Content

1. Prerequisites

There are two important prerequisites before we begin the deployment of Couchbase Autonomous Operator on EKS:

1. You have installed *kubectl* & *AWS CLI* on your local machine as described in the guide.
2. You have AWS account and have setup Amazon EKS cluster as per the EKS Instruction Guide.

In the labs below we will be using us-east-1 as the region and us-east-1a/1b/1c as three availability-zones but you can deploy to any region/zones by making minor changes to YAML files in the examples below.

2. Deploy Couchbase Autonomous Operator

Before we begin with the setup of Couchbase Operator, run 'kubectl get nodes' command from the local machine to confirm EKS cluster is up and running.

```
$ kubectl get nodes

NAME                                STATUS    ROLES     AGE       VERSION
ip-192-168-106-132.ec2.internal   Ready     <none>    110m      v1.11.9
ip-192-168-153-241.ec2.internal   Ready     <none>    110m      v1.11.9
ip-192-168-218-112.ec2.internal   Ready     <none>    110m      v1.11.9
```

After we have tested that we can connect to Kubernetes control plane running on Amazon EKS cluster from our local machine, we can now begin with the steps required to deploy Couchbase Autonomous Operator, which is the glue technology enabling Couchbase Server cluster to be managed by Kubernetes.

2.1. Download Operator package

Let's first begin by downloading the latest Couchbase Autonomous Operator and unzip onto the local machine. Change directory to the operator folder so we can find YAML files we need to deploy Couchbase operator:

```
$ cd couchbase-autonomous-operator-kubernetes_1.2.0-981_linux-x86_64

$ ls

License.txt         couchbase-cli-create-user.yaml  operator-role-binding.yaml  secret.yaml
README.txt          couchbase-cluster.yaml      operator-role.yaml
admission.yaml          crd.yaml            operator-service-account.yaml
bin             operator-deployment.yaml    pillowfight-data-loader.yaml
```

Note: This workshop is tested with Couchbase Autonomous Operator 1.2.

2.2. Create a namespace

Create a namespace that will allow cluster resources to be nicely separated between multiple users. To do that we will use a unique namespace called emart for our deployment and later will use this namespace to deploy Couchbase Cluster.

In your working directory create a namespace.yaml file with this content and save it in the Couchbase operator directory itself:

```
apiVersion: v1
kind: Namespace
metadata:
  name: emart
```

After saving the namespace configuration in a file, run kubectl cmd to create it:

```
$ kubectl create -f namespace.yaml
```

Run get namespace command to confirm it is created successfully:

```
$ kubectl get namespaces

output:

NAME          STATUS    AGE
default       Active    1h
emart         Active    12s
```

From now onwards we will use emart as the namespace for all resource provisioning.

2.3. Create TLS Certificate

We need to generate self-signed certificated which we will use later on to deploy: - Admission Control - Couchbase Autonomous Operator - Couchbase Cluster

Follow steps as described in the guide to generate self-signed certificate. After you are done generating the certs then proceed further.

```
$ cd easy-rsa/easyrsa3
```

Create these two TLS secrets to be used later on in the couchbase-cluster.yaml file during Couchbase Cluster Deployment section. ```

$ kubectl create secret generic couchbase-server-tls --from-file chain.pem \ --from-file pkey.key --namespace emart

secret/couchbase-server-tls created ```

```
$ kubectl create secret generic couchbase-operator-tls --from-file pki/ca.crt \
--namespace emart

secret/couchbase-operator-tls created
```

One more TLS secret to create which will be used during Admission-Control deployment. ```

$ kubectl create secret generic couchbase-operator-admission --from-file tls-cert-file \ --from-file tls-private-key-file --namespace emart

secret/couchbase-operator-admission created `Make sure all the secrets exist under the namespace.` $ kubectl get secret --namespace emart

NAME TYPE DATA AGE couchbase-operator-admission Opaque 2 24s couchbase-operator-tls Opaque 1 41s couchbase-server-tls Opaque 2 56s default-token-7w7vx kubernetes.io/service-account-token 3 94s

```
### 2.4. Install Admission Controller

The admission controller is a required component of the Couchbase Autonomous Operator and needs to be installed separately. The primary pur
pose of the admission controller is to validate Couchbase cluster configuration changes before the Operator acts on them, thus protecting y
our Couchbase deployment (and the Operator) from any accidental damage that might arise from an invalid configuration. For architecture det
ails please visit documentation page on the [Admission Controller](https://docs.couchbase.com/operator/current/install-admission-controller
.html#architecture)

#### Prepare the file

Use the following steps to edit the admission-controller file:

- From Couchbase Operator directory open ```admission.yaml``` file in the editor.
- Change all occurrences of ```namespace: default``` to ```namespace: emart``` which is the namespace we will be using in our deployment.

![](./cb-operator-guide/assets/admission-1.png)

Note: There will be three occurrences  where the replacement will happen (in v1.2 it is at line 47,132, 157).

- Similarly we will change occurrences of ```default.svc``` to ```emart.svc``` as **emart** is the namespace we will be using not default.

![](./cb-operator-guide/assets/admission-2.png)

Note: There will be two occurrences of replacement (in v1.2 it is at line 134, 159).

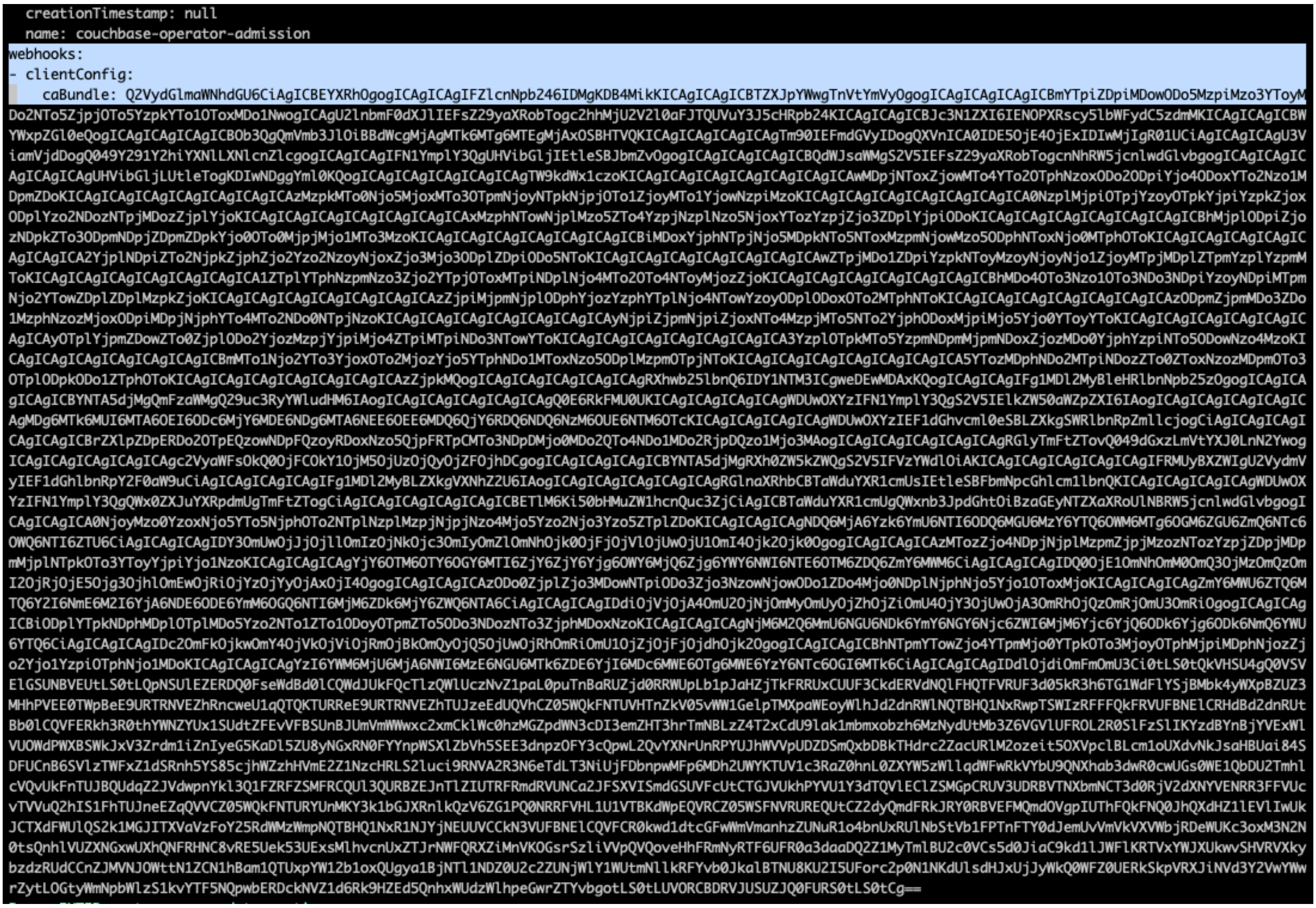
- Delete lines 49-58 as we have already created ```couchbase-operator-admission``` secret and don't need to mention ```tls-cert-file``` and
 ```tls-priavate-key-file```.

![](./cb-operator-guide/assets/admission-3.png)

- Next we will convert the cert file we generated in the TLS section, into base64 encoded file format so we can copy/paste into the YAML fi
le.
```

$ cd easy-rsa/easyrsa3

$ base64 -i tls-cert-file -o ./tls-cert-file-base64 `- Replace all instances of` caBundle `with the content of` tls-cert-file-base64```. There are total of two occurrences in the yaml file.
```



Ready to deploy

Deploy the admission controller using the command:

```
$ kubectl create -f admission.yaml --namespace emart
```

- Confirm the admission controller has deployed successfully:

```
$ kubectl get deployments --namespace emart

NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
couchbase-operator-admission  1         1         1            1           1m
```

2.5. Install CRD

The first step in installing the Operator is to install the custom resource definition (CRD) that describes the CouchbaseCluster resource type. This can be achieved with the following command:

```
kubectl create -f crd.yaml --namespace emart
```

2.6. Create a Operator Role

Next, we will create a cluster role that allows the Operator to access the resources that it needs to run. Since the Operator will manage many different namespaces, it is best to create a cluster role first because you can assign that role to a service account in any namespace.

To create the cluster role for the Operator, run the following command:

```
$ kubectl create -f operator-role.yaml --namespace emart
```

This cluster role only needs to be created once.

