

Istio (on Kubernetes Engine) - Basics to Advanced

Self-link: bit.ly/istio-lab

Author: Ray Tsang [@saturnism](https://twitter.com/saturnism)

Feedback: saturnism.me/feedback

Introduction

Duration: 5:00

This lab will take you through a journey of using Istio on Kubernetes. A prerequisite of this lab is basic understanding of Kubernetes. If you are not familiar with Kubernetes, please see [Kubernetes 101 content](#), as well as the [Kubernetes Code Lab](#).

This lab is an updated Google Doc version of [Ryan Knight's Istio Workshop](#) on GitHub, using Ray's [Istio by Example Spring Boot application sample](#).

There are several other code labs you can try to learn more about running Java applications on GCP:

1. [Google Cloud Native with Spring Boot, App Engine and Kubernetes](#) - bit.ly/spring-gcp-lab
2. [Serverless with Spring Cloud Function, Riff, and Knative](#) - bit.ly/spring-riff-lab

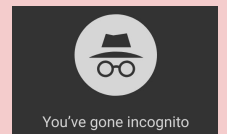
Lab 0 - Initial setup

Duration: 20:00

Task 1 - Logging In

1. The instructor will provide you with a temporary username / password to sign in to the Google Cloud Platform Console.

Important: To avoid conflicts with your personal account, please open a new incognito window for the rest of this lab.



2. Sign in to the Google Cloud Platform Console: <https://console.cloud.google.com/> with the provided credentials. In **Welcome to your new account** dialog, click **Accept**.

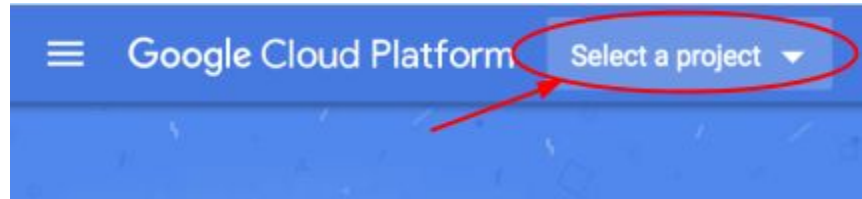
[Privacy Policy](#). Certain Additional Services may also have [service-specific terms](#). Your use of any services your administrator allows you to access constitutes acceptance of applicable service-specific terms.

Click "Accept" below to indicate that you understand this description of how your devstar3002@gcplab.me account works and agree to the [Google Terms of Service](#) and the [Google Privacy Policy](#).

Accept

3. If you see a top bar with **Sign Up for Free Trial** - DO NOT SIGN UP FOR THE FREE TRIAL. Click **Dismiss** since you'll be using a pre-provisioned lab account. If you are doing this on your own account, then you may want the free trial.

4. Click **Select a project**.



5. In the **All** tab, click on your project:

Select a project

[Search projects and folders](#)

RECENT **ALL**

Name	ID
▼ No organization	0
▼ Spring IO Barcelona 2401	springio18-bcn-2401

You should see the Project Dashboard:

The screenshot shows the Google Cloud Platform dashboard for a project named 'Spring IO Barcelona 2401'. The dashboard is divided into several sections:

- Project info:** Displays the project name, ID (springio18-bcn-2401), and number (1056330239641). A link to 'Go to project settings' is provided.
- Resources:** States 'This project has no resources'.
- Trace:** States 'No trace data from the past 7 days'.
- API APIs:** A graph showing 'Requests (requests/sec)' over time. A warning icon indicates 'No data is available for the selected time frame.' A link to 'Go to APIs overview' is provided.
- Google Cloud Platform status:** States 'All services normal' and provides a link to 'Go to Cloud status dashboard'.
- Billing:** Shows 'Estimated charges USD \$0.00 For the billing period May 1 – 23, 2018' and a link to 'View detailed charges'.
- Error Reporting:** States 'No sign of any errors. Have you set up Error Reporting?' and a link to 'Learn how to set up Error Reporting'.

Task 2 - Cloud Shell

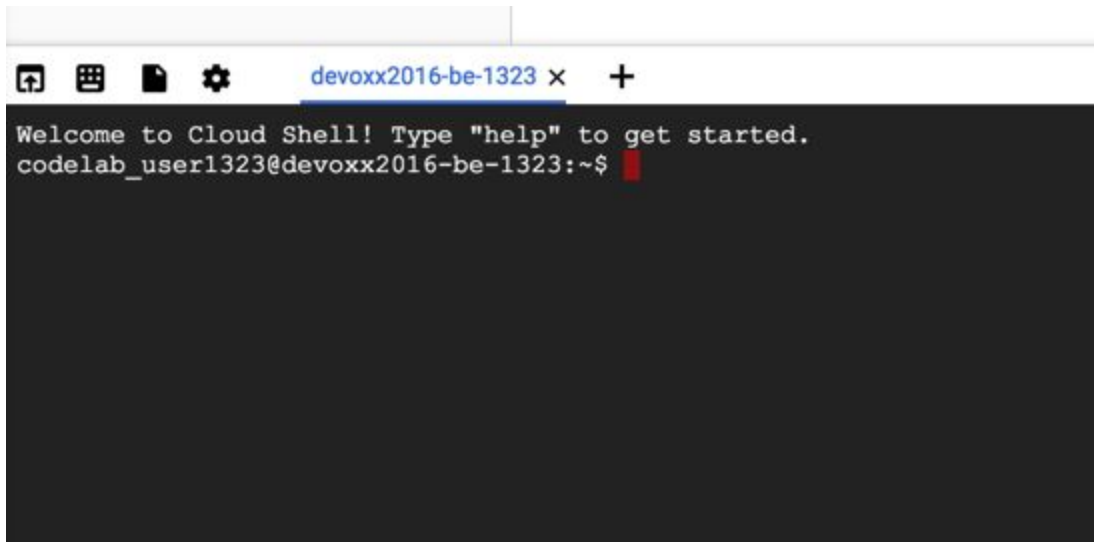
You will do most of the work from the [Google Cloud Shell](#), a command line environment running in the Cloud. This Debian-based virtual machine is loaded with all the development tools you'll need (`docker`, `gcloud`, `kubectl` and others) and offers a persistent 5GB home directory. Open the Google Cloud Shell by clicking on the icon on the top right of the screen:



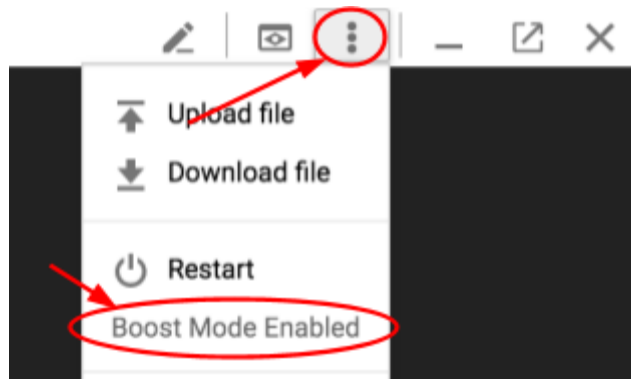
1. When prompted, click **Start Cloud Shell**:



You should see the shell prompt at the bottom of the window:



2. Check to see if Cloud Shell's Boost mode is Enabled.



3. If not, then enable **Boost Mode** for Cloud Shell.

Note: When you run `gcloud` on your own machine, the config settings will be persisted across sessions. But in Cloud Shell, you will need to set this for every new session / reconnection.

