IMPLEMENTATION OF: “ISTIO” (MESH SERVICES) & “IBM APP CONNECT12” CONTAINERIZED.

This document focuses on the ***PROCEDURE*** for the implementation of ***SERVICE MESH*** with ***ISTIO***, applied to ***IBM APP CONNECT v12***. The ***INDEX*** proposed will be the following:

I. DEFINITIONS.
II. INSTALLATION & CONFIGURATION.

- ✓ ***INSTANCE CREATION: "SERVICE MESH".***
- ✓ ***INSTANCE CREATION: "IBM APP CONNECT v12".***

III. MANAGEMENT OF: DASHBOARDS.
IV. DEPLOYMENT STRATEGIES & VERSION CONTROL.

- ✓ ***DEMO #1: (CANARY RELEASE).***
- ✓ ***DEMO #2: (BLUE & GREEN DEPLOYMENT).***
- ✓ ***DEMO #3: (TRAFFIC CONTROL: “INGRESS GATEWAY”).***
- ✓ ***DEMO #4: (UNIFIING EVERYTHING).***

DESCRIPTION	DETAIL
I. DEFINITIONS:	

WHAT IS SERVICE MESH?:

It is a layer of ***INFRASTRUCTURE*** dedicated which can be added to our ***APPLICATIONS***, with the aim of achieving: ***OBSERVABILITY, MANAGEMENT TRAFFIC, SECURITY.***

WHAT IS ISTIO?:

Currently, there are several ***IMPLEMENTATIONS of SERVICE MESH*** which the most important currently is: **ISTIO**.

It's a ***SERVICE MESH*** OpenSource that transparently overlays the ***DISTRIBUTED APPLICATIONS***. In addition, **ISTIO** provides:

- ***TRACEABILITY*** *of the* ***MICROSERVICES*** *& possible* ***DEPENDENCIES*** *among them.*
- *Secure communication between* ***MICROSERVICES*** *in an encrypted cluster* ***TLS****, authenticating & authorizing.*
- *Load balancing for* ***TRAFFIC: HTTP, TCP, WebSocket****, etc.*
- *Detailed control of* ***TRAFFIC****.*
- *Metrics, logs & tracking* ***TRAFFIC****: (entrance exit).*
- *Settings* ***PROXIES*** (**SIDECARs**)*, NOT embedded in the* ***MICROSERVICES*** (***OpenTracing***)*.*

IMPORTANT: *"Know that the idea is that the* ***MICROSERVICES handle only BUSINESS LOGIC*** *& that everything* ***ADDITIONAL as: treatment of*** *security protocols, etc., is managed by it:* ***SERVICE MESH****".*

Comparing Service Meshes

Feature	Istio	Linkerd	Consul Connect
Traffic Redirection (Blue/Green deployment)	Yes	No	No
Traffic Splitting (Canary deployment)	Yes	No	No
Attribute-based routing	Yes	No	No
Service Identification	Yes	No	Yes
Auto-proxy Injection	Yes	Yes	Yes
Non-Admin Installation	Yes	Yes	No
Built-in Dashboard	Yes	Yes	No
Certificate Management	Yes	No	Yes
Metrics Collection	Yes	Yes	No
Built-in Dashboard	Yes	Yes	No
TLS	Yes	Yes	Yes
External Service Support	Yes	No	Yes
Rate Limiting	Yes	No	No
Tracing	Yes	No	No

WHAT IS THE ISTIO ARCHITECTURE LIKE:

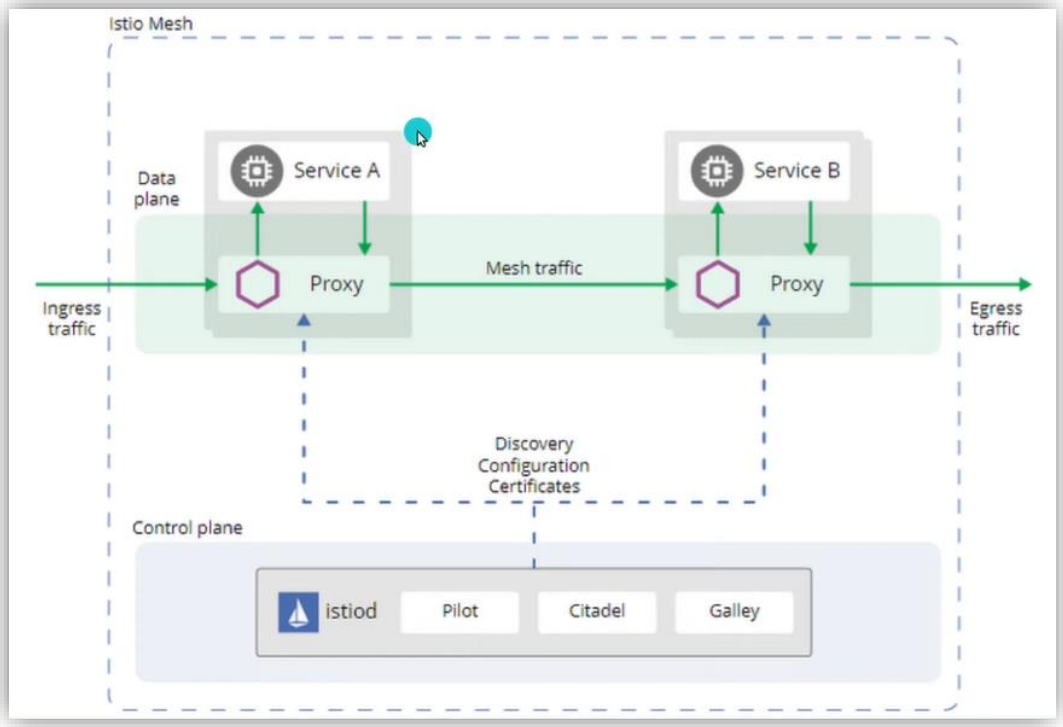
The ***COMPONENTS*** that drives **ISTIO** are:

- ✓ **PLANE CONTROL:** This is the one in charge of managing & configuring the ***PROXIES*** for the ***ROUTING*** of the ***TRAFFIC*** based on the ***RULES*** defined.

- ✓ **DATA PLAN:** Composed of a set of ***PROXIES*** smart **(SEND)** created in ***CONTAINER*** & displayed as **SIDECARs** in the ***PODs*** These control the entire ***COMMUNICATION*** between the ***MICROSERVICES*** & collect **TELEMETRY** of all the ***TRAFFIC*** of the ***SERVICE MESH***.

TELEMETRY is a characteristic of ***OBSERVABILITY*** that provides **ISTIO**, for ***NOTICE*** the behavior of the ***MICROSERVICES***:

- ✓ *Metrics (**LATENCY, ERRORS, SATURATION**).*
- ✓ *Distributed Traces (**TRAFFIC FLOW of MICROSERVICES**).*



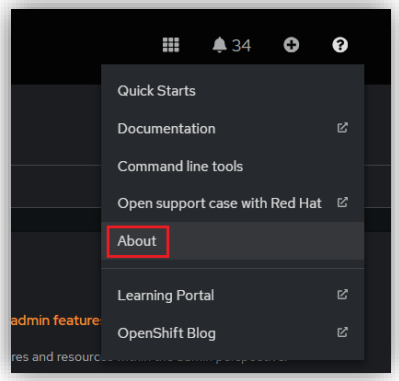
II. INSTALLATION & SETUP:

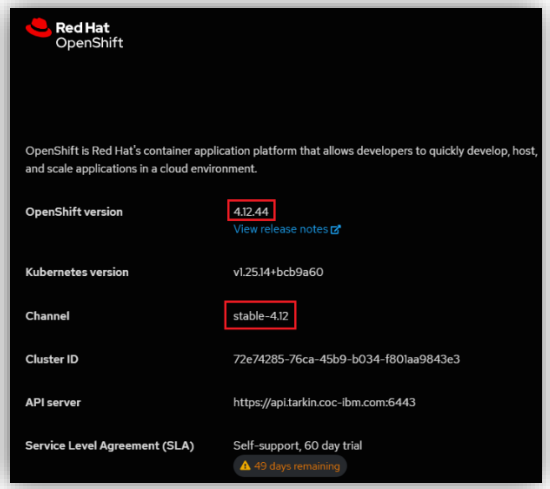
Initially you need to ***VALIDATE*** the version of **OPENSHIFT** that is installed.

In this case it ***VISUALIZE***:

- ✓ ***VERSION:****4.12.44*
- ✓ ***CHANNEL:****stable-412*

IMPORTANT: *"It must be considered that **INSTIO** supports the **OPENSHIFT** with the latest versions".*





Then, we proceed to create the 2 ***NAMESPACES*** called:

$ oc create ns istio-system
$ oc create ns ace-istio

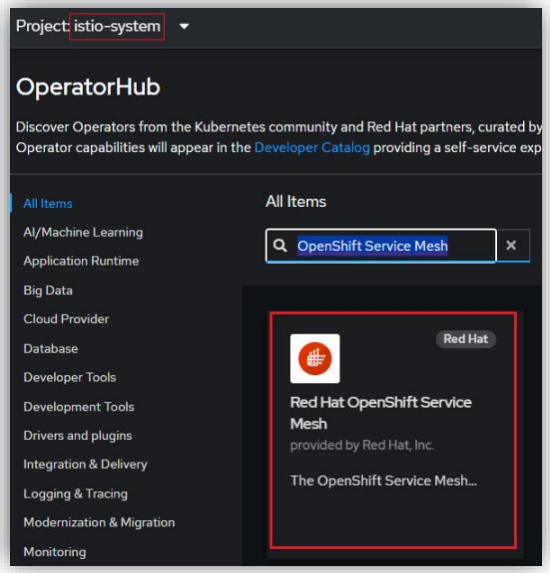
```
AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ oc create ns istio-system
namespace/istio-system  created

AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ oc create ns ace-istio
namespace/ace-istio  created
```

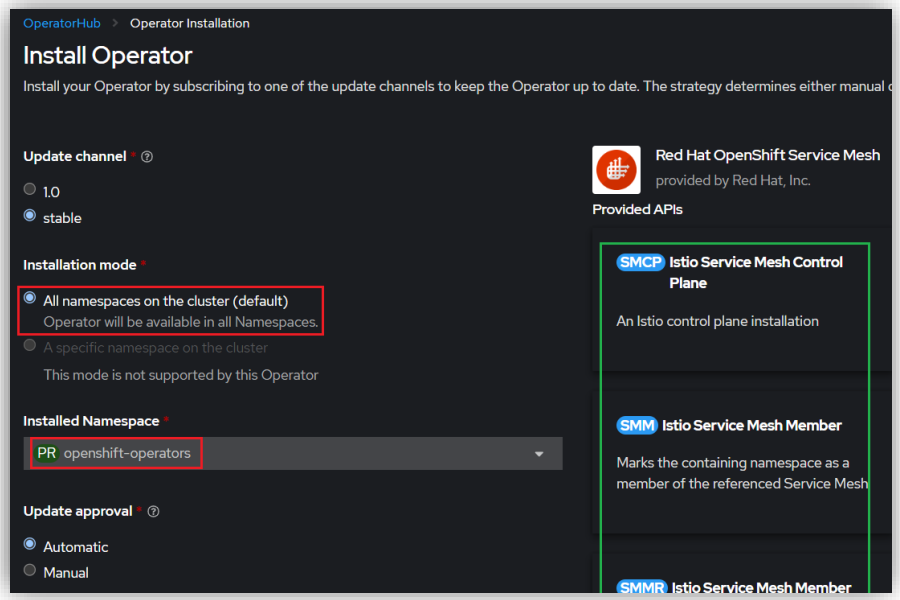
Then, we proceed to create the ***OPERATORS*** associates to the ***NAMESPACE***: *istio-system*.

First the ***OPERATOR*** called:

- ✓ ***NAME:*** *OpenShift Service Mesh*
- ✓ ***VERSION:*** *2.4.5-0*
- ✓ ***INSTALLATION MODE:*** *All Namespaces on the cluster*

IMPORTANT: *"The **INSTALLATION** may be accessed from all the **NAMESPACES**, but where it will mainly be installed will be in the **NAMESPACE**:* openshift-operators*".*

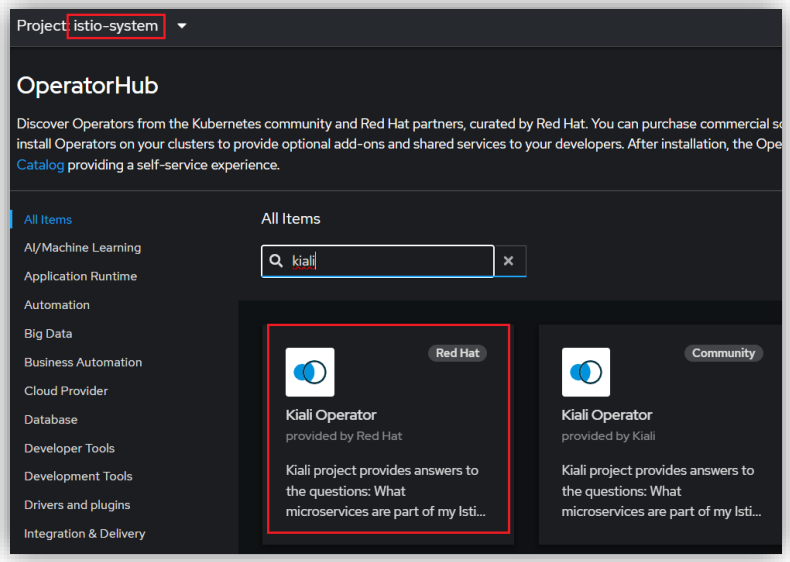


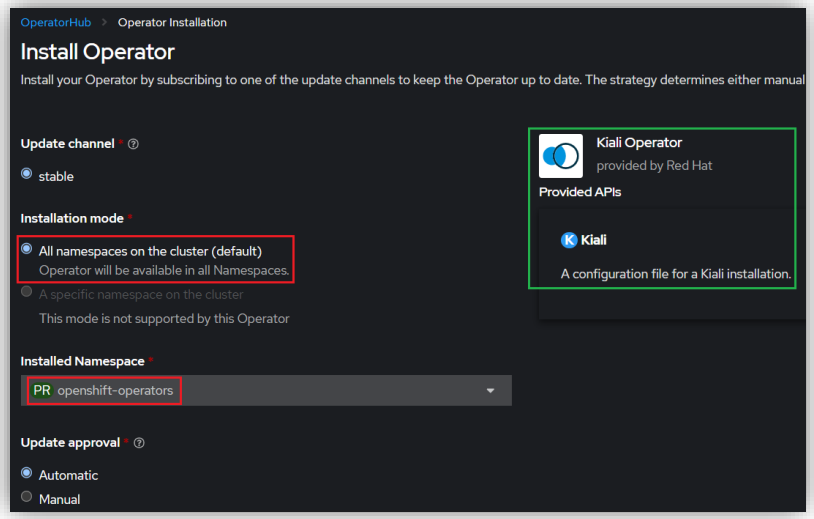


Then, inside the **NAMESPACE**: *istio-system*, proceeds to install the ***OPERATOR*** called:

