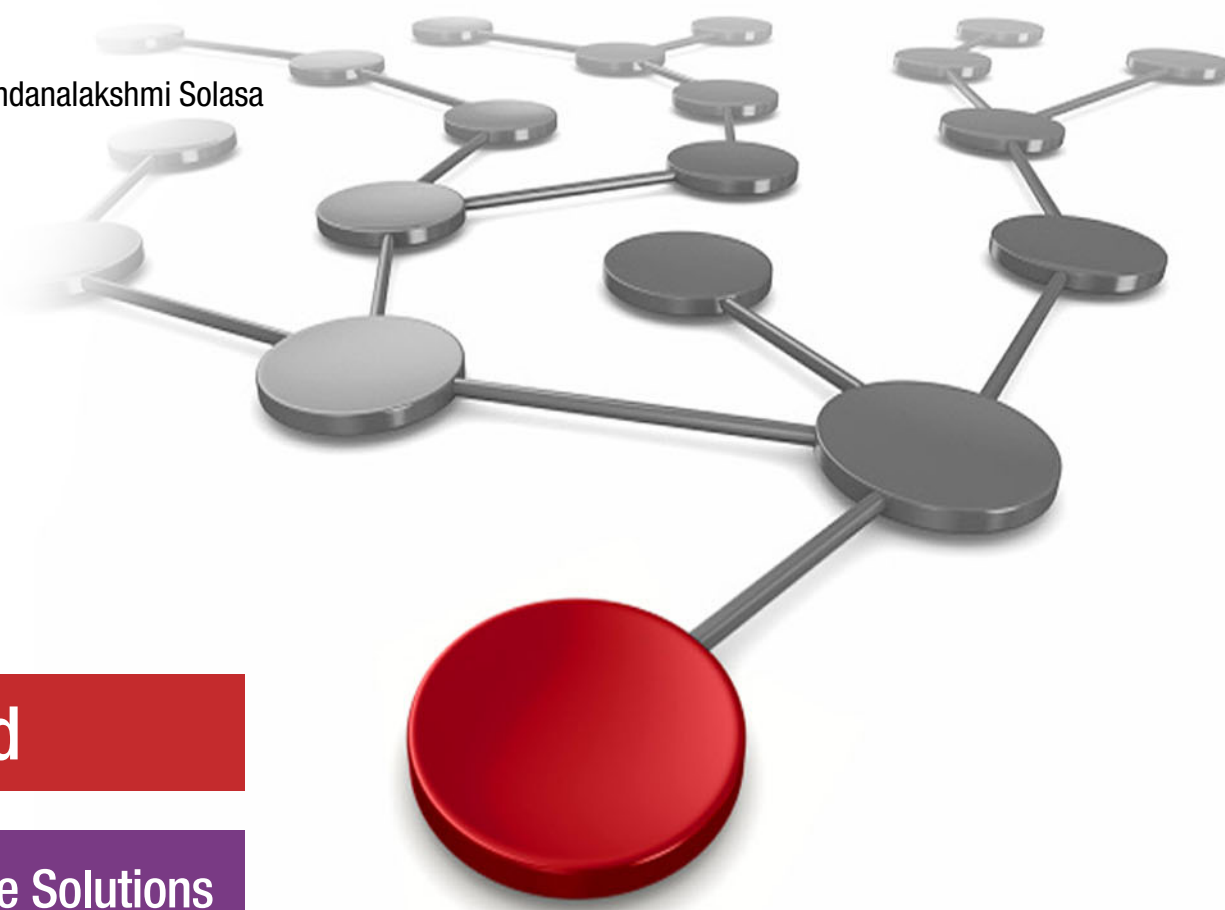


An Implementation of Red Hat OpenShift Network Isolation Using Multiple Ingress Controllers

Loïc Fura

Joyce Mak

Veera Santoshi Chandanalakshmi Solasa



 Cloud

Infrastructure Solutions



An Implementation of Red Hat OpenShift Network Isolation Using Multiple Ingress Controllers

This IBM® Redpaper describes network isolation on a multitenant Red Hat OpenShift cluster.

This publication describes the following topics:

- ▶ Introduction
- ▶ Problem statement
- ▶ Solution
- ▶ Cluster network configuration
- ▶ Ingress controller configuration
- ▶ Load balancer configuration
- ▶ Red Hat OpenShift network policy
- ▶ Application deployment

Introduction

Red Hat OpenShift is a great platform for developing, testing, and running applications. It handles multitenancy within Red Hat OpenShift Cluster by using users and namespaces, which allows it to run different production applications and workloads on the same Red Hat OpenShift Cluster.

From a network point of view, a default Red Hat OpenShift Cluster implementation has three networks, one load balancer, and one DNS domain:

- ▶ One external or physical network: This network is used to build the Red Hat OpenShift Cluster (for installation and initial setup, run **coreOS netboot**). This network is used to connect to each of the Red Hat OpenShift nodes (masters and workers) from outside of the Red Hat OpenShift Cluster. It is also the network on top of which Red Hat OpenShift defines the cluster network and the service network, which are two software-defined networks (SDNs) (overlaid on top of the external or physical network).
- ▶ One internal cluster network: This network is available or reachable only within the Red Hat OpenShift Cluster, and Red Hat OpenShift manages it (IP address management (IPAM) and access control lists (ACLs)).
- ▶ One internal service network: This network is available or reachable only within the Red Hat OpenShift Cluster, and Red Hat OpenShift manages it (IPAM and ACLs).
- ▶ One load balancer: This item is a node or appliance that has a network interface that is configured on the external or physical network. It must be configured to send incoming traffic to the different ingress controllers of the Red Hat OpenShift Cluster.
- ▶ One DNS domain: This item is managed by Red Hat OpenShift Cluster. A wildcard DNS entry must be added by a network administrator, and the wildcard entry must resolve to the IP address of the load balancer (for example, *.myocp.my.domain.com).

