it under the Namespace section.



Deploy Application

base

Helm chart for deploying Istio cluster resources and CRDs

Latest Version: 1.12.0
Repository: Istio

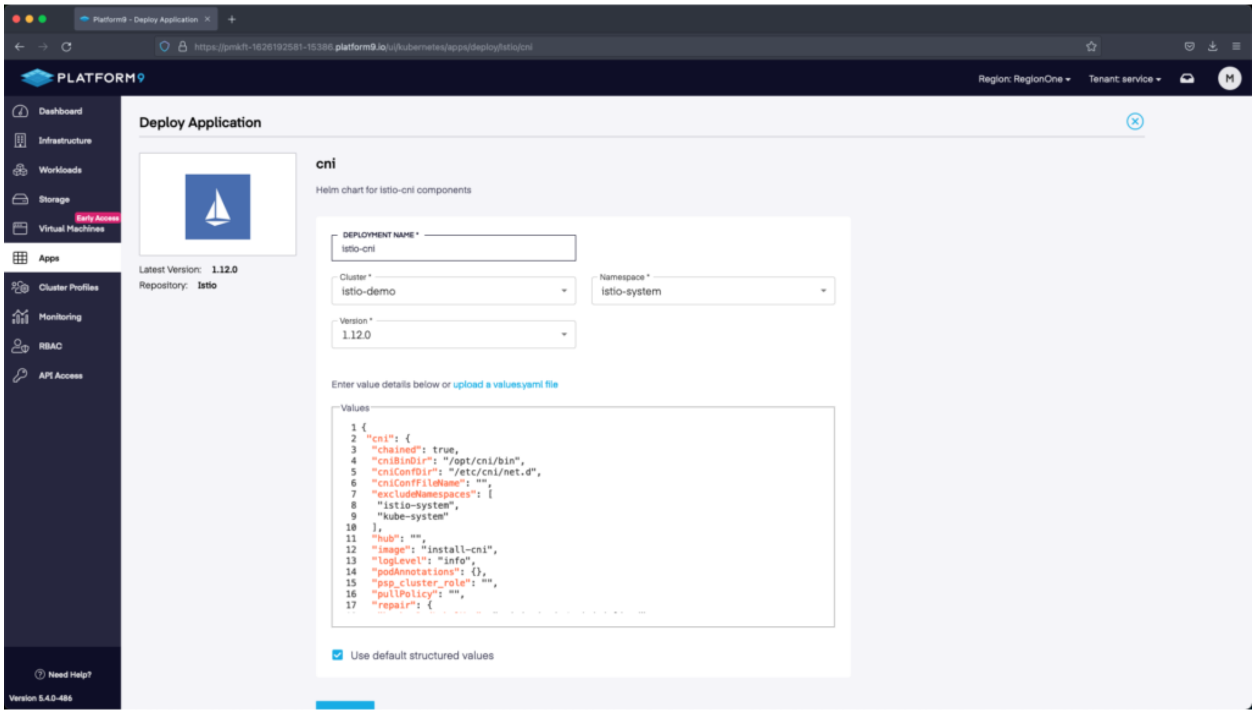
DEPLOYMENT NAME *: istio-base
Cluster *: istio-demo
Namespace *: istio-system
Version *: 1.12.0

Enter value details below or upload a values.yaml file

Values
```
{
 "base": {
  "enableCRDTemplates": false,
  "enableIstioConfigCRDs": true,
  "validationURL": ""
 },
 "defaultRevision": "default",
 "global": {
  "configValidation": true,
  "externalIstiod": false,
  "imagePullSecrets": [],
  "istioNamespace": "istio-system",
  "istiod": {
   "enableAnalysis": false
  },
  "remotePilotAddress": ""
 }
```

[✓] Use default structured values

Istio CNI is the next chart we will deploy. We are using the default values again, but will need to name it, select the cluster, and select istio-system as the namespace.



