Red Hat | TIGERA

Protect OpenShift containers with Tigera Calico

Lab Guide for Red Hat Product Demo System (RHPDS)

Table of Contents

Workshop Objectives 3

Introduction to Calico Cloud and Calico Enterprise 3

Join the Slack Channel (optional) 4

Part 1 - Getting familiar with your OpenShift Cluster 4

Part 2 - Joining your cluster to Calico Cloud

Part 3 - Configure demo applications 10

Part 4 - Try out some use cases 12

Identity-Aware Microsegmentation: App service control 12

Identity-Aware Microsegmentation: Pod Microsegmentation 16

Zero Trust Workload Access Controls: DNS Egress Controls 19

Runtime Threat Defense: Global Threadfeed 22

Observability: Manager UI 24

Observability: Kibana Dashboard 28

Observability: eBPF based Application-level Observability 30

Observability: Dynamic packet capture 32

Part 5 - Clean up your test environment 35

Sunnary 38

Learn More 38

Workshop Objectives

This workshop demonstrates how to protect your containers on Red Hat OpenShift Container Platform (OCP) with Tigera Calico. You will learn to mitigate security risks from build-time, deploy-time, and runtime threats.

Introduction to Calico Cloud and Calico Enterprise

Calico Cloud is the industry's only active security SaaS offering for cloud-native applications running on containers, Kubernetes, and the cloud. It enables organizations to prevent attacks using zero trust and detect, troubleshoot, and automatically remediate exposure risks from security issues in build, deploy, and runtime stages across multi-cloud and hybrid deployments. Calico Cloud is built on Calico Open Source, the most widely adopted container networking, and security solution.

Calico Enterprise is the industry's only active Cloud-Native Application Protection Platform (CNAPP) with full-stack observability for containers and Kubernetes. Calico Enterprise extends the declarative nature of Kubernetes to specify security and observability as code. This ensures consistent enforcement of security policies and compliance and provides observability for troubleshooting across multi-cluster, multi-cloud and hybrid deployments.

Join the Slack Channel (optional)

Calico User Group Slack is a great resource to ask any questions about Calico. If you are not a part of this Slack group yet, we highly recommend joining it to participate in discussions or ask questions.
[OCP User Group Slack]

Part 1 - Getting familiar with your OpenShift Cluster

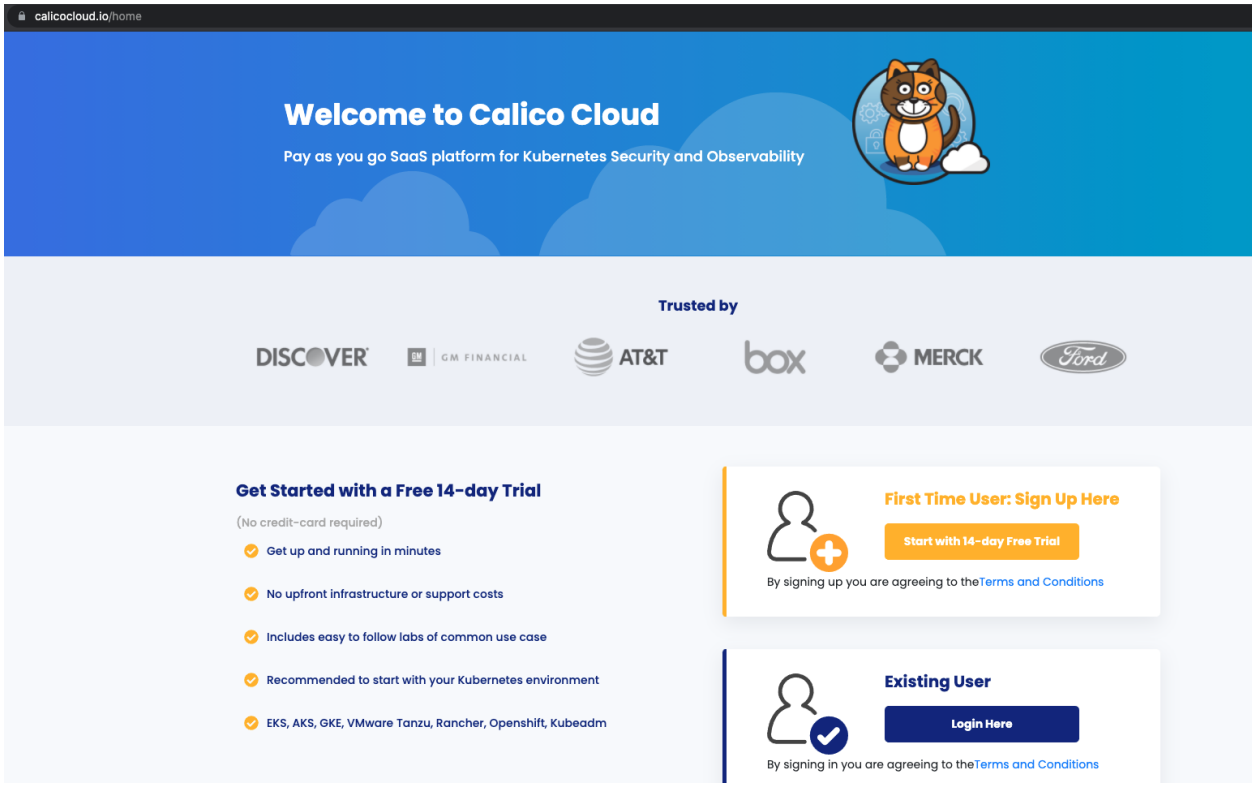
Emulate CyberArk Intro

Part 2 - Joining your cluster to Calico Cloud

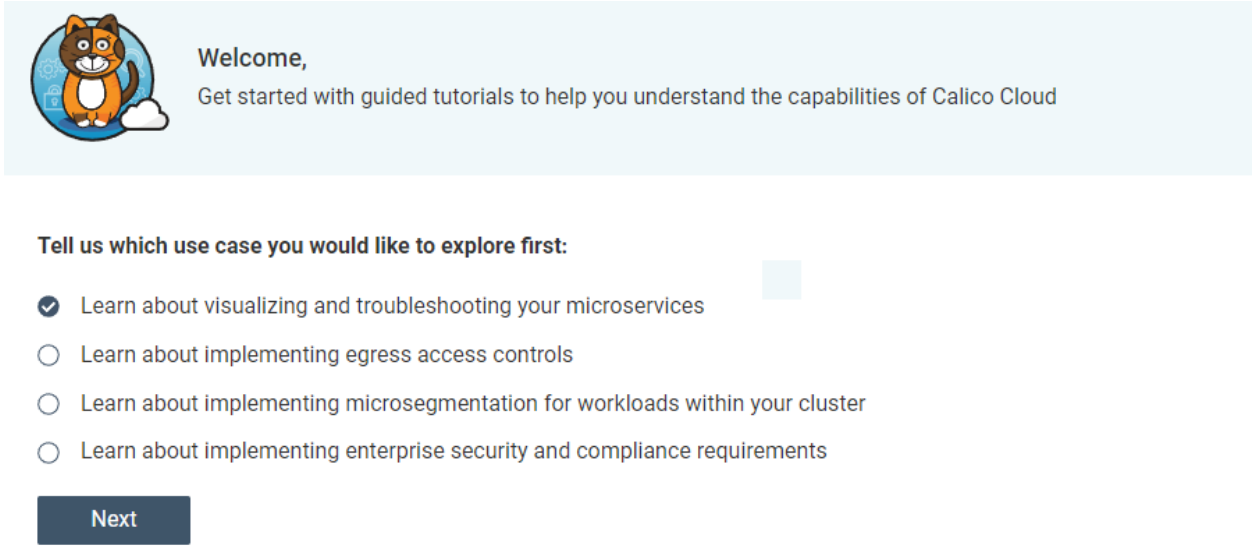
If you are unable to navigate menus in the UI please ensure you disable any script blockers in your browser.

Steps:

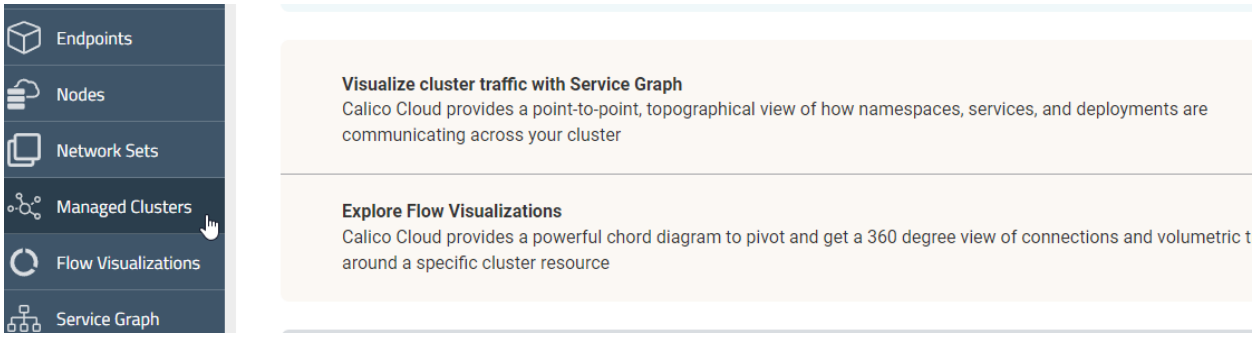
1. Go to https://www.calicocloud.io/home/ and click on Start Free Trial to start a free 14-day trial.- no credit cards required. Returning users can login, new users can create an account. To extend the period beyond 14 days, please send a request to partners@tigera.io before the end of the initial trial.



2. Upon signing into the Calico Cloud UI, close any pop-up windows that may appear. You may also see a short Survey. Pick one of the options and click Next to continue.