- ✓ ***NAME:*** *Kiali Operator*
- ✓ ***VERSION:*** *1.65.11*
- ✓ ***INSTALLATION MODE:*** *All Namespaces on the cluster*

IMPORTANT: *"The **INSTALLATION** may be accessed from all the **NAMESPACES**, but where it will mainly be installed will be in the **NAMESPACE** :* openshift-operators*".*

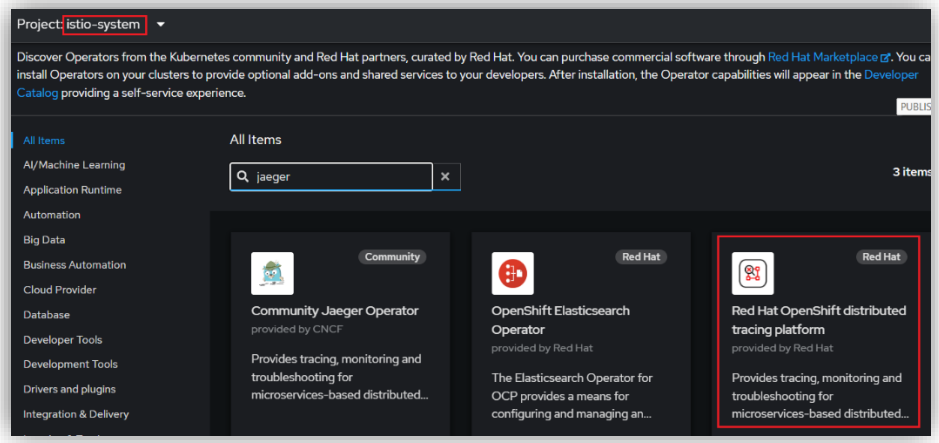


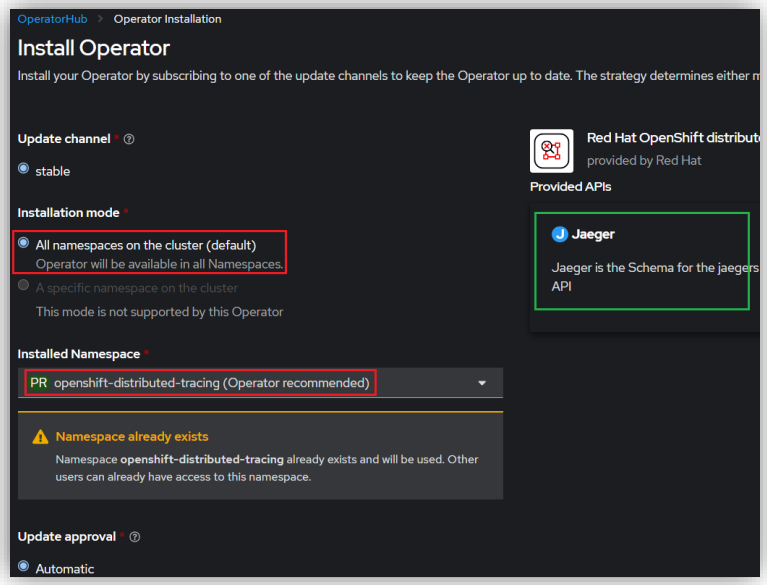


Then, inside the ***NAMESPACE***: *istio-system*, proceeds to install the ***OPERATOR*** called:

- ✓ ***NAME:*** *Red Hat Openshift distributed tracing platform*
- ✓ ***VERSION:*** *1.47.1-5*
- ✓ ***INSTALLATION MODE:*** *All Namespaces on the cluster*

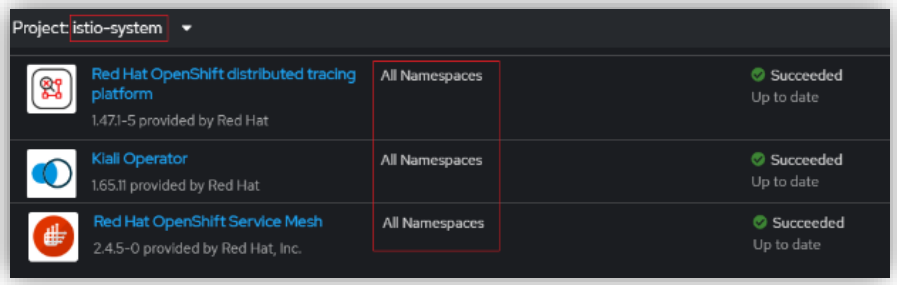
IMPORTANT: *"The **INSTALLATION** may be accessed from all the **NAMESPACES**, but where it will mainly be installed will be in the **NAMESPACE**:* openshift-distributed-tracing*".*





Then, we proceed to validate the ***OPERATORS*** installed within the ***NAMESPACE***: *istio-system* are:

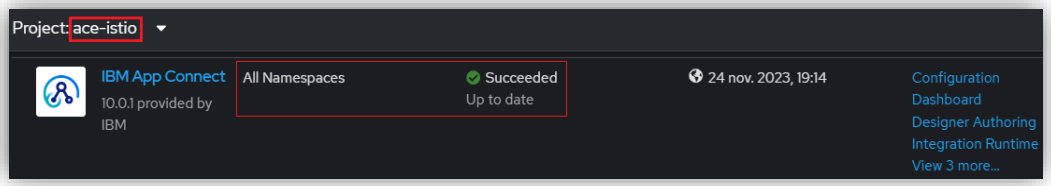
- *OpenShift Service Mesh*.
- *Kiali Operator*.
- *Red Hat Openshift distributed tracing platform*.



Then, we proceed to create the ***OPERATORS NAMESPACE*** associated: *ace-istio*.

First the ***OPERATOR*** called:

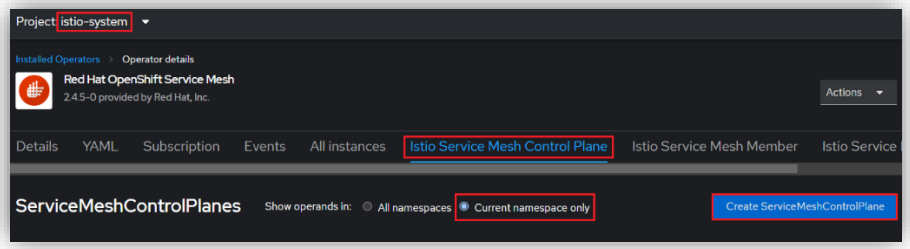
- ✓ ***NAME:*** *IBM App Connect*
- ✓ ***VERSION:*** *10.0.1*



A. CREATION OF INSTANCES:"SERVICE MESH"

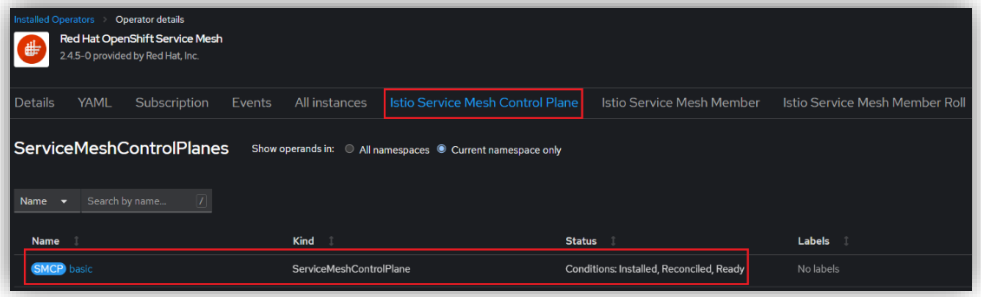
You proceed to enter the ***NAMESPACE***: ***istio-system*** & to the ***OPERATOR***: **Red Hat Openshift Service Mesh** & on the tab: ***Istio Service Mesh Control Plane***, we proceed to create an ***INSTANCE*** of the ***RESOURCE***: **ServiceMeshControlPlane**:

```
apiVersion: maistra.io/v2
kind: ServiceMeshControlPlane
metadata:
  name: basic
  namespace: istio-system
spec:
  addons:
    grafana:
      enabled: true
    jaeger:
      install:
        storage:
          type: Memory
    kiali:
      enabled: true
    prometheus:
      enabled: true
  policy:
    type: Istiod
  profiles:
    - default
```



```
telemetry:
  type: Istiod
tracing:
  sampling: 10000
  type: Jaeger
version: v2.4
```

IMPORTANT: *"We must consider that the **STATE** must show itself as shown in the picture: **Installed, Reconciled, Ready**".*

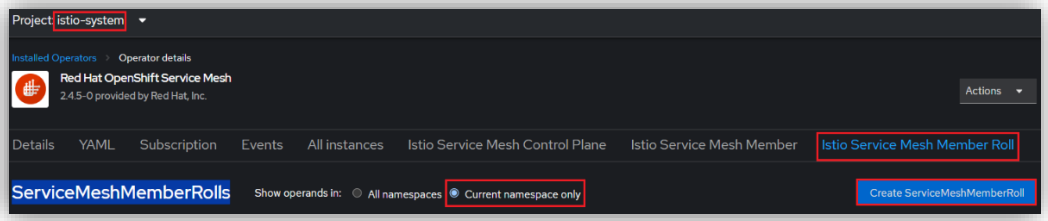


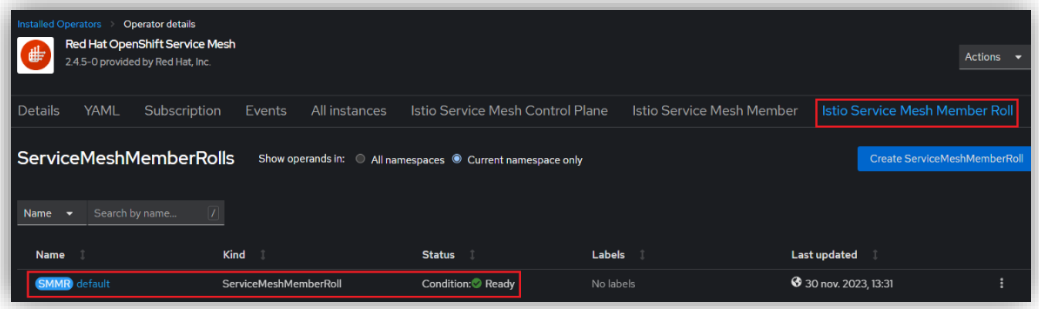
Then, within the same **OPERATOR**: **Red Hat Openshift Service Mesh** It is entered in the tab: ***Istio Service Mesh Member Roll*** & proceed to create a ***INSTANCE*** of the **RESOURCE**: **ServiceMeshMemberRoll**:

