Exposing applications using services

AUTOPILOT (/KUBERNETES-ENGINE/DOCS/CONCEPTS/AUTOPILOT-OVERVIEW)

STANDARD (/KUBERNETES-ENGINE/DOCS/CONCEPTS/TYPES-OF-CLUSTERS)

This page shows how to create Kubernetes Services in a Google Kubernetes Engine (GKE) cluster. For an explanation of the Service concept and a discussion of the various types of Services, see Service (/kubernetes-engine/docs/concepts/service).

Introduction

The idea of a Service (https://kubernetes.io/docs/concepts/services-networking/service/) is to group a set of Pod endpoints into a single resource. You can configure various ways to access the grouping. By default, you get a stable cluster IP address that clients inside the cluster can use to contact Pods in the Service. A client sends a request to the stable IP address, and the request is routed to one of the Pods in the Service.

There are five types of Services:

- ClusterIP (default)
- NodePort
- LoadBalancer
- ExternalName
- Headless

Autopilot clusters are public by default. If you opt for a private (/kubernetes-engine/docs/concepts/private-cluster-concept) Autopilot cluster, you must configure Cloud NAT (/nat/docs/set-up-network-address-translation) to make outbound internet connections, for example pulling images from DockerHub.

This topic has several exercises. In each exercise, you create a Deployment and expose its Pods by creating a Service. Then you send an HTTP request to the Service.

Before you begin

Before you start, make sure you have performed the following tasks:

- Enable the Google Kubernetes Engine API.

  Enable Google Kubernetes Engine API (https://console.cloud.google.com/flows/enableapi?apiid=container.googleapis.com)

- If you want to use the Google Cloud CLI for this task, install (/sdk/docs/install) and then initialize (/sdk/docs/initializing) the gcloud CLI. If you previously installed the gcloud CLI, get the latest version by running `gcloud components update`.

★ **Note:** For existing gcloud CLI installations, make sure to set the **`compute/region`** and **`compute/zone`** properties (/sdk/docs/properties#setting_properties). By setting default locations, you can avoid errors in gcloud CLI like the following: **`One of [--zone, --region] must be supplied: Please specify location`**.

* Create a GKE cluster (/kubernetes-engine/docs/how-to/creating-an-autopilot-cluster).

Creating a Service of type ClusterIP

In this section, you create a Service of type `ClusterIP` (/kubernetes-engine/docs/concepts/service#services_of_type_clusterip).

kubectl applyConsole (#console)
(#kubectl-apply)

Here is a manifest for a Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  selector:
    matchLabels:
      app: metrics
      department: sales
  replicas: 3
  template:
    metadata:
      labels:
        app: metrics
        department: sales
    spec:
      containers:
      - name: hello
        image: "us-docker.pkg.dev/google-samples/containers/gke/hello-app:2.0"
```

Copy the manifest to a file named `my-deployment.yaml`, and create the Deployment:

```
kubectl apply -f my-deployment.yaml
```

Verify that three Pods are running:

```
kubectl get pods
```

The output shows three running Pods:

```
NAME                             READY   STATUS    RESTARTS   AGE
my-deployment-dbd86c8c4-h5wsf    1/1     Running   0          7s
my-deployment-dbd86c8c4-qfw22    1/1     Running   0          7s
my-deployment-dbd86c8c4-wt4s6    1/1     Running   0          7s
```

Here is a manifest for a Service of type `ClusterIP`:

```
apiVersion: v1
kind: Service
metadata:
  name: my-cip-service
spec:
  type: ClusterIP
  # Uncomment the below line to create a Headless Service
  # clusterIP: None
  selector:
    app: metrics
    department: sales
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

The Service has a selector that specifies two labels:

- `app: metrics`
- `department: sales`

Each Pod in the Deployment that you created previously has those two labels. So the Pods in the Deployment will become members of this Service.

Copy the manifest to a file named `my-cip-service.yaml`, and create the Service:

```
kubectl apply -f my-cip-service.yaml
```

Wait a moment for Kubernetes to assign a stable internal address to the Service, and then view the Service:

```
kubectl get service my-cip-service --output yaml
```

The output shows a value for `clusterIP`:

```
spec:
  clusterIP: 10.59.241.241
```

Make a note of your `clusterIP` value for later.

Accessing your Service

List your running Pods:

```
kubectl get pods
```

In the output, copy one of the Pod names that begins with `my-deployment`.

```
NAME                             READY   STATUS    RESTARTS   AGE
my-deployment-dbd86c8c4-h5wsf    1/1     Running   0          2m51s
```

Get a shell into one of your running containers:

```
kubectl exec -it POD_NAME -- sh
```

Replace *`POD_NAME`* with the name of one of the Pods in `my-deployment`.