Deploy Application

cni

Helm chart for istio-cni components

Latest Version: 1.12.0
Repository: Istio

DEPLOYMENT NAME *: istio-cni
Cluster *: istio-demo
Namespace *: istio-system
Version *: 1.12.0

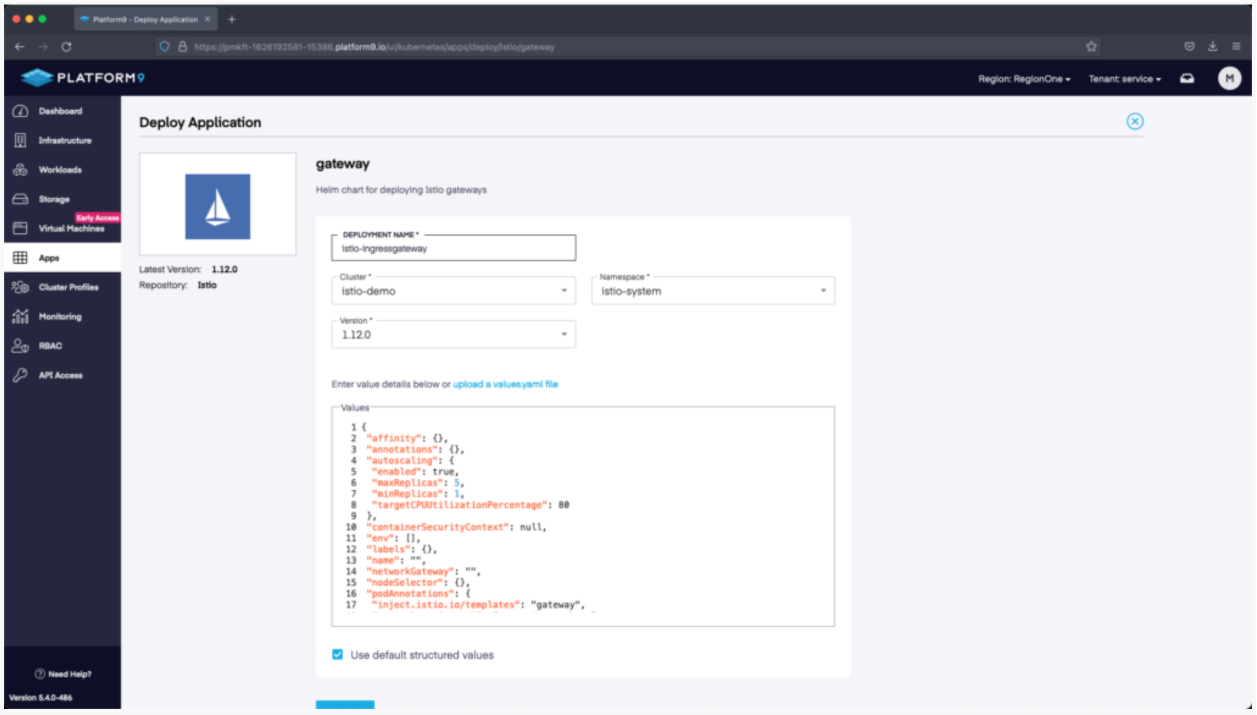
Enter value details below or upload a values.yaml file

Values
```
{
 "cni": {
  "chained": true,
  "cniBinDir": "/opt/cni/bin",
  "cniConfDir": "/etc/cni/net.d",
  "cniConfFileName": "",
  "excludeNamespaces": [
   "istio-system",
   "kube-system"
  ],
  "hub": "",
  "image": "install-cni",
  "logLevel": "info",
  "podAnnotations": {},
  "psp_cluster_role": "",
  "pullPolicy": "",
  "repair": {
```

[✓] Use default structured values

Istio Gateway is next. In this section we need to name the deployment `istio-ingressgateway` for the demo to work. We will use the same cluster and namespace as our other deployments. In the Gateway section you could modify some of the YAML if you wanted to customize how the gateway works, however for our demo we are using defaults.

