Kubernetes Networking

Services

Francesco Cappa, Jacopo Marino
Politecnico di Torino

Politecnico di Torino

Networking in K8s

- Pods have unique cluster-wide IPs
  - No need to explicitly create links between **pods** and almost most never need to deal with mapping container ports to host ports.
  - Pods can be treated much like VMs or physical hosts:
    - port allocation, naming, service discovery, load balancing, application configuration, and migration.

- Kubernetes imposes the following fundamental requirements on any networking implementation:
  - pods can communicate with all other pods on any other node without NAT
  - agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node

- This model is compatible with the desire for Kubernetes to enable low-friction porting of apps from VMs to containers.

Networking in Kubernetes cont'd

- Kubernetes networking addresses four concerns:
  1. Containers within a Pod use networking to communicate via loopback.
  2. Cluster networking provides communication between different Pods.
  3. The Service API lets you expose an application running in Pods to be reachable from outside your cluster.
     a) Ingress provides extra functionality specifically for exposing HTTP applications, websites and APIs.
     b) Gateway API is an add-on that provides an expressive, extensible, and role-oriented family of API kinds for modeling service networking.
  4. You can also use Services to publish services only for consumption inside your cluster.

Service

- Service is a method for exposing a network application that is running as one or more Pods in your cluster.
  - No need to modify your existing application to use an unfamiliar service discovery mechanism.
  - Code can be run in Pods; a Service to make that set of Pods available on the network so that clients can interact with it.
  - If using a Deployment to run an app, Pods can be created and destroyed dynamicaly.
  - Pods are ephemeral resources: configuration (e.g. IP configuration) might vary over time.
  - Example: **Backend Pods** provides functionality to **Frontend Pods** inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload?
- The Service API, part of Kubernetes, is an abstraction to help you expose groups of Pods over a network.
  - The set of Pods targeted by a Service is usually determined by a selector that you define. To learn about other ways to define Service endpoints, see Services *without* selectors.

Defining a Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Services without Selectors

- When used with a corresponding set of EndpointSlices objects and without a selector, the Service can abstract other kinds of backends, including ones that run outside the cluster.
- Possible use-cases:
  1. You want to have an external database cluster in production, but in your test environment you use your own databases.
  2. You want to point your Service to a Service in a different Namespace or on another cluster.
  3. You are migrating a workload to Kubernetes. While evaluating the approach, you run only a portion of your backends in Kubernetes.
- Because this Service has no selector, the corresponding EndpointSlice (and legacy Endpoints) objects are not created automatically. You can map the Service to the network address and port where it's running, by adding an EndpointSlice object manually