Lab 1 - Bootstrap a Kubernetes with Istio

Task 1 - Create a Kubernetes Cluster

1. Enable the Google Cloud Platform APIs so that you can create a Kubernetes cluster.

```
$ gcloud services enable compute.googleapis.com container.googleapis.com
```

2. Set the default region and zone for the zonal resources (e.g., VMs, Kubernetes cluster).

```
$ gcloud config set compute/zone us-east1-d  
$ gcloud config set compute/region us-east1
```

Note: For the lab, use the region/zone recommended by the instructor. Learn more about different zones and regions in [Regions & Zones documentation](#).

3. Creating a Kubernetes cluster in Google Cloud Platform is very easy! Use Kubernetes Engine to create a cluster.

```
$ gcloud container clusters create guestbook \  
  --cluster-version 1.12 \  
  --num-nodes 4 \  
  --machine-type n1-standard-2 \  
  --scopes cloud-platform
```

This will take a few minutes to run. Behind the scenes, it will create Google Compute Engine instances, and configure each instance as a Kubernetes node. These instances don't include the Kubernetes Master node. In Google Kubernetes Engine, the Kubernetes Master node is managed service so that you don't have to worry about it!

Warning: The `scopes` parameter is important for this lab. Scopes determine what Google Cloud Platform resources these newly created instances can access. By default, instances are able to read from Google Cloud Storage, write metrics to Google Cloud Monitoring, etc. For our lab, we add the `cloud-platform` scope to give us more privileges, such as writing to Cloud Storage as well.

Task 2 - Install Istio

1. Download Istio CLI and release.

```
$ cd ~/\  
$ export ISTIO_VERSION=1.1.5  
$ curl -L https://git.io/getLatestIstio | sh -  
$ ln -sf istio-$ISTIO_VERSION istio
```

Note: This code lab was last tested with Istio 1.1.2. If you would like to try a newer version, please help update this document with suggestions!

2. Add Istio binary path to `$PATH`.

```
$ export PATH=~/.istio/bin:$PATH  
$ echo 'export PATH=~/.istio/bin:$PATH' >> ~/.bashrc
```

3. Install Istio into Kubernetes cluster.

```
$ kubectl apply -f ~/.istio/install/kubernetes/helm/istio-init/files
```

```
$ kubectl apply -f ~/istio/install/kubernetes/istio-demo.yaml \
  --as=admin --as-group=system:masters
```

4. Wait until all of the Istio components have the `Running` or `Completed` status.

```
$ watch kubectl get pods -n istio-system
```

5. When you are ready, `Control+C` out of the watch loop.

A lot has been installed!

- Istio Custom Resource Definitions
- Istio Controllers and related RBAC rules
- Prometheus and Grafana for Monitoring
- Jaeger for Distributed Tracing
- Istio Sidecar Injector

Lab 2 - Istio Proxy Injection

1. Check out the sample application and configurations.

```
$ cd ~/
$ git clone https://github.com/saturnism/istio-by-example-java.git
$ cd ~/istio-by-example-java/spring-boot-example/
```

2. Examine the UI Deployment

```
$ less kubernetes/ui-deployment-v1.yaml
```

Note that there is nothing but the application container specified.

In order for Istio to intercept the requests, Istio sidecar must be installed as a sidecar alongside of the application container. There are 2 ways to do this:

1. Manual sidecar injection
2. Automatic sidecar injection via a Mutating Admission Webhook.

3. Use `istioctl` to see what manual sidecar injection will add to the deployment.

```
$ istioctl kube-inject -f kubernetes/ui-deployment-v1.yaml | less
...
  spec:
    containers:
      - image: saturnism/guestbook-ui-istio:1.0
        ...
      - args:
```

```
...
image: gcr.io/istio-release/proxyv2:...
...
initContainers:
- args:
...
image: gcr.io/istio-release/proxy_init:...
...
```

Notice that the output has more than just the application container. Specifically, it has an additional `istio-proxy` container, and an init container.

The init container is responsible for setting up the IP table rules to intercept incoming and outgoing connections and directing them to the Istio Proxy. The `istio-proxy` container is the Envoy proxy itself.

If you want to use manual Istio sidecar injection, then you would always filter your Kubernetes deployment file through `istioctl` utility, and deploy the resulting Deployment specification.

4. Instead of manual injection, you can also use Automatic Sidecar Injection. This works by using Kubernetes's Mutating Admission Webhook mechanism to intercept new Pod creations and automatically enhancing the Pod definition with Istio proxies.

```
$ kubectl get MutatingWebhookConfiguration istio-sidecar-injector -oyaml
```

5. This configuration points to the `istio-sidecar-injector` service in the `istio-system` namespace, which is backed by pods managed by the `istio-sidecar-injector` deployment

```
$ kubectl -n istio-system get svc istio-sidecar-injector
$ kubectl -n istio-system get deployment istio-sidecar-injector
```

6. You can turn on Automatic Sidecar Injection at the Kubernetes namespace level, by setting the label `istio-injection` to `enabled`.

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

7. Since Istio 1.0.3+, you will see MySQL connectivity issues using the default Istio installation documented in [GitHub Issue #11062](#). The workaround is documented in [Istio FAQ](#). For this lab, we'll apply this workaround.

```
$ kubectl apply -f - <<EOF
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: mysql-nomtls-authn
spec:
  targets:
  - name: mysql
```

```
EOF
```

8. Deploy the entire application in one shot.

```
$ kubectl apply -f kubernetes/
```

This will:

1. Provision a Persistent Volume to store MySQL data.
2. Deploy MySQL and mount the newly created Persistent Volume.
3. Deploy Redis, frontend UI, and 2 backend services.

9. Check that all components have the `Running` status, and that Ready column shows `2/2`.

```
$ watch kubectl get pods
```

Note: The Ready column shows `2/2` because there are 2 containers running inside of each of the Pods - the application container and the Istio Proxy container. `2/2` indicates both of these containers are up and running.

10. When you are ready, `Control+C` out of the watch loop.

11. You can see the sidecar proxy injected into the pod.

```
$ kubectl get pods -lapp=guestbook-service
NAME                                READY    STATUS    RESTARTS    AGE
guestbook-service-v1-7bbdb86c58-4bqq2  2/2      Running   0            15m
guestbook-service-v1-7bbdb86c58-t42q2  2/2      Running   0            15m

// Pick one of the pod from your list, and describe it.
$ kubectl describe pod guestbook-service-v1-...
```

You should see the initialization containers, and well as a container named `istio-proxy` automatically injected into the pod.

Lab 3 - Traffic Management

Taks 1 - Ingress Traffic

All of the services now have an internal load balancer. In this lab, you'll expose the UI service via the Istio Ingress. Istio Ingress is not a Kubernetes Ingress controller. I.e., you won't configure Istio Ingress with Kubernetes Ingress definitions.