2.7. Create a Service Account

After the cluster role is created, you need to create a service account in the namespace where you are installing the Operator. To create the service account:

```
$ kubectl create serviceaccount couchbase-operator --namespace emart
```

Now assign the operator role to the service account:

```
$ kubectl create rolebinding couchbase-operator --role couchbase-operator \
--serviceaccount emart:couchbase-operator --namespace emart

output:

clusterrolebinding.rbac.authorization.k8s.io/couchbase-operator created
```

Now before we proceed further let's make sure all the roles and service accounts are created under the namespace *emart*. To do that run these three checks and make sure each get returns something:

```
Kubectl get roles -n emart
Kubectl get rolebindings -n emart
Kubectl get sa -n emart
```

2.8. Deploy Couchbase Operator

We now have all the roles and privileges for our operator to be deployed. Deploying the operator is as simple as running the operator.yaml file from the Couchbase Autonomous Operator directory.

```
$ kubectl create -f operator-deployment.yaml --namespace emart

output:

deployment.apps/couchbase-operator created
```

Above command will download the Operator Docker image (specified in the operator.yaml file) and creates a deployment, which manages a single instance of the Operator. The Operator uses a deployment so that it can restart if the pod it's running in dies.

It would take less than a minute for Kubernetes to deploy the Operator and for the Operator to be ready to run.

a) Verify the Status of the Deployment

You can use the following command to check on the status of the deployment:

```
  $ kubectl get deployments --namespace emart
```

If you run the this command immediately after the Operator is deployed, the output will look something like the following:

```
NAME                DESIRED    CURRENT     UP-TO-DATE      AVAILABLE        AGE
couchbase-operator   1          1          1               0               10s
```

Note: Above output means your Couchbase operator is deployed and you can go ahead with deploying Couchbase cluster next.

b) Verify the Status of the Operator

You can use the following command to verify that the Operator has started successfully:

```
$ kubectl get pods -l app=couchbase-operator --namespace emart
```

If the Operator is up and running, the command returns an output where the READY field shows 1/1, such as:
`NAME READY STATUS RESTARTS AGE couchbase-operator-8c554cbc7-6vqgf 1/1 Running 0 57s` You can also check the logs to confirm that the Operator is up and running. Look for the message: CRD initialized, listening for events… module=controller.

```
$ kubectl logs couchbase-operator-8c554cbc7-6vqgf --namespace emart --tail 20

output:

time="2019-05-30T23:00:58Z" level=info msg="couchbase-operator v1.2.0 (release)" module=main
time="2019-05-30T23:00:58Z" level=info msg="Obtaining resource lock" module=main
time="2019-05-30T23:00:58Z" level=info msg="Starting event recorder" module=main
time="2019-05-30T23:00:58Z" level=info msg="Attempting to be elected the couchbase-operator leader" module=main
time="2019-05-30T23:00:58Z" level=info msg="I'm the leader, attempt to start the operator" module=main
time="2019-05-30T23:00:58Z" level=info msg="Creating the couchbase-operator controller" module=main
time="2019-05-30T23:00:58Z" level=info msg="Event(v1.ObjectReference{Kind:\"Endpoints\", Namespace:\"emart\", Name:\"couchbase-operator\",
UID:\"c96ae600-832e-11e9-9cec-0e104d8254ae\", APIVersion:\"v1\", ResourceVersion:\"950158\", FieldPath:\"\"}): type: 'Normal' reason: 'Lead
erElection' couchbase-operator-6cbc476d4d-2kps4 became leader" module=event_recorder
```

3. Deploy Couchbase cluster using persistent volumes

In a production environment where performance and SLA of the system matters most, we should always plan on deploying Couchbase cluster using persistent volumes because it helps in:

- **Data Recoverability**: Persistent Volumes allow the data associated within Pods to be recovered in the case that a Pod is terminated. This helps prevent data-loss and avoid time-consuming index building when using the data or index services.
- **Pod Relocation**: Kubernetes may decide to evict pods that reach resource thresholds such as CPU and Memory Limits. Pods that are backed with Persistent Volumes can be terminated and restarted on different nodes without incurring any downtime or data-loss.
- **Dynamic Provisioning**: The Operator will create Persistent Volumes on-demand as your cluster scales, alleviating the need to pre-provision your cluster storage prior to deployment.
- **Cloud Integration**: Kubernetes integrates with native storage provisioners available on major cloud vendors such as AWS and GCE.

In this next section we will see how you can define storage classes in different availability zone and build persistent volume claim template, which will be used in couchbase-cluster-with-pv-1.2.yaml file.

3.1. Create Secret for Couchbase Admin Console

First thing we need to do is create a secret credential which will be used by the administrative web console during login. For convenience, a sample secret is provided in the Operator package. When you push it to your Kubernetes cluster, the secret sets the username to Administrator and the password to password.

To push the secret into your Kubernetes cluster, run the following command:

```
$ kubectl create -f secret.yaml --namespace emart

Output:

Secret/cb-example-auth created
```

3.2 Create AWS storage class for the EKS cluster

Now in order to use PersistentVolume for Couchbase services (data, index, search, etc.), we need to create Storage Classes (SC) first in each of the Availability Zones (AZ). Let's begin by checking what storage classes exist in our environment.

Let's use kubectl command to find that out: ``` $ kubectl get storageclass

Output:

gp2 (default) kubernetes.io/aws-ebs 12m ```

Above output means we just have default gp2 storage class and we need to create separate storage-classes in all of the AZs where we are planning to deploy our Couchbase cluster.

1) Create a AWS storage class manifest file. Below example defines the structure of the storage class (sc-gp2.yaml), which uses the Amazon EBS gp2 volume type (aka general purpose SSD drive). This storage we will later use in our *VolumeClaimTemplate*.

```
For more information about the options available for AWS storage classes, see [AWS](https://kubernetes.io/docs/concepts/storage/storage-cla
sses/#aws) in the Kubernetes documentation.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
     labels:
       k8s-addon: storage-aws.addons.k8s.io
     name: gp2-multi-zone
parameters:
     type: gp2
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```
Above we used ```reclaimPolicy``` to _Delete_ which tells K8 to delete the volumes of deleted Pods but you can change it to _Retain_ depend
ing on your needs or if for troubleshooting purpose you would like to keep the volumes of deleted pods.
```

2) We will now use kubectl command to physically create storage class from the manifest files we defined above.

```
$ kubectl create -f sc-gp2.yaml

Output:

storageclass.storage.k8s.io/gp2-multi-zone created
```

3) Verify New Storage Class Once you've created all the storage classes, you can verify them through kubectl command:

```
$ kubectl get sc  --namespace emart

output:

NAME            PROVISIONER             AGE
gp2 (default)   kubernetes.io/aws-ebs   16h
gp2-multi-zone  kubernetes.io/aws-ebs   96s
```

3.3. Server Groups Awareness

Server Group Awareness provides enhanced availability as it protects a cluster from large-scale infrastructure failure, through the definition of groups.

Groups should be defined in accordance with the physical distribution of cluster-nodes. For example, a group should only include the nodes that are in a single server rack, or in the case of cloud deployments, a single availability zone. Thus, if the server rack or availability zone becomes unavailable due to a power or network failure, Group Failover, if enabled, allows continued access to the affected data.

We therefore going to place Couchbase servers onto separate `spec.servers.serverGroups`, which are going to be mapped to physically separated EKS node running in three different AZs (us-east-1a/b/c):

```
spec:
  servers:
    - name: data-east-1a
      size: 1
      services:
        - data
      serverGroups:
       - us-east-1a
```

3.4. Add Storage Class to Persistent Volume Claim Template

With Server groups defined, and Storage Classes available in all three AZs, we are now going to create dynamic storage volumes and mount them of each of the Couchbase server that requires persistent data. In order to do that we will first define Persistent Volume Claim Template in our couchbase-cluster.yaml file (which can be found from the operator folder).

```
Spec:
  volumeClaimTemplates:
    - metadata:
        name: pvc-default
      spec:
        storageClassName: gp2-multi-zone
        resources:
          requests:
            storage: 1Gi
    - metadata:
        name: pvc-data
      spec:
        storageClassName: gp2-multi-zone
        resources:
          requests:
            storage: 5Gi
    - metadata:
        name: pvc-index
      spec:
        storageClassName: gp2-multi-zone
        resources:
          requests:
            storage: 3Gi
```

After claim template being added the final step is to pair the volume claim template with server groups accordingly in each of the zones. For instance, Pods within Server-Group named data-east-1a should use volumeClaimTemplate named *pvc-data* to store data and *pvc-default* for Couchbase binaries and log files.

For example, the following shows the pairing of a Server Group and its associated VolumeClaimTemplate:

```
spec:
  servers:
    - name: data-east-1a
      size: 1
      services:
        - data
      serverGroups:
       - us-east-1a
      pod:
        volumeMounts:
          default: pvc-default
          data: pvc-data
    - name: data-east-1b
      size: 1
      services:
        - data
      serverGroups:
       - us-east-1b
      pod:
        volumeMounts:
          default: pvc-default
          data: pvc-data
    - name: data-east-1c
      size: 1
      services:
        - data
      serverGroups:
       - us-east-1c
      pod:
        volumeMounts:
          default: pvc-default
          data: pvc-data
```

Notice that we have created three separate data server groups (data-east-1a/-1b/-1c), each located in its own AZ, using persistent volume claim templates from that AZ. Now using the same concept we will add index, and query services and allocate them in separate server groups so they can scale independently of data nodes.

3.5. Deploy Couchbase Cluster

The full spec for deploying Couchbase cluster across 3 different zones using persistent volumes can be seen in the couchbase-cluster-with-pv-1.2.yaml file. This file along with other sample yaml files used in this article can be downloaded from this git repo.

Please open the yaml file and note that we are deploying data service in three AZs but deploying index & query service in two AZs only. You can change the configuration to meet your production requirements.

