Table of Contents

* [Networking Lab](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_networking_lab)
* [1. Configure Multitenant Network Plug-in](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#labexercises)
* [2. Add Content to Projects](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_add_content_to_projects)
* [3. Examine Effects of Project Multitenancy](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_examine_effects_of_project_multitenancy)
  + [3.1. Examine Virtual Networks](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_examine_virtual_networks)
  + [3.2. Test Isolation](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_test_isolation)
  + [3.3. Join Projects](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_join_projects)
* [4. Examine Service Layer Implementation](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_examine_service_layer_implementation)
  + [4.1. Set Up Scenario](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_set_up_scenario)
  + [4.2. Examine Round-Robin Services](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_examine_round_robin_services)
  + [4.3. Examine Session Affinity Services](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_examine_session_affinity_services)
* [5. Clean Up Environment](https://www.opentlc.com/labs/ocp_advanced_deployment/03_1_Networking_Solution_Lab.html#_clean_up_environment)

Networking Lab

In this lab, you explore subnet and multitenancy network plug-ins as well as the service layer implementation.

**Goals**

* Configure a multitenant network plug-in
* Add content to projects
* Examine the effects of project multitenancy
* Examine the service layer implementation

1. Configure Multitenant Network Plug-in

1. Enable the multitenant network plug-in in **/etc/ansible/hosts** for the OpenShift cluster.
   1. On the **bastion** host, make sure that the **[OSEv3:vars]** section in **/etc/ansible/hosts** contains the following:

os\_sdn\_network\_plugin\_name='redhat/openshift-ovs-multitenant'

1. You have two choices for how to reconfigure the network plug-in on each node:
   1. More reliable: Run separate ansible commands to change the plug-in and restart the services
   2. *# reconfigure all the config files*
   3. ansible masters -m shell -a "sed -i -e 's/openshift-ovs-subnet/openshift-ovs-multitenant/g' /etc/origin/master/master-config.yaml"
   4. ansible nodes -m shell -a "sed -i -e 's/openshift-ovs-subnet/openshift-ovs-multitenant/g' /etc/origin/node/node-config.yaml"
   5. *# Stop all the services*
   6. ansible masters -m shell -a'systemctl stop atomic-openshift-master-api'
   7. ansible masters -m shell -a'systemctl stop atomic-openshift-master-controllers'
   8. ansible nodes -m shell -a'systemctl stop atomic-openshift-node'
   9. ansible nodes -m shell -a'systemctl stop docker'
   10. *# restart openvswitch*
   11. ansible nodes -m shell -a'systemctl restart openvswitch'
   12. *# start all the services*
   13. ansible nodes -m shell -a'systemctl start docker'
   14. ansible masters -m shell -a'systemctl start atomic-openshift-master-api'
   15. ansible masters -m shell -a'systemctl start atomic-openshift-master-controllers'

ansible nodes -m shell -a'systemctl start atomic-openshift-node'

* 1. Less reliable: Run the **deploy\_cluster** playbook again to reconfigure the network plug-in on each node:

ansible-playbook -f 20 /usr/share/ansible/openshift-ansible/playbooks/deploy\_cluster.yml

* + - This takes about 30 minutes to complete.

1. Examine **netnamespaces** to see the virtual networks assigned to each project and verify that multitenancy is enabled:

oc get netnamespaces

**Sample Output**

NAME NETID EGRESS IPS

default 0 []

kube-public 13796860 []

kube-service-catalog 14031940 []

kube-system 6104895 []

logging 7137948 []

management-infra 3431091 []

openshift 676636 []

openshift-ansible-service-broker 4711691 []

openshift-infra 15765854 []

openshift-metrics 848547 []

openshift-node 14174162 []

openshift-template-service-broker 5826952 []

openshift-web-console 6477194 []

2. Add Content to Projects

In this section, you use RHOCP to deploy applications to the internal network. In later sections, you test network connectivity.