1. Find the Istio Ingress IP address.


```
$ kubectl get svc istio-ingressgateway -n istio-system
$ export INGRESS_IP=$(kubectl -n istio-system get svc istio-ingressgateway \
  -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
$ echo $INGRESS_IP
```

2. Connect to the Istio Ingress IP.

```
$ curl $INGRESS_IP
curl: (7) Failed to connect to ... port 80: Connection refused
```

The connection is refused because nothing is binding to this ingress.

3. Bind a Gateway to the Istio Ingress.

```
$ kubectl apply -f istio-rules/gateway.yaml
```

An Istio Gateway describes a load balancer operating at the edge of the mesh receiving incoming or outgoing HTTP/TCP connections. The specification describes a set of ports that should be exposed, the type of protocol to use, virtual host name to listen to, etc.

Warning: For the purpose of the lab, this will make the application accessible on the public internet. There are a number of production configurations you may consider for private endpoints:

- Accessing services within the cluster does not require the gateway.
- Accessing services across the cluster, but internal, consider using GKE private cluster, VPC, IP aliasing, and internal load balancer.
- Accessing services on the public internet but requiring credentials, consider using Identity Access Proxy.

4. Curl the Istio Ingress IP again, and observed that it's now returning 404 error..

```
$ curl -v $INGRESS_IP
```

It's returning 404 error because nothing is binding to the Gateway yet.

5. Create a Virtual Service and binding it to the Gateway.

```
$ kubectl apply -f istio-rules/guestbook-ui-vs.yaml
```

A Virtual Service defines the rules that control how requests for a service are routed within an Istio service mesh. For example, a virtual service can route requests to different versions of a service or to a completely different service than was requested. Requests can be routed based on the request source and destination, HTTP paths and header fields, and weights associated with individual service versions.

6. Find the Ingress IP again, and open it up in the browser.

```
$ echo http://$INGRESS_IP
```

7. Say Hi in the UI!

Guestbook **Home**

Your name:

Message:

Hello Ray from helloworld-service-v1-f8dd6bb88-tfj8j with 1.0

Ray Hello

Task 2 - Fault Injection

1. Inject a fault to make Guestbook Service reply HTTP 503 error 100% of the time.

```
$ kubectl apply -f istio-rules/guestbook-service-503.yaml
```

2. Go back to the browser and refresh the UI page.

Guestbook

Home

Your name:

Ray

Message:

Greet!

Hello Ray from helloworld-service-v1-f8dd6bb88-662qb with 1.0

system Guestbook Service is currently unavailable

3. Simply delete the rule to restore traffic.

```
$ kubectl delete -f istio-rules/guestbook-service-503.yaml
```

Note: There are different ways to inject fault. You can inject network failures (L4 faults), or HTTP errors (L7 faults).

Task 3 - Traffic Splitting

1. Install UI v2, which has a different background color than v1.

```
$ kubectl apply -f kubernetes-v2/ui-deployment-v2.yaml
```

2. Wait for v2 pods are up and running.

```
$ watch kubectl get pods -lapp=guestbook-ui
```

3. Visit the UI from the browser, and refresh a couple of times.

About 50% of the time you'll see v1 with white background, and the other 50% of the time you'll see v2 with green background.

Note: There are different load balancing methods you can configure via [Destination Rule's Load Balancer Setting](#), e.g., Round Robin, Least Connection, and Random. It can also [load balance with consistent hashing](#) to achieve affinity by using Cookie, Header, or Source IP.

4. Before you can control traffic splitting between these 2 versions, you first need to define destination subsets. Each subset can have a unique pod selector based on labels.

```
$ kubectl apply -f istio-rules/guestbook-ui-dest.yaml
```

5. Once you have the subsets defined, you can configure weight-based traffic split for the different subsets. First, shift all traffic to v1.

```
$ kubectl apply -f istio-rules/guestbook-ui-vs-v1.yaml
```

6. Go back to the browser and refresh several times. Confirm that all traffic is now going to v1.

7. You can update the weight as you need. Shift 20% of the traffic to v2.

```
$ kubectl apply -f istio-rules/guestbook-ui-vs-20p-v2.yaml
```

8. You can also use Virtual Service to direct traffic based on the request data from the header, URI, or HTTP method. Shift all traffic from Chrome browser to v2, and other browsers to v1.

```
$ kubectl apply -f istio-rules/guestbook-ui-vs-chrome.yaml
```

9. Try loading the UI page from a Chrome browser vs another one.

Task 4 - Egress Traffic

In Istio 1.1+, Egress traffic is allowed by default through the setting of [Outbound Traffic Policy Mode to ALLOW_ANY](#). You can change this mode to block all egress traffic by default and allowing only specific traffic through.

1. Change the traffic policy to REGISTRY_ONLY to block all egress traffic.

```
$ kubectl get configmap istio -n istio-system -o yaml | \
  sed 's/mode: ALLOW_ANY/mode: REGISTRY_ONLY/g' | \
  kubectl replace -n istio-system -f - .
```

2. Deploy a Shell pod into Kubernetes

```
$ kubectl create -f https://k8s.io/examples/application/shell-demo.yaml
```

3. Validate that the Pod has status of `Running`, and the `Ready` column shows `2/2`.

```
$ watch kubectl get pods
```

Note: The Ready column shows 2/2 because there are 2 containers running inside of each of the Pods - the application container and the Istio Proxy container. 2/2 indicates both of these containers are up and running.

4. When you are ready, `Control+C` out of the watch loop.

5. Connect into that Pod's shell.

```
$ kubectl exec -ti shell-demo /bin/bash
```

6. Inside the pod, try to update packages.

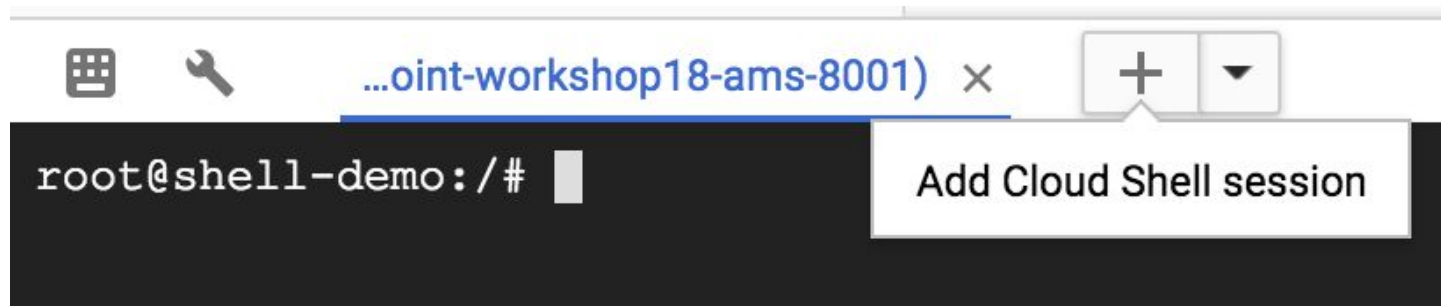
```
root@shell-demo:/# apt-get update
```

Notice that there are lots of connection errors! This is because all of the egress traffic is already intercepted by Istio and we configured it to block all egress traffic. You need to allow egress to `*.debian.org` destinations. There are 2 different ways to do this:

- Configure an IP range that should be intercepted by Istio. This will require determining the IP range that the Kubernetes cluster uses for Pod IPs.
- Or, add Service Entry to have fine-grained control over which external services you want to allow.