Istio Gateway Reference:
https://istio.io/latest/docs/reference/config/networking/gateway/



Latest Version: 1.12.0
Repository: Istio

gateway

Helm chart for deploying Istio gateways

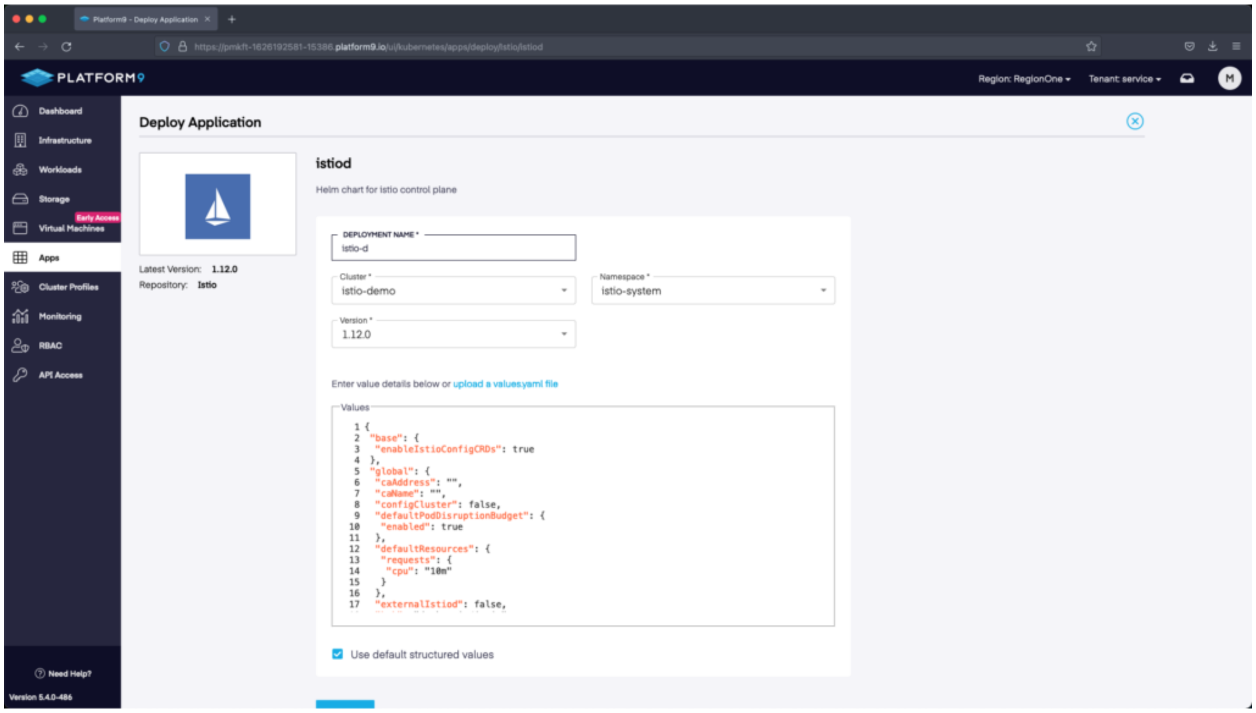
DEPLOYMENT NAME *: istio-ingressgateway
Cluster *: istio-demo
Namespace *: istio-system
Version *: 1.12.0

Enter value details below or upload a values.yaml file

Values
```
{
 "affinity": {},
 "annotations": {},
 "autoscaling": {
  "enabled": true,
  "maxReplicas": 5,
  "minReplicas": 1,
  "targetCPUUtilizationPercentage": 80
 },
 "containerSecurityContext": null,
 "env": [],
 "labels": {},
 "name": "",
 "networkGateway": "",
 "nodeSelector": {},
 "podAnnotations": {
  "inject.istio.io/templates": "gateway",
```

[✓] Use default structured values

The last chart we will install is istiod, the Istio Control Plane. We are using the default settings and are placing it in the same namespace as the other components.



https://pmkft-1626192581-15386.platform9.io/ui/kubernetes/apps/deploy/istio/istiod

