Example 2: Web Server Protection

This example creates a service with the nginx web server. The cSRX will use a forwarding policy and pass the incoming traffic to the back end web server. Finally, we'll scale up the cSRX or nginx web service using the imperative kubectl scale commands.

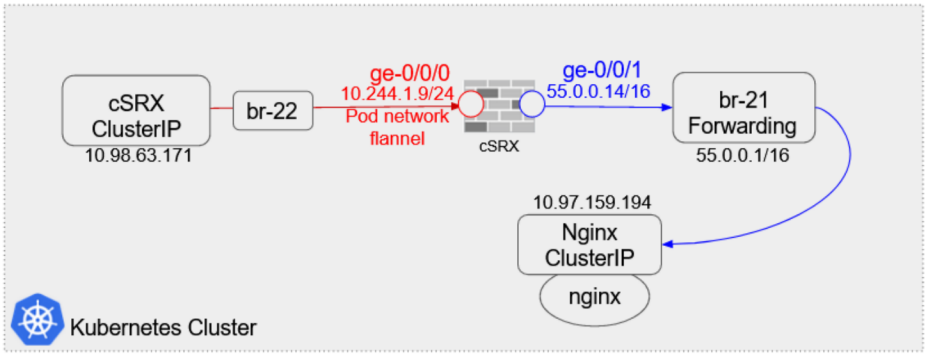

Figure 6.3 *Protect the nginx deployment on Kubernetes*

1. Create the network-attachment-definition with Multus bridges br-21 and br-22:

```
(Controller) # git clone https://github.com/csrx-dayone/k8s-demo.git
(Controller) # cd k8s-demo
(Controller) # kubectl create -f network.yaml
networkattachmentdefinition.k8s.cni.cncf.io/nw2-1 created
networkattachmentdefinition.k8s.cni.cncf.io/nw2-2 created
```

2. Create a web service with image nginx:

```
(Controller) # kubectl create -f run-my-nginx.yaml
deployment.apps/my-nginx created
# kubectl expose deployment/my-nginx
service/my-nginx exposed
```

3. Get the service ClusterIP once backend service is created:

```
(Controller) # kubectl get service
NAME         TYPE        CLUSTER-IP         EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1          <none>        443/TCP   8d
my-nginx     ClusterIP   10.97.159.194      <none>        80/TCP    9h
```

4. Modify configmap.yaml to change csrx baseline configuration to forward traffic to backend service. Replace "10.97.159.194" with your nginx service IP. Then add cSRX license information starting from Line 7. Please use four spaces indentation for each line.

```
# set routing-options static route 10.97.159.194/32 next-hop 55.0.0.1/32
# set security nat destination pool forward-pool address 10.97.159.194/32
```

5. Create the configMap which includes the cSRX license and the cSRX baseline configuration:

```
(Controller)# kubectl create -f configmap.yaml
configmap/csrx-config-map created
```

6. Create csrx deployment and service. Please make sure the environment variable CSRX_MGMT_PORT_REORDER should be set to "yes". This is to bind the last interface as MGMT port.

```
(Controller)# kubectl create -f csrx.yaml
deployment.apps/csrx1 created
service/csrx1 created
```

