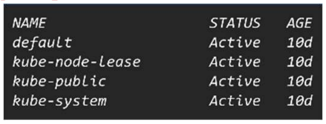
Service Mesh: Lab Sheet

Configure an Azure Kubernetes Cluster

Prerequisite: Azure Cluster is created and configured using at-least one node with 4 Core CPU and 8+ GB of RAM.

Connect to Azure Kubernetes cluster by logging to Azure CLI :

```
az account set –subscription < subscription id >
az aks get-credentials --resource-group --name <Azure Kubernetes cluster name>
kubectl config current-context
kubectl get nodes -o=wide
kubectl get namespaces
```



```
NAME              STATUS   AGE
default           Active   10d
kube-node-lease   Active   10d
kube-public       Active   10d
kube-system       Active   10d
```

Clone The Example Project “Invoice Manager” from GitHub

```
git clone https://github.com/AINULX00159358/InvoiceManager.git
cd InvoiceManager
```

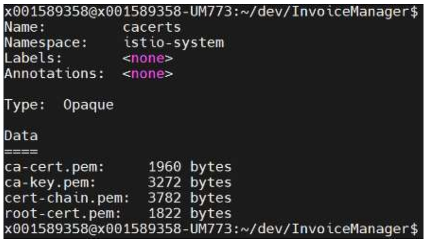
The example project “InvoiceManager” is bunch of 4 microservices, which generate invoice and process payment for the invoices. The 4 microservices are “data” which maintains a memory cache, a “generator” which creates an Invoice, “payment” which processes the payment, and “app” which is the front end for generating invoice and payment. Microservices talks to each other using HTTP REST interface.

Deploy Root CA Certificate for Istio Control Plane

Deploy Root CA Certificate for Istio to use. Istio will inject the Security Certificate for M-TLS service-to-service communication. The Security cert will use Root CA Certificate as Certificate Authority managed by ISTIO control plane. InvoiceManager project provides the Root CA Certificate and shell script to create namespace and deploy security certificates. Security Certs are in folder **certs**, while shell script is in **Istio/deploy-certs.sh**

The content of **deploy-certs.sh** is:

```
kubectl create namespace istio-system
kubectl create secret generic cacerts -n istio-system \
      --from-file=certs/ca-cert.pem \
      --from-file=certs/ca-key.pem \
      --from-file=certs/root-cert.pem \
      --from-file=certs/cert-chain.pem
kubectl describe secret cacerts -n istio-system
```



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$
Name:         cacerts
Namespace:    istio-system
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
ca-cert.pem:      1960 bytes
ca-key.pem:       3272 bytes
cert-chain.pem:   3782 bytes
root-cert.pem:    1822 bytes
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

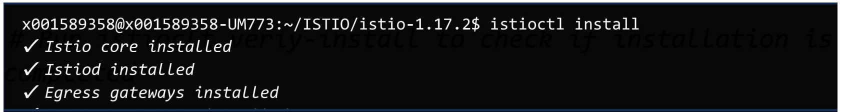
Install Istio

To install istio, *istioclt (a command line istio utility)* will be used. “istioctl” is obtained from Istio release, which can be downloaded by running.

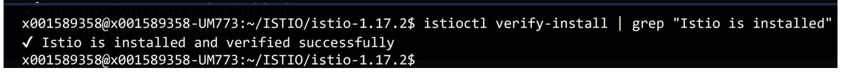
```
curl -L https://istio.io/downloadIstio | sh -   # download istio 1.18.0 to folder istio-1.18.0
cd istio-1.18.0
export PATH=$PWD/bin:$PATH
```

We can now run istio install command to install *istio* in the *istio-system* namespace, hosting the “control plane”.

```
istioctl install -y
```