Deploy Application

istiod

Helm chart for Istio control plane

Latest Version: 1.12.0
Repository: Istio

DEPLOYMENT NAME *: istio-d

Cluster *: istio-demo

Namespace *: istio-system

Version *: 1.12.0

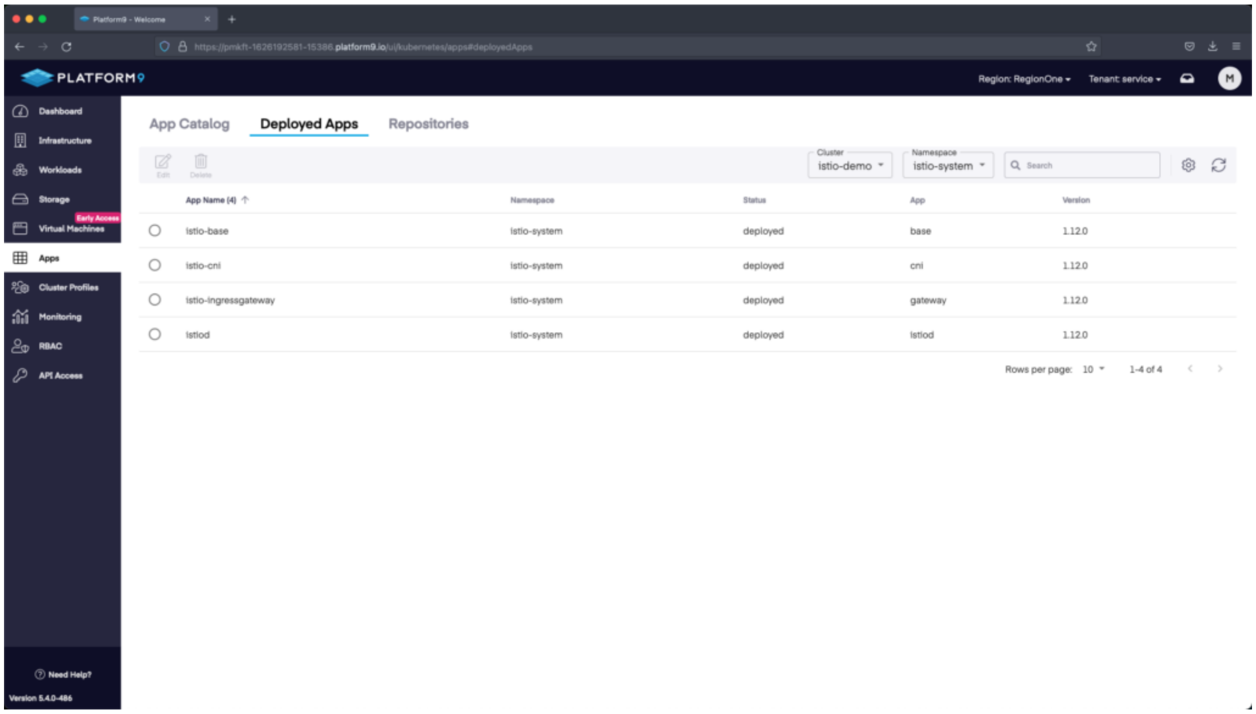
Enter value details below or upload a values.yaml file

Values

```
{
 "base": {
  "enableIstioConfigCRDs": true
 },
 "global": {
  "caAddress": "",
  "caName": "",
  "configCluster": false,
  "defaultPodDisruptionBudget": {
   "enabled": true
  },
  "defaultResources": {
   "requests": {
    "cpu": "10m"
   }
  },
  "externalIstiod": false,
```

☑ Use default structured values

Once we have installed each of the charts we can view them in the Deployed Apps section under the istio-system namespace. Verify that everything has been deployed and then move on to the BookInfo demo.



https://pmkft-1626192581-15386.platform9.io/ui/kubernetes/apps#deployedApps

Cluster: istio-demo
Namespace: istio-system

App Name (4)	Namespace	Status	App	Version
istio-base	istio-system	deployed	base	1.12.0
istio-cni	istio-system	deployed	cni	1.12.0
istio-ingressgateway	istio-system	deployed	gateway	1.12.0
istiod	istio-system	deployed	istiod	1.12.0

Rows per page: 10   1-4 of 4

Platform9 Managed Kubernetes App Catalog Reference and Documentation – https://platform9.com/docs/kubernetes/application-catalog