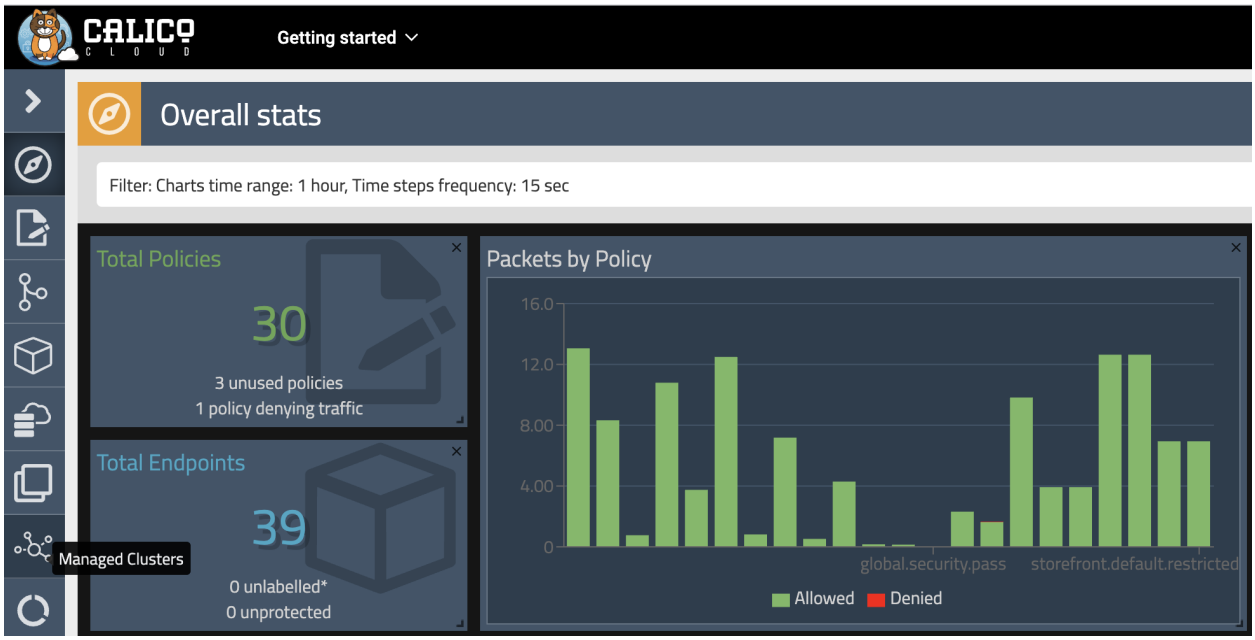


3. Expand the menu on the left to display the worded menu as shown:

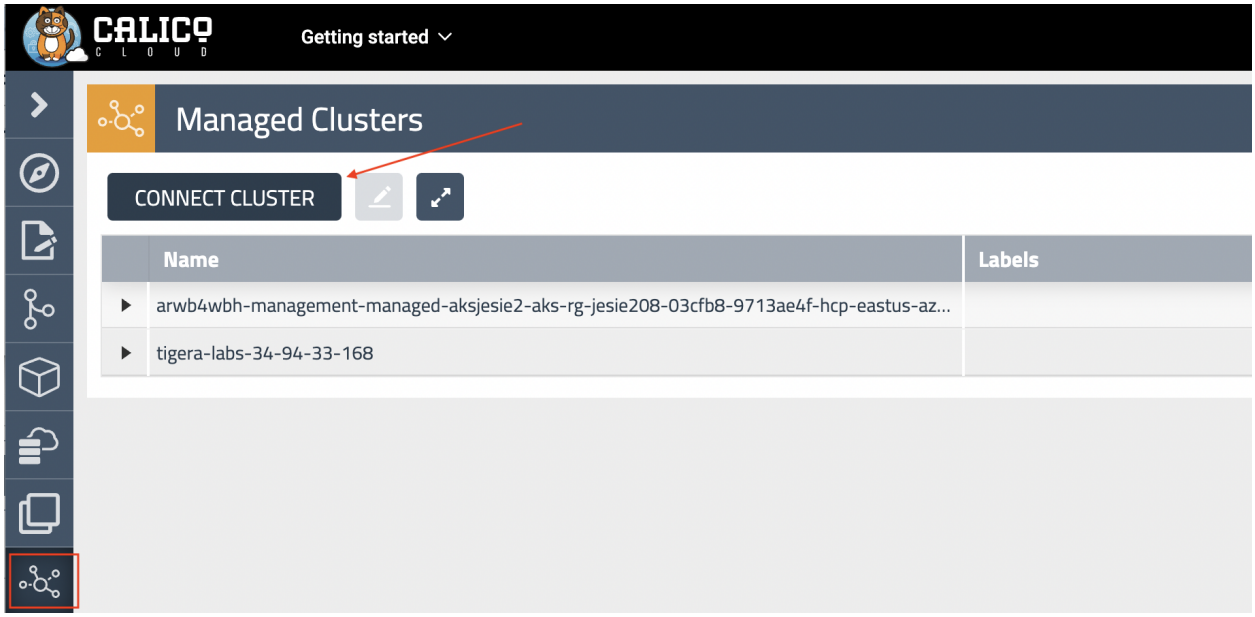


4. Join your cluster to the Calico Cloud management plane.

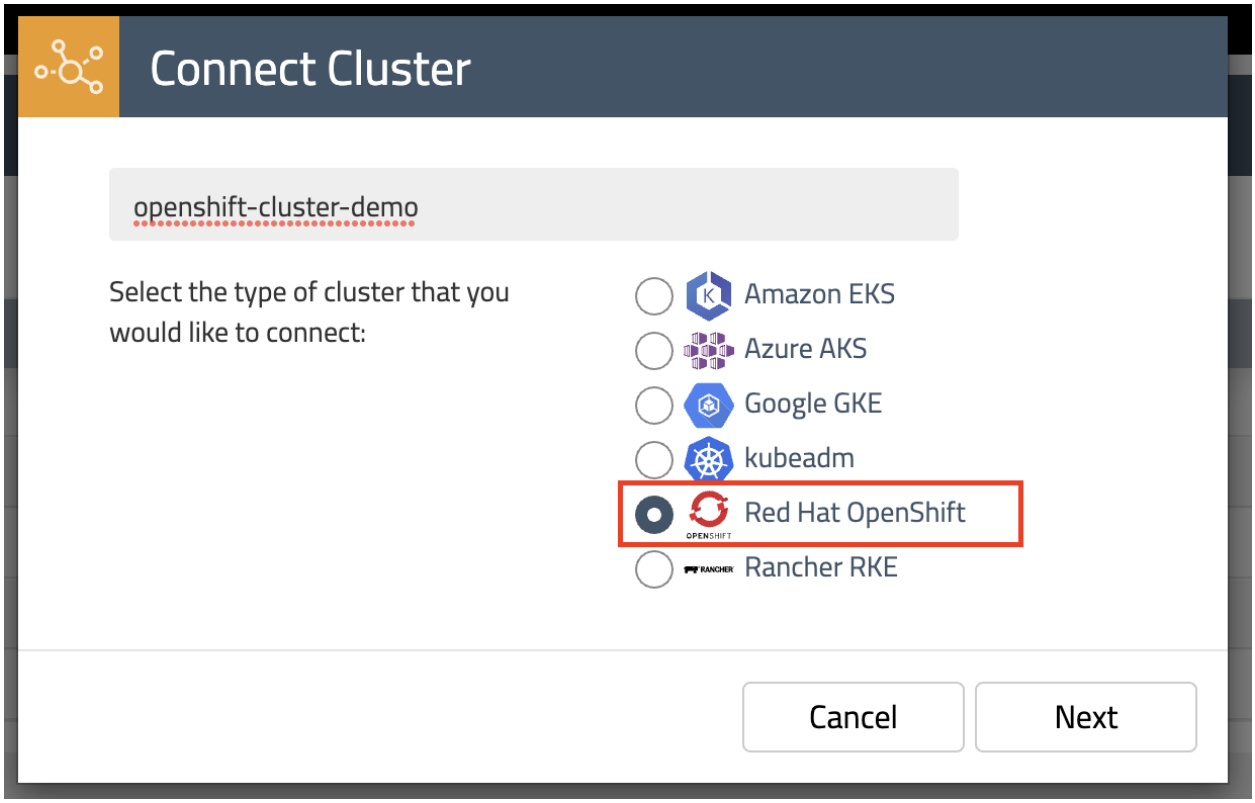
Click the "Managed Cluster" on the left side of the browser.



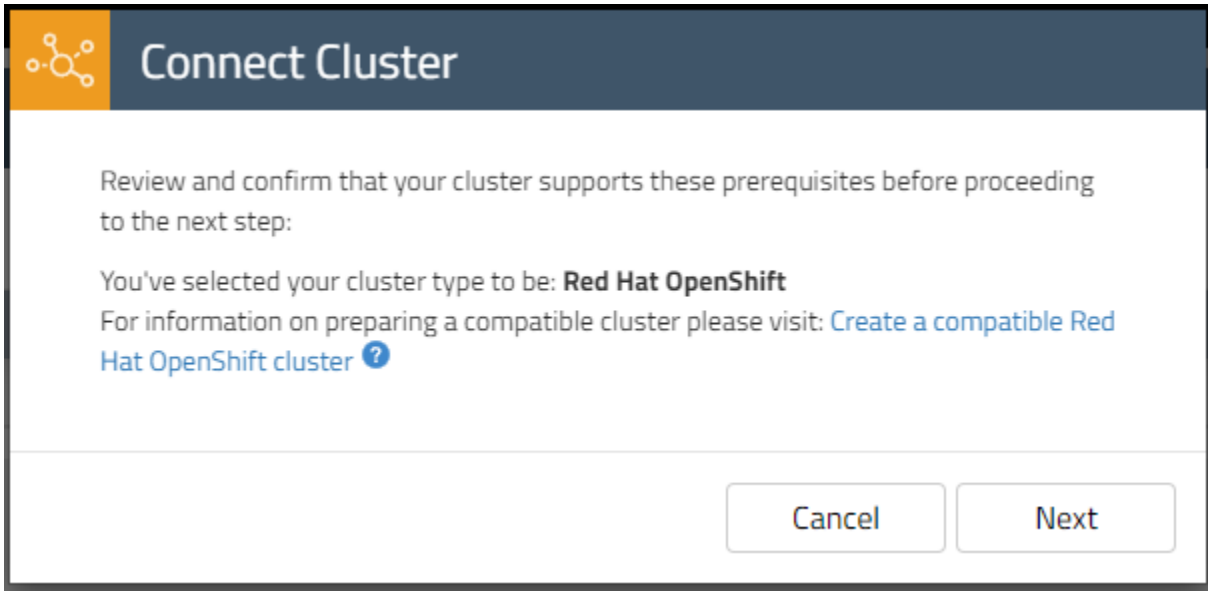
Click on "connect cluster".



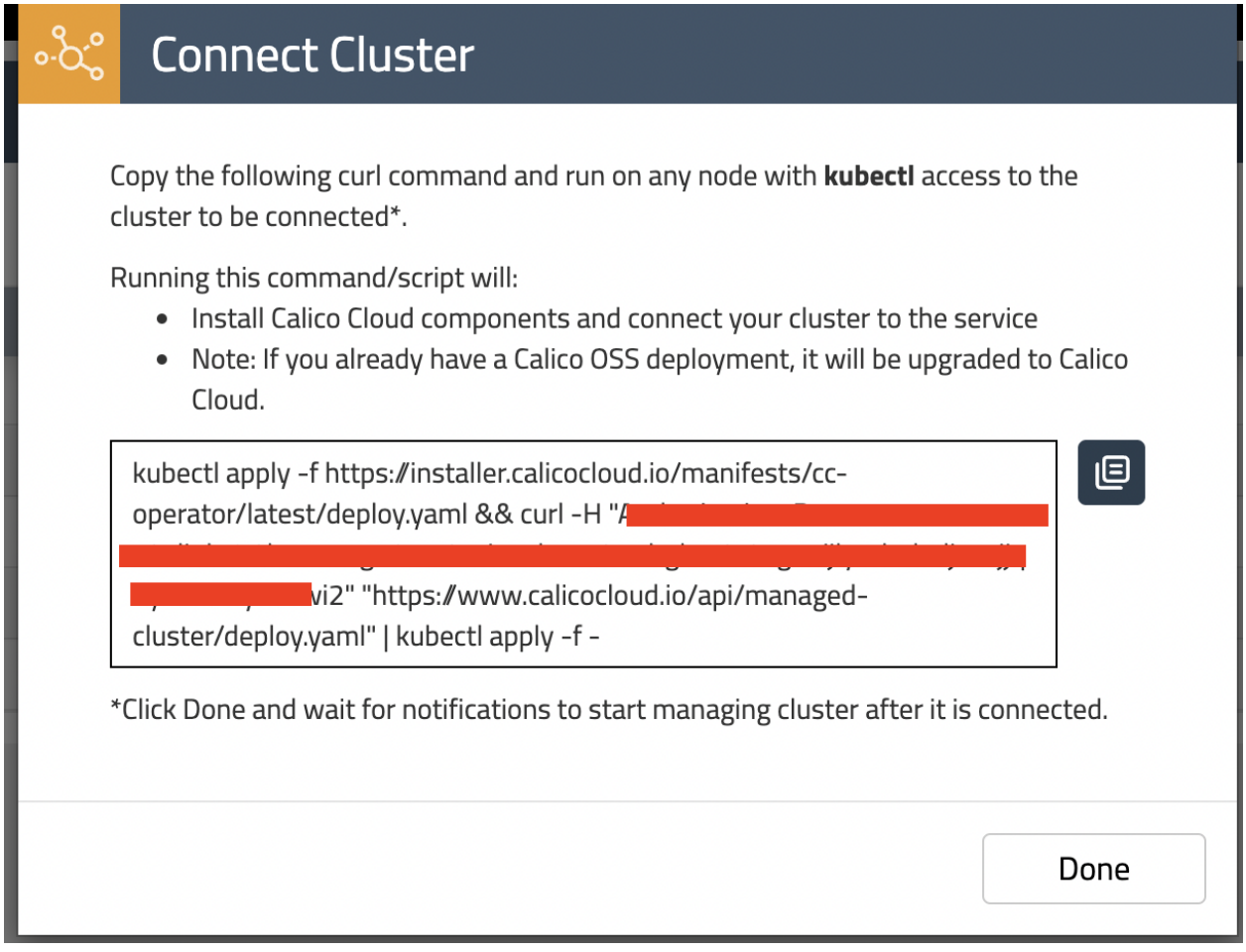
Choose a managed cluster name then choose Red hat OpenShift.



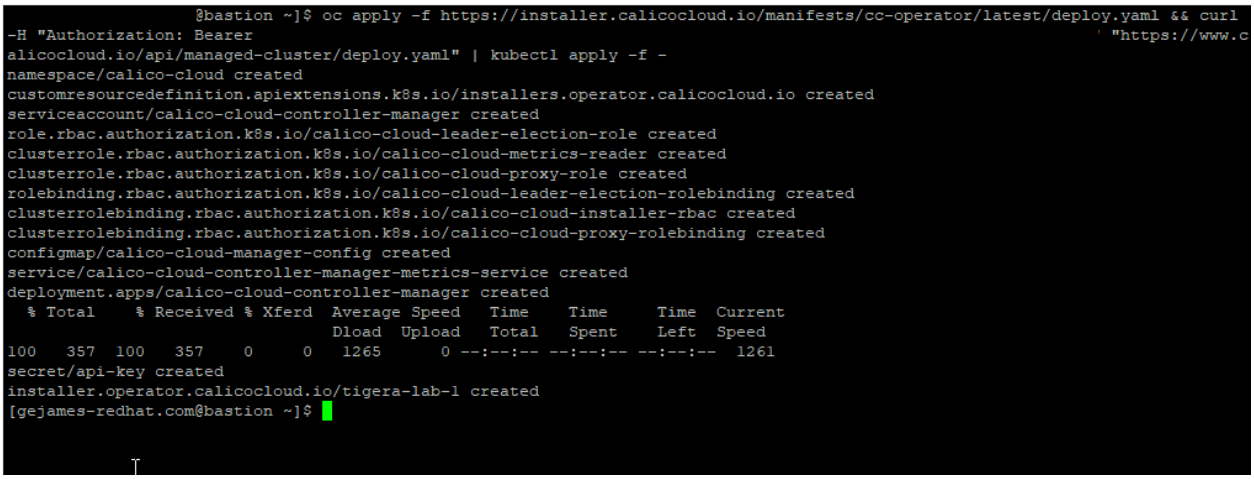
Click Next.