This lab will do the latter.

7. Open a new Cloud Shell session tab.



8. In the new tab, configure a Service Entry to enable egress traffic to *.debian.org.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: debian
spec:
  hosts:
  - "*.debian.org"
  location: MESH_EXTERNAL
  ports:
  - number: 80
    name: http
    protocol: HTTP
EOF
```

9. In the original Cloud shell session that is currently connected to the Shell Demo pod, and try to update packages again.

```
root@shell-demo:/# apt-get update -o Acquire::http::Pipeline-Depth=0
root@shell-demo:/# apt-get install -y curl -o Acquire::http::Pipeline-Depth=0
```

Note: This lab disables the HTTP pipelining to avoid outbound connection issues.

Notice that this time, you are able to connect and install curl successfully.

10. However, whenever you try to connect to the outside that hasn't been whitelisted by Service Entry, you'll get an error.

```
root@shell-demo:/# curl https://google.com
```

10. Typically Istio will examine the destination hostname using the HTTP host header. However, it's impossible to examine the header of a HTTPS request. For HTTPS requests, Istio uses SNI to inspect the host name instead.

```
$ cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: google
spec:
  hosts:
  - "google.com"
  location: MESH_EXTERNAL
  ports:
```

```
- number: 443
  name: https
  protocol: HTTPS
EOF
```

11. Reconnect and see that the traffic can now go through.

```
root@shell-demo:/# curl https://google.com
```

12. Exit from the pod.

```
root@shell-demo:/# exit
```

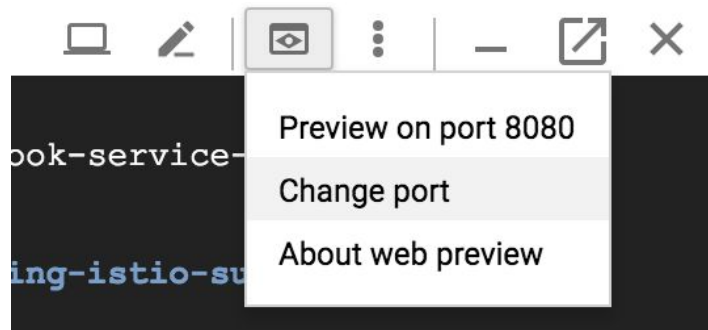
Lab 4 - Monitoring and Observability

Task 1 - Monitoring

1. Grafana is installed by default. Establish a secure tunnel to the Grafana pod.