1. Deploy the **hello-openshift** image to the development projects:
2. oc new-project pizzaparty-dev
3. oc new-project fancypants-dev
4. oc new-app openshift/hello-openshift:v1.1.1.1 -n pizzaparty-dev

oc new-app openshift/hello-openshift:v1.1.1.1 -n fancypants-dev

1. After a minute or two, display the state of the **hello-openshift** pods:

oc get pod --all-namespaces | grep hello-openshift

**Sample Output**

fancypants-dev hello-openshift-1-eiwlu 1/1 Running 0 41s

pizzaparty-dev hello-openshift-1-32s49 1/1 Running 0 1m

1. Launch a pod in each of the development projects to create a **bash** session in each pod:
2. oc run shelly -n pizzaparty-dev --image=openshift3/ose-deployer --command=true -- bash -c 'while true; do sleep 1; done'

oc run shelly -n fancypants-dev --image=openshift3/ose-deployer --command=true -- bash -c 'while true; do sleep 1; done'

|  |  |
| --- | --- |
|  | As an aside, are you curious about what **oc run** is? It is a job. See the [documentation](https://docs.openshift.com/container-platform/3.9/dev_guide/jobs.html) to learn more. |

1. Verify that the shell pods are running:

oc get pod --all-namespaces | grep shelly

**Sample Output**

fancypants-dev shelly-1-vnw0b 1/1 Running 0 1m

pizzaparty-dev shelly-1-0khgb 1/1 Running 0 1m

3. Examine Effects of Project Multitenancy

Because you enabled multitenancy, each project is placed on its own Open vSwitch (OVS) virtual network.

3.1. Examine Virtual Networks

1. Examine the **netnamespaces** to see the virtual networks assigned to each project:

oc get netnamespaces

**Sample Output**

NAME NETID EGRESS IPS

default 0 []

fancypants-dev 4745435 []

kube-public 13796860 []

kube-service-catalog 0 []

kube-system 6104895 []

logging 7137948 []

management-infra 3431091 []

openshift 676636 []

openshift-ansible-service-broker 4711691 []

openshift-infra 15765854 []

openshift-metrics 848547 []

openshift-node 14174162 []

openshift-template-service-broker 5826952 []

openshift-web-console 6477194 []

pizzaparty-dev 9652093 []

3.2. Test Isolation

In this section, you test the **hello-openshift** pods to see if they are isolated from one another.

1. Use either of the following methods to determine the pods' IP addresses:
   * Method 1: Discover the pods' IP addresses and note them.
     + Step 1

oc get pod -n pizzaparty-dev

**Sample Output**

NAME READY STATUS RESTARTS AGE

hello-openshift-1-6z7bb 1/1 Running 0 2m

shelly-1-sxxz4 1/1 Running 0 1m

* + - Step 2

oc describe pod -n pizzaparty-dev hello-openshift-1-6z7bb | egrep 'IP|Node:'

**Sample Output**

Node: node2.GUID.internal/192.199.0.153

IP: 10.1.10.7

* + - Step 3

oc get pod -n fancypants-dev

**Sample Output**

NAME READY STATUS RESTARTS AGE

hello-openshift-1-62x4w 1/1 Running 0 3m

shelly-1-5cxtm 1/1 Running 0 2m

* + - Step 4

oc describe pod -n fancypants-dev hello-openshift-1-62x4w | egrep 'IP|Node:'

**Sample Output**

Node: node2.GUID.internal/192.199.0.153

IP: 10.1.10.8

* + Method 2: Determine the pods' IP addresses with these commands:
    - Pod 1:

oc get pod -n pizzaparty-dev $(oc get pod -n pizzaparty-dev | grep hello-openshift | awk '{print $1}') -o template --template '{{.status.podIP}}{{"\n"}}'

**Sample Output**

10.1.10.7

* + - Pod 2:

oc get pod -n fancypants-dev $(oc get pod -n fancypants-dev | grep hello-openshift | awk '{print $1}') -o template --template '{{.status.podIP}}{{"\n"}}'

**Sample Output**

10.1.10.8

1. Invoke a shell session using **oc rsh** for the **shelly** pod in the **fancypants-dev** project:

oc rsh -n fancypants-dev $(oc get pod -n fancypants-dev | grep shelly | awk '{print $1}')

1. Once inside the container, try to access the **hello-openshift** pod in the **pizzaparty-dev** project (using the IP address you previously retrieved):
2. sh-4.2$ curl 10.1.10.7:8080 -m 1

curl: (28) Connection timed out after 1000 milliseconds

1. Contact the **hello-openshift** pod that belongs to the **fancypants-dev** project (again using the IP address you previously retrieved):
2. sh-4.2$ curl 10.1.10.8:8080 -m 1

Hello OpenShift!

