Updating the Cluster

In this lesson, we will change and update the configuration of our cluster.

WE'LL COVER THE FOLLOWING

Editing the Definition

With kops, we cannot update the cluster directly. Instead, we edit the desired state of the cluster stored, in our case, in the S3 bucket. Once the state is changed, kops will make the necessary changes to comply with the new desire.

We'll try to update the cluster so that the number of worker nodes is increased from one to two. In other words, we want to add one more server to the cluster.

The kops `edit` Commands

Let's see the sub-commands provided through `kops edit`.

```
kops edit --help
```

The **output**, limited to the available commands, is as follows.

```
...
Available Commands:
  cluster       Edit cluster.
  instancegroup Edit instancegroup
  instancegroup edit instancegroup.
...
```

We have **two types** of edits we can make.

Edit Cluster

You might think that `cluster` would provide the possibility to create a new worker node. However, that is not the case. If you execute `kops edit cluster --name $NAME`, you'll see that nothing in the configuration indicates how many nodes we should have. That is normal considering that we should not create servers in AWS directly. Just as Kubernetes, AWS also prefers a declarative approach over imperative. At least, when dealing with EC2 instances.

Edit Instance Group

Instead of sending an imperative instruction to create a new node, we'll change the value of the Auto-Scaling Group (ASG) related to worker nodes. Once we change ASG values, AWS will make sure that it complies with the new desire. It'll not only create a new server to comply with the new ASG sizes, but it will also monitor EC2 instances and maintain the desired number in case one of them fails.

So, we'll choose the third `kops edit` option.

```
kops edit ig --name $NAME nodes
```

We executed `kops edit ig` command, where `ig` is one of the aliases of `instancegroup`. We specified the name of the cluster with the `--name` argument. Finally, we set the type of the servers to `nodes`. As a result, we are presented with the `InstanceGroup` config for the Auto-Scaling Group associated with worker nodes.

The **output** is as follows.

```
apiVersion: kops/v1alpha2
kind: InstanceGroup
metadata:
  creationTimestamp: 2018-02-23T00:04:50Z
  labels:
    kops.k8s.io/cluster: devops23.k8s.local
  name: nodes
spec:
  image: kope.io/k8s-1.8-debian-jessie-amd64-hvm-ebs-2018-01-14
  machineType: t2.small
  maxSize: 1
  minSize: 1
  nodeLabels:
    kops.k8s.io/instancegroup: nodes
  role: Node
  subnets:
  - us-east-2a
  - us-east-2b
  - us-east-2c
```

Bear in mind that what you're seeing on the screen is not the standard output (`stdout`). Instead, the configuration is opened in your default editor. In our case, that is `vi`.

We can see some useful information from this config.

Line 9: The `image` used to create EC2 instances is based on Debian. It is custom made for kops.

