Lesson Plan

Network Policy for Hosts and NodePorts

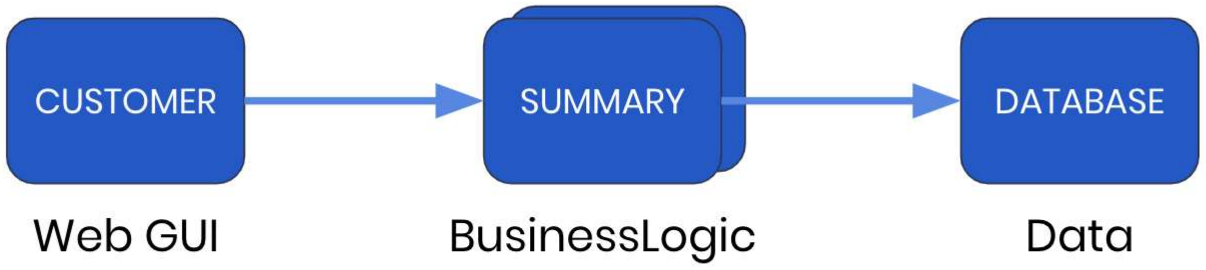
Summary

1. Lab Setup
2. Protect Kubernetes Nodes
3. Restrict Access to Kubernetes NodePorts

Lab Setup

This lab assumes you already have the "Yet Another Online Bank" (yaobank) installed, as created in the “Installing the Sample Application” module in Week 1. If you don’t already have it installed then go back and do so now.

For reference, this is architecture of YAOBank:



If you are not already on host1, you can enter host1 by using the multipass shell command.

```
multipass shell host1
```

Protect Kubernetes Nodes

Thus far, we've created policies that protect pods in Kubernetes. However, Calico Policy can also be used to protect the host interfaces in any standalone Linux node (such as a baremetal node, cloud instance or virtual machine) outside the cluster. Furthermore, it can also be used to protect the Kubernetes nodes themselves, including advanced use cases such as restricting access to NodePort services from outside the cluster.

Let's explore these more advanced scenarios, and how Calico policy can be used to protect these.

Observe that Kubernetes nodes are not yet secured

Run the command below from the standalone Host (host1) to see if we have access to FTP on the control node.

```
nc -w 3 198.19.0.1 21
```

This should succeed - i.e. you were able access the FTP server, which is not a good security posture. (Note that we used FTP in this example as an easy illustration. Your nodes probably aren't running an FTP daemon, but will likely be open to other attack vectors.)

Create Network Policy for Nodes

Let's create some policies for the kubernetes nodes. Host endpoints are non-namespaced. So in order to secure host endpoints we'll need to use Calico global network policies. In a similar fashion to how we created the default-app-policy for pods in the previous module which allowed DNS but default denied all other traffic, we'll create a default-node-policy that allows processes running in the host network namespace to connect to each other, but results in default-deny behavior for any other node connections.

```
cat <<EOF| calicoctl apply -f -
---
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: default-node-policy
spec:
  selector: has(kubernetes.io/hostname)
  ingress:
```

```
  - action: Allow
    protocol: TCP
    source:
      nets:
      - 127.0.0.1/32
  - action: Allow
    protocol: UDP
    source:
      nets:
      - 127.0.0.1/32
EOF
```

Create Host Endpoints

Let’s now create the Host Endpoints, allowing Calico to start policy enforcement on node interfaces.

First, verify there no existing Host Endpoints:

```
calicoctl get heps
```

Example output:

```
NAME   NODE
```

Now let’s configure Calico to automatically create Host Endpoints for Kubernetes nodes:

```
calicoctl patch kubecontrollersconfiguration default --patch='{"spec":
{"controllers": {"node": {"hostEndpoint": {"autoCreate": "Enabled"}}}}}'
```

WIthin a few moments you should see Host Endpoints for each of the Kubernetes nodes:

```
calicoctl get heps
```

Example output:

```
NAME                 NODE
node2-auto-hep       node2
control-auto-hep     control
node1-auto-hep       node1
```

Try to access FTP again

Run the nc again from the standalone host (host1):

```
nc -w 3 198.19.0.1 21
```

This time the netcat should fail, and timeout after 3 seconds.

Terrific, we've locked down the ftp daemon to only be accessible from the relevant places.

What about control plane traffic?

You might be wondering why the above network policy does not break the Kubernetes and Calico control planes. After all, they need to make non-local connections in order to function correctly.

The answer is that Calico has a configurable list of “failsafe” ports which take precedence over any policy. These failsafe ports ensure the connections required for the host networked Kubernetes and Calico control planes processes to function are always allowed (assuming your failsafe ports are correctly configured). This means you don’t have to worry about defining policy rules for these. The default failsafe ports also allow SSH traffic so you can always log into your nodes.

If it wasn’t for these failsafe rules then the above policy would actually stop the Calico and Kubernetes control planes from working, and to fix the situation you would need to fix the network policy and then reboot each node so it can regain access to the control plane. So we always recommend ensuring you have the correct failsafe ports configured before you start applying network policies to host endpoints!

Restrict Access to Kubernetes NodePorts

Depending on your security posture, you may want to restrict who can access services from outside the cluster. In this section we’ll lock down all node port access, and selectively allow access to the customer front end.

Lock down node port access

Kube-proxy load balances incoming connections to node ports to the pods backing the corresponding service. This process involves using DNAT (Destination Network Address Translation) to map the connection to the node port to a pod IP address and port. (We’ll dig deeper into Kubernetes services and how NAT works next week’s modules.)

Calico GlobalNetworkPolicy allows you to write policy that is enforced before this translation takes place. i.e. Policy that sees the original node port as the destination, not the backing pod that is being load balanced to as the destination.

```
cat <<EOF | calicoctl apply -f -
---
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: nodeport-policy
spec:
  order: 100
  selector: has(kubernetes.io/hostname)
  applyOnForward: true
  preDNAT: true
  ingress:
  - action: Deny
    protocol: TCP
    destination:
      ports: ["30000:32767"]
  - action: Deny
    protocol: UDP
    destination:
      ports: ["30000:32767"]
EOF
```

Verify you cannot access yaobank frontend

```
curl --connect-timeout 3 198.19.0.1:30180
```

This should fail.

Selectively allow access to customer front end

Let’s update our policy to allow only host1 to access the customer node port:

```
cat <<EOF | calicoctl apply -f -
---
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: nodeport-policy
spec:
  order: 100
  selector: has(kubernetes.io/hostname)
  applyOnForward: true
  preDNAT: true
  ingress:
  - action: Allow
    protocol: TCP
    destination:
      ports: [30180]
    source:
      nets:
      - 198.19.15.254/32
  - action: Deny
    protocol: TCP
    destination:
      ports: ["30000:32767"]
  - action: Deny
    protocol: UDP
    destination:
      ports: ["30000:32767"]
EOF
```

Verify access

Now attempt to access the front end from host1:

```
curl 198.19.0.1:30180
```

This should succeed from host1, but will fail from any other location.

Conclusion

We hope this module illustrated the power of Calico for enabling advanced network security across your Kubernetes cluster, including hostports and host-networked pods, and even for services and nodePorts!