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    - ace-istio
```

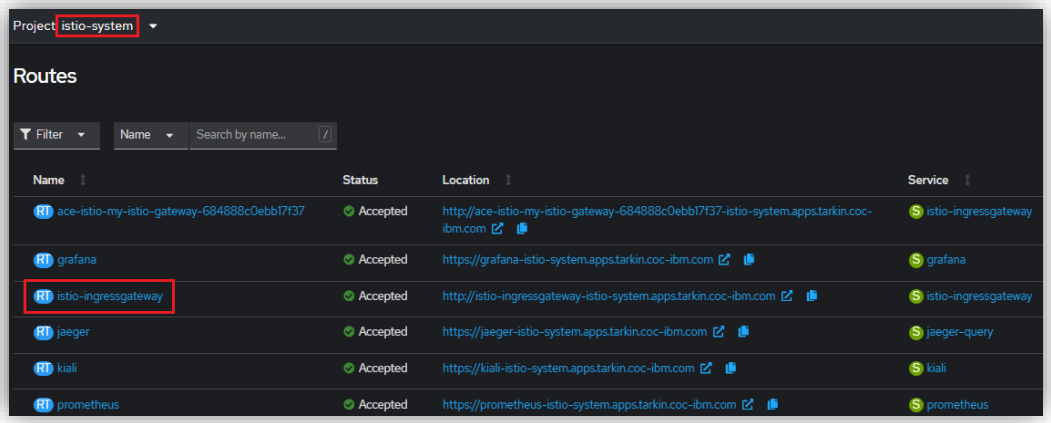
Here it will be done ***REFERENCE*** to the ***NAMESPACES*** that is required That ***INJECT ISTIO***, in this case where they are installed ***MICROSERVICES*** in ***ACE12***.

IMPORTANT: *"We must consider that the **STATE** must be shown as shown in the **IMAGE**: **Ready**".*





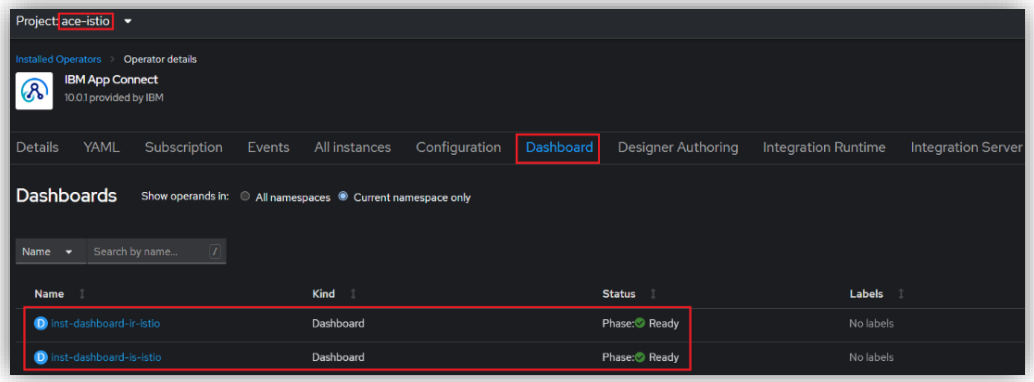
Finally, it must be considered that it must be ***VALIDATED*** that the **CREATION** of the***2*** ***INSTANCES*** in the ***OPERATOR***: **Red Hat Openshift Service Mesh**, will generate the ***ROUTEs*** in the ***IMAGE***, of which the most important for the test is he: istio-ingressgateway.



B. CREATION OF INSTANCES: "IBM APP CONNECT v12"

First of all, you should consider ***CREATE*** **2** **DASHBOARDS**, one for each ***MODALITY*** managed:

- ✓ ***INTEGRATION-SERVER:*** *inst-dashboard-**es**-istio*
- ✓ ***INTEGRATION-RUNTIME:*** *inst-dashboard-**go**-istio*



Then, proceed to enter the ***NAMESPACE***: ***ace-istio*** & to the ***OPERATOR***: **IBM App Connect**& is entered in the tab: ***Integration Server*** & we proceed to create an ***INSTANCE*** of the ***RESOURCE***: **IntegrationServer**:

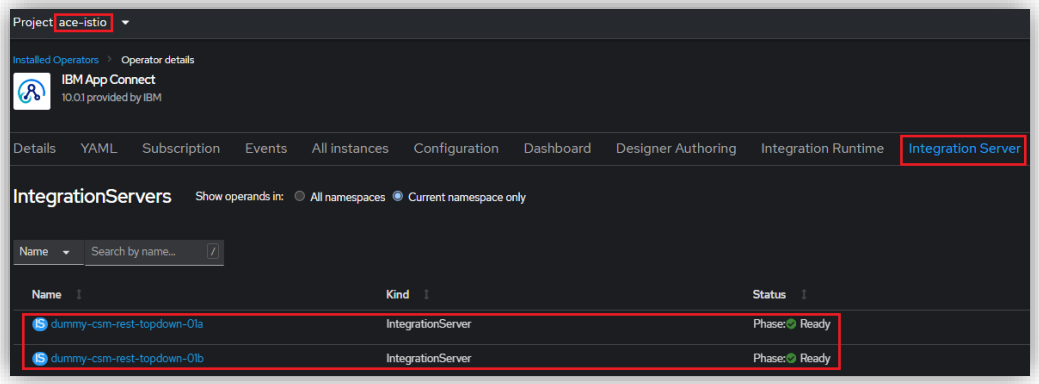
```
apiVersion: appconnect.ibm.com/v1beta1
kind: IntegrationServer
metadata:
  name: dummy-csm-rest-topdown-01a
  namespace: ace-istio
spec:
  enableMetrics: true
  license:
    accept: true
    license: L-SEWB-GH63KR
    use: CloudPakForIntegrationNonProductionFREE
  annotations:
    sidecar.istio.io/inject: 'true'
  pod:
    containers:
      runtime:
        resources:
          limits:
            cpu: 300m
            memory: 368Mi
          requests:
            cpu: 300m
            memory: 368Mi
  adminServerSecure: true
  router:
    timeout: 120s
  designerFlowsOperationMode: disabled
  createDashboardUsers: true
  service:
    endpointType: http
  version: '12.0'
  replicas: 1
  barURL: >-
    https://inst-dashboard-is-istio-dash:3443/v1/directories/dummy_csm_rest_topdown?2a2ccba2-2125-462e-927b-ee51c51bf849
  configurations: []
  labels:
    sidecar.istio.io/inject: 'true'
```

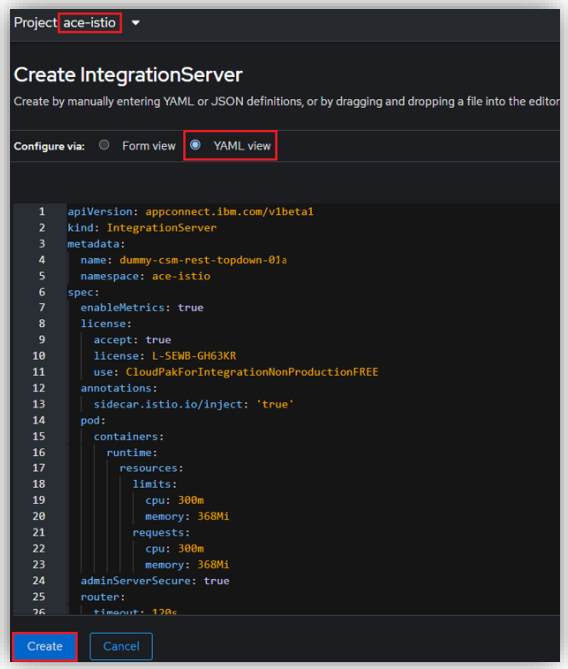
In this case, the deployment of the ***MICROSERVICE*** in the **IntegrationServer** will be done 2 times *(to have 2 MICROSERVICES deployed)*.

These are:

- ✓ *dummy-csm-rest-topdown-**01a***
- ✓ *dummy-csm-rest-topdown-**01b***

Do not forget in the Script the considerations of the ***LABEL*** and/or the ***ANNOTATION*** with the: ***sidecar.istio.io/inject: 'true'***, which is that ***WILL INJECT*** the functionality of ***ISTIO***.





IMPORTANT: *"The bar **URL** must be associated with the **ENDPOINT** of the **COMPILED**.bar, existing in the **DASHBOAR** of type: **INTEGRATION-SERVER**".*	

Then, proceed to enter the ***NAMESPACE:*** ***ace-istio*** in the ***OPERATOR:*** **IBM App Connect**& the tab is selected: ***Integration Runtime*** & we proceed to create an ***INSTANCE*** of the ***RESOURCE:*** **IntegrationRuntime**:

```
apiVersion: appconnect.ibm.com/v1beta1
kind: IntegrationRuntime
metadata:
  name: dummy-csm-rest-topdown-01a-ir
  namespace: ace-istio
spec:
  license:
    accept: true
    license: L-SEWB-GH63KR
    use: CloudPakForIntegrationNonProductionFREE
  replicas: 1
  template:
    spec:
      containers:
        - name: runtime
          resources:
            requests:
              cpu: 300m
              memory: 368Mi
    metadata:
      labels:
        sidecar.istio.io/inject: 'true'
  version: '12.0'
  barURL:
    - >-
      https://inst-dashboard-ir-istio-dash.ace-istio:3443/v1/directories/dummy_csm_rest_topdown?0333a2a5-bcbb-4417-b212-b238fc48934d
```

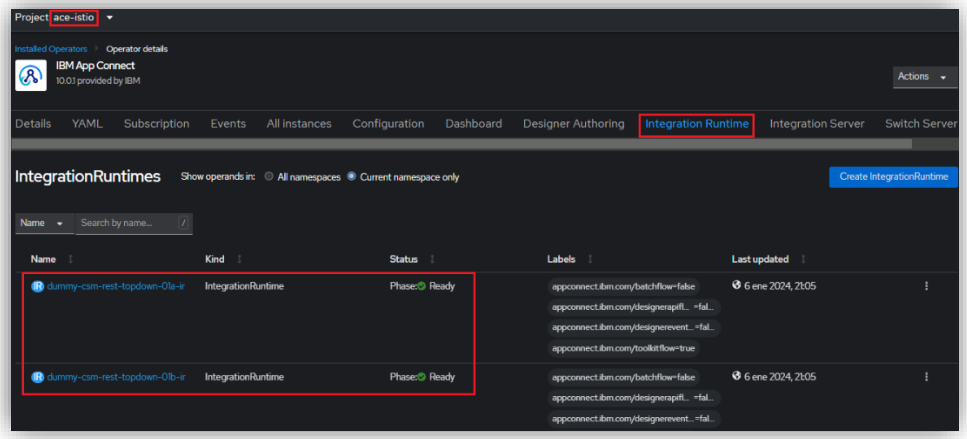
In this case, the deployment of the ***MICROSERVICE*** in the **IntegrationRuntime** will be done 2 times *(to have 2* ***MICROSERVICES*** *deployed)*.

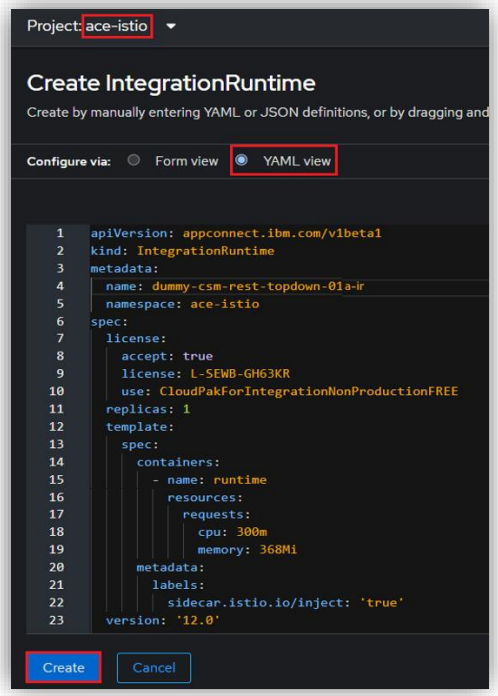
These are:

- ✓ *dummy-csm-rest-topdown-**01a**-go*
- ✓ *dummy-csm-rest-topdown-**02b**-go*

Do not forget the ***LABEL*** and/or ***ANNOTATION*** considerations in the Script with the: ***sidecar.istio.io/inject: 'true'***, which is that ***WILL INJECT*** the functionality of ***ISTIO***.

IMPORTANT: *"The **barURL** must be associated with the **ENDPOINT** of the **COMPILED**.bar, existing in the **DASHBOAR** of type: **INTEGRATION-RUNTIME**".*

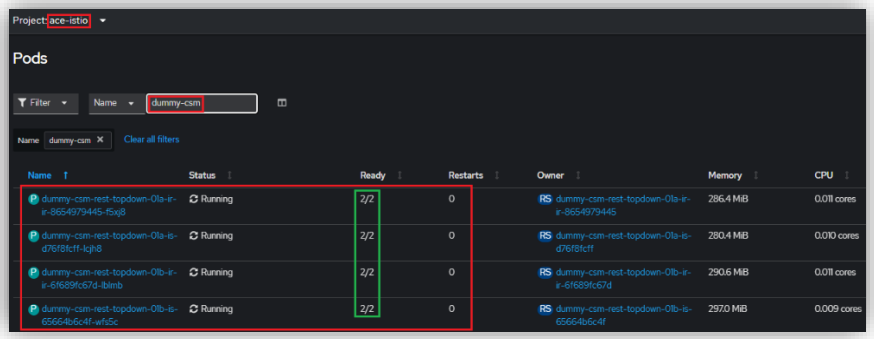


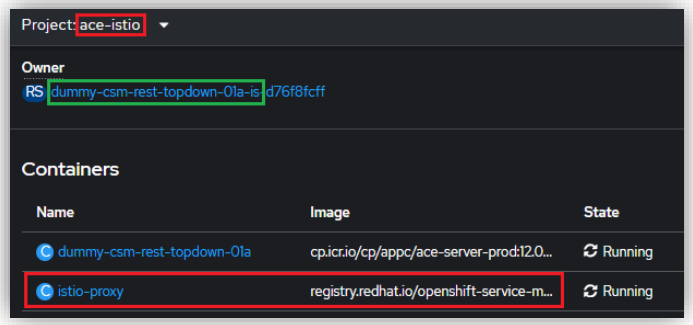


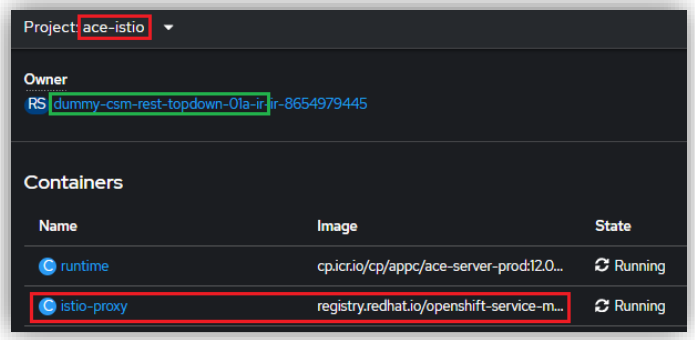
Finally, after the creation of the ***INSTANCES*** of: **INTEGRATION-SERVER** & **INTEGRATION-RUNTIME**, for ***MICROSERVICES***, it is ***VALIDATED*** that the ***PODs*** (dummy-csm) have the ***SECOND CONTAINER***, associated to ***PROXY*** of ***ISTIO***, entering ***PODs/DETAIL***:

The ***IMAGE*** shows that regardless of the deployment ***MODE***: **INTEGRATION-SERVER** or **INTEGRATION-RUNTIME**, the ***CONTAINER*** with the ***ISTIO PROXY*** is automatically generated.

With this ***ISTIO*** It is already applied to our ***MICROSERVICES*** in ***IBM APP CONNECT 12***.







III. MANAGEMENT: DASHBOARDS

In relation to the **DASHBOARDS** is important to know the *list* of ***IPs*** & ***PORTS*** that it manages internally ***ISTIO***. To do this, you must enter the command:

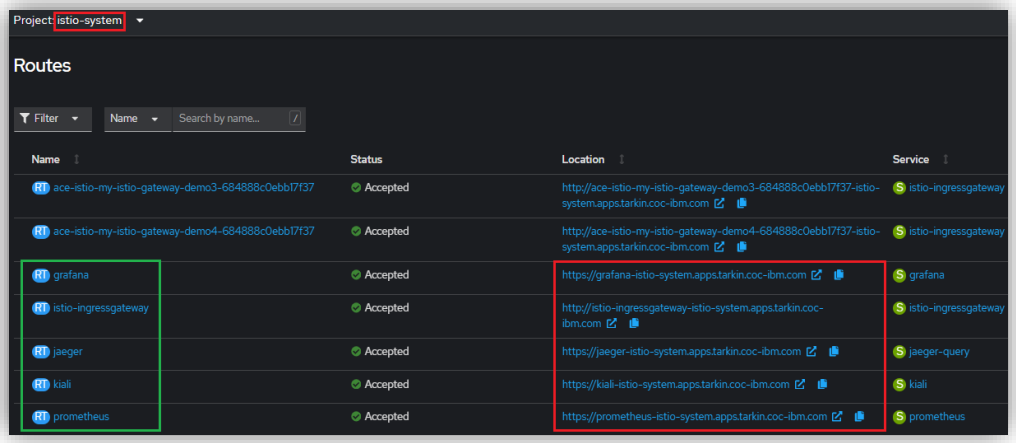
$ kubectl get services -n istio-system

Likewise, to identify the ***URLs*** of the ***DASHBOARDS*** of the ***TOOLS*** associated with the work of ***ISTIO***, you enter:

$ kubectl get routes -n istio-system

```
AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ kubectl get services -n istio-system
NAME                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
grafana                  ClusterIP   172.30.232.103   <none>        3000/TCP
istio-egressgateway      ClusterIP   172.30.213.195   <none>        80/TCP,443/TCP
istio-ingressgateway     ClusterIP   172.30.7.222     <none>        15021/TCP,80/TCP,443/TCP
istiod-basic             ClusterIP   172.30.166.189   <none>        15010/TCP,15012/TCP,443/TCP,15014/TCP,8188/TCP
jaeger-agent             ClusterIP   None             <none>        5775/UDP,5778/TCP,6831/UDP,6832/UDP,14271/TCP
jaeger-collector         ClusterIP   172.30.34.135    <none>        9411/TCP,14250/TCP,14267/TCP,14268/TCP,14269/TCP,4317/TCP,4318/TCP
jaeger-collector-headless ClusterIP  None             <none>        9411/TCP,14250/TCP,14267/TCP,14268/TCP,14269/TCP,4317/TCP,4318/TCP
jaeger-query             ClusterIP   172.30.179.172   <none>        443/TCP,16685/TCP,16687/TCP
kiali                    ClusterIP   172.30.47.198    <none>        20001/TCP,9090/TCP
prometheus               ClusterIP   172.30.170.84    <none>        9090/TCP
zipkin                   ClusterIP   172.30.185.198   <none>        9411/TCP
```