When a Pod is created in Red Hat OpenShift, it receives an IP address from the Cluster network that is not static, which means that if the Pod is shut down and then restarted, the IP address can change.

This situation is where the concept of services is useful. When you create a service, you bind it to a Pod, and this service receives a fixed IP address on the service network. When you connect to that IP address on the service network, you are automatically connected to the Pod to which the service is bound. This service is what is being used by other applications running within Red Hat OpenShift (in other Pods) to connect between them. For example, a Pod running a web server connects to another Pod running a database by using the service that is bound to the database Pod. The service also allows a single IP to potentially reach a Pod that has several replicas, which makes the application highly available (Figure 1 on page 3).

From within the Red Hat OpenShift Cluster, you can reach Pods from either the cluster network or the service network if a service is bound to the Pod.

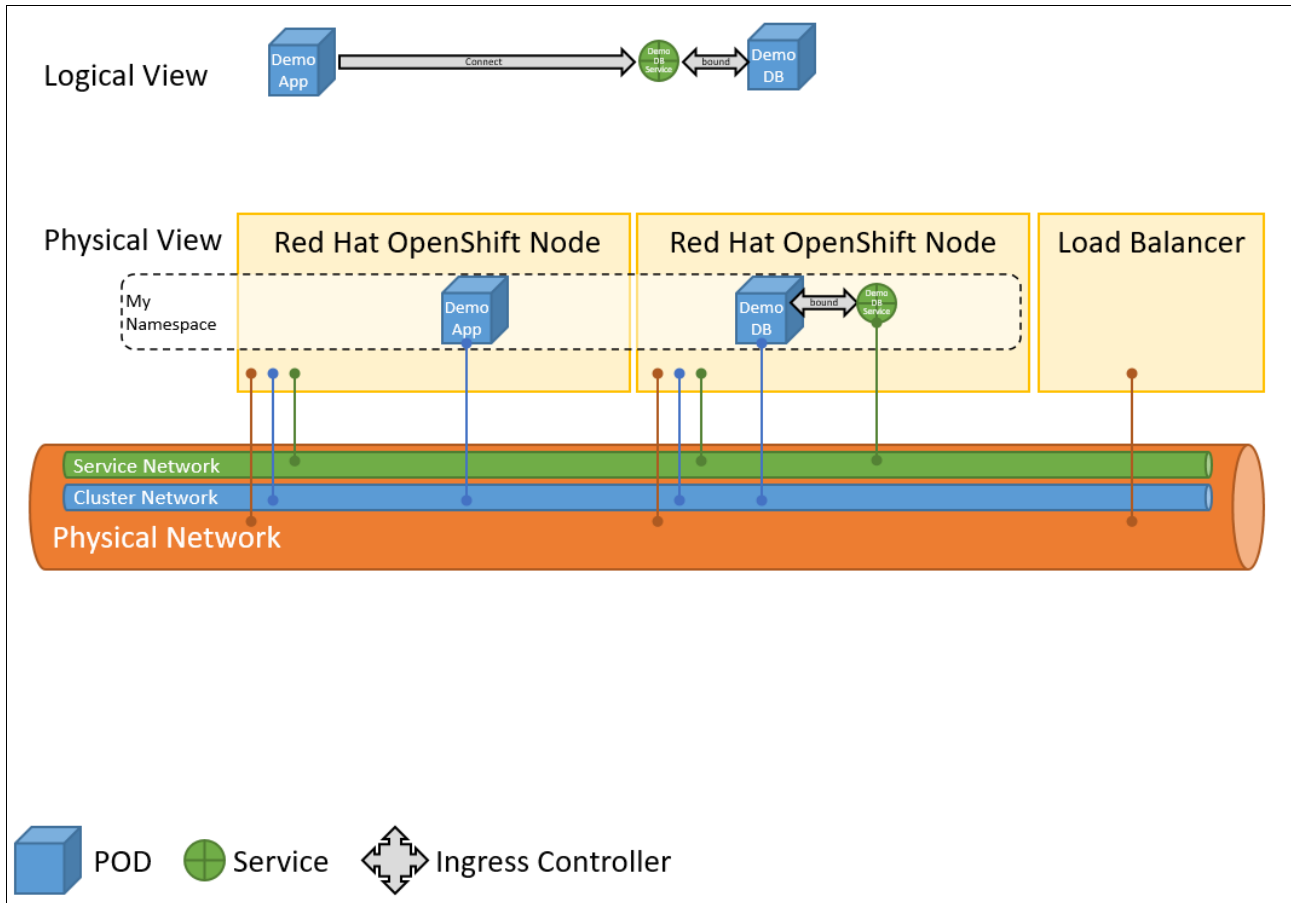


Figure 1 Red Hat OpenShift Cluster diagram: Internal access only

The cluster and service networks from Red Hat OpenShift are software-defined. Red Hat OpenShift uses network policies to define how Pods and services can communicate between them.

By default, all Pods and services can talk to each other within Red Hat OpenShift. To isolate the Pods from different projects, you can define network policies as a part of your network configuration, which is described in “Red Hat OpenShift network policy” on page 11.

