**Pre-requisites**

1. We need an Ubuntu instance from where we will run all our commands. Follow steps in below URL to install an ubuntu instance.

<https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10#1-overview>

1. Once Ubuntu is installed, below dependencies needs to be installed.
   * Python
   * AWS CLI
   * EKSCTL
   * KUBECTL
   * HELM

**Python Installation**

1. Open the ubuntu terminal and run the below command to install python.

*sudo apt install python3.7*

1. Run below command to validate the python installation. If installed correctly, you will get the version of the python.

*python --version*

**AWS CLI – Installation**

1. Run the below command to install the aws-cli package.

*pip install --upgrade awscli && hash -r*

1. Once installed, run below command to setup the AWS account in your machine.

*aws configure*

1. Provide access keys, secret keys, region, desired output format

**EKSCTL - Installation**

1. eksctl is the command line tool used to install and communicate with our EKS cluster.
2. Run the command below to install eksctl.

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

1. Once installed, run below command to make eksctl available for any user.

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

1. Finally run below command to validate the installation. If installed correctly, you will get the version of the eksctl.

*eksctl version*

**KUBECTL - Installation**

1. kubectl is the command line tool used to communicate with kunernetes running in our EKS cluster.
2. Run below command to install kubectl.

*curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"*

1. Run below commands to make kubectl accessible by any user.

*chmod +x ./kubectl*

*sudo mv ./kubectl /usr/local/bin/kubectl*

1. Finally run command below to validate the installation. If installed correctly, you will get the version of the kubectl.

*kubectl version --client*

**HELM - Installation**

1. Helm is the Kubernetes deployment tool used to install airflow in to the EKS cluster.
2. Run the command below to install Helm.

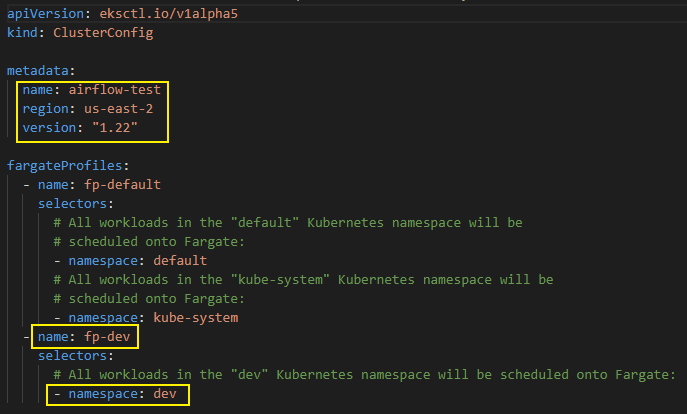
*curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash*

1. Finally run command below to validate the installation. If installed correctly, you will get the version of the helm.

*helm version --short*

**Steps to create EKS Cluster**

1. Once all the pre-requisite installations are completed successfully, we can start creating the cluster.
2. Open cluster.yml file and make relevant changes as highlighted in below screenshot.



1. Give a relevant cluster name in the “metadata.name” field and the aws region in “metadata.region” field. You can check the latest version of Kubernetes available for EKS and update the value in “metadata.version” field.
2. Give relevant names to “fargateProfiles.name” and “fargateProfiles.selectors.namespace”. Make sure that the value of field “fargateProfiles.selectors.namespace” is same as that of namespace that we will create in step 12 below.
3. Copy the cluster.yml file to the ubuntu workspace.
4. Run the below command in the directory where the cluster.yml file is available.

*eksctl create cluster -f cluster.yml*

1. The cluster creation will take some time. You can verify the installation in the AWS management console by navigating to Cloudformation service or EKS service.
2. Once installation is complete, run below command to validate whether the cluster is installed correctly.

*kubectl get nodes*

1. You should be able to see the nodes of your cluster as shown in below screenshot.

Text

Description automatically generated

1. Once the cluster is created successfully, next step is to create a namespace. Namespace helps in isolating our applications in the same EKS cluster. Suppose if you need two different environments – dev and prod, to be run in isolation, we can create two namespaces and install our applications in them, instead of having two different EKS clusters.
2. Open the namespace.yml file and make the changes highlighted in below screenshot.

Text

Description automatically generated

1. Provide relevant name in the “metadata.name” field. Make sure the name is same as what we gave in step 4.
2. Copy the namespace.yml file to the ubuntu workspace and run the below command to create the namespace.

*kubectl apply -f namespace.yml*

**Environment Variables**

1. Some of the constant values can be stored into the environment variable of our ubuntu workspace, so that it will be easier to run our installation commands.
2. Make sure to execute these commands whenever you open new ubuntu instance.
3. Replace the highlighted part with the actual values. You can get the EKS Stack name from the cloudformation aws service, where a stack would have been created for our EKS cluster creation.