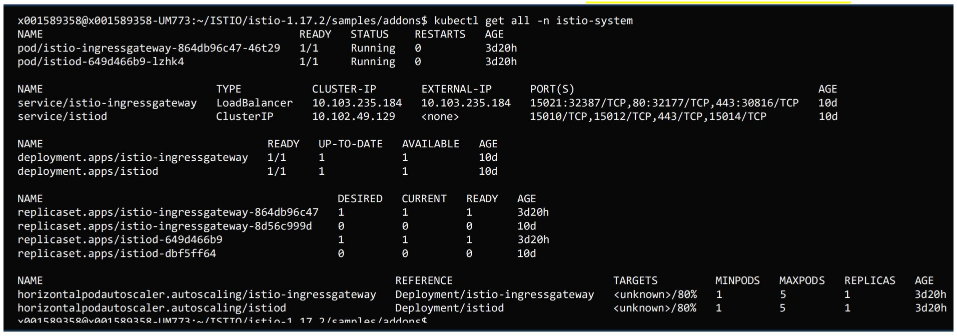


```
x001589358@x001589358-UM773:~/ISTIO/istio-1.17.2$ istioctl install
✓ Istio core installed
✓ Istiod installed
✓ Egress gateways installed
```



```
x001589358@x001589358-UM773:~/ISTIO/istio-1.17.2$ istioctl verify-install | grep "Istio is installed"
✓ Istio is installed and verified successfully
x001589358@x001589358-UM773:~/ISTIO/istio-1.17.2$
```

To look at deployment of istio control plane, run `kubectl get all -n istio-system`



```
x001589358@x001589358-UM773:~/ISTIO/istio-1.17.2/samples/addons$ kubectl get all -n istio-system
NAME                                        READY   STATUS    RESTARTS   AGE
pod/istio-ingressgateway-864db96c47-46t29   1/1     Running   0          3d20h
pod/istiod-649d466b9-lzhk4                  1/1     Running   0          3d20h

NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)                                      AGE
service/istio-ingressgateway   LoadBalancer   10.103.235.184   10.103.235.184   15021:32387/TCP,80:32177/TCP,443:30816/TCP   10d
service/istiod                 ClusterIP      10.102.49.129    <none>           15010/TCP,15012/TCP,443/TCP,15014/TCP        10d

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/istio-ingressgateway   1/1     1            1           10d
deployment.apps/istiod                 1/1     1            1           10d

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/istio-ingressgateway-864db96c47   1         1         1       3d20h
replicaset.apps/istio-ingressgateway-8d56c999d    0         0         0       10d
replicaset.apps/istiod-649d466b9                  1         1         1       3d20h
replicaset.apps/istiod-dbf5ff64                   0         0         0       10d

NAME                                                       REFERENCE                         TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/istio-ingressgateway   Deployment/istio-ingressgateway   <unknown>/80%   1         5         1          3d20h
horizontalpodautoscaler.autoscaling/istiod                 Deployment/istiod                 <unknown>/80%   1         5         1          3d20h
x001589358@x001589358-UM773:~/ISTIO/istio-1.17.2/samples/addons$
```

Label Namespaces with ISTIO injection enabled.

Istio control plane will inject Envoy Proxy as sidecar to Kubernets PODs which are deployed in a namespace that have a label **istio-injection=enabled**.

We will inject this label in “default” and “ui” namespace.

```
kubectl label namespace default istio-injection=enabled
kubectl create namespace ui
kubectl label namespace ui istio-injection=enabled
```

Add Service Account to both Default and UI namespaces.

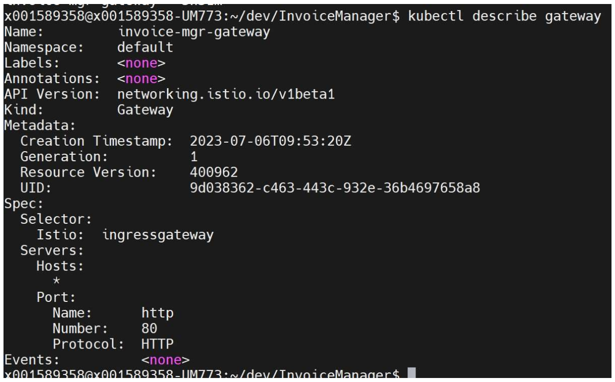
Invoice Manager provides a Service Account which will be added to both default and UI namespaces. All namespaces which will be controlled by Istio will need a service account. The Service account is available in folder **sa** in InvoiceManager source.

```
kubectl apply --namespace=default -f ./sa/invoice-mgr-sa.yaml
kubectl apply --namespace=ui -f ./sa/invoice-mgr-sa.yaml
```