You will then be presented with a code snippet to run on your cluster. Replace the *kubectl* command with the *oc* command.



You should have received instructions on how to access the cluster bastion from the lab instructor or in the lab deployment email. Using a terminal program, connect to the bastion and run the command from the previous step.



Joining the cluster to Calico Cloud can take a few minutes. Wait for the installation script to finish before proceeding to the next step.

In the Calico Cloud management UI, you can see your cluster added in "Managed Cluster".

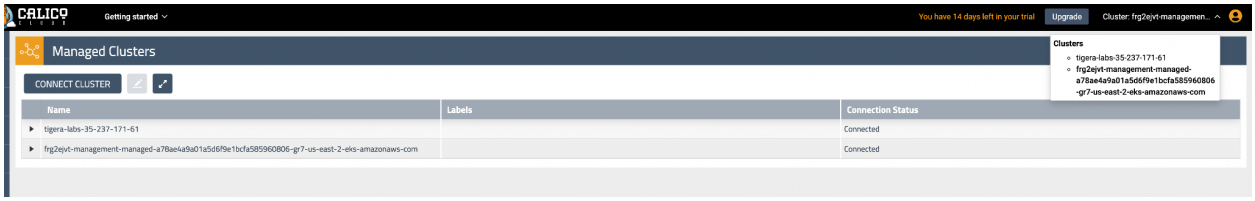
You can also confirm by running the following command.

```
oc get tigerastatus
```

```
#make sure all customer resources are "AVAILABLE=True"
NAME                              AVAILABLE   PROGRESSING   DEGRADED   SINCE
apiserver                         True        False         False      5m38s
calico                            True        False         False      4m44s
compliance                        True        False         False      4m34s
intrusion-detection               True        False         False      4m49s
log-collector                     True        False         False      4m19s
management-cluster-connection     True        False         False      4m54s
```

5. Navigating the Calico Cloud UI.

   Once the cluster has successfully connected to Calico Cloud, you can review the cluster status in the UI. Click on Managed Clusters from the left side menu and look for the connected status of your cluster. You will also see a Tigera-labs cluster for demo purposes. Ensure you are in the correct cluster context by clicking the Cluster dropdown in the top right corner. This will list the connected clusters. Click on your cluster to switch context. The current cluster context is in *bold* font.



6. To configure log aggregation and flush intervals, we will use 15s instead of the default 300s.

```
oc patch felixconfiguration.p default -p
'{"spec":{"flowLogsFlushInterval":"15s"}}'
oc patch felixconfiguration.p default -p
'{"spec":{"dnsLogsFlushInterval":"15s"}}'
oc patch felixconfiguration.p default -p
'{"spec":{"flowLogsFileAggregationKindForAllowed":1}}'
```

7. Configure Felix to collect TCP stats - this uses eBPF TC program and requires minimum Kernel version of v5.3.0. Further documentation.

```
oc patch felixconfiguration default -p
'{"spec":{"flowLogsCollectTcpStats":true}}'
```

Part 3 - Configure demo applications

1. Download this repo into your environment:

```
git clone
https://github.com/tigera-solutions/calicocloud-openshift-workshop.git
cd calicocloud-openshift-workshop
```

2. Deploy policy tiers.

To take advantage of hierarchical policy management, deploy policies into the policy tier.

```
oc apply -f demo/setup/tiers/
```

This will add tiers security and platform to the Calico cluster.

3. Deploy base policy.

We will implement a network policy that explicitly allows workloads to connect to the Kubernetes DNS component. We will also deploy an *allow* policy for logging and constraint for PCI compliance.

```
oc apply -f demo/setup/stage0/
```

4. Deploy demo applications.

```
#deploy dev app stack
oc apply -f demo/setup/dev

#deploy acme app stack
oc apply -f demo/setup/acme

#deploy storefront app stack
oc apply -f demo/setup/storefront

#deploy hipstershop app stack
oc apply -f demo/setup/hipstershop
```

5. Deploy compliance reports which will be scheduled as a *cronjob* that runs every hour for cluster and cis benchmark reports. These compliance reports will be needed for a later part of this lab.

```
oc apply -f demo/compliance-reports/cis-benchmark-report.yaml


oc apply -f demo/compliance-reports/cluster-reports.yaml
```

6. Deploy global alerts.
   The alerts will also be explored in a later lab.

```
oc apply -f demo/alerts/
```

7. Confirm the global compliance report and global alert are running.

```
oc get globalreport
oc get globalalert
```

The output looks like as below:

```
NAME                        CREATED AT
cis-results                 2021-09-30T15:42:33Z
cluster-inventory           2021-09-30T15:42:33Z
cluster-network-access      2021-09-30T15:42:33Z
cluster-policy-audit        2021-09-30T15:42:33Z


NAME                        CREATED AT
dns.unsanctioned.access     2021-09-30T15:42:40Z
network.lateral.access      2021-09-30T15:42:40Z
policy.globalnetworkset     2021-09-30T15:42:39Z
```

Part 4 - Try out some use cases

In this workshop we are going to focus on these main use cases:

- Identity-Aware Microsegmentation
- Zero Trust Workload Access Controls
- Runtime Threat Defense
- Observability and Troubleshooting

Identity-Aware Microsegmentation: App service control

Goal: Leverage network policies to segment connections within an OpenShiftcluster.

Steps

1. Test connectivity between application components and across application stacks. All of these tests should succeed as there are no policies in place.

a. Test connectivity between workloads within each namespace, use dev and hipstershop namespaces as example

```
# switch to hipstershop namespace
oc project hipstershop
# test connectivity within hipstershop namespace in 8080 port, the expected
result is "recommendationservice (x.x.x.x:8080) open"
oc exec -it $(kubectl get po -l app=frontend
-ojsonpath='{.items[0].metadata.name}')  -c server -- sh -c 'nc -zv
recommendationservice 8080'
```

b. Test connectivity across namespaces dev/centosand hipstershop/frontend.

```
# test connectivity from dev namespace to hipstershop namespace, the
expected result is "HTTP/1.1 200 OK"
oc project dev
oc exec -t centos -- sh -c 'curl -m3 -sI http://frontend.hipstershop
2>/dev/null | grep -i http'
```

c. Test connectivity from each namespace dev and default to the Internet.

```
# test connectivity from dev namespace to the Internet, the expected result
is "HTTP/1.1 200 OK"
oc exec -t centos -- sh -c 'curl -m3 -sI http://www.google.com 2>/dev/null |
grep -i http'

# test connectivity from default namespace to the Internet, the expected
result is "HTTP/1.1 200 OK"
oc project default
oc exec -it curl-demo -- sh -c 'curl -m3 -sI http://www.google.com
2>/dev/null | grep -i http'
```

2. Apply staged default-deny policy.
   Staged default-deny policy is a good way of catching any traffic that is not explicitly allowed by a policy without explicitly blocking it. We included dev and hipstershop namespace here as examples. You can add more namespace in default-deny policy when moving to "zero-trust" day by day.

```
oc apply -f demo/app-control/staged.default-deny.yaml
```

You can view the effect of the staged default-deny policy if you navigate to the Dashboard view in the Enterprise Manager UI and look at the Packets by Policy histogram.

```
# make a request across namespaces and view Packets by Policy histogram, the
expected result is "HTTP/1.1 200 OK"
for i in {1..5}; do kubectl -n dev exec -t centos -- sh -c 'curl -m3 -sI
http://frontend.hipstershop 2>/dev/null | grep -i http'; sleep 2; done
```

The staged policy does not affect the traffic directly but allows you to view the policy impact if it were to be enforced. You can see the denied traffic in the staged policy.

3. Apply network policies to your application to explicitly allow and deny control.

```
# deploy policy to control centos ingress and egress
oc apply -f demo/app-control/default.centos.yaml
```

4. Test connectivity with policies in place.
   a. The connections within namespace hipstershop should be allowed as usual.

```
# test connectivity within hipstershop namespace in 8080 port, the expected
result is "recommendationservice (x.x.x.x:8080) open"


oc project hipstershop
oc exec -it $(oc get po -l app=frontend
-ojsonpath='{.items[0].metadata.name}')  -c server -- sh -c 'nc -zv
recommendationservice 8080'
```

   b. The connections across dev/centos pod and hipstershop/frontend pod should be blocked by the application policy.

```
# test connectivity from dev namespace to hipstershop namespace, the
expected result is "command terminated with exit code 1"
oc project dev
oc exec -t centos -- sh -c 'curl -m3 -sI http://frontend.hipstershop
2>/dev/null | grep -i http'
```

   c. Test connectivity from each namespace dev and default to the Internet.

```
# test connectivity from dev namespace to the Internet, the expected result
is "command terminated with exit code 1"
```

```
oc exec -t centos -- sh -c 'curl -m3 -sI http://www.google.com 2>/dev/null |
grep -i http'
# test connectivity from default namespace to the Internet, the expected
result is "HTTP/1.1 200 OK"
oc project default
oc exec -it curl-demo -- sh -c 'curl -m3 -sI www.google.com 2>/dev/null |
grep -i http'
```

5. Implement an explicit policy to allow egress access from a workload in one namespace/pod, e.g. dev/centos, to hipstershop/frontend.

   a. Deploy an egress policy between two namespaces dev and hipstershop.

```
oc apply -f demo/app-control/platform.centos-to-frontend.yaml
```

   b. Test connectivity between dev/centos pod and hipstershop/frontend service again, should be allowed now.

```
oc project dev
oc exec -t centos -- sh -c 'curl -m3 -sI http://frontend.hipstershop
2>/dev/null | grep -i http'
#output is HTTP/1.1 200 OK
```

The access should be allowed once the egress policy is in place.

Now that we have proper policies in place, we can test the staged.default-deny policy as a beta version to test the E-W control. For example, adding default namespace and using *curl* to reach google, you should be able to see the deny flow under staged.default-deny policy. Once you are happy with the results, you can use the Policies Board view in the Enterprise Manager UI to enforce it as default-deny policy manifest.

Identity-Aware Microsegmentation: Pod Microsegmentation

Goal: Configure a zone-based architecture for our storefront application, and understand how to troubleshoot with flow visualization.

Steps

1. Create a zone-based policy for storefront application.

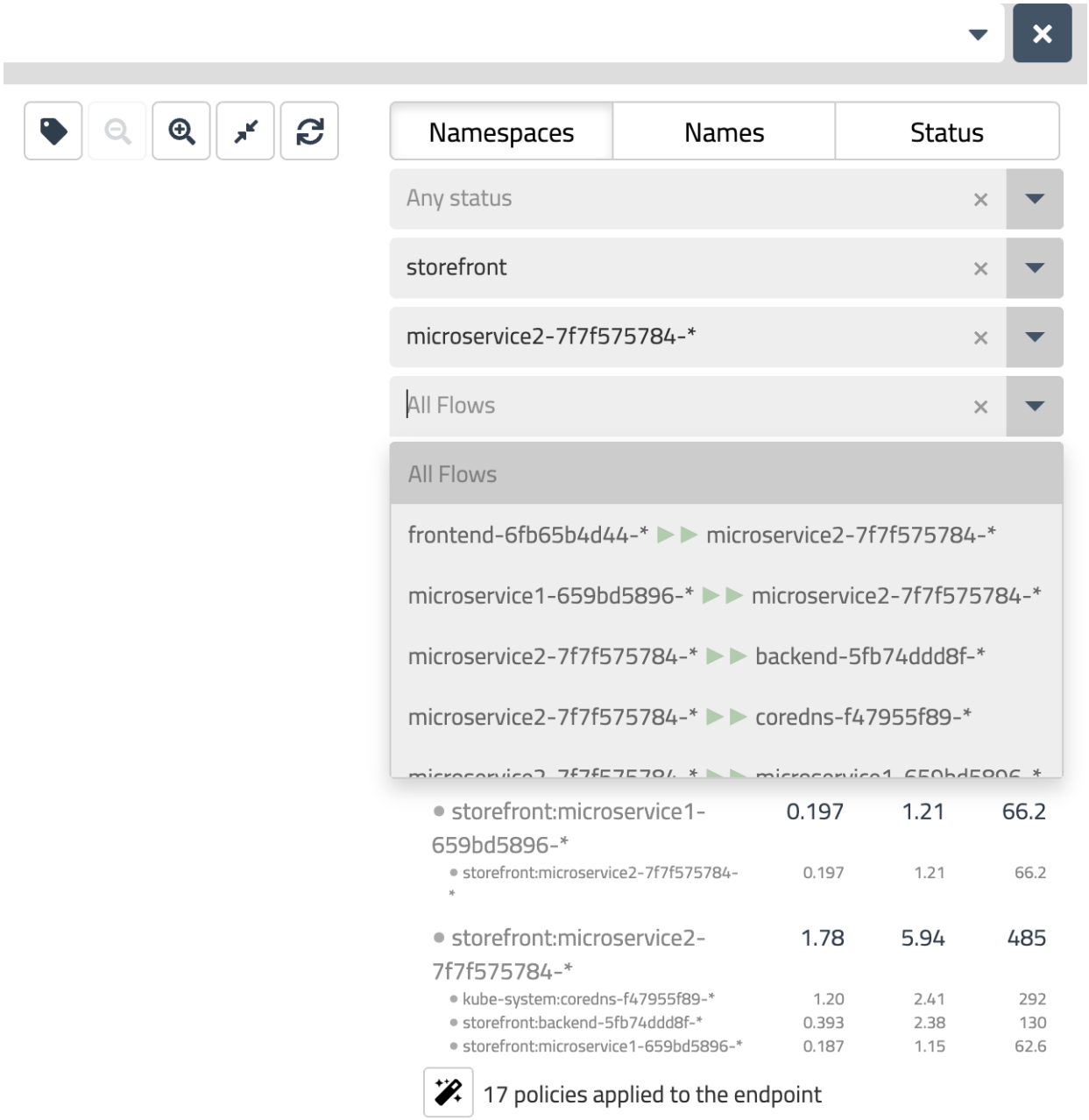
   a. Check that each microservice has the proper zone-based label.

```
oc project storefront
oc get pods --show-labels
```

   b. Apply the zone-based policy.

```
oc apply -f demo/app-control/FirewallZonesPolicies.yaml
```