7. Check the created resource status and verify that the license and configuration are applied on the cSRX. Remember which node has the cSRX located.

```
(Controller)# kubectl get all -o wide
NAME                             READY   STATUS    RESTARTS   AGE    IP             NODE …
pod/csrx1-77d874d5b9-qtqhj       1/1     Running      0       116s   10.244.1.18    worker1
pod/my-nginx-5b56ccd65f-dsn7l    1/1     Running      0       10h    10.244.2.3     worker2

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE …
service/csrx1           ClusterIP   10.98.63.171     <none>        80/TCP     116s
service/kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP    8d
service/my-nginx        ClusterIP   10.97.159.194    <none>        80/TCP     10h

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE …
deployment.apps/csrx1         1/1     1            1             116s
deployment.apps/my-nginx      1/1     1            1             10h

NAME                                        DESIRED   CURRENT   READY    AGE …
replicaset.apps/csrx1-77d874d5b9            1         1         1        116s
replicaset.apps/my-nginx-5b56ccd65f         1         1         1        10h

# kubectl exec -it csrx1-77d874d5b9-qtqhj /bin/bash
root@csrx1-77d874d5b9-qtqhj:/# cli
root@csrx1-77d874d5b9-qtqhj> show system license
(Please make sure the license has been applied successfully on the pod.)
root@csrx1-77d874d5b9-qtqhj> show configuration | display set
set version 20200819.234446.1_builder.r1131461
set interfaces ge-0/0/0 unit 0 family inet address 10.244.1.18/24
set interfaces ge-0/0/1 unit 0 family inet address 55.0.0.16/16
set routing-options static route 10.97.159.194/32 next-hop 55.0.0.1/32
set routing-options static route 0.0.0.0/0 next-hop 10.244.1.1/32
set routing-options static route 10.244.0.0/16 next-hop 10.244.1.1/32
set routing-options static route 10.244.1.0/24 next-hop 0.0.0.0/32
set security nat source rule-set s-forward from zone trust
set security nat source rule-set s-forward to zone untrust
set security nat source rule-set s-forward rule s-forward-rule match source-address 0.0.0.0/0
set security nat source rule-set s-forward rule s-forward-rule then source-nat interface
set security nat destination pool forward-pool address 10.97.159.194/32
set security nat destination pool forward-pool address port 80
set security nat destination rule-set forword from zone trust
set security nat destination rule-set forword rule forward-rule match destination-address 0.0.0.0/0
set security nat destination rule-set forword rule forward-rule match destination-port 80
set security nat destination rule-set forword rule forward-rule then destination-nat pool forward-pool
```

```
set security policies default-policy permit-all
set security zones security-zone trust host-inbound-traffic system-services all
set security zones security-zone trust host-inbound-traffic protocols all
set security zones security-zone trust interfaces ge-0/0/0.0
set security zones security-zone untrust host-inbound-traffic system-services all
set security zones security-zone untrust host-inbound-traffic protocols all
set security zones security-zone untrust interfaces ge-0/0/1.0
root@csrx1-77d874d5b9-qtqhj> exit
root@csrx1-77d874d5b9-qtqhj:/# exit
exit
(controller) #
```

8. Add iptables forward accept rules for Multus-created bridge br-21 on the node where cSRX has been implemented:

```
(Worker1) # iptables -A FORWARD -i br-21 -o br-21 -j ACCEPT
```

9. Config br-21 IP address as cSRX gateway on the node where cSRX is implemented. When br-21 was assigned an IP address, it turns into a gateway for the container cSRX according to Figure 6.3.

```
(Worker1) # ifconfig br-21 55.0.0.1/16
```

Now the setup of the cSRX in front of web server is ready. If the cSRX has license and configuration applied, the NAT function will be working as expected and you can verify it by accessing the ClusterIP of cSRX, displayed in the output of kubectl get all command:

```
(controller) # curl 10.98.63.171
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
…
```

That verifies that the contents of the web server has been forwarded to the port 80 of the cSRX successfully!

Ingress Controller

Here's a little more on the cSRX protecting web server scenario just discussed… when the ClusterIP service of the cSRX has been successfully deployed, access to the cSRX port 80 is still within the cluster. If you are outside the cluster and would like to access the web server, implementing an ingress controller is necessary.

