Accessing the Cluster: Adding the Load Balancer

In this lesson, we will start with the creation of resources based on a kops add-on file and verify the addition of a new load balancer to our cluster.

WE'LL COVER THE FOLLOWING

kops Add-Ons

kops has a solution for the addition of a load balancer. We can use kops' add-ons to deploy additional core services. You can get the list of those currently available by exploring directories here. Even though most of them are useful, we'll focus only on the task at hand.

Add-ons are, in most cases, Kubernetes resources defined in a YAML file. All we have to do is pick the addon we want, choose the version we prefer, and execute `kubectl create`.

We'll create the resources defined in `ingress-nginx` version `v1.6.0`.

Creating Resources

We won't go into details behind the definition YAML file we are about to use to create the resources kops assembled for us. Instead, we'll proceed with `kubectl create`.

```
kubectl create \
    -f https://raw.githubusercontent.com/kubernetes/kops/master/addons/ingress-nginx/v1.6.0.y
```

The **output** is as follows.

```
namespace "kube-ingress" created
serviceaccount "nginx-ingress-controller" created
clusterrole "nginx-ingress-controller" created
role "nginx-ingress-controller" created
clusterrolebinding "nginx-ingress-controller" created
rolebinding "nginx-ingress-controller" created
service "nginx-default-backend" created
deployment "nginx-default-backend" created
configmap "ingress-nginx" created
service "ingress-nginx" created
deployment "ingress-nginx" created
```

We can see that quite a few resources were created in the Namespace `kube-ingress`. Let's take a look what's inside.

```
kubectl --namespace kube-ingress \
    get all
```

The **output** is as follows.

```
NAME                          DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
deploy/ingress-nginx          3       3       3          3         1m
deploy/nginx-default-backend 1       1       1          1         1m
NAME                                  DESIRED CURRENT READY AGE
rs/ingress-nginx-768fc7997b          3       3       3     1m
rs/nginx-default-backend-74f9cd546d 1       1       1     1m
NAME                                       READY STATUS  RESTARTS AGE
po/ingress-nginx-768fc7997b-4xfq8          1/1   Running 0        1m
po/ingress-nginx-768fc7997b-c7zvx          1/1   Running 0        1m
po/ingress-nginx-768fc7997b-clr5m          1/1   Running 0        1m
po/nginx-default-backend-74f9cd546d-mtct8 1/1   Running 0        1m
NAME                       TYPE         CLUSTER-IP     EXTERNAL-IP      PORT(S)
svc/ingress-nginx          LoadBalancer 100.66.190.165 abb5117871831... 80:31895/TCP,443:32697
svc/nginx-default-backend ClusterIP    100.70.227.240 <none>           80/TCP
```

We can see that it created two deployments, which created two ReplicaSets, which created Pods. In addition, we got two Services as well. As a result, Ingress is running inside our cluster and are a step closer to being able to test it. Still, we need to figure out how to access the cluster.

> The above output may have some formatting differences due to continuous version upgrades.

The Load Balancer

The Load Balancer

One of the two Services (`ingress-nginx`) is `LoadBalancer`. We did not explore that type when we discussed Services.

LoadBalancer Service type exposes the service externally using a cloud provider's load balancer. NodePort and ClusterIP services, to which the external load balancer will route, are automatically created. Ingress is "intelligent" enough to know how to create and configure an AWS ELB. All it needed is an annotation `service.beta.kubernetes.io/aws-load-balancer-proxy-protocol` (defined in the YAML file).

You'll notice that the `ingress-nginx` Service published port `31895` and mapped it to `80`. `32697` was mapped to `443`. That means that, from inside the cluster, we should be able to send HTTP requests to `31895` and HTTPS to `32697`. However, that is only part of the story. Since the Service is the `LoadBalancer` type, we should expect some changes to AWS Elastic Load Balancers (ELBs) as well.

Verification

Let's check the state of the load balancers in our cluster.

```
aws elb describe-load-balancers
```

The **output**, limited to the relevant parts, is as follows.

```
{
  "LoadBalancerDescriptions": [
    {
      ...
      "LoadBalancerName": "api-devops23-k8s-local-ivnbim",
      ...
    },
    {
      ...
      "ListenerDescriptions": [
        {
          "Listener": {
            "InstancePort": 31895,
            "LoadBalancerPort": 80,
            "Protocol": "TCP",
            "InstanceProtocol": "TCP"
          },
          "PolicyNames": []
        },
```

```
      {
        "Listener": {
          "InstancePort": 32697,

          "LoadBalancerPort": 443,
          "Protocol": "TCP",
          "InstanceProtocol": "TCP"
        },
        "PolicyNames": []
      }
    ],
    ...
    "Instances": [
      {
        "InstanceId": "i-063fabc7ad5935db5"
      },
      {
        "InstanceId": "i-04d32c91cfc084369"
      }
    ],
    "DNSName": "a1c431cef1bfa11e88b600650be36f73-2136831960.us-east-2.elb.amazonaws.com",
    ...
    "LoadBalancerName": "a1c431cef1bfa11e88b600650be36f73",
    ...
```

We can observe from the output that a new load balancer was added.

The new load balancer publishes port `80` (HTTP) and maps it to `31895`. That port is the same the `ingress-nginx` Service published. Similarly, the LB published port `443` (HTTPS) and mapped it to `32697`. From the `Instances` section, we can see that it currently maps to the two worker nodes.

Further down, we can see the `DNSName`. We should retrieve it but, unfortunately, `LoadBalancerName` does not follow any format. However, we do know that now there are two load balancers and that the one dedicated to masters has a name that starts with `api-devops23`. So, we can retrieve the other LB by specifying that it should not contain that prefix. We'll use `jq`'s instruction `not` for that.

The command that retrieves DNS from the new load balancer is as follows.

```
CLUSTER_DNS=$(aws elb \
    describe-load-balancers | jq -r \
    ".LoadBalancerDescriptions[] \
    | select(.DNSName \
    | contains (\"api-devops23\") \
    | not).DNSName")
```

We'll come back to the newly created Ingress and the load balancer soon.

In the next lesson, we'll move on and deploy the `go-demo-2` application.