```
$ kubectl get routes -n istio-system
NAME                                                HOST/PORT
ace-istio-my-istio-gateway-demo3-684888c0ebb17f37   ace-istio-my-istio-gateway-demo3-684888c0ebb17f37-istio-system.apps.tarkin.coc-ibm.com
ace-istio-my-istio-gateway-demo4-684888c0ebb17f37   ace-istio-my-istio-gateway-demo4-684888c0ebb17f37-istio-system.apps.tarkin.coc-ibm.com
grafana                                             grafana-istio-system.apps.tarkin.coc-ibm.com
istio-ingressgateway                                istio-ingressgateway-istio-system.apps.tarkin.coc-ibm.com
jaeger                                              jaeger-istio-system.apps.tarkin.coc-ibm.com
kiali                                               kiali-istio-system.apps.tarkin.coc-ibm.com
prometheus                                          prometheus-istio-system.apps.tarkin.coc-ibm.com
```

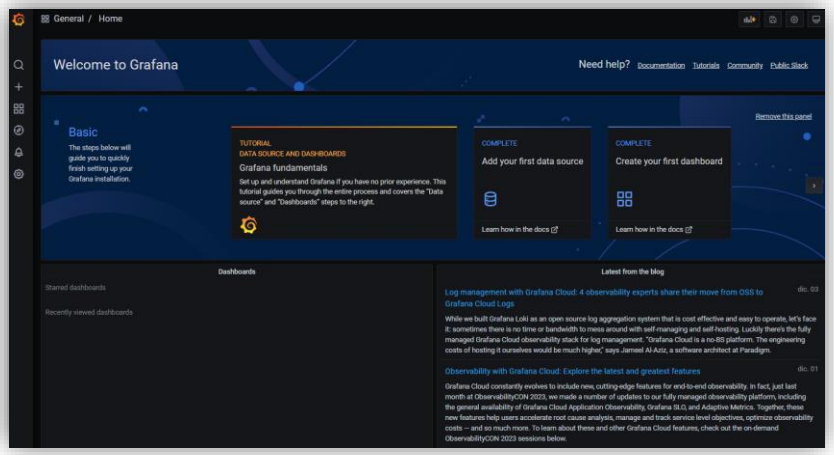


Then, the **DASHBOARDS** generated from ***TOOLS*** for access are:

A. GRAFANA:

*"It is a monitoring **TOOL** that provides visualizations & **GRAPHIC** panels for **METRICS** & **MONITORING** of clusters".*

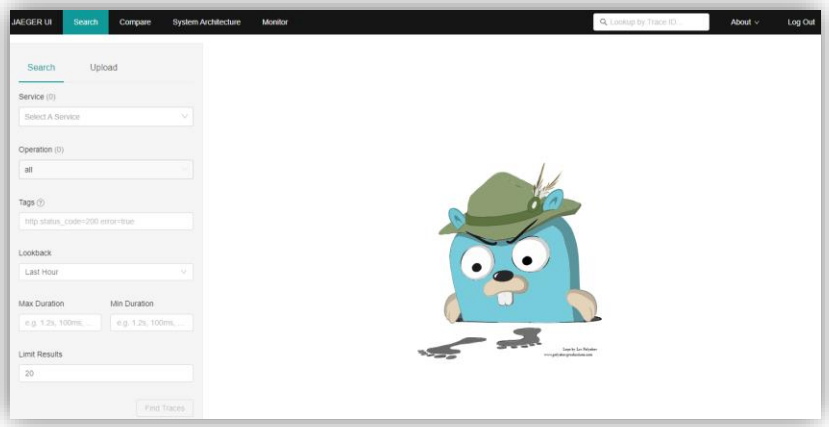
✓ ***URL:*** *https://grafana-istio-system.apps.tarkin.coc-ibm.com*



B. JAEGER:

*"It is a monitoring **TOOL**, whose **OBJECTIVE** is monitoring the **FLOW OF INFORMATION**, through **Distributed Applications & Microservices**, allowing **VISIBILITY** of how data is displayed through different **COMPONENTS** of a **DISTRIBUTED ARCHITECTURE**".*

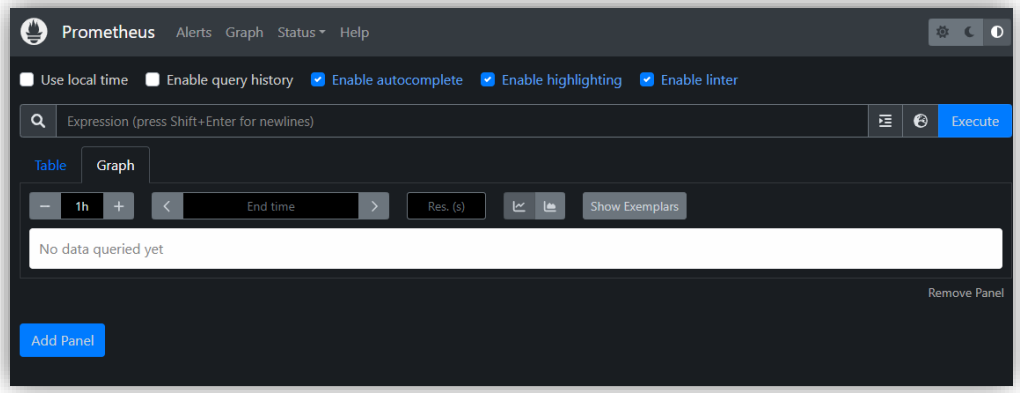
✓ ***URL:*** *https://jaeger-istio-system.apps.tarkin.coc-ibm.com*



C. **PROMETHEUS:**

*"It is a monitoring & **ALERTING TOOL**, which aims to collect **METRICs** of: **Distributed Applications & Microservices,** especially for those based on **CONTAINERS**".*

✓ ***URL:*** *https://prometheus-istio-system.apps.tarkin.coc-ibm.com*

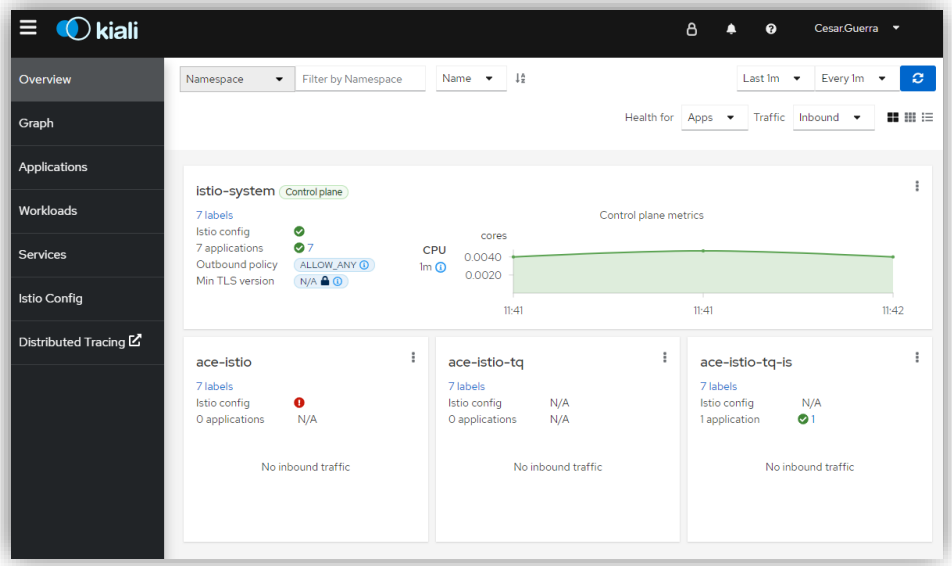


D. **KIALI:**

*"It is a monitoring & **OBSERVABILITY TOOL** designed specifically for **SERVICE MESH** like **ISTIO**, provides an abstraction layer for interactions between **Distributed Applications & Microservices**.*

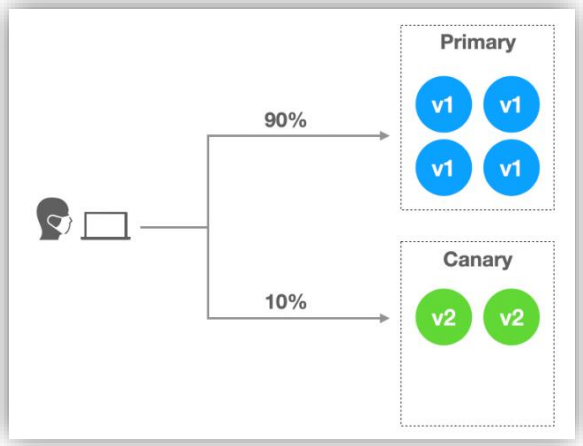
- Provides a ***GRAPHIC*** representation of ***ARCHITECTURE*** of ***MICROSERVICES***, showing the relationships between the different: ***SERVICES, VERSIONS, ROUTES*** & ***TRAFFIC***.
- Provides a ***TOPOLOGY*** map to understand how the ***SERVICES COMMUNICATE*** with each other with their ***DEPENDENCIES***.
- Provides a ***TRACING*** to follow the flow of a ***REQUEST***, through the different ***SERVICES***, avoiding bottlenecks.

✓ ***URL:*** *https://kiali-istio-system.apps.tarkin.coc-ibm.com*



IV. DEPLOYMENT STRATEGIES & VERSION CONTROL:

A. DEMO #1: (CANARY RELEASE):

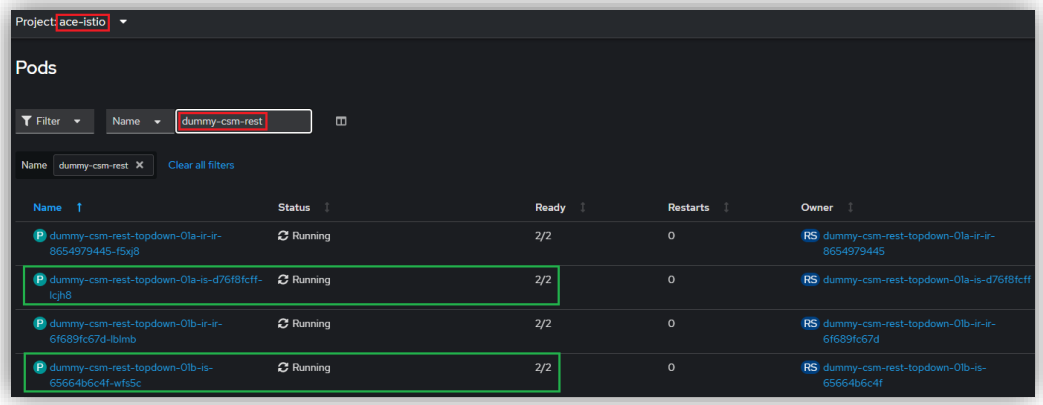


- This ***DEPLOYMENT STRATEGY*** handles a ***NEW VERSION*** of an application *(Microservice Service)*, focused on ***TRAFFIC*** processing *(before it can be **AVAILABLE** in **PRODUCTION**)*.
- Here in the new ***VERSION*** of the application in ***PRODUCTION*** it is called: **CANARY**, but the idea is that only a small part of the ***TRAFFIC*** (5% or 10%) be directed towards said version.