Now use kubectl to deploy the cluster.

```
$ kubectl create -f couchbase-cluster-with-pv-1.2.yaml --save-config --namespace emart
```

This will start deploying the Couchbase cluster and if all goes fine then we will have five Couchbase cluster pods hosting the services as per the configuration file above. To check the progress run this command, which will watch (-w argument) the progress of pods creating:

```
$ kubectl get pods --namespace emart -w

output:

NAME                                          READY     STATUS    RESTARTS   AGE
ckdemo-0000                                   1/1       Running   0          6m3s
ckdemo-0001                                   1/1       Running   0          5m17s
ckdemo-0002                                   1/1       Running   0          4m7s
ckdemo-0003                                   1/1       Running   0          2m57s
ckdemo-0004                                   1/1       Running   0          2m2s
couchbase-operator-admission-6bf9bf8848-zdc7x   1/1       Running   0          9m1s
couchbase-operator-f6f7b6f75-pzb9m            1/1       Running   0          8m17s
```

If for any reason there is an exception, then you can find the details of exception from the couchbase-operator log file. To display the last 20 lines of the log, copy the name of your operator pod and run below command by replacing the operator name with the name in your environment.

```
$ kubectl logs couchbase-operator-f6f7b6f75-pzb9m --namespace emart --tail 25

time="2019-08-27T18:08:22Z" level=info msg="server config query-east-1c: " cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="Cluster status: balanced" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="Node status:" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="┌──────────────┬──────────────────┬──────────────┬─────────────────┐" cluster-name=ckdemo module=
cluster
time="2019-08-27T18:08:22Z" level=info msg="| Server      | Version          | Class        | Status         |" cluster-name=ckdemo module=
cluster
time="2019-08-27T18:08:22Z" level=info msg="├──────────────┼──────────────────┼──────────────┼─────────────────┤" cluster-name=ckdemo module=
cluster
time="2019-08-27T18:08:22Z" level=info msg="| ckdemo-0000 | enterprise-5.5.4 | data-east-1a | managed+active |" cluster-name=ckdemo module=
cluster
time="2019-08-27T18:08:22Z" level=info msg="└──────────────┴──────────────────┴──────────────┴─────────────────┘" cluster-name=ckdemo module=
cluster
time="2019-08-27T18:08:22Z" level=info msg="Scheduler status:" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="┌──────────────┬─────────────┬──────────────┐" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="| Class        | Zone       | Server      |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="├──────────────┼─────────────┼──────────────┤" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="| data-east-1a | us-east-1a | ckdemo-0000 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info msg="└──────────────┴─────────────┴──────────────┘" cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:22Z" level=info cluster-name=ckdemo module=cluster
time="2019-08-27T18:08:27Z" level=info msg="Creating a pod (ckdemo-0001) running Couchbase enterprise-5.5.4" cluster-name=ckdemo module=clu
ster
time="2019-08-27T18:09:37Z" level=info msg="added member (ckdemo-0001)" cluster-name=ckdemo module=cluster
time="2019-08-27T18:09:37Z" level=info msg="Creating a pod (ckdemo-0002) running Couchbase enterprise-5.5.4" cluster-name=ckdemo module=clu
ster
time="2019-08-27T18:10:47Z" level=info msg="added member (ckdemo-0002)" cluster-name=ckdemo module=cluster
time="2019-08-27T18:10:47Z" level=info msg="Creating a pod (ckdemo-0003) running Couchbase enterprise-5.5.4" cluster-name=ckdemo module=clu
ster
time="2019-08-27T18:11:42Z" level=info msg="added member (ckdemo-0003)" cluster-name=ckdemo module=cluster
time="2019-08-27T18:11:42Z" level=info msg="Creating a pod (ckdemo-0004) running Couchbase enterprise-5.5.4" cluster-name=ckdemo module=clu
ster
time="2019-08-27T18:12:33Z" level=info msg="added member (ckdemo-0004)" cluster-name=ckdemo module=cluster
time="2019-08-27T18:12:38Z" level=info msg="Rebalance progress: 0.000000" cluster-name=ckdemo module=cluster
time="2019-08-27T18:12:42Z" level=info msg="reconcile finished" cluster-name=ckdemo module=cluster
time="2019-08-27T18:12:52Z" level=info msg="Created bucketName default" cluster-name=ckdemo module=cluster
```

When all the pods are ready you can view what External-IP are assigned to each pod. Please note that in the couchbase-cluster-with-pv-1.2.yaml file, we used `exposedFeatureServiceType: LoadBalancer` therefore our pods get assigned with public facing IP.

```
$  kubectl get svc --namespace emart -w

NAME                            TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PO
RT(S)                           AGE
ckdemo-0000-exposed-ports       LoadBalancer   10.100.218.99    <pending>     18091:31325/TCP,11207:31716/TCP   2m
ckdemo-0001-exposed-ports       LoadBalancer   10.100.67.128    <pending>     18091:31463/TCP,11207:30847/TCP   2m
ckdemo-0002-exposed-ports       LoadBalancer   10.100.107.146   <pending>     18091:32242/TCP,11207:30585/TCP   2m
ckdemo-0003-exposed-ports       LoadBalancer   10.100.202.204   <pending>     18091:32766/TCP,18092:32672/TCP   2m
ckdemo-0004-exposed-ports       LoadBalancer   10.100.162.33    <pending>     18091:30420/TCP,18092:30063/TCP   2m
ckdemo-srv                      ClusterIP      None             <none>        11210/TCP,11207/TCP               6m
ckdemo-ui                       NodePort       10.100.111.34    <none>        8091:31106/TCP,18091:30156/TCP    6m
couchbase-operator-admission    ClusterIP      10.100.70.83     <none>        443/TCP                           23h
ckdemo-0000-exposed-ports       LoadBalancer   10.100.218.99    a3e1e07e1c9bb11e988541276313ac26-929659888.us-east-1.elb.amazonaws.com    18
091:31325/TCP,11207:31716/TCP   5m6s
ckdemo-0001-exposed-ports       LoadBalancer   10.100.67.128    a3e302ed5c9bb11e988541276313ac26-1624506791.us-east-1.elb.amazonaws.com   18
091:31463/TCP,11207:30847/TCP   5m5s
ckdemo-0002-exposed-ports       LoadBalancer   10.100.107.146   a3e0f0732c9bb11e988541276313ac26-1179001987.us-east-1.elb.amazonaws.com   18
091:32242/TCP,11207:30585/TCP   5m6s
ckdemo-0003-exposed-ports       LoadBalancer   10.100.202.204   a3e14de9cc9bb11e988541276313ac26-1849814018.us-east-1.elb.amazonaws.com   18
091:32766/TCP,18092:32672/TCP   5m6s
ckdemo-0004-exposed-ports       LoadBalancer   10.100.162.33    a3e198391c9bb11e988541276313ac26-11092324.us-east-1.elb.amazonaws.com     18
091:30420/TCP,18092:30063/TCP   5m6s
```

At this point you can open up a browser and type https://a3e1e07e1c9bb11e988541276313ac26-929659888.us-east-1.elb.amazonaws.com:18091 (external-ip of ckdemo-0000 pod) which will bring Couchbase web-console from where you can monitor server stats, create buckets, run queries all from one single place.

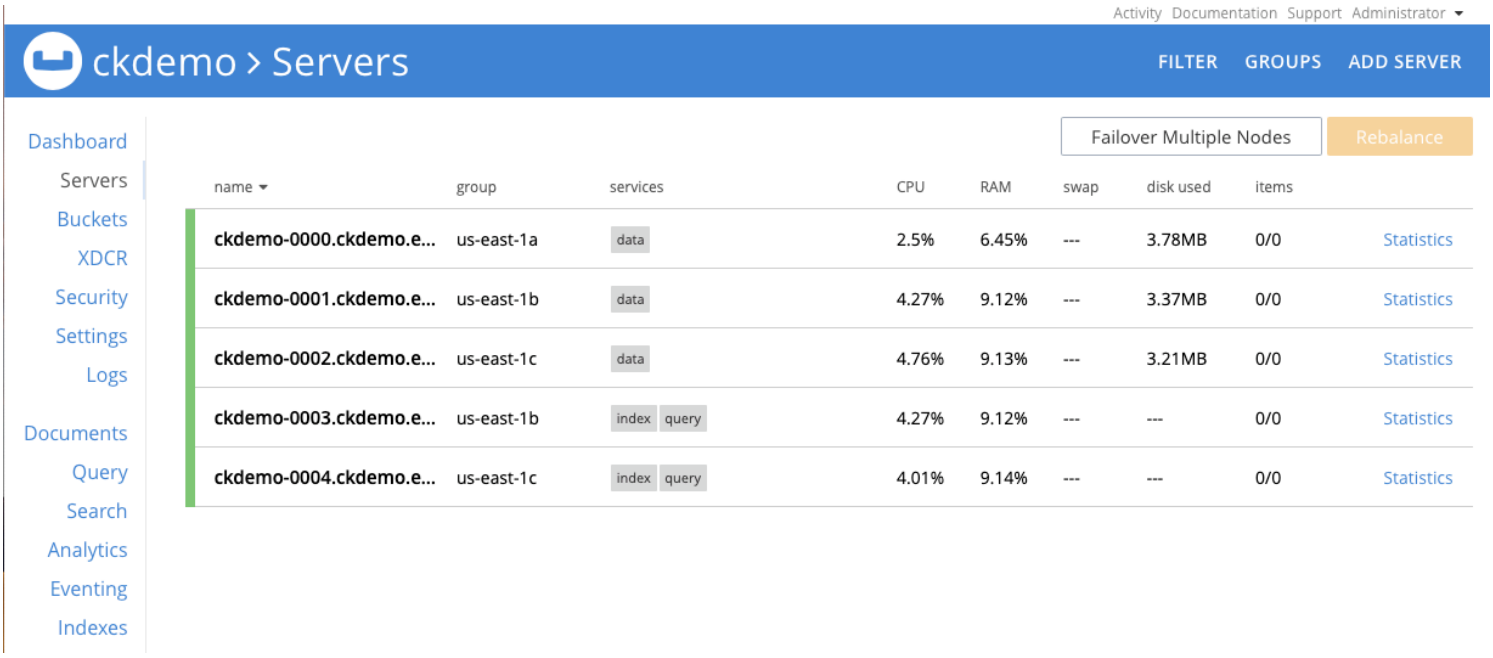


Figure 1: Five node Couchbase cluster using persistent volumes.

If for any reason you don't get external-ip assigned to your pods you can always do the port forwarding like this:

```
$ kubectl port-forward ckdemo-0000 18091:18091 --namespace emart
```

And access web-console by typing https://localhost:18091 on the browser.

4. Test Client-Application

After deploying a secured Couchbase Cluster, we would like to perform some load test so that we can confirm that a client application running locally on our laptop can write JSON documents into the Couchbase bucket.

To connect to an SSL enabled Couchbase Cluster we need to setup Keystore locally on our laptop machine so we can securely persist the certificates we used to setup the cluster in the first place. The steps are simple and covered in How to setup Keystore document.

Once you have setup the Keystore then we are going to download a Java jar file from here:

```
$ wget https://raw.githubusercontent.com/sahnianuj/cb-loadgen/master/bin/cbloadgen.jar

--2019-08-31 21:40:22--  https://raw.githubusercontent.com/sahnianuj/cb-loadgen/master/bin/cbloadgen.jar
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.188.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.188.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15646628 (15M) [application/octet-stream]
Saving to: ‘cbloadgen.jar’

cbloadgen.jar     100%[=====================================================================>]  14.92M  32.7MB/s    in 0.5s
```

Also, make sure that you have java installed on your machine before we run the workload test. Please run this command to make sure you have Java 1.8 or higher installed.

```
$ java -version

java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
```

We are ready to write a few thousand documents into our *default* bucket by using the jar file we downloaded before. Here is the command to use:

```
$ java -jar cbloadgen.jar -t 10 -d 1000 -h localhost -u Administrator \
-p password -b default -e true -ks ~/.keystore -kp password

Connection created.
..............
************************************************************
Time elapsed: 1785 ms
Average Throughput: 560 ops/sec
Average Latency: 1.78 ms
************************************************************
```

In order to learn more about the arguments used, please follow the README file.

Note: There could be a significant network latency between your laptop and Amazon EKS cluster so please take performance numbers with a grain of salt.