* + This indicates that the pods running in the **fancypants-dev** project do not have a network path to the pods running in the **pizzaparty-dev** project, but can still contact the other pods in their own project.

1. Exit out of the **hello-openshift** pod and connect to the **shelly** pod in the **pizzaparty-dev** project:

oc rsh -n pizzaparty-dev $(oc get pod -n pizzaparty-dev | grep shelly | awk '{print $1}') "/bin/bash"

1. Try again to access the **hello-openshift** pod in the **pizzaparty-dev** project:
2. /usr/bin/id: cannot find name for user ID 1000130000
3. [I have no name!@shelly-1-sxxz4 origin]$ curl 10.1.10.7:8080 -m 1

Hello OpenShift!

* + As expected, the **shelly** pod in the **pizzaparty-dev** project has network access to other pods in the same project.

1. Exit from the pod.

3.3. Join Projects

The **oc adm pod-network** command provides a way to join projects, putting them on the same Open vSwitch virtual network.

1. Join the **pizzaparty-dev** and **fancypants-dev** projects:

oc adm pod-network join-projects --to=fancypants-dev pizzaparty-dev

1. Inspect the results and note that both of the **-dev** projects have the same **NETID**:

oc get netnamespaces | grep dev

**Sample Output**

fancypants-dev 4745435 []

pizzaparty-dev 4745435 []

1. Verify that the join worked by opening a remote shell in the **fancypants-dev** project’s **shelly** pod again:

oc rsh -n fancypants-dev $(oc get pod -n fancypants-dev | grep shelly | awk '{print $1}')

1. Attempt to access the two **hello-openshift** pods:
2. sh-4.2$ curl 10.1.10.7:8080 -m 1
3. Hello OpenShift!
4. *# test connection to the pod from the other project*
5. sh-4.2$ curl 10.1.10.8:8080 -m 1

Hello OpenShift!

* + Note that you are now able to access both of the **hello-openshift** pods.

1. Exit from the pod.

4. Examine Service Layer Implementation

4.1. Set Up Scenario

1. Create a project to house the services you want to examine:

oc new-project servicelayer

1. Use **oc new-app** to deploy the **hello-openshift** image:

oc new-app openshift/hello-openshift

1. Wait for the pod to enter the **Running** state.
2. Scale the application to four instances:

oc scale dc/hello-openshift --replicas=4

1. Wait for the three additional pods to enter the **Running** state.

4.2. Examine Round-Robin Services

By default, **oc new-app** creates round-robin services.

1. Describe the service:

oc describe service hello-openshift

**Sample Output**

Name: hello-openshift

Namespace: servicelayer

Labels: app=hello-openshift

Annotations: openshift.io/generated-by=OpenShiftNewApp

Selector: app=hello-openshift,deploymentconfig=hello-openshift

Type: ClusterIP

IP: 172.30.162.40

Port: 8080-tcp 8080/TCP

TargetPort: 8080/TCP

Endpoints: 10.1.10.10:8080,10.1.4.11:8080,10.1.8.14:8080 + 1 more...

Port: 8888-tcp 8888/TCP

TargetPort: 8888/TCP

Endpoints: 10.1.10.10:8888,10.1.4.11:8888,10.1.8.14:8888 + 1 more...

Session Affinity: None

Events: <none>

* + You are likely to have a service IP address that is different from **172.30.162.40** shown in the example.

1. Because the bastion is not part of the pod network (it is not a node), use SSH to access one of the masters and switch to **root**:
2. ssh master1.GUID.internal
3. sudo -i

oc project servicelayer

1. Examine the port 8080 rules for the service:

iptables-save | grep servicelayer/hello-openshift | grep 8080

**Sample Output**

-A KUBE-SEP-5ZI5JU6AOMYX4VMW -s 10.1.8.15/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-5ZI5JU6AOMYX4VMW -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m tcp -j DNAT --to-destination 10.1.8.15:8080

-A KUBE-SEP-62S6ODWP6EU53XGI -s 10.1.8.14/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-62S6ODWP6EU53XGI -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m tcp -j DNAT --to-destination 10.1.8.14:8080

