Microservice Architecture on Istio and Kubernetes

As more and more Microservices generated, there are some problems come out. Especially when a new environment needs to be deployed, some issues such as service discovery, load balancing, trace tracing, traffic management, and security authentication will be explored. At this moment, a new Microservice framework Service Mesh is innovated which is been to solve these series of problems. Istio is the mainstream open source tool of service mesh. So we can use it to show the service mesh.

Before

You need to create 3 VMS and install the CentOs 7 or have the aliyun's cloud hosting, now, the majority cloud hostings support the istio and k8s.

Create VMS (If you use cloud hosting, please skip this)

1. Download the Centos 7 ova
   https://onevmw-my.sharepoint.com/:f:/g/personal/dni_vmware_com/EqvT_Pc5RchJv5LBmlp_PKgBHem_Y2RjiyCFFQsX2zuriw?e=fBHMgB
2. create VM in the VMware Fusion or workstation pro and change the config

   File import centos set vm name change the storage(40) and network(NAT) start. After these steps, you can try to ping 192.168.56.11.

   After you create the VM, you should add your DNS to the **/etc/sysconfig/network-scripts/ifcfg-eth0** Command:

```
vim vim /etc/sysconfig/network-scripts/ifcfg-eth0Add: DNS1=192.1168.31.1   // This should same with your
mac.
```

3. Clone the VM to generate other VMS
   Close node 1 create full clone login change ip and add dns run: service network restart change hostname

   After this, try to ping them.

Config the k8s cluster and istio(https://github.com/unixhot/salt-kubeadm)

1. **install Salt-SSH and download salt-kubeadm**
    a. set ssh key and id

    ```
    ssh-keygen -t rsa -q -N ''
    ssh-copy-id linux-node1
    ssh-copy-id linux-node2
    ssh-copy-id linux-node3
    ```

    b. install Salt SSH
    The Salt SSH manages the vms and k8s roles, you need to do this in every vms.

    ```
    wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
    yum install -y https://repo.saltstack.com/py3/redhat/salt-py3-repo-latest.el7.noarch.rpm
    yum install -y salt-ssh git unzip ntpdate
    ```

    c. download salt kubeadm and config

    ```
    git clone https://github.com/unixhot/salt-kubeadm.git
    cd salt-kubeadm/
    cp -r * /srv/
    /bin/cp /srv/roster /etc/salt/roster
    /bin/cp /srv/master /etc/salt/master
    ```

Warning:

If you see the `unable to resolve host address ‘mirrors.aliyun.com’`, you should add the `nameserver 8.8.8.8` to the `/etc/resolv.conf`, `8.8.8.8` is google's DNS.

If the clone fails, you can run the following command in node1

```
alias vmproxy='export http_proxy=http://proxy.vmware.com:3128; export https_proxy=http://proxy.vmware.com:
3128'
alias unvmproxy='export http_proxy=; export https_proxy='
```

2. **Use salt ssh manage the VM and roles**
    a. change salt config

    ```
    vim /etc/salt/roster (Check the ssh key and k8s-roles, if need, you can change it)
    vim /srv/pillar/k8s.sls (You can change this config base your request, but we use the default
    config, you can skip this step)
    ```

    b. test the salt ssh

    ```
    // This will install the python,close the swap check for the k8s, and sync the date of the system,
    you can change the `ntpdate time1.aliyun.com` to `ntpdate time.apple.com`
    salt-ssh -i '*' -r 'yum install -y python3 && swapoff -a && ntpdate time1.aliyun.com' salt-ssh -i
    '*' test.ping // Check the ssh status, if True, the config is fine.
    linux-node2:
    True
    linux-node3:
    True
    linux-node1:
    True
    ```

3. **Cluster config**
    a. Config the k8s plugins

    ```
    // This command will config k8s plugins(kubeadmkubeletdocker) and sync them to the node2 and
    node3, this command will change the config file, you don't need to config them.
    salt-ssh '*' state.highstate
    ```

b. master node init

```
kubeadm init --config /etc/sysconfig/kubeadm.yml --ignore-preflight-errors=NumCPU
```

c. Config kubectl

```
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

d. config Flannel

All the nodes need the network plugin to confirm the pod has the same LAN. We choose the Flannel.