5. Operations

In this section we are going to perform some operational tasks like cluster expansion, cluster upgrade and test self-recovery feature of the Couchbase Autonomous operator. Let's start with the last topic first.

5.1. Self-Recovery from Failure

One of the best feature of Kubernetes in general is that it provides Auto-Healing capability to the services that are managed by it. With Couchbase Autonomous Operator we leverage the same Auto-Healing capability of K8 for Couchbase Cluster.

We are going to test this capability by manually deleting the pod, which would be detected by K8 as an exception, which will inform the operator to bring back the current state of the cluster to the desired state as specified in the couchbase-cluster-with-pv-1.2.yaml file.

Let's induce the fault now using kubectl delete command:

```
$ kubectl get pod --namespace emart
NAME                                          READY     STATUS    RESTARTS   AGE
ckdemo-0000                                   1/1       Running   0          15m
ckdemo-0001                                   1/1       Running   0          15m
ckdemo-0002                                   1/1       Running   0          13m
ckdemo-0003                                   1/1       Running   0          12m
ckdemo-0004                                   1/1       Running   0          11m
couchbase-operator-admission-6bf9bf8848-zdc7x 1/1       Running   0          18m
couchbase-operator-f6f7b6f75-pzb9m            1/1       Running   0          18m
```

```
$ kubectl delete pod ckdemo-0001 --namespace emart
pod "ckdemo-0001" deleted
```

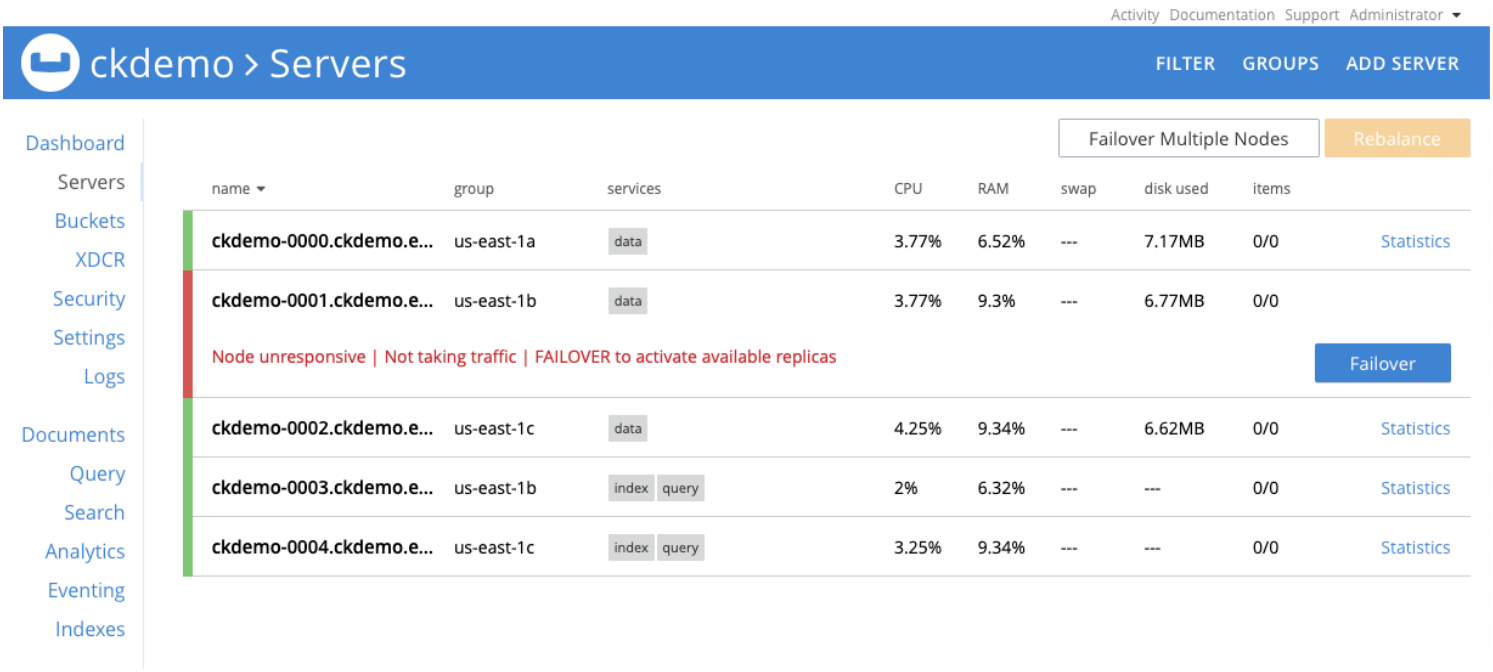


Figure 2: One of the Data pod is dropped.

After Couchbase Autonomous Operator detects the failure it triggers the healing process.

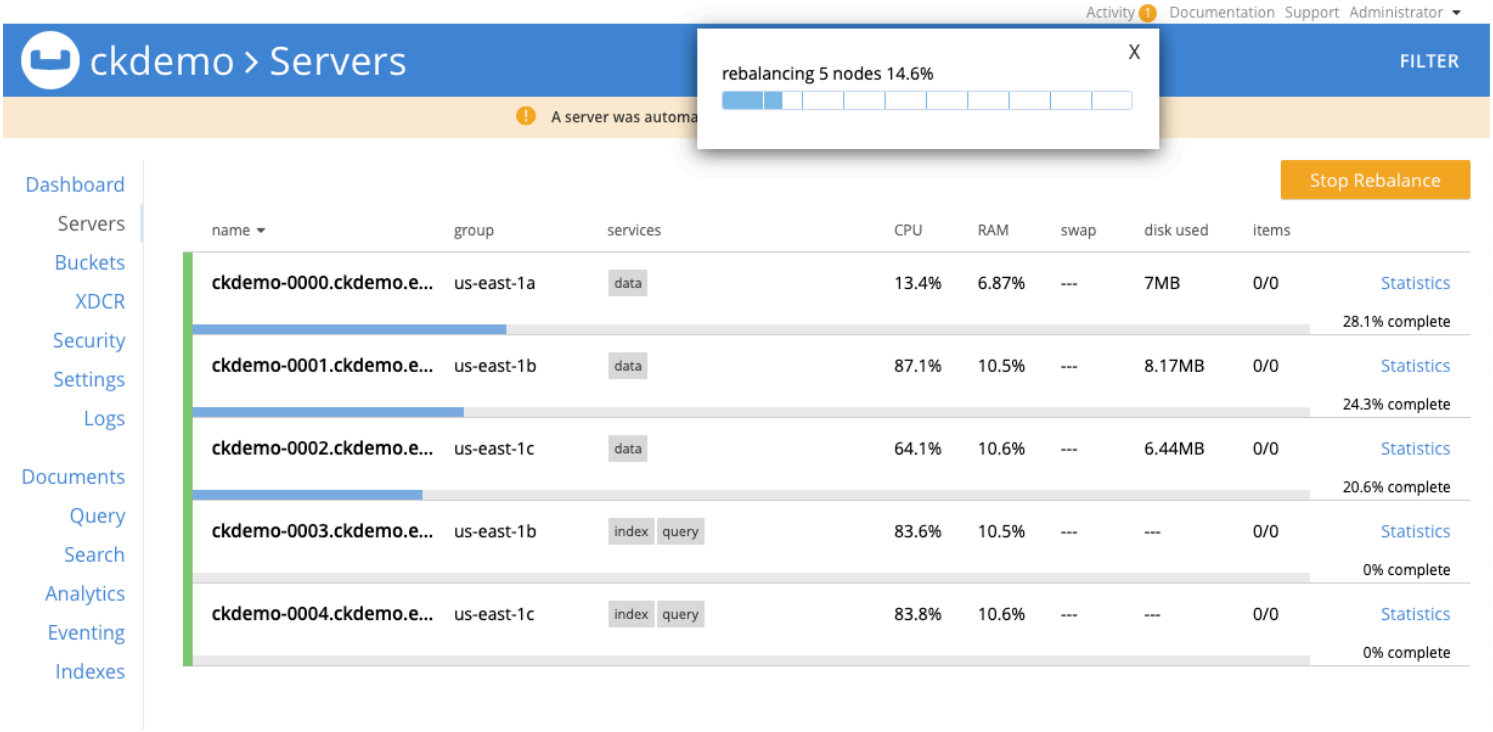


Figure 3: Data node with same name and persistent volume will be restored automatically.

5.2. On-Demand Scaling - Up & Down

If you have ever scaled-out or scaled-in a database cluster you would know that it is a non-trivial process as it entails lot of manually triggered steps which are not only time consuming but also error prone.

With Couchbase Autonomous Operator scaling-out or scaling-in is as simple as changing the desired number servers for a specific service in the couchbase-cluster-with-pv-1.2.yaml file. Let's open this YAML file again and add one server node in us-east-1a server-group running Index and Query service.

Notice we don't have any Index and Query service in the **us-east-1a** serverGroups:

```
- name: query-east-1b
  size: 1
  services:
    - query
    - index
  serverGroups:
   - us-east-1b
  pod:
    volumeMounts:
      default: pvc-default
      index: pvc-index
- name: query-east-1c
  size: 1
  services:
    - index
    - query
  serverGroups:
   - us-east-1c
  pod:
    volumeMounts:
      default: pvc-default
      index: pvc-index
```

So we are going to add one more server in **us-east-1a** server group hosting both index and query service like this:

```
  - name: query-east-1a
    size: 1
    services:
      - query
      - index
    serverGroups:
     - us-east-1a
    pod:
      volumeMounts:
        default: pvc-default
        index: pvc-index
  - name: query-east-1b
    ....
  - name: query-east-1c
    ....
```

A separate couchbase-cluster-sout-with-pv-1.2.yaml file is provided just for convenience but if you want you can also make changes yourself in the couchbase-cluster-with-pv-1.2.yaml

```
$ kubectl apply -f couchbase-cluster-sout-with-pv-1.2.yaml  --namespace emart
```

Notice a new pod will be getting ready to be added to the cluster:

```
$ kubectl get pods --namespace emart NAME READY STATUS RESTARTS AGE ckdemo-0000 1/1 Running 0 27m ckdemo-0001 1/1 Running 0 10m ckdemo-0
```

After pod is ready you can view it from the Web Console as well.

