Lesson 4: Scale and Expose Applications to External Access

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

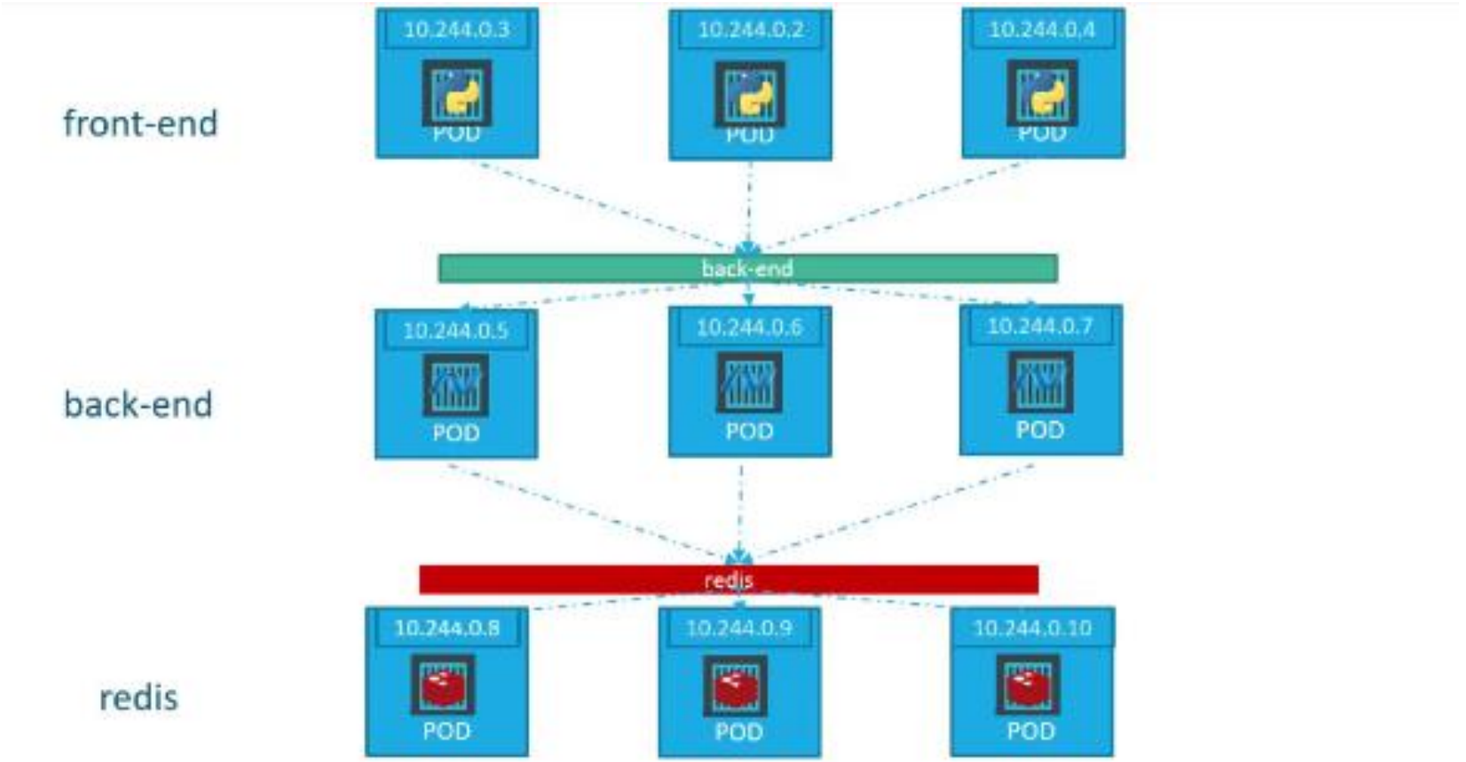
IP Addresses for Pods and Services

- Deployment runs multiple pods on multiple nodes
- Application needs scale horizontally to meet growing user demand
- A Service
  - defines stable single IP/port to pool of pods
  - provides load-balancing across member pods
- CNI assign IPs to services and pods
- Services uses label selector to connect to pods
- Services are updated when pods are restarted, replicated, or rescheduled to different nodes.

