



Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and O..



PREV

4. Storage



Aa



NEXT

at Architect...



© William Caban 2019

W. Caban, *Architecting and Operating OpenShift Clusters*

https://doi.org/10.1007/978-1-4842-4985-7_5

5. Load Balancers

William Caban¹

(1) Columbia, MD, USA

As seen in Chapters 1 through 4, the OpenShift platform integrates and builds on top of Kubernetes to provide an environment to run and scale

containerized applications reliably. To maintain the most resiliency and benefit the most from the HA capabilities of the platform, the infrastructure hosting the cluster should use load balancers to steer the traffic to the Nodes in the cluster serving the application at a given time. This chapter explores various configuration options when using load balancers with OpenShift.

Load Balancer Overview

When considering the use of external Load Balancers with the OpenShift platform, there are general target areas or traffic types. Each type of traffic will have different requirements based on the desired outcome and the capabilities of the external device or virtual appliance used at load balancing. The use cases for load balancer can be grouped in, at least, the following three types:

- **Load balancing traffic to the *Master Nodes*** (#1 in Figure [5-1](#)): This load balancer should be present for any highly available deployment. For small deployments and lab environments, OpenShift provides the option to deploy a software load balancer based on HAProxy. (Refer to “HA for Masters Services” section in Chapter [2](#)).
- **Load balancing traffic to the *Infrastructure Nodes*** (#2 in Figure [5-1](#)): This is the load balancer handling the traffic to applications running on the cluster and using the *OpenShift Router* as their ingress endpoint. This load balancer is recommended for any highly available deployment even though it can be as simple as a round-robin DNS resolution for the apps wildcard subdomain.
- **Load balancing traffic directly to *Application Nodes or Pods*** (#3, #4, and #5 in Figure [5-1](#)): This load balancer only exists in nonstandard deployments requiring specialized networking interaction between the client and the application *Nodes* or directly with the *Pods*.

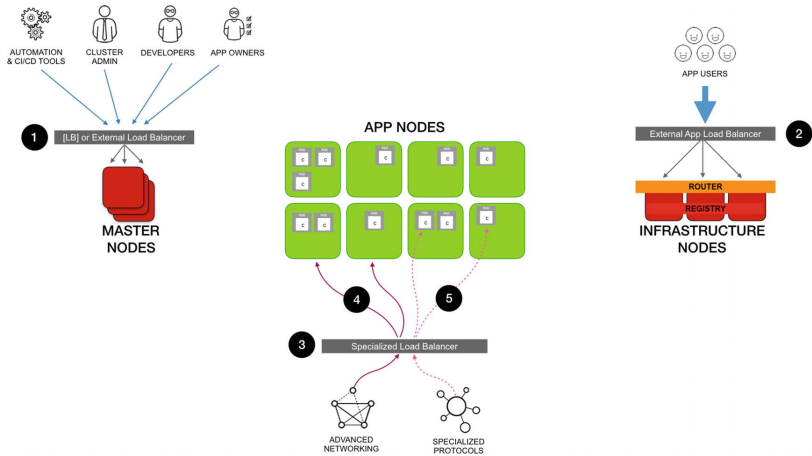


Figure 5-1 OpenShift and Load Balancers

Load Balancer Considerations

There are many load balancer options in the market. Instead of focusing on a particular software or hardware solution, let's focus on the basic requirements for each type of traffic and destination in an OpenShift cluster.

CONSIDERATIONS FOR MASTER NODES

As presented during the discussion of *High Availability for Master Nodes* in *Chapter 2*, these *Nodes* are the ones exposing the *Kubernetes APIs*, the web interface for the *Developer* or *Application Console*, the *Service Portal*, and the *Operations Console* (see #1 in Figure 5-1). From the perspective of web sessions, the *Master Nodes* are stateless, meaning it does not matter which *Master* receives the request during interactions with the *API*. There are no special requirements for persistent sessions or sticky sessions. Because of this, the load balancing service functioning as the front end for the *Master Nodes* can use simple load balancing algorithms (i.e., source IP, round-robin, etc.) to distribute the load among the *Master Nodes*.

Refer to *Chapter 2* for details on the requirements for load balancers for

Master Nodes.