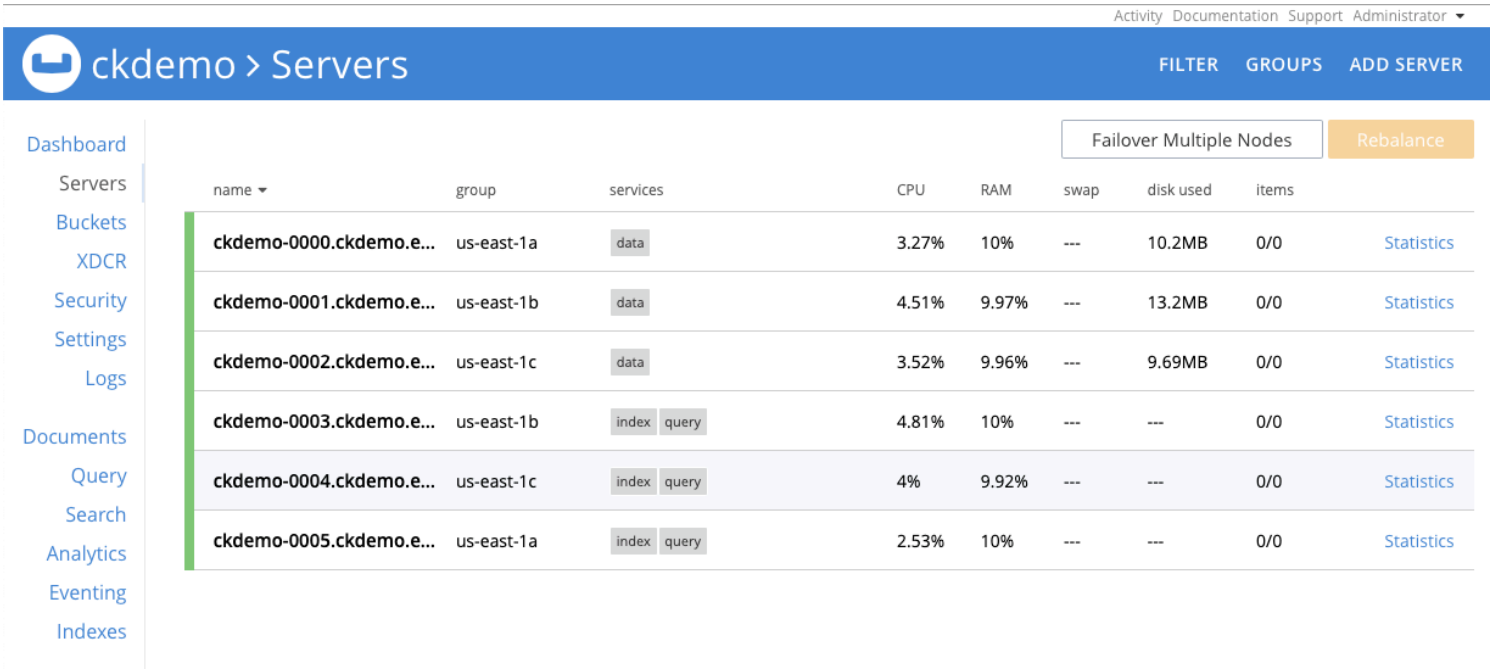


Figure 4: Cluster now has three Couchbase server pods hosting Index and Query services spread across three server-groups.

```
$ kubectl logs couchbase-operator-f6f7b6f75-pzb9m --namespace emart --tail 20

time="2019-08-27T18:47:06Z" level=info msg="┌─────────────┬─────────────────┬──────────────┬─────────────────┐" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| Server      | Version         | Class        | Status          |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="├─────────────┼─────────────────┼──────────────┼─────────────────┤" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| ckdemo-0000 | enterprise-5.5.4 | data-east-1a  | managed+active |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| ckdemo-0001 | enterprise-5.5.4 | data-east-1b  | managed+active |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| ckdemo-0002 | enterprise-5.5.4 | data-east-1c  | managed+active |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| ckdemo-0003 | enterprise-5.5.4 | query-east-1b | managed+active |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| ckdemo-0004 | enterprise-5.5.4 | query-east-1c | managed+active |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="| ckdemo-0005 | enterprise-5.5.4 | query-east-1a | managed+active |" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="└─────────────┴─────────────────┴──────────────┴─────────────────┘" cluster-name=ckdemo module
=cluster
time="2019-08-27T18:47:06Z" level=info msg="Scheduler status:" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="┌─────────────────┬─────────────┬──────────────┐" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| Class           | Zone        | Server       |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="├─────────────────┼─────────────┼──────────────┤" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| data-east-1a    | us-east-1a | ckdemo-0000 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| data-east-1b    | us-east-1b | ckdemo-0001 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| data-east-1c    | us-east-1c | ckdemo-0002 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| query-east-1a | us-east-1a | ckdemo-0005 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| query-east-1b | us-east-1b | ckdemo-0003 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="| query-east-1c | us-east-1c | ckdemo-0004 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info msg="└─────────────────┴─────────────┴──────────────┘" cluster-name=ckdemo module=cluster
time="2019-08-27T18:47:06Z" level=info cluster-name=ckdemo module=cluster
```

5.3. Couchbase Automated Upgrade

Any software in service goes through continuous improvement and there is definitely going to be the moments when you would like to upgrade Couchbase Autonomous Operator too because of some new feature or the patch which is critical for your business.

Upgrading a distributed cluster like Couchbase requires careful orchestration of steps if you manually run the online upgrade operation. With Couchbase Autonomous Operator the whole symphony of these operations are completely automated, so the management becomes very easy.

Let's take a look at how you can upgrade the system in an online fashion.

5.3.1. Preparing for Upgrade

Before beginning an upgrade to your Kubernetes cluster, review the following considerations and prerequisites:

- As an eviction deletes a pod, ensure that the Couchbase cluster is scaled correctly so that it can handle the increased load of having a pod down while a new pod is balanced into the cluster.
- To minimize disruption, ensure that a short failover period is configured with the autoFailoverTimeout parameter to reduce down time before another node takes over the load.
- Ensure that there is capacity in your Kubernetes cluster to handle the scheduling of replacement Couchbase pods. For example, if a Couchbase cluster were running on Kubernetes nodes marked exclusively for use by Couchbase, and anti-affinity were enabled as per the deployment best practices, the Kubernetes cluster would require at least one other node capable of scheduling and running your Couchbase workload.

5.3.2. Perform Automatic Upgrade

To prevent downtime or a data loss scenario, the Operator provides controls for how automated Kubernetes upgrades proceed.

A PodDisruptionBudget is created for each CouchbaseCluster resource created. The PodDisruptionBudget specifies that at least the cluster size minus one node (N-1) be ready at any time. This constraint allows, at most, one node to be evicted at a time. As a result, it's recommended that to support an automatic Kubernetes upgrade, the cluster be deployed with anti-affinity enabled to guarantee only a single eviction at a time.

Now let's open couchbase-cluster-with-pv-1.2.yaml file and change:

```
spec:
  baseImage: couchbase/server
  version: enterprise-5.5.4
```

to `spec: baseImage: couchbase/server version: enterprise-6.0.2` Now using kubectl to deploy the cluster.

```
$ kubectl apply -f couchbase-cluster-with-pv-1.2.yaml  --namespace emart
```

At this point you would notice the pods will be evicted one by one and new pod will be joined to the existing cluster but with an upgraded Couchbase version (6.0.2).

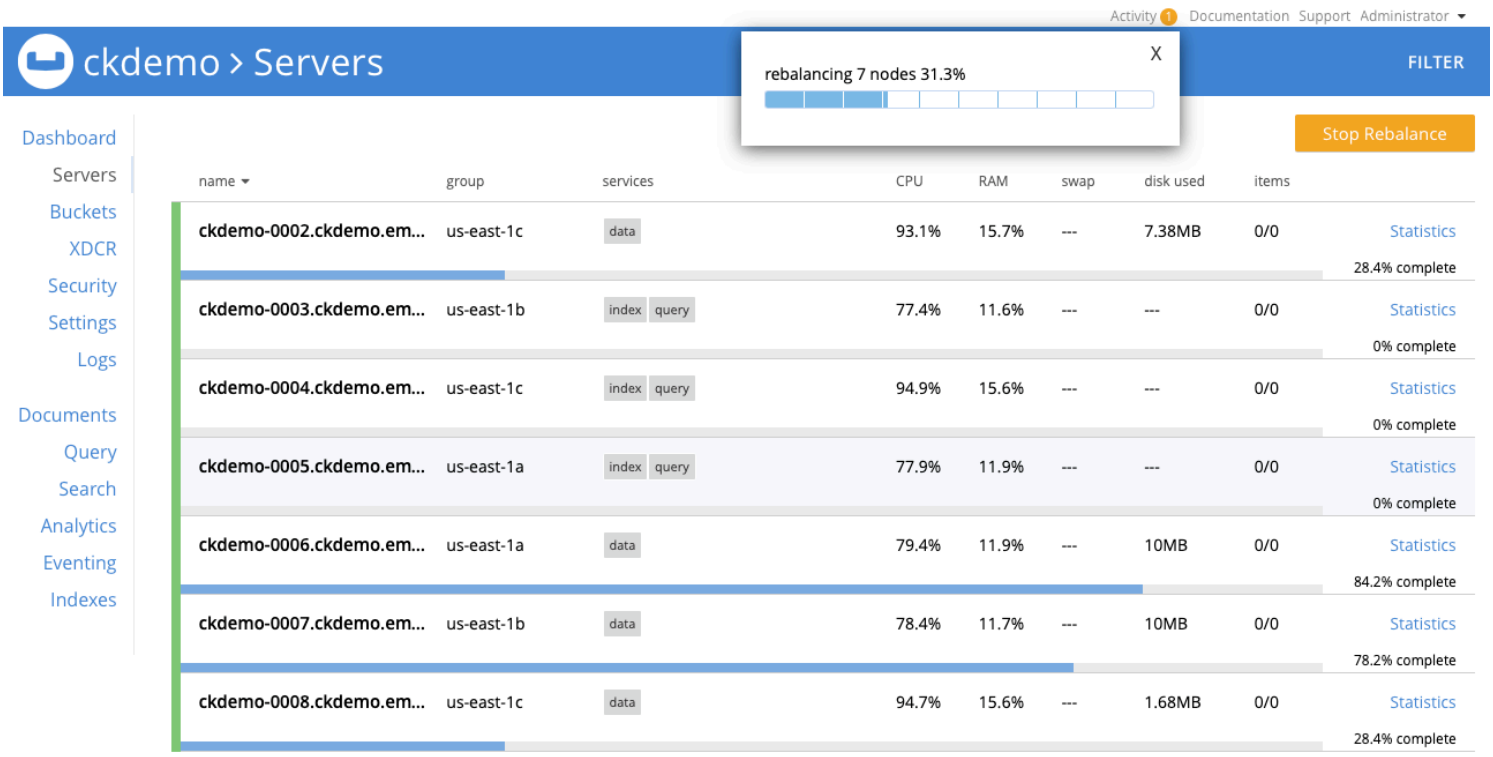


Figure 6: Couchbase Cluster getting upgraded one pod at a time in and online fashion.

