Clover

Release Latest

Dec 18, 2018

Contents

CHAPTER 1

Clover User Guide

1.1 Clover User Guide (Gambia Release)

This document provides the Clover user guide for the OPNFV Gambia release.

1.1.1 Description

Clover Gambia builds on previous release to further enhance the toolset for cloud native network functions operations. The two emphasis on the release are:

1. Integration of Spinnaker to support continuous delivery
2. Centralizing Operational Data for Visibility

1.1.2 What is in Gambia?

- Sample micro-service composed VNF named Service Delivery Controller (SDC)
- Istio 1.0 support
- clover-collector: gathers and collects metrics and traces from Prometheus and Jaeger, and provides a single access point for such data
- Visibility: utilizes an analytic engine to correlate and organize data collected by clover-collector
- cloverctl: Clover's new CLI
- Clovisor: Clover's cloud native, CNI-plugin agnostic network tracing tool
- Integration of HTTP Security Modules with Istio 1.0
- JMeter: integrating jmeter as test client
- Clover UI: sample UI to offer single pane view / configuration point of the Clover system

- Spinnaker Integration: add ability to add/update/delete cloud provider via cloverctl, and sample pipeline utilized by Clover project to deploy SDC

1.1.3 Usage

- Please refer to configguildes for usage detail on various modules

CHAPTER 2

OPNFV Clover Release Notes

This document provides Clover project's release notes for the OPNFV Gambia release.

2.1 Version history

Date	Ver.	Author	Comment
2018-03-14	Gambia 1.0	Stephen Wong	First draft

2.1.1 Important notes

The Clover project for OPNFV Gambia is tested on Kubernetes version 1.9 and 1.11. It is only tested on Istio 1.0.

2.1.2 Summary

Clover Gambia release further enhances the Fraser release by providing various tools to help operators deploy cloud native network functions. These tools include

1. Collector: gathers and collects metrics and traces from Prometheus and Jaeger, respectively, and provides a single access point for such data
2. Visibility: utilizes an analytic engine to correlate and organize data gathered by the collector
3. CLI: comprehensive Clover CLI called cloverctl, offering a single management tool for operating Clover toolset
4. Network Tracing: CNI plugin agnostic network tracing tool
5. Extended HTTP Security: integrate modsecurity (Web Application Firewall) and Snort with Istio gateway via Istio newly added mechanisms to redirect and mirror traffic to the network functions
6. HTTP Test Client: bundle JMeter as test client for testing
7. UI: developmental / sample UI to offer single pane view of Clover system
8. Spinnaker Integration: provides automated / programmable cloud provider add/update/delete; sample pipeline and installation scripts

2.1.3 Release Data

Version change

Module version changes

Clover Gambia release will no longer support Istio 0.6, the version of Istio supported by Clover Gambia release

Document version changes

Clover Gambia has updated the config guide and user guide accordingly, including new documents for the new features

Reason for version

Feature additions

See Summary above

Bug corrections

<None>

2.1.4 Known Limitations, Issues and Workarounds

System Limitations

TBD

Known issues

TBD

Workarounds

2.1.5 Test Result

2.1.6 References

CHAPTER 3

Clover Configuration Guide

3.1 Clover Controller Services Configuration Guide

This document provides a guide to use the Clover controller services, which are introduced in the Clover Gambia release.

3.1.1 Overview

Clover controller services allow users to control and access information about Clover microservices. Two new components are added to Clover to facilitate an ephemeral, cloud native workflow. A CLI interface with the name **cloverctl** interfaces to the Kubernetes (k8s) API and also to **clover-controller**, a microservice deployed within the k8s cluster to instrument other Clover k8s services including sample network services, visibility/validation services and supporting datastores (redis, cassandra). The **clover-controller** service provides message routing communicating REST with cloverctl or other API/UI interfaces and gRPC to internal k8s cluster microservices. It acts as an internal agent and reduces the need to expose multiple Clover services outside of a k8s cluster.

The **clover-controller** is packaged as a docker container with manifests to deploy in a Kubernetes (k8s) cluster. The **cloverctl** CLI is packaged as a binary (Golang) within a tarball with associated yaml files that can be used to configure and control other Clover microservices within the k8s cluster via **clover-controller**. The **cloverctl** CLI can also deploy/delete other Clover services within the k8s cluster for convenience.

The **clover-controller** service provides the following functions:

- **REST API:** interface allows CI scripts/automation to control sample network sample services, visibility and validation services. Analyzed visibility data can be consumed by other services with REST messaging.
- **CLI Endpoint:** acts as an endpoint for many **cloverctl** CLI commands using the **clover-controller** REST API and relays messages to other services via gRPC.
- **UI Dashboard:** provides a web interface exposing visibility views to interact with Clover visibility services. It presents analyzed visibility data and provides basic controls such as selecting which user services visibility will track.

The **cloverctl** CLI command syntax is similar to k8s kubectl or istio istioctl CLI tools, using a <verb> <noun> convention.

Help can be accessed using the `--help` option, as shown below:

```
$ cloverctl --help
```

3.1.2 Deploying Clover system services

Prerequisites

The following assumptions must be met before continuing on to deployment:

- Installation of Docker has already been performed. It’s preferable to install Docker CE.
- Installation of k8s in a single-node or multi-node cluster.

Download Clover CLI

Download the cloverctl binary from the location below:

```
$ curl -L https://github.com/opnfv/clover/raw/stable/gambia/download/cloverctl.tar.gz␣
↪| tar xz
$ cd cloverctl
$ export PATH=$PWD:$PATH
```

To begin deploying Clover services, ensure the correct k8s context is enabled. Validate that the CLI can interact with the k8s API with the command:

```
$ cloverctl get services
```

The command above must return a listing of the current k8s services similar to the output of ‘kubectl get svc –all-namespaces’.

Deploying clover-controller

To deploy the **clover-controller** service, use the command below:

```
$ cloverctl create system controller
```

The k8s pod listing below must include the **clover-controller** pod in the **clover-system** namespace:

```
$ kubectl get pods --all-namespaces | grep clover-controller

NAMESPACE      NAME                                    READY      STATUS
clover-system  clover-controller-74d8596bb5-jczqz      1/1        Running
```

3.1.3 Exposing clover-controller

To expose the **clover-controller** deployment outside of the k8s cluster, a k8s NodePort or LoadBalancer service must be employed.

Using NodePort

To use a NodePort for the **clover-controller** service, use the following command:

```
$ cloverctl create system controller nodeport
```

The NodePort default is to use port 32044. To modify this, edit the yaml relative to the **cloverctl** path at `yaml/controller/service_nodeport.yaml` before invoking the command above. Delete the `nodePort:` key in the yaml to let k8s select an available port within the the range 30000-32767.

Using LoadBalancer

For k8s clusters that support a LoadBalancer service, such as GKE, one can be created for **clover-controller** with the following command:

```
$ cloverctl create system controller lb
```

Setup with cloverctl CLI

The **cloverctl** CLI will communicate with **clover-controller** on the service exposed above and requires the IP address of either the load balancer or a cluster node IP address, if a NodePort service is used. For a LoadBalancer service, **cloverctl** will automatically find the IP address to use and no further action is required.

However, if a NodePort service is used, an additional step is required to configure the IP address for **cloverctl** to target. This may be the CNI (ex. flannel/weave) IP address or the IP address of an k8s node interface. The **cloverctl** CLI will automatically determine the NodePort port number configured. To configure the IP address, create a file named `.cloverctl.yaml` and add a single line to the yaml file with the following:

```
ControllerIP: <IP addresss>
```

This file must be located in your `HOME` directory or in the same directory as the **cloverctl** binary.

3.1.4 Uninstall from Kubernetes environment

Delete with Clover CLI

When you're finished working with Clover system services, you can uninstall it with the following command:

```
$ cloverctl delete system controller
$ cloverctl delete system controller nodeport # for NodePort
$ cloverctl delete system controller lb # for LoadBalancer
```

The commands above will remove the clover-controller deployment and service resources created from the current k8s context.

3.1.5 Uninstall from Docker environment

The OPNFV docker image for the **clover-controller** can be removed with the following commands from nodes in the k8s cluster.

```
$ docker rmi opnfv/clover-controller
```

3.2 Clover SDC Sample Configuration Guide

This document provides a guide to use the Service Delivery Controller (SDC) sample, which is initially delivered in the Clover Fraser release.

3.2.1 Overview

The SDC is a sample set of web-oriented network services that allow the flow of ingress HTTP traffic to be controlled and inspected in an Istio service mesh within Kubernetes. It provides the ability to demonstrate the Istio sandbox including service mesh concepts and surrounding tools including tracing, monitoring, and logging.

The SDC sample comprises the following services:

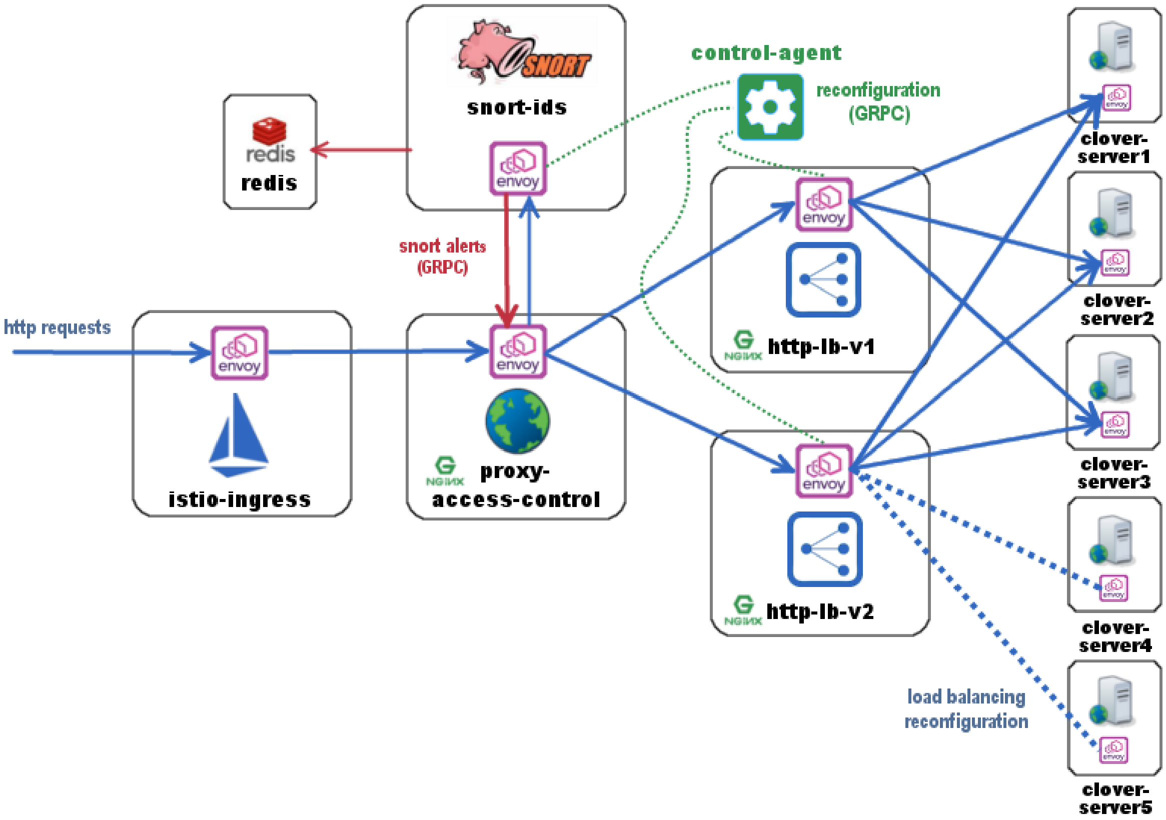
- **Proxy** - used to mirror traffic to security (**snort-ids**) services and propagate traffic to load balancing services. In future releases, the proxy will process security alerts and provide access control by blacklisting clients based on source IP address.
- **Load Balancer** - provides basic round-robin load balancing to other downstream services without Istio provisions. Istio features built-in load balancing to provide request routing for canary and A/B scenarios. The SDC sample employs both tiers of load balancing to demonstrate how load balancing algorithms can be controlled to address both network and application requirements.
- **Intrusion Detection System** - used to detect web security vulnerabilities using limited set of rules/signatures and send security alerts to the proxy.
- **Server** - simple web servers used to terminate web requests from the load balancing services to enable end-to-end traffic flow.

The table below shows key details of the sample Kubernetes manifest for the services outlined above:

Service	Kubernetes Deployment App Name	Docker Image	Ports
Proxy	proxy-access-control	clover-ns-nginx-proxy	HTTP: 9180 GRPC: 50054
Load Balancers	app: http-lb version: http-lb-v1 version: http-lb-v2	clover-ns-nginx-lb	HTTP: 9180 GRPC: 50054
Intrusion Detection System (IDS)	snort-ids	clover-ns-snort-ids	HTTP: 80, Redis: 6379 GRPC: 50052 (config) GRPC: 50054 (alerts)
Servers	clover-server1 clover-server2 clover-server3 clover-server4 clover-server5	clover-ns-nginx-server	HTTP: 9180 GRPC: 50054

Additionally, the sample uses other ancillary elements including:

- A Redis in-memory data store for the snort IDS service to write alerts. It can also be used by the Clover tracing module to analyze traces over time. Standard community containers of Redis are employed by Clover.
- A Kubernetes Ingress resource (**proxy-gateway**) to manage external access to the service mesh.
- Clover docker container that is used to invoke deployment and cleanup scripts for the sample. It can also be used to execute scripts that modify run-time service configurations. Using the container avoids the need to clone the source code.
- Optional deployment of Jaeger tracing and Prometheus monitoring tools with access to their browser-based UIs.



The diagram above shows the flow of web traffic where all blue arrows denote the path of incoming HTTP requests through the service mesh. Requests are directed to the istio-ingress entry point using the Ingress resource (**proxy-gateway**). Istio-ingress acts as a gateway and sends traffic to the **proxy-access-control** service. **Proxy-access-control** mirrors traffic to the **snort-ids** service for it to monitor all incoming HTTP requests. The **snort-ids** asynchronously sends alert notifications to **proxy-access-control** over GRPC on port 50054, which is denoted in red, and stores the details of the alert events into Redis for other services to potentially inspect.

Proxy-access-control also sends traffic to the **http-lb** load balancing service. **Http-lb** deploys two versions (**http-lb-v1**, **http-lb-v2**) of itself by sharing the same app name (**http-lb**) but using a distinct version in the Kubernetes manifest. By default, without any further configuration, Istio will load balance requests with a 50/50 percentage split among these two **http-lb** versions. Both the load balancers are internally configured by default to send traffic to **clover-server1/2/3** in round-robin fashion.

A controlling agent that can reside inside or outside of the mesh can be used to modify the run-time configuration of the services, which is denoted in green. Python sample scripts that implement a GRPC client act as a control-agent and are used to reconfigure **http-lb-v2** to load balance across **clover-server4/5** instead of servers 1/2/3. The sample provides additional examples of modifying run-time configurations such as adding user-defined rules to the **snort-ids** service to trigger alerts on other network events.