In your shell, install `curl`:

```
apk add --no-cache curl
```

In the container, make a request to your Service by using your cluster IP address and port 80. Notice that 80 is the value of the `port` field of your Service. This is the port that you use as a client of the Service.

```
curl CLUSTER_IP:80
```

Replace *`CLUSTER_IP`* with the value of `clusterIP` in your Service.

Your request is forwarded to one of the member Pods on TCP port 8080, which is the value of the `targetPort` field. Note that each of the Service's member Pods must have a container listening on port 8080.

The response shows the output of `hello-app`:

```
Hello, world!
Version: 2.0.0
Hostname: my-deployment-dbd86c8c4-h5wsf
```

To exit the shell to your container, enter `exit`.

Note: You need to know ahead of time that each of your member Pods has a container listening on TCP port 8080. In this exercise, you did not do anything to make the containers listen on port 8080. You can see that **`hello-app`** listens on port 8080 by looking at the Dockerfile and the source code (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/tree/main/quickstarts/hello-app) for the app.

Creating a Service of type NodePort

In this section, you create a Service of type `NodePort` (/kubernetes-engine/docs/concepts/service#service_of_type_nodeport).

kubectl applyConsole (#console)
(#kubectl-apply)

Here is a manifest for a Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment-50000
spec:
  selector:
    matchLabels:
      app: metrics
      department: engineering
  replicas: 3
  template:
    metadata:
      labels:
        app: metrics
        department: engineering
    spec:
      containers:
      - name: hello
        image: "us-docker.pkg.dev/google-samples/containers/gke/hello-app:2.0"
        env:
        - name: "PORT"
          value: "50000"
```

Notice the `env` object in the manifest. The `env` object specifies that the `PORT` environment variable for the running container will have a value of `50000`. The `hello-app` application listens on the port specified by the `PORT` environment variable. So in this exercise, you are telling the container to listen on port 50000.

Copy the manifest to a file named `my-deployment-50000.yaml`, and create the Deployment:

```
kubectl apply -f my-deployment-50000.yaml
```

Verify that three Pods are running:

```
kubectl get pods
```

Here is a manifest for a Service of type NodePort:

```
apiVersion: v1
kind: Service
metadata:
  name: my-np-service
spec:
  type: NodePort
  selector:
    app: metrics
    department: engineering
  ports:
  - protocol: TCP
```

```
    port: 80
    targetPort: 50000
```

Copy the manifest to a file named `my-np-service.yaml`, and create the Service:

```
kubectl apply -f my-np-service.yaml
```

View the Service:

```
kubectl get service my-np-service --output yaml
```

The output shows a `nodePort` value:

```
...
  spec:
    ...
    ports:
    - nodePort: 30876
      port: 80
      protocol: TCP
      targetPort: 50000
    selector:
      app: metrics
      department: engineering
    sessionAffinity: None
    type: NodePort
...
```

Create a firewall rule to allow TCP traffic on your node port:

```
gcloud compute firewall-rules create test-node-port \
    --allow tcp:NODE_PORT
```

Replace *`NODE_PORT`* with the value of the `nodePort` field of your Service.

Get a node IP address

Find the external IP address of one of your nodes:

```
kubectl get nodes --output wide
```

The output is similar to the following:

```
NAME          STATUS    ROLES     AGE    VERSION         EXTERNAL-IP
gke-svc-...   Ready     none      1h     v1.9.7-gke.6    203.0.113.1
```

Not all clusters have external IP addresses for nodes. For example, the nodes in private clusters (/kubernetes-engine/docs/how-to/private-clusters) do not have external IP addresses.

Access your Service

In your browser's address bar, enter the following:

```
NODE_IP_ADDRESS:NODE_PORT
```

Replace the following:

- *`NODE_IP_ADDRESS`*: the external IP address of one of your nodes, found when creating the service in the previous task.
- *`NODE_PORT`*: your node port value.

The output is similar to the following:

```
Hello, world!
Version: 2.0.0
Hostname: my-deployment-50000-6fb75d85c9-g8c4f
```

Creating a Service of type LoadBalancer

In this section, you create a Service of type **`LoadBalancer`** (/kubernetes-engine/docs/concepts/service#services_of_type_loadbalancer).

kubectl apply Console (#console)
(#kubectl-apply)

Here is a manifest for a Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment-50001
spec:
  selector:
    matchLabels:
      app: products
      department: sales
  replicas: 3
  template:
    metadata:
      labels:
        app: products
        department: sales
    spec:
      containers:
      - name: hello
        image: "us-docker.pkg.dev/google-samples/containers/gke/hello-app:2.0"
        env:
        - name: "PORT"
          value: "50001"
```

Notice that the containers in this Deployment will listen on port 50001.