Just wait for some time and cluster will upgraded one pod at a time in a rolling fashion.

```
$ kubectl logs couchbase-operator-f6f7b6f75-pzb9m --namespacemart --tail 20

time="2019-08-27T18:55:40Z" level=info msg="| Class          | Zone       | Server      |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="├────────────────┼────────────┼─────────────┤" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="| data-east-1a   | us-east-1a | ckdemo-0006 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="| data-east-1b   | us-east-1b | ckdemo-0007 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="| data-east-1c   | us-east-1c | ckdemo-0008 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="| query-east-1a  | us-east-1a | ckdemo-0005 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="| query-east-1b  | us-east-1b | ckdemo-0009 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="| query-east-1c  | us-east-1c | ckdemo-0004 |" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info msg="└────────────────┴────────────┴─────────────┘" cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:40Z" level=info cluster-name=ckdemo module=cluster
time="2019-08-27T18:55:43Z" level=info msg="Planning upgrade of candidate ckdemo-0004 from enterprise-5.5.4 to enterprise-6.0.2" cluster-na
me=ckdemo module=cluster
time="2019-08-27T18:55:44Z" level=info msg="Creating a pod (ckdemo-0010) running Couchbase enterprise-6.0.2" cluster-name=ckdemo module=clu
ster
```

Note: At some point during upgrade all the pods are going to be recycled and external-ip addresses are also going to be reset. So you would need to check the services again to find the latest external-ip by running below command so you can reconnect to the web-console:

```
$  kubectl get svc -n emart
NAME                           TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PO
RT(S)                           AGE

ckdemo-0005-exposed-ports      LoadBalancer   10.100.184.222   ad15bb757c9cc11e988541276313ac26-1745975236.us-east-1.elb.amazonaws.com   18
091:30716/TCP,11207:31845/TCP   89m
ckdemo-0006-exposed-ports      LoadBalancer   10.100.24.45     a1546a98ec9cd11e988541276313ac26-1471357696.us-east-1.elb.amazonaws.com   11
207:30645/TCP,18091:31792/TCP   87m
ckdemo-0007-exposed-ports      LoadBalancer   10.100.225.136   a58c87ecbc9cd11e988541276313ac26-105279754.us-east-1.elb.amazonaws.com    11
207:32754/TCP,18091:30044/TCP   85m
ckdemo-0008-exposed-ports      LoadBalancer   10.100.105.23    a9bb9db5bc9cd11e988541276313ac26-425926447.us-east-1.elb.amazonaws.com    18
091:31156/TCP,18092:31766/TCP   83m
ckdemo-0009-exposed-ports      LoadBalancer   10.100.156.109   ad2fcf7c1c9cd11e988541276313ac26-549415956.us-east-1.elb.amazonaws.com    18
091:31214/TCP,18092:31299/TCP   82m

  $ kubectl port-forward ckdemo-0006 18091:18091 --namespace emart
```

Refreshing external-ip is not a desired behavior from Couchbase client perspective and in later section we will describe how you can add `externalDNS` in front of your EKS cluster so that there is no impact of IP being changed as client code will use Fully Qualified Domain Names (FQDN) instead.

6. Conclusion

Couchbase Autonomous Operator makes management and orchestration of Couchbase Cluster seamless on the Kubernetes platform. What makes this operator unique is its ability to easily use storage classes offered by different cloud vendors (AWS, Azure, GCP, RedHat OpenShift, etc) to create persistent volumes, which is then used by the Couchbase database cluster to persistently store the data. In the event of pod or container failure, Kubernetes re-instantiate a new pod/container automatically and simply remounts the persistent volumes back, making the recovery fast. It also helps maintain the SLA of the system during infrastructure failure recovery because only delta recovery is needed as opposed to full-recovery, if persistent volumes are not being used.

We walked through step-by-step on how you will setup persistent volumes on Amazon EKS in this article but the same steps would also be applicable if you are using any other open-source Kubernetes environment (AKS, GKE, etc). We hope you will give Couchbase Autonomous Operator a spin and let us know of your experience.

7. Appendix

7.1. Prerequisite Tools

In this section, you will perform the necessary steps to install and configure the prerequisite tools required to interact with kubernetes cluster deployed on AWS. Follow the steps below and reach out for help if you get stuck. You can skip any step if you already have the corresponding tool(s) installed on your computer.

Remember: it is important to have the right minimum versions of the tools. This guide focuses on installing the right versions, so you get the functionality needed to complete all steps successfully.

For Mac OS, it is recommended you install Homebrew package manager; it will make things much easier.

There are separate sections for Mac and Windows setup steps where needed. If there are no such separate sections, the steps are similar or identical for both operating systems, however terminal commands shown are based on Mac OS and some may need to be modified before being run on Windows, e.g. change / to \, remove trailing & character to run commands in the background.

This guide does not cover installation steps on Linux due to a variety of popular Linux distributions. Please ask for help if you are using Linux and get stuck.

Step 1: Install Python 2.7.15 and pip Package Manager

We will use the Python programming language for automating some tasks in the labs. The installers are available from the following page: Python 2.7.15 Downloads. Python 3.x will also work if you already have that installed.

Mac - Option 1 - Homebrew

Use Homebrew to install Python 2.7.15 and pip by issuing the following command in a terminal: `$ brew install python2`

Mac - Option 2 - Installer

You can download and run the following installer: Python 2.7.15 MacOS installer. If you run the installer and accept all the defaults, it will install pip (Python package manager) too.

Check Python and pip versions in a terminal to make sure they successfully installed:
`$ python -V Python 2.7.15 $ pip -V pip 18.1 from /usr/local/lib/python2.7/site-packages/pip (python 2.7)`

Windows - Installer

Download and run the following installer: Python 2.7.15 Windows 64-bit installer. If you run the installer and accept all the defaults, it will install pip (Python package manager) too. You can also choose to add python.exe to your PATH environment variable as part of the installation (recommended):

Check Python and pip versions in a terminal to make sure they successfully installed:

```
> python -V
Python 2.7.15
> pip -V
pip 9.0.3 from c:\python27\lib\site-packages (python 2.7)
```

Step 2: Install kubectl

We will use kubectl (the Kubernetes command-line tool) to deploy and manage applications on Kubernetes.
There are multiple options to download and install kubectl for your operating system, you can find them on the following page: Install and setup kubectl. Below are the recommended methods for Mac and Windows (please do only one of the 3 options provided for Windows).

Mac - Homebrew

Use Homebrew package manager to install kubectl by issuing the following command in a terminal: `$ brew install kubernetes-cli` Check kubectl version in a terminal to make sure it successfully installed: ``` $ kubectl version

Client Version: version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.3", GitCommit:"721bfa751924da8d1680787490c54b9179b1fed0", GitTreeState:"clean", BuildDate:"2019-02-04T04:48:03Z", GoVersion:"go1.11.5", Compiler:"gc", Platform:"darwin/amd64"} The connection to the server localhost:8080 was refused - did you specify the right host or port? ```

Windows - Option 1 - Download kubectl.exe

1. Download the kubectl.exe binary from here: kubectl 1.13.0 binary for Windows. Put the kubectl.exe file in any directory of your choice on your computer, e.g. `C:\kubectl`.

2. Modify/edit your PATH environment variable to include the path where you put kubectl.exe, e.g., C:\kubectl. Use the Environment Variables dialog box (Control Panel → System → Advanced system settings → Environment Variables) to change the PATH variable permanently or use the terminal as shown below to change the PATH variable for the duration of the session: ```

> set PATH=%PATH%;C:\kubectl ```
>
> ### Windows - Option 2 - PowerShell Gallery
>
> Use PowerShell Gallery package manager. This works best on Windows 10, since Install-PackageProvider cmdlet has not been part of the OS prior to Windows 10.

Run Windows PowerShell as Administrator and execute the following commands: ```

> Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser -Force Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force Name Version Source Summary
>
> ---
>
> nuget 2.8.5.208 https://onege... NuGet provider for the OneGet meta-package manager Install-Script -Name install-kubectl -Scope CurrentUser -Force install-kubectl.ps1 -DownloadLocation C:\YOUR_PATH ==>Getting download link from https://kubernetes.io/docs/tasks/tools/install-kubectl/ ==>analyzing Downloadlink ==>starting Download from https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/windows/amd64/kubectl.exe using Bitstransfer ==>starting 'C:\kubectl\kubectl.exe version' Client Version: version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.0", GitCommit:"ddf47ac13c1a9483ea035a79cd7c10005ff21a6d", GitTreeState:"clean", BuildDate:"2018-12-03T21:04:45Z", GoVersion:"go1.11.2", Compiler:"gc", Platform:"windows/amd64"} Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it.

You can now start kubectl from C:\kubectl\kubectl.exe ``` **Note**: If you do not specify -DownloadLocation parameter, kubectl.exe will be installed in your temp directory.

Windows - Option 3 - Chocolatey

Using Chocolatey package manager. This works well on Windows 7 and later versions.

Once you install Chocolatey, run Windows PowerShell as Administrator and execute the following commands: ```

> choco install kubernetes-cli Chocolatey v0.10.11 Installing the following packages: kubernetes-cli By installing you accept licenses for the packages. Progress: Downloading kubernetes-cli 1.13.3... 100%

kubernetes-cli v1.13.3 [Approved] kubernetes-cli package files install completed. Performing other installation steps. The package kubernetes-cli wants to run 'chocolateyInstall.ps1'. Note: If you don't run this script, the installation will fail. Note: To confirm automatically next time, use '-y' or consider: choco feature enable -n allowGlobalConfirmation Do you want to run the script?([Y]es/[N]o/[P]rint): Y

..... ShimGen has successfully created a shim for kubectl.exe The install of kubernetes-cli was successful. Software installed to 'C:\ProgramData\chocolatey\lib\kubernetes-cli\tools'

Chocolatey installed 1/1 packages. See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

> cd $HOME mkdir .kube Mode LastWriteTime Length Name
>
> ---
>
> d---- 02/04/2019 4:42 PM .kube cd .kube New-Item config -type file Mode LastWriteTime Length Name
>
> ---
>
> -a--- 02/04/2019 4:43 PM 0 config `Check kubectl version in a terminal to make sure it successfully installed:` kubectl version

Client Version: version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.0", GitCommit:"ddf47ac13c1a9483ea035a79cd7c10005ff21a6d", GitTreeState:"clean", BuildDate:"2018-12-03T21:04:45Z", GoVersion:"go1.11.2", Compiler:"gc", Platform:"windows/amd64"} Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it. ```

Step 3: Install aws-iam-authenticator tool

The aws-iam-authenticator tool uses AWS Identity and Access Management (IAM) credentials to authenticate to a Kubernetes cluster.