3.2.2 Deploying the sample

Prerequisites

The following assumptions must be met before continuing on to deployment:

- Ubuntu 16.04 was used heavily for development and is advised for greenfield deployments.
- Installation of Docker has already been performed. It's preferable to install Docker CE.
- Installation of Kubernetes has already been performed. The installation in this guide was executed in a single-node Kubernetes cluster.
- Installation of a pod network that supports the Container Network Interface (CNI). It is recommended to use flannel, as most development work employed this network add-on. Success using Weave Net as the CNI plugin has also been reported.
- Installation of Istio and Istio client (istioctl) is in your PATH (for deploy from source)

Deploy with Clover container

The easiest way to deploy the sample into your Kubernetes cluster is to use the Clover container by pulling the container and executing a top-level deploy script using the following two commands:

```
$ docker pull opnfv/clover:<release_tag>
```

The <release_tag> is **opnfv-7.0.0** for the Gambia release. However, the latest will be pulled if the tag is unspecified. To deploy the Gambia release use these commands:

```
$ docker pull opnfv/clover:opnfv-7.0.0
$ sudo docker run --rm \
-v ~/.kube/config:/root/.kube/config \
opnfv/clover \
/bin/bash -c '/home/opnfv/repos/clover/samples/scenarios/deploy.sh'
```

The deploy script invoked above begins by installing Istio 1.0.0 into your Kubernetes environment. It proceeds to deploy the entire SDC manifest. If you've chosen to employ this method of deployment, you may skip the next section.

Deploy from source

Ensure Istio 1.0.0 is installed, as a prerequisite, using the following commands:

```
$ curl -L https://github.com/istio/istio/releases/download/1.0.0/istio-1.0.0-linux.
↪tar.gz | tar xz
$ cd istio-1.0.0
$ export PATH=$PWD/bin:$PATH
$ kubectl apply -f install/kubernetes/istio-demo.yaml
```

The above sequence of commands installs Istio with manual sidecar injection without mutual TLS authentication between sidecars.

To continue to deploy from the source code, clone the Clover git repository and navigate within the samples directory as shown below:

```
$ git clone https://gerrit.opnfv.org/gerrit/clover
$ cd clover/samples/scenarios
$ git checkout stable/gambia
```

To deploy the sample in the default Kubernetes namespace, use the following command for Istio manual sidecar injection:

```
$ istioctl kube-inject -f service_delivery_controller_opnfv.yaml | kubectl apply -f -
```

To deploy in another namespace, use the ‘-n’ option. An example namespace of ‘sdc’ is shown below:

```
$ kubectl create namespace sdc
$ istioctl kube-inject -f service_delivery_controller_opnfv.yaml | kubectl apply -n␣
↪sdc -f -
```

When using the above SDC manifest, all required docker images will automatically be pulled from the OPNFV public Dockerhub registry. An example of using a Docker local registry is also provided in the `/clover/samples/scenario` directory.

Verifying the deployment

To verify the entire SDC sample is deployed, ensure the following pods have been deployed with the command below:

```
$ kubectl get pod --all-namespaces
```

The listing below must include the following SDC pods assuming deployment in the default Kubernetes namespace:

```
$ NAMESPACE      NAME                                    READY     STATUS
default        clover-server1-68c4755d9c-7s5q8           2/2       Running
default        clover-server2-57d8b786-rf5x7             2/2       Running
default        clover-server3-556d5f79cf-hk6rv           2/2       Running
default        clover-server4-6d9469b884-8srbk           2/2       Running
default        clover-server5-5d64f74bf-l7wqc            2/2       Running
default        http-lb-v1-59946c5744-w658d               2/2       Running
default        http-lb-v2-5df78b6849-splp9               2/2       Running
default        proxy-access-control-6b564b95d9-jg5wm     2/2       Running
default        redis                                     2/2       Running
default        snort-ids-5cc97fc6f-zhh5l                 2/2       Running
```

The result of the Istio deployment must include the following pods:

```
$ NAMESPACE    NAME                                             READY     STATUS
istio-system   grafana-6995b4fbd7-pjgbh                         1/1       Running
istio-system   istio-citadel-54f4678f86-t2dng                   1/1       Running
istio-system   istio-egressgateway-5d7f8fcc7b-hs7t4             1/1       Running
istio-system   istio-galley-7bd8b5f88f-wtrdv                    1/1       Running
istio-system   istio-ingressgateway-6f58fdc8d7-vqwzj            1/1       Running
istio-system   istio-pilot-d99689994-b48nz                      2/2       Running
istio-system   istio-policy-766bf4bd6d-l89vx                    2/2       Running
istio-system   istio-sidecar-injector-85ccf84984-xpmxp          1/1       Running
istio-system   istio-statsd-prom-bridge-55965ff9c8-q25rk        1/1       Running
istio-system   istio-telemetry-55b6b5bbc7-qrg28                 2/2       Running
istio-system   istio-tracing-77f9f94b98-zljrt                   1/1       Running
istio-system   prometheus-7456f56c96-zjd29                      1/1       Running
istio-system   servicegraph-684c85ffb9-9h6p7                    1/1       Running
```

Determining the ingress IP and port

To determine how incoming http traffic on port 80 will be translated, use the following command:

```
$ kubectl get svc -n istio-system | grep LoadBalancer
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
istio-ingressgateway   LoadBalancer   10.111.40.165    <pending>     80:32410/TCP,
↪443:31390/TCP
```

Note, the CLUSTER-IP of the service will be unused in this example since load balancing service types are unsupported in this configuration. It is normal for the EXTERNAL-IP to show status <pending> indefinitely

In this example, traffic arriving on port 32410 will flow to istio-ingressgateway. The istio-ingressgateway service will route traffic to the **proxy-access-control** service based on configured Istio `Gateway` and `VirtualService` resources, which are shown below. The `Gateway` defines a gateway for external traffic to enter the Istio service mesh based on incoming protocol, port and domain (`hosts:` section currently using wildcard). The `VirtualService` associates to a particular `Gateway` (sdc-gateway here) and allows for route rules to be setup. In the example below, any URL with prefix '/' will be routed to the service **proxy-access-control** on port 9180. Additionally, ingress traffic can be mirrored by adding a directive to the `VirtualService` definition. Below, all matching traffic will be mirrored to the **snort-ids** (duplicating internal mirroring performed by the **proxy-access-control** for illustrative purposes)

This allows the traffic management and policy features of Istio available to external services and clients.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: sdc-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: sdcsample
spec:
  hosts:
  - "*"
  gateways:
  - sdc-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        host: proxy-access-control
        port:
          number: 9180
    mirror:
      host: snort-ids
```

3.2.3 Using the sample

To confirm the scenario is running properly, HTTP GET requests can be made from an external host with a destination of the Kubernetes cluster. Requests can be invoked from the host OS of the Kubernetes cluster. Modify the port used below (32410) with the port obtained from section *Determining the ingress IP and port*. If flannel is being used, requests can use the default flannel CNI IP address, as shown below:

```
$ wget http://10.244.0.1:32410/
$ curl http://10.244.0.1:32410/
```

An IP address of a node within the Kubernetes cluster may also be employed.

An HTTP response will be returned as a result of the wget or curl command, if the SDC sample is operating correctly. However, the visibility into what services were accessed within the service mesh remains hidden. The next section *Exposing tracing and monitoring* shows how to inspect the internals of the Istio service mesh.

Exposing tracing and monitoring

The Jaeger tracing UI is exposed outside of the Kubernetes cluster via any node IP in the cluster using the following commands **(above command already executes the two commands below)**:

```
$ kubectl expose -n istio-system deployment istio-tracing --port=16686 --type=NodePort
```

Likewise, the Prometheus monitoring UI is exposed with the following command:

```
$ kubectl expose -n istio-system deployment prometheus --port=9090 --type=NodePort
```

To find the ports the Jaeger tracing and Prometheus monitoring UIs are exposed on, use the following command:

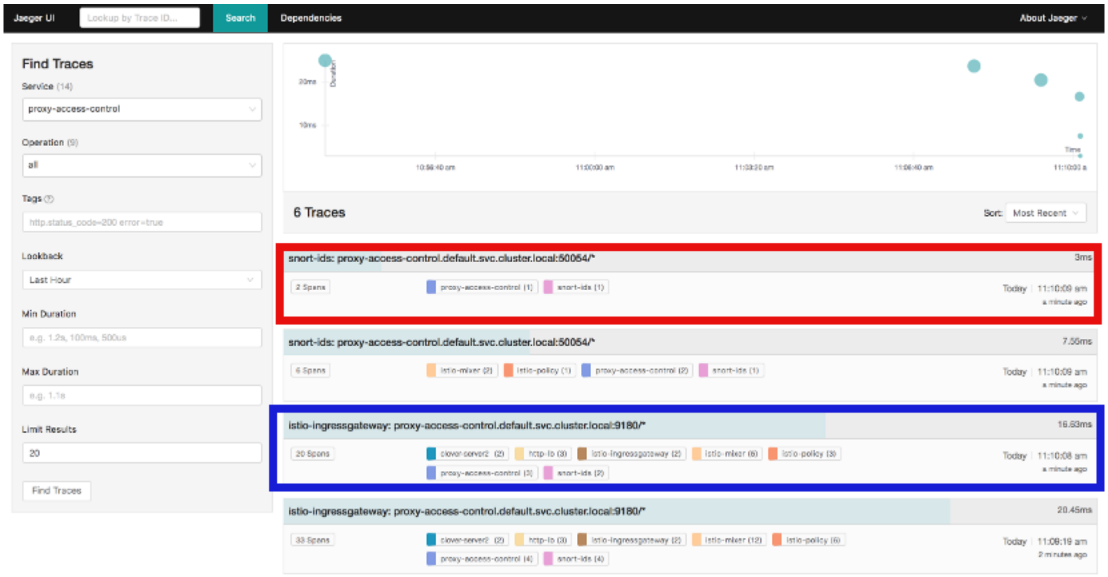
```
$ kubectl get svc -n istio-system | grep NodePort
NAMESPACE      NAME              TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)
istio-system   istio-tracing     NodePort  10.105.94.85 <none>        16686:32174/TCP
istio-system   prometheus        NodePort  10.97.74.230 <none>        9090:32708/TCP
```

In the example above, the Jaeger tracing web-based UI will be available on port 32174 and the Prometheus monitoring UI on port 32708. In your browser, navigate to the following URLs for Jaeger and Prometheus respectively:

```
http://<node IP>:32174
http://<node IP>:32708
```

Where node IP is an IP of one of the Kubernetes cluster node(s) on a CNI IP address. Alternatively, the tracing and monitoring services can be exposed with a LoadBalancer service if supported by your Kubernetes cluster (such as GKE), as shown below for tracing:

```
kind: Service
apiVersion: v1
metadata:
  name: istio-tracing
spec:
  selector:
    app: jaeger
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 16686
  type: LoadBalancer
```



Service (14): proxy-access-control
Operation (9): all
Tags: http.status_code=200 error=true
Lookback: Last Hour
Min Duration: e.g. 1.2s, 100ms, 500us
Max Duration: e.g. 1.1s
Limit Results: 20

6 Traces
Sort: Most Recent

snort-ids: proxy-access-control.default.svc.cluster.local:50054/* 3ms
2 Spans proxy-access-control (1) snort-ids (1) Today | 11:10:09 am a minute ago

snort-ids: proxy-access-control.default.svc.cluster.local:50054/* 7.56ms
6 Spans istio-mixer (2) istio-policy (1) proxy-access-control (2) snort-ids (1) Today | 11:10:09 am a minute ago

istio-ingressgateway: proxy-access-control.default.svc.cluster.local:9180/* 16.63ms
20 Spans clover-server2 (2) http-lb (3) istio-ingressgateway (2) istio-mixer (6) istio-policy (3) proxy-access-control (3) snort-ids (2) Today | 11:10:08 am a minute ago

istio-ingressgateway: proxy-access-control.default.svc.cluster.local:9180/* 20.45ms
33 Spans clover-server2 (2) http-lb (3) istio-ingressgateway (2) istio-mixer (12) istio-policy (6) proxy-access-control (4) snort-ids (4) Today | 11:09:19 am 2 minutes ago

The diagram above shows the Jaeger tracing UI after traces have been fetched for the **proxy-access-control** service. After executing an HTTP request using the simple curl/wget commands outlined in *Using the sample* , a list of SDC services will be displayed in the top left drop-down box labelled `Service`. Choose `proxy-access-control` in the drop-down and click the `Find Traces` button at the bottom of the left controls. The blue box denotes what should be displayed for the services that were involved in handling the request including:

- istio-ingressgateway
- proxy-access-control
- snort-ids
- http-lb
- clover-server1 OR clover-server2 OR clover-server3

The individual traces can be clicked on to see the details of the messages between services.

3.2.4 Modifying the run-time configuration of services

The following control-plane actions can be invoked via GRPC messaging from a controlling agent. For this example, it is conducted from the host OS of a Kubernetes cluster node using Clover system services. This requires **clover-controller** and **cloverctl** CLI be deployed. See instructions at *Deploying clover-controller*.

Modifying the http-lb server list

By default, both versions of the load balancers send incoming HTTP requests to **clover-server1/2/3** in round-robin fashion. To have the version 2 load balancer (**http-lb-v2**) send its traffic to **clover-server4/5** instead, issue the following command from the **cloverctl** CLI:

```
$ cloverctl set lb -f lbv2.yaml
```

The `lbv2.yaml` is available from the yaml directory relative to the **cloverctl** binary.

If the command executes successfully, the return message should appear as below:

```
Modified nginx config
```

If several more HTTP GET requests are subsequently sent to the ingress, the Jaeger UI should begin to display requests flowing to **clover-server4/5** from **http-lb-v2**. The **http-lb-v1** version of the load balancer will still balance requests to **clover-server1/2/3**.

Adding rules to snort-ids

The snort service installs the readily available community rules. An initial, basic provision to allow custom rule additions has been implemented within this release. A custom rule will trigger alerts and can be defined in order to inspect network traffic. This capability, including rule manipulation, will be further expounded upon in subsequent releases. For the time being, the following basic rule additions can be performed using a client sample script.

A snort IDS alert can be triggered by adding the HTTP User-Agent string shown below. The signature that invokes this alert is part of the community rules that are installed in the snort service by default. Using the curl or wget commands below, an alert can be observed using the Jaeger tracing browser UI. It will be displayed as a GRPC message on port 50054 from the **snort-ids** service to the **proxy-access-control** service. The red box depicted in the Jaeger UI diagram in section *Exposing tracing and monitoring* shows what should be displayed for the alerts. Drilling down into the trace will show a GPRC message from snort with HTTP URL `http://proxy-access-control:50054/nginx.Controller/ProcessAlerts`.