Deploying the BookInfo Application

Istio comes packaged with several sample applications. We'll deploy the bookinfo application using kubectl. The following assumes you're still in the root directory for the Istio installation.

```
$ kubectl apply -f sample/bookinfo/platform/kube/bookinfo.yaml
```

The YAML configuration file creates several services for us, and we can see a summary of all the installed services if we run the following.

```
$ kubectl get services
```

The output should look similar to the following:

```
NAME     TYPE        CLUSTER-IP  EXTERNAL-IP PORT(S)     AGE
details     ClusterIP   10.21.2.22  < none >  9080/TCP    39s
kubernetes  ClusterIP   10.21.0.1    < none >  443/TCP     48m
productpage ClusterIP   10.21.0.74  < none >  9080/TCP    38s
ratings     ClusterIP   10.21.1.6   < none >  9080/TCP    39s
reviews     ClusterIP   10.21.3.241 < none >  9080/TCP    39s
```

We also need to ensure that all of the pods are ready to go, as this may take a little more time. We can see this on our node directly by executing kubectl get pods, or we can view their status on the Platform9 dashboard.

```
$ kubectl get pods
```

We need two (2) of each pod to have a status of running.

```
NAME                READY   STATUS     RESTARTS    AGE
details-v1-558b8b4b76-ck1g5 2/2 Running     0       10m
productpage-v1-6987489c74-b64sn 2/2 Running     0       10m
ratings-v1-7dc98c7588-qb4ng 2/2 Running     0       10m
reviews-v17f99cc4496-7cd47  2/2 Running     0       10m
reviews-v2-7d79d5bd5d-5gz7j 2/2 Running     0       10m
reviews-v3-7dbcdcbc56-k7wh1 2/2 Running     0       10m
```

Managing Ingress

This configuration creates two resources in the cluster:

- gateway.networking.istio.io/bookinfo-gateway

- virtualservice.networking.istio.io/bookinfo

Istio includes an analysis tool that validates the installation. Now with everything we've completed deploying our application, we can use this tool to validate our namespace.

```
$ istioctl analyze
✔ No validation issues found when analyzing namespace: default
```

For the next step, we need to determine if the environment has an internal or external load balancer. Execute the following command.

```
$ kubectl get svc istio-ingressgateway -n istio-system
```

In the output, look for the EXTERNAL-IP. If the results show an IP Address or a Host Name, then you have an external load balancer. If the results show either or , you don't have access to an external load balancer. Export the following Environment Variables (EVs) based on whether you have an external load balancer or not.

External Load Balancer

```
$ export INGRESS_HOST=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
$ export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].port}')
$ export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].port}')
```

No External Load Balancer

```
$ export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
$ export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
$ export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n istio-system -o jsonpath='{.items[0].status.hostIP}')
```

With those EVs set, you can now set your Gateway URL.

```
$ export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

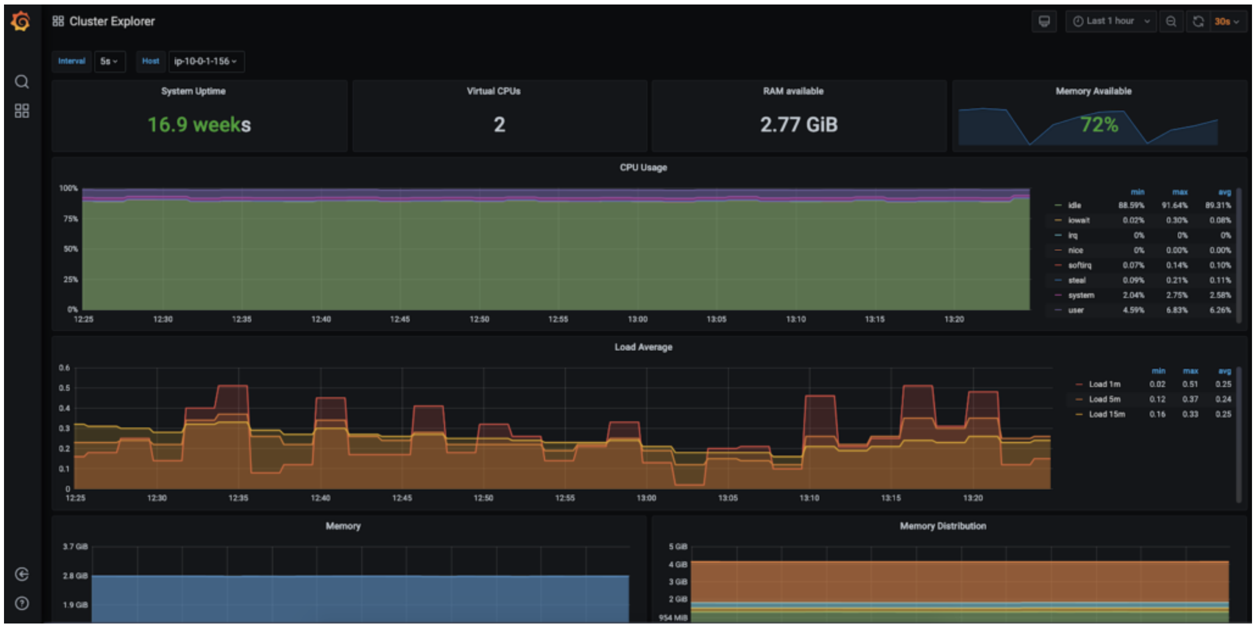
Execute the following command to get the external address for the BookInfo application, and paste it into your web browser to access the application.