Mac - Homebrew

Use Homebrew package manager to install aws-iam-authenticator by issuing the following command in a terminal: `$ brew install aws-iam-authenticator` Check aws-iam-authenticator help page in a terminal to make sure it successfully installed: ``` $ aws-iam-authenticator help

A tool to authenticate to Kubernetes using AWS IAM credentials

Usage: aws-iam-authenticator [command]

Available Commands: help Help about any command init Pre-generate certificate, private key, and kubeconfig files for the server. server Run a webhook validation server suitable that validates tokens using AWS IAM token Authenticate using AWS IAM and get token for Kubernetes verify Verify a token for debugging purpose version Version will output the current build information

Flags: -i, --cluster-id ID Specify the cluster ID, a unique-per-cluster identifier for your aws-iam-authenticator installation. -c, --config filename Load configuration from filename -h, --help help for aws-iam-authenticator -l, --log-format string Specify log format to use when logging to stderr text or json

Use "aws-iam-authenticator [command] --help" for more information about a command. ```

Windows - Download aws-iam-authenticator.exe

1. Download the aws-iam-authenticator.exe binary from here: aws-iam-authenticator 1.11.5 binary for Windows. Put the aws-iam-authenticator.exe file in any directory on your computer, e.g., C:\aws .
2. Modify/edit your PATH environment variable to include the path where you put aws-iam-authenticator.exe, e.g., C:\aws. Use the Environment Variables dialog box (Control Panel → System → Advanced system settings → Environment Variables) to change the PATH variable permanently or use the terminal as shown below to change the PATH variable for the duration of the session: ```

> set PATH=%PATH%;C:\aws `Check aws-iam-authentication help page in a terminal to make sure it successfully installed:` $ aws-iam-authenticator help

A tool to authenticate to Kubernetes using AWS IAM credentials

Usage: aws-iam-authenticator [command]

Available Commands: help Help about any command init Pre-generate certificate, private key, and kubeconfig files for the server. server Run a webhook validation server suitable that validates tokens using AWS IAM token Authenticate using AWS IAM and get token for Kubernetes verify Verify a token for debugging purpose version Version will output the current build information

Flags: -i, --cluster-id ID Specify the cluster ID, a unique-per-cluster identifier for your aws-iam-authenticator installation. -c, --config filename Load configuration from filename -h, --help help for aws-iam-authenticator -l, --log-format string Specify log format to use when logging to stderr text or json

Use "aws-iam-authenticator [command] --help" for more information about a command. ```

Step 4: Install AWS CLI (version 1.16.73 or greater)

AWS Command Line Interface (CLI) is a unified tool to manage all AWS services.

Mac - Homebrew

Use Homebrew package manager to install awscli by issuing the following command in a terminal: `$ brew install awscli` If aws is already installed, but the version is older than 1.16.73, you need to upgrade to the latest version: `$ brew upgrade awscli` Check aws version in a terminal to make sure it successfully installed: ``` $ aws --version

aws-cli/1.16.78 Python/2.7.10 Darwin/18.0.0 botocore/1.12.68 ```

Windows Installer

Download and run the following installer: AWS CLI Windows 64-bit installer.

Check aws version in a terminal to make sure it successfully installed: ```

> aws --version

aws-cli/1.16.159 Python/2.7.16 Windows/10 botocore/1.12.149 ```

Step 5: Configure AWS CLI

We will configure AWS CLI credentials and options by running aws configure command. Use the following values for AWS Access Key ID, AWS Secret Access Key, Default region name, and Default output format:

NAME	VALUE
AWS Access Key ID	A**********
AWS Secret Access Key	L**********/8****
Default region name	us-east-2
Default output format	json

Run the aws configure command and enter each of the values provided above, as shown below: ``` $ aws configure

AWS Access Key ID [none]: A********** AWS Secret Access Key [none]: L**********/8**** Default region name [none]: us-east-2 Default output format [none]: json ```

Step 6: Install git CLI

git CLI is a tool to work with GitHub repositories.

Mac - git via Homebrew

git CLI is usually pre-installed on Mac. You can also install it by using Homebrew package manager: `$ brew install git`

Windows - git via Installer

git CLI for Windows can be downloaded from the following location: Git for Windows. Here is the direct link for the current 64-bit installer: Git Windows 64-bit installer.

7.2. EKS Setup

The following section will walk through the steps to create the EKS cluster. This EKS cluster will be used to deploy Couchbase Autonomous Operator later in the lab.

Set up EKS Creation Script

In order to automate the setup we have written python based scripts which can be configured to deploy the EKS Cluster under anybody's AWS account. Please follow through the steps below:

Step 1. Clone EKS Creation Scripts repo

```
  $ git clone https://github.com/couchbaselabs/cbsummit-create-eks-cluster.git
  Cloning into 'create_eks_cluster'...
  remote: Enumerating objects: 30, done.
  remote: Counting objects: 100% (30/30), done.
  remote: Compressing objects: 100% (21/21), done.
  remote: Total 30 (delta 13), reused 20 (delta 7), pack-reused 0
  Unpacking objects: 100% (30/30), done.
```

Step 2. CD into newly clone directory

```
$ cd cbsummit-create-eks-cluster
$ ls -l
total 48
-rw-r--r--  1 mmaster  staff  4633 14 Jan 06:40 README.md
-rw-r--r--  1 mmaster  staff  8893 14 Jan 06:40 create_eks_script.py
-rw-r--r--  1 mmaster  staff  1383 14 Jan 06:55 parameters.py
```

More details on this script can be found in the README.md provided within the repository

Step 3. Change the parameters for your set up

The following shows the list of parameters in parameters.py that you are likely to change. It is required to change VPC*STACK*NAME, EKS*CLUSTER*NAME, EKS*NODES*STACK*NAME, and EKS*NODE*GROUPNAME. The value of EKS*CLUSTER*NAME is the cluster name that will be provided to the participants/users. All other parameters can be left as default or adjusted based on the requirements of the summit, e.g. number of participants expected:

Parameter Name	Description
VPC*STACK*NAME	This is the name of the Stack for the VPC
EKS*CLUSTER*NAME	This is the name of the EKS Cluster to create
EKS*NODES*STACK_NAME	This is the name of the Stack for the EC2 worker nodes
EKS*NODE*GROUP_NAME	This is the name of the Autoscaling group for the EC2 worker nodes
EKS*NODE*AS*GROUP*MIN	The minimum size of the autoscaling group
EKS*NODE*AS*GROUP*MAX	The maximum size of the autoscaling group
EKS*NODE*AS*GROUP*DESIRED	The desired cluster size, should be >= * EKS*NODE*AS*GROUP*MIN and <= EKS*NODE*AS*GROUP*MAX
EKS*NODE*INSTANCE_TYPE.	The type of instance you want to use as a worker node
EKS*IMAGE*ID	This only needs to be changed if you are not deploying into the us-east-2 region.
EKS*NODE*VOLUME_SIZE	If you need additional disk space

```
$ cat parameters.py
ATTEMPTS=10
WAIT_SEC=120

#================================
#   VPC Information
#================================
VPC_STACK_NAME="ckteststackmm"
VPC_TEMPLATE="https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/amazon-eks-vpc-sample.yaml"

#================================
#   EKS Cluster
#================================
EKS_CLUSTER_NAME="cktestclustermm"
EKS_ROLE_ARN="arn:aws:iam::669678783832:role/cbd-eks-role"

#================================
#   EKS Worker Nodes
#================================
EKS_NODES_TEMPLATE="https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/amazon-eks-nodegroup.yaml"
EKS_NODES_STACK_NAME="cktestclustermm-nodes"
EKS_NODE_GROUP_NAME="cktestclustermm-eks-nodes"
EKS_NODE_AS_GROUP_MIN="3"
EKS_NODE_AS_GROUP_MAX="3"
EKS_NODE_AS_GROUP_DESIRED="3"

#Amazon instance type - Refer to Amazon Documentation for available values
EKS_NODE_INSTANCE_TYPE="m4.xlarge"

#Amazon Image Id - Refer to https://docs.aws.amazon.com/eks/latest/userguide/getting-started.html for full list.  The region is important f
or AMI to use.  The below is for us-east-2
EKS_IMAGE_ID="ami-053cbe66e0033ebcf"

#The IAM Key to use
EKS_KEY_NAME="cb-day-se"

EKS_NODE_VOLUME_SIZE="20"

#================================
#   Secondary User
#================================
AWS_SECOND_USER_ARN="arn:aws:iam::669678783832:user/cb-day-participant"
AWS_SECOND_USER_NAME="cb-day-participant"
```

EKS*IMAGE*ID If you are not deploying to the us-east-2 region, first validate that the region supports EKS, and also that you are using the correct AMI for that region. The following table lists the regions with the assigned regions for each SE team, i.e. US & Canada should use us-east-2 and EMEA should use eu-west-1.

The list of AMI's for available regions can be found at the below here.

Step 4: Execute the create*eks*script

This step will create the EKS cluster and the nodes as defined in the parameters.py file. The name of the EKS cluster defined earlier in the parameters file is what will have to be provided to the participants/users. This will appear in the output of the command, e.g. *cktestclustermm* as shown in the example below for the aws eks create-cluster command. Verify that the correct cluster name has been issued in that command and provide that cluster name to the participants.