```
kubectl create -f /etc/sysconfig/kube-flannel.yml
```

e. add the node to the cluster

```
// This command will generate a token and discovery-token-ca-cert-hash, this need to use to the
following two commands
[node1] kubeadm token create --print-join-command

// Change the token and discovery-token-ca-cert-hash to your generate in above command.
[node2] kubeadm join 192.168.56.11:6443 --token qnlyhw.cr9n8jbpbkg94szj     --discovery-token-ca-
cert-hash sha256:cca103afc0ad374093f3f76b2f91963ac72eabea3d379571e88d403fc7670611

[node3] kubeadm join 192.168.56.11:6443 --token qnlyhw.cr9n8jbpbkg94szj     --discovery-token-ca-
cert-hash sha256:cca103afc0ad374093f3f76b2f91963ac72eabea3d379571e88d403fc7670611
```

Warning:
If you meet some error, you can check the date of the 3 vms, if the date same, please check if the token and discovery-token-ca-cert-hash are the same as the first command's result

4. **Test Kubernetes**

```
kubectl get node // Check the k8s node

kubectl run net-test --image=alpine sleep 360000 //create a pod

kubectl get pod -o wide  // check the pod status and ip

ping -c 1 10.2.12.2 // visit the pod use the ip
```

5. **Download istio(All the test cases in the istio)**

You can download the istio from your mac and copy it to node1 or run the following command, download from github.

```
wget https://github.com/istio/istio/releases/download/1.11.4/istio-1.11.4-linux-amd64.tar.gz
```

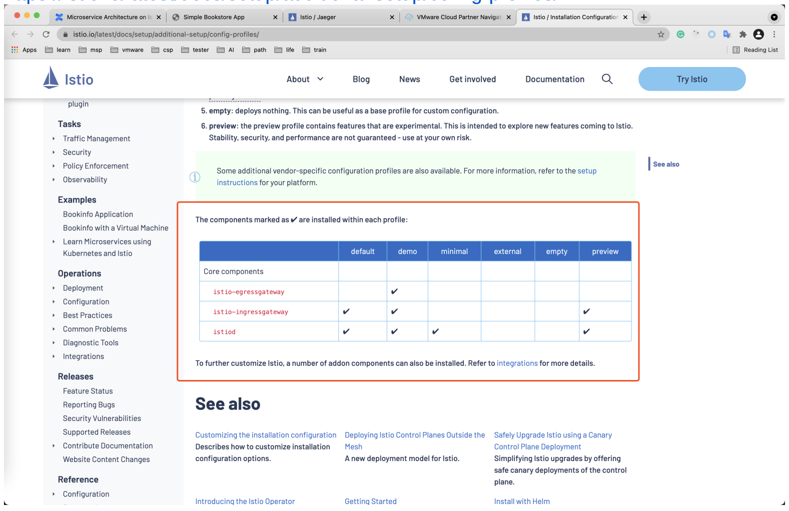
After you download, you should run the following command to config the istio to the bin

```
tar zxf istio-1.11.4-linux-amd64.tar.gz

cd istio-1.11.4

cp bin/istioctl /usr/local/bin/

istioctl install --set profile=demo -y // This use the demo's configuration profile
```

Add the label to the namespace, you must add this, istio will check this, if you missing this, will show unexpected exception

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

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



5. **empty**: deploys nothing. This can be useful as a base profile for custom configuration.
6. **preview**: the preview profile contains features that are experimental. This is intended to explore new features coming to Istio. Stability, security, and performance are not guaranteed - use at your own risk.

Some additional vendor-specific configuration profiles are also available. For more information, refer to the setup instructions for your platform.

The components marked as ✔ are installed within each profile:

	default	demo	minimal	external	empty	preview
Core components						
`istio-egressgateway`		✔				
`istio-ingressgateway`	✔	✔				✔
`istiod`	✔	✔	✔			✔

To further customize Istio, a number of addon components can also be installed. Refer to integrations for more details.

See also

Customizing the installation configuration
Describes how to customize installation configuration options.

Deploying Istio Control Planes Outside the Mesh
A new deployment model for Istio.

Safely Upgrade Istio using a Canary Control Plane Deployment
Simplifying Istio upgrades by offering safe canary deployments of the control plane.

Introducing the Istio Operator

Getting Started

Install with Helm

Demo istio case

The book project has 3 version's review, v1 doesn't have the start, v2 has the black start, v3 has the red start.

1. **Request Routing**

   https://istio.io/latest/docs/tasks/traffic-management/request-routing/

   We can use the virtual service config the rule, make the user visit the different version.