- If this version **CANARY** works correctly & without problems or errors, you can gradually ***INCREASE TRAFFIC*** towards said version.
- Finally, once the new version **CANARY** has been ***VALIDATED*** & is ***STABLE***, all ***TRAFFIC*** can be redirected to that version.

The ***PURPOSE*** of this **DEMO#1** is to be able to ***CREATE* 2 VERSIONS** of a ***MICROSERVICE*** & based on: **V1** or **V2** be able to ***REDIRECT*** the ***TRAFFIC*** respective by means of ***RULES***.

We proceed to ***VALIDATE*** that the ***MICROSERVICES*** are deployed. In this case, those displayed in the ***MODALITY*** of: **INTEGRATION-SERVER**, filtering by: dummy-csm-rest

- ✓ *dummy-csm-rest-topdown-**01a***
- ✓ *dummy-csm-rest-topdown-**01b***



Then, it is necessary to create the ***RESOURCES***:

- ✓ ***SERVICE:****my-istio-service-demo1*
- ✓ ***VIRTUAL-SERVICE:****my-istio-service-demo1*
- ✓ ***DESTINATION-RULE:****my-istio-service-demo1*
- ✓ ***ROUTE:****my-istio-service-demo1*

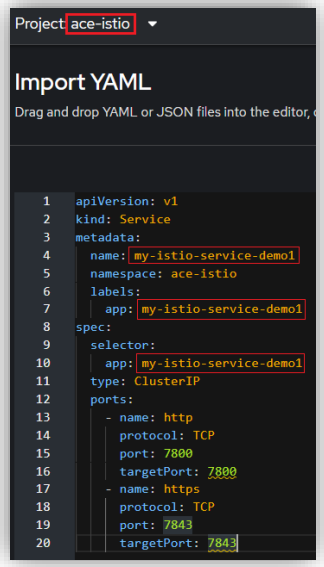
Likewise, the ***LABELs*** & ***SELECTORs*** will handle the same identifier: *my-istio-service-demo1*

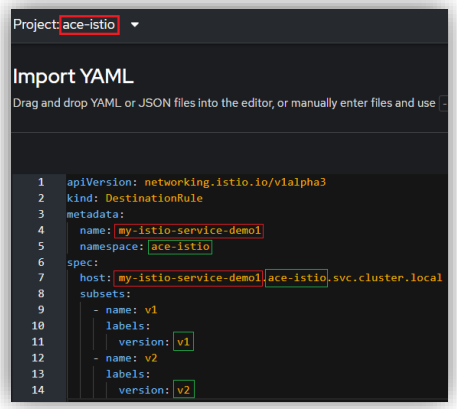
Furthermore, the value of **HOST** It is based on ***PATTERN***: *<ServiceName>.<Namespace>.svc.cluster.local*.

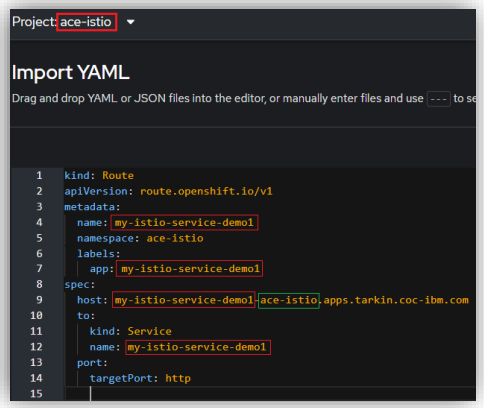
Istio_Demo01.txt

EXPLANATION: *"It must be considered that the **RESOURCE**: **VirtualService** is handling an attribute:(SubSet) which will specify **THE VERSION** of the **MICROSERVICE** where the **TRAFFIC WILL BE REDIRECTED** & the attribute: (Weight) which will specify the **% PRIORITY** that will be taken for **REDIRECTION**".*

The **GOAL** is to pass ***PROGRESSIVELY*** he **100%** of ***TRAFFIC*** of one: **v1** to **v2**, as shown in the picture manipulating ***(%)*** & leaving at the ***END*** only the ***VERSION*** stable.



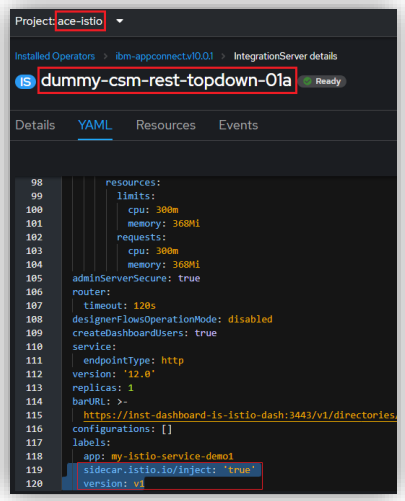


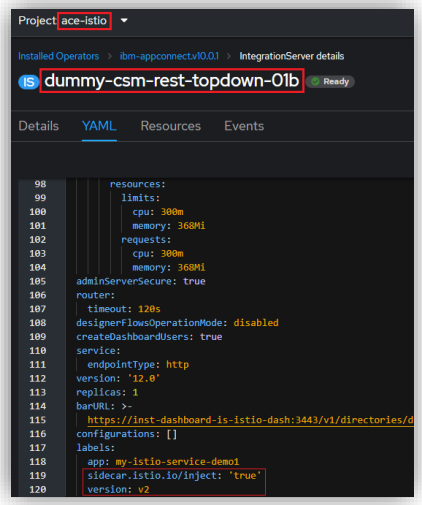


Then, we must consider that the ***LABELS*** at the level of the **INTEGRATION-SERVER** I**INTEGRATION-RUNTIME**, handle for this scenario the ***LABELS***.

MICROSERVICE #1:
labels:
app: my-istio-service-demo1
*version: **v1***

MICROSERVICE #2:
labels:
app: my-istio-service-demo1
*version: **v2***



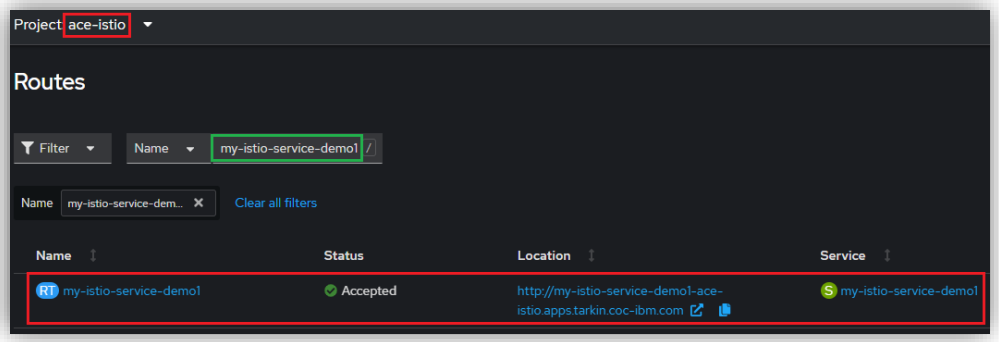


Then, we proceed to ***VALIDATE*** that all the ***RESOURCES*** have been ***CREATED*** in the ***CLUSTER***:

*$ oc get Service,VirtualService,DestinationRule,Route -n ace-istio | grep my-istio-service**demo1***

Then, to ***TEST*** the **DEMO#1** it will be necessary to obtain the ***URL*** of the ***ROUTE*** created: *my-istio-service-demo1*

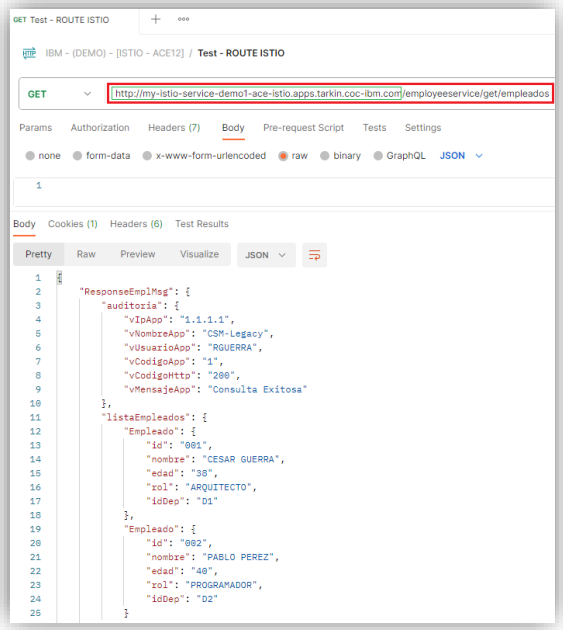
In this case it would be: *http://my-istio-service-demo1-ace-istio.apps.tarkin.coc-ibm.com*



Then, the idea is to ***TEST*** as many times as possible. the ***MICROSERVICE***, in this case by: **POSTMAN**.

Likewise, you can ***TEST*** from the command line, to send ***MASSIVELY*** *(Sequentially or Parallel)* ***REQUESTs*** through the ***TOOL***: **SIEGE**:

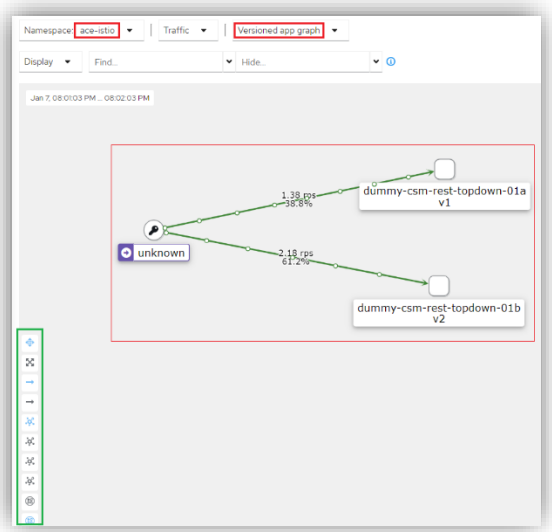
$ siege --concurrent=20 --reps=2 -vhttp://my-istio-service-demo1-ace-istio.apps.tarkin.coc-ibm.com/employeeservice/get/employees

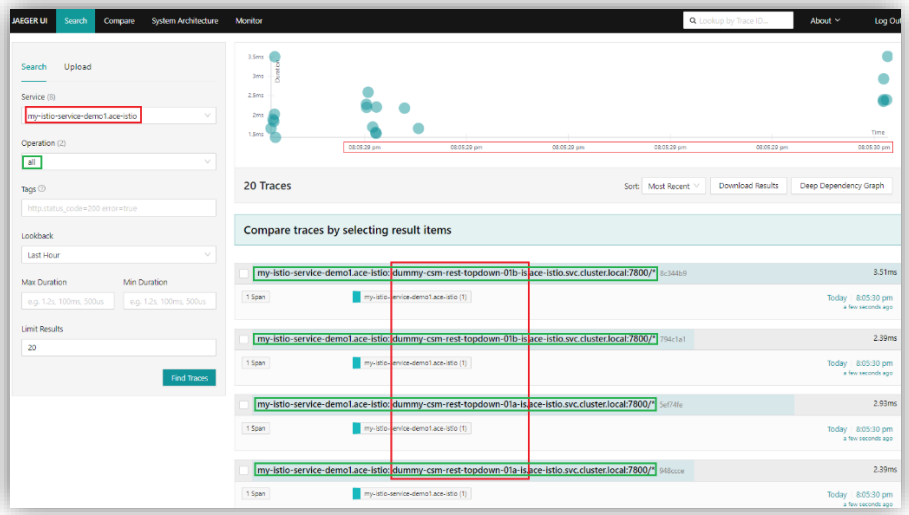



