Exploring the High-Availability and Fault-Tolerance

In this lesson, we will explore the high-availability and fault tolerance of our cluster.

WE'LL COVER THE FOLLOWING

The cluster would not be reliable if it wouldn't be fault tolerant. Kops intents to make it so, but we're going to validate that anyways.

Terminating a Worker Node

Let's retrieve the list of worker node instances.

```
aws ec2 \
    describe-instances | jq -r \
    ".Reservations[].Instances[] \
    | select(.SecurityGroups[]\
    .GroupName==\"nodes.$NAME\")\
    .InstanceId"
```

We used `aws ec2 describe-instances` to retrieve all the instances (five in total). The output was sent to `jq`, which filtered them by the security group dedicated to worker nodes.

The **output** is as follows.

```
i-063fabc7ad5935db5
i-04d32c91cfc084369
```

We'll terminate one of the worker nodes. To do that, we'll pick a random one, and retrieve its ID.

```
INSTANCE_ID=$(aws ec2 \
    describe-instances | jq -r \
    ".Reservations[].Instances[] \

    | select(.SecurityGroups[]\
    .GroupName==\"nodes.$NAME\")\
    .InstanceId" | tail -n 1)
```

We used the same command as before and added `tail -n 1`, so that the output is limited to a single line (entry). We stored the result in the `INSTANCE_ID` variable. Now we know which instance to terminate.

```
aws ec2 terminate-instances \
    --instance-ids $INSTANCE_ID
```

The **output** is as follows.

```
{
  "TerminatingInstances": [
    {
      "InstanceId": "i-063fabc7ad5935db5",
      "CurrentState": {
        "Code": 32,
        "Name": "shutting-down"
     },
     "PreviousState": {
        "Code": 16,
        "Name": "running"
     }
    }
  ]
}
```

We can see from the output that the instance is shutting down. We can confirm that by listing all the instances from the security group `nodes.devops23.k8s.local`.

```
aws ec2 describe-instances | jq -r \
    ".Reservations[].Instances[] \
    | select(\
    .SecurityGroups[].GroupName \
    ==\"nodes.$NAME\").InstanceId"
```

The **output** is as follows.

```
i-04d32c91cfc084369
```

As expected, we are now running only one instance.

Exploring the Effects

All that's left is to wait for a minute, and repeat the same command.

```
aws ec2 \
    describe-instances | jq -r \
    ".Reservations[].Instances[] \
    | select(.SecurityGroups[]\
    .GroupName==\"nodes.$NAME\")\
    .InstanceId"
```

The **output** is as follows.

```
i-003b4b1934d85641a
i-04d32c91cfc084369
```

This time, we can see that there are again two instances. The only difference is that this time one of the instance IDs is different.

AWS auto-scaling group discovered that the instances do not match the desired number, and it created a new one.

The fact that AWS created a node to replace the one we terminated does not mean that the new server joined the Kubernetes cluster. Let's verify that.

```
kubectl get nodes
```

The **output** is as follows.

```
NAME                                         STATUS ROLES  AGE VERSION
ip-172-20-55-183.us-east-2.compute.internal Ready  master 30m v1.9.3
ip-172-20-61-82.us-east-2.compute.internal  Ready  node   13m v1.9.3
ip-172-20-71-53.us-east-2.compute.internal  Ready  master 30m v1.9.3
ip-172-20-97-39.us-east-2.compute.internal  Ready  master 30m v1.9.3
```

If you were fast enough, your output should also show that there is only one (worker) `node`. Once AWS created a new server, it takes a bit of time until Docker, Kubelet, and Protokube are installed, containers are pulled and run, and the node is registered through one of the masters.

Let's try it again.

```
kubectl get nodes
```

The **output** is as follows.

```
NAME                                          STATUS ROLES  AGE VERSION
ip-172-20-55-183.us-east-2.compute.internal Ready  master 32m v1.9.3
ip-172-20-61-82.us-east-2.compute.internal  Ready  node   15m v1.9.3
ip-172-20-71-53.us-east-2.compute.internal  Ready  master 32m v1.9.3
ip-172-20-79-161.us-east-2.compute.internal Ready  node   2m  v1.9.3
ip-172-20-97-39.us-east-2.compute.internal  Ready  master 32m v1.9.3
```

This time, the number of (worker) nodes is back to two. Our cluster is back in the desired state.

What we just experienced is, basically, the same as when we executed the rolling upgrade. The only difference is that we terminated an instance as a way to simulate a failure. During the upgrade process, kops does the same. It shuts down one instance at a time and waits until the cluster goes back to the desired state.

Feel free to do a similar test with master nodes. The only difference is that you'll have to use `masters` instead of `nodes` as the prefix of the security group name. Since everything else is the same, most probably you won't need instructions and explanations.

In the next lesson, we will allow the users to access our cluster.