2. Confirm the connection from microservice2 to backend has been allowed from flow visualization.

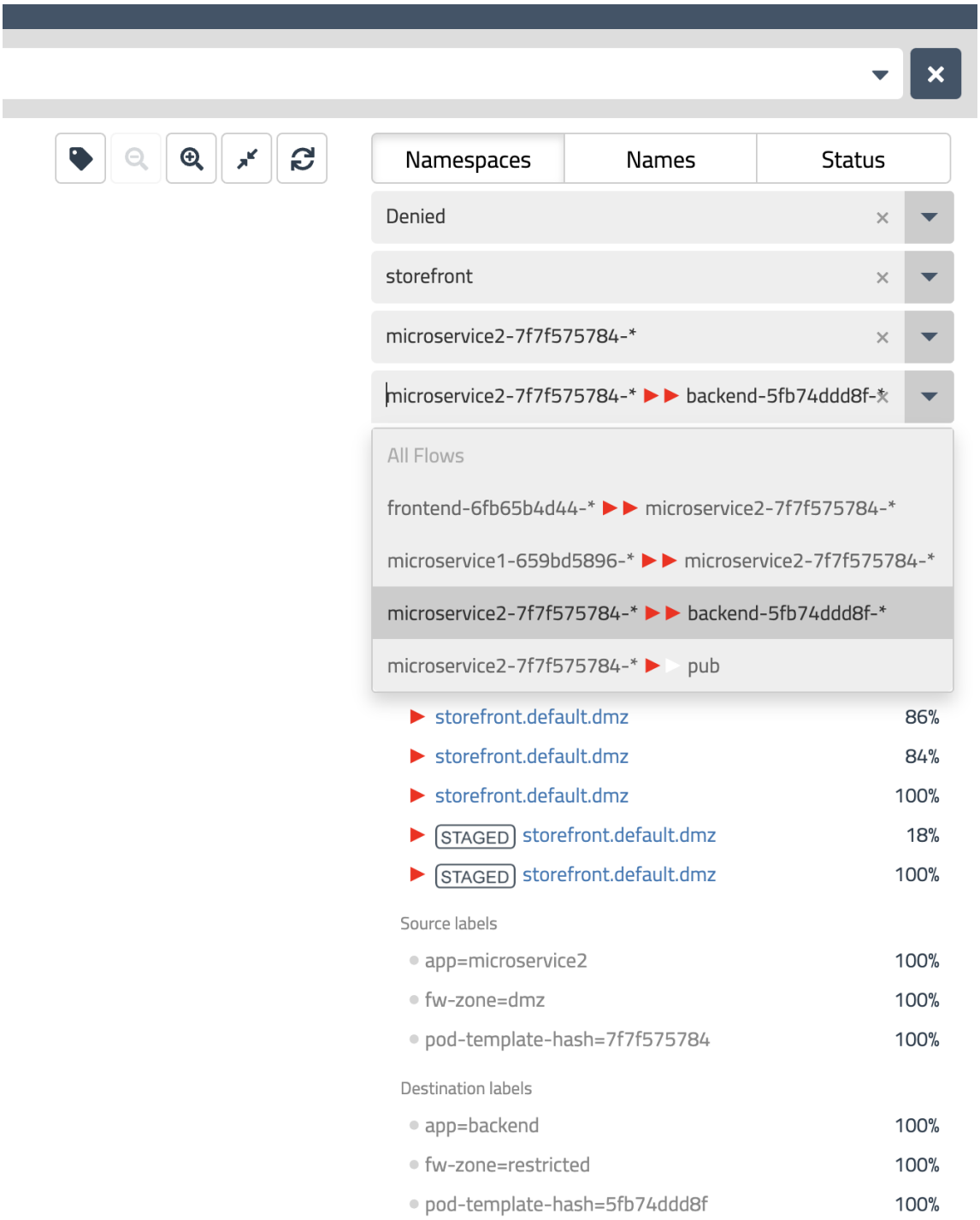


3. Change the label of pod mircoservice2 and see the deny traffic in flow visualization.

```
#remove the label
oc label pod $(kubectl -n storefront get po -l app=microservice2
-ojsonpath='{.items[0].metadata.name}') fw-zone-
```

```
#add the label as dmz zone
oc label pod $(kubectl -n storefront get po -l app=microservice2
-ojsonpath='{.items[0].metadata.name}')  fw-zone=dmz
```

4. Confirm the connection from microservice2 to backend has been denied.



5. Reverse the label of pod mircoservice2 with overwrite.

```
oc label pod $(kubectl -n storefront get po -l app=microservice2
-ojsonpath='{.items[0].metadata.name}') fw-zone=trusted --overwrite
```

Zero Trust Workload Access Controls: DNS Egress Controls

Goal: Configure egress access control for specific workloads so they are allowed to connect to external DNS domain.

Steps

1. Implement a DNS policy to allow the external endpoint access from a specific workload, e.g. dev/centos.

   a. Apply a policy to allow access to api.twilio.com endpoint using a DNS rule.

```
# deploy dns policy
oc apply -f demo/dns-egress-control/dns-policy.yaml
# test egress access to api.twilio.com
oc project dev
oc exec -t centos -- sh -c 'curl -m3 -skI https://api.twilio.com
2>/dev/null | grep -i http'
# test egress access to www.google.com
oc exec -t centos -- sh -c 'curl -m3 -skI https://www.google.com
2>/dev/null | grep -i http'
```

Access to the api.twilio.com endpoint should be allowed by the DNS policy. Any other external endpoints, like www.google.com, should be denied.

   b. Modify the dns policy to include *.google.com and test egress access to www.google.com.

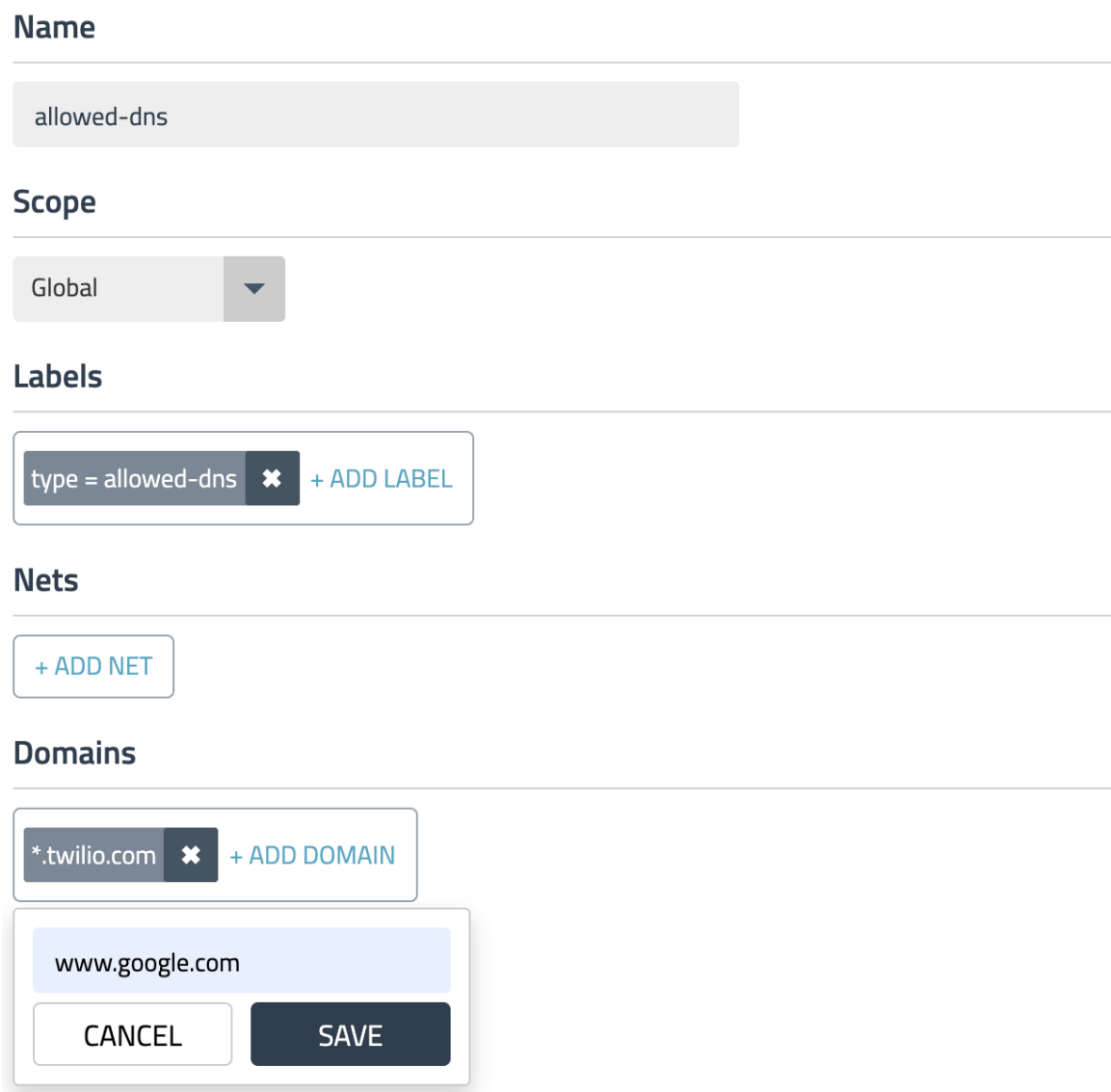
```
# test egress access to www.google.com again and it should be
allowed.
oc exec -t centos -- sh -c 'curl -m3 -skI https://www.google.com
```

```
2>/dev/null | grep -i http'
```

2. Edit the policy to use a NetworkSet with DNS domain instead of inline DNS rule.
   a. Apply a policy to allow access to api.twilio.com endpoint using a DNS policy.

```
# deploy network set
oc apply -f demo/dns-egress-control/netset.allowed-dns.yaml
# deploy DNS policy using the network set
oc apply -f demo/dns-egress-control/dns-policy.netset.yaml
# test egress access to api.twilio.com
oc exec -t centos -- sh -c 'curl -m3 -skI https://api.twilio.com
2>/dev/null | grep -i http'
# test egress access to www.google.com
oc exec -t centos -- sh -c 'curl -m3 -skI https://www.google.com
2>/dev/null | grep -i http'
```

   b. Modify the NetworkSet to include www.google.com in dns domain and test egress access to www.google.com again.




```
# test egress access to www.google.com again and it should be allowed.
oc exec -t centos -- sh -c 'curl -m3 -skI https://www.google.com 2>/dev/null
| grep -i http'
```

Runtime Threat Defense: Global Threadfeed

Goal: Configure egress access control for outside threadfeed policy, so workloads within the cluster are not allowed to external networkset.