*CLUSTER\_NAME="YOUR\_CLUSTER\_NAME"*

*NAMESPACE=”YOUR\_NAMESPACE\_NAME”*

*AWS\_REGION="YOUR\_AWS\_REGION"*

*STACK\_NAME="YOUR\_EKS\_STACK\_NAME"*

*VPC\_ID=$(aws cloudformation describe-stacks --stack-name "$STACK\_NAME" | jq -r '[.Stacks[0].Outputs[] | {key: .OutputKey, value: .OutputValue}] | from\_entries' | jq -r '.VPC')*

*AWS\_ACCOUNT\_ID=$(aws sts get-caller-identity | jq -r '.Account')*

**Install Persistent Storage**

1. As we are using fargate profile as our worker nodes, the processed data by a worker pod will get deleted when a pod is deleted. This should not be the case for our dags folder, where we will store our dag files. To overcome this, we will be using external EFS storage to store our dags.
2. In AWS management console, navigate to EFS service and create a new EFS storage with same VPC linked to our EKS cluster.
3. Open efs.yml file and make necessary changes highlighted in below screenshot.

Text

Description automatically generated

1. In both “storage” fields enter the maximum storage you need. “1Gi” denotes one GB. In “spec.csi.volumeHandle” enter the EFS filesystem id that can be retrieved from the AWS EFS home page.

Graphical user interface, application

Description automatically generated

1. Once the changes are made, copy the efs.yml to the ubuntu workspace and run the below command.

*kubectl apply -f efs.yml -n $NAMESPACE*

1. This will mount the EFS storage to the dev namespace of our EKS cluster.

**Attach AWS IAM Policy to dev Namespace**

1. Airflow in our EKS cluster will need access to interact with other AWS services like S3, Glue. To achieve this, we will create a service account and attach a policy to it. This policy will have all the necessary access statements in it.
2. First create a policy in AWS IAM, with relevant access needed for the airflow.

Graphical user interface, text, application

Description automatically generated

1. OIDC authentication must be configured in first place, to attach service accounts to the cluster. Run the below command to approve OIDC authentication in this cluster.

*eksctl utils associate-iam-oidc-provider --cluster $CLUSTER\_NAME --approve*

1. Now execute the below command. Replace the highlighted values with relevant details. Replace “POLICY\_ARN” with the ARN of the policy.

*eksctl create iamserviceaccount --name dev-service-account --namespace $NAMESPACE --cluster $CLUSTER\_NAME --role-name "airflow-eks-role" --attach-policy-arn POLICY\_ARN --approve --override-existing-serviceaccounts*

1. Above command will create a new role with the name “airflow-eks-role”. Idetify the ARN number of that role and replace “ROLE\_ARN” in below command as highlighted.

*kubectl annotate serviceaccount -n $NAMESPACE service-account-name eks.amazonaws.com/role-arn=ROLE\_ARN*

**Install Airflow**

1. Next, we will install airflow in the cluster using Helm charts.
2. First, we need to add the Airflow Helm repo in our local machine. To do that, execute the below set of commands.

*helm repo add airflow https://airflow-helm.github.io/charts*

*helm repo update*

1. Above commands would have downloaded the airflow helm repo with the name “airflow”
2. All the configuration details regarding our airflow will be declared in values.yml file. Changes to be done in values.yml will be discussed separately. Please go through that post and modify the values.yml file accordingly.
3. Once the modification to values.yml file is done, copy it to the ubuntu workspace and run the below command.

*helm install airflow airflow/airflow --namespace $NAMESPACE --values values.yml*

1. Once you receive “Successfully installed” message, run below command to check whether the airflow pods are up and running.

*kubectl get pods -n $NAMESPACE*

1. Status of all the pods must be “Running”. If not, it means there is some issue with the installation, and we need to debug to identify the issue. This will be explained in separate document.

**Create Domain for Airflow UI**

1. We need a domain name for our airflow UI page for two reasons. Firstly, a domain name is easy to remember compared to the URL provided by the AWS load balancer. Secondly, the airflow UI can be accessed securely using HTTPS URL, only when we have a domain name.
2. Using AWS Route53 service, create a new domain as per your requirement. Steps regarding this will documented separately.
3. Once domain name is created, generate an SSL certificate from the AWS ACM service. This certificate is used to access our airflow web page securely. Steps will be documented separately.

**Setup Ingress - Expose Airflow Web Server**

1. The airflow web server will be running in one of the airflow pods. Ingress must be configured to expose this pod, only then the airflow UI will be accessible outside the cluster.
2. Copy the ingress\_iam\_policy.json file to the ubuntu workspace and execute the below command to create IAM policy required for the ingress setup.

*aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document ingress\_iam\_policy.json*

1. Execute below command to create a service account with the policy created in above step attached to it. This service account will be used for the ingress controller setup.

*eksctl create iamserviceaccount --cluster=$CLUSTER\_NAME --namespace=kube-system --name=aws-load-balancer-controller --attach-policy-arn=arn:aws:iam::$AWS\_ACCOUNT\_ID:policy/AWSLoadBalancerControllerIAMPolicy --override-existing-serviceaccounts --approve*

1. We will use AWS load balancer to expose our airflow web server to the public internet. Run below commands to install the AWS Load balancer controller in to the cluster.