```
// Install the book deployment
kubectl apply -f /root/istio-1.11.4/samples/bookinfo/platform/kube/bookinfo.yaml
// You can use this link visit the book website. Check you svc ip and port, someone's deploy is in
192.168.56.12, port is random, remember change it. Confirm you can visit this, and refresh the chrome,
check the review block.
http://192.168.56.13:32760/productpage


// Case: make all the user visit v1
// Switch the review to v1, other services only have one version.
kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml
// Check the reviews block, you can see the page don't show the start(v1)
http://192.168.56.13:32760/productpage

// Case: Only the match rule's user can visit the v2, other user visit v1
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
http://192.168.56.13:32760/productpage // Check the reviews block. Login use jason, no passwork, check
the reviews block. The yaml said add the end-user in header, this is added in the productpage visit the
review, you can't find the end-user in the network.
```

2. **Fault Injection**

   https://istio.io/latest/docs/tasks/traffic-management/fault-injection/
   We can use this case test the fault affect the web

```
// Case: show the error when you login use jason, other users is fine.
// This is testing the timeout of the api, sometimes, api will limit the request time.
kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yaml
// Login jason and the page will show after 6-7s. The review block will show the error.
http://192.168.56.13:32760/productpage

Case: Test the error case
kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-abort.yaml
// Login jason and the reviews block will show the errors message.
http://192.168.56.13:32760/productpage
```

3. **Traffic Shifting**

   https://istio.io/latest/docs/tasks/traffic-management/traffic-shifting/
   We can migrate the partial user to the new version, if they are satific and not bug, we can migrate all the user to the new version.

```
// Reset before change, we need a clear deploy
kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.yaml

// 50% weight use the v3, 50% weight use the v1, you can refresh the http://192.168.56.13:32760
/productpage and check the reviews block.
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml

// If the v3 is ok, migrate all the user to the v3. Refresh the browser
kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-v3.yaml
```

4. **Request Timeouts**

   https://istio.io/latest/docs/tasks/traffic-management/request-timeouts/
   We can config the delay to check the api timeout

```
// visit v2, v2 set the api timeout 1s
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2
EOF
```

```
// Set the delay, check the review block. This review is fine, but this place will show after 2s.
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        percent: 100
        fixedDelay: 2s
    route:
    - destination:
        host: ratings
        subset: v1
EOF
```

```
// Add timeout, you will see the review show error. Timeout is 0.5s, but in poductpage, it retry to the
review, so call the review twice before it return.
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2
    timeout: 0.5s
EOF
```

5. Mirroring

https://istio.io/latest/docs/tasks/traffic-management/mirroring/
We can use this case test the test env use the actual env data

```
// Create two httpbin, they will accecpt the request and output the log
// httpbin1
cat <<EOF | istioctl kube-inject -f - | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin-v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        command: ["gunicorn", "--access-logfile", "-", "-b", "0.0.0.0:80", "httpbin:app"]
        ports:
        - containerPort: 80
EOF
// httpbin2
cat <<EOF | istioctl kube-inject -f - | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v2
  template:
    metadata:
      labels:
        app: httpbin
        version: v2
    spec:
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        command: ["gunicorn", "--access-logfile", "-", "-b", "0.0.0.0:80", "httpbin:app"]
        ports:
        - containerPort: 80
EOF

// Create service, this service don't select which httpbin, we control this in virtual service
kubectl create -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  labels:
    app: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
EOF
```

Sleep deployment

```
// Create the sleep, this will provide the curl to visit the httpbin1
cat <<EOF | istioctl kube-inject -f - | kubectl create -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sleep
  template:
    metadata:
      labels:
        app: sleep
    spec:
      containers:
      - name: sleep
        image: curlimages/curl
        command: ["/bin/sleep","3650d"]
        imagePullPolicy: IfNotPresent
EOF
```

Virtual Service

```
// Make the service visit v1
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
    - httpbin
  http:
  - route:
    - destination:
        host: httpbin
        subset: v1
      weight: 100
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: httpbin
spec:
  host: httpbin
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
EOF
```

Test config

```
// Vist the httpbin
export SLEEP_POD=$(kubectl get pod -l app=sleep -o jsonpath={.items..metadata.name})
kubectl exec "${SLEEP_POD}" -c sleep -- curl -sS http://httpbin:8000/headers