So, you now know how to isolate Pods and services from each other on the Red Hat OpenShift internal networks (cluster and service), but none of the Pods and services that you create can be reached from outside the Red Hat OpenShift Cluster. If you take an application running on a bare metal system or a virtual machine (VM) (or any host outside of the Red Hat OpenShift Cluster), this application does not have access to the internal cluster or service network from the Red Hat OpenShift Cluster.

In order for anything external to the Red Hat OpenShift Cluster to communicate with a Pod, the service that is bound to that Pod must be exposed.

With the default implementation of a Red Hat OpenShift Cluster, exposing a service is done by using a *route*, which is a URL that is tied to the DNS domain that is managed by the Red Hat OpenShift Cluster, for example, `demo.apps.mydomain.com`.

A route is managed by an [ingress controller](#).

An ingress controller is a Pod running within Red Hat OpenShift that provides a way to access Pods and services from outside the Red Hat OpenShift Cluster. For example, it can provide a way for an application running on a bare metal system, VM, or external host to reach a Pod or service in the Red Hat OpenShift Cluster.

Typically, ingress controllers are running on Red Hat OpenShift Workers nodes and listen on TCP ports of the external or physical network IP address of the worker node.

The load balancer points to the workers nodes IP and ports to which the ingress controller is listening.

When a route is created to expose a service, the ingress controller now listens for an incoming connection on that specific route (`demo.apps.mydomain.com`) and redirects it to the corresponding service.

In the end, when an application running on a bare metal system, VM, or external host tries to reach the URL of that route, for example, `demo.apps.mydomain.com`, it points first to the load balancer (due to the wildcard DNS entry). Then, the load balancer points to the ingress controllers, and the ingress controllers parse the URL (check the hostname / registered name, which in this example is `demo`) and redirects to the appropriate service within Red Hat OpenShift.

The application running in a Pod in Red Hat OpenShift is now accessible from outside the Red Hat OpenShift Cluster, as shown in Figure 2.

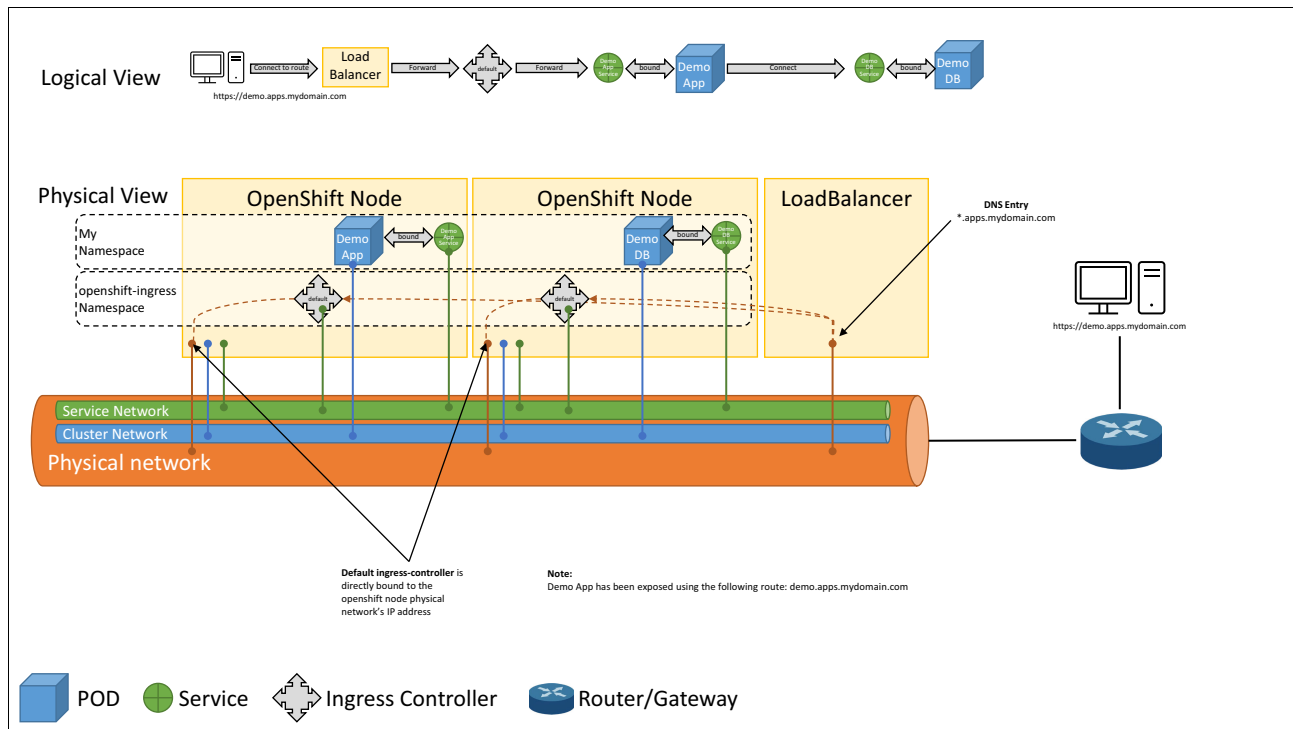


Figure 2 Red Hat OpenShift Cluster diagram: External access

In summary, you must make sure that any bare metal system, VM, or external host that needs to access Pod or service can reach the IP address of the load balancer.