Copy the manifest to a file named **`my-deployment-50001.yaml`**, and create the Deployment:

```
kubectl apply -f my-deployment-50001.yaml
```

Verify that three Pods are running:

```
kubectl get pods
```

Here is a manifest for a Service of type **`LoadBalancer`**:

```
apiVersion: v1
kind: Service
metadata:
  name: my-lb-service
spec:
  type: LoadBalancer
```

```
  selector:
    app: products
    department: sales
  ports:
  - protocol: TCP
    port: 60000
    targetPort: 50001
```

Copy the manifest to a file named `my-lb-service.yaml`, and create the Service:

```
kubectl apply -f my-lb-service.yaml
```

When you create a Service of type `LoadBalancer`, a Google Cloud controller wakes up and configures an external passthrough Network Load Balancer (/load-balancing/docs/network). Wait a minute for the controller to configure the external passthrough Network Load Balancer and generate a stable IP address.

View the Service:

```
kubectl get service my-lb-service --output yaml
```

The output shows a stable external IP address under `loadBalancer:ingress`:

```
...
spec:
  ...
  ports:
  - ...
    port: 60000
    protocol: TCP
    targetPort: 50001
  selector:
    app: products
    department: sales
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 203.0.113.10
```

Access your Service

Wait a few minutes for GKE to configure the load balancer.

In your browser's address bar, enter the following:

```
LOAD_BALANCER_ADDRESS:60000
```

Replace *`LOAD_BALANCER_ADDRESS`* with the external IP address of your load balancer.

The response shows the output of `hello-app`:

```
Hello, world!
Version: 2.0.0
Hostname: my-deployment-50001-68bb7dfb4b-prvct
```

Notice that the value of `port` in a Service is arbitrary. The preceding example demonstrates this by using a `port` value of 60000.

Creating a Service of type ExternalName

In this section, you create a Service of type `ExternalName` (/kubernetes-engine/docs/concepts/service#service_of_type_externalname).

A Service of type `ExternalName` provides an internal alias for an external DNS name. Internal clients make requests using the internal DNS name, and the requests are redirected to the external name.

Here is a manifest for a Service of type `ExternalName`:

```
apiVersion: v1
kind: Service
metadata:
  name: my-xn-service
spec:
  type: ExternalName
  externalName: example.com
```

In the preceding example, the DNS name is my-xn-service.default.svc.cluster.local. When an internal client makes a request to my-xn-service.default.svc.cluster.local, the request gets redirected to example.com.

Using `kubectl expose` to create a Service

As an alternative to writing a Service manifest, you can create a Service by using `kubectl expose` to expose a Deployment.

To expose `my-deployment`, shown earlier in this topic, you could enter this command:

```
kubectl expose deployment my-deployment --name my-cip-service \
    --type ClusterIP --protocol TCP --port 80 --target-port 8080
```

To expose `my-deployment-50000`, show earlier in this topic, you could enter this command:

```
kubectl expose deployment my-deployment-50000 --name my-np-service \
    --type NodePort --protocol TCP --port 80 --target-port 50000
```

To expose `my-deployment-50001`, shown earlier in this topic, you could enter this command:

```
kubectl expose deployment my-deployment-50001 --name my-lb-service \
    --type LoadBalancer --port 60000 --target-port 50001
```

Cleaning up

After completing the exercises on this page, follow these steps to remove resources and prevent unwanted charges incurring on your account:

kubectl applyConsole (#console)
(#kubectl-apply)

Deleting your Services

```
kubectl delete services my-cip-service my-np-service my-lb-service
```

Deleting your Deployments

```
kubectl delete deployments my-deployment my-deployment-50000 my-deployment-50001
```

Deleting your firewall rule

```
gcloud compute firewall-rules delete test-node-port
```

What's next

- Services (/kubernetes-engine/docs/concepts/service)
- StatefulSets (/kubernetes-engine/docs/concepts/statefulset)
- Ingress (/kubernetes-engine/docs/concepts/ingress)
- HTTP Load Balancing with Ingress (/kubernetes-engine/docs/tutorials/http-balancer)

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see the Google Developers Site Policies (https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2023-12-04 UTC.