```
$ wget -U 'asafaweb.com' http://10.244.0.1:32410/
```

Or alternatively with curl, issue this command to trigger the alert: : .. code-block:: bash

> $ curl -A ‘asafaweb.com’ http://10.244.0.1:32410/

The community rule can be copied to local rules in order to ensure an alert is generated each time the HTTP GET request is observed by snort using the following commands from the **cloverctl** CLI:

```
$ cloverctl create idsrules -f idsrule_scan.yaml
$ cloverctl stop ids
$ cloverctl start ids
```

The `idsrule_scan.yaml` is available from the yaml directory relative to the **cloverctl** binary. Successful completion of the above commands will yield output similar to the following:

```
Added to local rules
Stopped Snort on pid: 48, Cleared Snort logs
Started Snort on pid: 155
```

To add an ICMP rule to snort service, use the following command:

```
$ cloverctl create idsrules -f idsrule_icmp.yaml
$ cloverctl stop ids
$ cloverctl start ids
```

The `idsrule_icmp.yaml` is available from the yaml directory relative to the **cloverctl**

Successful execution of the above commands will trigger alerts whenever ICMP packets are observed by the snort service. An alert can be generated by pinging the snort service using the flannel IP address assigned to the **snort-ids** pod. The Jaeger UI can again be inspected and should display the same `ProcessAlert` messages flowing from the **snort-ids** to the **proxy-access-control** service for ICMP packets.

3.2.5 Advanced Usage

Inspect Redis

This section assumes alert messages have already been successfully generated from the **snort-ids** service using the instructions outlined in section *Adding rules to snort-ids*.

The **snort-ids** service writes the details of alert events into a Redis data store deployed within the Kubernetes cluster. This event and packet data can be inspected by first installing the `redis-tools` Linux package on one of the nodes within the Kubernetes cluster. For a Ubuntu host OS, this can be performed with the following command:

```
$ sudo apt-get install redis-tools
```

Assuming a flannel CNI plugin, Redis can then be accessed by finding the IP assigned to the Redis pod with the command:

```
$ kubectl get pod --all-namespaces -o wide
NAMESPACE      NAME        READY     STATUS    RESTARTS   AGE       IP
default        redis       2/2       Running   0          2d        10.244.0.176
```

In the example listing above, the Redis pod IP is at 10.244.0.176. This IP can be used to access the Redis CLI with the command:

```
$ redis-cli -h 10.244.0.176
10.244.0.176:6379>
```

The redis CLI prompt ensues and the alert event indexes can be fetched with the Redis `SMEMBERS` set command with the key **snort_events** for the argument, as shown below:

```
10.244.0.176:6379> SMEMBERS snort_events
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
```

The individual alert details are stored as Redis hashes and can be retrieved with the Redis `HGETALL` hash command to get the values of the entire hash with key **snort_event:1** formed by using the prefix of **snort_event:** concatenated with an index retrieved from the prior listing output from the `SMEMBERS` command, as shown below:

```
10.244.0.176:6379> HGETALL snort_event:1
1)  "blocked"
2)  "0"
3)  "packet-microsecond"
4)  "726997"
5)  "packet-second"
6)  "1524609217"
7)  "pad2"
8)  "None"
9)  "destination-ip"
10) "10.244.0.183"
11) "signature-revision"
12) "1"
13) "signature-id"
14) "10000001"
15) "protocol"
```

(continues on next page)

(continued from previous page)

```
16) "1"
17) "packets"
18) "[]"
19) "source-ip.raw"
20) "\n\xf4\x00\x01"
21) "dport-icode"
22) "0"
23) "extra-data"
24) "[]"
25) "length"
26) "98"
27) "priority"
28) "0"
29) "linktype"
30) "1"
31) "classification-id"
32) "0"
33) "event-id"
34) "1"
35) "destination-ip.raw"
36) "\n\xf4\x00\xb7"
37) "generator-id"
38) "1"
39) "appid"
40) "None"
41) "sport-itype"
42) "8"
43) "event-second"
44) "1524609217"
45) "impact"
46) "0"
47) "data"
48)
↪"\nX\n\xf4\x00\xb7\nX\n\xf4\x00\x01\b\x00E\x00\x00T\x95\x82@\x00@\x01\x8e\x87\n\xf4\x00\x01\n\xf4\
↪{\x02\x00\x01\xc1\xb0\xdfZ\x00\x00\x00\x00\xbe\x17\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15
↪!\"#$%&'()*+,-./01234567"
49) "mpls-label"
50) "None"
51) "sensor-id"
52) "0"
53) "vlan-id"
54) "None"
55) "event-microsecond"
56) "726997"
57) "source-ip"
58) "10.244.0.1"
59) "impact-flag"
60) "0"
```

The alert above was generated for an ICMP packet after adding the custom rule for ICMP outlined in section *Adding rules to snort-ids*. The ICMP rule/signature ID that was used when adding the custom rule is `10000001` and is output in the above listing.

To exit the Redis CLI, use the command `exit`.

3.2.6 Uninstall from Kubernetes envionment

Delete with Clover container

When you're finished working on the SDC sample, you can uninstall it with the following command:

```
 $ sudo docker run --rm \
-v ~/.kube/config:/root/.kube/config \
opnfv/clover \
/bin/bash -c '/home/opnfv/repos/clover/samples/scenarios/clean.sh'
```

The command above will remove the SDC sample services, Istio components and Jaeger/Prometheus tools from your Kubernetes environment.

Delete from source

The SDC sample services can be uninstalled from the source code using the commands below:

```
$ cd clover/samples/scenarios
$ kubectl delete -f service_delivery_controller_opnfv.yaml

pod "redis" deleted
service "redis" deleted
deployment "clover-server1" deleted
service "clover-server1" deleted
deployment "clover-server2" deleted
service "clover-server2" deleted
deployment "clover-server3" deleted
service "clover-server3" deleted
deployment "clover-server4" deleted
service "clover-server4" deleted
deployment "clover-server5" deleted
service "clover-server5" deleted
deployment "http-lb-v1" deleted
deployment "http-lb-v2" deleted
service "http-lb" deleted
deployment "snort-ids" deleted
service "snort-ids" deleted
deployment "proxy-access-control" deleted
service "proxy-access-control" deleted
ingress "proxy-gateway" deleted
```

Istio components will not be uninstalled with the above command, which deletes using the SDC manifest file. To remove the Istio installation, navigate to the root directory where Istio was installed from source and use the following command:

```
$ cd istio-1.0.0
$ kubectl delete -f install/kubernetes/istio-demo.yaml
```

3.2.7 Uninstall from Docker environment

The OPNFV docker images can be removed with the following commands:

```
$ docker rmi opnfv/clover-ns-nginx-proxy
$ docker rmi opnfv/clover-ns-nginx-lb
$ docker rmi opnfv/clover-ns-nginx-server
$ docker rmi opnfv/clover-ns-snort-ids
$ docker rmi opnfv/clover
```

If deployment was performed with the Clover container, the first four images above will not be present. The Redis docker images can be removed with the following commands, if deployed from source:

```
$ docker rmi k8s.gcr.io/redis
$ docker rmi kubernetes/redis
```

If docker images were built locally, they can be removed with the following commands:

```
$ docker rmi localhost:5000/clover-ns-nginx-proxy
$ docker rmi clover-ns-nginx-proxy
$ docker rmi localhost:5000/clover-ns-nginx-lb
$ docker rmi clover-ns-nginx-lb
$ docker rmi localhost:5000/clover-ns-nginx-server
$ docker rmi clover-ns-nginx-server
$ docker rmi localhost:5000/clover-ns-snort-ids
$ docker rmi clover-ns-snort-ids
```

3.3 JMeter Validation Configuration Guide

This document provides a guide to use the JMeter validation service, which is introduced in the Clover Gambia release.

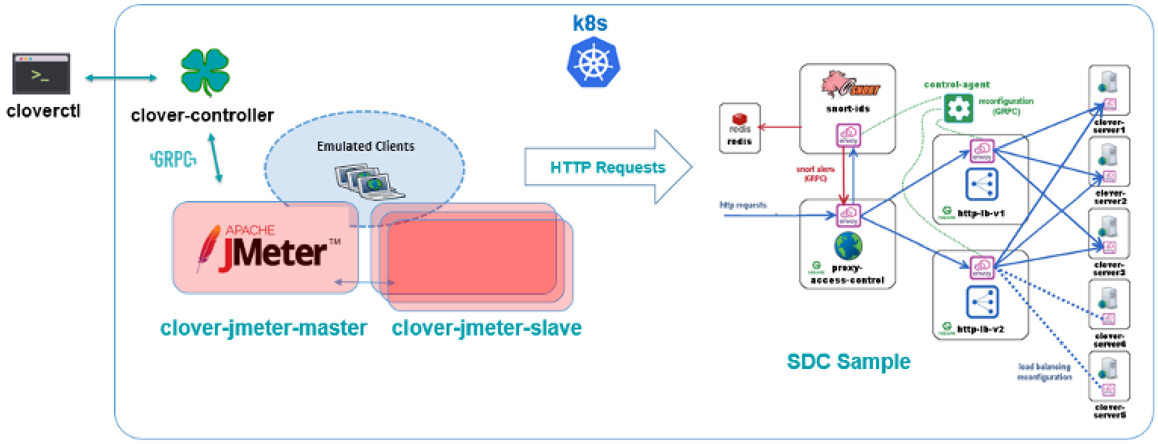
3.3.1 Overview

Apache JMeter is a mature, open source application that supports web client emulation. Its functionality has been integrated into the Clover project to allow various CI validations and performance tests to be performed. The system under test can either be REST services/APIs directly or a set of L7 network services. In the latter scenario, Clover nginx servers may be employed as an endpoint to allow traffic to be sent end-to-end across a service chain.

The Clover JMeter integration is packaged as docker containers with manifests to deploy in a Kubernetes (k8s) cluster. The Clover CLI (**cloverctl**) can be used to configure and control the JMeter service within the k8s cluster via **clover-controller**.

The Clover JMeter integration has the following attributes:

- **Master/Slave Architecture:** uses the native master/slave implementation of JMeter. The master and slaves have distinct OPNFV docker containers for rapid deployment and usage. Slaves allow the scale of the emulation to be increased linearly for performance testing. However, for functional validations and modest scale, the master may be employed without any slaves.
- **Test Creation & Control:** JMeter makes use of a rich XML-based test plan. While this offers a plethora of configurable options, it can be daunting for a beginner user to edit directly. Clover provides an abstracted yaml syntax exposing a subset of the available configuration parameters. JMeter test plans are generated on the master and tests can be started from **cloverctl** CLI.
- **Result Collection:** summary log results and detailed per-request results can be retrieved from the JMeter master during and after tests from the **cloverctl** or from a REST API exposed via **clover-controller**.



3.3.2 Deploying Clover JMeter service

Prerequisites

The following assumptions must be met before continuing on to deployment:

- Installation of Docker has already been performed. It's preferable to install Docker CE.
- Installation of k8s in a single-node or multi-node cluster.
- Clover CLI (**cloverctl**) has been downloaded and setup. Instructions to deploy can be found at *Deploying clover-controller*
- The **clover-controller** service is deployed in the k8s cluster the validation services will be deployed in. Instructions to deploy can be found at *Deploying clover-controller*.

Deploy with Clover CLI

The easiest way to deploy Clover JMeter validation services into your k8s cluster is to use the **cloverctl** CLI using the following command:

```
$ cloverctl create system validation
```

Container images with the Gambia release tag will pulled if the tag is unspecified. The release tag is **opnfv-7.0.0** for the Gambia release. To deploy the latest containers from master, use the command shown below:

```
$ cloverctl create system validation -t latest
```

The Clover CLI will add master/slave pods to the k8s cluster in the default namespace.

The JMeter master/slave docker images will automatically be pulled from the OPNFV public Dockerhub registry. Deployments and respective services will be created with three slave replica pods added with the **clover-jmeter-slave** prefix. A single master pod will be created with the **clover-jmeter-master** prefix.

Deploy from source

To continue to deploy from the source code, clone the Clover git repository and navigate within to the directory, as shown below:

```
$ git clone https://gerrit.opnfv.org/gerrit/clover
$ cd clover/clover/tools/jmeter/yaml
$ git checkout stable/gambia
```

To deploy the master use the following two commands, which will create a manifest with the Gambia release tags and creates the deployment in the k8s cluster:

```
$ python render_master.py --image_tag=opnfv-7.0.0 --image_path=opnfv
$ kubectl create -f clover-jmeter-master.yaml
```

JMeter can be injected into an Istio service mesh. To deploy in the default namespace within the service mesh, use the following command for manual sidecar injection:

```
$ istioctl kube-inject -f clover-jmeter-master.yaml | kubectl apply -f -
```

Note, when injecting JMeter into the service mesh, only the master will function for the Clover integration, as master-slave communication is known not to function with the Java RMI API. Ensure 'istioctl' is in your path for the above command.

To deploy slave replicas, render the manifest yaml and create in k8s adjusting the `--replica_count` value for the number of slave pods desired:

```
$ python render_slave.py --image_tag=opnfv-7.0.0 --image_path=opnfv --replica_count=3
$ kubectl create -f clover-jmeter-slave.yaml
```

Verifying the deployment

To verify the validation services are deployed, ensure the following pods are present with the command below:

```
$ kubectl get pod --all-namespaces
```

The listing below must include the following pods assuming deployment in the default namespace:

```
NAMESPACE   NAME                                      READY     STATUS
default     clover-jmeter-master-688677c96f-8nnnr      1/1      Running
default     clover-jmeter-slave-7f9695d56-8xh67        1/1      Running
default     clover-jmeter-slave-7f9695d56-fmpz5        1/1      Running
default     clover-jmeter-slave-7f9695d56-kg76s        1/1      Running
default     clover-jmeter-slave-7f9695d56-qfgqj        1/1      Running
```

3.3.3 Using JMeter Validation

Creating a test plan

To employ a test plan that can be used against the *Clover SDC Sample Configuration Guide* sample, navigate to cloverctl yaml directory and use the sample named 'jmeter_testplan.yaml', which is shown below.