Problem statement

In “Introduction” on page 2, we indicated that we use a network policy on the Red Hat OpenShift internal networks that prevents Pods and services from a namespace within Red Hat OpenShift to communicate with Pods and services from another namespace, thus guaranteeing network isolation within Red Hat OpenShift.

But now that our Pod and service is exposed by using a route, any bare metal system, VM, or external host outside of the Red Hat OpenShift Cluster that has access to the load balancer IP address can potentially access our exposed Pod and service if they know the URL.

In a production environment, for security reasons, you want to ensure that not all bare metal systems, VMs, or external hosts in the data center can access the exposed Pod or services from the Red Hat OpenShift Cluster. Instead, it is a best practice to allow only certain bare metal systems, VMs, and external hosts to access specific exposed Pods or services. But how do you achieve this goal?

Many production environments rely on the concept of virtual local area networks (VLANs) to isolate different components of the production environment on different networks.

VLANs are widely available on enterprise-grade switches and provide a simple and secure way to ensure that two components cannot communicate between each other by isolating them on different Ethernet broadcast domain (OSI Layer 2).

This network option is one of several preferred network security methods in many data centers for bare metal servers and VMs because it offers low-level network isolation.

How can we combine VLANs isolation, which is widely used inside typical production environments (by using bare metal systems and VMs), with the network policies that are offered by Red Hat OpenShift SDNs and the mechanism to expose Pods and services outside of the Red Hat OpenShift Cluster to maintain full network isolation inside and outside of Red Hat OpenShift?

Solution

The solution that is described in this section tries to isolate each project in the following way:

- ▶ Red Hat OpenShift projects are isolated from each other inside the Red Hat OpenShift Cluster by using network policies.
- ▶ An ingress controller is created for each project to allow a specific project to expose services.
- ▶ Each project is allocated a specific VLAN so that services that belong to a different project are exposed through a different VLAN.
- ▶ There is only one load balancer, and the load balancer is listening to the different VLANs.
- ▶ When request comes to the load balancer on a VLAN, the load balancer forwards the request to the corresponding ingress controller.

Figure 3 shows the architecture summary of the solution.

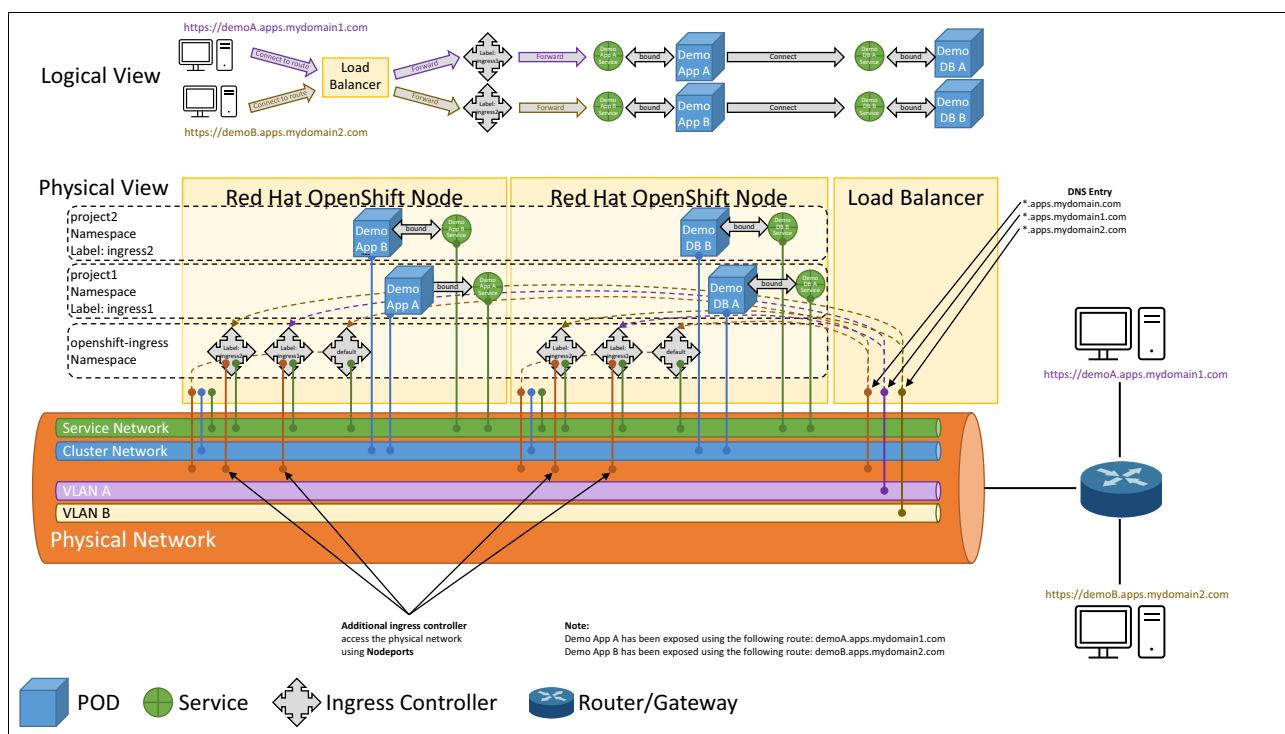


Figure 3 Architectural solution summary

The following sections describe implementation details for the solution.