Deploy Gateway

We will deploy gateway to default namespace which will handle all incoming traffic to configured virtual service and its destination. Gateway manifest is provided with source of InvoiceManager in the folder “gateway”

```
kubectl apply -f gateway/invoice-mgr-gtw.yaml
kubectl describe gateway
```



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ kubectl describe gateway
Name:         invoice-mgr-gateway
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  networking.istio.io/v1beta1
Kind:         Gateway
Metadata:
  Creation Timestamp:  2023-07-06T09:53:20Z
  Generation:          1
  Resource Version:    400962
  UID:                 9d038362-c463-443c-932e-36b4697658a8
Spec:
  Selector:
    Istio:  ingressgateway
  Servers:
    Hosts:
      *
    Port:
      Name:      http
      Number:    80
      Protocol:  HTTP
Events:          <none>
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

Deploy Microservice for Invoice Manager

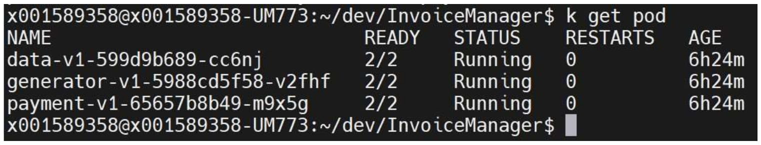
All Microservices for Invoice Manager example will be deployed using Kubernetes manifest files. These manifest files are in kubernetes folder, containing sub-folders for the microservices. Each folder have manifest files for deployment, service, destination rules, MTLS and virtual services.

To deploy use kubectl apply with command with “recursive” option.

```
kubectl apply -R -f kubernetes/data
kubectl apply -R -f kubernetes/generator
kubectl apply -R -f kubernetes/simulator
```

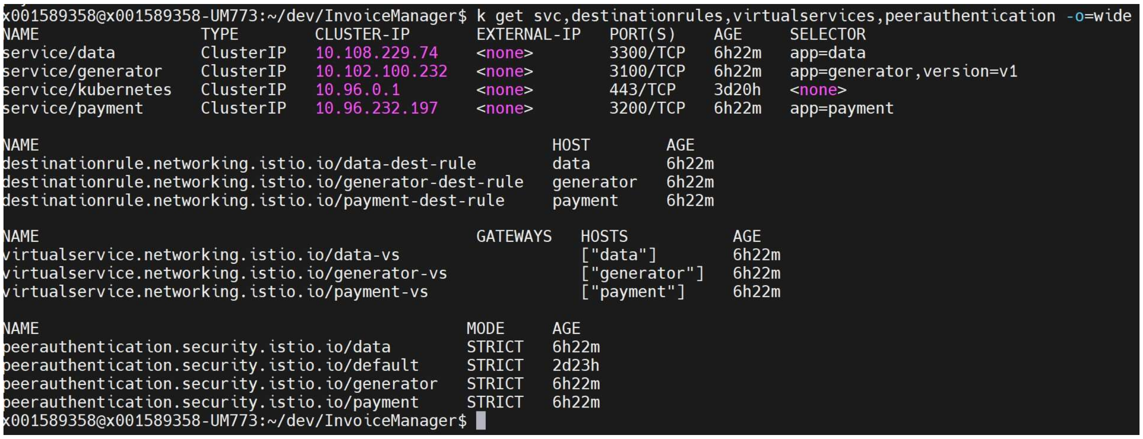
Run kubectl get all to get all deployed components in default namespace

```
kubectl get pod
```



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ k get pod
NAME                          READY   STATUS    RESTARTS   AGE
data-v1-599d9b689-cc6nj       2/2     Running   0          6h24m
generator-v1-5988cd5f58-v2fhf 2/2     Running   0          6h24m
payment-v1-65657b8b49-m9x5g   2/2     Running   0          6h24m
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

```
kubectl get svc,destinationrules,virtualservices,peerauthetication -o=wide
```