```
$ kubectl -n istio-system port-forward $(kubectl -n istio-system get pod \
-l app=grafana -o jsonpath='{.items[0].metadata.name}') 3000:3000
```

2. In Cloud Shell, click **Web Preview icon** → **Change port**



3. Enter port 3000, and click **Change and Preview**

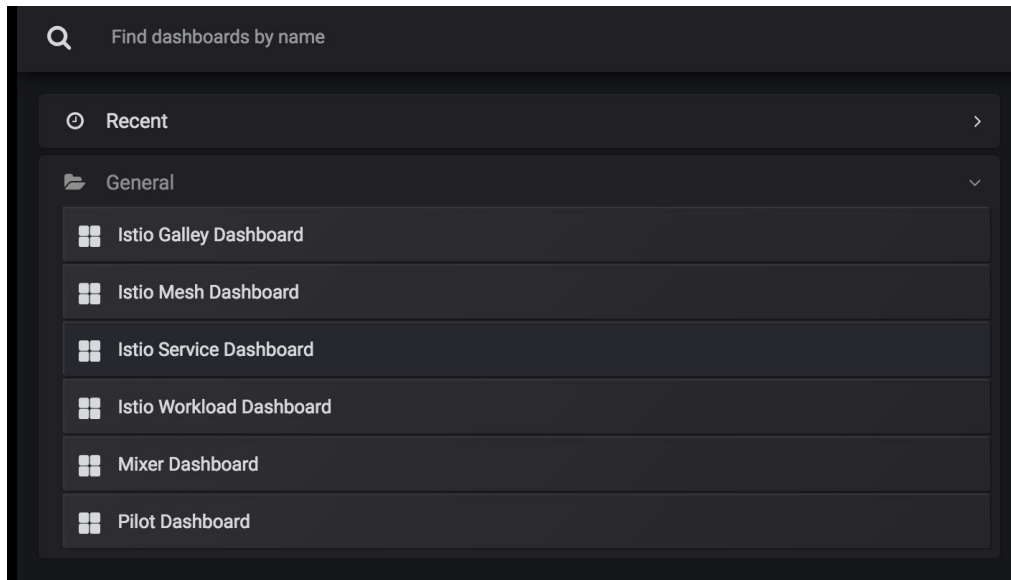
Change Preview Port

Port Number:

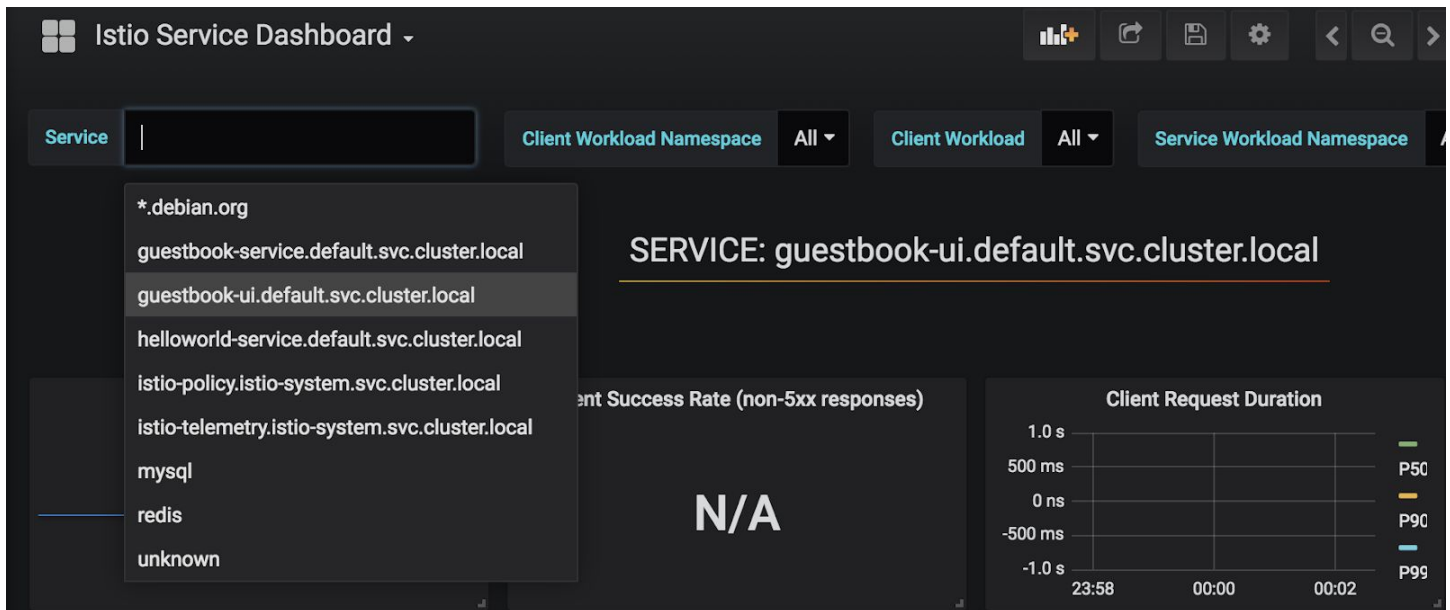
[CANCEL](#) [CHANGE AND PREVIEW](#)

This will establish a connection to your Cloud Shell machine's port 3000, which is connected to a Grafana pod inside of your Kubernetes cluster.

4. In the Grafana Dashboard, click **Home** → **Istio Service Dashboard**

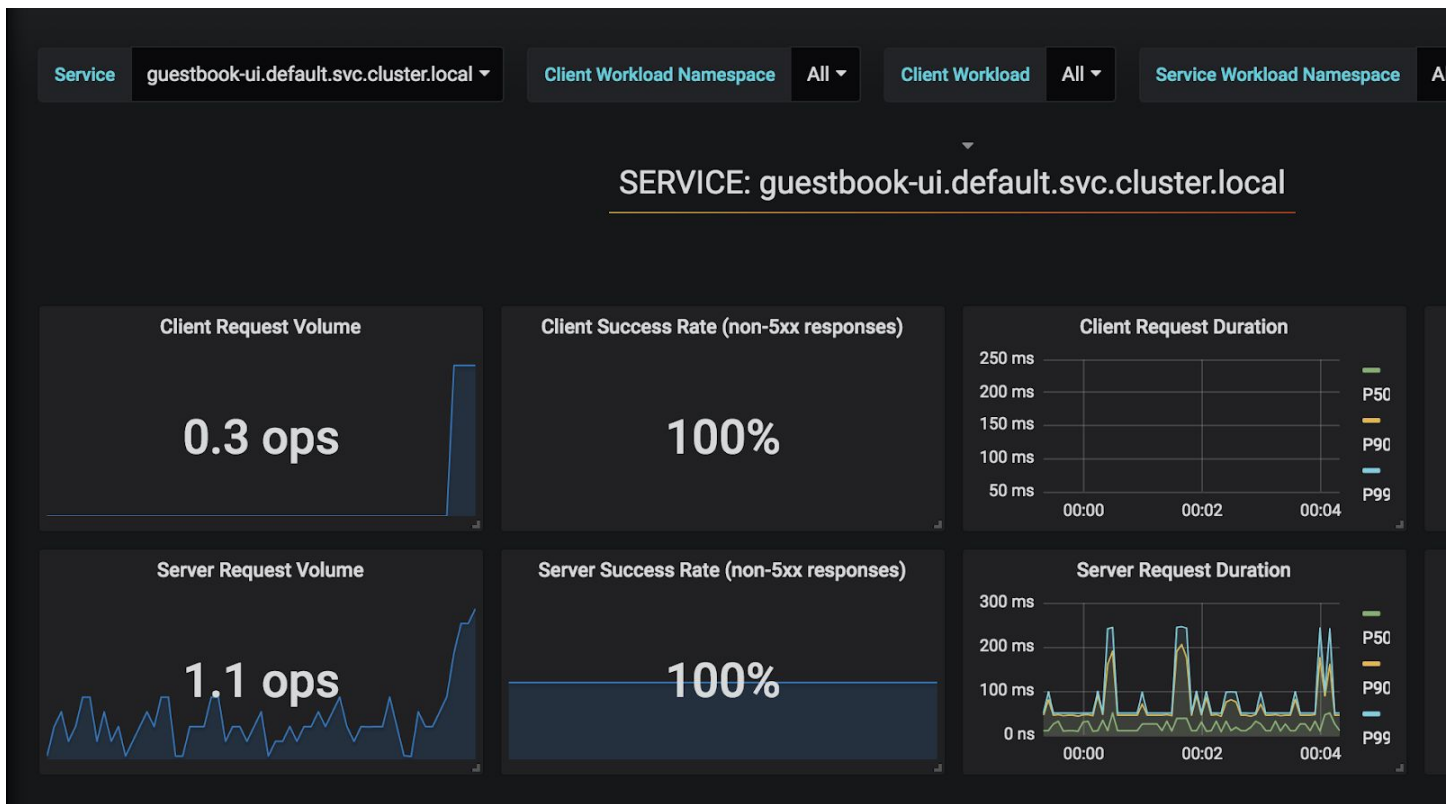


5. In Service, find and click **guestbook-ui.default.svc.cluster.local**



This will pull up the metrics specific to the Guestbook UI service.

6. Make a few requests to the Guestbook UI from your browser, and you should also see the Istio Dashboard update with the metrics.

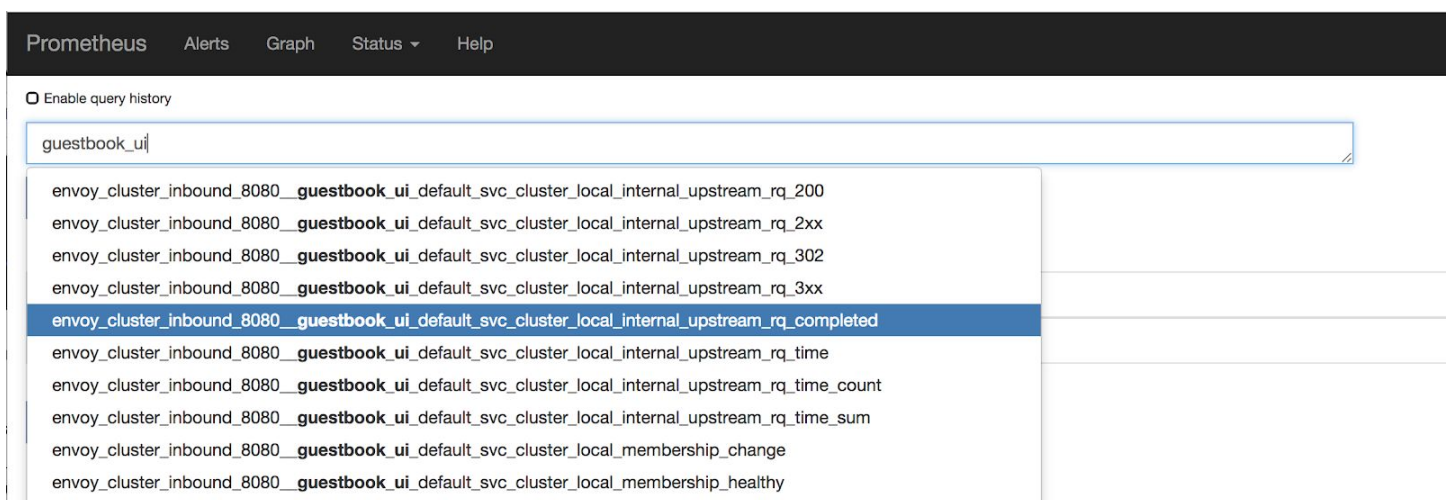


The metrics are actually coming from Prometheus. Behind the scenes, Prometheus agent is scraping metrics from Envoy proxy, which has all the monitoring metrics such as latency, response status distributions, etc.

7. Establish a tunnel to Prometheus instead.