Defining a Service without Selectors

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```yaml
apiVersion: discovery.k8s.io/v1
kind: EndpointSlice
metadata:
  name: my-service-1 # by convention, use the name of the Service
                     # as a prefix for the name of the EndpointSlice
  labels:
    # You should set the "kubernetes.io/service-name" label.
    # Set its value to match the name of the Service
    kubernetes.io/service-name: my-service
addressType: IPv4
ports:
  - name: '' # empty because port 9376 is not assigned as a well-known
             # port (by IANA)
    appProtocol: http
    protocol: TCP
    port: 9376
endpoints:
  - addresses:
      - "10.4.5.6"
  - addresses:
      - "10.1.2.3"
```

EndpointSlices

- EndpointSlices are objects that represent a subset (a *slice*) of the backing network endpoints for a Service.
- Your Kubernetes cluster tracks how many endpoints each EndpointSlice represents.
- If there are so many endpoints for a Service that a threshold is reached, then Kubernetes adds another empty EndpointSlice and stores new endpoint information there.
- See EndpointSlices for more information about this API

Service types

- For some parts of your application you may want to expose a Service onto an external IP address, one that's accessible from outside of your cluster.
- Kubernetes Service types allow you to specify what kind of Service you want.
- The available type values and their behaviors are:
  1. ClusterIP
  2. NodePort
  3. LoadBalancer
  4. ExternalName

Service types: ClusterIP

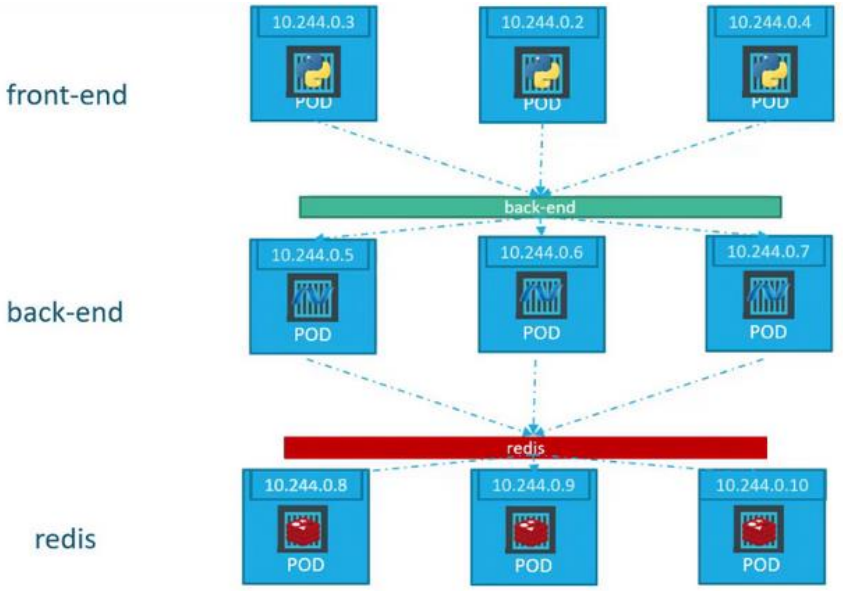
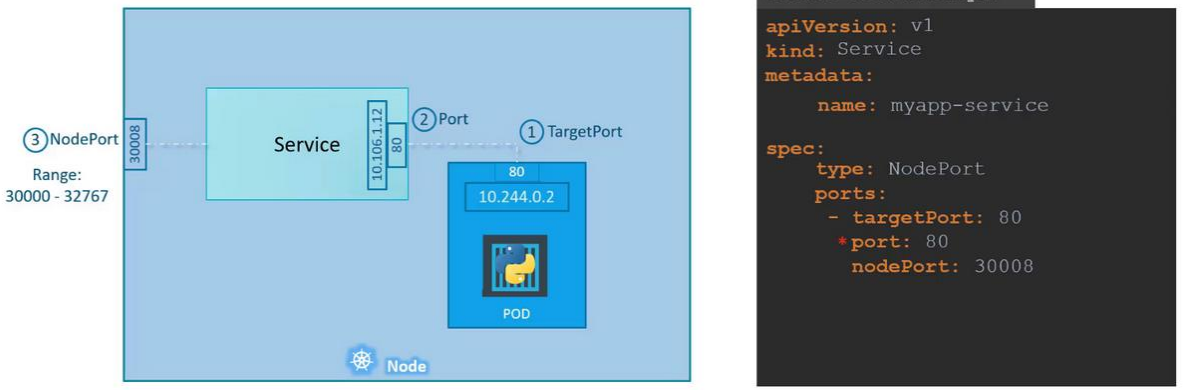


Image source: Certified Kubernetes Administrator (CKA) course on KodeKloud

- ClusterIP exposes the Service on a cluster-internal IP.
- Choosing this value makes the Service only reachable from within the cluster.
- This is the default that is used if you don't explicitly specify a type for a Service.
- You can expose the Service to the public internet using an Ingress or a Gateway.

Service types: NodePort



service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
       port: 80
       nodePort: 30008
```

- NodePort exposes the Service on each Node's IP at a static port (the NodePort).
- To make the node port available, Kubernetes sets up a cluster IP address, the same as if you had requested a Service of type: ClusterIP.
  - This makes the service accessible internally
- Kubernetes control plane allocates a port from a range specified by --service-node-port-range flag (default: 30000-32767).