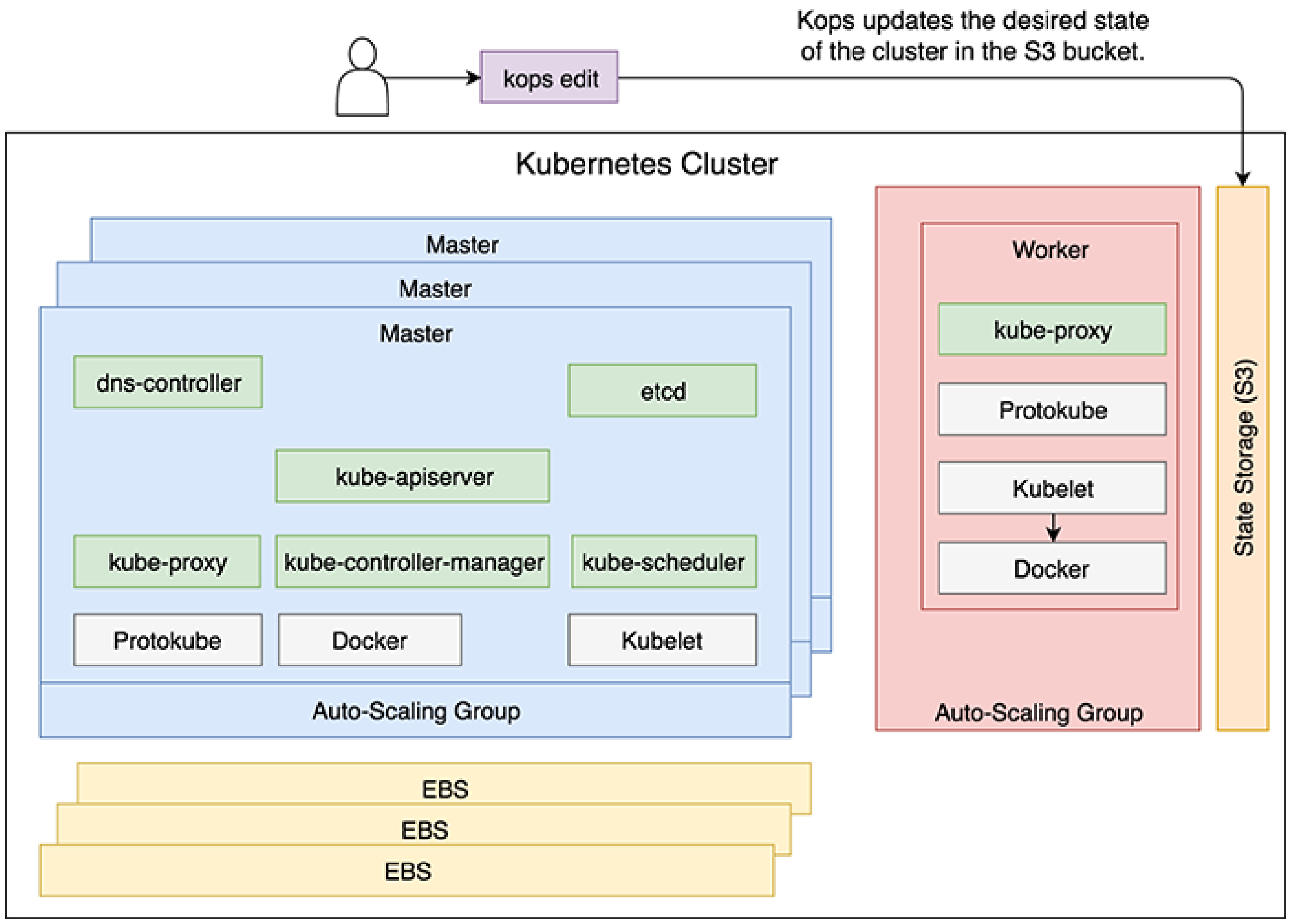
Line 10: The `machineType` represents EC2 size which is set to `t2.small`.

Line 17-19: We're running the VMs in three subnets or since we're in AWS, three availability zones.

Line 11-12: The parts of the config we care about are the `spec` entries `maxSize` and `minSize`. Both are set to `1` since that is the number of worker nodes we specified when we created the cluster. Please change the values of those two entries to `2`, *save, and exit*.

> If you're using `vi` as your default editor, you'll need to press `i` to enter into the `insert` mode. From there on, you can change the values. Once you're finished editing, please press the `ESC` key, followed by `:wq`. Colon (`:`) allows us to enter commands, `w` is translated to save, and `q` to quit. Don't forget to press the enter key.

The below figure illustrates the working of `kops edit` command.



The process behind the `kops edit` command

Updating the Definition

Now that we changed the configuration, we need to tell kops that we want it to update the cluster to comply with the new desired state.

```
kops update cluster --name $NAME --yes
```

The **output**, limited to the last few lines, is as follows.

```
...
kops has set your kubectl context to devops23.k8s.local

Cluster changes have been applied to the cloud.


Changes may require instances to restart: kops rolling-update cluster
```

We can see that kops set our `kubectl` context to the cluster we updated. There was no need for that since that was already our context, but it did that anyway. Further on, we got the confirmation that the changes `have been applied to the cloud`.

The last sentence is interesting. It informed us that we can use `kops rolling-update`. The `kops update` command applies all the changes to the cluster at once. That can result in downtime.

Actually, the `kops rolling-update` command intends to apply the changes without downtime. It would apply them to one server at the time so that most of the servers are always running. In parallel, Kubernetes would be rescheduling the Pods that were running on the servers that were brought down.

> As long as our applications are scaled, `kops rolling-update` should not produce downtime.

In the next lesson, we will go through the sequential breakdown and verification of the cluster updating process.