```
$ echo "http://$GATEWAY_URL/productpage"
```

Monitoring Kubernetes and Istio

Finally, as with any complex system, we need to monitor and observe what is happening. We can monitor different aspects of Istio with Prometheus, Grafana, Jaeger, and Kiali.

Platform9 Managed Kubernetes Free Tier deploys monitoring with every cluster to ensure that workloads run with a purpose built metrics platform. Platform9 Monitoring provides metrics across all nodes in a cluster as well as metrics from the Kubernetes cluster itself; covering pod metrics, cluster metrics, API Server metrics and OS level collections. When enabled, Monitoring deploys a pre-configured and integrated Prometheus, Alertmanager and Grafana that instantly provides insight into every aspect of the Kubernetes cluster, including a set of rules that fire alerts for the most critical of Kubernetes infrastructure.



Interval: 5s
Host: ip-10-0-1-156

System Uptime: 16.9 weeks
Virtual CPUs: 2
RAM available: 2.77 GiB
Memory Available: 72%

	min	max	avg
idle	88.59%	91.64%	89.31%
iowait	0.02%	0.30%	0.08%
irq	0%	0%	0%
nice	0%	0.00%	0.00%
softirq	0.07%	0.14%	0.10%
steal	0.09%	0.21%	0.11%
system	2.04%	2.75%	2.58%
user	4.59%	6.83%	6.26%

	min	max	avg
Load 1m	0.02	0.51	0.25
Load 5m	0.12	0.37	0.24
Load 15m	0.16	0.33	0.25

Refer to the Monitoring documentation for more information.

For Kubernetes service mesh specific monitoring, Kiali is especially useful because it was built for Istio and gives excellent insights into your service mesh, including what services are attached and how they are connected and performing.

First, let’s install the tool from the sample/addons directory.

```
$ kubectl apply -f samples/addons
```

And now, let’s deploy Kiali to view the dashboard.

```
$ kubectl rollout status deployment/kiali -n istio-system
```

We can open the dashboard using:

```
$ istioctl dashboard kiali
```

Further Reading

For more information on Istio as a Kubernetes service mesh, the latest documentation is available here.

If you're looking for more information about service meshes, including tips on selecting and implementing a service mesh, the following articles have a great deal of helpful information.

1. Comparing Kubernetes service mesh options and how to migrate between them
2. How to set up Linkerd as a Kubernetes service mesh
3. Top tips for configuring your Kubernetes service mesh
4. Best Practices for Selecting and Implementing Your Service Mesh

Next Steps

In this blog, we walked through a tutorial on setting up Istio as a Kubernetes service mesh using a Platform9 Managed Kubernetes Free Tier account. We hope you found this blog informative and engaging. For more reads like this one, visit our blog page or subscribe for up-to-date news, projects, and related content in real-time.