CONSIDERATIONS FOR INFRASTRUCTURE NODES

Traffic load balancing for the *Infrastructure Nodes* refers to a load balancer handling the traffic destined to the *OpenShift Routers* (see #2 in Figure [5-1](#)) which serve as the main ingress point for any external traffic destined to applications and services running on the cluster. A simple *DNS round-robin* resolution can be used to spread traffic across *Infrastructure Nodes* and, from that perspective, an external load balancer for traffic destined to these *Nodes* is optional. Normally, production environments prefer to have more advanced load balancing capabilities to distribute the traffic among the *OpenShift Routers*. In those cases, an external load balancer is used.

This external load balancer for the *OpenShift Routers* should be configured in passthrough mode (see Listings [5-1](#) and [5-2](#)). This means the load balancer will do *connection tracking* and *Network Address Translation (NAT)*, but the *TCP* connections are not terminated by the load balancer; instead, they are forwarded to one of the *Router* instances at the *Infrastructure Nodes* (see #1 in Figure [5-2](#)).

```
# NOTE: extract from nginx.conf
<snip>
stream {
    # Passthrough required for the routers
    upstream ocp-http {
        # Worker Nodes running OCP Router
        server worker-0.ocp.example.com:80;
        server worker-1.ocp.example.com:80;
    }
    upstream ocp-https {
        # Worker Nodes running OCP Router
        server worker-0.ocp.example.com:443;
        server worker-1.ocp.example.com:443;
    }
    server {
```

```

        listen 443;
        proxy_pass ocp-https;
    }

    server {
        listen 80;
        proxy_pass ocp-http;
    }
}
<snip>

```

Listing 5-1 Passthrough configuration example with NGINX

```

# NOTE: extract from haproxy.cfg
<snip>
frontend ocp-http
    bind *:8080
    default_backend ocp-http
    mode tcp
    option tcplog

backend ocp-http
    balance source
    mode tcp
    server worker-0 192.168.1.15:80 check
    server worker-1 192.168.1.16:80 check

frontend ocp-https
    bind *:443
    default_backend ocp-https
    mode tcp
    option tcplog

backend ocp-https
    balance source
    mode tcp
    server worker-0 192.168.1.15:443 check
    server worker-1 192.168.1.16:443 check
<snip>

```

Listing 5-2 Passthrough configuration example with HAProxy

At the *OpenShift Router*, this traffic is matched with a *Route* (see #3 in Figure 5-2), and it is load balanced among the *Pods* of the corresponding *Service* object (see #4 in Figure 5-2).

The *OpenShift Router* supports *roundrobin*, *leastconn*, and *source* as the load balancing algorithms or load balancing strategies.¹ The *source* is considered the default load balancing strategy.

The default load balancing strategy and other *OpenShift Router* parameters can be configured by setting the corresponding *Environment Variable* for the *OpenShift Router DeploymentConfig*.²

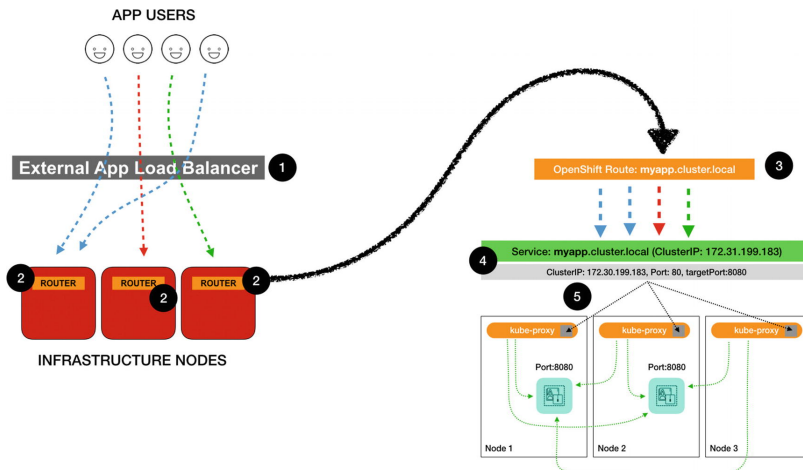


Figure 5-2 Traffic flow from external load balancers to OpenShift Routers

NOTE The specific behavior of the traffic at the *Router* level may be different if using third-party *Router* plugins.

The *OpenShift Router* supports the following protocols:

- HTTP
- HTTPS with SNI³
- WebSockets
- TLS with SNI

Any traffic for protocols outside these web protocols cannot make use of the *OpenShift Router* and *Routes* capabilities. Those cases are covered in the following section.