// check httpbin1 and httpbin2 log
export V1_POD=$(kubectl get pod -l app=httpbin,version=v1 -o jsonpath={.items..metadata.name})
kubectl logs "$V1_POD" -c httpbin
export V2_POD=$(kubectl get pod -l app=httpbin,version=v2 -o jsonpath={.items..metadata.name})
kubectl logs "$V2_POD" -c httpbin
You can see the httpbin2 don't have log, they are independent.
```

Now, you can see the log only send to the httpbin1, httpbin2 doesn't have the log, we need to config the morror.

Config Morror

```
// Add the mirror, then you can see the httpbin2 will see the log.
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
    - httpbin
  http:
  - route:
    - destination:
        host: httpbin
        subset: v1
      weight: 100
    mirror:
      host: httpbin
      subset: v2
    mirrorPercentage:
      value: 100.0
EOF
```

Test and clear

```
// Send the request and check the httpbin1 and httpbin2, you will see the log in httpbin2
kubectl exec "${SLEEP_POD}" -c sleep -- curl -sS http://httpbin:8000/headers
kubectl logs "$V1_POD" -c httpbin
kubectl logs "$V2_POD" -c httpbin

// Clear, this case will effect the log dashboard for the productpage
kubectl delete virtualservice httpbin
kubectl delete destinationrule httpbin
kubectl delete deploy httpbin-v1 httpbin-v2 sleep
kubectl delete svc httpbin
```

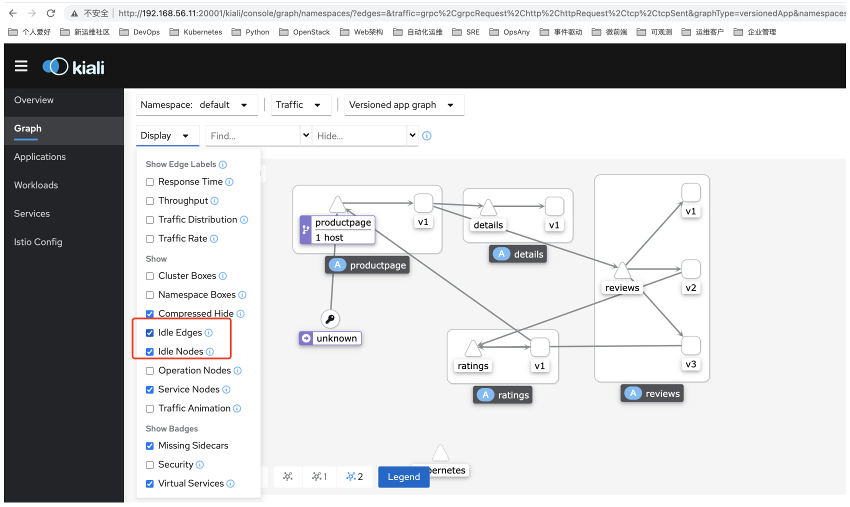
6. Kiali

Kiali is an observability console for Istio with service mesh configuration and validation capabilities. It helps you understand the structure and health of your service mesh by monitoring traffic flow to infer the topology and report errors.

```
kubectl port-forward svc/kiali --address 192.168.56.11 20001:20001 -n istio-system
```

Chrome: http://192.168.56.11:20001/

1. Open Graph,
2. click display, select the Idle Edges, Idle Nodes, and Service Nodes.
3. Select the legend at the bottom
4. Refresh: http://192.168.56.13:32760/productpage



http://192.168.56.11:20001/kiali/console/graph/namespaces/?edges=&traffic=grpc%2CgrpcRequest%2Chttp%2ChttpRequest%2Ctcp%2CtcpSent&graphType=versionedApp&namespaces

Namespace: default | Traffic | Versioned app graph

Display | Find... | Hide...

Show Edge Labels
- [ ] Response Time
- [ ] Throughput
- [ ] Traffic Distribution
- [ ] Traffic Rate

Show
- [ ] Cluster Boxes
- [ ] Namespace Boxes
- [✓] Compressed Hide
- [✓] Idle Edges
- [✓] Idle Nodes
- [ ] Operation Nodes
- [✓] Service Nodes
- [ ] Traffic Animation

Show Badges
- [✓] Missing Sidecars
- [ ] Security
- [✓] Virtual Services

7. **Prometheus**

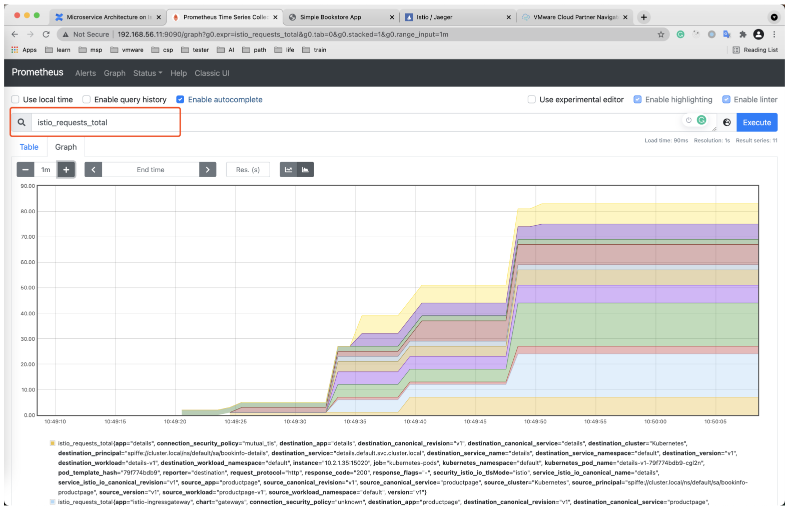
Prometheus is an open source monitoring system and time series database. You can use Prometheus with Istio to record metrics that track the health of Istio and of applications within the service mesh