```
$ kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l \
  app=prometheus -o jsonpath='{.items[0].metadata.name}') 9090:9090
```

8. Use Cloud Shell web preview to preview port 9090. This will take you to the Prometheus console.



10. Let's take a look at the proxy in more detail. Find a pod to investigate:

```
$ kubectl get pods -lapp=guestbook-ui
```

NAME	READY	STATUS	RESTARTS	AGE
guestbook-ui-v1-7764f9d866-jrjbg	2/2	Running	0	13m
guestbook-ui-v1-7764f9d866-n9lw1	2/2	Running	0	14m

11. Pick one of the 2 pods and port forward to Envoy.

```
$ kubectl port-forward guestbook-ui-v1-..... 15000:15000
```

12. Use Cloud Shell web preview to preview port 15000. This will take you to this Envoy instance's Admin console.

Command	Description
certs	print certs on machine
clusters	upstream cluster status
config_dump	dump current Envoy configs (experimental)
<code>cpuprofiler</code>	enable/disable the CPU profiler
<code>healthcheck/fail</code>	cause the server to fail health checks
<code>healthcheck/ok</code>	cause the server to pass health checks
help	print out list of admin commands
hot_restart_version	print the hot restart compatibility version
listeners	print listener addresses
<code>logging</code>	query/change logging levels
<code>quitquitquit</code>	exit the server
<code>reset_counters</code>	reset all counters to zero
runtime	print runtime values
<code>runtime_modify</code>	modify runtime values
server_info	print server version/status information
stats	print server stats
stats/prometheus	print server stats in prometheus format

13. Click **stats/prometheus** to see the exported Prometheus metrics.

14. The Prometheus scraper is in the `istio-system` namespace.

```
$ kubectl -n istio-system get pods
```

...				
prometheus-84bd4b9796-fpjtn	1/1	Running	0	57m
...				

15. The Prometheus scraper configuration is stored in the config map.

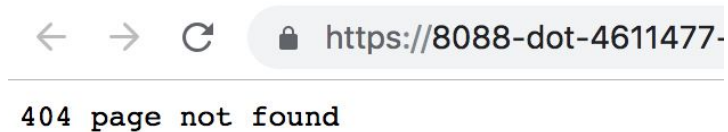
```
$ kubectl -n istio-system get configmap prometheus -oyaml --export | less
```

Task 2 - Observability

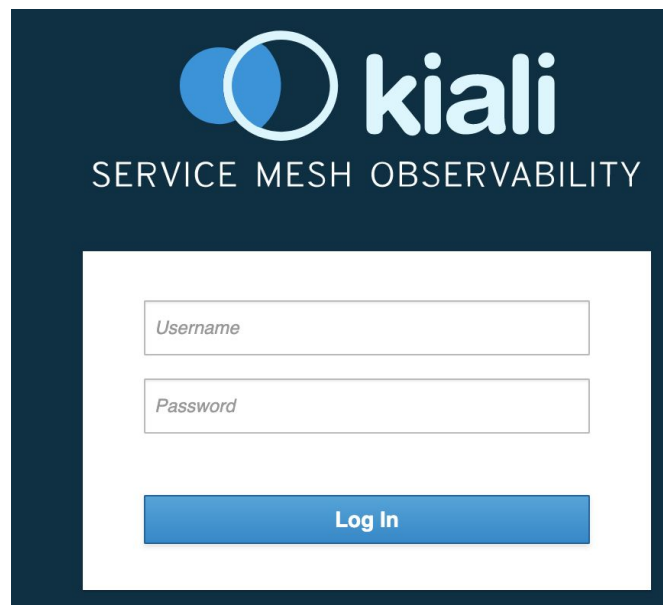
1. Establish a tunnel to Kiali.

```
$ kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l \
  app=kiali -o jsonpath='{.items[0].metadata.name}') 20001:20001
```

2. Use Cloud Shell web preview to preview port 20001. You'll first give you a 404 not found error.

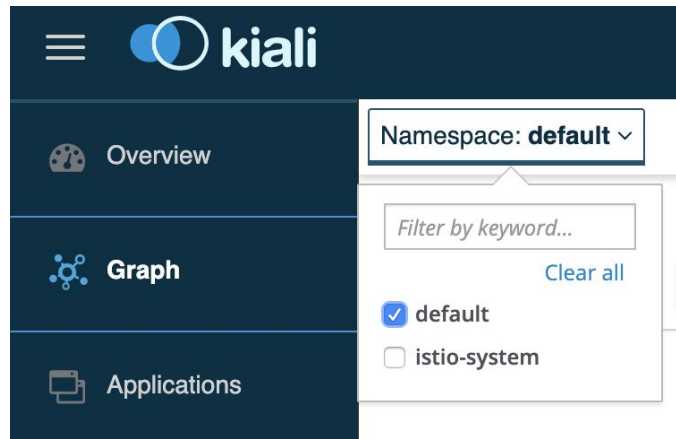


3. Using the same Cloud Shell generated URL, append the path to `kiali`. E.g., `https://20001-dot-...-dot-devshell.appspot.com/kiali`.

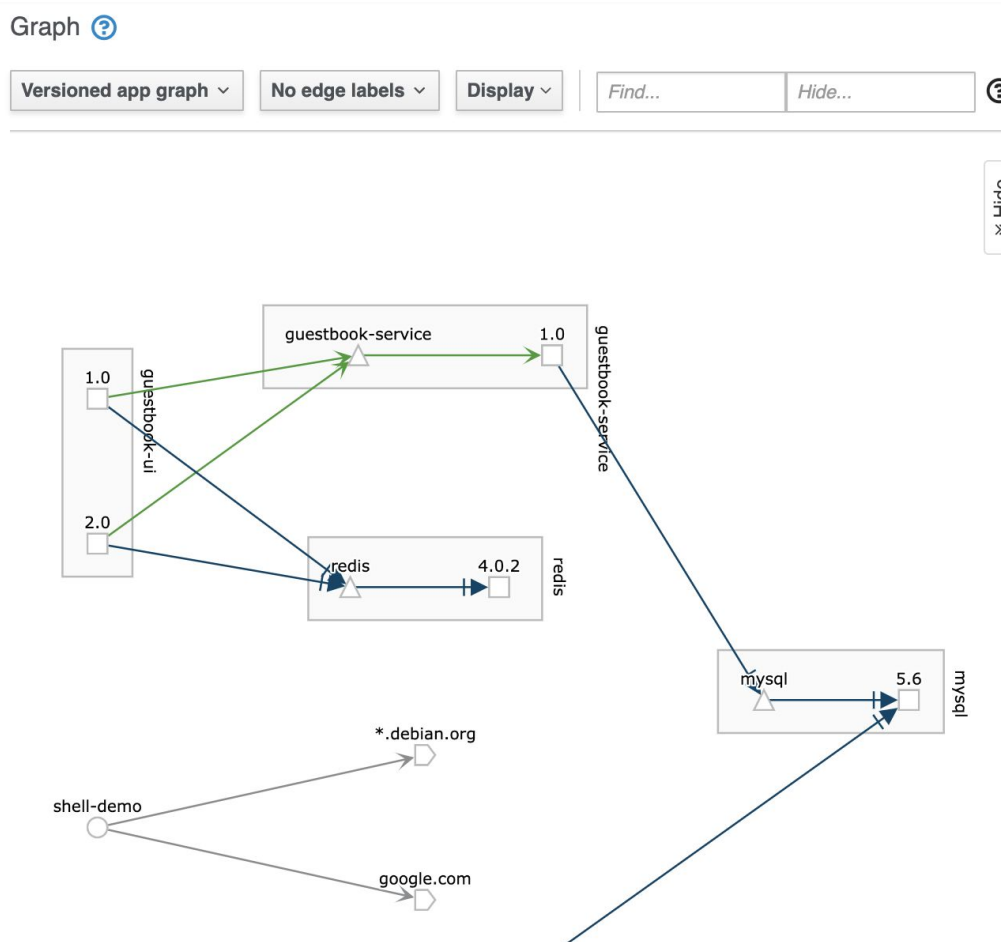


4. Login with username `admin` and password `admin`.

5. Click **Graph**, and select the default **Namespace**.



6. This will bring up the service graph that shows service to service connections.



7. To see traces for requests, establish a tunnel to Jaeger.

