Scale and Expose Applications to External Access

Objectives

- Expose applications to clients outside the cluster by using Kubernetes ingress and OpenShift routes.

Outcomes

- Allocate IP addresses for pods and services in a Red Hat OpenShift Container Platform (RHOCP) cluster.
- Configure services to enable communication between pods.
- Expose applications to external networks by using routes and ingress objects.

IP Addresses for Pods and Services

Real-world applications typically require multiple pods to scale horizontally. To meet growing user demand, RHOCP deploys many pods that run the same containers from a single pod resource definition. A `Service` resource defines a single IP and port combination. The resource assigns a stable IP address to a pool of pods. The resource balances client requests across those pods.

RHOCP assigns each service a unique IP address for clients to connect to. The assignment uses a round-robin load-balancing strategy by default. The service IP address originates from an internal OVN-Kubernetes network. The network, which is separate from the pod network, is accessible only to pods within the cluster. RHOCP adds each pod to match the selector of the service as an endpoint.

NOTE

OVN-Kubernetes, which is the default network provider starting with RHOCP 4.18, manages pod-to-pod and service-to-pod communication. The provider ensures efficient traffic routing within the cluster.

Containers within Kubernetes pods must avoid connecting directly to dynamic IP addresses of other pods. Instead, services link stable OVN-Kubernetes IP addresses to pods to ensure scalability and fault tolerance. When pods restart, replicate, or reschedule to different nodes, RHOCP updates the service endpoints accordingly.

Service Types

RHOCP offers several service types to meet diverse application needs, cluster infrastructure, and security requirements.

ClusterIP

The `ClusterIP` type, which is the default unless otherwise specified, assigns a cluster-internal IP address to the service. The assignment makes the service accessible only within the RHOCP cluster.

`ClusterIP` services enable pod-to-pod communication. RHOCP allocates IP addresses from a dedicated service network. The network, which is accessible only inside the cluster, supports internal communication. Most applications benefit from this service type because Kubernetes automates management of the service.

LoadBalancer

The `LoadBalancer` type directs RHOCP to provision a load balancer of the cloud provider. The load balancer assigns an externally accessible IP address to the application.

Use caution when deploying `LoadBalancer` services because costs can be high when assigning a load balancer to every application. These services expose applications to external networks. The exposure requires additional security configurations to prevent unauthorized access.

ExternalIP

The `ExternalIP` service redirects traffic from a virtual IP address on a cluster node to a pod. A cluster administrator assigns the virtual IP address to a node. The administrator configures failover to another node if needed.

WARNING

Starting with RHOCP 4.18, `ExternalIP` is a legacy feature due to security concerns. Direct network connections to cluster nodes, which this service requires, conflict with most security policies. Use the `LoadBalancer` or the `NodePort` type unless the `ExternalIP` type meets specific requirements.

RHOCP configures Network Address Translation (NAT) rules to route traffic from the virtual IP address to the pod. Administrators must ensure that external IP addresses route correctly to nodes. Additional security measures protect the cluster from external access.

NodePort

The `NodePort` service exposes a service on a specific port, with a default range of 30000–32767, on the IP address of each cluster node. Each node redirects traffic from this port to the endpoints of the service.

WARNING

`NodePort` services require direct network connections to cluster nodes, which poses a security risk. Implement strict firewall rules and access controls to mitigate risks.

ExternalName

The `ExternalName` service maps a Kubernetes service to an external DNS name that is specified in the `externalName` field. The Kubernetes DNS server returns the `externalName` in a Canonical Name (CNAME) record. The record directs clients to resolve the external name to an IP address.

Using Routes for External Connectivity

RHOCP provides `Route` resources to expose applications to external networks. `Route` resources support HTTP, HTTPS, TCP, and other protocols, and prioritize HTTP and TLS-based applications for external exposure. Non-HTTP applications, such as databases, typically remain internal. `Route` and `Ingress` resources handle ingress traffic in RHOCP.

A `Route` resource assigns a publicly accessible hostname to an application. The OpenShift ingress controller, which is HAProxy-based, redirects traffic from a public IP address to pods. The OpenShift ingress controller is the default, with support for third-party ingress controllers that are deployed in parallel. `Route` resources offer advanced features, which surpass standard Kubernetes `Ingress` capabilities. Features include TLS re-encryption, passthrough, and split traffic for blue-green deployments.

To create a route by using the `oc` CLI, use the following command:

```
[user@host ~]$ oc expose service api-frontend --hostname api.apps.acme.com
```

The `--target-port` option of the `oc expose` command specifies the name or number of the container port that the service uses to communicate with the pods. If you provide no port value, then the port copies from the configuration of the deployment.