Steps

1. Protect workloads with Global Threatfeed from known bad actors.

   Calicocloud offers Global Threatfeed resource to prevent known bad actors from accessing OpenShiftpods.

```
oc get globalthreatfeeds
```

Output is

```
NAME                          CREATED AT
alienvault.domainthreatfeeds  2021-09-28T15:01:33Z
alienvault.ipthreatfeeds      2021-09-28T15:01:33Z
```

You can get these domain/ip list from the yaml file. The url would be:

```
oc get globalthreatfeeds alienvault.domainthreatfeeds -ojson | jq -r
'.spec.pull.http.url'

oc get globalthreatfeeds alienvault.ipthreatfeeds -ojson | jq -r
'.spec.pull.http.url'
```

Output is

```
https://installer.calicocloud.io/feeds/v1/domains

https://installer.calicocloud.io/feeds/v1/ips
```

```
# deploy feodo and snort threatfeeds
oc apply -f demo/threatfeeds/feodo-tracker.yaml
oc apply -f demo/threatfeeds/feodo-block-policy.yaml

# Confirm and check the tracker threatfeed
oc get globalthreatfeeds
```

```
NAME                           CREATED AT
alienvault.domainthreatfeeds   2021-09-28T15:01:33Z
alienvault.c       2021-09-28T15:01:33Z
feodo-tracker                  2021-09-28T17:32:13Z
```

2. Generate alerts by accessing the IP from feodo-tracker list.

```
# confirm you are in `dev` project
oc status
```

```
# try to ping any of the IPs in from the feodo tracker list.
FIP=$(kubectl get globalnetworkset threatfeed.feodo-tracker -ojson |
jq -r '.spec.nets[0]' | sed -e 's/^"//' -e 's/"$//' -e 's/\/32//')
oc exec -t netshoot -- sh -c "ping -c1 $FIP"
```

3. Generate alerts by accessing the IP from alienvault.ipthreatfeeds list.

```
# try to ping any of the IPs in from the ipthreatfeeds list.
AIP=$(kubectl get globalnetworkset
threatfeed.alienvault.ipthreatfeeds -ojson | jq -r '.spec.nets[0]' |
sed -e 's/^"//' -e 's/"$//' -e 's/\/32//')
oc exec -t netshoot -- sh -c "ping -c1 $AIP"
```

4. Add more threatfeeds into networkset.

```
# deploy embargo and other threatfeeds
oc apply -f demo/threatfeeds/embargo.networkset.yaml
oc apply -f demo/threatfeeds/security.embargo-countries.yaml
```

5. Confirm you are able to see the alert in alert list.

wep dev/netshoot connected to suspicious IP 51.178.161.32 from list feodo-tracker | Suspicious IP Detected | 01-27-22 22:07:59 GMT

QUERIES

Protocol	Source			Destination				Flow Action	Feeds	Severity	Suspicious Prefix	Alert
	IP	Namespace	Name	Port	IP	Namespace	Name					
icmp	10.131.156.74	dev	netshoot		51.178.161.32	-	threatfeed.feodo-tracker	denied	feodo-tracker	100		

wep dev/netshoot connected to suspicious IP 5.188.108.22 from list alienvault.ipthreatfeeds | Suspicious IP Detected | 01-27-22 22:31:09 GMT

QUERIES

Protocol	Source			Destination				Flow Action	Feeds	Severity	Suspicious Prefix	Alert
	IP	Namespace	Name	Port	IP	Namespace	Name					
icmp	10.131.156.74	dev	netshoot		5.188.108.22	-	threatfeed.alienvault.ipthreatfeeds	denied	alienvault.ipthreatfeeds	100		

Observability: Manager UI

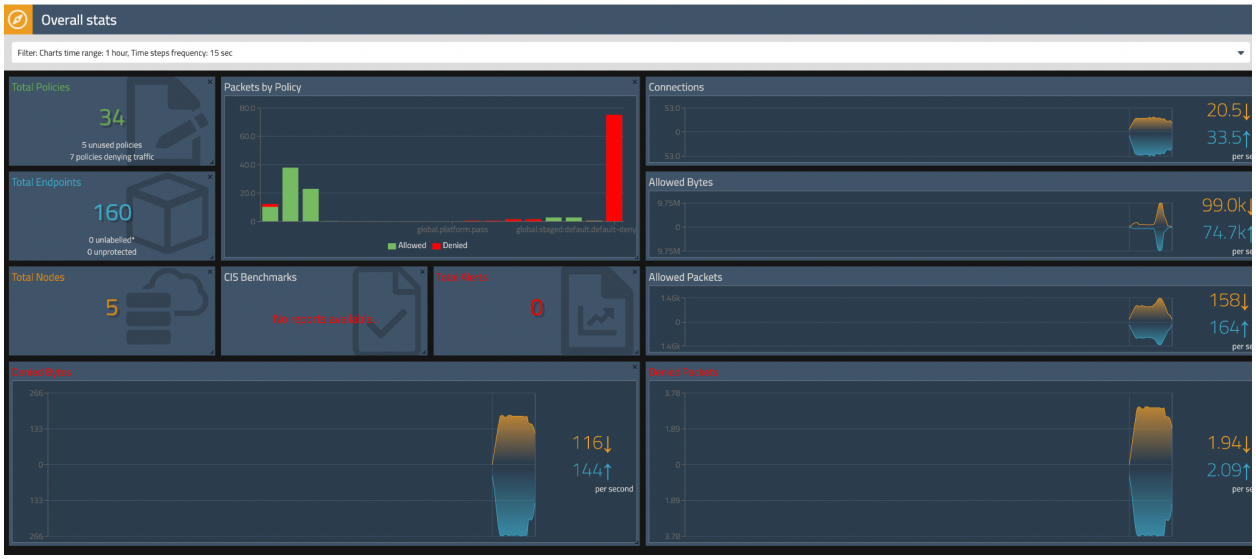
Goal: Explore Calico observability tools in manager UI.

Calico observability tools

1. Dashboard

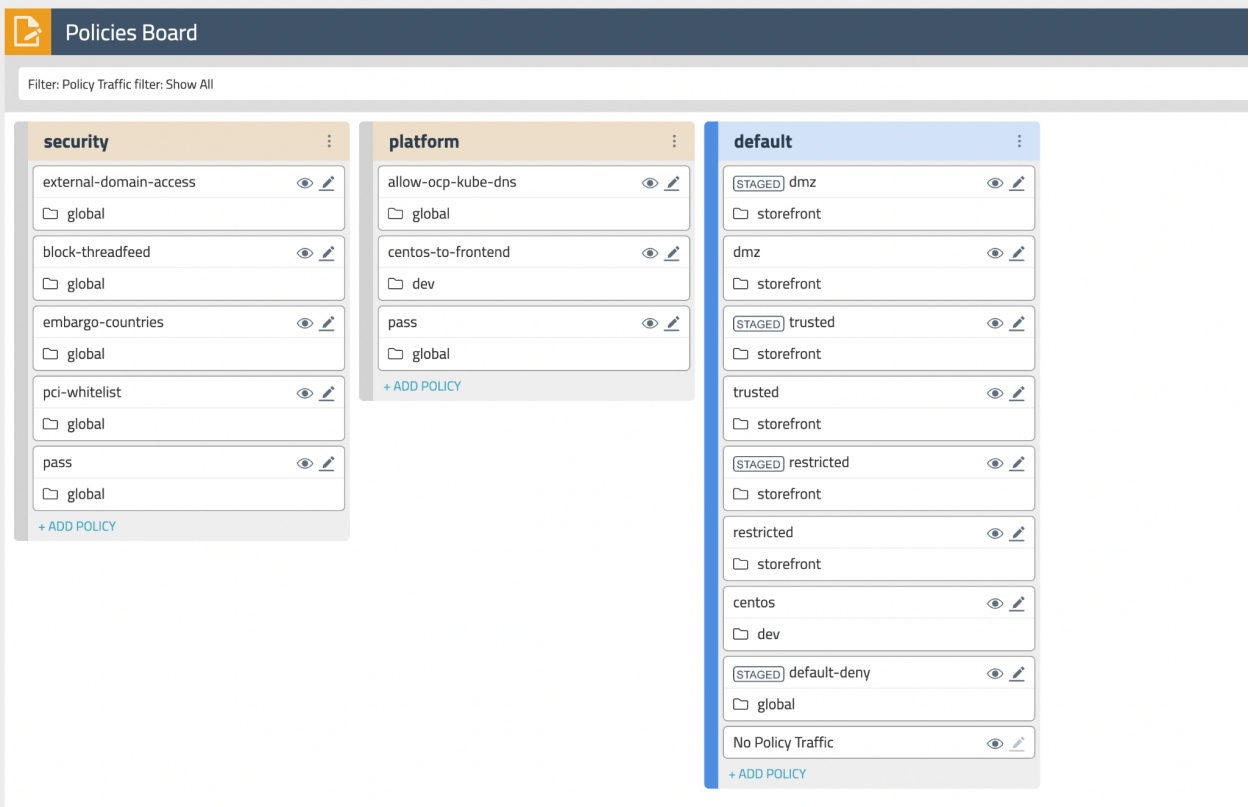
   The Dashboard view in the Enterprise Manager UI presents a high level overview of the status of your cluster. The view shows the following information:

   - Connections, Allowed Bytes and Packets
   - Denied Bytes and Packets
   - Total number of Policies, Endpoints and Nodes
   - Summary of CIS benchmarks
   - Count of triggered alerts
   - Packets by Policy histogram. This shows allowed and denied traffic as it is being evaluated by network policies



2. Policies Board

   The Policies Board shows all policies deployed in the cluster and organized into policy tiers. You can control what a user can see and do by configuring OpenShift RBAC roles which determine what the user can see in this view. You can also use controls to hide away tiers you're not interested in at any given time.



3. Audit timeline
   The Timeline view shows an audit trail of created, deleted, or modified resources.

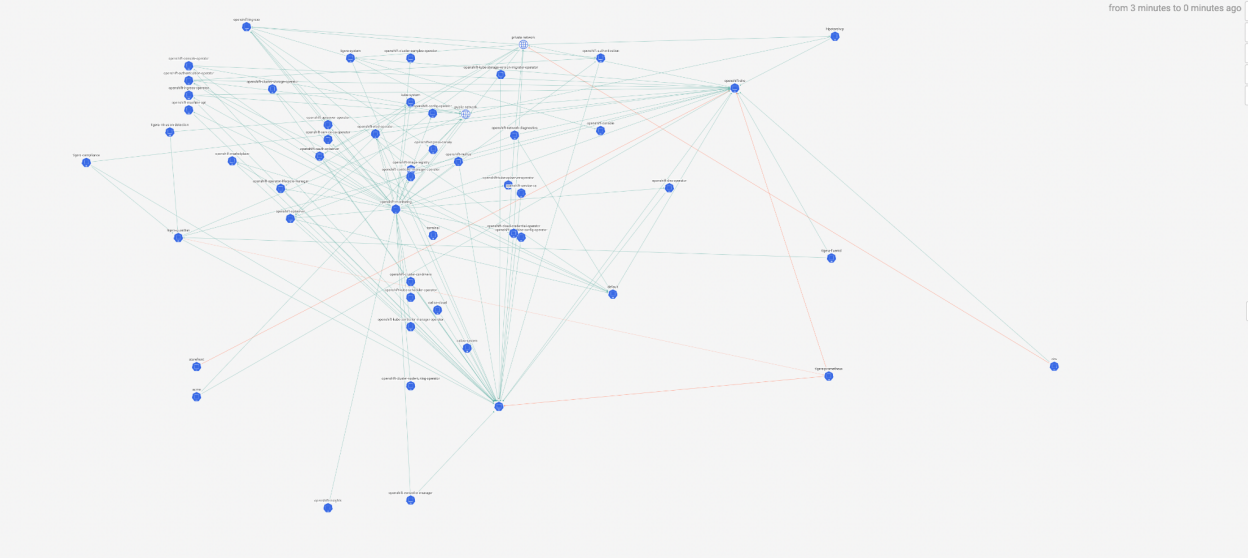
4. Endpoints

   The Endpoints view lists all endpoints known to Calico. It includes all Kubernetes endpoints, such as Pods, as well as Host endpoints that can represent a Kubernetes host or an external VM or bare metal machine. We will elaborate host endpoints in later module.