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ k get svc,destinationrules,virtualservices,peerauthentication -o=wide
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE     SELECTOR
service/data         ClusterIP   10.108.229.74    <none>        3300/TCP   6h22m   app=data
service/generator    ClusterIP   10.102.100.232   <none>        3100/TCP   6h22m   app=generator,version=v1
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    3d20h   <none>
service/payment      ClusterIP   10.96.232.197    <none>        3200/TCP   6h22m   app=payment

NAME                                                  HOST        AGE
destinationrule.networking.istio.io/data-dest-rule        data        6h22m
destinationrule.networking.istio.io/generator-dest-rule   generator   6h22m
destinationrule.networking.istio.io/payment-dest-rule     payment     6h22m

NAME                                          GATEWAYS   HOSTS           AGE
virtualservice.networking.istio.io/data-vs                 ["data"]        6h22m
virtualservice.networking.istio.io/generator-vs            ["generator"]   6h22m
virtualservice.networking.istio.io/payment-vs              ["payment"]     6h22m

NAME                                           MODE     AGE
peerauthentication.security.istio.io/data        STRICT   6h22m
peerauthentication.security.istio.io/default     STRICT   2d23h
peerauthentication.security.istio.io/generator   STRICT   6h22m
peerauthentication.security.istio.io/payment     STRICT   6h22m
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

Deploy Main Application and connect to Gateway.

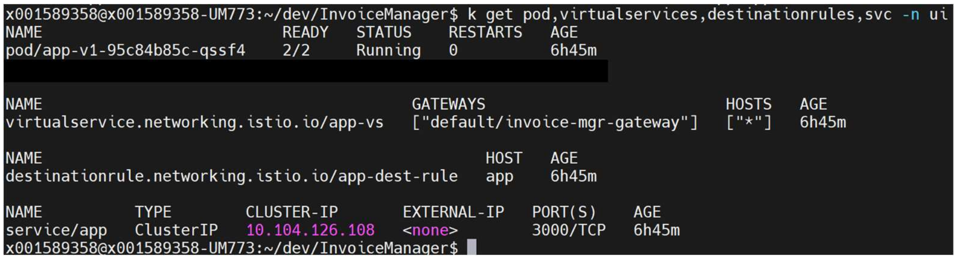
The Invoice Manager provides the frontend application which will accept REST calls on the public IP which will travel from Istio Gateway to application. The application will be deployed in UI namespace. It will use the Gateway installed in the previous steps when defining the virtual services. To the external traffic can come form gateway to virtual services and the to destination which will then map to the service end point.

There is two versions of app, version 1 and canary version 2. The manifests for version 1 is in folder kubernetes/app while manifests for version 2 is in Kubernetes/app-v2 folder.

To Deploy the version 1 of application:

```
kubectl apply -R -f kubernetes/app
```

```
kubectl get pods,virtualservices,destinationrules,svc -n ui
```



