1. Create EC2 Instance with Ubuntu AMI with t2.large
2. Login to EC2 from your Terminal
3. Become root user
   1. sudo -i

**Install the IAM Authenticator**

curl -o aws-iam-authenticator https://amazon-eks.s3.us-west-2.amazonaws.com/1.15.10/2020-02-22/bin/linux/amd64/aws-iam-authenticator

chmod +x ./aws-iam-authenticator

sudo mv ./aws-iam-authenticator /usr/local/bin

aws-iam-authenticator help

**Installing the EKS Cluster**

1. Setup kubectl  
   a. Download kubectl version 1.20  
   b. Grant execution permissions to kubectl executable  
   c. Move kubectl onto /usr/local/bin  
   d. Test that your kubectl installation was successful

curl -o kubectl <https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl>

chmod +x ./kubectl

mv ./kubectl /usr/local/bin

kubectl version --short --client

1. Setup eksctl  
   a. Download and extract the latest release  
   b. Move the extracted binary to /usr/local/bin  
   c. Test that your eksclt installation was successful

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

1. Create an IAM Role and attach it to EC2 instance  
     
   Atach AdministratorAccess Policy to IAM Role
2. Create your cluster and nodes

eksctl create cluster --name my-cluster --region us-east-2 --node-type t2.small

1. Once the Cluster is created, Verify it by executing below commands

* kubectl get nodes
* kubectl get svc

1. Launch the application
   1. Create Manifest file (nginx.yaml)
   2. kubectl create -f nginx.yaml
2. kubectl get service and copy the External IP for nginx-service and access it from browser

**ARGO CD SETUP and TESTING**

**INSTALL ARGO CD**

kubectl create namespace argocd

kubectl apply -n argocd -f <https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml>

**CONFIGURE ARGOCD**

Expose argocd-server

By default argocd-server is not publicaly exposed.

kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

apt install jq -y

Wait about 2 minutes for the LoadBalancer creation

export ARGOCD\_SERVER=`kubectl get svc argocd-server -n argocd -o json | jq --raw-output '.status.loadBalancer.ingress[0].hostname'`

**INSTALL ARGO CD CLI**

sudo curl --silent --location -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/download/v2.4.7/argocd-linux-amd64

sudo chmod +x /usr/local/bin/argocd

**LOGIN**

The initial password is autogenerated with the pod name of the ArgoCD API server:

export ARGO\_PWD=`kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d`

Using admin as login and the autogenerated password:

argocd login $ARGOCD\_SERVER --username admin --password $ARGO\_PWD --insecure

**DEPLOY AN APPLICATION**

Fork application repository

First step is to create a fork for the Github application we will deploy.

Login to github, go to: https://github.com/brentley/ecsdemo-nodejs.git and Fork the repo

Then into your select the https URL by clicking into button Clone or download:

This URL will be needed when we will configure the application into ArgoCD.

CONTEXT\_NAME=`kubectl config view -o jsonpath='{.current-context}'`

argocd cluster add $CONTEXT\_NAME

Type Y if it is asked to continue for Insecure access

Configure the application and link to your fork (replace the GITHUB\_USERNAME):

kubectl create namespace ecsdemo-nodejs

argocd app create ecsdemo-nodejs --repo https://github.com/skattagmail/ecsdemo-nodejs.git --path kubernetes --dest-server https://kubernetes.default.svc --dest-namespace ecsdemo-nodejs

Application is now setup, let’s have a look at the deployed application state:

argocd app get ecsdemo-nodejs

You can see the details of application in the Console, if you wanted to access the application from ARGO CD UI

* Copy the URL and paste it in browser
* Provide Username as admin

**To get the admin password, please follow below steps**

* kubectl get secret -n argocd
* kubectl edit secret argocd-initial-admin-secret -n argocd
* Copy the Password
* echo <Password> | base64 --decode
* Copy the password and login to Argo CD UI (Username is admin)

We can see that the application is in an OutOfSync status since the application has not been deployed yet. We are now going to sync our application:

argocd app sync ecsdemo-nodejs

After a couple of minutes our application should be synchronized.

**UPDATE THE APPLICATION**

Our application is now deployed into our ArgoCD. We are now going to update our github repository synced with our application

Update your application

Go to your Github fork repository:

Update spec.replicas: 2 in ecsdemo-nodejs/kubernetes/deployment.yaml

Add a commit message and click on Commit changes

Access ArgoCD Web Interface

To deploy our change we can access to ArgoCD UI. Open your web browser and go to the Load Balancer url:

echo $ARGOCD\_SERVER

Login using admin / $ARGO\_PWD. You now have access to the ecsdemo-nodejds application. After clicking to refresh button status should be OutOfSync:

This means our Github repository is not synchronised with the deployed application. To fix this and deploy the new version (with 2 replicas) click on the sync button, and select the APPS/DEPLOYMENT/DEFAULT/ECSDEMO-NODEJS and SYNCHRONIZE:

After the sync completed our application should have the Synced status with 2 pods:

All those actions could have been made with the Argo CLI also.

**CLEANUP**

argocd app delete ecsdemo-nodejs -y

watch argocd app get ecsdemo-nodejs

Wait until all resources are cleared with this message:

FATA[0000] rpc error: code = NotFound desc = applications.argoproj.io "ecsdemo-nodejs" not found

And then delete ArgoCD from your cluster:

kubectl delete -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml

Delete namespaces created for this chapter:

kubectl delete ns argocd

kubectl delete ns ecsdemo-nodejs

You may also delete the cloned repository ecsdemo-nodejs within your GitHub account.

**Delete the Cluster**

eksctl delete cluster my-cluster --region us-east-2

After the Cluster is deleted successfully then Terminate the Kubernetes Master Server