Name	Namespace	IP addresses	Node	Type	Labels	Policies
apiserver-6fc59fd57f-szlh9	openshift-apiserver	10.130.189.89/32	ip-10-0-133-149.us-east-2.compute.internal		apiserver: true, app: openshift-apiserver-a, openshift-...	7
apiserver-74ff7cfc94-s4xlm	openshift-oauth-apiserver	10.130.189.127/32	ip-10-0-133-149.us-east-2.compute.internal		apiserver: true, app: openshift-oauth-apiserver, oauth...	7
aws-ebs-csi-driver-operator-6cd6f59c48-5g9cw	openshift-cluster-csi-drivers	10.130.189.122/32	ip-10-0-133-149.us-east-2.compute.internal		name: aws-ebs-csi-driver-operator, pod-template-ha...	7
cluster-samples-operator-6574bc78dd-rkl76	openshift-cluster-samples-operator	10.130.189.117/32	ip-10-0-133-149.us-east-2.compute.internal		name: cluster-samples-operator, pod-template-hash: ...	7
compliance-benchmarker-2qkx9	tigera-compliance	10.130.189.82/32	ip-10-0-133-149.us-east-2.compute.internal		controller-revision-hash: 67558f89b, k8s-app: comp...	9
console-cf486d57d-tmn5g	openshift-console	10.130.189.70/32	ip-10-0-133-149.us-east-2.compute.internal		app: console, component: ui, pod-template-hash: cf48...	7
console-operator-578895bddc-lj7jr	openshift-console-operator	10.130.189.65/32	ip-10-0-133-149.us-east-2.compute.internal		name: console-operator, pod-template-hash: 578895...	7
controller-manager-9g2ng	openshift-controller-manager	10.130.189.123/32	ip-10-0-133-149.us-east-2.compute.internal		app: openshift-controller-manager, controller-manage...	7
csi-snapshot-controller-f6cff65bd-jjvff	openshift-cluster-storage-operator	10.130.189.115/32	ip-10-0-133-149.us-east-2.compute.internal		app: csi-snapshot-controller, pod-template-hash: f6cf...	7
csi-snapshot-webhook-6c5f8c99bc-s5mdb	openshift-cluster-storage-operator	10.130.189.71/32	ip-10-0-133-149.us-east-2.compute.internal		app: csi-snapshot-webhook, pod-template-hash: 6c5f...	7
dns-default-xm5m8	openshift-dns	10.130.189.126/32	ip-10-0-133-149.us-east-2.compute.internal		controller-revision-hash: 5bdd89c464, dns.operator.o...	7
downloads-7dcfbf9955-djs4b	openshift-console	10.130.189.68/32	ip-10-0-133-149.us-east-2.compute.internal		app: console, component: downloads, pod-template-h...	7
fluentd-node-qtsx8	tigera-fluentd	10.130.189.67/32	ip-10-0-133-149.us-east-2.compute.internal		controller-revision-hash: 77bd5b6fb4, k8s-app: fluent...	8
machine-config-controller-5dd46888845-qcd2n	openshift-machine-config-operator	10.130.189.79/32	ip-10-0-133-149.us-east-2.compute.internal		k8s-app: machine-config-controller, pod-template-ha...	7
migrator-f7d666d47-22wf2	openshift-kube-storage-version-migrator	10.130.189.83/32	ip-10-0-133-149.us-east-2.compute.internal		app: migrator, pod-template-hash: f7d666d47, projec...	7
multus-admission-controller-zvtcw	openshift-multus	10.130.189.69/32	ip-10-0-133-149.us-east-2.compute.internal		app: multus-admission-controller, component: networ...	7
network-check-target-rnj7l	openshift-network-diagnostics	10.130.189.120/32	ip-10-0-133-149.us-east-2.compute.internal		app: network-check-target, controller-revision-hash: ...	7
network-metrics-daemon-j2np2	openshift-multus	10.130.189.88/32	ip-10-0-133-149.us-east-2.compute.internal		app: network-metrics-daemon, component: network, ...	7
oauth-openshift-5b864d7c7-k989q	openshift-authentication	10.130.189.114/32	ip-10-0-133-149.us-east-2.compute.internal		app: oauth-openshift, oauth-openshift-anti-affinity: tr...	7
packageserver-59f96979f9-g2rhg	openshift-operator-lifecycle-manager	10.130.189.104/32	ip-10-0-133-149.us-east-2.compute.internal		app: packageserver, pod-template-hash: 59f96979f9, ...	7
pod-identity-webhook-654f5d9455-2pgnj	openshift-cloud-credential-operator	10.130.189.125/32	ip-10-0-133-149.us-east-2.compute.internal		app: pod-identity-webhook, pod-template-hash: 654f...	7
prometheus-operator-8cf88c6db-w5jh2	openshift-monitoring	10.130.189.92/32	ip-10-0-133-149.us-east-2.compute.internal		app.kubernetes.io/component: controller, app.kuberne...	7

5. Dynamic Service and Threat Graph.

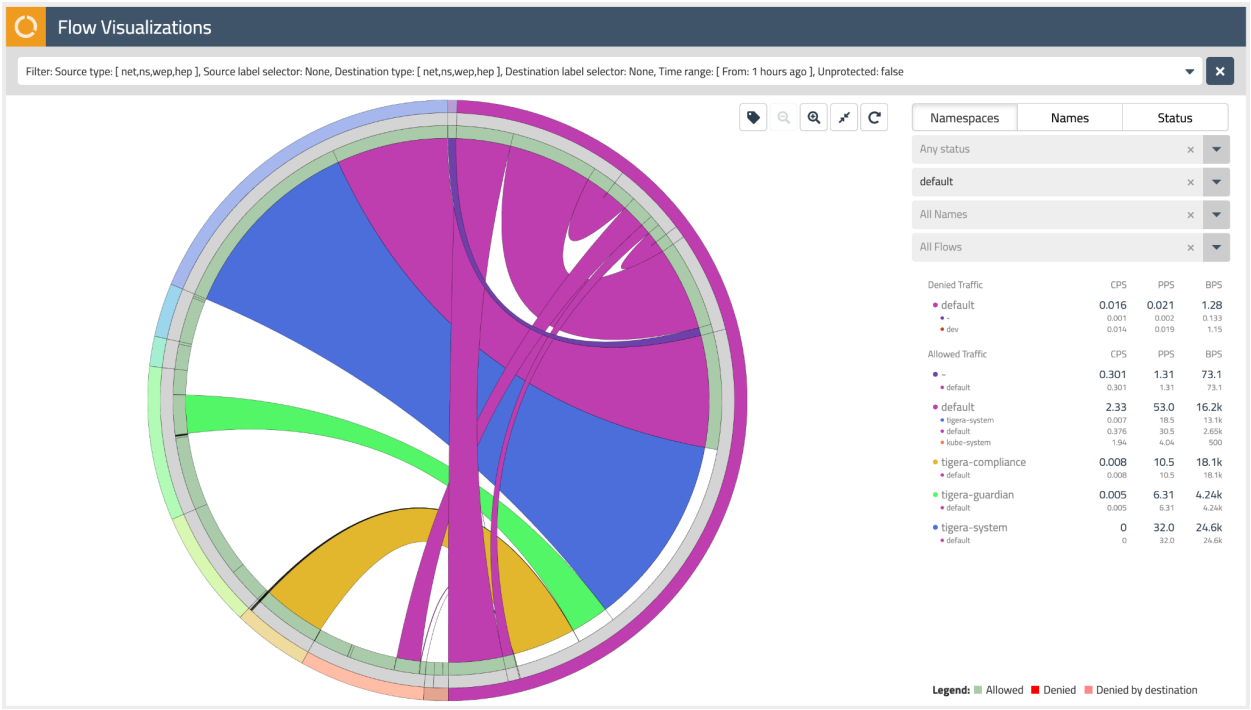
   The Dynamic Service and Threat Graph presents network flows from a service level perspective. Top level view shows how traffic flows between namespaces and external and internal endpoints.



- When you select any node representing a namespace, you will get additional details about the namespace, such as incoming and outgoing traffic, policies evaluating each flow, and DNS metrics.
- When you select any edge, you will get details about the flows representing that edge.
- If you expand a namespace by double-clicking on it, you will get the view of all components of the namespace.

6. Flow Visualizations

   The Flow Visualizations view shows all point-to-point flows in the cluster. It allows you to see the cluster traffic from the network point of view.



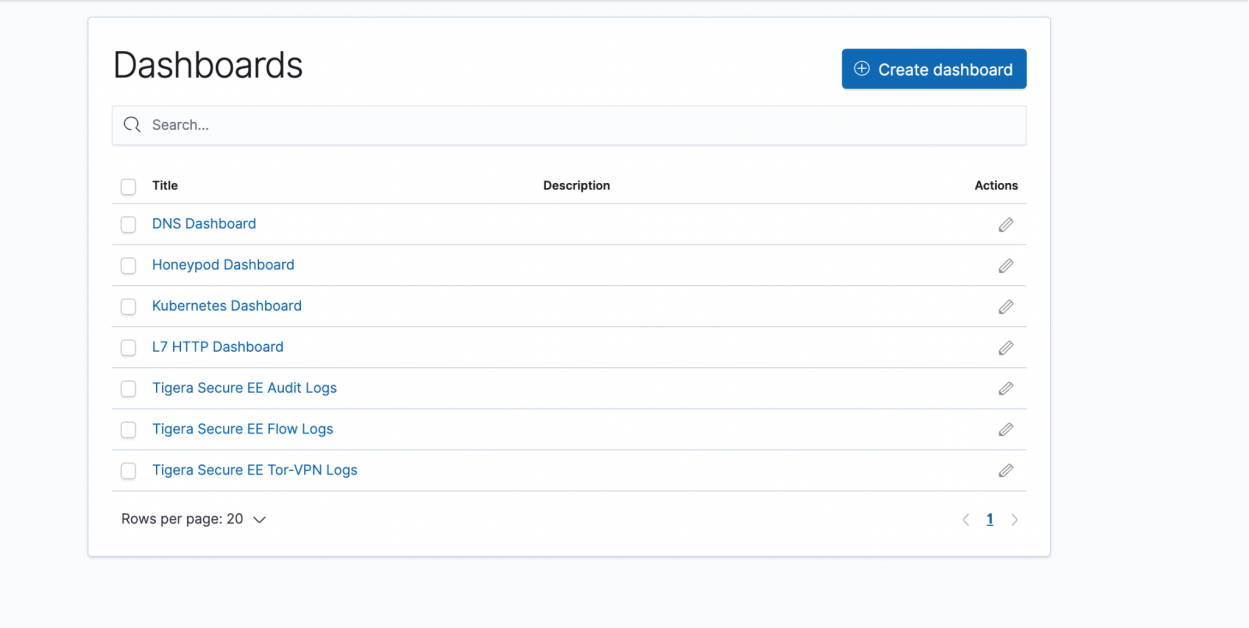
Observability: Kibana Dashboard

Goal: Explore Calico observability tools in Kibana UI.

Calico observability tools

1. Kibana dashboards

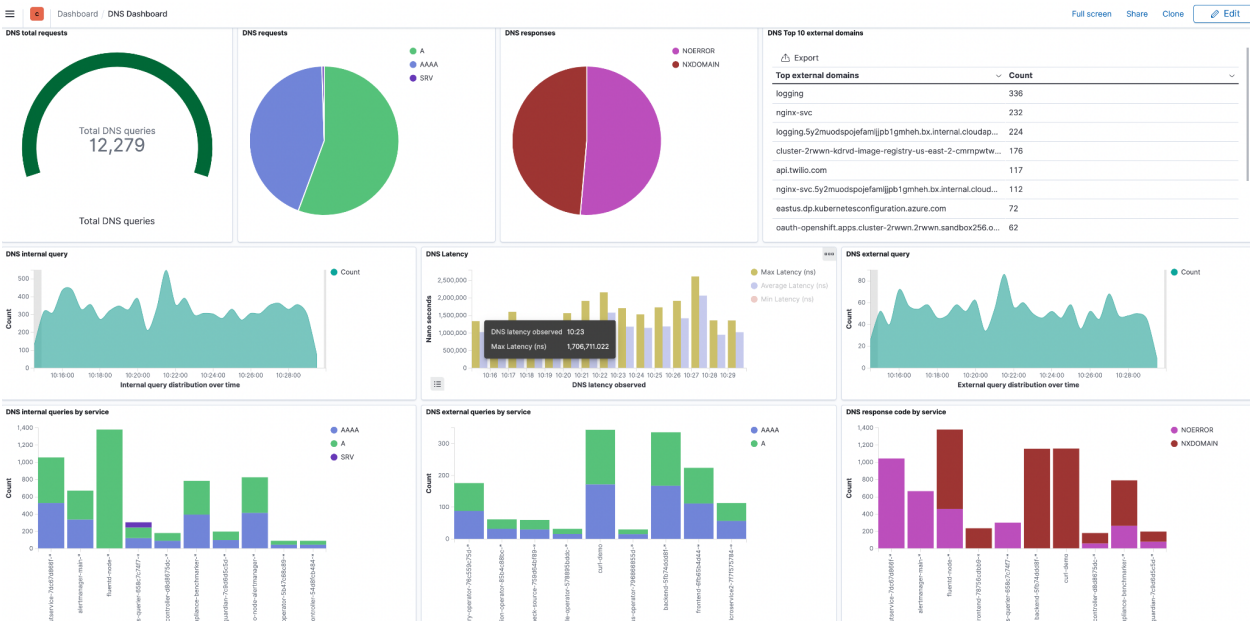
   The Kibana components comes with Calico commercial offerings and provides access to raw flow, audit, and dns logs, as well as the ability to visualize the collected data in various dashboards.



Calico provides some of the default dashboards you get access to, including DNS Logs, Flow Logs, Audit Logs, Kubernetes API calls, L7 HTTP metrics etc. You can also customize different dashboards.