```
load_spec:
    num_threads: 5
    loops: 2
    ramp_time: 60
    duration: 80
url_list:
```

(continues on next page)

(continued from previous page)

```
  - name: url1
    url: http://proxy-access-control.default:9180
    method: GET
    user-agent: chrome
  - name: url2
    url: http://proxy-access-control.default:9180
    method: GET
    user-agent: safari
```

The composition of the yaml file breaks down as follows:

- `load_spec` section of the yaml defines the load profile of the test.
- *num_threads*‘ parameter defines the maximum number of clients/users the test will emulate.
- `ramp_time` determines the rate at which threads/users will be setup.
- `loop` parameter reruns the same test and can be set to 0 to loop forever.
- `duration` parameter is used to limit the test run time and be used as a hard cutoff when using loop forever.
- `url_list` section of the yaml defines a set of HTTP requests that each user will perform. It includes the request URL that is given a name (used as reference in detailed per-request results) and the HTTP method to use (ex. GET, POST). The `user-agent` parameter allows this HTTP header to be specified per request and can be used to emulate browsers and devices.

The `url` syntax is <domain or IP>:<port #>. The colon port number may be omitted if port 80 is intended.

The test plan yaml is an abstraction of the JMeter XML syntax (uses .jmx extension) and can be pushed to the master using the **cloverctl** CLI with the following command:

```
$ cloverctl create testplan -f jmeter_testplan.yaml
```

The test plan can now be executed and will automatically be distributed to available JMeter slaves.

Starting the test

Once a test plan has been created on the JMeter master, a test can be started for the test plan with the following command:

```
$ cloverctl start testplan
```

The test will be executed from the **clover-jmeter-master** pod, whereby HTTP requests will originate directly from the master. The number of aggregate threads/users and request rates can be scaled by increasing the thread count or decreasing the ramp time respectively in the test plan yaml. However, the scale of the test can also be controlled by adding slaves to the test. When slaves are employed, the master will only be used to control slaves and will not be a source of traffic. Each slave pod will execute the test plan in its entirety.

To execute tests using slaves, add the flag ‘-s’ to the start command from the Clover CLI as shown below:

```
$ cloverctl start testplan -s <slave count>
```

The **clover-jmeter-slave** pods must be deployed in advance before executing the above command. If the steps outlined in section *Deploy with Clover CLI* have been followed, three slaves will have already been deployed.

Retrieving Results

Results for the test can be obtained by executing the following command:

```
$ cloverctl get testresult
$ cloverctl get testresult log
```

The bottom of the log will display a summary of the test results, as shown below:

```
3  in 00:00:00 =  111.1/s Avg:     7 Min:     6 Max:     8 Err:     0 (0.00%)
20 in 00:00:48 =    0.4/s Avg:    10 Min:     6 Max:    31 Err:     0 (0.00%)
```

Each row of the summary table is a snapshot in time with the final numbers in the last row. In this example, 20 requests (5 users/threads x 2 URLs) x loops) were sent successfully with no HTTP responses with invalid/error (4xx/5xx) status codes. Longer tests will produce a larger number of snapshot rows. Minimum, maximum and average response times are output per snapshot.

To obtain detailed, per-request results use the `detail` option, as shown below:

```
$ cloverctl get testresult detail

1541567388622,14,url1,200,OK,ThreadGroup 1-4,text,true,,843,0,1,1,14,0,0
1541567388637,8,url2,200,OK,ThreadGroup 1-4,text,true,,843,0,1,1,8,0,0
1541567388646,6,url1,200,OK,ThreadGroup 1-4,text,true,,843,0,1,1,6,0,0
1541567388653,7,url2,200,OK,ThreadGroup 1-4,text,true,,843,0,1,1,7,0,0
1541567400622,12,url1,200,OK,ThreadGroup 1-5,text,true,,843,0,1,1,12,0,0
1541567400637,8,url2,200,OK,ThreadGroup 1-5,text,true,,843,0,1,1,8,0,0
1541567400645,7,url1,200,OK,ThreadGroup 1-5,text,true,,843,0,1,1,7,0,0
1541567400653,6,url2,200,OK,ThreadGroup 1-5,text,true,,843,0,1,1,6,0,0
```

Columns are broken down on the following fields:

- timeStamp, elapsed, label, responseCode, responseMessage, threadName, dataType, success
- failureMessage bytes, sentBytes, grpThreads, allThreads, Latency, IdleTime, Connect

`elapsed` or `Latency` values are in milliseconds.

3.3.4 Uninstall from Kubernetes environment

Delete with Clover CLI

When you're finished working with JMeter validation services, you can uninstall it with the following command:

```
$ cloverctl delete system validation
```

The command above will remove the clover-jmeter-master and clover-jmeter-slave deployment and service resources from the current k8s context.

Delete from source

The JMeter validation services can be uninstalled from the source code using the commands below:

```
$ cd clover/samples/scenarios
$ kubectl delete -f clover-jmeter-master.yaml
$ kubectl delete -f clover-jmeter-slave.yaml
```

3.3.5 Uninstall from Docker environment

The OPNFV docker images can be removed with the following commands from nodes in the k8s cluster.

```
$ docker rmi opnfv/clover-jmeter-master
$ docker rmi opnfv/clover-jmeter-slave
$ docker rmi opnfv/clover-controller
```

3.4 Clover Visibility Services Configuration Guide

This document provides a guide to use Clover visibility services, which are initially delivered in the Clover Gambia release. A key assumption of this guide is that Istio 1.0.x has been deployed to Kubernetes (k8s), as it is a foundational element for Clover visibility services.

3.4.1 Overview

Clover visibility services are an integrated set of microservices that allow HTTP/gRPC traffic to be observed and analyzed in an Istio service mesh within k8s managed clusters. It leverages observability open source projects from the CNCF community such as Jaeger for distributed tracing and Prometheus for monitoring. These tools are packaged with Istio and service mesh sidecars have extensive hooks built in to interface with them. They gather low-level, per HTTP request driven data. Clover visibility services focus on enriching the data, gathering it from various sources and analyzing it at the system or aggregate level.

The visibility services are comprised of the following microservices all deployed within the **clover-system** namespace in a k8s cluster:

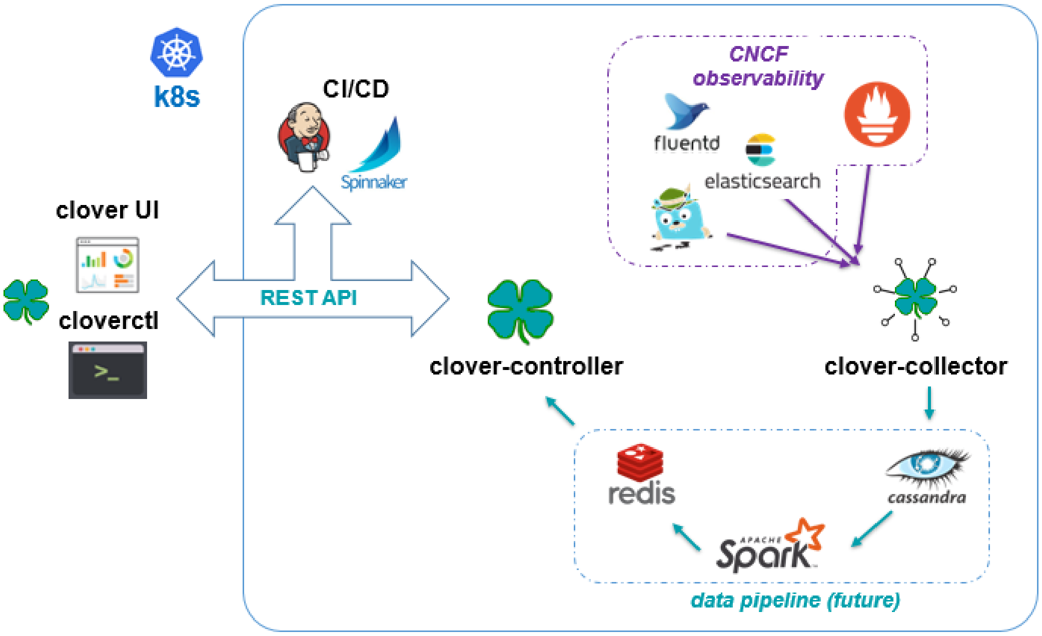
- **clover-controller** - exposes REST interface external to the k8s cluster and used to relay messages to other Clover services via gRPC from external agents including **cloverctl** CLI, web browsers and other APIs, scripts or CI jobs. It incorporates a web application with dashboard views to consume analyzed visibility data and control other Clover services.
- **clover-collector** - gathers data from tracing (Jaeger) and monitoring (Prometheus) infrastructure that is integrated with Istio using a pull model.
- **clover-spark** - is a Clover specific Apache Spark service. It leverages Spark 2.3.x native k8s support and includes visibility services artifacts to execute Spark jobs.
- **clover-spark-submit** - simple service to continually perform Spark job submits interacting with the k8s API to spawn driver and executor pods.
- **cassandra** - a sink for visibility data from **clover-collector** with specific schemas for monitoring and tracing.
- **redis** - holds configuration data and analyzed data for visibility services. Used by **clover-controller** web application and REST API to maintain state and exchange data.

The table below shows key details of the visibility service manifests outlined above:

Service	Kubernetes Deployment App Name	Docker Image	Ports
Controller	clover-controller	opnfv/clover-controller	HTTP: 80 (external) gRPC: 50052, 50054
Collector	clover-collector	opnfv/clover-collector	Jaeger: 16686 Prometheus: 9090 gRPC: 50054 Datastore: 6379, 9042
Spark	clover-spark clover-spark-submit	opnfv/clover-spark opnfv/clover-spark-submit	Datastore: 6379, 9042
Data Stores	cassandra redis	cassandra:3 k8s.gcr.io/redis:v1 kubernetes/redis:v1	9042 6379

The **redis** and **cassandra** data stores use community container images while the other services use Clover-specific Dockerhub OPNFV images.

Additionally, visibility services are operated with the **cloverctl** CLI. Further information on setting up **clover-controller** and **cloverctl** can be found at *Clover Controller Services Configuration Guide*.



The diagram above shows the flow of data through the visibility services where all blue arrows denote the path of data ingestion originating from the observability tools. The **clover-collector** reads data from these underlying tools using their REST query interfaces and inserts into schemas within the **cassandra** data store.

Apache Spark jobs are used to analyze data within **cassandra**. Spark is deployed using native Kubernetes support added since Spark version 2.3. The **clover-spark-submit** container continually submits jobs to the Kubernetes API. The API spawns a Spark driver pod which in turn spawns executor pods to run Clover-specific jobs packaged in the **clover-spark** service.

Analyzed data from **clover-spark** jobs is written to **redis**, an in-memory data store. The **clover-controller** provides a REST API for the analyzed visibility data to be read by other services (**cloverctl**, CI jobs, etc.) or viewed using a Clover provided visibility web dashboard.

3.4.2 Deploying the visibility engine

Prerequisites

The following assumptions must be met before continuing on to deployment:

- Installation of Docker has already been performed. It's preferable to install Docker CE.
- Installation of k8s in a single-node or multi-node cluster with at least twelve cores and 16GB of memory. Google Kubernetes Engine (GKE) clusters are supported.
- Installation of Istio in the k8s cluster. See *Deploy with Clover container*.
- Clover CLI (**cloverctl**) has been downloaded and setup. Instructions to deploy can be found at *Deploying clover-controller*.

Deploy with Clover CLI

To deploy the visibility services into your k8s cluster use the **cloverctl** CLI command shown below:

```
$ cloverctl create system visibility
```

Container images with the Gambia release tag will pulled if the tag is unspecified. The release tag is **opnfv-7.0.0** for the Gambia release. To deploy the latest containers from master, use the command shown below:

```
$ cloverctl create system visibility -t latest

Using config file: /home/earrage/.cloverctl.yaml
Creating visibility services
Created clover-system namespace
Created statefulset "cassandra".
Created service "cassandra"
Created pod "redis".
Created service "redis"
Created deployment "clover-collector".
Image: opnfv/clover-collector:latest
Created service "clover-collector"
Created deployment "clover-controller".
Image: opnfv/clover-controller:latest
Created service "clover-controller-internal"
Created serviceaccount "clover-spark".
Created clusterrolebinding "clover-spark-default".
Created clusterrolebinding "clover-spark".
Created deployment "clover-spark-submit".
Image: opnfv/clover-spark-submit:latest
```

Verifying the deployment

To verify the visibility services deployment, ensure the following pods have been deployed with the command below:

```
$ kubectl get pod --all-namespaces

NAMESPACE      NAME                                    READY     STATUS
clover-system   clover-collector-7dcc5d849f-6jc6m       1/1       Running
clover-system   clover-controller-74d8596bb5-qrr6b      1/1       Running
clover-system   cassandra-0                             1/1       Running
```

(continues on next page)

(continued from previous page)

```
clover-system   redis                                   2/2       Running
clover-system   clover-spark-submit-6c4d5bcdf8-kc6l9    1/1       Running
```

Additionally, spark driver and executor pods will continuously be deployed as displayed below:

```
clover-system   clover-spark-0fa43841362b3f27b35eaf6112965081-driver
clover-system   clover-spark-fast-d5135cdbdd8330f6b46431d9a7eb3c20-driver
clover-system   clover-spark-0fa43841362b3f27b35eaf6112965081-exec-3
clover-system   clover-spark-0fa43841362b3f27b35eaf6112965081-exec-4
```

3.4.3 Initializing visibility services

In order to setup visibility services, initialization and start commands must be invoked from the **cloverctl** CLI. There are sample yaml files in yaml directory from the **cloverctl** binary path. Navigate to this directory to execute the next sequence of commands.

Initialize the visibility schemas in cassandra with the following command:

```
$ cloverctl init visibility

Using config file: /home/earrage/.cloverctl.yaml
clover-controller address: http://10.145.71.21:32044
Added visibility schemas in cassandra
```

The initial configuration to the visibility services are the Jaeger tracing and Prometheus connection parameters and sample interval to **clover-collector**. To start visibility use the sample yaml provided and execute the command:

```
cloverctl start visibility -f start_visibility.yaml

Started collector on pid: 44
```

The `start_visibility.yaml` has defaults for the tracing and monitoring modules packaged with Istio 1.0.0.

3.4.4 Configure and control visibility

The core requirement for Clover visibility services to function, is for your services to be added to the Istio service mesh. Istio deployment and usage instructions are in the *Clover SDC Sample Configuration Guide* and the Service Delivery Controller (SDC) sample can be used to evaluate the Clover visibility services initially. A user may inject their own web-based services into the service mesh and track separately.

Connecting to visibility dashboard UI

The **clover-controller** service comes packaged with a web-based UI with a visibility view. To access the dashboard, navigate to the **clover-controller** address for either a `NodePort` or `LoadBalancer` service

- http://<node or CNI IP address>:<`NodePort` port>/
- http://<`LoadBalancer` IP address>/

See *Exposing clover-controller* to expose **clover-controller** externally with a k8s service.

Set runtime parameters using Clover CLI

The services visibility will track are based on the deployment/pod names specified in the k8s resources. Using some sample services from the SDC guide, the **proxy-access-control**, **clover-server1**, **clover-server2** and **clover-server3** services are specified in the `set_visibility.yaml` sample yaml referenced below.

To modify the configuration of the services visibility will track, use the **cloverctl CLI**, executing the following command:

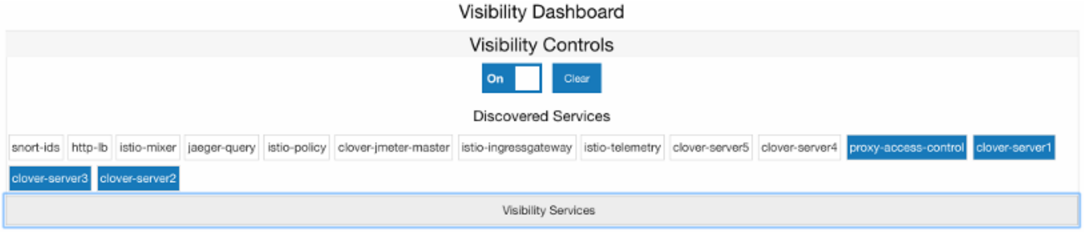
```
cloverctl set visibility -f set_visibility.yaml
```

Use the `services:` section of the yaml to configure service names to track.