*helm repo add eks https://aws.github.io/eks-charts*

*kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller//crds?ref=master"*

*helm install aws-load-balancer-controller eks/aws-load-balancer-controller --set clusterName=$CLUSTER\_NAME --set serviceAccount.create=false --set region=$AWS\_REGION --set vpcId=$VPC\_ID --set serviceAccount.name=aws-load-balancer-controller -n kube-system*

1. Execute below command to find the labels associated with the pods. Note down the labels associated with the web server pod. These labels are used to identify the web server pod while creating the ingress service.

*kubectl get pods -n $NAMESPACE --show-labels*

A screenshot of a computer

Description automatically generated with medium confidence

1. Open airflow-web-service.yml and make the necessary changes. Give the namespace name in “metadata.namespace”. Give the labels we noted down in step 5, as key value pairs in the field “spec.selector” as highlighted in below screenshot.



1. Copy the airflow-web-service.yml to ubuntu workspace and execute below command.

*kubectl apply -f airflow-web-service.yml*

1. Open airflow-web-ingress.yml file and make the necessary changes as highlighted in below screenshot. Give Namespace name in “metadata.namespace” field. Retrieve the ARN of ACM certificate you created already and paste it to the “metadata.annotations. alb.ingress.kubernetes.io/certificate-arn” field. Give the labels we noted down in step 5, as key value pairs in the field “metadata.labels”. Provide the domain name you created already in both “spec.rules.host” and “spec.tls.hosts” fields.

Text

Description automatically generated

1. Copy the airflow-web-ingress.yml file to the ubuntu workspace and execute below command.

*kubectl apply -f airflow-web-ingress.yml*

1. The above command will create a load balancer and attach it to the web server pod. Now you will be able to access the airflow web UI using the URL provided by the AWS load balancer. To retrieve the URL, execute the below command.

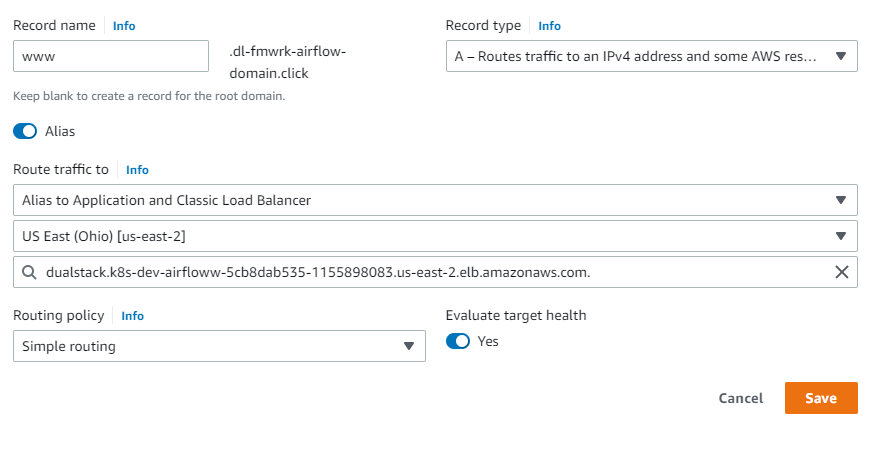
kubectl get ingress -n $NAMESPACE



1. Retrieve the URL from “ADDRESS” column and fire it in your browser. You should be able to view the login page of airflow.
2. The airflow web UI to be accessible using the domain name, follow the steps in next section.

**Configure Domain name to Airflow web UI**

1. Open AWS management console and navigate to Route 53 service.
2. Go to Hosted Zones and click on yourdomain.com.
3. Click on “Create Record” button.
4. In “Record name” field provide “www”. In the drop downs “Record type”, “Route traffic to” and “Routing policy” select the values as shown in below screenshot.



1. Select your region in the drop down below “Route traffic to” drop down. In the next drop down, select the load balancer which you have created in the previous topic.
2. Enable both “Alias” and “Evaluate target health” slider buttons.
3. Click on “Create” button once everything is selected correctly.

**Add HTTPS listener to your load balancer**

1. In AWS, go to “EC2 Console”, then “Load Balancing” and “Load Balancers” from left menu.
2. Click on your ALB name checkbox, then select “Listeners” Tab.
3. Click on “Add Listener”. Choose “*HTTPS”* and “*Port”* (443 usually). In “Default action(s)” select “Forward to...” option and then your “Target Group” for your Load Balancer. “Weight” it's ok in 1. Choose the appropriate “Security policy” and then add your “Default SSL certificate” that you have created from “AWS Certificate Manager”. Then click on “Save”.
4. Now, you can go to browser and navigate your domain URL that you have created. You should be able to access airflow login page securely.