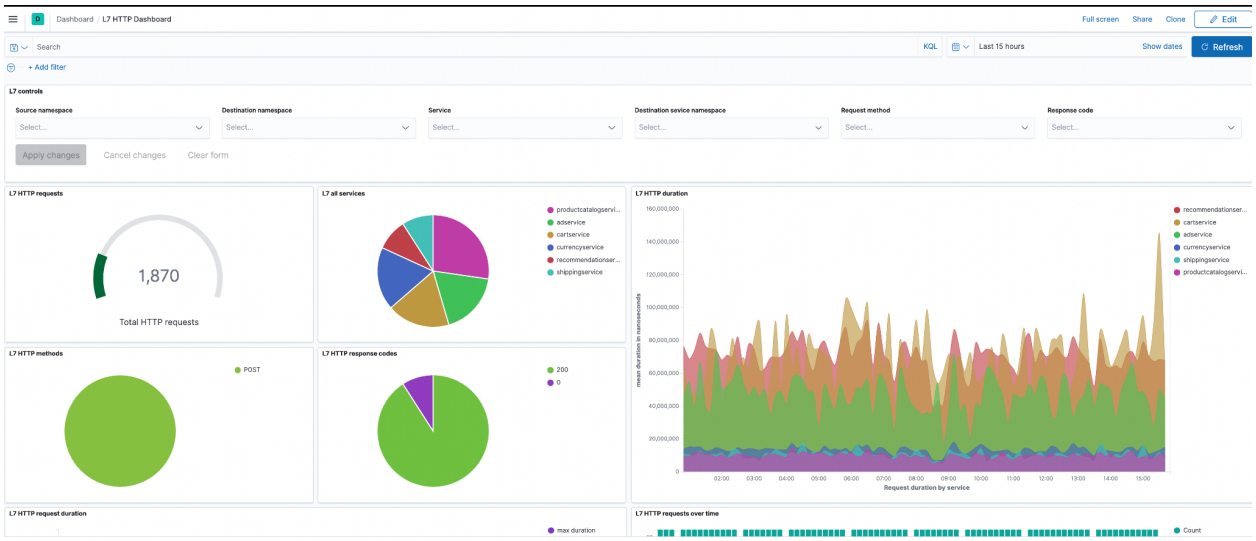
2. DNS dashboards

   The DNS dashboard shows an overview of how DNS behaves in the cluster, including internal & external queries, and DNS latency below.



3. Application-level Logs

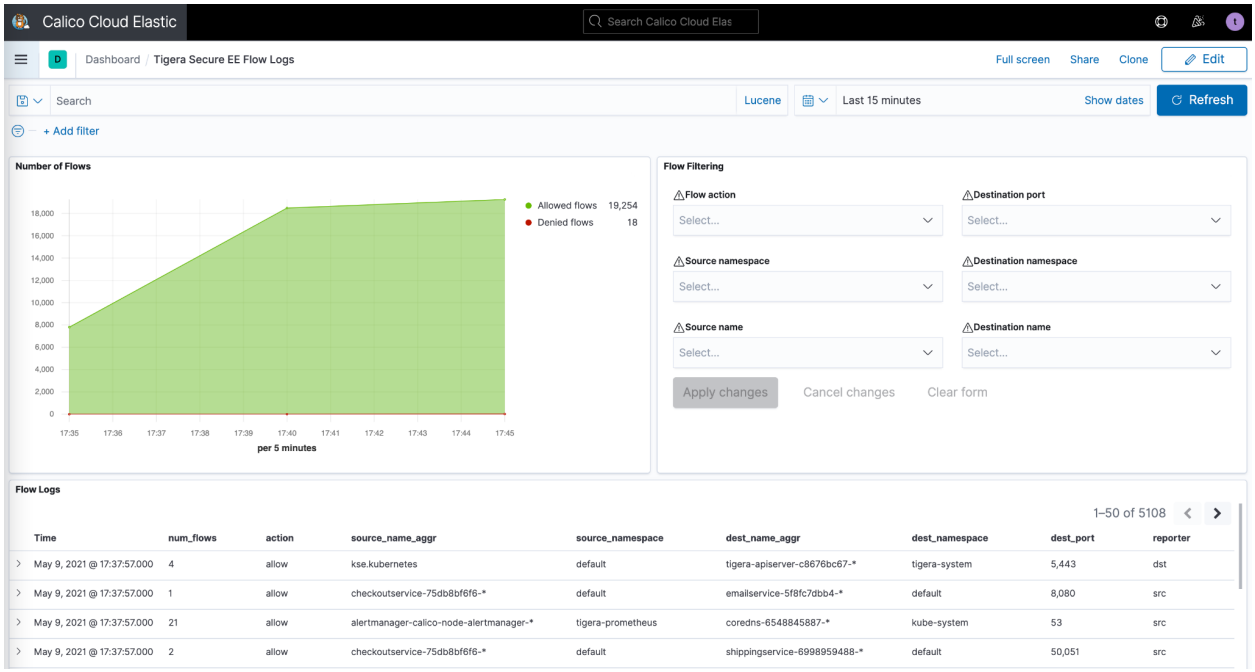
   The L7 dashboard shows all details related to http protocol including method, response code, url etc. We will enable L7 logs in a later module.



4. Flow logs

   The Tigera Secure ee flow log dashboard will give you all details related to every flow log, including denying and allowing action and the reporter.

   In addition, you can find out a full Kubernetes context while expanding each flow log.



5. Audit logs

   The audit logs dashboard is useful as you can filter by username and verb. The example below filters by user cc-demo@tigera.io, which means the action is from the UI manager user.

Observability: eBPF based Application-level Observability

Goal: Enable L7/HTTP flow logs in hipstershop with Calico cloud.

Calico cloud not only can provide L3 flow logs, but can also provide L7 visibility without service mesh. For more details refer to Configure L7 logs documentation.

Steps

1. Configure Felix for log data collection.

```
oc patch felixconfiguration default --type='merge' -p
'{"spec":{"policySyncPathPrefix":"/var/run/nodeagent"}}'
```

2. Apply application layer resource and ensure that l7-collector and envoy-proxy containers are in the Running state. Refer to Log Collection Spec with different options.

```
cat > configs/alr7.yaml << EOF
apiVersion: operator.tigera.io/v1
kind: ApplicationLayer
metadata:
  name: tigera-secure
spec:
  logCollection:
    collectLogs: Enabled
    logIntervalSeconds: 5
    logRequestsPerInterval: -1
EOF
```

```
oc create -f configs/alr7.yaml
```

3. Confirm the daemonset is running for each node.

```
oc project calico-system
oc get daemonset.apps/l7-log-collector
```

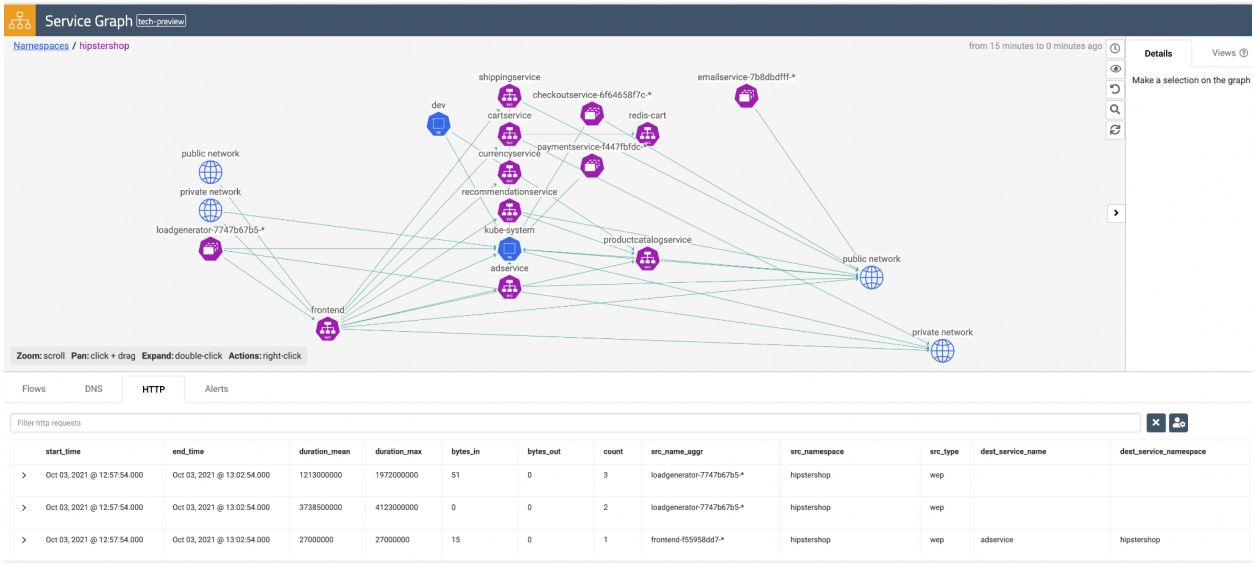
4. Select traffic for L7 log collection.

```
#Annotate the services you wish to collect L7 logs as shown. Use
hipstershop as example
oc project hipstershop
oc annotate svc --all projectcalico.org/l7-logging=true
```

5. [Optional] restart the pods in hipstershop if you want to see l7 logs right away.

```
oc delete pods --all
```

Now, view the L7 logs in Kibana by selecting the tigera_secure_ee_l7 index pattern. You should also see the relevant HTTP log from service graph.



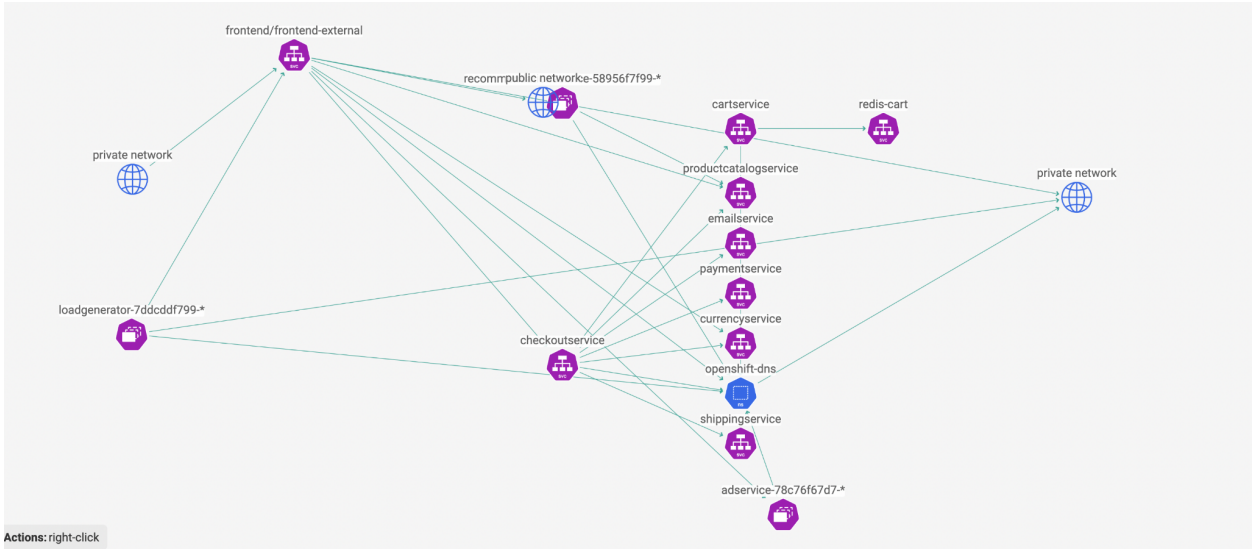
Observability: Dynamic packet capture

Goal: Configure packet capture for specific pods and review captured payload.

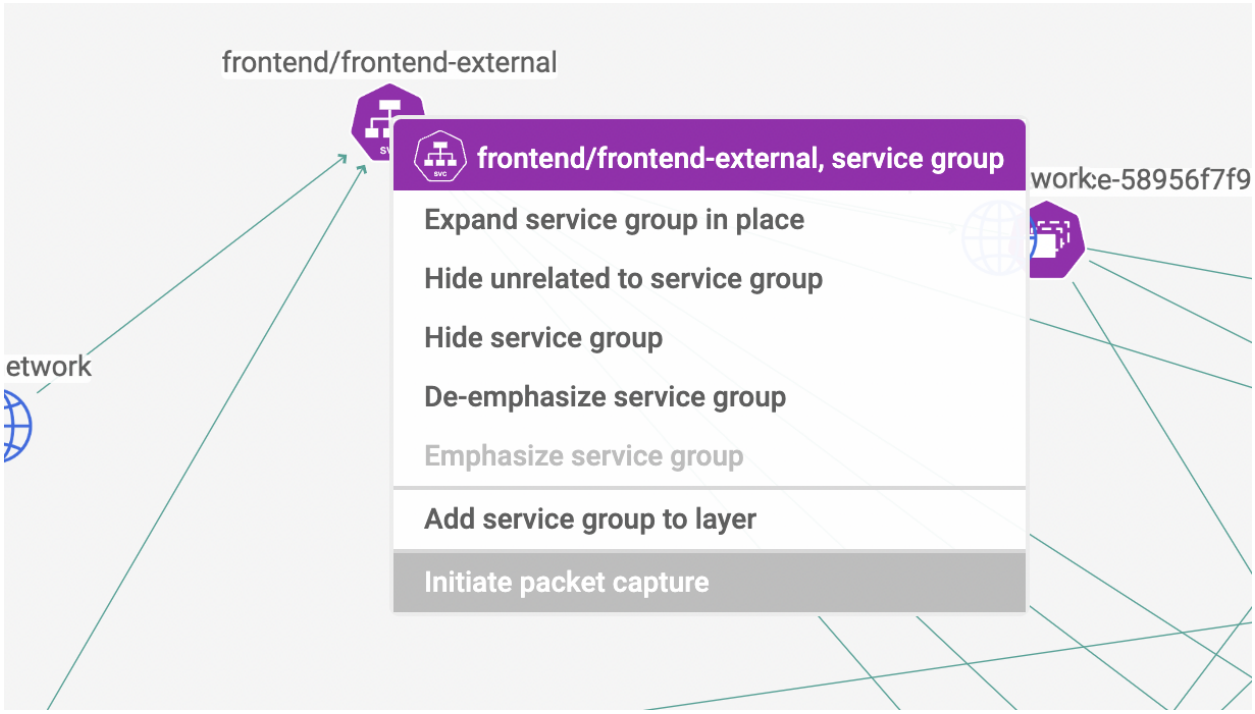
Calico enterprise/cloud provide enhanced packet capture feature for DevOps troubleshooting. Packet captures are Kubernetes Custom Resources and thus native Kubernetes RBAC can be used to control which users/groups can run and access Packet Captures; this may be useful if Compliance or Governance policies mandate strict controls on running Packet Captures for specific workloads. This demo is simplified without RBAC. Further details can be found here.