CONSIDERATIONS FOR SPECIALIZED PROTOCOLS

As we saw in the previous section, the OpenShift Router cannot be used with traffic using non-web-based protocols or with traffic using the UDP protocol. This book aggregates all these cases under the “specialized protocols” category. The details on how to provide load balancing to these protocols are highly dependent on the Kubernetes and OpenShift options used to expose these services.

As illustrated in Figure [5-3](#), some configuration options will rely on the native capabilities of kube-proxy, while others may depend on the capabilities provided by the specific SDN solution used in the cluster.

NODES

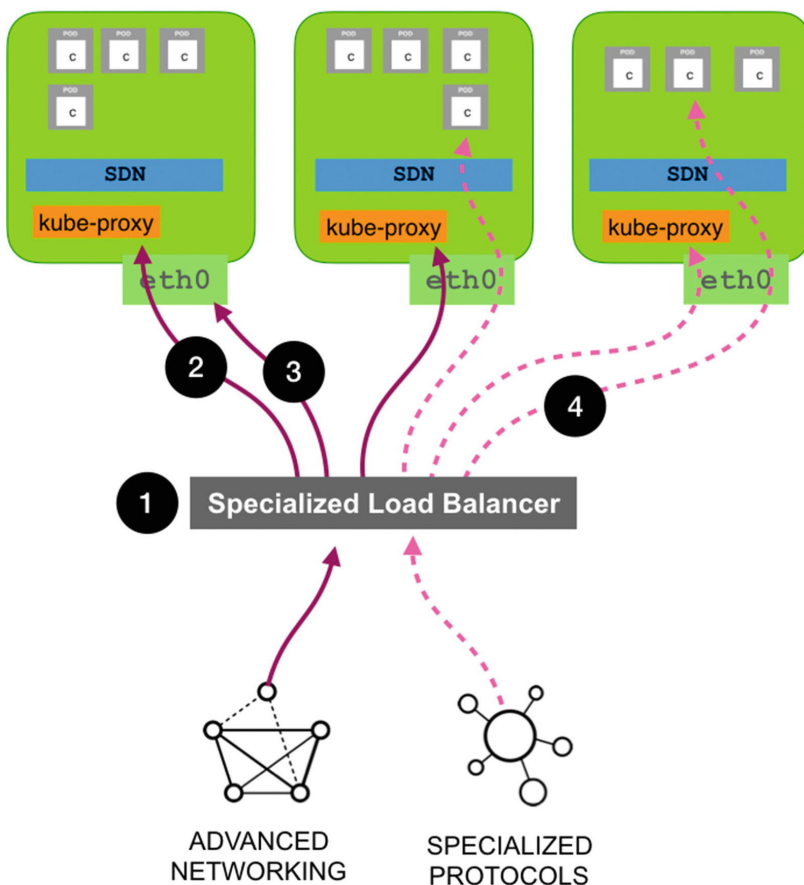


Figure 5-3 Representation of load balancer for non-HTTP/HTTPS/TLS protocols

OpenShift provides several options to support non-web-based or UDP-based traffic. The following list provides a general description of the options and their functionalities:

- Service External IP:** This option allocates an *External IP* for the *Service* from the CIDR defined by *externalIPNetworkCIDRs* in the *Master Nodes* configuration.⁴ When using this option, the *ExternalIP* is defined and managed by the kube-proxy agent in each node (see #2 in Figure 5-3). From a load balancer perspective, the traffic can be directed to any of the *Application* or *Infrastructure Nodes*. Once the traffic arrives to the *Node*, the incoming traffic is forwarded internally by *kube-proxy* to the corresponding *Pods* as it does for any other *Service* communication.