```
$ kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l \
  app=jaeger -o jsonpath='{.items[0].metadata.name}') 16686:16686
```

9. Use Cloud Shell web preview to preview port 16686 to see the Jaeger console.

Jaeger UI

Lookup by Trace ID...

Search

Dependencies

Find Traces

Service (8)

Select A Service

Operation (0)

all

Tags ?

http.status_code=200 error=true

Lookback


Last Hour

Min Duration

e.g. 1.2s, 100ms, 500us

Max Duration

e.g. 1.1s



Logo by Levi Pezabadi
www.pelabadi.com

10. In **Find Traces** → **Services**, click **istio-ingressgateway**.

Find Traces

Service (9)

istio-ingressgateway

^

guestbook-service

guestbook-ui

helloworld-service

istio-ingressgateway

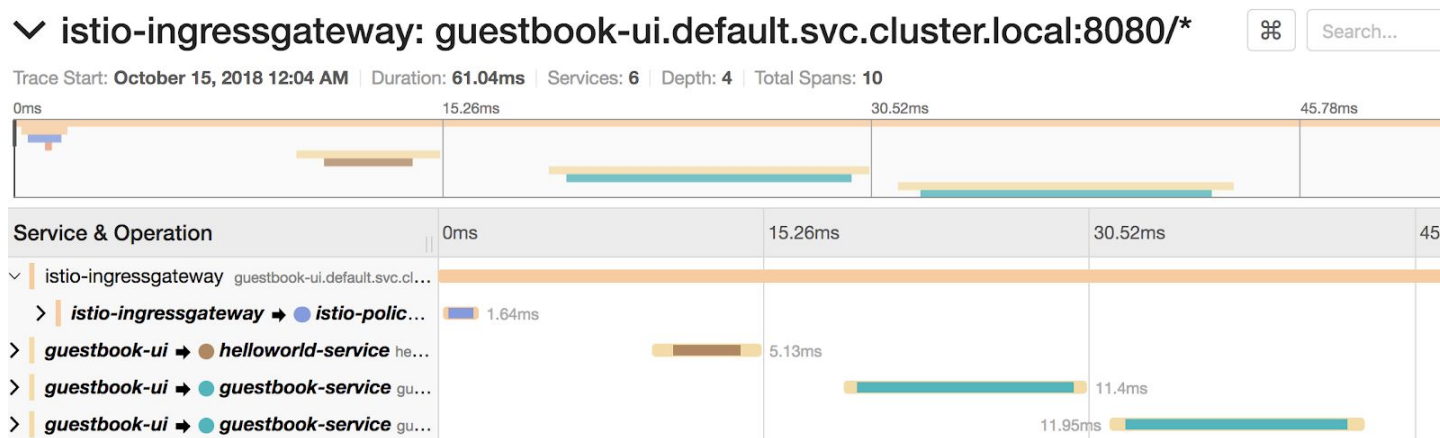
istio-mixer

istio-policy

7. Click **Find Traces**. This will show you all the requests that came through the Istio Ingress.



8. Click into one of the traces to see more information.



Note: Remember, Istio cannot correlate traces for you unless you propagate the trace headers downstream everytime you make a microservice call. This example uses Spring Cloud Sleuth to propagate the default `x-b3-*` headers as well as 2 additional Istio-specific headers.



Ray Tsang
@saturnism

Follow



.@springcloud Sleuth propagates x-b3 headers & works seamlessly w/ @IstioMesh. You can configure additional header propagation for more Istio headers: spring.sleuth.propagation-keys=x-request-id,x-ot-span-context (thnx @MGrzejszczak!)

Lab 5 - Service Security with mTLS

Task 1 - Install Istio with Auth

Congratulations! You went through the basics of Istio. In the following labs, you'll add Mutual TLS and relevant capabilities to your deployment.

1. Install Istio Auth. (Normally if you know you want mTLS, you would install Istio Auth from the start).

```
$ kubectl apply -f ~/istio/install/kubernetes/istio-demo-auth.yaml \
  --as=admin --as-group=system:masters
```

2. At the beginning of the lab, we created a policy for MySQL to workaround a known issue. The policy permitted communication to MySQL w/o mTLS. Delete that policy workaround so that MySQL connection can be mTLS.

```
$ kubectl delete policy mysql-nomtls-authn
```

2. Enable mTLS in the default namespace.

```
$ kubectl apply -f - <<EOF
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "default"
  namespace: "default"
spec:
  peers:
  - mtls:
      mode: STRICT
EOF
```

3. Configure Destination Policy.