Service Types

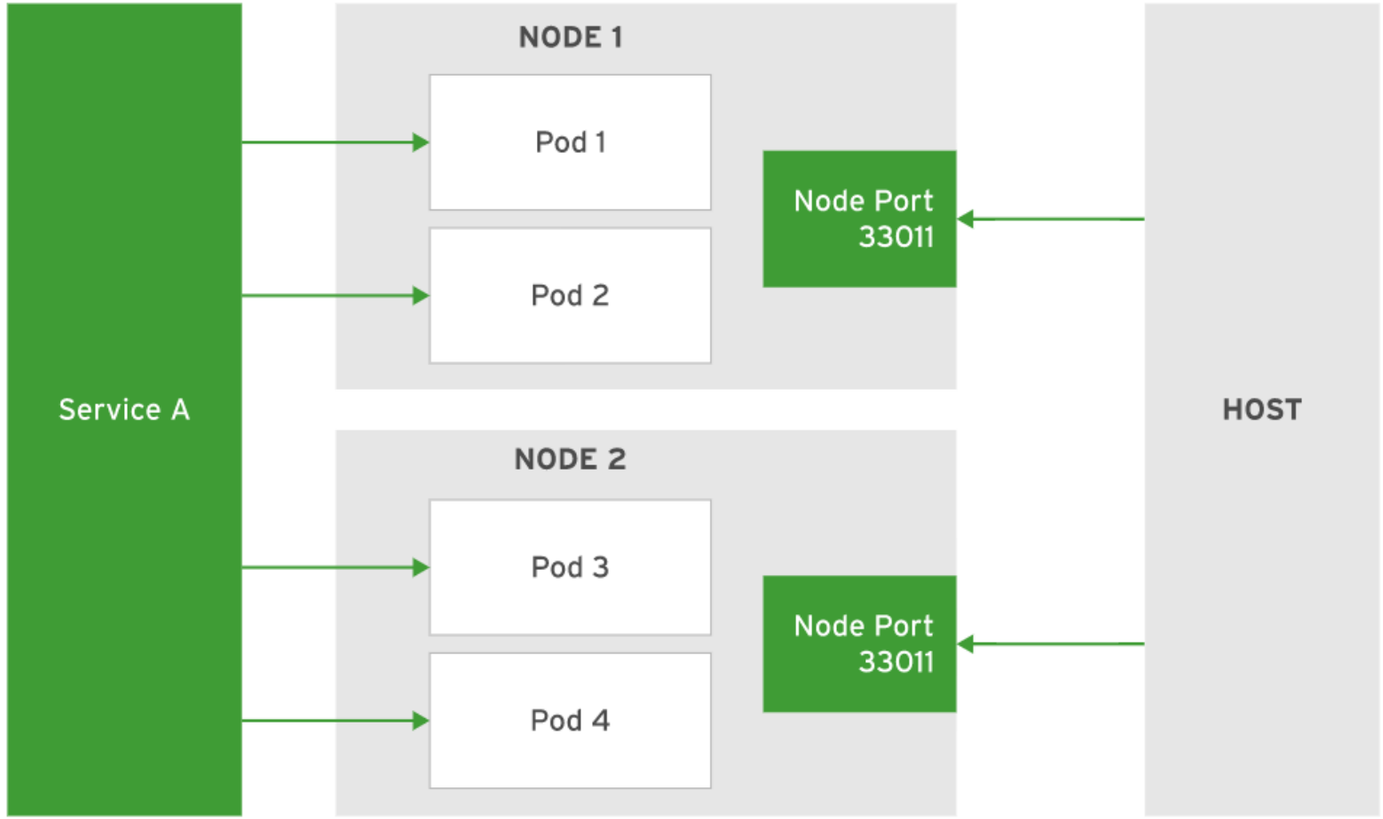
- ClusterIP
  - Is default. Exposes service on cluster-internal IP
  - Service is reachable only from within cluster
- NodePort
  - Exposes service to external clients
  - Opening ports (range from 30000 to 32767) on all worker nodes
  - Client connect to node's IP address, on port :3xxxx
- Load Balancer
  - Exposes service to external clients
  - Applicable when Kubernetes is on cloud platform
- Deciding factor
  - Application requirements
  - Security requirements
  - Cluster Infrastructure