- **LoadBalancer:** The *LoadBalancer* option behaves differently when used in a Cloud provider vs. when used locally. At a Public Cloud provider, this option will allocate an ExternalIP for the Cloud provider Load Balancing service. When this option is used in non-Cloud environments, it allocates an *ExternalIP* from the *ingressIPNetworkCIDR* network. When this variable is not specified in the *Master Nodes* configuration,⁵ the default network for this type of *Service* is 172.29.0.0/16. From a load balancer perspective, the traffic can be directed to any of the *Application* or *Infrastructure Nodes*. The incoming traffic to the *Nodes* is forwarded by the *kube-proxy* to the selected Pods (see #2 in Figure 5-3).
- **nodePort:** This option allows the user to specify a port for the *Service* from the default *nodePort* range of 30000–32767. When the *Service* is created with this option, *kube-proxy* starts listening to that port in every *Node*. From a load balancer perspective, the traffic can be directed to any of the *Application* or *Infrastructure Nodes*. The incoming traffic to the *Nodes* is forwarded by the *kube-proxy* to the selected Pods (see #2 in Figure 5-3).
- **hostPort:** This option allows the user to bind a *Pod* to any *Port* of the *Node*, and the *Container* will be exposed to the external network as <hostIP>:<hostPort>, where *hostIP* is the physical IP of the *Node* running the particular *Pod*, and *hostPort* is the port number specified in the *Pod* definition. The load balancer for this option needs to be configured to send traffic to the physical IP of the *Node* running the *Pod* (see #3 in Figure 5-3). A consideration when using this option is that the *hostIP* of the <hostIP>:<hostPort> pair will change if the *Pod* recreated in another *Node*.
- **hostNetwork:** This option enables the *Pod* to have full visibility of the *Node* network interfaces. This is the equivalent of the *Pod* sharing the network namespace with the *Node*. This option is not recommended for running application. It is normally used by SDN plugins and other network functions deployed as DaemonSets or privileged containers.
- **IP Failover:** The *IP Failover*⁶ option is an OpenShift-specific capability which enables the creation of a *Virtual IP address (VIP)* for the applications. When this configuration is enabled, OpenShift deploys *Keepalived* privileged containers to handle the particular *VIP*. These *Keepalived Pods* for the IP Failover capability can be deployed cluster-wide

or in a subset of Nodes matching a particular label. These *Pods* use the *VRRP* protocol to maintain the *VIP* address active. Only one of the *Keepalived Pods* will be active or in *MASTER* state serving the *VIP* address at a time; the others will be on standby or in *BACKUP* state. The *VRRP* protocol is used to determine which *Pod* gets to be active for a particular *VIP*. From a load balancer perspective, the *Node* with the active *Pod* serving the *VIP* address is the only one capable of handling the traffic destined to that *VIP* address.

In addition to the options described here, there are other techniques which are more relevant to *Cloud* environments. One of these options is the *LoadBalancer* which requires external support by a Cloud provider. In this case, Kubernetes interacts with the *Cloud* platform to provision a *Cloud Load Balancer* with an *External IP* for the *Service*. Another option are SDNs like Calico or MacVLAN which can be configured to expose the *Pods IPs* to the upstream networking equipment enabling direct access to the *Pods* from the external networks (see #4 in Figure [5-3](#)). In this case, it is up to the networking team to manage the network traffic directed to the *Pods*.

Summary

Configuring a load balancer service in *OpenShift* for *Master Nodes* and *OpenShift Routers* at the *Infrastructure Nodes* can be a simple pass-through load balancing configuration. These can be considered web-friendly protocols: HTTP, HTTPS, TLS, and WebSockets. Supporting UDP or non-web-friendly protocols with Kubernetes and OpenShift requires the use of a different set of objects and capabilities. The particular load balancer configuration for these use cases requires an understanding of the workload, the option being used, and the level of exposure of the *Services* and *Pods* to the external networks.

Having a base understanding of how the networking, storage, and traffic routing options work for OpenShift, Chapter [6](#) will focus on putting all this knowledge together for a successful deployment of a cluster.

The supported load balancing strategies are described here:

1

<https://docs.openshift.com/container-platform/3.11/architecture/networking/routes.html#load-balancing>

2

A list of available Environment Variables to fine-tune the OpenShift

Router is available at <https://docs.openshift.com/container-platform/3.11/architecture/networking/routes.html#env-variables>

3

Standard Name Indication (SNI) is an extension of the TLS protocol.

With this extension, the client indicates the *hostname* it is trying to contact at the start of the *handshaking* process.

4

To use this option, the *externalIPNetworkCDIRs* must be configured and

enabled: https://docs.openshift.com/container-platform/3.11/dev_guide/expose_service/expose_internal_ip_service.html

5

To customize the ExternalIPs for this option, use the


ingressIPNetworkCIDR variable in the *Master Node* configuration:

https://docs.openshift.com/container-platform/3.11/admin_guide/tcp_ingress_external_ports.html#unique-external-ips-ingress-traffic-configure-cluster

6

For configuration requirements for the OpenShift IP Failover capability,

refer to https://docs.openshift.com/container-platform/3.11/admin_guide/high_availability.html#configuring-ip-failover

 [PREV](#)
[4. Storage](#)

[NEXT](#) 
[6. Deployment Architect...](#)