The ingress controller needs a specific namespace, service account, cluster role bindings, configmaps, etc. These can be created by running ingress-roles.yaml in the repository:

```
(Controller) # kubectl create -f ingress-roles.yaml
namespace/ingress-nginx created
configmap/nginx-configuration created
```

```
configmap/tcp-services created
configmap/udp-services created
serviceaccount/nginx-ingress-serviceaccount created
clusterrole.rbac.authorization.k8s.io/nginx-ingress-clusterrole created
role.rbac.authorization.k8s.io/nginx-ingress-role created
rolebinding.rbac.authorization.k8s.io/nginx-ingress-role-nisa-binding created
clusterrolebinding.rbac.authorization.k8s.io/nginx-ingress-clusterrole-nisa-binding created
```

The next step is to create a service of Type NodePort to expose the nginx controller deployment outside the cluster:

```
(Controller) # kubectl create -f ingress-service.yaml
deployment.apps/nginx-ingress-controller created
service/ingress-nginx created
```

And the last step is to create the ingress, which binds the cSRX service with the ingress controller:

```
(Controller) # kubectl create -f ingress-csrx.yaml
ingress.networking.k8s.io/csrx-ingress created
```

To verify the ingress controller has implemented, on the controller node, list all resources created for namespace ingress-nginx. A pod, a service, a deployment and a replicaset has been created:

```
(Controller) # kubectl get all -n ingress-nginx -o wide
NAME                             READY   STATUS    RESTARTS  AGE   IP                NODE
NOMINATED NODE   READINESS GATES
pod/nginx-ingress-controller-…    1/1     Running   0         9h    192.168.189.151   worker2
<none>           <none>

NAME                    TYPE       CLUSTER-IP        EXTERNAL-IP          PORT(S)
AGE    SELECTOR
service/ingress-nginx   NodePort   10.105.19.182     <none>               80:32256/TCP,443:30557/TCP
9h     app.kubernetes.io/name=ingress-…

NAME                                       EADY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS
deployment.apps/nginx-ingress-controller   1/1    1            1           9h    nginx-ingress-
controller  …

NAME                                                 DESIRED   CURRENT   READY   AGE   CONTAINERS
replicaset.apps/nginx-ingress-controller-57c4b94788   1         1         1       9h    nginx-ingress-
controller   …
```

From this information, you can see that the ingress controller pod has been implemented on node worker2. Using a host outside of the cluster, browse on worker2's IP address , and the default nginx page will show up. This is because the service ingress-nginx has forwarded the user's access request to the cSRX service.

For the ingress controller, you can also define rules and configure DNS with the path for redirecting requests to different services. Please refer to Kubernetes official documentation for more details.

Scale Up the cSRX and Web Server

After implementing the web server workload on a Kubernetes cluster as in the topology above, the DevOps team needs to deal with the situation when there are increased number of visitors for the web server contents. If visitors keep increasing, it will cause increased CPU usage and memory consumption for both the cSRX and the web server, which will cause slow response for visitors. So we need to scale up the workloads. It can be implemented simply with two commands:

```
(Controller) # kubectl scale deploy csrx1 --replicas=3
deployment.apps/csrx1 scaled
(Controller) # kubectl scale deploy my-nginx --replicas=3
deployment.apps/my-nginx scaled
```

One extra step is to add some iptables forward accept rules for Multus-created bridge br-21, and config br-21 IP addresses as cSRX gateways for all other nodes when replicas of the cSRX have spun up. Commands can be found in Steps 8 and 9 in this chapter's section, Implementing cSRX in Front of a Webserver.

When those procedures have been completed, the cSRX service and the nginx service will be able to replicate themselves to three copies and distribute visitor hits to all pods averagely.

To see the replicas have been created successfully, run:

```
(Controller) # kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
csrx1-68b79bbb7-7vkmx      1/1    Running   0          4m25s
csrx1-68b79bbb7-hc9wx      1/1    Running   0          11m
csrx1-68b79bbb7-sv8v2      1/1    Running   0          4m25s
my-nginx-5b56ccd65f-92nz6   1/1    Running   0          4m16s
my-nginx-5b56ccd65f-lqq6f   1/1    Running   0          4m16s
my-nginx-5b56ccd65f-wd26k   1/1    Running   0          19m
```

To see that the workload has been distributed to the three pods, it can be verified by opening three windows for each replica of the cSRX. Keep visiting the cSRX pod and observe security flow sessions. Just in case one of the replica pods has been accidently terminated, another replica will start running automatically to keep the number of replica pods to three all the times. This is all handled automatically by Kubernetes deployments.