```
$ siege --concurrent=20 --reps=2 -v http://my-istio-service-demo1-ace-istio.apps.tarkin.coc-ibm.com/employeeservice/get/empleados
** SIEGE 3.0.5
** Preparing 20 concurrent users for battle.
The server is now under siege...
done.

Transactions:                  40 hits
Availability:              100.00 %
Elapsed time:                2.43 secs
Data transferred:            0.01 MB
Response time:               0.21 secs
Transaction rate:           16.46 trans/sec
Throughput:                  0.01 MB/sec
Concurrency:                 3.41
Successful transactions:       40
Failed transactions:            0
Longest transaction:         0.31
Shortest transaction:        0.18

HTTP/1.1 200   0.20 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.21 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.21 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.21 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.21 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.22 secs:     377 bytes ==> GET  /employeeservice/get/empleados
```

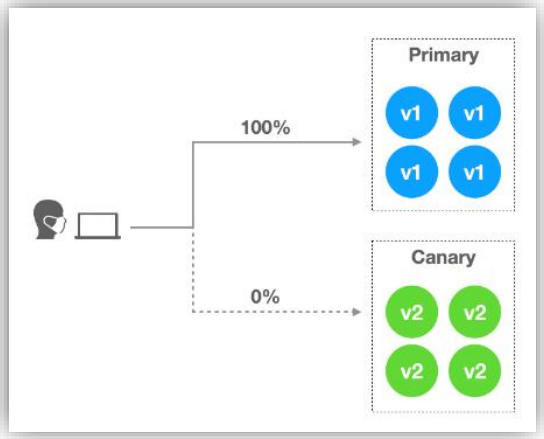
Finally, you enter the ***DASHBOARD*** of **KIALI** & you can see **GRAPHICALLY** how the ***REQUESTs*** are redirected ***(in green)***, by the defined ***TRAFFIC RULE***, at the level of the ***MICROSERVICES*** associated with: ***INTEGRATION-SERVERS***.

Also, this ***REQUEST*** trace connects to the ***DASHBOARD*** of **JAEGER**, as shown in ***IMAGE***.





B. DEMO #2: (BLUE & GREEN DEPLOYMENT):

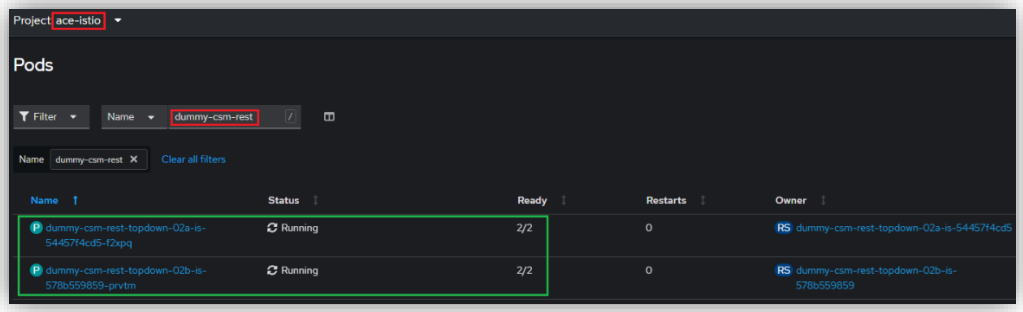


- This ***STRATEGY*** manages ***2 PRODUCTION*** environments:**1 ACTIVE (BLUE)** which will handle live ***TRAFFIC*** & **1 INACTIVE (GREEN)**, where the ***NEW VERSION*** of the ***APPLICATION*** is deployed.
- Initially, all ***TRAFFIC*** is sent to the environment **(BLUE)**, which is the ***STABLE*** version & when the ***NEW VERSION*** of the **APP** is developed, it will be deployed in the environment **(GREEN)** for ***TESTING***, while the environment **(BLUE)** will continue to process ***TRAFFIC***.
- Finally, when the APP in the environment **(GREEN)** has been **100%** tested, the ***TRAFFIC*** is changed directly (completely) from the environment: **(BLUE)** to **(GREEN)**.

The **PURPOSE** of this**DEMO#2** is to change the ***TRAFFIC*** handled between the environments ***MAJOR*** **(BLUE)** & ***CANARY*** **(GREEN)** progressively, towards a ***DIRECT*** way *(100% of **TRAFFIC**)*.

We proceed to ***VALIDATE*** that the ***MICROSERVICES*** are deployed. In this case, those displayed in the ***MODALITY*** of: **INTEGRATION-SERVER**, filtering by: dummy-csm-rest:

- ✓ *dummy-csm-rest-topdown-**02a***
- ✓ *dummy-csm-rest-topdown-**02b***



Then, it is necessary to create the ***RESOURCES***:

- ✓ ***SERVICE:*** *my-istio-service-demo2*
- ✓ ***VIRTUAL-SERVICE:*** *my-istio-service-demo2*
- ✓ ***DESTINATION-RULE:*** *my-istio-service-demo2*
- ✓ ***ROUTE:*** *my-istio-service-demo2*

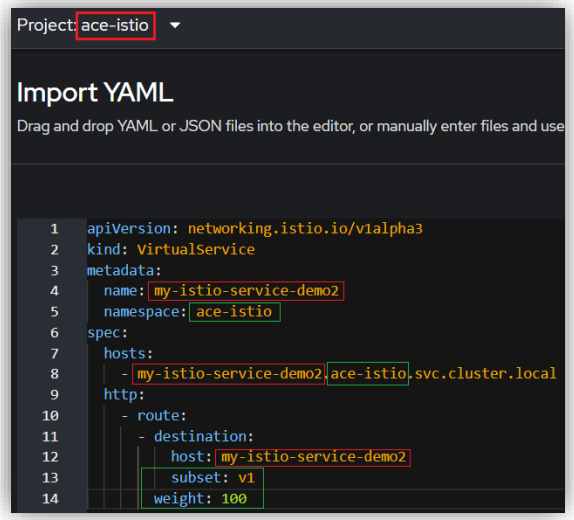
Likewise, the ***LABELs*** & ***SELECTORs*** will handle the same identifier: *my-istio-service-demo2*

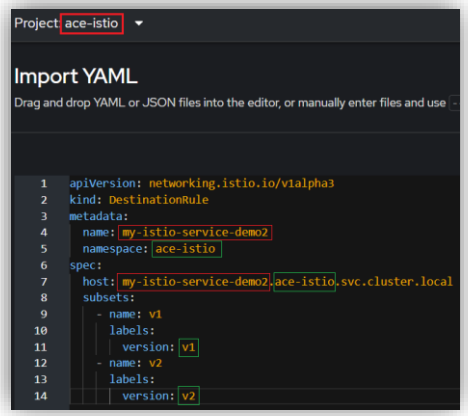
Furthermore, the value of **HOST** It is based on ***PATTERN***: *<ServiceName>.<Namespace>.svc.cluster.local*.

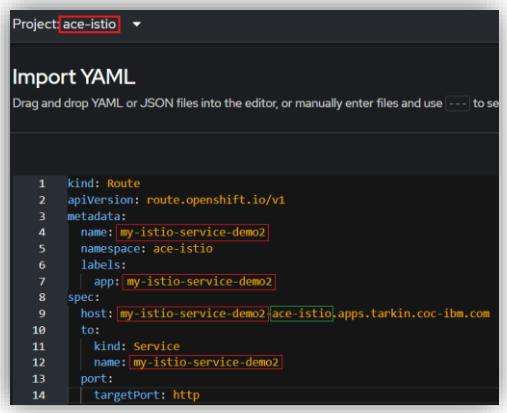
Istio_Demo02.txt

EXPLANATION: *"It must be considered that the **RESOURCE**: **VirtualService** is handling an attribute:(SubSet) which will specify **THE VERSION** of the **MICROSERVICE** where the **TRAFFIC WILL BE REDIRECTED** & the attribute: (Weight) which will specify the **% PRIORITY** that will be taken for **REDIRECTION**."*

The ***GOAL*** in **BLUE & GREEN** is that the **100%** of traffic be applied to a ***VERSION***, as shown in the ***IMAGE***.







Then, we must consider that the ***LABELS*** at the level of the **INTEGRATION-SERVER** I**INTEGRATION-RUNTIME**, handle for this scenario the ***LABEL***.

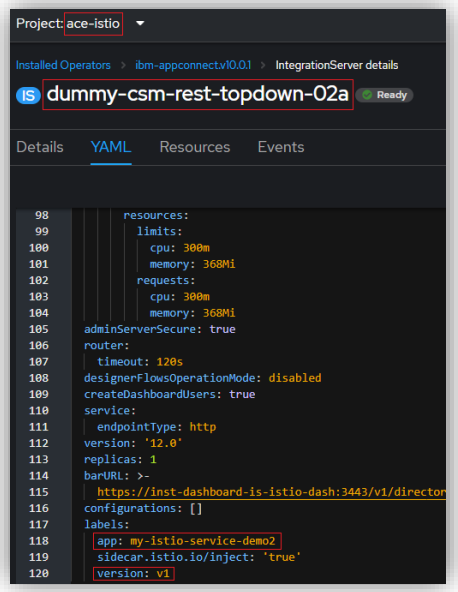
MICROSERVICE #1:
labels:
app: my-istio-service-demo2
*version: **v1***

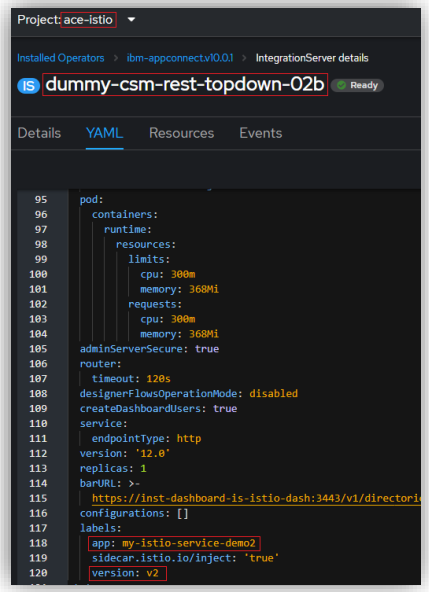
MICROSERVICE #2:
labels:
app: my-istio-service-demo2
*version: **v2***

IMPORTANT: *"The equivalence in colors is managed by the **VERSION**".*

- ✓ **V1** => **BLUE**.
- ✓ **V2** => **GREEN**.

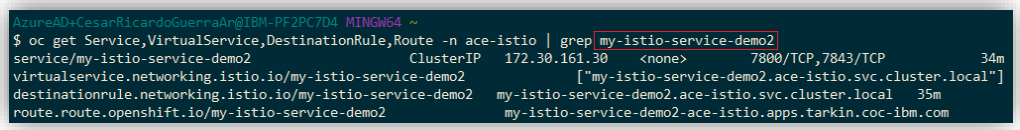
*"And the one you want to **ACTIVATE**, will be the one that must be **ACTIVATED** at the level of the **VIRTUAL-SERVICE**".*





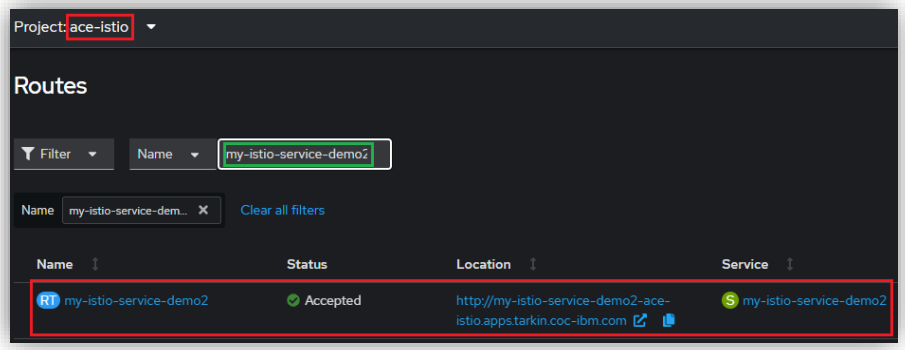
Then, we proceed to ***VALIDATE*** that all the ***RESOURCES*** have been ***CREATED*** in the ***CLUSTER***:

*$ oc get Service,VirtualService,DestinationRule,Route -n ace-istio | grep **my-istio-service-demo2***



Then, to ***TEST*** the **DEMO#2** it will be necessary to obtain the ***URL*** of the ***ROUTE*** created: *my-istio-service-demo2*

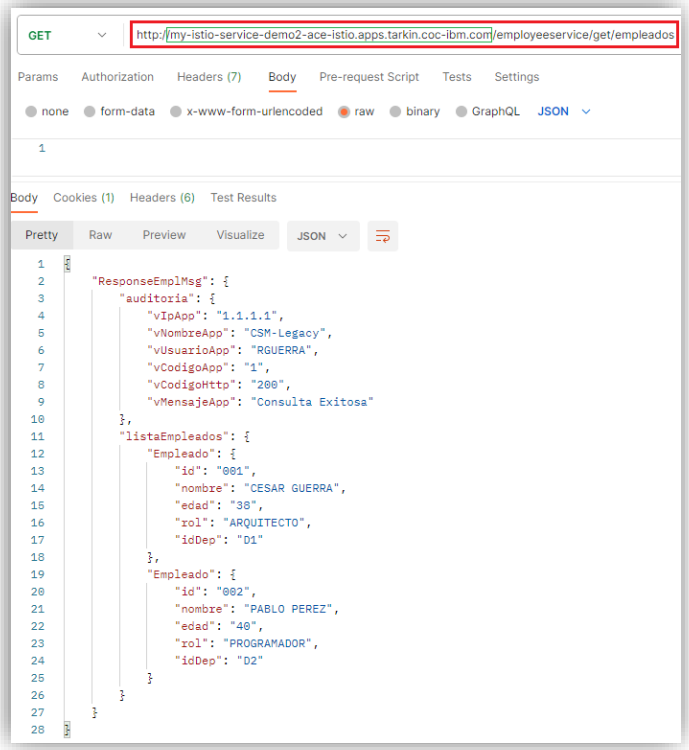
In this case it would be: *http://my-istio-service-demo2-ace-istio.apps.tarkin.coc-ibm.com*



Then, the idea is ***TEST*** as many times as possible the ***MICROSERVICE***, in this case by: **POSTMAN**.

Likewise, you can ***TEST*** from the command line to send ***MASSIVELY*** *(Sequentially or Parallel)* the ***REQUEST*** for the ***TOOL***: **SIEGE**:

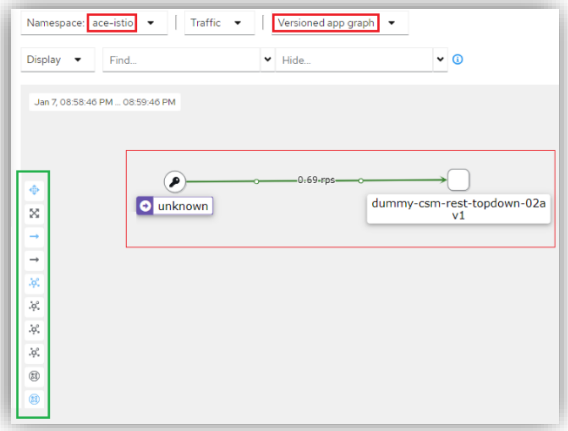
$ siege --concurrent=20 --reps=2 -vhttp://my-istio-service-demo2-ace-istio.apps.tarkin.coc-ibm.com/employeeservice/get/employees

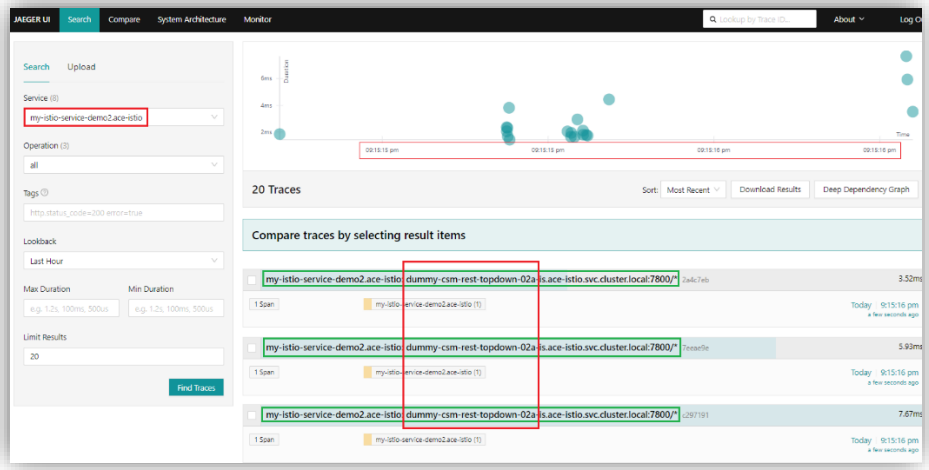


```
AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ siege --concurrent=20 --reps=2 -v http://my-istio-service-demo2-ace-istio.apps.tarkin.coc-ibm.com/employeeservice/get/empleados
** SIEGE 3.0.5
** Preparing 20 concurrent users for battle.
The server is now under siege...
done.

Transactions:                  40 hits
Availability:              100.00 %
Elapsed time:                2.46 secs
Data transferred:            0.01 MB
Response time:               0.25 secs
Transaction rate:           16.23 trans/sec
Throughput:                  0.01 MB/sec
Concurrency:                 4.01
Successful transactions:       40
Failed transactions:            0
Longest transaction:         0.45
Shortest transaction:        0.18