-A KUBE-SEP-U5I424KT5DGH54WY -s 10.1.10.10/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-U5I424KT5DGH54WY -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m tcp -j DNAT --to-destination 10.1.10.10:8080

-A KUBE-SEP-YJ3IYHPR22CWJDUT -s 10.1.4.11/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-YJ3IYHPR22CWJDUT -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m tcp -j DNAT --to-destination 10.1.4.11:8080

-A KUBE-SERVICES -d 172.30.162.40/32 -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp cluster IP" -m tcp --dport 8080 -j KUBE-SVC-BXIL3WE3EN2EVZ7N

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-U5I424KT5DGH54WY

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-YJ3IYHPR22CWJDUT

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-62S6ODWP6EU53XGI

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-SEP-5ZI5JU6AOMYX4VMW

1. Note that the first **iptables** rule in the service path is the destination-matching rule:

-A KUBE-SERVICES -d 172.30.162.40/32 -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp cluster IP" -m tcp --dport 8080 -j KUBE-SVC-BXIL3WE3EN2EVZ7N

* + This rule jumps to another **iptables** rule chain, **KUBE-SVC-BXIL3WE3EN2EVZ7N**.

1. Examine the second stop in the service path:
2. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-U5I424KT5DGH54WY
3. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-YJ3IYHPR22CWJDUT
4. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-62S6ODWP6EU53XGI

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-SEP-5ZI5JU6AOMYX4VMW

* + This is a chain with a series of probability rules. The packet is randomly passed to one of the subsequent chains.

1. Examine the next stop in the service path:
2. -A KUBE-SEP-U5I424KT5DGH54WY -s 10.1.10.10/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-U5I424KT5DGH54WY -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m tcp -j DNAT --to-destination 10.1.10.10:8080

* + This is a chain with the actual DNAT rule that sends the packet to one of the pods in the service.

1. Examine the rule chains to see that all of the pod destinations are listed:
   * 10.1.8.15
   * 10.1.8.14
   * 10.1.10.10
   * 10.1.4.11

The class C subnets are indicative of the nodes that the pods have landed on—in this case, **.2** and **.3**.

4.3. Examine Session Affinity Services

1. Modify the service using **oc edit**:

oc edit service hello-openshift

1. Change **sessionAffinity** to **ClientIP**:

sessionAffinity: ClientIP

1. Save and exit.
2. Describe the service:

oc describe service hello-openshift

**Sample Output**

Name: hello-openshift

Namespace: servicelayer

Labels: app=hello-openshift

Annotations: openshift.io/generated-by=OpenShiftNewApp

Selector: app=hello-openshift,deploymentconfig=hello-openshift

Type: ClusterIP

IP: 172.30.162.40

Port: 8080-tcp 8080/TCP

TargetPort: 8080/TCP

Endpoints: 10.1.10.10:8080,10.1.4.11:8080,10.1.8.14:8080 + 1 more...

Port: 8888-tcp 8888/TCP

TargetPort: 8888/TCP

Endpoints: 10.1.10.10:8888,10.1.4.11:8888,10.1.8.14:8888 + 1 more...

Session Affinity: ClientIP

Events: <none>

* + Note that **sessionAffinity** changed to **ClientIP**.

1. Examine the port 8080 rules for the service:

iptables-save | grep servicelayer/hello-openshift | grep 8080

**Sample Output**

-A KUBE-SEP-5ZI5JU6AOMYX4VMW -s 10.1.8.15/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-5ZI5JU6AOMYX4VMW -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --set --name KUBE-SEP-5ZI5JU6AOMYX4VMW --mask 255.255.255.255 --rsource -m tcp -j DNAT --to-destination 10.1.8.15:8080

-A KUBE-SEP-62S6ODWP6EU53XGI -s 10.1.8.14/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-62S6ODWP6EU53XGI -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --set --name KUBE-SEP-62S6ODWP6EU53XGI --mask 255.255.255.255 --rsource -m tcp -j DNAT --to-destination 10.1.8.14:8080