Cluster network configuration

In a typical Red Hat OpenShift Cluster environment, all the nodes have a single external or physical network interface. For multitenancy, we must separate each tenant's ingress traffic into its own isolated network. To achieve such isolation without adding more servers, we add a physical Ethernet adapter to our Bastion or load balancer node. This adapter is connected to a VLAN-enabled 8021.q standard switch. All VLANs that we plan to use for our network isolation are tagged or trunked to the connected switch port. Instead of tagging all VLANs to the switch port at initial setup, VLANs can be tagged to the switch port as more isolated networks are needed. For each VLAN, the network administrator also must create a DNS wildcard entry that resolves to an IP address within that VLAN.

The primary or admin network interface is configured with the Bastion IP address and used as the front end of the Red Hat OpenShift API and the cluster's default ingress route. For our cluster, the backing device for this interface is virtual Ethernet. For the physical device, VLAN interfaces are created and configured with the IP addresses from the network administrator. Instead of creating all VLAN interfaces during initial setup, create VLAN interfaces when you add tenants to the Red Hat OpenShift cluster. Each tenant is given its own DNS wildcard entry (subdomain), which corresponds to the IP address of the VLAN interface. This entry or domain is what you use to expose its Pods and services.

The following steps show an example about how to create a VLAN interface on the Bastion node. There are multiple methods that you can use to configure VLAN interfaces, and depending on which method or methods that you choose, packages like NetworkManager-config-routing-rules might need to be installed.

For our example, we use the **NetworkManager nmcli** command to create the VLAN interface and configure its IP information. Then, we change to use the traditional route and rules files to implement policy routing.

Complete the following steps:

1. Determine the device name:

```
[root@bastion ~]# nmcli connection show
NAME                UUID                                TYPE      DEVICE
System env32        4e8d61b2-0043-00e9-937c-a0835d16ea18  ethernet  env32
System enP545p80s0f0 042b6f0a-9971-e4b7-6afd-171a0ad62ada  ethernet  --
Wired connection 1   f9a8b089-11e8-3845-a5a3-e9a15c5d4c0d  ethernet  --
```

2. Create a VLAN interface on a tagged device (the tagged device is enP545p80s0f0; our VLAN is 1285, and we pick the interface name vlan1285):

```
[root@bastion ~]# nmcli connection add type vlan con-name vlan1285 ifname
vlan1285 vlan.parent enP545p80s0f0 vlan.id 1285
Connection 'vlan1285' (d5cce68d-c003-4ee4-b801-5ab521739bbd) successfully
added.
```

3. Add IP information to the VLAN interface:

```
[root@bastion ~]# nmcli connection modify vlan1285 ipv4.address 129.40.94.9/29
[root@bastion ~]# nmcli connection modify vlan1285 ipv4.method manual
```

```
[root@bastion ~]# nmcli connection show
NAME                UUID                                TYPE      DEVICE
System env32        4e8d61b2-0043-00e9-937c-a0835d16ea18  ethernet  env32
vlan1285            d5cce68d-c003-4ee4-b801-5ab521739bbd  vlan       vlan1285
System enP545p80s0f0 042b6f0a-9971-e4b7-6afd-171a0ad62ada  ethernet  --
Wired connection 1   f9a8b089-11e8-3845-a5a3-e9a15c5d4c0d  ethernet  --
```

4. Remove the default route from the VLAN interface:

```
[root@bastion ~]# sed -i 's/DEFROUTE=yes/DEFROUTE=no/'
/etc/sysconfig/network-scripts/ifcfg-vlan1285
```

5. Create route and rule policy files for the VLAN interface:

```
[root@bastion ~]# echo "default via 129.40.94.14 dev vlan1285 table 1285" >
/etc/sysconfig/network-scripts/route-vlan1285
```

```
[root@bastion ~]# echo "from 129.40.94.9 table 1285
to 129.40.94.9 table 1295" > /etc/sysconfig/network-scripts/rule-vlan1285
```

6. Apply the policy files to the VLAN interface:

```
[root@bastion ~]# nmcli connection reload
[root@bastion ~]# nmcli connection up vlan1285
```

The new VLAN interface looks like the following output:

```
[root@bastion ~]# ip address show dev vlan1285
11: vlan1285@enP577p80s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP group default qlen 1000
    link/ether 40:f2:e9:31:5c:d0 brd ff:ff:ff:ff:ff:ff
    inet 129.40.94.9/29 brd 129.40.94.15 scope global noprefixroute vlan1285
```

```
[root@bastion ~]# ip route show table 1285
default via 129.40.94.14 dev vlan1285
```

```
[root@bastion ~]# ip rule show table 1285
```

32727: from all to 129.40.94.9 lookup VLAN1285
32730: from 129.40.94.9 lookup VLAN1285

Now that we have a VLAN interface that is configured on one of our isolated networks, we now describe the Red Hat OpenShift ingress controller configuration.

Ingress controller configuration

Red Hat OpenShift Container Platform allows external application access by using an ingress controller, which can be further configured by using one of the following methods:

- ▶ Load balancer
- ▶ Nodeport
- ▶ External IP address

To achieve network isolation within our multitenant Red Hat OpenShift cluster, we configure one ingress controller per namespace that is dedicated to a tenant, and every ingress controller uses the service type nodeport. Each tenant also has a subdomain and must use this subdomain when deploying web applications so that the tenant can be accessed externally.

Here is the environment information that we use in our example:

- ▶ Cluster ID: ocp4
- ▶ Domain: example.ihost.com
- ▶ DNS wildcard entry: *.apps.ocp4.example.ihost.com
- ▶ Subdomain: apps.ocp4.example.ihost.com
- ▶ Ingress controller label: ingress1
- ▶ Project name: demoproject

As a part of the implementation, we dedicated one project to each tenant. Every project is labeled, which is then used as a part of an ingress controller and network policies configuration, as shown in Example 1 and Example 2 on page 9.