HTTP/1.1 200   0.44 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.44 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.44 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.44 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.44 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.44 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.45 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.20 secs:     377 bytes ==> GET  /employeeservice/get/empleados
```

Finally, you enter the ***DASHBOARD*** of **KIALI** & you can see ***GRAPHICALLY*** how the ***REQUESTs*** are redirected ***(in green)***, by the defined ***TRAFFIC RULE***, at the level of the ***MICROSERVICES*** associated with: ***INTEGRATION-SERVERS***.

Also, this ***REQUEST*** trace connects to the ***DASHBOARD*** of **JAEGER**, as shown in the picture.

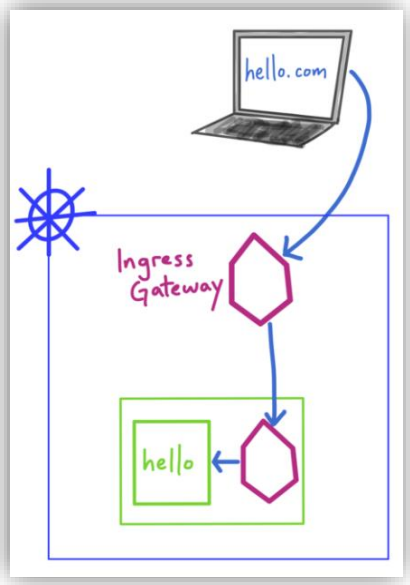




C. DEMO #3: *(TRAFFIC CONTROL: "INGRESS GATEWAY"):*

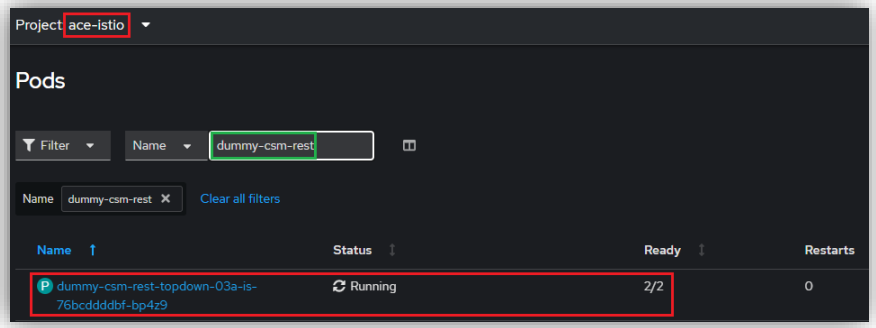
ISTIO provides a shape to manage the ***TRAFFIC*** that ***INCOME*** to the ***MESH*** from outside the ***CLUSTER***, through of a resource of type: **GATEWAY** ***(ISTIO's own)***.

IMPORTANT: *"Do not confuse the resource: **INGRESS-GATEWAY** with the resource: **GATEWAY**, the latter is used to configure the: **INGRESS-GATEWAY**".*



We proceed to ***VALIDATE*** that the ***MICROSERVICES*** are deployed. In this case, those displayed in the ***MODALITY*** of: **INTEGRATION-SERVER**, filtering by: dummy-csm-rest:

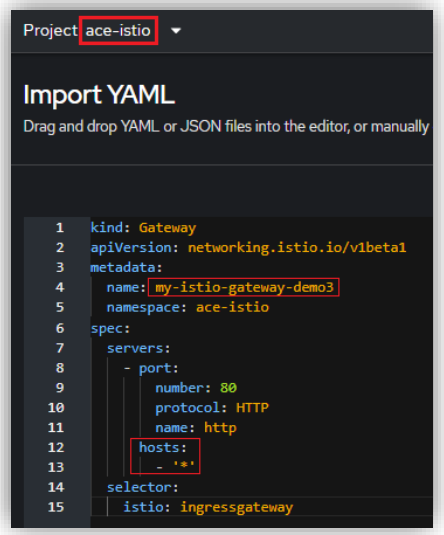
- ✓ *dummy-csm-rest-topdown-**03a***

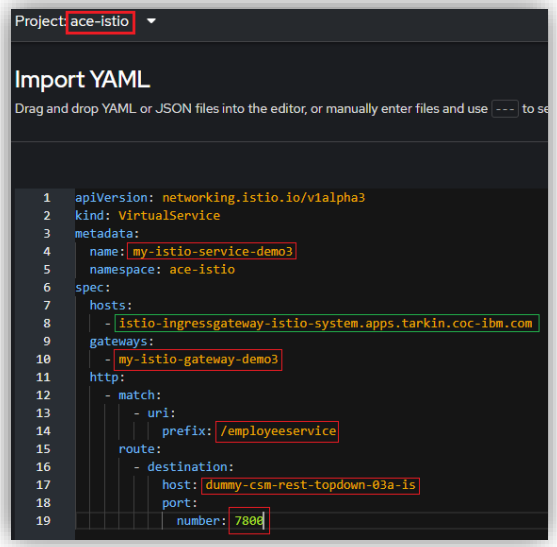


Then, it is necessary to create the ***RESOURCES***:

- ✓ ***GATEWAY:*** *my-istio-gateway-demo3*
- ✓ ***VIRTUAL-SERVICE:*** *my-istio-service-demo3*

EXPLANATION: *"It must be considered that the **RESOURCE**: **VirtualService** will refer to the **RESOURCE**: **Gateway**, in which the **URI** handling is that of the **BackEnd** that will be exposed & where the reference to the **SERVICE** of the **INTEGRATION-SERVER** instance must be defined & **ACE12 PORT** (7800)".*

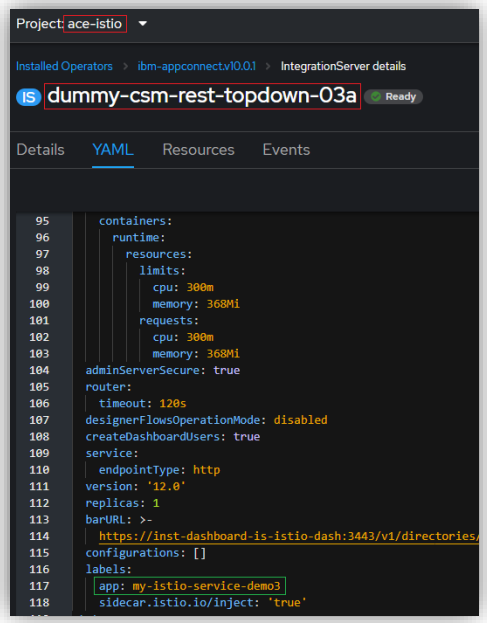




Then, we must consider that the ***LABELS*** at the level of the **INTEGRATION-SERVER** and/or **INTEGRATION-RUNTIME**, handle for this scenario the ***LABELS***.

MICROSERVICE #1:
labels:
app: my-istio-service-demo3

IMPORTANT: *"For this **DEMO** the **VERSION** is not required."*



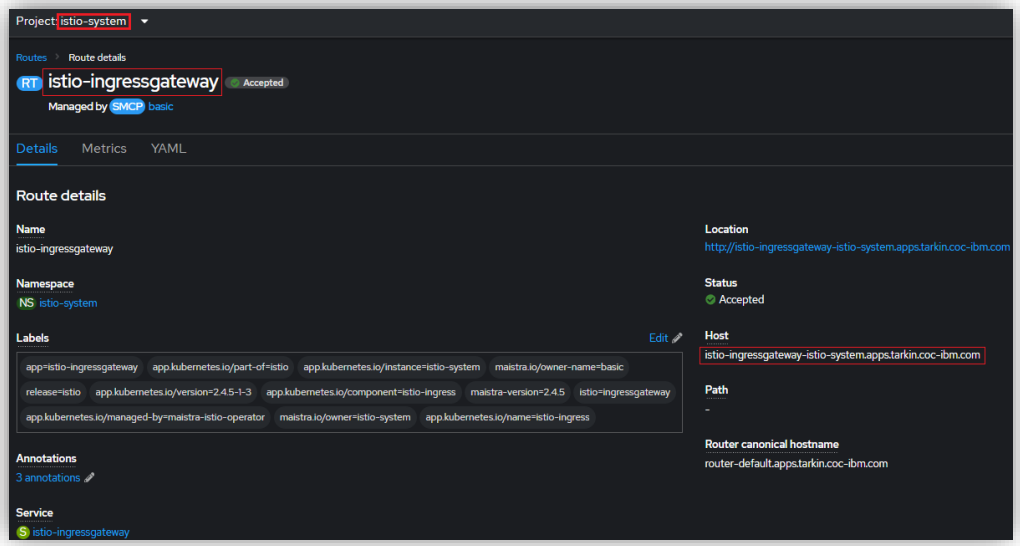
Then, we proceed to ***VALIDATE*** that all the ***RESOURCES*** have been ***CREATED*** in the ***CLUSTER***:

$ oc get Gateway,VirtualService -n ace-istio | grep ***demo3***

```
AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ oc get Gateway,VirtualService  -n ace-istio | grep demo3
gateway.networking.istio.io/my-istio-gateway-demo3   24m
virtualservice.networking.istio.io/my-istio-service-demo3   ["my-istio-gateway-demo3"]   ["*"]
```

Then, to ***TEST*** the **DEMO #3** it will be necessary to obtain the ***URL*** of the automatically generated ***ROUTE***: *istio-ingressgateway*, in the ***NAMESPACE***: istio-system

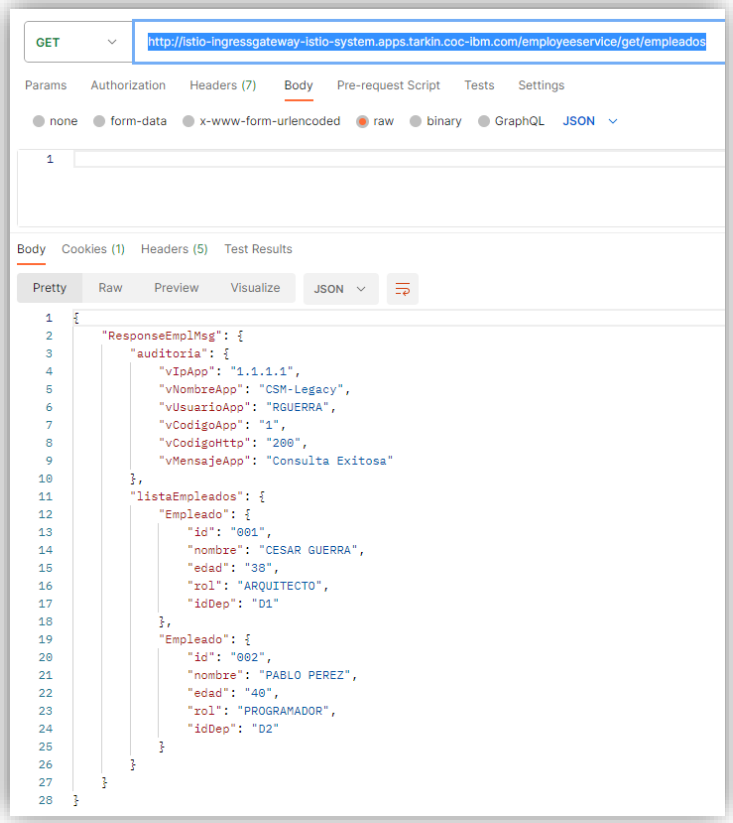
In this case it would be: *http://istio-ingressgateway-istio-system.apps.tarkin.coc-ibm.com*



Then, the idea is to ***TEST*** as many times as possible the ***MICROSERVICE***, in this case by: **POSTMAN**.

Likewise, you can ***TEST*** from the command line, to send ***MASSIVELY*** *(Sequentially or Parallel)* ***REQUESTs*** through the ***TOOL***: **SIEGE**:

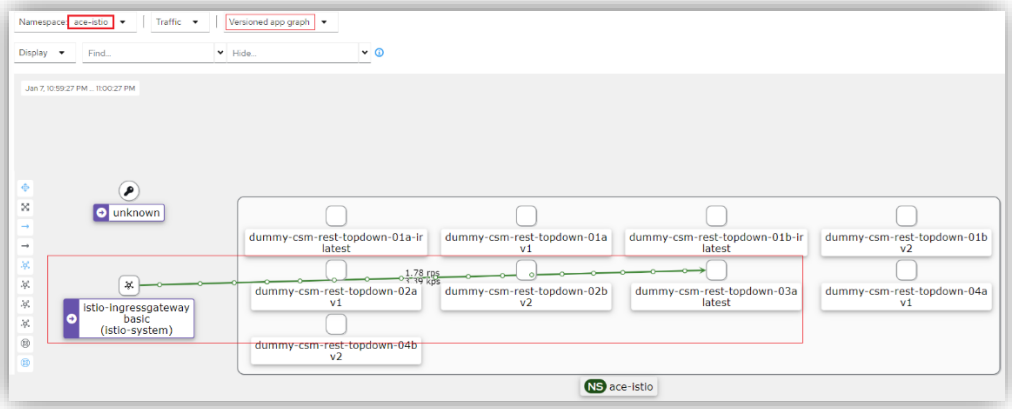
$ siege --concurrent=20 --reps=2 -v http://istio-ingressgateway-istio-system.apps.tarkin.coc-ibm.com/employeeservice/get/employees

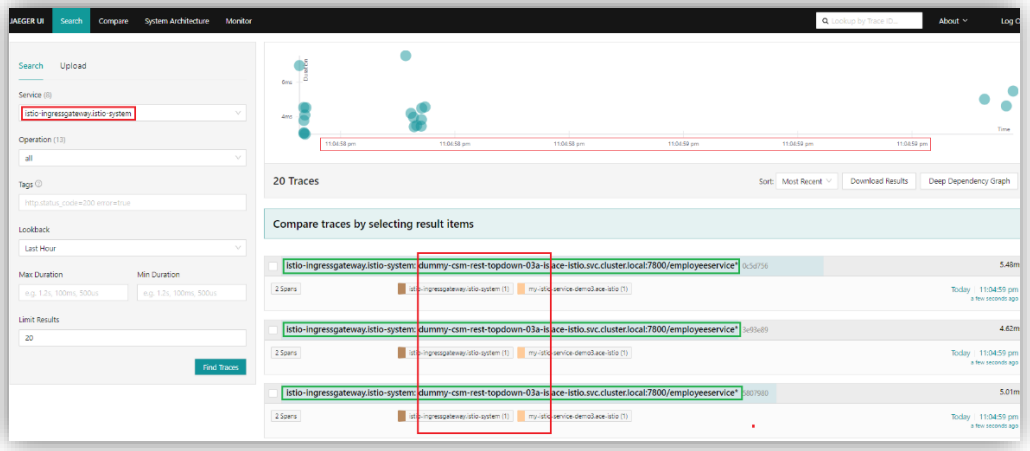


```
AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ siege --concurrent=20 --reps=2 -v http://istio-ingressgateway-istio-system.apps.tarkin.coc-ibm.com/employeeservice/get/empleados
** SIEGE 3.0.5
** Preparing 20 concurrent users for battle.
The server is now under siege...
done.

Transactions:                 40 hits
Availability:             100.00 %
Elapsed time:               2.47 secs
Data transferred:           0.01 MB
Response time:              0.24 secs
Transaction rate:          16.18 trans/sec
Throughput:                 0.01 MB/sec
Concurrency:                3.82
Successful transactions:      40
Failed transactions:           0
Longest transaction:        0.39
Shortest transaction:       0.18