-A KUBE-SEP-U5I424KT5DGH54WY -s 10.1.10.10/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-U5I424KT5DGH54WY -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --set --name KUBE-SEP-U5I424KT5DGH54WY --mask 255.255.255.255 --rsource -m tcp -j DNAT --to-destination 10.1.10.10:8080

-A KUBE-SEP-YJ3IYHPR22CWJDUT -s 10.1.4.11/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-YJ3IYHPR22CWJDUT -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --set --name KUBE-SEP-YJ3IYHPR22CWJDUT --mask 255.255.255.255 --rsource -m tcp -j DNAT --to-destination 10.1.4.11:8080

-A KUBE-SERVICES -d 172.30.162.40/32 -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp cluster IP" -m tcp --dport 8080 -j KUBE-SVC-BXIL3WE3EN2EVZ7N

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-U5I424KT5DGH54WY --mask 255.255.255.255 --rsource -j KUBE-SEP-U5I424KT5DGH54WY

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-YJ3IYHPR22CWJDUT --mask 255.255.255.255 --rsource -j KUBE-SEP-YJ3IYHPR22CWJDUT

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-62S6ODWP6EU53XGI --mask 255.255.255.255 --rsource -j KUBE-SEP-62S6ODWP6EU53XGI

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-5ZI5JU6AOMYX4VMW --mask 255.255.255.255 --rsource -j KUBE-SEP-5ZI5JU6AOMYX4VMW

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-U5I424KT5DGH54WY

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-YJ3IYHPR22CWJDUT

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-62S6ODWP6EU53XGI

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-SEP-5ZI5JU6AOMYX4VMW

* + Note that the **iptables** rules have changed.

1. Examine the first rule in the service path:

-A KUBE-SERVICES -d 172.30.162.40/32 -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp cluster IP" -m tcp --dport 8080 -j KUBE-SVC-BXIL3WE3EN2EVZ7N

* + Note that this rule did not change and still jumps to a new chain if the packet is destined for the service.

1. Examine the second stop in the service path:
2. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-U5I424KT5DGH54WY --mask 255.255.255.255 --rsource -j KUBE-SEP-U5I424KT5DGH54WY
3. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-YJ3IYHPR22CWJDUT --mask 255.255.255.255 --rsource -j KUBE-SEP-YJ3IYHPR22CWJDUT
4. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-62S6ODWP6EU53XGI --mask 255.255.255.255 --rsource -j KUBE-SEP-62S6ODWP6EU53XGI
5. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --rcheck --seconds 10800 --reap --name KUBE-SEP-5ZI5JU6AOMYX4VMW --mask 255.255.255.255 --rsource -j KUBE-SEP-5ZI5JU6AOMYX4VMW
6. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-U5I424KT5DGH54WY
7. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-YJ3IYHPR22CWJDUT
8. -A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-62S6ODWP6EU53XGI

-A KUBE-SVC-BXIL3WE3EN2EVZ7N -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-SEP-5ZI5JU6AOMYX4VMW

* + This is similar to the previous output—with the exception of the reap/recheck options used for affinity. The result is that a packet is sent to the same destination chain as the previous packet, as long as no more than 180 seconds have elapsed since the last packet. Otherwise, the packet is sent to a random chain.

1. Examine this output to see that the last chain contains the actual DNAT rules:
2. -A KUBE-SEP-YJ3IYHPR22CWJDUT -s 10.1.4.11/32 -m comment --comment "servicelayer/hello-openshift:8080-tcp" -j KUBE-MARK-MASQ

-A KUBE-SEP-YJ3IYHPR22CWJDUT -p tcp -m comment --comment "servicelayer/hello-openshift:8080-tcp" -m recent --set --name KUBE-SEP-YJ3IYHPR22CWJDUT --mask 255.255.255.255 --rsource -m tcp -j DNAT --to-destination 10.1.4.11:8080

* + The IP addresses of the pods have not changed and are still reflected here in the rules.

5. Clean Up Environment

1. Delete the **fancypants-dev**, **pizzaparty-dev**, and **servicelayer** projects:
2. oc delete project fancypants-dev
3. oc delete project pizzaparty-dev

oc delete project servicelayer

1. Disconnect from the master.

Build Version: c3147ce9f77191e30b447cc423f2f68a0c40fc03 : Last updated 2018-07-31 01:29:02 EDT