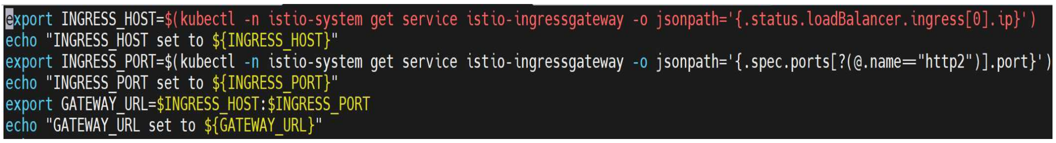
```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ k get pod,virtualservices,destinationrules,svc -n ui
NAME                         READY   STATUS    RESTARTS   AGE
pod/app-v1-95c84b85c-qssf4   2/2     Running   0          6h45m
[redacted]

NAME                                        GATEWAYS                          HOSTS   AGE
virtualservice.networking.istio.io/app-vs   ["default/invoice-mgr-gateway"]   ["*"]   6h45m

NAME                                               HOST   AGE
destinationrule.networking.istio.io/app-dest-rule   app    6h45m

NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/app   ClusterIP   10.104.126.108   <none>        3000/TCP   6h45m
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

To get the URL of the Gateway run the following command.



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

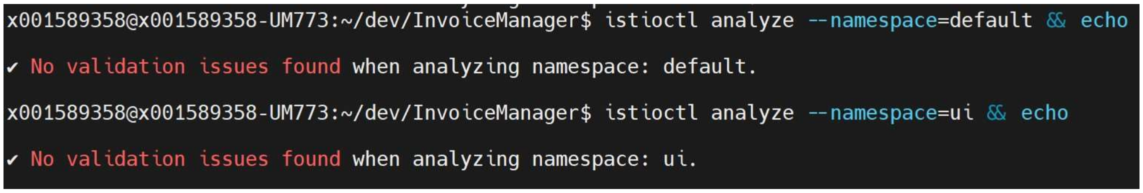
The GATEWAY_URL exposed will be the Public URL which is provided by Azure Service load balancer. From external we can call the public IP which will connect to app and return Application Name and version.

```
curl ${GATWAY_URL} && echo
```

```
Invoice app version 1.0.0
```

Analyse Istio connection

To analyse if all configuration is correct, we can run istoctl analyze command on both default and ui namespaces.



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ istioctl analyze --namespace=default && echo
✔ No validation issues found when analyzing namespace: default.
x001589358@x001589358-UM773:~/dev/InvoiceManager$ istioctl analyze --namespace=ui && echo
✔ No validation issues found when analyzing namespace: ui.
```

Run Integration Test

Prerequisite: NodeJS and NPM should be installed

Invoice Manager also provides Integration test using nodejs Cypress framework. It will generate invoice and process payment, hitting all microservices endpoints and generating traffic.

To run the test type the following command from Invoice Manager source folder.

```
npm install  # we need to run this once only
```

```
CYPRESS_APPURL=${GATEWAY_URL} CYPRESS_TIMES=50 npx cypress run
```

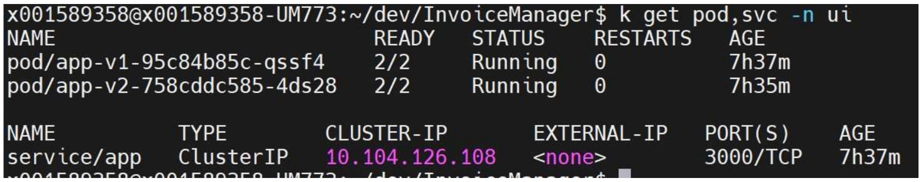
```
# CYPRESS_APPURL ➔ URL of the Gateway (or Public IP)
# CYPRESS_TIMES  ➔ Number of Invoices that will be generated
```

This test will pass with one failure. This is because Application version 2 is not yet deployed.

Deploying Canary Version and Split Traffic

We can now deploy Canary version (i.e. version 2) and Split Traffic to 50% between version 1 and 2 Run the following command to deploy application version 2. Only new version of the “app” microservice in “ui” namespace will be deployed. This deployment will override the virtualservices and destination rules to split traffic.

```
kubectl apply -R -f kubectl/app-v2
kubectl get pods,virtualservices,destinationrules,svc -n ui
```