Example 1 shows a sample YAML file that is used to create the tenant's ingress controller. In Example 1, the tenant has a subdomain that is called apps.ocp4.example.ihost.com.

Example 1 *YAML file to create an ingress controller*

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: ingress1
spec:
  endpointPublishingStrategy:
    type: NodePortService
  domain: apps.ocp4.example.ihost.com
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: ingress1
  replicas: 1
```

Using the YAML definition in Example 1 on page 8, we create an ingress controller Pod in the Red Hat OpenShift ingress namespace by running the following command:

```
[root@bastion ~]# oc create -f ingress-controller.yaml
```

Example 2 shows snapshots of the Pods and ingress controllers that are created by using the YAML definition.

Example 2 Pods and ingress controllers that are created

```
[root@bastion ~]# oc get pods -n openshift-ingress
NAME                                READY  STATUS   RESTARTS  AGE
router-ingress1-68dd69c965-8rjtt    1/1    Running  0         5d21h
router-default-74b7fd9887-9ldq7     1/1    Running  0         5d4h
router-default-74b7fd9887-rhhv8     1/1    Running  0         5d4h
```

```
[root@bastion ~]# oc get svc -n openshift-ingress
NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)
AGE
router-internal-default             ClusterIP     172.30.187.184 <none>
80/TCP,443/TCP,1936/TCP            6d23h
router-nodeport-ingress1           NodePort     172.30.198.7   <none>
80:30892/TCP,443:31722/TCP         5d21h
```

```
[root@mtocp-prod-lb1 ~]# oc get ingresscontrollers -n openshift-ingress-operator
NAME      AGE
ingress1  4d1h
default   6d23h
```

By default, Red Hat OpenShift uses ports 30000 - 32767 when nodeport is used as a service to configure applications. This port range can be increased for large clusters. As shown in Example 2, ports 30892 and 31722 are allocated to service with the name router-nodeport-ingress1. These ports play a major role for the network traffic routing and load balancer configuration that is explained in “Load balancer configuration” on page 9.

Example 2 adds a label to the ingress controller Pod, and this same label, ingress1, must be added to the namespace that is created for the tenant by running the command that is shown in Example 3.

Example 3 Add label to namespace to match the ingress controller

```
[root@bastion ~]# oc label namespace demoproject ingress1
```

After we deploy the ingress controller and label the namespace with the same label on the ingress controller, network traffic is enabled between the ingress controller Pod and applications that are deployed in the namespace, which in this case is demoproject.

Load balancer configuration

Depending on your setup, this section might not apply directly.

In our setup, we use HAProxy as the main front-end load balancer for Red Hat OpenShift. If you use a different load balancer, you need to adapt the following configuration to your load balancer.

HAProxy is an open source software for load balancing TCP/HTTP requests. It uses a configuration file, `/etc/haproxy/haproxy.cfg`, to determine how traffic is divided across multiple nodes. There are multiple sections in the configuration file, but for our network isolation description, we look at only the front-end and back-end sections. Front-end rules describe the type of incoming traffic that HAProxy monitors, and back-end rules describe how HAProxy load balances the network traffic. There are usually multiple front-end and back-end rules in a configuration file.

Example 4 shows a segment from the `/etc/haproxy/haproxy.cfg` file on the Bastion node that is configured for the default Red Hat OpenShift setup. It shows the front-end and back-end rules for the HTTP and HTTPS traffic.

Example 4 Bastion node /etc/haproxy/haproxy.cfg segment

```
frontend ingress-http
  bind *:80
  default_backend ingress-http
  option tcplog

backend ingress-http
  balance source
  server worker1-http-router0 129.40.242.154:80 check
  server worker2-http-router1 129.40.242.155:80 check

frontend ingress-https
  bind *:443
  default_backend ingress-https
  option tcplog

backend ingress-https
  balance source
  server worker1-https-router0 129.40.242.154:443 check
  server worker2-https-router1 129.40.242.155:443 check
```

Both front-end rules tell HAProxy to listen to all requests on the port. The rules also specify which back-end rule is used to load balance a request. For example, the front-end rule `ingress-http` tells HAProxy to listen to all requests on port 80 and uses the back-end rule `ingress-http` to proxy the request. Then, the back-end rule `ingress-http` load balances the request to port 80 of either `worker1` or `worker2`.

In “Ingress controller configuration” on page 8, we created a nodeport ingress controller, `router-nodeport-ingress1` and Red Hat OpenShift assigned two ports, 30892 and 31722:

```
[root@bastion ~]# oc get svc -n openshift-ingress
NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)                                     AGE
router-internal-default             ClusterIP     172.30.187.184 <none>       80/TCP,443/TCP,1936/TCP                 6d23h
router-nodeport-ingress1            NodePort     172.30.198.7   <none>       80:30892/TCP,443:31722/TCP              5d21h
```

These ports must be added to the HAProxy configuration file so that HAProxy knows how to proxy network traffic. Example 5 on page 11 shows the additional front-end and back-end rules to accomplish this task. The front-end rule `vlan1285-http` tells HAProxy to listen on port 80 for network traffic that is directed to IP address 129.40.94.9 only (this IP address is the one that we assigned to our secured VLAN interface), and the back-end rule `vlan1285-http` load balances the network request to port 30892 (the HTTP port that is assigned by our new nodeport ingress controller) of either `worker1` or `worker2`. A similar pair of front-end and back-end rules is also needed to handle HTTPS traffic.

Example 5 Additional front-end and back-end rules