```
kubectl port-forward svc/prometheus --address 192.168.56.11 9090:9090 -n istio-system
```

visit: http://192.168.56.11:9090/ you can select what you want to see, and then you will see the chart (istio_requests_total)

If you don't want to see the chart, you can see the metrics(This can be found in all the functions and data).
http://192.168.56.11:9090/metrics all the info also can be seen in this place,
Format:
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.67629352e+08
first line: show the description
second line: header
third line: data, this match the second line.

metrics have 4 types:
Counter(Only increase or reset to zero)
Gauge: (Can be increased or decrease)
Histogram: This use show the histogram chart
summary: Total



Not Secure | 192.168.56.11:9090/graph?g0.expr=istio_requests_total&g0.tab=0&g0.stacked=1&g0.range_input=1m

- [ ] Use local time
- [ ] Enable query history
- [✓] Enable autocomplete
- [ ] Use experimental editor
- [✓] Enable highlighting
- [✓] Enable linter

istio_requests_total

Load time: 90ms Resolution: 1s Result series: 11

🟨 istio_requests_total{**app**="details", **connection_security_policy**="mutual_tls", **destination_app**="details", **destination_canonical_revision**="v1", **destination_canonical_service**="details", **destination_cluster**="Kubernetes", **destination_principal**="spiffe://cluster.local/ns/default/sa/bookinfo-details", **destination_service**="details.default.svc.cluster.local", **destination_service_name**="details", **destination_service_namespace**="default", **destination_version**="v1", **destination_workload**="details-v1", **destination_workload_namespace**="default", **instance**="10.2.1.35:15020", **job**="kubernetes-pods", **kubernetes_namespace**="default", **kubernetes_pod_name**="details-v1-79f774bdb9-cgl2n", **pod_template_hash**="79f774bdb9", **reporter**="destination", **request_protocol**="http", **response_code**="200", **response_flags**="-", **security_istio_io_tlsMode**="istio", **service_istio_io_canonical_name**="details", **service_istio_io_canonical_revision**="v1", **source_app**="productpage", **source_canonical_revision**="v1", **source_canonical_service**="productpage", **source_cluster**="Kubernetes", **source_principal**="spiffe://cluster.local/ns/default/sa/bookinfo-productpage", **source_version**="v1", **source_workload**="productpage-v1", **source_workload_namespace**="default", **version**="v1"}

🟦 istio_requests_total{**app**="istio-ingressgateway", **chart**="gateways", **connection_security_policy**="unknown", **destination_app**="productpage", **destination_canonical_revision**="v1", **destination_canonical_service**="productpage",

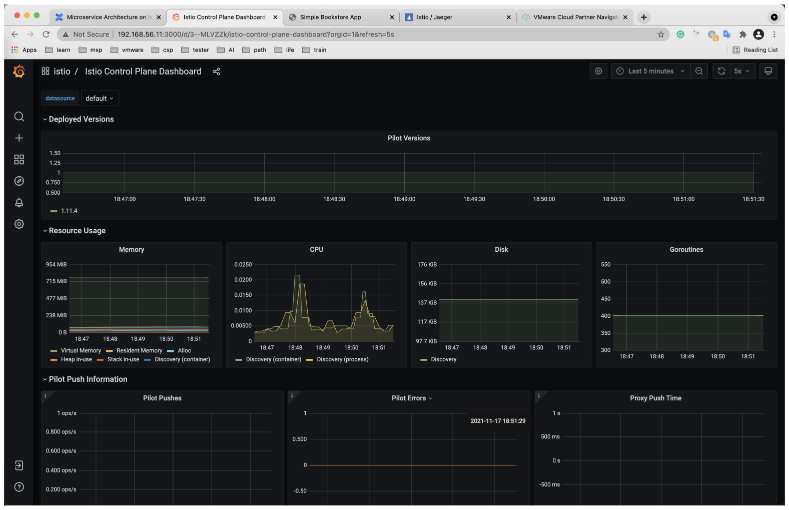
8. **Crafana**

Grafana is an open source monitoring solution that can be used to configure dashboards for Istio. You can use Grafana to monitor the health of Istio and of applications within the service mesh.