Omitting the `--hostname` option prompts RHOCP to generate a hostname in the `<route-name>-<project-name>.<default-domain>` format. For example, a `frontend` route in an `api` project with a wildcard domain of `apps.example.com` becomes `frontend-api.apps.example.com`.

IMPORTANT

The DNS server for the wildcard domain resolves all names to configured IPs. The server does not recognize specific route hostnames. The OpenShift ingress controller treats route hostnames as HTTP virtual hosts. Invalid or non-existent route hostnames trigger an HTTP 503 error.

When creating a route, configure the following settings:

1. **Service Name**: Specifies the service to determine target pods.
2. **Hostname**: Sets the route hostname as a subdomain of the cluster wildcard domain, which RHOCP can autogenerate.
3. **Path**: Defines an optional path for path-based routing.
4. **Target Port**: Matches the `targetPort` in the service, and specifies the listening port of the application.
5. **Encryption Strategy**: Chooses secure (TLS) or insecure routing.

The following example shows a minimal route definition:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: a-simple-route (1)
  labels: (2)
    app: API
    name: api-frontend
spec:
  host: api.apps.acme.com (3)
  to:
    kind: Service
    name: api-frontend (4)
  port:
    targetPort: 8443 (5)
```

(1) Specifies a unique route name.

(2) Defines selectors for the route.

(3) Sets the hostname, which is a subdomain of the cluster wildcard domain.

(4) Identifies the service to redirect traffic to, by determining target pods.

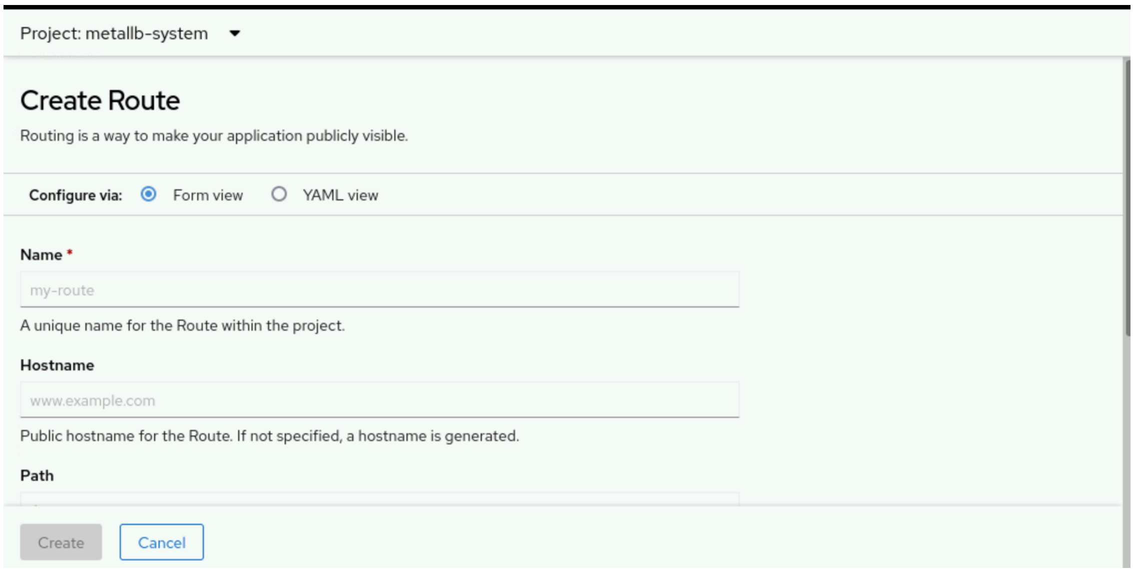
(5) Maps the router port to the endpoint port of the service.

> **NOTE**
>
> Some ecosystem components integrate with `Ingress` resources but not with `Route` resources. Starting with RHOCP 4.18, creating an `Ingress` object automatically generates a managed `Route` object, which is deleted when the `Ingress` object is removed.

To delete a route, use the following command:

```
[user@host ~]$ oc delete route a-simple-route
```

To create a route from the RHOCP web console, go to **Networking** → **Routes**. Click **Create Route**. Customize the name, hostname, path, and target service by using the form or the YAML editor.


Project: metallb-system

Create Route

Routing is a way to make your application publicly visible.

Configure via: [✓ Form view] YAML view

Name *

my-route

A unique name for the Route within the project.

Hostname

www.example.com

Public hostname for the Route. If not specified, a hostname is generated.

Path

Using Ingress Objects for External Connectivity

A Kubernetes `Ingress` resource provides similar functions to RHOCP `Route` resources, and supports HTTP, HTTPS, Server Name Indication (SNI), and TLS with SNI. The OpenShift ingress controller processes `Ingress` objects, and enables features such as TLS termination, path redirection, and sticky sessions.

> **NOTE**
>
> Although `Ingress` resources are standard in Kubernetes, `Route` resources are preferred for advanced features and native integration with the OpenShift ingress controller.

To create an `Ingress` object, use the following command:

```
[user@host ~]$ oc create ingress ingr-sakila --rule="ingr-
sakila.apps.ocp4.example.com/*=sakila-service:8080"
```

The following example shows a minimal `Ingress` definition:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend (1)
spec:
  rules:
  - host: www.example.com (2)
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend (3)
            port:
              number: 80 (4)
  tls:
  - hosts:
    - www.example.com
    secretName: example-com-tls-certificate (5)
```

(1) Specifies a unique name for the `Ingress` object.

(2) Sets the hostname for inbound traffic.

(3) Identifies the service to redirect traffic to.

(4) Defines the port for the service back end.

(5) Configures TLS for secure paths, which requires a matching host and a certificate secret.

To delete an `Ingress` object, use the following command:

```
[user@host ~]$ oc delete ingress frontend
```

Sticky Sessions

Sticky sessions ensure that stateful application traffic reaches the same pod for a user session. The OpenShift ingress controller can use cookies to manage session persistence for `Route` and `Ingress` resources. The controller selects a pod for a user request. The controller generates a cookie. The controller includes the cookie in the response. The client sends the cookie with subsequent requests, which enables the controller to route traffic to the same pod.

To configure session persistence for an `Ingress` object by using a cookie, use the following command:

```
[user@host ~]$ oc annotate ingress ingr-example ingress.kubernetes.io/affinity=cookie
```

To configure session persistence for a `Route` object with a custom cookie named `myapp`, use the following command:

```
[user@host ~]$ oc annotate route route-example router.openshift.io/cookie_name=myapp
```

To capture the route hostname, use the following command:

```
[user@host ~]$ ROUTE_NAME=$(oc get route route-example -o jsonpath='{.spec.host}')
```

To access the route and save the cookie, use the following command:

```
[user@host ~]$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

To use the saved cookie for subsequent requests, use the following command:

```
[user@host ~]$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

Using the saved cookie ensures that requests reach the same pod.

NOTE

Cookies cannot be set on passthrough routes because the HTTP traffic is encrypted, and the router does not terminate TLS or read the contents of the request.

Load Balance and Scale Applications

Administrators and developers can scale the number of replica pods in a deployment to handle traffic surges or to conserve resources.

To scale a deployment, use the following command:

```
[user@host ~]$ oc scale --replicas=5 deployment/scale
```

The deployment updates the replica set. The replica set creates or deletes pods to match the intended replica count.

Avoid modifying a replica set directly, because the deployment controller manages these changes.

Load Balance Pods

A Kubernetes `Service` resource functions as an internal load balancer, by providing access to a workload via the service name. The service distributes incoming connections across replicated pods. The OpenShift ingress controller uses the selector of the service to identify the service and endpoints. When both the controller and a service provide load balancing, RHOCP relies on the controller to direct traffic to pods. The controller detects changes in IP addresses of the service. The controller updates the configuration. Custom controllers can propagate API object changes to external routing solutions.

The OpenShift ingress controller maps external hostnames. The controller balances service endpoints over protocols that include distinguishing information, such as HTTP host headers.

REFERENCES

For more information, see the *Understanding Networking* section in the Red Hat OpenShift Container Platform 4.18 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/networking_overview/understanding-networking

For more information about services, refer to the *Service [v1]* section in the Red Hat OpenShift Container Platform 4.18 *Network APIs* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/network_apis/service-v1

For more information about routes, refer to the *Configuring Routes* section in the Red Hat OpenShift Container Platform 4.18 *Ingress and Load Balancing* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/ingress_and_load_balancing/configuring-routes

For more information about load balancing, refer to the *Ingress and Load Balancing* section in the Red Hat OpenShift Container Platform 4.18 *Ingress and Load Balancing* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/ingress_and_load_balancing/index

For more information, see the *Cluster Network Operator in OpenShift Container Platform* chapter in the Red Hat OpenShift Container Platform 4.18 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/networking_operators/cluster-network-operator

For more information, see the *About the OVN-Kubernetes Network Plug-in* chapter in the Red Hat OpenShift Container Platform 4.18 *Networking* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/ovn-kubernetes_network_plugin/index