HTTP/1.1 200   0.34 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.35 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.36 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.36 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.36 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.37 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.38 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.39 secs:     377 bytes ==> GET  /employeeservice/get/empleados
HTTP/1.1 200   0.20 secs:     377 bytes ==> GET  /employeeservice/get/empleados
```

Finally, you enter the ***DASHBOARD*** of **KIALI** & you can see ***GRAPHICALLY*** how the ***REQUESTs*** are redirected ***(in green)***, by the defined ***TRAFFIC RULE***, at the level of the ***MICROSERVICES*** associated with: ***INTEGRATION-SERVERS***.

Also, this ***REQUEST*** trace connects to the ***DASHBOARD*** of **JAEGER**, as shown in ***IMAGE***.

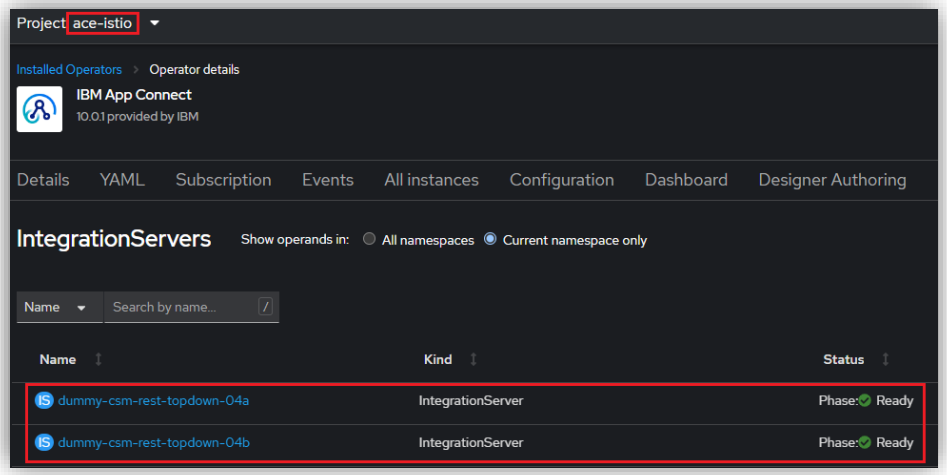




D. DEMO #4: *(UNIFIING EVERYTHING):*

We proceed to ***VALIDATE*** that the ***MICROSERVICES*** are deployed. In this case, those displayed in the ***MODALITY*** of: **INTEGRATION-SERVER**, filtering by: dummy-csm-rest:

- ✓ *dummy-csm-rest-topdown-**04a***
- ✓ *dummy-csm-rest-topdown-**04b***



Then, it is necessary to create the ***RESOURCES***:

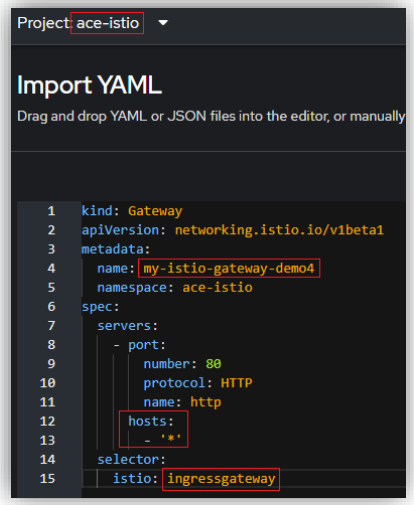
- ✓ ***GATEWAY:*** *my-istio-gateway-demo4*
- ✓ ***VIRTUAL-SERVICE:*** *my-istio-service-demo4*
- ✓ ***DESTINATION-RULE:*** *my-istio-service-demo4*
- ✓ ***SERVICE:*** *my-istio-service-demo4*
- ✓ ***ROUTE:*** *my-istio-service-demo4*

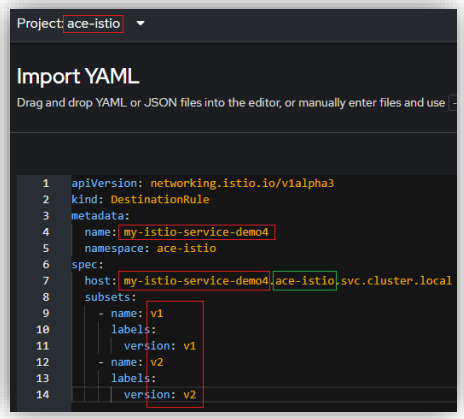
Likewise, the ***LABELs*** & ***SELECTORs*** will handle the same identifier: *my-istio-service-demo4*

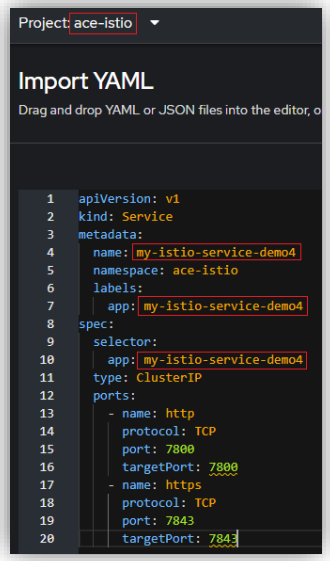
Furthermore, the value of **HOST** is based on ***PATTERN***: *<ServiceName>.<Namespace>.svc.cluster.local*.

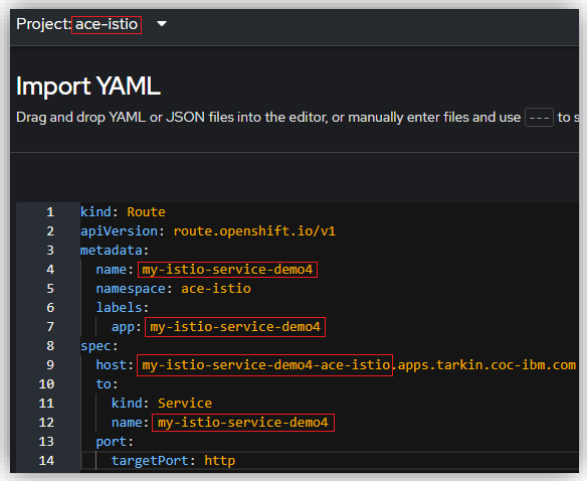
Istio_Demo04.txt

EXPLANATION: *"It must be considered that the **RESOURCE**: **VirtualService** is handling an attribute: (SubSet) which will specify **THE VERSION** of the **MICROSERVICE** where the **TRAFFIC WILL BE REDIRECTED** & the attribute: (Weight) which will specify the **% PRIORITY** that will be taken for **REDIRECTION**".*




Project: ace-istio
Import YAML
Drag and drop YAML or JSON files into the editor, or manually enter files and use
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
name: my-istio-service-demo4
namespace: ace-istio
spec:
host: my-istio-service-demo4.ace-istio.svc.cluster.local
subsets:
- name: v1
labels:
version: v1
- name: v2
labels:
version: v2

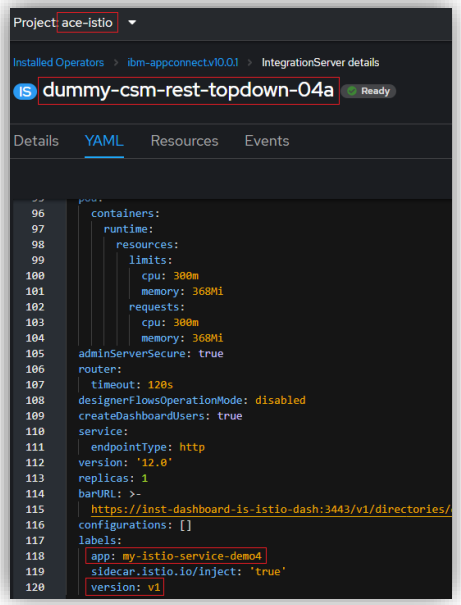

Project: ace-istio
Import YAML
Drag and drop YAML or JSON files into the editor, o
apiVersion: v1
kind: Service
metadata:
name: my-istio-service-demo4
namespace: ace-istio
labels:
app: my-istio-service-demo4
spec:
selector:
app: my-istio-service-demo4
type: ClusterIP
ports:
- name: http
protocol: TCP
port: 7800
targetPort: 7800
- name: https
protocol: TCP
port: 7843
targetPort: 7843

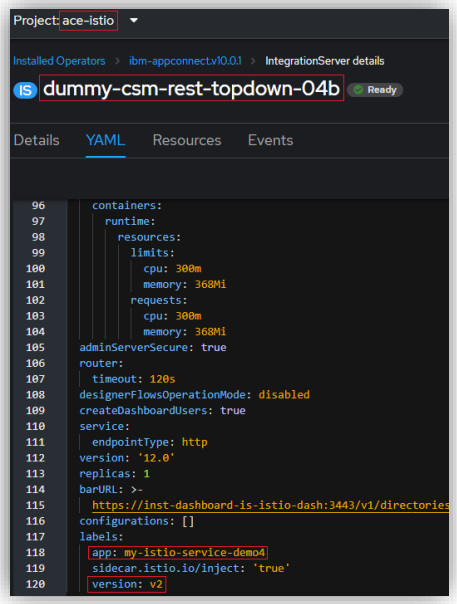

Project: ace-istio
Import YAML
Drag and drop YAML or JSON files into the editor, or manually enter files and use --- to s
kind: Route
apiVersion: route.openshift.io/v1
metadata:
name: my-istio-service-demo4
namespace: ace-istio
labels:
app: my-istio-service-demo4
spec:
host: my-istio-service-demo4-ace-istio.apps.tarkin.coc-ibm.com
to:
kind: Service
name: my-istio-service-demo4
port:
targetPort: http

Then, we must consider that the ***LABELS*** at the level of the **INTEGRATION-SERVER** and/or **INTEGRATION-RUNTIME**, handle for this scenario the ***LABEL***.

MICROSERVICE #1:
labels:
app: my-istio-service-demo4
*version: **v1***

MICROSERVICE #2:
labels:
app: my-istio-service-demo4
*version: **v2***





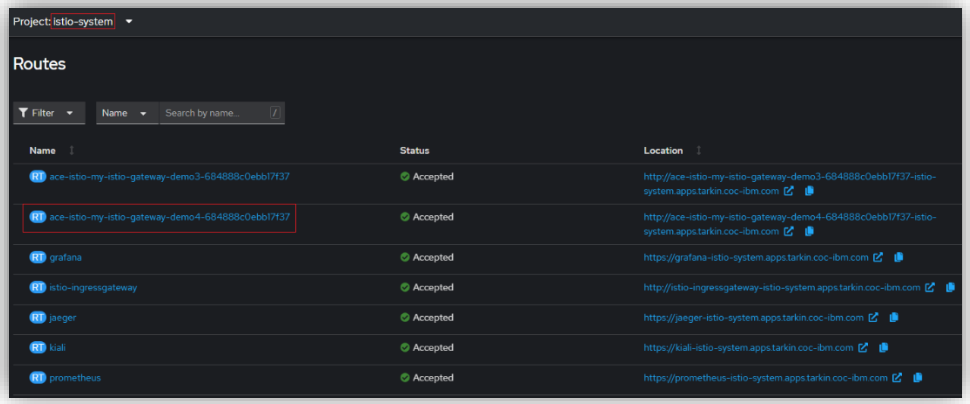
Then, we proceed to ***VALIDATE*** all the ***RESOURCES*** that have been ***CREATED*** in the ***CLUSTER***:

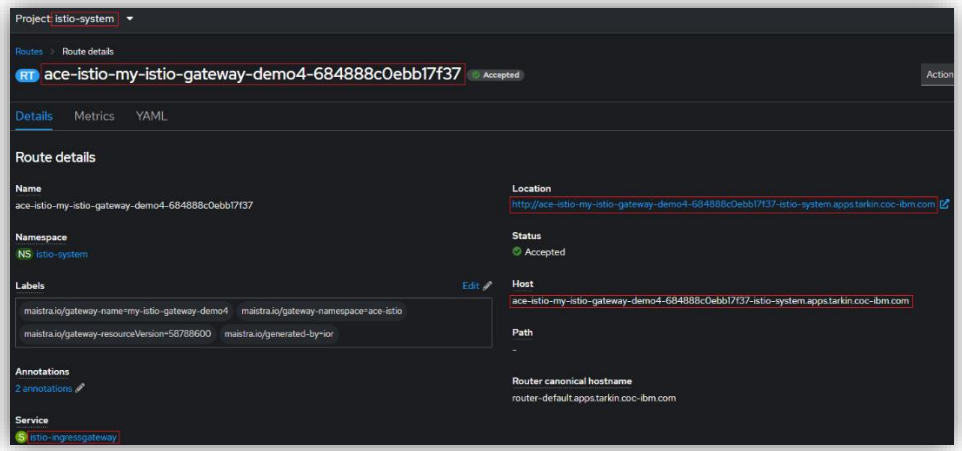
*$ oc get Service,VirtualService,DestinationRule,Route -n ace-istio | grep **my-istio-service-demo4***

```
AzureAD+CesarRicardoGuerraAr@IBM-PF2PC7D4 MINGW64 ~
$ oc get Service,VirtualService,DestinationRule,Route -n ace-istio | grep my-istio-service-demo4
service/my-istio-service-demo4                    ClusterIP   172.30.126.34    <none>        7800/TCP,7843/TCP
virtualservice.networking.istio.io/my-istio-service-demo4   ["my-istio-gateway-demo4"]   ["*"]
             8h
destinationrule.networking.istio.io/my-istio-service-demo4   my-istio-service-demo4.ace-istio.svc.cluster.local   8h
route.route.openshift.io/my-istio-service-demo4               my-istio-service-demo4-ace-istio.apps.tarkin.coc-ibm.com
    my-istio-service-demo4            http                            None
```

Then, to ***TEST*** the **DEMO#4** will be necessary to obtain the ***URL*** of the automatically generated ***ROUTE***: ace-istio-my-istio-gateway-demo4-xxxx, in the ***NAMESPACE***: istio-system

In this case it would be: *http://ace-istio-my-istio-gateway-demo4-684888c0ebb17f37-istio-system.apps.tarkin.coc-ibm.com*



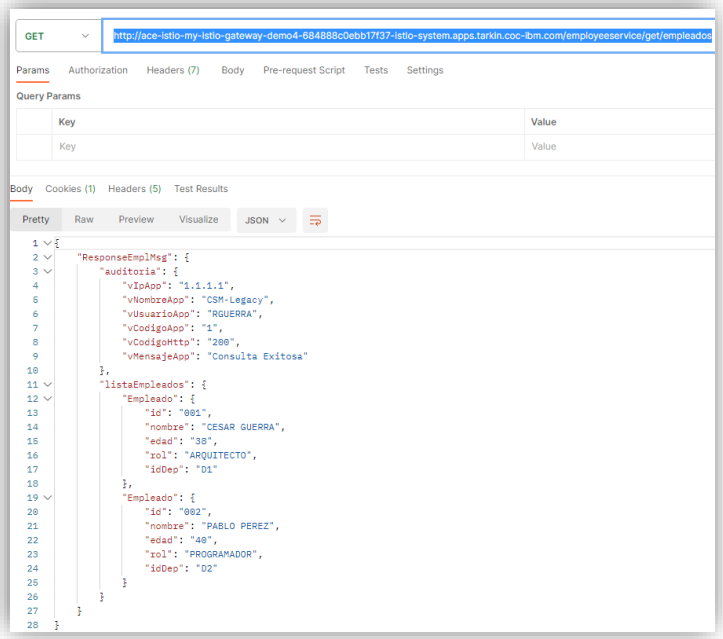


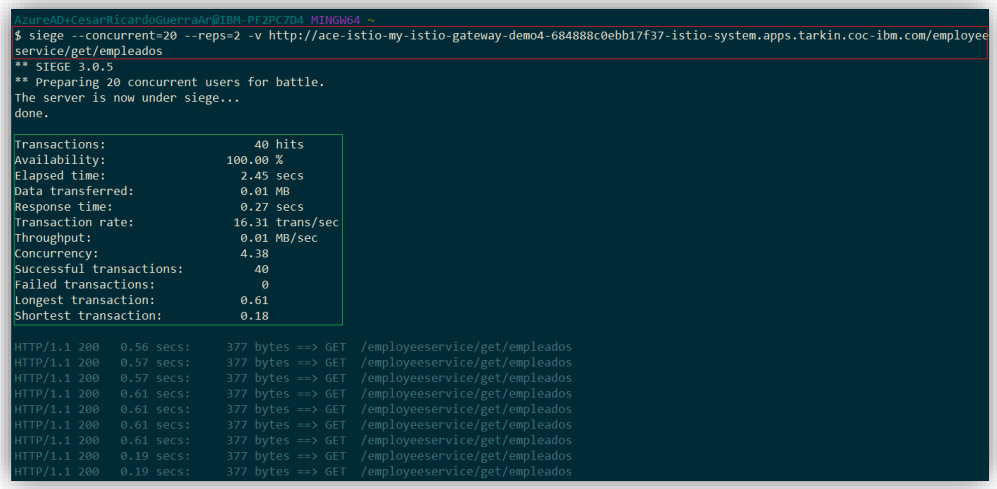
Then, the idea is to ***TEST*** as many times as possible. the ***MICROSERVICE***, in this case by: **POSTMAN**.

Likewise, you can ***TEST*** from the command line, to send ***MASSIVELY*** *(Sequentially or Parallel)* ***REQUESTs*** through the ***TOOL***: **SIEGE**:

$ siege --concurrent=20 --reps=2 -v http://ace-istio-my-istio-gateway-demo4-684888c0ebb17f37-istio-system.apps.tarkin.coc-ibm.com/employeeservice/get/employees

IMPORTANT: *"You could also target **DIRECTLY** without going through the **GATEWAY** but from the **ROUTE** from here: http://my-istio-service-demo4-ace-istio.apps.tarkin.coc-ibm.com/employeeservice/get/employees".*





Finally, you enter the ***DASHBOARD*** of **KIALI** & you can see ***GRAPHICALLY*** how the ***REQUESTs*** are redirected ***(in green)***, by the defined ***TRAFFIC RULE***, at the level of the ***MICROSERVICES*** associated with: ***INTEGRATION-SERVERS***.

Also, this ***REQUEST*** trace connects to the ***DASHBOARD*** of **JAEGER**, as shown in ***IMAGE***.