```
kubectl port-forward svc/grafana --address 192.168.56.11 3000:3000 -n istio-system
```

vist: http://192.168.56.11:3000/ -> dashboard -> manage -> open file -> select one -> if you don't see the data, please refresh http://192.168.56.13:32760/productpage

You can change the date range in the header, the default is 5 minutes.

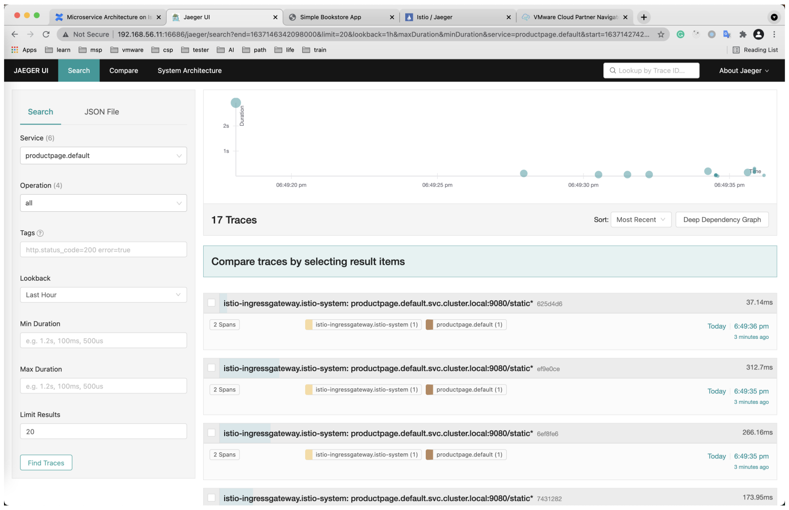


9. **jaeger**

Jaeger is an open source end to end distributed tracing system, allowing users to monitor and troubleshoot transactions in complex distributed systems.

```
istioctl dashboard --address 192.168.56.11 jaeger
```

This command will return a link, open it in the chrome, then you need select the service and click the find traces, you will see the data and chart.



Not Secure | 192.168.56.11:16686/jaeger/search?end=1637146342098000&limit=20&lookback=1h&maxDuration&minDuration&service=productpage.default&start=1637142742...
JAEGER UI
Search
Compare
System Architecture
Lookup by Trace ID...
About Jaeger
Search
JSON File
Find Traces
Deep Dependency Graph

Service (6): productpage.default
Operation (4): all
Tags: http.status_code=200 error=true
Lookback: Last Hour
Min Duration: e.g. 1.2s, 100ms, 500us
Max Duration: e.g. 1.2s, 100ms, 500us
Limit Results: 20

17 Traces — Sort: Most Recent

Compare traces by selecting result items

istio-ingressgateway.istio-system: productpage.default.svc.cluster.local:9080/static* 625d4d6 — 37.14ms
2 Spans — istio-ingressgateway.istio-system (1) — productpage.default (1) — Today 6:49:36 pm, 3 minutes ago

istio-ingressgateway.istio-system: productpage.default.svc.cluster.local:9080/static* ef9e0ce — 312.7ms
2 Spans — istio-ingressgateway.istio-system (1) — productpage.default (1) — Today 6:49:35 pm, 3 minutes ago

istio-ingressgateway.istio-system: productpage.default.svc.cluster.local:9080/static* 6ef8fe6 — 266.16ms
2 Spans — istio-ingressgateway.istio-system (1) — productpage.default (1) — Today 6:49:35 pm, 3 minutes ago

istio-ingressgateway.istio-system: productpage.default.svc.cluster.local:9080/static* 7431282 — 173.95ms