Create an External Load Balancer

This page shows how to create an external load balancer.

When creating a Service, you have the option of automatically creating a cloud load balancer. This provides an externally-accessible IP address that sends traffic to the correct port on your cluster nodes, *provided your cluster runs in a supported environment and is configured with the correct cloud load balancer provider package*.

You can also use an Ingress in place of Service. For more information, check the Ingress documentation.

Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:

- Killercoda
- Play with Kubernetes

Your cluster must be running in a cloud or other environment that already has support for configuring external load balancers.

Create a Service

Create a Service from a manifest

To create an external load balancer, add the following line to your Service manifest:

```
    type: LoadBalancer
```

Your manifest might then look like:

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
    - port: 8765
      targetPort: 9376
  type: LoadBalancer
```

Create a Service using kubectl

You can alternatively create the service with the `kubectl expose` command and its `--type=LoadBalancer` flag:

```
kubectl expose deployment example --port=8765 --target-port=9376 \
        --name=example-service --type=LoadBalancer
```

This command creates a new Service using the same selectors as the referenced resource (in the case of the example above, a Deployment named `example`).

For more information, including optional flags, refer to the `kubectl expose` reference.

Finding your IP address

You can find the IP address created for your service by getting the service information through `kubectl`:

```
kubectl describe services example-service
```

which should produce output similar to:

```
Name:                     example-service
Namespace:                default
Labels:                   app=example
Annotations:              <none>
Selector:                 app=example
Type:                     LoadBalancer
IP Families:              <none>
IP:                       10.3.22.96
IPs:                      10.3.22.96
LoadBalancer Ingress:     192.0.2.89
Port:                     <unset>  8765/TCP
TargetPort:               9376/TCP
NodePort:                 <unset>  30593/TCP
Endpoints:                172.17.0.3:9376
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

The load balancer's IP address is listed next to `LoadBalancer Ingress`.

Note:
If you are running your service on Minikube, you can find the assigned IP address and port with:

```
minikube service example-service --url
```

Preserving the client source IP

By default, the source IP seen in the target container is *not the original source IP* of the client. To enable preservation of the client IP, the following fields can be configured in the `.spec` of the Service:

- `.spec.externalTrafficPolicy` - denotes if this Service desires to route external traffic to node-local or cluster-wide endpoints. There are two available options: `Cluster` (default) and `Local`. `Cluster` obscures the client source IP and may cause a second hop to another node, but should have good overall load-spreading. `Local` preserves the client source IP and avoids a second hop for LoadBalancer and NodePort type Services, but risks potentially imbalanced traffic spreading.
- `.spec.healthCheckNodePort` - specifies the health check node port (numeric port number) for the service. If you don't specify `healthCheckNodePort`, the service controller allocates a port from your cluster's NodePort range.
  You can configure that range by setting an API server command line option, `--service-node-port-range`. The Service will use the user-specified `healthCheckNodePort` value if you specify it, provided that the Service `type` is set to LoadBalancer and `externalTrafficPolicy` is set to `Local`.

Setting `externalTrafficPolicy` to Local in the Service manifest activates this feature. For example:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
    - port: 8765
      targetPort: 9376
  externalTrafficPolicy: Local
  type: LoadBalancer
```

Caveats and limitations when preserving source IPs

Load balancing services from some cloud providers do not let you configure different weights for each target.

With each target weighted equally in terms of sending traffic to Nodes, external traffic is not equally load balanced across different Pods. The external load balancer is unaware of the number of Pods on each node that are used as a target.

Where `NumServicePods << NumNodes` or `NumServicePods >> NumNodes`, a fairly close-to-equal distribution will be seen, even without weights.

Internal pod to pod traffic should behave similar to ClusterIP services, with equal probability across all pods.

Garbage collecting load balancers

FEATURE STATE: `Kubernetes v1.17 [stable]`

In usual case, the correlating load balancer resources in cloud provider should be cleaned up soon after a LoadBalancer type Service is deleted. But it is known that there are various corner cases where cloud resources are orphaned after the associated Service is deleted. Finalizer Protection for Service LoadBalancers was introduced to prevent this from happening. By using finalizers, a Service resource will never be deleted until the correlating load balancer resources are also deleted.

Specifically, if a Service has `type` LoadBalancer, the service controller will attach a finalizer named `service.kubernetes.io/load-balancer-cleanup`. The finalizer will only be removed after the load balancer resource is cleaned up. This prevents dangling load balancer resources even in corner cases such as the service controller crashing.

External load balancer providers

It is important to note that the datapath for this functionality is provided by a load balancer external to the Kubernetes cluster.

When the Service `type` is set to LoadBalancer, Kubernetes provides functionality equivalent to `type` equals ClusterIP to pods within the cluster and extends it by programming the (external to Kubernetes) load balancer with entries for the nodes hosting the relevant Kubernetes pods. The Kubernetes control plane automates the creation of the external load balancer, health checks (if needed), and packet filtering rules (if needed). Once the cloud provider allocates an IP address for the load balancer, the control plane looks up that external IP address and populates it into the Service object.

What's next

- Follow the Connecting Applications with Services tutorial
- Read about Service
- Read about Ingress

Feedback

Was this page helpful?

Yes No

Last modified October 22, 2023 at 6:49 PM PST: Fixed a typo on page "Create an External Load Balancer" (28c7a312af)