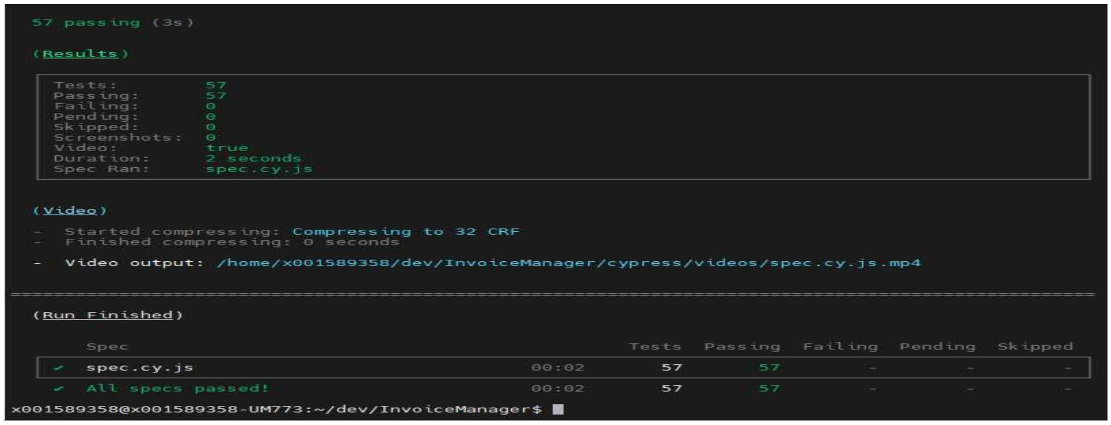
```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ k get pod,svc -n ui
NAME                              READY   STATUS    RESTARTS   AGE
pod/app-v1-95c84b85c-qssf4        2/2     Running   0          7h37m
pod/app-v2-758cddc585-4ds28       2/2     Running   0          7h35m

NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/app   ClusterIP   10.104.126.108   <none>        3000/TCP   7h37m
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

Rerun then Cypress Integration Tests

```
CYPRESS_APPURL=${GATEWAY_URL} CYPRESS_TIMES=150 npx cypress run
```

All the test is expected to pass



```
  57 passing (3s)

  (Results)

  ┌──────────────────────────────────────────────────────────────────────┐
  │ Tests:        57                                                     │
  │ Passing:      57                                                     │
  │ Failing:      0                                                      │
  │ Pending:      0                                                      │
  │ Skipped:      0                                                      │
  │ Screenshots:  0                                                      │
  │ Video:        true                                                   │
  │ Duration:     2 seconds                                              │
  │ Spec Ran:     spec.cy.js                                             │
  └──────────────────────────────────────────────────────────────────────┘


  (Video)

  -  Started compressing: Compressing to 32 CRF
  -  Finished compressing: 0 seconds

  -  Video output: /home/x001589358/dev/InvoiceManager/cypress/videos/spec.cy.js.mp4


====================================================================================

  (Run Finished)


       Spec                                  Tests  Passing  Failing  Pending  Skipped
  ┌──────────────────────────────────────────────────────────────────────────────────┐
  │ ✓  spec.cy.js                  00:02       57       57        -        -        - │
  └──────────────────────────────────────────────────────────────────────────────────┘
    ✓  All specs passed!           00:02       57       57        -        -        -
x001589358@x001589358-UM773:~/dev/InvoiceManager$ █
```

Deploy Observatory Tools

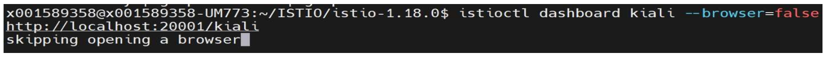
Istio Provide Observatory tools to capture and present telemetric, performance and configuration information about Istio control plane and Envoy. The tools we will deploy is Prometheus and Kiali. The Deployment manifest file is provided in istio release folder, download when deploying Istio. These manifest files are in **istio-1.18.0/samples/addon** folder.

Run the following commands to deploy Prometheus and Kiali

```
cd istio-1.18.0/samples/addon
kubectl apply -f prometheus.yaml
kubectl apply -f kiali.yaml
```

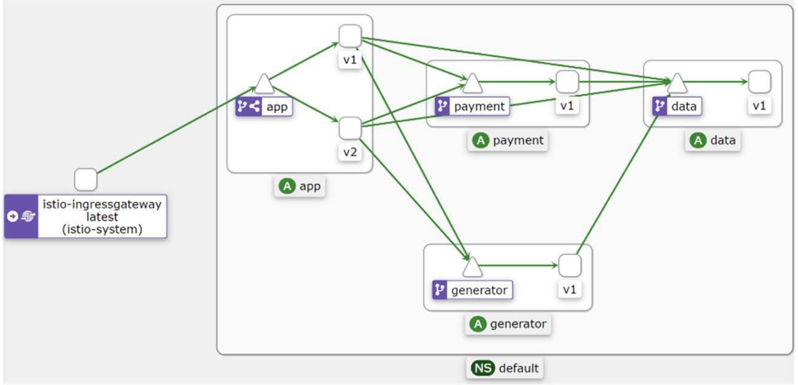
To start the dashboard istioctl will port-forward Kiali