```
$ kubectl apply -f - <<EOF
apiVersion: "networking.istio.io/v1alpha3"
kind: "DestinationRule"
metadata:
  name: "default"
  namespace: "default"
spec:
  host: "*.default.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
EOF
```

4. Immediately, you'll notice that all of the pods are failing readiness probes and liveness probes!

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
guestbook-service-v1-7bbdb86c58-c7pqf	1/2	Running	1	1h
guestbook-service-v1-7bbdb86c58-st6dh	1/2	Running	1	1h
guestbook-ui-v1-7764f9d866-jrjbq	1/2	Running	2	1h
guestbook-ui-v1-7764f9d866-n9lwl	1/2	Running	1	1h
guestbook-ui-v2-56cc49c4cd-2kvsh	1/2	Running	1	1h
guestbook-ui-v2-56cc49c4cd-bgxns	1/2	Running	1	1h
helloworld-service-v1-f8dd6bb88-662qb	1/2	Running	1	2h
helloworld-service-v1-f8dd6bb88-tfj8j	1/2	Running	1	2h

This is because once mTLS is enabled, Kubernetes is unable to perform network-based probes since Kubernetes probes can't perform mTLS against Istio services.

Task 2 - mTLS Friendly Deployment

There are a number of solutions to this:

- Set [mTLS to Permissive mode](#), so that Istio allows both mTLS and non-TLS connections.
- Use a local health checker program, e.g. `curl`, or your own executable, to check against the port.
- [Rewrite health checks to send to pilot agent](#) through automatic sidecar injector.
- Expose [health check endpoint on a different port](#) that doesn't require mTLS.

In this lab, we'll simply move all the health checks to local executable, `wget`.

Secondly, in order for each service to identify itself, each pod needs to be associated to a service account unique to the service. E.g., all of the Guestbook UI pods will be associated with the `guestbook-ui` service account.

1. Apply updated Kubernetes configuration files.

```
$ kubectl apply -f kubernetes-mtls/
```


2. Wait for all pods to come back healthy.

```
$ watch kubectl get pods
```

3. Delete all Istio rules and reinstate the Gateway bindings.

```
$ kubectl delete -f istio-rules/  
$ kubectl apply -f istio-rules/gateway.yaml  
$ kubectl apply -f istio-rules/guestbook-ui-vs.yaml
```

4. Validate that you can still visit the Guestbook UI from the browser.

```
$ export INGRESS_IP=$(kubectl -n istio-system get svc istio-ingressgateway \  
-o jsonpath='{.status.loadBalancer.ingress[0].ip}')  
$ curl -v $INGRESS_IP
```

That should be it!

Task 3 - Validating mTLS Connection

1. Create a new Namespace that doesn't have Istio automatic sidecar injection.

```
$ kubectl create ns noistio
```

2. Run a Toolbox Pod in the noistio namespace.

```
$ kubectl -n noistio run toolbox --image centos:7 \  
-l app=toolbox /bin/sh -- -c 'sleep 84600'
```

3. Wait for this pod to start.

```
$ watch kubectl get pods -n noistio
```

4. Find the Toolbox Pod Name and store that in the environmental variable.

```
$ export TB=$(kubectl -n noistio get po -l app=toolbox -o template --template '{{index  
.items 0).metadata.name}}')
```

5. Connect to Helloworld Service from the Toolbox that doesn't have Istio mTLS.

```
$ kubectl -n noistio exec -it $TB \  

```

```
curl -- http://helloworld-service.default.svc.cluster.local:8080/hello/Ray -k
```

```
curl: (56) Recv failure: Connection reset by peer  
command terminated with exit code 56
```

5. Execute curl from the `shell-demo` that you installed in the previous lab. Shell-demo is in the namespace that has Istio enabled.

```
$ kubectl exec -ti shell-demo \  
  curl -- http://helloworld-service.default.svc.cluster.local:8080/hello/Ray -k \  
  '{"hostname":"helloworld-service-v1-7b6755fb68-v9gwt","greeting":"Hello Ray from \  
  helloworld-service-v1-7b6755fb68-v9gwt with 1.0","version":"1.0"}'
```

6. It works in `shell-demo` because the Istio proxy is automatically configured with the certificates.

```
$ kubectl exec shell-demo -c istio-proxy -- ls /etc/certs  
cert-chain.pem  
key.pem  
root-cert.pem
```

7. Let's steal these certificates!

```
$ mkdir -p ~/certs  
$ FILES=(key.pem cert-chain.pem root-cert.pem)  
$ for file in ${FILES[@]}; \  
  do kubectl exec -c istio-proxy shell-demo /bin/cat \  
    -- /etc/certs/$file > ~/certs/$file; done  
$ ls ~/certs
```

8. Take a look at the certificate in more detail.

```
$ openssl x509 -in ~/certs/cert-chain.pem -text  
Certificate:  
  Data:  
    Version: 3 (0x2)  
    Serial Number:  
      49:bd:70:a6:aa:71:81:4d:22:c7:59:1c:0c:b3:00:ec  
    Signature Algorithm: sha256WithRSAEncryption  
    Issuer: O = k8s.cluster.local  
    Validity  
      Not Before: Oct 14 19:04:04 2018 GMT  
      Not After : Jan 12 19:04:04 2019 GMT  
    Subject: O =  
    Subject Public Key Info:  
      ...  
    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 Alternative Name:
    URI:spiffe://cluster.local/ns/default/sa/default
...
```

Notice the Subject Alternative Name. This name is tied to the service account associated with the pod. We didn't assign a service account to Shell Demo, so the service account name is default.

9. Let's steal the certificate from Guestbook UI.

```
$ FILES=(key.pem cert-chain.pem root-cert.pem)
$ POD=$(kubectl get pod -l app=guestbook-ui -o template --template \
'{{(index .items 0).metadata.name}}')
$ for file in ${FILES[@]}; \
do kubectl exec -c istio-proxy $POD /bin/cat \
-- /etc/certs/$file > ~/certs/$file; done
$ openssl x509 -in ~/certs/cert-chain.pem -text
...
    X509v3 Subject Alternative Name:
        URI:spiffe://cluster.local/ns/default/sa/guestbook-ui
...
```

10. Put the stolen certificates back into the Toolbox so that you can fake this connection.

```
$ for file in ${FILES[@]}; do kubectl cp $HOME/certs/$file noistio/$TB:$file; done
```

11. Make the connection with these certificates.

```
$ kubectl -n noistio exec -it $TB \
curl -- https://helloworld-service.default.svc.cluster.local:8080/hello/Ray \
-v --key ./key.pem --cert ./cert-chain.pem --cacert ./root-cert.pem -k
```

Task 4 - Istio RBAC Authorization

1. Turn on Istio RBAC for the `default` namespace.

```
$ kubectl apply -f istio-rules/istio-auth-rbac.yaml
```

2. Try accessing the Guestbook UI.

```
$ export INGRESS_IP=$(kubectl -n istio-system get svc istio-ingressgateway \
-o jsonpath='{.status.loadBalancer.ingress[0].ip}')
$ curl -v $INGRESS_IP
```

It should have failed with RBAC Access Denied.

3. Create a `viewer` role that has access to all `GET` methods.

```
$ kubectl apply -f istio-rules/istio-role-viewer.yaml
```

Note: See [a list of properties](#) you can use for authorization constraints.

4. Try accessing the Guestbook UI.

```
$ curl -v $INGRESS_IP
```

Not bad! Now you can read. But if you access the page from the browser, and attempted to `POST` a new message, it'll still fail because the RBAC rule only allows `GET` method.

4. Create a `webuser` role that has all access to `guestbook-ui`.

```
$ kubectl apply -f istio-rules/istio-role-webuser.yaml
```

Now you can `POST` to Guestbook UI, however, Guestbook UI cannot `POST` to Guestbook Service backend.

5. Finally, create a Guestbook Service Read/Write role and bind it to Guestbook UI.

```
$ kubectl apply -f istio-rules/istio-role-guestbook-service-rw.yaml
```