Steps

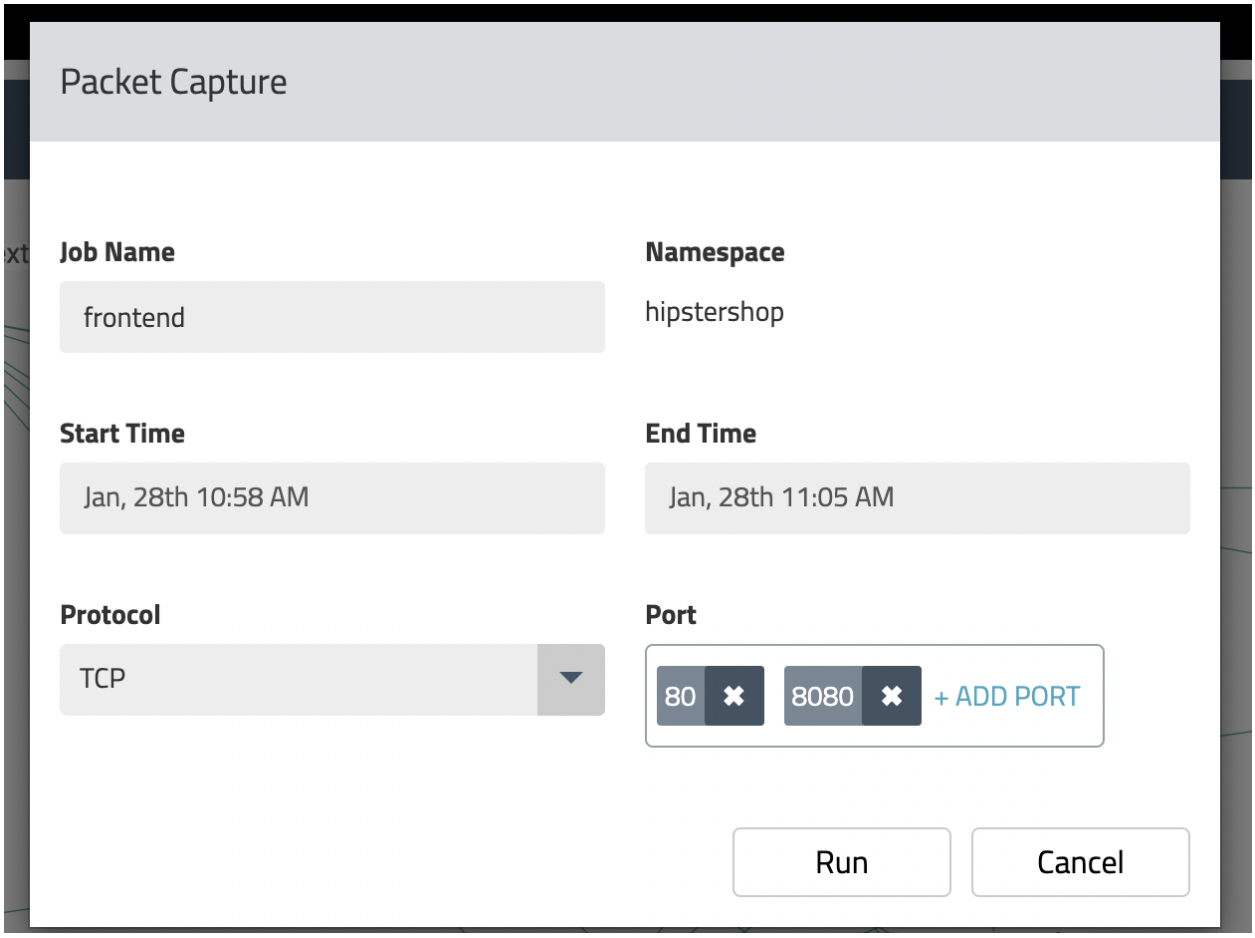
1. Login to the hipstershop namespace from service graph.



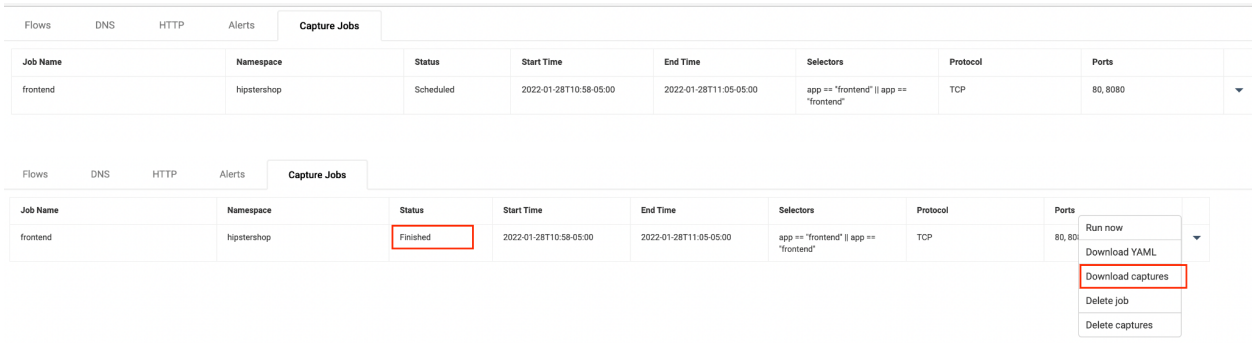
2. Choose a mircroservice which you want to run the packet capture job on. We will use frontend as an example.



3. We will schedule a packet capture job for TCP port 80 and 8080.



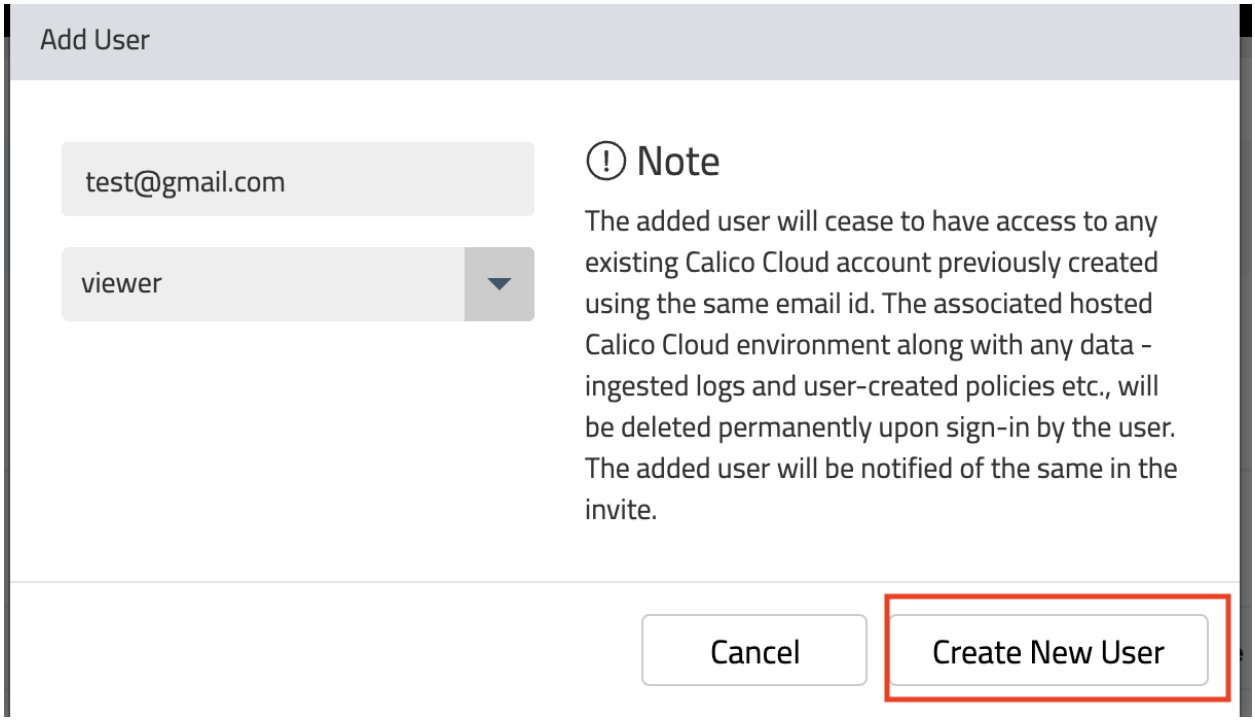
4. You will notice that the pcap job is now in scheduled mode.



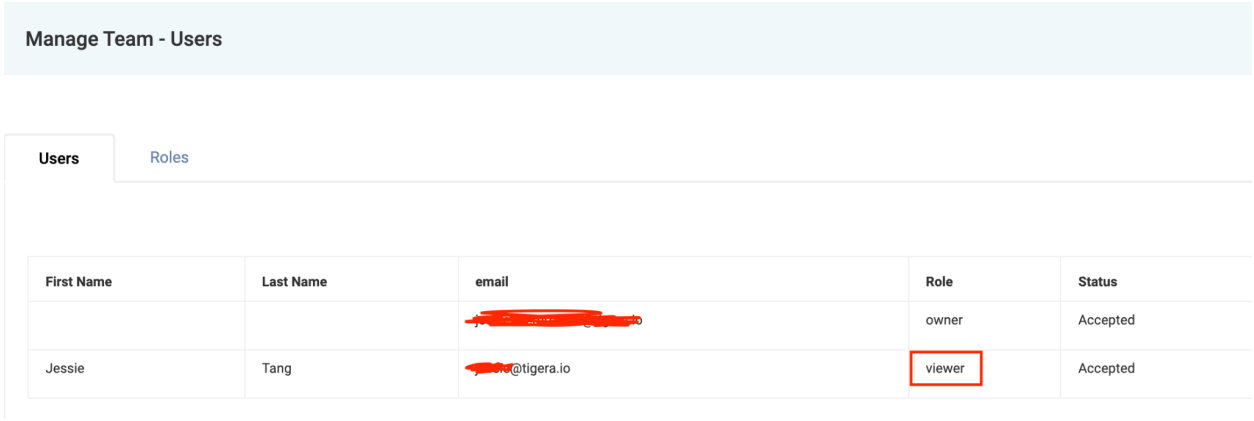
[Bonus] Calico Cloud provides a different RBAC role for packet capture resources.

Calico Cloud provides a pre-defined “viewer” role that doesn't have permissions to get/list Packet Capture resources.

1. Add one of your team members to this Calico Cloud management plane.



2. Confirm they accepted your invite.



Part 5 - Clean up your test environment

1. Delete the application stack to clean up any loadbalancer services.

```
oc delete -f demo/setup/dev
oc delete -f demo/setup/acme
```

```
oc delete -f demo/setup/storefront
oc delete -f demo/setup/hipstershop/
```

2. Remove calicocloud components from the cluster.
   - Download the script

```
curl -O
https://installer.calicocloud.io/manifests/v3.11.1-1/downgrade.sh
```

   - Make the script executable

```
chmod +x downgrade.sh
```

   - Run the script and read the help to determine if you need to specify any flags

```
./downgrade.sh --help
```

   - Run the script with any needed flags, for example:

```
./downgrade.sh --remove-prometheus --remove-all-calico-policy
```

3. Logout your managed cluster.

```
oc logout
```

SUMMARY

By the end of this workshop we have:

- Deployed an application consisting of microservices
- Implemented calico's zero-trust workload access controls to secure the flow of data between workloads in the openshift cluster and external resources.
- Used calico's identity-aware microsegmentation to implement the least privilege access controls for east-west traffic inside the cluster.
- Implemented a runtime threat defense mechanism to prevent any communication to malicious IP or domain using global threadfeed.
- Seen how the full-stack observability powered by eBPF significantly shortens the time to troubleshoot and plays a significant role in design decisions, workload placement, and resiliency.

End of Workshop!

Learn more

- Tigera for Red Hat OpenShift
  https://www.tigera.io/partners/redhat/
- Tigera Solution Brief hold
  https://www.redhat.com/watch?v=KLrwrxGSr5
- Extending OpenShift Security and Observability with Calico from Tigera
  https://cloud.redhat.com/blog/extending-openshift-security-and-observability-for-kubernetes-networking-and-microservices-with-calico-enterprise-from-tigera
- Tigera Operator for Red Hat OpenShift
  https://catalog.redhat.com/software/operators/detail/5e98747f6c5dcb34dfbb1a0f
- Podcast: OpenShift Network Security and Compliance using Calico Featuring: Tigera
  https://catalog.redhat.com/podcasts/4018052