Image source: Certified Kubernetes Administrator (CKA) course on KodeKloud

Service types: LoadBalancer

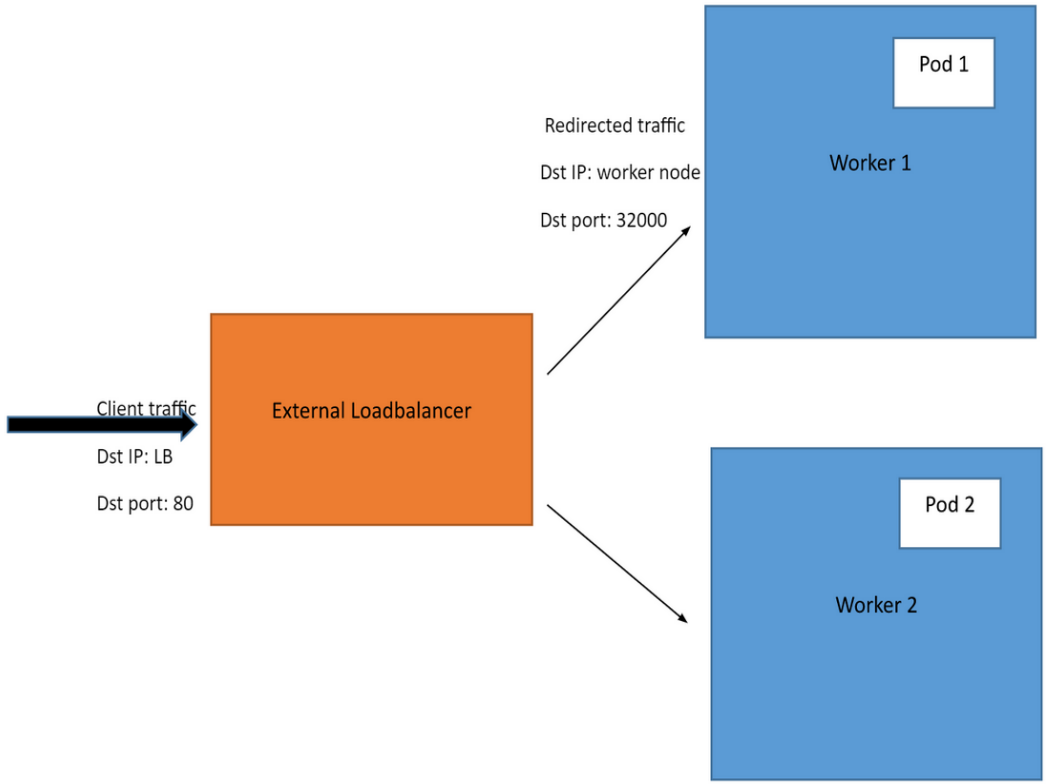


Image source: KodeKloud blog explaining "What Is Kubernetes Ingress?"

- LoadBalancer exposes the Service externally using an external load balancer.
- Kubernetes does not directly offer a load balancing component; you must provide one, or you can integrate your Kubernetes cluster with a cloud provider.
- The actual creation of the load balancer happens asynchronously, and information about the provisioned balancer is published in the Service's .status.loadBalancer field

Service types: ExternalName

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

- ExternalName maps the Service to the contents of the externalName field (for example, to the hostname api.foo.bar.example). The mapping configures your cluster's DNS server to return a CNAME record with that external hostname value. No proxying of any kind is set up.

Headless Services

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  clusterIP: None
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Image source: Medium blog explaining "Headless Kubernetes Service"

- When there is no need of load-balancing and a single Service IP, a *headless Services*, by explicitly specifying "None" for the cluster IP address (.spec.clusterIP).
- **With selectors**: the endpoints controller creates EndpointSlices in the Kubernetes API, and modifies the DNS configuration to return A or AAAA records (IPv4 or IPv6 addresses) that point directly to the Pods backing the Service.
- **Without selectors**: the control plane does not create EndpointSlice objects. However, the DNS system looks for and configures either.
  - DNS CNAME records for type: ExternalName Services.
  - DNS A / AAAA records for all IP addresses of the Service's ready endpoints, for all Service types other than ExternalName.
    - For IPv4 endpoints, the DNS system creates A records.
    - For IPv6 endpoints, the DNS system creates AAAA records.

Summary

- Kubernetes Services abstract the identity of Pods from their accessibility by providing network exposure.
- Different types of Service
  1. ClusterIP
  2. NodePort
  3. LoadBalancer
  4. ExternalName

- While LoadBalancer offers comprehensive traffic management, it can become costly when deploying multiple services. Kubernetes Ingress proves advantageous in such scenarios, providing both cost efficiency and logic for redirecting traffic to specific services."

Questions