See the README.md for more details.

```
$ python create_eks_script.py install

---------------------------------------
Starting installation of EKS from step 0
---------------------------------------
---------------------------------------
Running aws cloudformation create-stack --stack-name ckteststackmm --template-url https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformat
ion/2018-12-10/amazon-eks-vpc-sample.yaml
---------------------------------------
Executing command : aws cloudformation create-stack --stack-name ckteststackmm --template-url https://amazon-eks.s3-us-west-2.amazonaws.com
/cloudformation/2018-12-10/amazon-eks-vpc-sample.yaml
{

    "StackId": "arn:aws:cloudformation:us-east-2:669678783832:stack/ckteststackmm/d278e040-265a-11e9-a4be-02649197f7a0"

}

Checking completion status...
Checking attempt #0
```

```
Status :: "CREATE_IN_PROGRESS"
Checking attempt #1
Status :: "CREATE_COMPLETE"
Adding key SecurityGroups with value sg-0677229e743e97317
Adding key VpcId with value vpc-074ac91b494db16f2
Adding key SubnetIds with value subnet-06d16b69a017ce7a6,subnet-07d794f0b706800a3,subnet-0a4aee63ad0296add
---------------------------------------
Running aws eks create-cluster --name cktestclustermm --role-arn arn:aws:iam::669678783832:role/cbd-eks-role --resources-vpc-config subnetI
ds=subnet-06d16b69a017ce7a6,subnet-07d794f0b706800a3,subnet-0a4aee63ad0296add,securityGroupIds=sg-0677229e743e97317
---------------------------------------
Executing command : aws eks create-cluster --name cktestclustermm --role-arn arn:aws:iam::669678783832:role/cbd-eks-role --resources-vpc-co
nfig subnetIds=subnet-06d16b69a017ce7a6,subnet-07d794f0b706800a3,subnet-0a4aee63ad0296add,securityGroupIds=sg-0677229e743e97317
{

    "cluster": {

        "status": "CREATING",

        "name": "cktestclustermm",

        "certificateAuthority": {},

        "roleArn": "arn:aws:iam::669678783832:role/cbd-eks-role",

        "resourcesVpcConfig": {

            "subnetIds": [

                "subnet-06d16b69a017ce7a6",

                "subnet-07d794f0b706800a3",

                "subnet-0a4aee63ad0296add"

            ],

            "vpcId": "vpc-074ac91b494db16f2",

            "securityGroupIds": [

                "sg-0677229e743e97317"

            ]

        },

        "version": "1.11",

        "arn": "arn:aws:eks:us-east-2:669678783832:cluster/cktestclustermm",

        "platformVersion": "eks.1",

        "createdAt": 1549050837.583

    }

}

Checking completion status...
Checking attempt #0
Status :: "CREATING"
Checking attempt #1
Status :: "CREATING"
Checking attempt #2
Status :: "CREATING"
Checking attempt #3
Status :: "CREATING"
Checking attempt #4
Status :: "CREATING"
Checking attempt #5
Status :: "ACTIVE"
---------------------------------------
Executing command : aws eks update-kubeconfig --name cktestclustermm
Updated context arn:aws:eks:us-east-2:669678783832:cluster/cktestclustermm in /Users/mmaster/.kube/config

---------------------------------------
Running aws cloudformation create-stack --stack-name cktestclustermm-nodes --template-url https://amazon-eks.s3-us-west-2.amazonaws.com/clo
```

```
udformation/2018-12-10/amazon-eks-nodegroup.yaml --parameters ParameterKey=ClusterName,ParameterValue=cktestclustermm ParameterKey=ClusterC
ontrolPlaneSecurityGroup,ParameterValue=sg-0677229e743e97317 ParameterKey=NodeGroupName,ParameterValue=cktestclustermm-eks-nodes ParameterK
ey=NodeAutoScalingGroupMinSize,ParameterValue=3 ParameterKey=NodeAutoScalingGroupMaxSize,ParameterValue=3 ParameterKey=NodeInstanceType,Par
ameterValue=m4.4xlarge ParameterKey=NodeImageId,ParameterValue=ami-053cbe66e0033ebcf ParameterKey=KeyName,ParameterValue=cb-day-se Parameter
rKey=VpcId,ParameterValue=vpc-074ac91b494db16f2 ParameterKey=Subnets,ParameterValue='subnet-06d16b69a017ce7a6\,subnet-07d794f0b706800a3\,su
bnet-0a4aee63ad0296add' ParameterKey=NodeVolumeSize,ParameterValue=20 --capabilities CAPABILITY_IAM
--------------------------------------
Executing command : aws cloudformation create-stack --stack-name cktestclustermm-nodes --template-url https://amazon-eks.s3-us-west-2.amazo
naws.com/cloudformation/2018-12-10/amazon-eks-nodegroup.yaml --parameters ParameterKey=ClusterName,ParameterValue=cktestclustermm Parameter
Key=ClusterControlPlaneSecurityGroup,ParameterValue=sg-0677229e743e97317 ParameterKey=NodeGroupName,ParameterValue=cktestclustermm-eks-node
s ParameterKey=NodeAutoScalingGroupMinSize,ParameterValue=3 ParameterKey=NodeAutoScalingGroupMaxSize,ParameterValue=3 ParameterKey=NodeInst
anceType,ParameterValue=m4.4xlarge ParameterKey=NodeImageId,ParameterValue=ami-053cbe66e0033ebcf ParameterKey=KeyName,ParameterValue=cb-day
-se ParameterKey=VpcId,ParameterValue=vpc-074ac91b494db16f2 ParameterKey=Subnets,ParameterValue='subnet-06d16b69a017ce7a6\,subnet-07d794f0b
706800a3\,subnet-0a4aee63ad0296add' ParameterKey=NodeVolumeSize,ParameterValue=20 --capabilities CAPABILITY_IAM
{

    "StackId": "arn:aws:cloudformation:us-east-2:669678783832:stack/cktestclustermm-nodes/8640ef40-265c-11e9-8cdb-0659c08a3f80"

}

Checking completion status...
Checking attempt #0
Status :: "CREATE_IN_PROGRESS"
Checking attempt #1
Status :: "CREATE_IN_PROGRESS"
Checking attempt #2
Status :: "CREATE_COMPLETE"
--------------------------------------
Executing command : curl -O https://amazon-eks.s3-us-west-2.amazonaws.com/cloudformation/2018-12-10/aws-auth-cm.yaml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current

                                 Dload  Upload   Total   Spent    Left  Speed

100   282  100   282    0     0    828      0 --:--:-- --:--:-- --:--:--   826

Adding key NodeInstanceRole with value arn:aws:iam::669678783832:role/cktestclustermm-nodes-NodeInstanceRole-9TNSS18FKU3L
Adding key NodeSecurityGroup with value sg-0c3e6913ef7c0f3cf
--------------------------------------
Executing command : kubectl apply -f aws-auth-cm.yaml
configmap/aws-auth created

--------------------------------------
Executing command : kubectl get -n kube-system configmap/aws-auth -o yaml > aws-auth-patch.yaml
--------------------------------------
Executing command : kubectl apply -n kube-system -f aws-auth-patch.yaml
configmap/aws-auth configured
```

This script will then run through the different steps to create the EKS Cluster, which can take up to 20 minutes. When completed you should see the output shown above.

Step 5: Verify number of nodes

Make sure number of nodes requested in parameters.py is what has been deployed ``` $ kubectl get nodes

NAME STATUS ROLES AGE VERSION ip-192-168-166-206.us-east-2.compute.internal Ready 11m v1.11.5 ip-192-168-248-36.us-east-2.compute.internal Ready 11m v1.11.5 ip-192-168-64-16.us-east-2.compute.internal Ready 11m v1.11.5 ```

7.3. Custom Certificate Setup

Couchbase supports transport layer security (TLS) in order to encrypt communications on the wire and provide mutual authentication between peers. Couchbase clients (including the Couchbase Autonomous Operator) require the usage of TLS in order to communicate with the Couchbase cluster.

The basic requirements are:

- A certificate authority (CA) certificate which will be used by all actors to validate that peer certificates have been digitally signed by a trusted CA.
- A server certificate/key pair for all nodes in the Couchbase cluster. If you are using a hierarchy of intermediate CAs, these must be appended to the client certificate, ending in the intermediate CA that is signed by the top-level CA. Server certificates must have a subject alternative name (SAN) set so that a client can assert that the host name that it's connecting to is the same as that in the certificate.
- Couchbase currently supports using wildcard entries only. For example, **.cluster.domain.svc** where cluster is the name of the CouchbaseCluster resource, and domain is the namespace that the cluster is running in (typically **default**).

Important: TLS certificates are installed as part of the pod creation process, and cannot be enabled on an existing cluster.

Creating Certificates

Creating X.509 certificates is beyond the scope of this documentation and is given only for illustrative purposes only.

EasyRSA

EasyRSA by OpenVPN makes operating a public key infrastructure (PKI) relatively simple, and is the recommended method to get up and running quickly.

First clone the repository:

```
$ git clone http://github.com/OpenVPN/easy-rsa

Cloning into 'easy-rsa'...
warning: redirecting to https://github.com/OpenVPN/easy-rsa/
remote: Enumerating objects: 53, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (37/37), done.
remote: Total 1313 (delta 17), reused 44 (delta 16), pack-reused 1260
Receiving objects: 100% (1313/1313), 5.53 MiB | 2.12 MiB/s, done.
Resolving deltas: 100% (594/594), done.
```

Initialize and create the CA certificate/key. You will be prompted for a private key password and the CA common name (CN), something like Couchbase CA is sufficient. The CA certificate will be available as pki/ca.crt.

```
$ cd easy-rsa/easyrsa3
```

```
$ ./easyrsa init-pki

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: ~/pki
```

```
$ ./easyrsa build-ca

Using SSL: openssl LibreSSL 2.6.5

Enter New CA Key Passphrase:
Re-Enter New CA Key Passphrase:
Generating RSA private key, 2048 bit long modulus
....+++
.....................................................................................+++
e is 65537 (0x10001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:Couchbase CA

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
~/pki/ca.crt
```

Note: Specify a passphrase, for this demo we will use “couchbase”. It will also prompt you to add a CN for the Certificate Authority, for this demo we will use “Couchbase CA”.

Generate Wildcart Certificate

Create a server wildcard certificate and key to be used on Couchbase Server pods. The Operator/clients will access the pods via Kubernetes services (for example **<SERVICE*NAME-0000*>.<*CLUTSER*NAME>.<NAMESPACE>.svc**) so this needs to be in the SAN list in order for a client to verify the certificate belongs to the host that is being connected to.

To this end, we add in a **--subject-alt-name**, this can be specified multiple times in case your client uses a different method of addressing. The key/certificate pair can be found in **pki/private/couchbase-server.key** and **pki/issued/couchbase-server.crt** and used as **pkey.pem** and **chain.pem**, respectively, in the **serverSecret**.

Run this command by replacing the parameters you chose. In our case we have CLUSTER*NAME*=***ckdemo***, *NAMESPACE*=***emart***, *DOMAIN*NAME=sewestus.com:

```
Sample command
$ ./easyrsa --subject-alt-name=”DNS:*.<CLUSTER_NAME>.<NAMESPACE>.svc,DNS:*.<NAMESPACE>.svc,\
  DNS:*.<CLUSTER_NAME>.<DOMAIN_NAME>” build-server-full couchbase-server nopass
```

```
$ ./easyrsa --subject-alt-name="DNS:*.ckdemo.emart.svc,DNS:*.emart.svc,\
DNS:*.ckdemo.sewestus.com" build-server-full couchbase-server nopass



Using SSL: openssl LibreSSL 2.6.5
Generating a 2048 bit RSA private key
..................................+++
............+++
writing new private key to '~/pki/easy-rsa-19774.jlNiVl/tmp.vnxTqG'
-----
Using configuration from ~/pki/easy-rsa-19774.jlNiVl/tmp.2QJ8Jq
Enter pass phrase for ~/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'couchbase-server'
Certificate is to be certified until Aug  10 19:18:11 2022 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated
```

Note: password-protected keys are not supported by Couchbase Server or the Operator.

Private Key Formatting

Due to an issue with Couchbase Server’s private key handling, server keys need to be PKCS#1 formatted. This can be achieved with the following commands:

```
$ cp pki/private/couchbase-server.key pkey.key
```

```
$ openssl rsa -in pkey.key -out pkey.key.der -outform DER

writing RSA key
```

```
$ openssl rsa -in pkey.key.der -inform DER -out pkey.key -outform PEM

writing RSA key
```

Next copy TLS certificate file to current directory for easy access. `$ cp pki/issued/couchbase-server.crt chain.pem` Also make copy of certificate and private key to `tls-cert-file` and `tls-private-key-file` to be used in creating `couchbase-operator-admission` secret in later section:

```
$ cp chain.pem tls-cert-file
$ cp pkey.key tls-private-key-file
```