```
# set_visibility.yaml
services:
    - name: proxy_access_control
    - name: clover_server1
    - name: clover_server2
    - name: clover_server3
metric_prefixes:
    - prefix: envoy_cluster_outbound_9180__
    - prefix: envoy_cluster_inbound_9180__
metric_suffixes:
    - suffix: _default_svc_cluster_local_upstream_rq_2xx
    - suffix: _default_svc_cluster_local_upstream_cx_active
custom_metrics:
    - metric: envoy_tracing_zipkin_spans_sent
```

Set runtime parameters using dashboard UI

The services being tracked by visibility can also be configured by selecting from the boxes under **Discovered Services** within the dashboard, as shown in the graphic below. Services can be multi-selected by using by holding the `Ctrl` or `command` (Mac OS) keyboard button down while selecting or unselecting. The SDC services that were configured from the **cloverctl** CLI above are currently active, denoted as the boxes with blue backgrounds.



In order for any services to be discovered from Jaeger tracing and displayed within the dashboard, some traffic must target the services of interest. Using curl/wget to send HTTP requests to your services will cause services to be discovered. Using Clover JMeter validation services, as detailed *JMeter Validation Configuration Guide* against SDC sample services will also generate a service listing. The **cloverctl** CLI commands below will generate traces through the SDC service chain with the JMeter master injected into the service mesh:

```
$ cloverctl create testplan -f yaml/jmeter_testplan.yaml # yaml located with␣
↪cloverctl binary
$ cloverctl start testplan
```

Clearing visibility data

To clear visibility data in cassandra and redis, which truncates **cassandra** tables and deletes or zeros out **redis** keys, use the following command:

```
$ cloverctl clear visibility
```

This can be useful when analyzing or observing an issue during a particular time horizon. The same function can be performed from the dashboard UI using the `Clear` button under `Visibility Controls`, as illustrated in the graphic from the previous section.

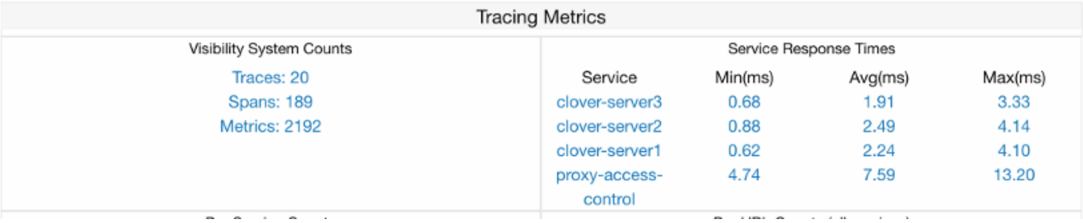
3.4.5 Viewing visibility data

The visibility dashboard can be used to view visibility data in real-time. The page will automatically refresh every 5 seconds. To disable continuous page refresh and freeze on a snapshot of the data, use the slider at the top of the page that defaults to `On`. Toggling it will result in it displaying `Off`.

The visibility dashboard displays various metrics and graphs of analyzed data described in subsequent sections.

System metrics

System metrics provide aggregate counts of cassandra tables including total traces, spans and metrics, as depicted on the left side of the graphic below.



Tracing Metrics

Visibility System Counts	Service Response Times			
	Service	Min(ms)	Avg(ms)	Max(ms)
Traces: 20	clover-server3	0.68	1.91	3.33
Spans: 189	clover-server2	0.88	2.49	4.14
Metrics: 2192	clover-server1	0.62	2.24	4.10
	proxy-access-control	4.74	7.59	13.20

The metrics counter will continually increase, as it is based on time series data from Prometheus. The trace count will correspond to the number of HTTP requests sent to services within the Istio service mesh. The span count ties to trace count, as it is a child object under Jaeger tracing data hierarchy and is based on the service graph (number of interactions between microservices for a given request). It will increase more rapidly when service graph depths are larger.

Per service response times

Per service response times are displayed on the right side of the graphic above and are calculated from tracing data when visibility is started. The minimum, maximum and average response times are output over the entire analysis period.

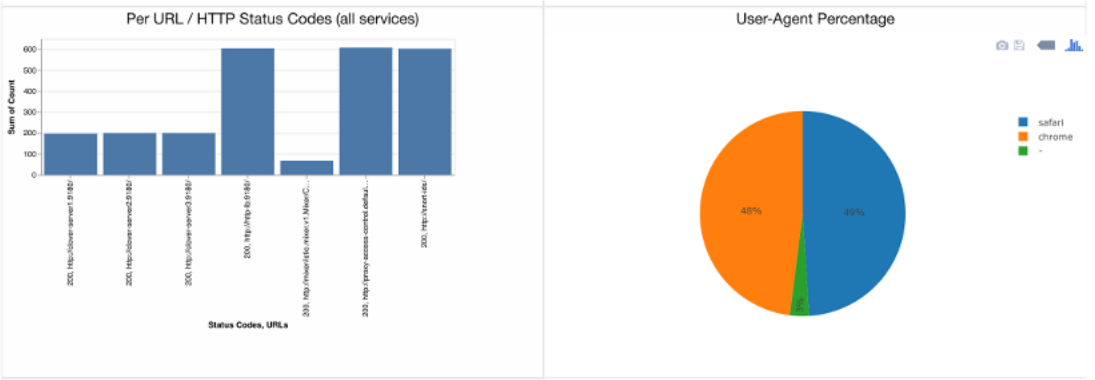
Group by span field counts

This category groups schema fields in various combinations to gain insight into the composition of HTTP data and can be used by CI scripts to perform various validations. Metrics include:

- Per service
- Distinct URL
- Distinct URL / HTTP status code

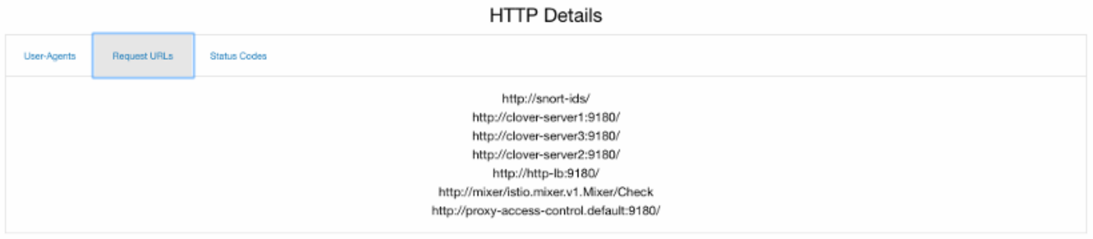
- Distinct user-agent (HTTP header)
- Per service / distinct URL

The dashboard displays bar/pie charts with counts and percentages, as depicted below. Each distinct key is displayed when hovering your mouse over a chart value.



Distinct HTTP details

A listing of distinct HTTP user-agents, request URLs and status codes is shown below divided with tabs.

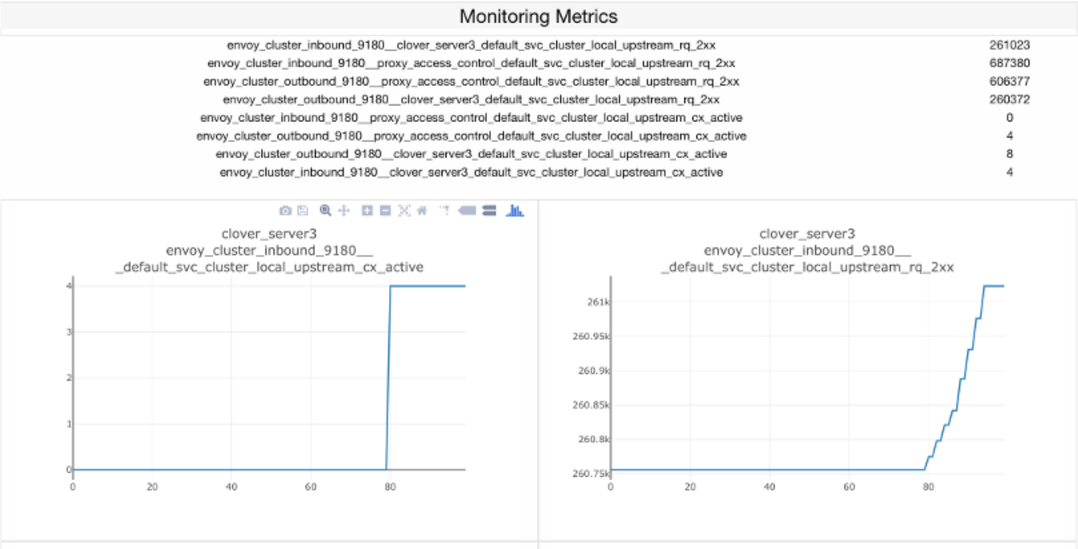


http://snort-ids/
http://clover-server1:9180/
http://clover-server3:9180/
http://clover-server2:9180/
http://http-lb:9180/
http://mixer/istio.mixer.v1.Mixer/Check
http://proxy-access-control.default:9180/

Monitoring Metrics

The Istio sidecars (Envoy) provide a lengthy set of metrics exposed through Prometheus. These metrics can be analyzed with the visibility service by setting up metrics, as outlined in section *Set runtime parameters using Clover CLI*. Use `metric_prefixes` and `metric_suffixes` sections of the set visibility yaml for many Envoy metrics that have a key with the service straddled by a prefix/suffix. A row in the table and a graph will be displayed for each combination of service, prefix and suffix.

The metrics are displayed in tabular and scatter plots over time formats from the dashboard, as shown in the graphic below:



Monitoring Metrics

envoy_cluster_inbound_9180__clover_server3_default_svc_cluster_local_upstream_rq_2xx	261023
envoy_cluster_inbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rq_2xx	687380
envoy_cluster_outbound_9180__proxy_access_control_default_svc_cluster_local_upstream_rq_2xx	606377
envoy_cluster_outbound_9180__clover_server3_default_svc_cluster_local_upstream_rq_2xx	260372
envoy_cluster_inbound_9180__proxy_access_control_default_svc_cluster_local_upstream_cx_active	0
envoy_cluster_outbound_9180__proxy_access_control_default_svc_cluster_local_upstream_cx_active	4
envoy_cluster_outbound_9180__clover_server3_default_svc_cluster_local_upstream_cx_active	8
envoy_cluster_inbound_9180__clover_server3_default_svc_cluster_local_upstream_cx_active	4

clover_server3
envoy_cluster_inbound_9180__
_default_svc_cluster_local_upstream_cx_active
4
3
2
1
0
0
20
40
60
80
clover_server3
envoy_cluster_inbound_9180__
_default_svc_cluster_local_upstream_rq_2xx
261k
260.95k
260.9k
260.85k
260.8k
260.75k
0
20
40
60
80

3.4.6 Uninstall from Kubernetes envionment

Delete with Clover CLI

When you're finished working with Clover visibility services, you can uninstall them with the following command:

```
$ cloverctl delete system visibility
```

The command above will remove the SDC sample services, Istio components and Jaeger/Prometheus tools from your Kubernetes environment.

3.4.7 Uninstall from Docker environment

The OPNFV docker images can be removed with the following commands:

```
$ docker rmi opnfv/clover-collector
$ docker rmi opnfv/clover-spark
$ docker rmi opnfv/clover-spark-submit
$ docker rmi opnfv/clover-controller
$ docker rmi k8s.gcr.io/redis
$ docker rmi kubernetes/redis
$ docker rmi cassandra:3
```

3.5 ModSecurity Configuration Guide

This document provides a guide to setup the ModSecurity web application firewall as a security enhancement for the Istio ingressgateway.

3.5.1 ModSecurity Overview

ModSecurity is an open source web application firewall. Essentially, ModSecurity is an Apache module that can be added to any compatible version of Apache. To detect threats, the ModSecurity engine is usually deployed embedded within the webserver or as a proxy server in front of a web application. This allows the engine to scan incoming and outgoing HTTP communications to the endpoint.

In Clover, we deploy ModSecurity on an Apache server and running it as a Kubernetes service that reside in “clover-gateway” namespace.

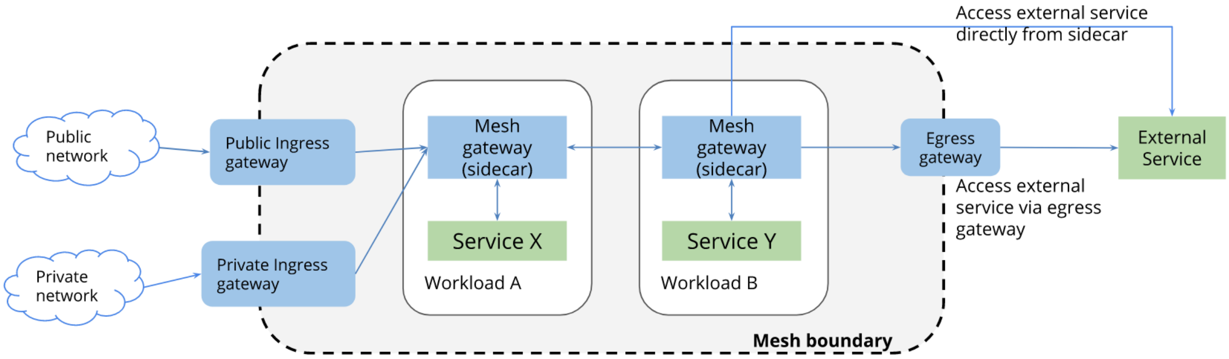
ModSecurity provides very little protection on its own. In order to become useful, ModSecurity must be configured with rules. Dependent on the rule configuration the engine will decide how communications should be handled which includes the capability to pass, drop, redirect, return a given status code, execute a user script, and more.

In Clover, we choose the OWASP ModSecurity Core Rule Set (CRS) for use with ModSecurity.

The OWASP ModSecurity Core Rule Set (CRS) is a set of generic attack detection rules. The CRS aims to protect web applications from a wide range of attacks, including the OWASP Top Ten, with a minimum of false alerts.

3.5.2 Ingress traffic security enhancement

In a typical Istio service mesh, ingressgateway terminates TLS from external networks and allows traffic into the mesh.



Clover enhances the security aspect of ingressgateway by redirecting all incoming HTTP requests through the ModSecurity WAF. To redirect HTTP traffic to the ModSecurity, Clover enables ext_authz (external authorization) Envoy filter on the ingressgateway.

For all incoming HTTP traffic, the ext_authz filter will authenticate each ingress request with the ModSecurity service. To perform authentication, an HTTP subrequest is sent from ingressgateway to ModSecurity where the subrequest is verified. If the subrequest is clean, ModSecurity will return a 2xx response code, access is allowed; If it returns 401 or 403, access is denied.

3.5.3 Deploying the ModSecurity WAF

Prerequisites

The following assumptions must be met before continuing on to deployment:

- Installation of Kubernetes has already been performed.
- Installation of Istio and Istio client (istioctl) is in your PATH.

Deploy from source

Clone the Clover git repository and navigate within the samples directory as shown below:

```
$ git clone https://gerrit.opnfv.org/gerrit/clover
$ cd clover/samples/scenarios
$ git checkout stable/gambia
```

To deploy the ModSecurity WAF in the “clover-gateway” Kubernetes namespace, use the following command:

```
$ kubectl create namespace clover-gateway
$ kubectl apply -n clover-gateway -f modsecurity_all_in_one.yaml
```

Verifying the deployment

To verify the ModSecurity pod is deployed, executing the command below:

```
$ kubectl get pod -n clover-gateway
```

The listing below must include the following ModSecurity pod:

```
$ NAME                                        READY     STATUS      RESTARTS   AGE
modsecurity-crs-cf5fffcc-whwqm                1/1       Running     0          1d
```

To verify the ModSecurity service is created, executing the command below:

```
$ kubectl get svc -n clover-gateway
```

The listing below must include the following ModSecurity service:

```
$ NAME                     TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)       ␣
 ↪  AGE
modsecurity-crs            NodePort       10.233.11.72    <none>        80:31346/TCP  ␣
 ↪  1d
```

To verify the ext-authz Envoy filter is created, executing the command below:

```
$ istioctl get envoyfilter -n clover-gateway
```

The listing below must include the following Envoy filter:

```
$ NAME        KIND                                       NAMESPACE      AGE
ext-authz   EnvoyFilter.networking.istio.io.v1alpha3     istio-system   1d
```

3.5.4 ModSecurity configuration

OWASP ModSecurity CRS mode

The OWASP ModSecurity CRS can run in two modes:

- **Anomaly Scoring Mode** - In this mode, each matching rule increases an

‘anomaly score’. At the conclusion of the inbound rules, and again at the conclusion of the outbound rules, the anomaly score is checked, and the blocking evaluation rules apply a disruptive action, by default returning an error 403.

- **Self-Contained Mode** - In this mode, rules apply an action instantly. Rules

inherit the disruptive action that you specify (i.e. deny, drop, etc). The first rule that matches will execute this action. In most cases this will cause evaluation to stop after the first rule has matched, similar to how many IDSs function.

By default, the CRS runs in Anomally scoring mode.

You can configurate CRS mode by editing the **crs-setup.conf** in the modsecurity-crs container:

```
$ kubectl exec -t -i -n clover-gateway [modsecurity-crs-pod-name] -c modsecurity-crs -
↪- bash
$ vi /etc/apache2/modsecurity.d/owasp-crs/crs-setup.conf
```

Alert logging

By default, CRS enables all detailed logging to the ModSecurity audit log. You can check the audit log using the command below:

```
$ kubectl exec -t -i -n clover-gateway [modsecurity-crs-pod-name] -c modsecurity-crs -
↪- cat /var/log/modsec_audit.log
```

CRS Rules

By default, Clover enables all OWASP CRS rules. Below is a short description of all enabled rules:

- **REQUEST-905-COMMON-EXCEPTIONS**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-905-COMMON-EXCEPTIONS.conf

Some rules are quite prone to causing false positives in well established software, such as Apache callbacks or Google Analytics tracking cookie. This file offers rules that will allow the transactions to avoid triggering these false positives.

- **REQUEST-910-IP-REPUTATION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-910-IP-REPUTATION.conf

These rules deal with detecting traffic from IPs that have previously been involved with malicious activity, either on our local site or globally.

- **REQUEST-912-DOS-PROTECTION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-912-DOS-PROTECTION.conf

The rules in this file will attempt to detect some level 7 DoS (Denial of Service) attacks against your server.

- **REQUEST-913-SCANNER-DETECTION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-913-SCANNER-DETECTION.conf

These rules are concentrated around detecting security tools and scanners.

- **REQUEST-920-PROTOCOL-ENFORCEMENT**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf

The rules in this file center around detecting requests that either violate HTTP or represent a request that no modern browser would generate, for instance missing a user-agent.

- **REQUEST-921-PROTOCOL-ATTACK**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-921-PROTOCOL-ATTACK.conf

The rules in this file focus on specific attacks against the HTTP protocol itself such as HTTP Request Smuggling and Response Splitting.

- **REQUEST-930-APPLICATION-ATTACK-LFI**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-930-APPLICATION-ATTACK-LFI.conf

These rules attempt to detect when a user is trying to include a file that would be local to the webserver that they should not have access to. Exploiting this type of attack can lead to the web application or server being compromised.

- **REQUEST-931-APPLICATION-ATTACK-RFI**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-931-APPLICATION-ATTACK-RFI.conf

These rules attempt to detect when a user is trying to include a remote resource into the web application that will be executed. Exploiting this type of attack can lead to the web application or server being compromised.

- **REQUEST-941-APPLICATION-ATTACK-SQLI**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-941-APPLICATION-ATTACK-SQLI.conf

Within this configuration file we provide rules that protect against SQL injection attacks. SQL attackers occur when an attacker passes crafted control characters to parameters to an area of the application that is expecting only data. The application will then pass the control characters to the database. This will end up changing the meaning of the expected SQL query.

- **REQUEST-943-APPLICATION-ATTACK-SESSION-FIXATION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-943-APPLICATION-ATTACK-SESSION-FIXATION.conf

These rules focus around providing protection against Session Fixation attacks.

- **REQUEST-949-BLOCKING-EVALUATION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf

These rules provide the anomaly based blocking for a given request. If you are in anomaly detection mode this file must not be deleted.

- **RESPONSE-954-DATA-LEAKAGES-IIS**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-954-DATA-LEAKAGES-IIS.conf

These rules provide protection against data leakages that may occur because of Microsoft IIS

- **RESPONSE-952-DATA-LEAKAGES-JAVA**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-952-DATA-LEAKAGES-JAVA.conf

These rules provide protection against data leakages that may occur because of Java

- **RESPONSE-953-DATA-LEAKAGES-PHP**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-953-DATA-LEAKAGES-PHP.conf

These rules provide protection against data leakages that may occur because of PHP

- **RESPONSE-950-DATA-LEAKAGES**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-950-DATA-LEAKAGES.conf

These rules provide protection against data leakages that may occur genericly

- **RESPONSE-951-DATA-LEAKAGES-SQL**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-951-DATA-LEAKAGES-SQL.conf

These rules provide protection against data leakages that may occur from backend SQL servers. Often these are indicative of SQL injection issues being present.

- **RESPONSE-959-BLOCKING-EVALUATION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-959-BLOCKING-EVALUATION.conf

These rules provide the anomaly based blocking for a given response. If you are in anomaly detection mode this file must not be deleted.

- **RESPONSE-980-CORRELATION**

Configuration Path: /etc/apache2/modsecurity.d/owasp-crs/rules/RESPONSE-980-CORRELATION.conf

The rules in this configuration file facilitate the gathering of data about successful and unsuccessful attacks on the server.

3.6 Spinnaker Configuration Guide

This document provides a guide to setup the spinnaker in kubernetes as a continuous delivery platform.

3.6.1 Spinnaker Overview

Spinnaker is an open-source, multi-cloud continuous delivery platform that helps you release software changes with high velocity and confidence.

Spinnaker provides two core sets of features:

1. application management

You use Spinnaker's application management features to view and manage your cloud resources.

2. application deployment

You use Spinnaker's application deployment features to construct and manage continuous delivery workflows.

For more information on Spinnaker and its capabilities, please refer to documentation.

3.6.2 Setup Spinnaker

Prerequisites

The following assumptions must be met before continuing on to deployment:

- Ubuntu 16.04 was used heavily for development and is advised for greenfield deployments.
- Installation of Docker has already been performed. It's preferable to install Docker CE.
- Installation of Kubernetes has already been performed.
- A PersistentVolume resource need to be setup in k8s for the PersistentVolumeClaim to use. we supply the manifest file minio-pv.yml to create the PV, But it is not suitable for use in production.

Deploy from source

Clone the Clover git repository and navigate within the samples directory as shown below:

```
$ git clone https://gerrit.opnfv.org/gerrit/clover
$ cd clover/clover/spinnaker/install
$ git checkout stable/gambia
```

To deploy the Spinnaker in the "spinnaker" Kubernetes namespace, use the following command:

```
$ kubectl create -f quick-install-spinnaker.yml
```

Verifying the deployment

To verify the Spinnaker pods is deployed, executing the command below:

```
$ kubectl get pod -n spinnaker
```

The listing below must include the following Spinnaker pods:

```
$ NAME                                READY     STATUS    RESTARTS   AGE
minio-deployment-5d84f45dd5-zjdzb   1/1       Running   0          22h
spin-clouddriver-795575c5cb-ph8qc   1/1       Running   0          22h
spin-deck-7c5d75bfcd-vr58q          1/1       Running   0          22h
spin-echo-7986796c94-4285v          1/1       Running   0          22h
spin-front50-5744674fdc-d9xsw       1/1       Running   0          22h
spin-gate-7978d55d57-jcsmq          1/1       Running   0          22h
spin-halyard                        1/1       Running   0          22h
spin-igor-6f8c86bbbb-cs8gd          1/1       Running   0          22h
spin-orca-8659c57c5c-rs69z          1/1       Running   0          22h
spin-redis-558db8d5bd-kdmjz         1/1       Running   0          22h
spin-rosco-dfbbcbccd-db65b          1/1       Running   0          22h
```

To verify the Spinnaker services is created, executing the command below:

```
$ kubectl get svc -n spinnaker
```

The listing below must include the following Spinnaker services:

```
$ NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
minio-service       ClusterIP   10.233.21.175   <none>        9000/TCP   22h
spin-clouddriver    ClusterIP   10.233.9.27     <none>        7002/TCP   22h
spin-deck           ClusterIP   10.233.34.86    <none>        9000/TCP   22h
spin-echo           ClusterIP   10.233.29.150   <none>        8089/TCP   22h
spin-front50        ClusterIP   10.233.5.221    <none>        8080/TCP   22h
spin-gate           ClusterIP   10.233.33.196   <none>        8084/TCP   22h
spin-halyard        ClusterIP   10.233.2.187    <none>        8064/TCP   22h
spin-igor           ClusterIP   10.233.29.93    <none>        8088/TCP   22h
spin-orca           ClusterIP   10.233.23.140   <none>        8083/TCP   22h
spin-redis          ClusterIP   10.233.20.95    <none>        6379/TCP   22h
spin-rosco          ClusterIP   10.233.48.79    <none>        8087/TCP   22h
```

To publish the spin-deck service, we need change the type to NodePort, executing the command below:

```
$ kubectl get svc spin-deck -n spinnaker -o yaml |sed 's/ClusterIP/NodePort/'␣
↪|kubectl replace -f -
$ kubectl get svc -n spinnaker
```

The listing below must include the following services

```
$ NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
minio-service       ClusterIP   10.233.21.175   <none>        9000/TCP         22h
spin-clouddriver    ClusterIP   10.233.9.27     <none>        7002/TCP         22h
spin-deck           NodePort    10.233.34.86    <none>        9000:31747/TCP   22h
```

(continues on next page)

(continued from previous page)

```
spin-echo          ClusterIP   10.233.29.150   <none>        8089/TCP          22h
spin-front50       ClusterIP   10.233.5.221    <none>        8080/TCP          22h
spin-gate          ClusterIP   10.233.33.196   <none>        8084/TCP          22h
spin-halyard       ClusterIP   10.233.2.187    <none>        8064/TCP          22h
spin-igor          ClusterIP   10.233.29.93    <none>        8088/TCP          22h
spin-orca          ClusterIP   10.233.23.140   <none>        8083/TCP          22h
spin-redis         ClusterIP   10.233.20.95    <none>        6379/TCP          22h
spin-rosco         ClusterIP   10.233.48.79    <none>        8087/TCP          22h
```

In your browser, navigate to the following URLs for Spinnaker respectively:

```
http://<node IP>:31747
```

Where node IP is an IP from one of the Kubernetes cluster node(s).

release/configguide/imgs/spinnaker.png

3.6.3 Spinnaker Configuration

When the default installation is ready, there are many different components that you can turn on with Spinnaker. In order to customize Spinnaker, you can use the halyard command line or clover command line to edit the configuration and apply it to what has already been deployed.

Halyard Command

Halyard has an in-cluster daemon that stores your configuration. You can exec a shell in this pod to make and apply your changes.

For example:

```
$ kubectl exec spin-halyard -n spinnaker -it -- bash -il
spinnaker@spin-halyard:/workdir$ hal version list
```

How to use the halyard command line to configurate the spinnaker, please refer to commands documentation.

Clover Command

Clover provider the cloverctl and clover-controller to controll the server. So we can use the cloverctl to configurate the spinnaker. So far, clover provide the capabilities to create/get/delete docker-registry and kubernetes provider in spinnaker.

NOTE: Before using clover command, you need build the clover command and setup the clover-controller in your local kubernetes cluster, where spinnaker deploy in.

Docker Registry

You need a configuration file written in YAML that describe the information about you Docker Registry as shown below:

docker.yml:

```
name: mydockerhub
address: https://index.docker.io
username: if-you-images-aren't-publicly-available
password: fill-this-field
repositories:
- opnfv/clover
```

If any of your images aren't publicly available, you need fill your DockerHub username & password. Ortherwise you can delete the username & password field.

Creating the Docker Registry in spinnaker:

```
$ cloverctl create provider docker-registry -f docker.yml
```

Getting the Docker Registry in spinnaker:

```
$ cloverctl get provider docker-registry
```

Deleting the Docker Registry in spinnaker:

```
$ cloverctl delete provider docker-registry -n dockerhub
```

Kubernetes

By default, installing the manifest only registers the local cluster as a deploy target for Spinnaker. If you want to add arbitrary clusters you can use the cloverctl command

You need a running Kubernetes cluster, with corresponding credentials in a kubeconfig file(/path/to/kubeconfig). And You also need a configuration file written in YAML that describe the information about your kubernetes cluseter as shown below:

kubernetes.yml:

```
# name must match pattern ^[a-z0-9]+([-a-z0-9]*[a-z0-9])?$'
name: my-kubernetes
providerVersion: V1
# make sure the kubeconfigFile can be use
kubeconfigFile: /path/to/kubeconfig
dockerRegistries:
- accountName: dockerhub
```

Creating the kubernetes provider in spinnaker:

```
$ cloverctl create provider kubernetes -f kubernetes.yml
```

Getting the kubernetes provider in spinnaker:

```
$ cloverctl get provider kubernetes
```

Deleting the kubernetes provider in spinnaker:

```
$ cloverctl delete provider kubernetes -n my-kubernetes
```

3.6.4 Deploy Helm Charts

Currently, spinnaker support to deploy applications with the helm chart. More information please refer to Deploy Helm Charts.

Upload helm charts to artifacts

Before doing this, please package the helm chart first. how to package the chart, refer to helm documentation.