Use ClusterIP



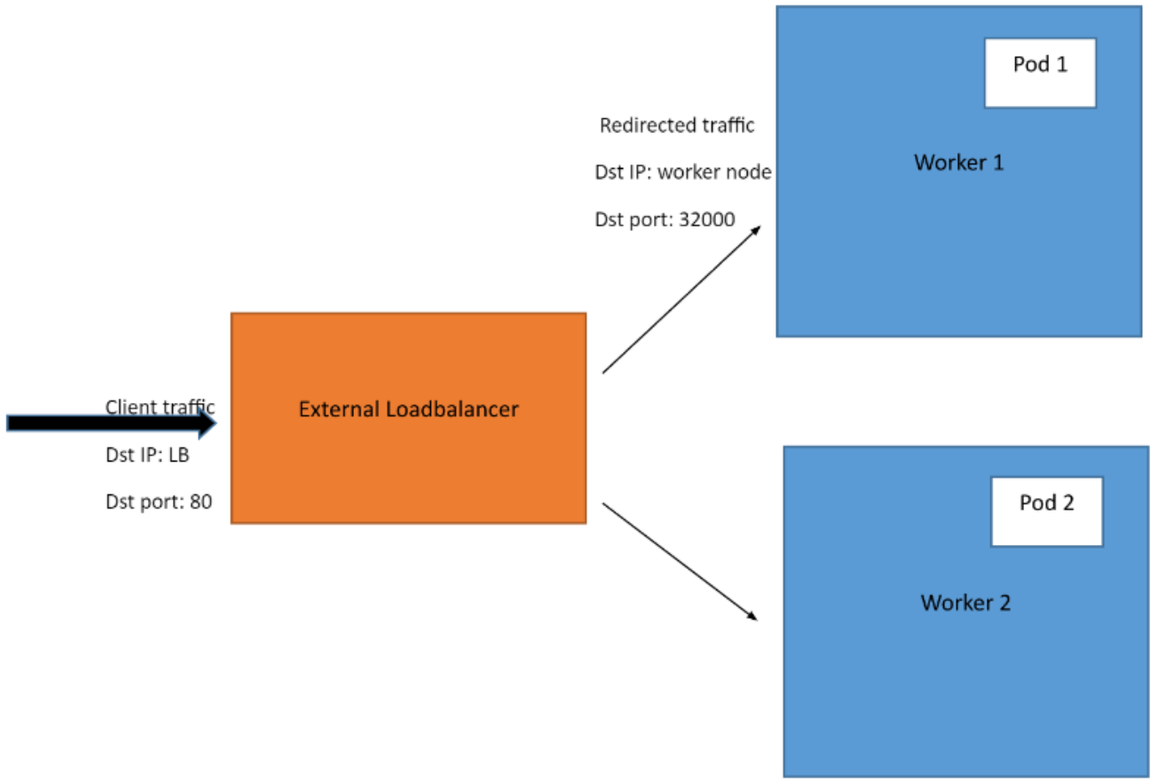
- Is default. Exposes service on cluster-internal IP
- Service is reachable only from within cluster
- Secured environment, however not reachable from external network

Use NodePort



- Exposes service to external clients
- Opening ports (range from 30000 to 32767) on all worker nodes
- Client connect to node's IP address, on port :3xxxx
- Does not load balance automatically
- Does not scale well and insecured

Use LoadBalancer



- Accessible from public network
- Must deploy on supported cloud platform
- Scale well

Using Routes for External Connectivity

- Expose HTTP or HTTPS
- Expose TCP or UDP traffics
- Use of TLS-based applications
- RHOCP uses Routes and ingress
  - Routes to expose applications to external networks
  - Ingress for handling ingress traffic
- OpenShift ingress operator
  - provides ingress controller
  - can use various third-party ingress controller
- Routes support TLS re-encryption, TLS passthrough, split traffic

Creating Routes

- Use *oc expose service* subcommand.
- Or *oc create route* subcommand.
- The syntax:

```
$ oc expose service SERVICE_NAME [--name ROUTE_NAME]
```

--name customise FQDN.

- If name not specified, automatically generated using following format:

route-name-project-name.default-domain

- Default-domain is configured on OpenShift master using wildcard DNS domain name. Example *ocp4.example.com.*
- Create route from a service named quoted and customize the FQDN

```
$ oc expose service quoted --name quoted-by-me.apps.ocp4.example.com

$ oc get route
```

Consideration when creating route

- The name of a service. The route uses the service to determine the pods to direct the traffic to.
- A hostname for the route. A route is always a subdomain of your cluster wildcard domain. For example, if you are using a wildcard domain of `apps.dev-cluster.acme.com`, and need to expose a `frontend` service through a route, then the route name is as follows:

  ```
  frontend.apps.dev-cluster.acme.com.
  ```

  RHOCP can also automatically generate a hostname for the route.
- An optional path, for path-based routes.
- A target port that the application listens to. The target port corresponds to the port that you define in the `targetPort` key of the service.
- An encryption strategy, depending on whether you need a secure or an insecure route.

Minimal definition for the route

```
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: 8080 (5)
    targetPort: 8443
```

1. The name of the route. This name must be unique.
2. A set of labels that you can use as selectors.
3. The hostname of the route. This hostname must be a subdomain of your wildcard domain, because RHOCP routes the wildcard domain to the routers.
4. The service to redirect the traffic to. Although you use a service name, the route uses this information only to determine the list of pods that receive the traffic.
5. Port mapping from a router to an endpoint in the service endpoints. The target port on pods that are selected by the service that this route points to.