```
istioctl dashboard kiali --browser=false
```



```
x001589358@x001589358-UM773:~/ISTIO/istio-1.18.0$ istioctl dashboard kiali --browser=false
http://localhost:20001/kiali
skipping opening a browser█
```

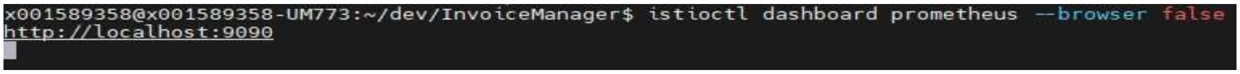
You can now start the Brower on Port 20001 to view Kiali Dashboard



The figure shows Kiali Dashboard traffic version view showing traffic Split between both va1 and v2. It also shows other microservices like payment, generator and data processing the call. External Traffic enters through isto-igressgatway.

To Run Grafana we will port-forward Prometheus from istioctl. We will run Grafana locally in our system connecting to Prometheus port-forwarded by istioctl.

To port-forward Prometheus to port 9090



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ istioctl dashboard prometheus --browser false
http://localhost:9090
█
```

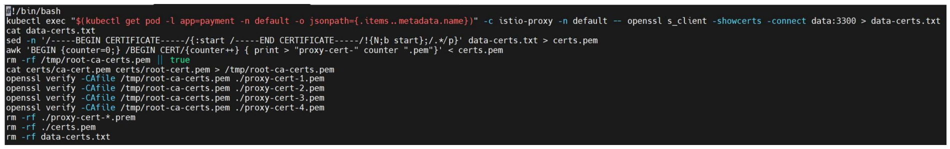
|

Verify m-TLS security certs between Service-To-Service calls.

Mutual TLS is enabled between service-to-service calls. This is configured when deploying the microservices. The manifest files is present in kubernets/data/mtls.yaml file. The Manifest states the TLS policy to be STRICT which means communication between the services will be using m-TLS with security certs signed by Root CA of Istio Control Plane.

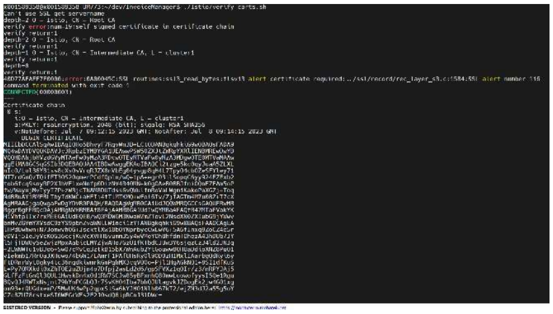
Invoice Manager provide script Istio/verify-carts.sh, which can be executed to verify the security certs used for communication is signed by Isto Control Plane ROOT cert.

The content of the file if attached below.