```
$ wget https://dl.minio.io/client/mc/release/linux-amd64/mc
$ chmod +x mc
$ ./mc config host add my_minio  http://{minio-service-ip}:9000 dont-use-this for-production S3v4
$ ./mc mb my_minio/s3-account
$ ./mc cp test-0.1.0.tgz my_minio/s3-account/test-0.1.0.tgz
```

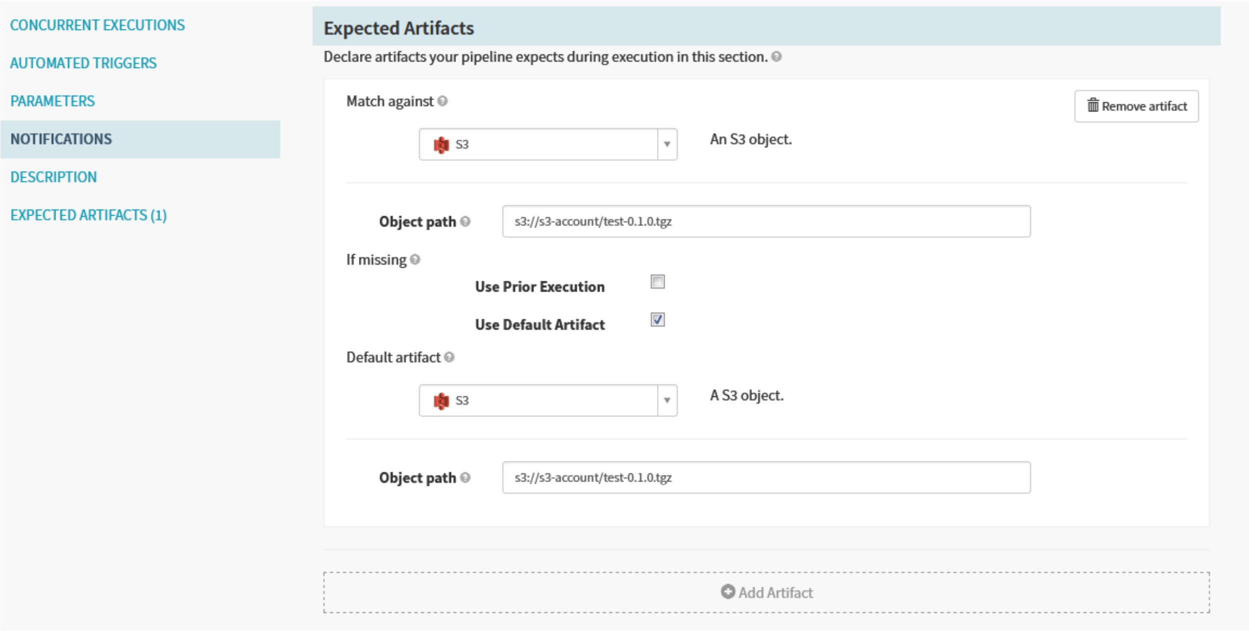
NOTE: the minio-service-ip is 10.233.21.175 in this example

Configure Pipeline

This pipeline include three stages,configuration, bake and deploy.

Configuration stage

We can configure Automated triggers and expected artifacts in this stage. We just declare expected artifacts and trigger the pipeline manually.



Expected Artifacts

Declare artifacts your pipeline expects during execution in this section.

Match against: S3 — An S3 object.

Object path: s3://s3-account/test-0.1.0.tgz

If missing

Use Prior Execution □

Use Default Artifact ☑

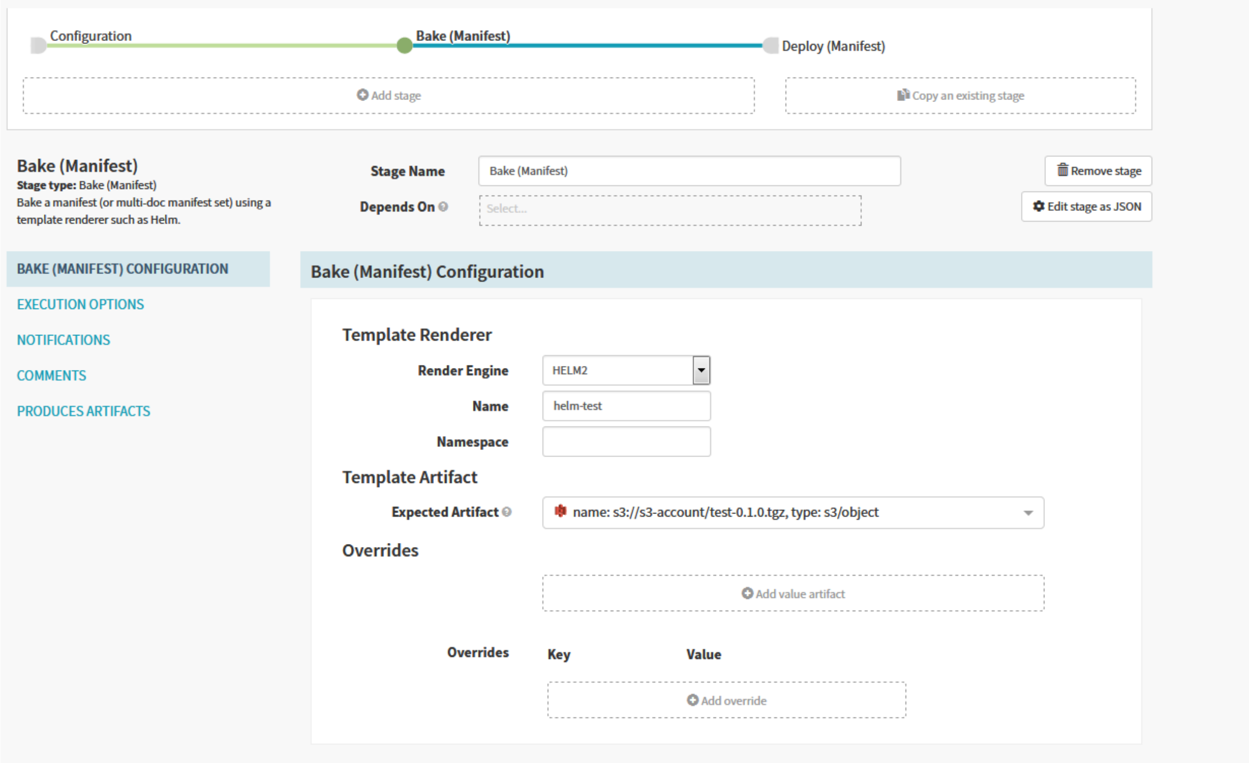
Default artifact: S3 — A S3 object.

Object path: s3://s3-account/test-0.1.0.tgz

NOTE: We need to enable “Use Default Artifact”, when we need trigger the pipeline manually

Bake Manifest stage

For example, we have a test “Bake(Manifest)” stage below



Bake (Manifest)
Stage type: Bake (Manifest)
Bake a manifest (or multi-doc manifest set) using a template renderer such as Helm.

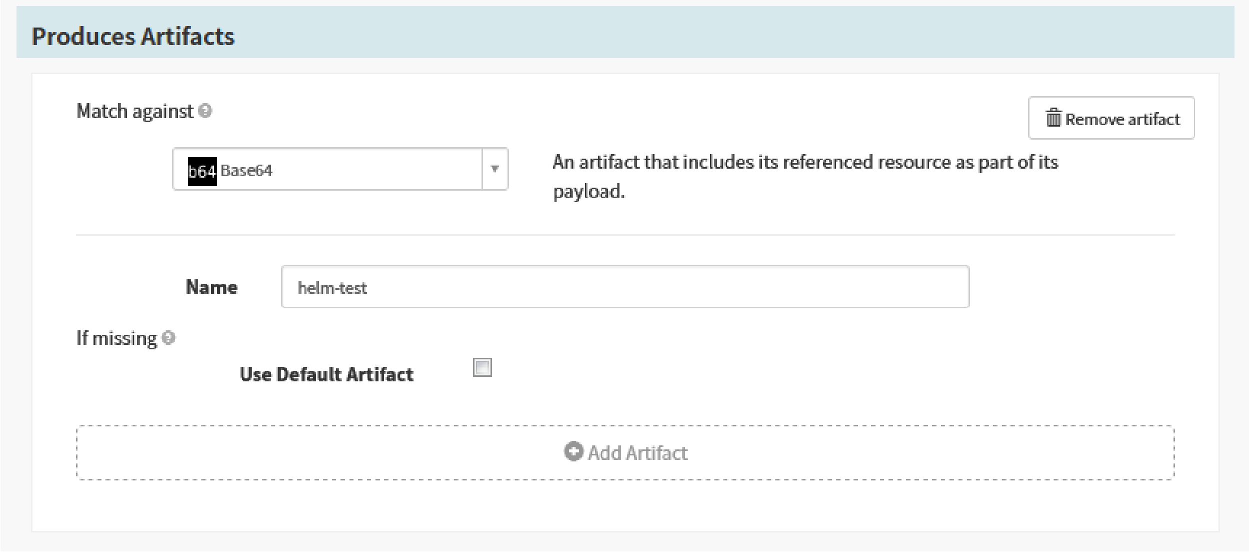
Stage Name: Bake (Manifest)
Depends On: Select...

Template Renderer
Render Engine: HELM2
Name: helm-test
Namespace:

Template Artifact
Expected Artifact: name: s3://s3-account/test-0.1.0.tgz, type: s3/object

Overrides
Overrides: Key Value

Spinnaker has automatically created an embedded/base64 artifact that is bound when the stage completes, representing the fully baked manifest set to be deployed downstream.



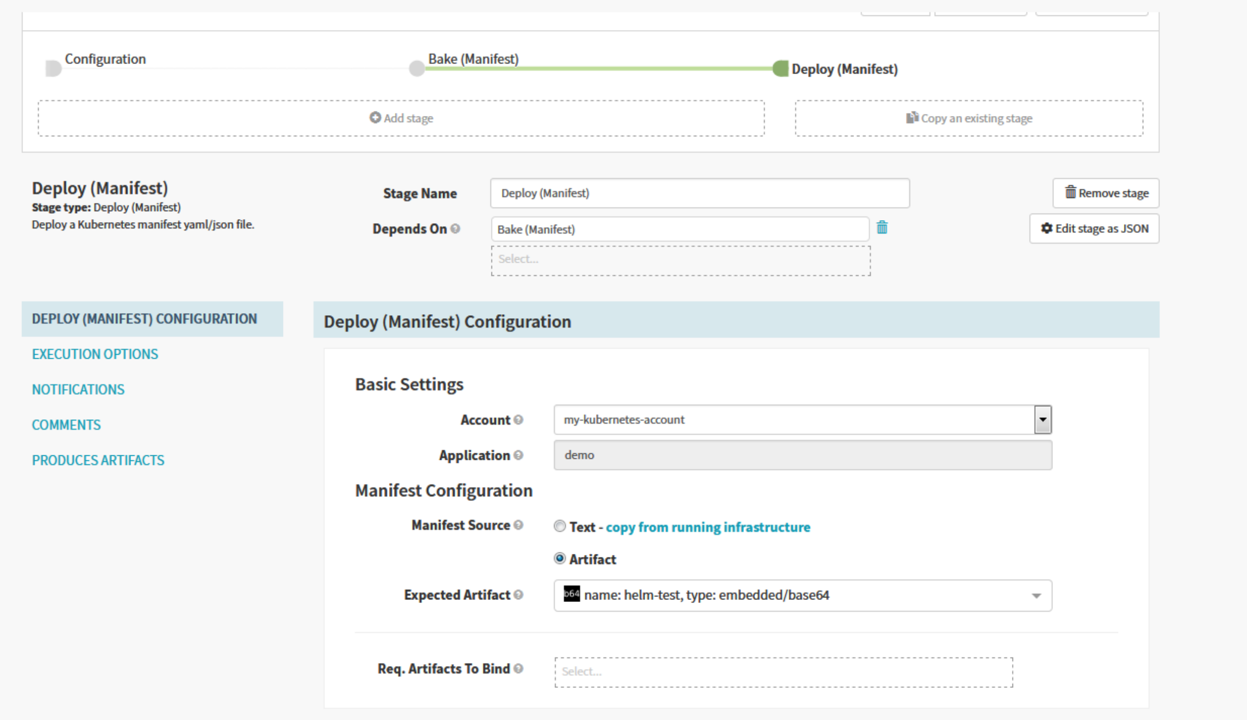
Match against: b64 Base64
An artifact that includes its referenced resource as part of its payload.

Name: helm-test

If missing
Use Default Artifact ☐

Deploy Manifest stage

After the chart was baked by helm, we can configure a "Deploy(Manifest)" stage to deploy the manifest produced by previous stage as shown below.



Deploy (Manifest)
Stage type: Deploy (Manifest)
Deploy a Kubernetes manifest yaml/json file.

Stage Name: Deploy (Manifest)
Depends On: Bake (Manifest)

Deploy (Manifest) Configuration

Basic Settings

Account: my-kubernetes-account
Application: demo

Manifest Configuration

Manifest Source: ○ Text - copy from running infrastructure ● Artifact
Expected Artifact: name: helm-test, type: embedded/base64

Req. Artifacts To Bind: Select...

Once this pipeline runs completely, you can see every resource in your Helm chart get deployed.

3.7 Clovisor Configuration Guide

Clovisor requires minimal to no configurations to function as a network tracer. It expects configurations to be set at a redis sever running at clover-system namespace.

3.7.1 No Configuration

If redis server isn't running as service name **redis** in namespace **clover-system** or there isn't any configuration related to Clovisor in that redis service, then Clovisor would monitor all pods under the **default** namespace. The traces would be sent to **jaeger-collector** service under the **clover-system** namespace

3.7.2 Using redis-cli

Install `redis-cli` on the client machine, and look up redis IP address:

```
$ kubectl get services -n clover-system
```

which one may get something like the following:

```
$
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
redis     ClusterIP   10.109.151.40   <none>        6379/TCP   16s
```

if (like above), the external IP isn't visible, one may be able to get the pod IP address directly via the pod (for example, it works with Flannel as CNI plugin):

```
$ kubectl get pods -n clover-system -o=wide
NAME      READY     STATUS    RESTARTS   AGE       IP             NODE
redis     2/2       Running   0          34m       10.244.0.187   clover1804
```

and one can connect to redis via:

```
redis-cli -h 10.244.0.187 -p 6379
```

3.7.3 Jaeger Collector Configuration

Clovisor allows user to specify the Jaeger service for which Clovisor would send the network traces to. This is configured via setting the values for keys **clovisor_jaeger_collector** and **clovisor_jaeger_agent**:

```
redis> SET clovisor_jaeger_collector "jaeger-collector.istio-system:14268"
"OK"
redis> SET clovisor_jaeger_agent "jaeger-agent.istio-system:6831"
"OK"
```

3.7.4 Configure Monitoring Namespace and Labels

Configruation Value String Format:

> <namespace>[:label-key:label-value]

User can configure namespace(s) for Clovisor to tap into via adding namespace configuration in redis list **clovisor_labels**:

```
redis> LPUSH clovisor_labels "my-namespace"
(integer) 1
```

the above command will cause Clovisor to **NOT** monitor the pods in **default** namespace, and only monitor the pods under **my-namespace**.

If user wants to monitor both 'default' and 'my-namespace', she needs to explicitly add 'default' namespace back to the list:

```
redis> LPUSH clovisor_labels "default"
(integer) 2
redis> LRANGE clovisor_labels 0 -1
1.) "default"
2.) "my-namespace"
```

Clovisor allows user to optionally specify which label match on pods to further filter the pods to monitor:

```
redis> LPUSH clovisor_labels "my-2nd-ns:app:database"
(integer) 1
```

the above configuration would result in Clovisor only monitoring pods in my-2nd-ns namespace which matches the label "app:database"

User can specify multiple labels to filter via adding more configuration entries:

```
redis> LPUSH clovisor_labels "my-2nd-ns:app:web"
(integer) 2
redis> LRANGE clovisor_labels 0 -1
1.) "my-2nd-ns:app:web"
2.) "my-2nd-ns:app:database"
```

the result is that Clovisor would monitor pods under namespace my-2nd-ns which match **EITHER** app:database **OR** app:web

Currently Clovisor does **NOT** support filtering of more than one label per filter, i.e., no configuration option to specify a case where a pod in a namespace needs to be matched with TWO or more labels to be monitored

3.7.5 Configure Egress Match IP address, Port Number, and Matching Pods

Configuration Value String Format:

> <IP Address>:<TCP Port Number>[:<Pod Name Prefix>]

By default, Clovisor only traces packets that goes to a pod via its service port, and the response packets, i.e., from pod back to client. User can configure tracing packet going **OUT** of the pod to the next microservice, or an external service also via the **clovior_egress_match** list:

```
redis> LPUSH clovior_egress_match "10.0.0.1:3456"
(integer) 1
```

the command above will cause Clovisor to trace packet going out of ALL pods under monitoring to match IP address 10.0.0.1 and destination TCP port 3456 on the **EGRESS** side — that is, packets going out of the pod.

User can also choose to ignore the outbound IP address, and only specify the port to trace via setting IP address to zero:

```
redis> LPUSH clovior_egress_match "0:3456"
(integer) 1
```

the command above will cause Clovisor to trace packets going out of all the pods under monitoring that match destination TCP port 3456.

User can further specify a specific pod prefix for such egress rule to be applied:

```
redis> LPUSH clovior_egress_match "0:3456:proxy"
(integer) 1
```

the command above will cause Clovisor to trace packets going out of pods under monitoring which have name starting with the string "proxy" that match destination TCP port 3456

CHAPTER 4

OPNFV Clover Design Specification

4.1 Clovisor

4.1.1 What is Clovisor?

One of Clover's goals is to investigate an optimal way to perform network tracing in cloud native environment. Clovisor is project Clover's initial attempt to provide such solution.

Clovisor is named due to it being "Clover's use of IOVisor". IOVisor is a set of tools to ease eBPF code development for tracing, monitoring, and other networking functions. BPF stands for Berkeley Packet Filter, an in-kernel virtual machine like construct which allows developers to inject bytecodes in various kernel event points. More information regarding BPF can be found here. Clovisor utilizes the goBPF module from IOVisor as part of its control plane, and primarily uses BPF code to perform packet filtering in the data plane.

4.1.2 Clovisor Functionality

Clovisor is primarily a session based network tracing module, that is, it generates network traces on a per-session basis, i.e., on a request and response pair basis. It records information pertaining to L3/L4 and L7 (just HTTP 1.0 and 1.1 for now) regarding the session. The traces are sent to Jaeger server who acts as tracer, or trace collector.

4.1.3 Clovisor Requirement

Clovisor is tested on kernel versions 4.14.x and 4.15.x. For Ubuntu servers built-in kernel, it requires Ubuntu version 18.04.

4.1.4 Clovisor Workflow

Clovisor runs as a DaemonSet — that is, it runs on every nodes in a Kubernetes cluster, including being automatically launched in newly joined node. Clovior runs in the "clovisor" Kubernetes namespace, and it needs to run in privilege mode and be granted at least pod and service readable right for the Kubernetes namespace(s) in which it is monitoring, i.e., a RBAC needs to be set up to grant such access right to the clovisor namespace service account.

Clovisor looks for its configuration(s) from redis server in clover-system namespace. The three config info for Clovisor for now are:

1. clovisor_labels, a list of labels which Clovisor would filter for monitoring
2. clovisor_egress_match, a list of interested egress side IP/port for outbound traffic monitoring
3. clovisor_jaeger_server, specifying the Jaeger server name / port to send traces to

By default Clovisor would monitor all the pods under the 'default' namespace. It will read the service port name associated with the pod under monitoring, and use the service port name to determine the network protocol to trace. Clovisor expects the same service port naming convention / nomenclature as Istio, which is specified in istio. Clovisor extracts expected network protocol from these names; some examples are

```
apiVersion: v1
kind: Service
[snip]
spec:
  ports:
  - port: 1234
    name: http
```

With the above example in the service specification, Clovisor would specifically look to trace HTTP packets for packets matching that destination port number on the pods associated with this service, and filter everything else. The following has the exact same bahavior

```
apiVersion: v1
kind: Service
[snip]
spec:
  ports:
  - port: 1234
    name: http-1234
```

Clovisor derived what TCP port to monitor via the container port exposed by the pod in pod spec. In the following example:

```
spec:
  containers:
  - name: foo
    image: localhost:5000/foo
    ports:
    - containerPort: 3456
```

Packets with destination TCP port number 3456 will be traced for the pod on the ingress side, likewise for packet with source TCP port number 3456 on the ingress side (for receiving response traffic tracing). This request-response pair is sent as a span.

In addition, Clovisor provides egress match configurion where user can configure the (optional) IP address of the egress side traffic and TCP port number for EGRESS or outbound side packet tracing. This is particularly useful for the use case where the pod sends traffic to an external entity (for example, sending to an external web site on port 80). User can further specify which pod prefix should the rules be applied.

Clovisor is a session-based network tracer, therefore it would trace both the request and response packet flow, and extract any information necessary (the entire packet from IP header up is copied to user space). In Gambia release Clovisor control plane extracts source/destination IP addresses (from request packet flow perspective), source/destination

TCP port number, and HTTP request method/URL/protocol as well as response status/status code/protocol, and overall session duration. These information is being logged via OpenTracing APIs to Jaeger.

4.1.5 Clovisor Control Plane

There are two main elements of Clovisor control plane: Kubernetes client and BPF control plane using IOVisor BCC.

Kubernetes client is used for the following needs:

1. fetches the pods pertaining to filter ('default' namespace by default without filter)
2. fetches corresponding service port name to determine network protocol to trace (TCP by default)
3. extracts veth interface index for pod network interface
4. watches for pod status change, or if new pod got launched that matches the filter

Clovisor uses goBPF from IOVisor BCC project to build its control plane for BPF datapath, which does:

1. via netlink, under the pod veth interface on the Linux host side, creates a QDisc with name 'classact' with ingress and egress filters created under it
2. dynamically compiles and loads BPF code "session_tracing.c" and sets ingress and egress functions on the filters created above
3. sets up perfMap (shared packet buffer between user space and kernel) and sets up kernel channel to poll map write event
4. sets up timer task to periodically logs and traces interested packets

4.1.6 Clovisor Data Plane

Clovisor utilizes BPF for data plane packet analysis in kernel. BPF bytecode runs in kernel and is executed as an event handler. Clovisor's BPF program has an ingress and egress packet handling functions as loadable modules for respective event trigger points, i.e., ingress and egress on a particular Linux network interface, which for Clovisor is the pod associated veth. There are three tables used by the Clovisor BPF program:

1. dports2proto: control plane -> data plane: the container/service port and corresponding protocol (TCP, HTTP…etc) to trace on the ingress side
2. egress_lookup_table: control plane -> data plane: the list of egress IP address / ports which Clovisor should trace on the egress side
3. sessions: data plane -> control plane: BPF creates entries to this table to record TCP sessions

4.1.7 Clovisor Clean Up

As mentioned above, on a per pod basis, Clovisor creates a qdisc called 'classact' per each pod veth interface. This kernel object does not get deleted by simply killing the Clovisor pod. The cleanup is done via Clovisor either via pod removal, or when the Clovisor pod is deleted. However, IF the qdisc is not cleaned up, Clovisor would not be able to tap into that same pod, more specifically, that pod veth interface. The qdisc can be examined via the following command:

```
sudo tc qdisc show
```

and you should see something like this:

```
qdisc clsact ffff: dev veth4c47cc75 parent ffff:fff1
```

in case it wasn't removed at the end, user can manually remove it via:

```
sudo tc qdisc del dev veth4c47cc75 clsact
```

(of course, the qdisc should be removed by Clovisor, otherwise it is a Clovisor bug)

4.2 Logging

4.2.1 Installation

Currently, we use the sample configuration in Istio to install fluentd:

```
cd clover/logging
```

First, install logging stack Elasticsearch, Fluentd and Kibana:

```
kubectl apply -f install/logging-stack.yaml
```

Note that, it must be done in separated steps. If you run `kubectl apply -f install` instead, the mixer adapter may fail to intialize because the target service can not be found. You may find an error message from mixer container:

```
2018-05-09T02:43:14.435156Z error   Unable to initialize adapter:
snapshot='6', handler='handler.fluentd.istio-system', adapter='fluentd',
err='adapter instantiation error: dial tcp: lookup fluentd-es.logging on
10.96.0.10:53: no such host'.
```

Then configure fluentd for istio:

```
kubectl apply -f install/fluentd-istio.yaml
```

Configure fluentd for node level logging:

```
kubectl apply -f install/fluentd-daemonset-elasticsearch-rbac.yaml
```

4.2.2 Validate

The scripts in `clover/logging` validates fluentd installation:

```
python clover/logging/validate.py
```

It validates the installation with the following criterias

1. existence of fluented pod
2. fluentd input is configured correctly
3. TBD

4.2.3 Understanding how it works

In clover stack, Istio is configured to automatically gather logs for services in a mesh. More specificly, it is configured in Mixer:

```
- when to log
- what to log
- where to log
```

When to log

Istio defines when to log by creating a custom resource `rule`. For example:

```
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: newlogtofluentd
  namespace: istio-system
spec:
  match: "true" # match for all requests
  actions:
  - handler: handler.fluentd
    instances:
    - newlog.logentry
```

This rule specifies that all instances of `newlog.logentry` that matches the expression will be handled by the specified handler `handler.fluentd`. We shall explain `instances` and `handler` later. The expression `true` means whenever a request arrive at Mixer, it will trigger the actions defined belows.

`rule` is a custom resource definition from Istio installation.

```
# Rule to send logentry instances to the fluentd handler
kind: CustomResourceDefinition
apiVersion: apiextensions.k8s.io/v1beta1
metadata:
  name: rules.config.istio.io
  labels:
    package: istio.io.mixer
    istio: core
spec:
  group: config.istio.io
  names:
    kind: rule
    plural: rules
    singular: rule
scope: Namespaced
version: v1alpha2
```

What to log

The instance defines what content to be logged.

> A (request) instance is the result of applying request attributes to the > template mapping. The mapping is specified as an instance configuration.

For example:

```
# Configuration for logentry instances
apiVersion: "config.istio.io/v1alpha2"
kind: logentry
metadata:
  name: newlog
  namespace: istio-system
spec:
  severity: '"info"'
  timestamp: request.time
  variables:
    source: source.labels["app"] | source.service | "unknown"
    user: source.user | "unknown"
    destination: destination.labels["app"] | destination.service | "unknown"
    responseCode: response.code | 0
    responseSize: response.size | 0
    latency: response.duration | "0ms"
  monitored_resource_type: '"UNSPECIFIED"'
```

The keys under `spec` should conform to the template. To learn what fields are available and valid type, you may need to reference the corresponding template, in this case, Log Entry template.

The values of each field could be either Istio attributes or an expression.

> A given Istio deployment has a fixed vocabulary of attributes that it > understands. The specific vocabulary is determined by the set of attribute > producers being used in the deployment. The primary attribute producer in > Istio is Envoy, although Mixer and services can also introduce attributes.

Refer to the Attribute Vocabulary to learn the full set.

By the way, `logentry` is also a custom resource definition created by Istio.

Where to log

For log, the handler defines where these information will be handled, in this example, a fluentd daemon on fluentd-es.logging:24224.

```
# Configuration for a fluentd handler
apiVersion: "config.istio.io/v1alpha2"
kind: fluentd
metadata:
  name: handler
  namespace: istio-system
spec:
  address: "fluentd-es.logging:24224"
```

In this example, handlers (`handler.fluentd`) configure Adapters (`fluentd`) to handle the data delivered from the created instances (`newlog.logentry`).

An adapter only accepts instance of specified kind. For example, fluentd adapter accepts logentry but not other kinds.

4.3 Monitoring

4.3.1 Installation

Currently, we use the Istio build-in prometheus addon to install prometheus:

```
cd <istio-release-path>
kubectl apply -f install/kubernetes/addons/prometheus.yaml
```

4.3.2 Validate

Setup port-forwarding for prometheus by executing the following command:

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

Run the scripts in `clover/monitoring` validates prometheus installation:

```
python clover/monitoring/validate.py
```

It validates the installation with the following criterias

1. [DONE] prometheus pod is in Running state
2. [DONE] prometheus is conneted to monitoring targets
3. [TODO] test collecting telemetry data from istio
4. [TODO] TBD

4.4 Tracing

4.4.1 Installation

Currently, we use the Jaeger tracing all-in-one Kubernetes template for development and testing, which uses in-memory storage. It can be deployed to the istio-system namespace with the following command:

```
kubectl apply -n istio-system -f https://raw.githubusercontent.com/jaegertracing/
↪jaeger-kubernetes/master/all-in-one/jaeger-all-in-one-template.yml
```

The standard Jaeger REST port is at 16686. To make this service available outside of the Kubernetes cluster via any node IP in the cluster, use the following command:

```
kubectl expose -n istio-system deployment jaeger-deployment --port=16686 --
↪type=NodePort
```

Kubernetes will expose the Jaeger service on another port from 30000-32767 and the assignment can be found with:

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

An example listing from the command above is shown below where the Jaeger service is exposed externally on port 30888 in this case:

```
istio-system   jaeger-deployment   NodePort  10.104.113.94  <none> 16686:30888/TCP
```

Jaeger will be accessible using the host IP of any node in Kubernetes cluster and port provided. With this method, the Jaeger UI will also be available from a remote host. If external access is required to Jaeger but restricted to cluster localhost(s), an alternate method is to use the **port-forward** command in the foreground, as shown below:

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

4.4.2 Validate

The script in `clover/tracing` validates Jaeger installation:

```
python clover/tracing/validate.py
```

It validates the installation with the following criteria:

1. Existence of Jaeger all-in-one deployment using Kubernetes
2. Jaeger service is accessible using IP address and port configured in installation steps
3. Optionally, if Jaeger can retrieve service listing for default Istio components (istio-ingress, istio-mixer). At least one HTTP request must be sent to istio-ingress after initial Jaeger deployment for this validation to function.