```
frontend vlan1285-http
  bind 129.40.94.9:80
  default_backend vlan1285-http
  mode tcp
  option tcplog

backend vlan1285-http
  balance source
  mode tcp
  server worker1-http-router0 129.40.242.154:30892 check
  server worker2-http-router1 129.40.242.155:30892 check

frontend vlan1285-https
  bind 129.40.94.9:443
  default_backend vlan1285-http
  mode tcp
  option tcplog

backend vlan1285-https
  balance source
  mode tcp
  server worker1-https-router0 129.40.242.154:31722 check
  server worker2-https-router1 129.40.242.155:31722 check
```

With the updated configuration file, network traffic that is directed to 129.40.242.150:80 (our Bastion node main interface) is proxied by using the front-end and back-end rules with the label `ingress-http`, and traffic that is directed to 129.40.94.9:80 (our secured VLAN interface) is proxied by using the front-end and back-end rules with the label `vlan1285-http`. The same is true for network traffic on port 443.

As we create more secured VLAN interface and nodeport ingress controllers, we must add similar front-end and back-end rules for the new IP address and ports.

Red Hat OpenShift network policy

Red Hat OpenShift Container Platform uses a Red Hat OpenShift SDN as the default container network interface (CNI) plug-in to allow communication between Pods and services. This default SDN plug-in can be configured in three modes:

1. Network policy
2. Multitenant
3. Subnet

By default, all Pods and services in Red Hat OpenShift can communicate with each other. After we create a network policy object for a project by using a Pod selector that matches the namespace's label, all Pods and services within the project are isolated from every other project within the cluster.

To achieve network isolation between projects and allow traffic between the Red Hat OpenShift ingress controller Pods that we created for the project, we create a network policy object that we name `allow-pod-and-namespace-both`.

Example 6 shows a namespace selector and Pod selector that are used as part of the configuration to allow the traffic from the openshift-ingress namespace, which is labeled ingress, and the Pod within the openshift-ingress namespace, which is labeled ingress1, to the tenant namespace, which is named demoproject.

To achieve network isolation between projects and allow traffic between the Red Hat OpenShift ingress controller Pods that are created for the project, we create a network policy object that is named allow-pod-and-namespace-both and list it by running the following command (Example 6):

```
[root@bastion ~]# oc create -f network-isolation.yaml -n demoproject
```

Example 6 The network-isolation.yaml contents

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
        podSelector:
          matchLabels:
            ingresscontroller.operator.openshift.io/deployment-ingresscontroller:
            ingress1
```

The following command shows the network policies:

```
[root@mtocp-dev-1b1 ~]# oc get networkpolicies
NAME                                POD-SELECTOR  AGE
allow-pod-and-namespace-both        <none>       5h51m
[root@mtocp-dev-1b1 ~]#
```

Application deployment

The following steps show an example of how to deploy an application by using the new subdomain and ingress controller that we created. From the Red Hat OpenShift console, complete the following steps:

1. Select the **Developer** account, as shown in Figure 4.

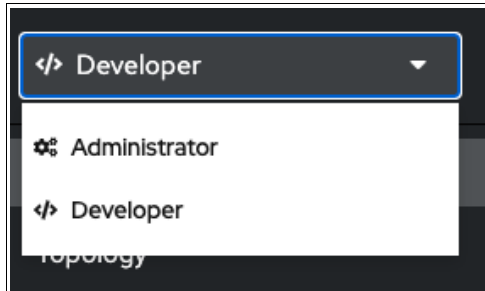


Figure 4 Selecting the account type

2. If you choose to add a From Catalog application, as shown in Figure 5, go to step 3.

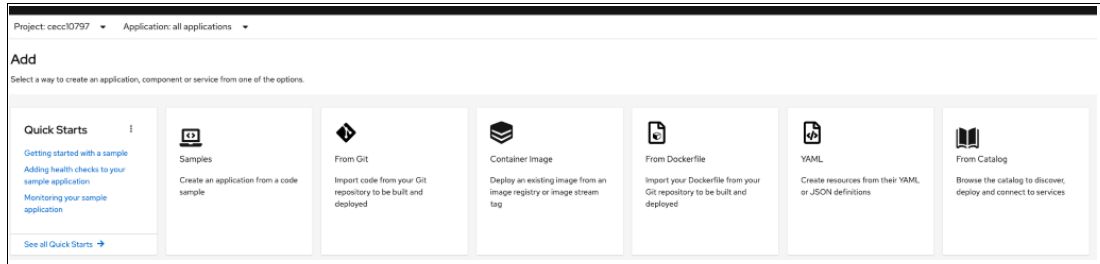


Figure 5 Adding an application

3. Select the Apache template, as shown in Figure 6.

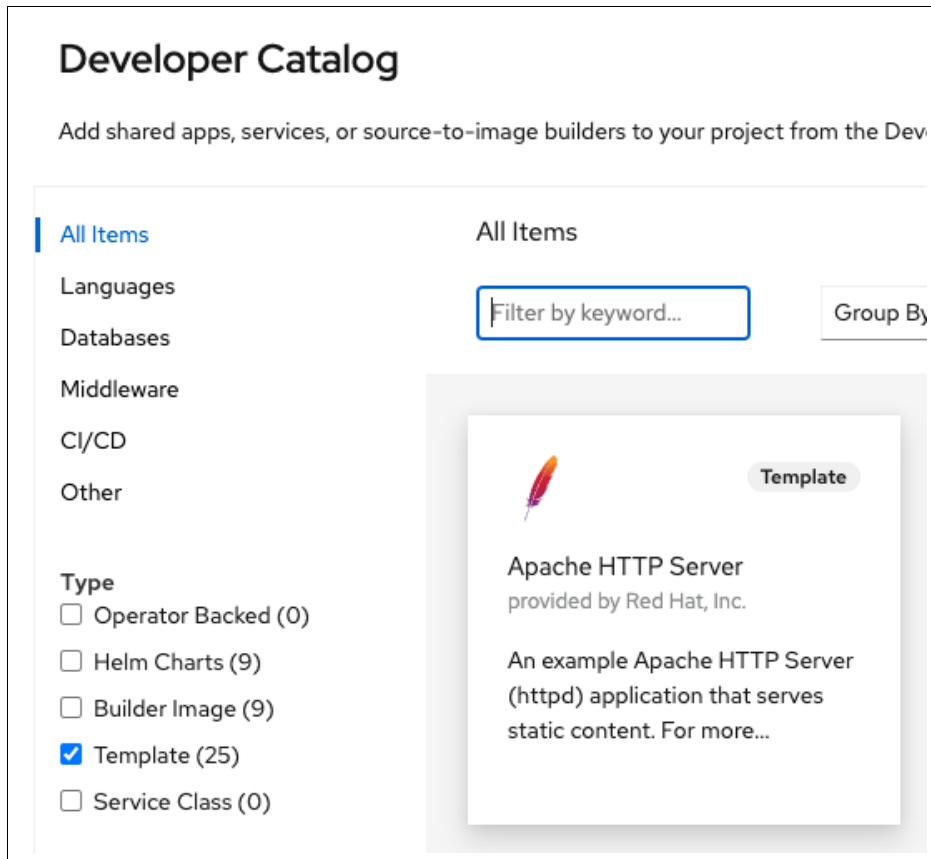


Figure 6 Developer Catalog window

4. The template autopopulates most information. An Application Hostname must be specified by using the new subdomain. As part of the application deployment, Red Hat OpenShift creates the route and lets you access the applications from your web browser. Because our subdomain is `apps.ocp4.example.ihost.com`, we name the Application Hostname `example-to-showusers.apps.ocp4.example.ihost.com`, as shown in Figure 7.

Instantiate Template

Namespace *
PR default

Name *
httpd-example
The name assigned to all of the frontend objects defined in this template.

Namespace *
openshift
The OpenShift Namespace where the ImageStream resides.

Memory Limit *
512Mi
Maximum amount of memory the container can use.

Git Repository URL *
https://github.com/sclorg/httpd-ex.git
The URL of the repository with your application source code.

Git Reference

Set this to a branch name, tag or other ref of your repository if you are not using the default branch.

Context Directory

Set this to the relative path to your project if it is not in the root of your repository.

Application Hostname
example-to-showusers.apps.ocp4.example.ihost.com
The exposed hostname that will route to the httpd service, if left blank a value will be defaulted.

GitHub Webhook Secret
(generated if empty)
Github trigger secret. A difficult to guess string encoded as part of the webhook URL. Not encrypted.

Generic Webhook Secret
(generated if empty)
A secret string used to configure the Generic webhook.

Create **Cancel**

Figure 7 Instantiate Template window

Authors

This paper was produced by a team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center.

Loïc Fura is a Power Systems Architect in IBM Garage™ for Systems at Poughkeepsie, NY, US. He started with IBM in Montpellier, France at the Benchmark Center in 2007, where he acquired deep expertise with IBM Power Systems administration and performance tuning. In 2013, he moved to Poughkeepsie to join IBM Garage for Systems to design solutions to showcase and prove the latest technologies on Power Systems.

Joyce Mak is a Power Systems Technical Specialist with IBM Garage for Systems and IBM Technology Sales. In addition to Red Hat OpenShift, she has worked extensively on Power Systems, including IBM PowerVM®, IBM AIX®, and Linux. She designed and automated the deployment of multiple Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) solutions within IBM Garage for Systems.

Veera Santoshi Chandanalakshmi Solasa is an IBM Garage for Systems Technical Specialist at IBM Poughkeepsie, NY, US. She started her career with IBM India Pvt Ltd in 2010 as a Linux systems engineer and moved to IBM Poughkeepsie in 2016 as a Power Systems Specialist. She is a Red Hat Certified Ansible Specialist, Red Hat Certified Linux Engineer, Red Hat Certified System Administrator, and Red Hat Certified OpenStack Administrator. She has knowledge about Red Hat OpenShift, Red Hat Ansible, Kubernetes Contain to help the progression of IBM sales opportunities.

Thanks to the following people for their contributions to this project:

Dino Quintero
IBM Redbooks®, Poughkeepsie Center

Judy Kurkela
IBM Garage for Systems, Poughkeepsie, NY, US

Daniel Casali
IBM Client Technical Specialist, New Jersey, US

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience by using leading-edge technologies. Your efforts help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.


Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

AIX®
IBM®

IBM Garage™
PowerVM®

Redbooks®
Redbooks (logo) ®

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Ansible, OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.



REDP-5641-00

ISBN 0738459895

Printed in U.S.A.

Get connected