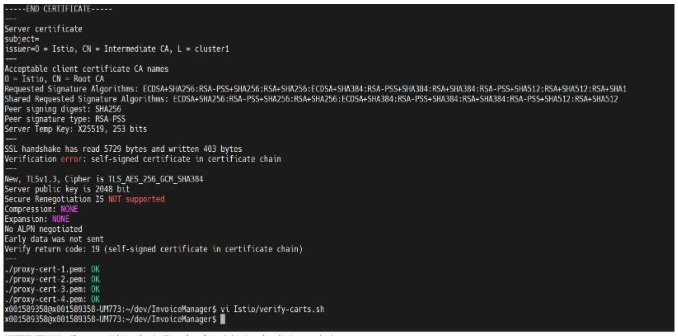


```
#!/bin/bash
kubectl exec "$(kubectl get pod -l app=payment -n default -o jsonpath={.items..metadata.name})" -c istio-proxy -n default -- openssl s_client -showcerts -connect data:3300 > data-certs.txt
cat data-certs.txt
sed -n '/-----BEGIN CERTIFICATE-----/{:start /-----END CERTIFICATE-----/!{N;b start};/.*/p}' data-certs.txt > certs.pem
awk 'BEGIN {counter=0;} /BEGIN CERT/{counter++} { print > "proxy-cert-" counter ".pem"}' < certs.pem
rm -rf /tmp/root-ca-certs.pem || true
cat certs/ca-cert.pem certs/root-cert.pem > /tmp/root-ca-certs.pem
openssl verify -CAfile /tmp/root-ca-certs.pem ./proxy-cert-1.pem
openssl verify -CAfile /tmp/root-ca-certs.pem ./proxy-cert-2.pem
openssl verify -CAfile /tmp/root-ca-certs.pem ./proxy-cert-3.pem
openssl verify -CAfile /tmp/root-ca-certs.pem ./proxy-cert-4.pem
rm -rf ./proxy-cert-*.prem
rm -rf ./certs.pem
rm -rf data-certs.txt
```

Script will connect to "payment" microservice connect call "data" microservice using "openssl" command with "showcert" option. This will print all the certs used. We split the certs and verify it with Root CA certs provided in Invoice Manager. This same cert is deployed in Istio Control plane before deploying istio in the cluster.



```
x001589358@x001589358-UM773:~/dev/InvoiceManager$ ./Istio/verify-carts.sh
Can't use SSL_get_servername
depth=2 O = Istio, CN = Root CA
verify error:num=19:self signed certificate in certificate chain
verify return:1
depth=2 O = Istio, CN = Root CA
verify return:1
depth=1 O = Istio, CN = Intermediate CA, L = cluster1
verify return:1
depth=0
verify return:1
4BD77AFAFF7F0000:error:0A00045C:SSL routines:ssl3_read_bytes:tlsv13 alert certificate required:../ssl/record/rec_layer_s3.c:1584:SSL alert number 116
command terminated with exit code 0
CONNECTED(00000003)
---
Certificate chain
 0 s:
   i:O = Istio, CN = Intermediate CA, L = cluster1
   a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
   v:NotBefore: Jul  7 09:12:15 2023 GMT; NotAfter: Jul  8 09:14:15 2023 GMT
-----BEGIN CERTIFICATE-----
[illegible]
```



```
-----END CERTIFICATE-----
---
Server certificate
subject=
issuer=O = Istio, CN = Intermediate CA, L = cluster1
---
Acceptable client certificate CA names
O = Istio, CN = Root CA
Requested Signature Algorithms: ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512:RSA+SHA1
Shared Requested Signature Algorithms: ECDSA+SHA256:RSA-PSS+SHA256:RSA+SHA256:ECDSA+SHA384:RSA-PSS+SHA384:RSA+SHA384:RSA-PSS+SHA512:RSA+SHA512
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 5729 bytes and written 403 bytes
Verification error: self-signed certificate in certificate chain
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 19 (self-signed certificate in certificate chain)
---
./proxy-cert-1.pem: OK
./proxy-cert-2.pem: OK
./proxy-cert-3.pem: OK
./proxy-cert-4.pem: OK
x001589358@x001589358-UM773:~/dev/InvoiceManager$ vi Istio/verify-carts.sh
x001589358@x001589358-UM773:~/dev/InvoiceManager$
```

The certs are verified using OpenSSL verify command **`openssl verify -CAfile /tmp/root-ca-certs.pem ./proxy-cert-1.pem.`** The should return "OK" is the proxy cert is signed by root-ca-cert.

References:

- Istio: https://istio.io/latest/docs/setup/getting-started
- Git Hub Example Project: https://github.com/AINULX00159358/InvoiceManager.git
- Azure Cluster: https://learn.microsoft.com/en-us/azure/aks/learn/quick-kubernetes-deploy-portal
- Kiali: https://istio.io/latest/docs/ops/integrations/kiali
- Istio Security: https://istio.io/latest/docs/concepts/security/#authentication