Creating Routes using web console

Using Ingress Objects for External Connectivity

- Ingress is K8s resource
- Route is OpenShift resource
- Can accept external requests and transfer request to pods
- Support HTTP, HTTPs, server name identification (SNI), path redirecting, sticky sessions
- Ingress resources commonly configured in K8s environment.
- Route resource is the preferred method in OpenShift
- Creating ingress object

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

- Deleting ingress object

```
[user@host ~]$ oc delete ingress example-ingress
```

Minimal definition for ingress object

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

1 The name of the ingress object. This name must be unique.

2 The HTTP or HTTPS rule for the ingress object.

3 The host for the ingress object. Applies the HTTP rule to the inbound HTTP traffic of the specified host.

4 The backend to redirect traffic to. Defines the service name, port number, and port names for the ingress object. To connect to the back end, incoming requests must match the host and path of the rule.

5 The configuration of TLS for the ingress object; it is required for secured paths. The host in the TLS object must match the host in the rules object.

Deleting Routes

-

- The syntax:

```
$ oc delete route
```

Sticky Sessions

- Enable stateful application
- Ensure all requests from same client reach the same endpoint
- Applicable to ingress and route resources
- Openshift auto-generate cookies
- User can override cookie by using annotate command

Sticky Sessions - The workflow

- User can create cookie manually for ingress object

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

- User can create cookie manually for route object

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

- After annotate route, capture route hostname in variable

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

- Then, use the curl command to save the cookie and access the route

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

- The cookie is passed back in response to the request, and is saved to the /tmp/cookie_jar directory. Use the curl command and the cookie that was saved from the previous command to connect to the route:

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

- By using the saved cookie, the request is sent to the same pod as the previous request.

Load Balance and Scale Applications

- Add more pods in event of surge in traffic
- Or reduce pods to reclaim resources, so that cluster can use elsewhere
- Manipulate replica set or deployment
- Use of oc scale command

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

- Load Balance Pods
  - Service is used as an internal load balancer
  - Service uses selector label to find endpoints
  - Service will always use round-robin to load balance user traffic to endpoints
  - Routes map external hostname, perform TLS, path-based routing

Guided Exercise: Scale and Expose Applications to External Access

You should be able to:

- Deploy two web applications.
- Create a route and an ingress object to access the web applications.
- Enable the sticky sessions for the web applications.
- Scale the web applications to load-balance the service.

Lab: Deploy Managed and Networked Applications on Kubernetes

You should be able to:

- Deploy a MySQL database from a container image.
- Deploy a web application from a container image.
- Configure environment variables for a deployment.
- Expose the web application for external access.

Summary

MANY RESOURCES IN KUBERNETES AND RHOCP CREATE OR AFFECT PODS.

RESOURCES ARE CREATED IMPERATIVELY OR DECLARATIVELY. THE IMPERATIVE STRATEGY INSTRUCTS THE CLUSTER WHAT TO DO. THE DECLARATIVE STRATEGY DEFINES THE STATE THAT THE CLUSTER MATCHES.

THE OC NEW-APP COMMAND CREATES RESOURCES THAT ARE DETERMINED VIA HEURISTICS.

THE MAIN WAY TO DEPLOY AN APPLICATION IS BY CREATING A DEPLOYMENT.

THE WORKLOAD API INCLUDES SEVERAL RESOURCES TO CREATE PODS. THE CHOICE BETWEEN RESOURCES DEPENDS ON FOR HOW LONG AND HOW OFTEN THE POD NEEDS TO RUN.

A *JOB* RESOURCE EXECUTES A ONE-TIME TASK ON THE CLUSTER VIA A POD. THE CLUSTER RETRIES THE JOB UNTIL IT SUCCEEDS, OR IT RETRIES A SPECIFIED NUMBER OF ATTEMPTS.

RESOURCES ARE ORGANIZED INTO PROJECTS AND ARE SELECTED VIA LABELS.

A *ROUTE* CONNECTS A PUBLIC-FACING IP ADDRESS AND A DNS HOSTNAME TO AN INTERNAL-FACING *SERVICE* IP ADDRESS. SERVICES PROVIDE NETWORK ACCESS BETWEEN PODS, WHEREAS ROUTES PROVIDE NETWORK ACCESS TO PODS FROM USERS AND APPLICATIONS OUTSIDE THE RHOCP CLUSTER.