How to build docker image and push to ECR using Github Actions.

Pre-requisites:

1. ECR registry
2. Github account
3. AWS account with credentials.
4. EKS cluster server with helm installed

Step 1:

First we create repository in ECR.

Login to AWS console. Search for ECR.

ECR -> Get started.

Give name to repo. Then click on create.

A screenshot of a computer

Description automatically generated

A screenshot of a computer error message

Description automatically generated

Now repo will be created.

**Step 2:**

Setup EKS cluster.

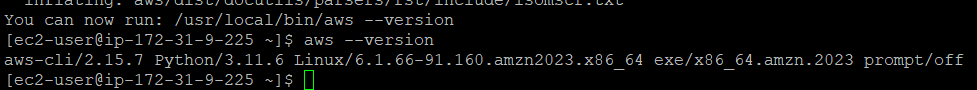
**Install AWS cli:**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

sudo apt install unzip

unzip awscliv2.zip

sudo ./aws/install



Okay now after installing the AWS CLI, let's configure the **AWS CLI** so that it can authenticate and communicate with the AWS environment.

aws configure

A black screen with white text

Description automatically generated

## **Install and Setup Kubectl**

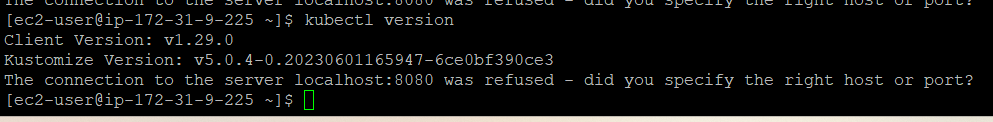
Moving forward now we need to set up the [**kubectl**](https://kubernetes.io/docs/reference/kubectl/overview/) also onto the EC2 instance.

curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin

kubectl version

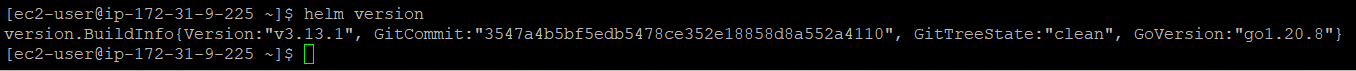


## **Install Helm chart**

$ curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

$ chmod 700 get\_helm.sh

$ ./get\_helm.sh



This way we install all AWS CLI, kubectl, eksctl and Helm.

**Install Terraform:**

Follow below steps to install terraform on AmazonLinux.

sudo yum install -y yum-utils shadow-utils

sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

sudo yum -y install terraform

A black screen with white text

Description automatically generated

# **Creating an Amazon EKS cluster using terraform**

Code available in https://github.com/ksnithya/blue-green.git

git clone <https://github.com/ksnithya/blue-green.git>

cd blue-green

terraform init

terraform plan

terraform apply

aws eks --region ap-south-1 update-kubeconfig --name eks\_cluster\_demo

**Step 3:**

If we are going to use AWS and ECR in code we need to save their credentials in repo under secrets. First we create the workflow to move image to ECR.

Login to github and go inside our repository and click on settings.

A close-up of a computer screen

Description automatically generatedUnder “General tab” select “Secret and variables” and click on actions -> New repository secrte.

A screenshot of a computer

Description automatically generated

We can add our secret variable.

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

Step 3:

Now we can create our workflow file. Go inside the repo. Click on Action.

A close-up of a computer screen

Description automatically generated

A close-up of a text

Description automatically generated

name: Build and push image to ECR

on:

# push:

workflow\_dispatch:

inputs:

tag:

description: "Enter image tag"

required: true

type: string

jobs:

build:

name: Build Image

runs-on: ubuntu-latest

steps:

- name: Check out code

uses: actions/checkout@v2

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ secrets.AWS\_REGION }}

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v2

- name: Build, tag, and push image to Amazon ECR

env:

ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

ECR\_REPOSITORY: python-flask-demo

IMAGE\_TAG: ${{ github.event.inputs.tag }}

run: |

cd blue/flask-demo

docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG .

docker images

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

After wiring the content click on commit. Now work flow is created.

Click on “Run workflow” and give tag name. Click on “Ru workflow”.

A screenshot of a computer

Description automatically generated

Now our job will be ran successfully.

A screenshot of a computer

Description automatically generated

We can see image in our Repo.

A screenshot of a computer

Description automatically generated

**Step 4:**

Now we modify the workflow to connect to EKS and deploy app using helm chart.

I have already have one python demo app chart in my repo.

How to create chart:

**[ec2-user@ip-172-31-9-225 helm]$ helm create demo1**

Creating demo1

[ec2-user@ip-172-31-9-225 helm]$ cd demo1/

[ec2-user@ip-172-31-9-225 demo1]$ ls -l

total 8

-rw-r--r--. 1 ec2-user ec2-user 1141 Jan 30 07:42 Chart.yaml

drwxr-xr-x. 2 ec2-user ec2-user 6 Jan 30 07:42 charts

drwxr-xr-x. 3 ec2-user ec2-user 162 Jan 30 07:42 templates

-rw-r--r--. 1 ec2-user ec2-user 2250 Jan 30 07:42 values.yaml

[ec2-user@ip-172-31-9-225 demo1]$

Sample configuration files will be available under “template” folder.

[ec2-user@ip-172-31-9-225 demo1]$ cd templates/

[ec2-user@ip-172-31-9-225 templates]$ ls -l

total 28

-rw-r--r--. 1 ec2-user ec2-user 1739 Jan 30 07:42 NOTES.txt

-rw-r--r--. 1 ec2-user ec2-user 1762 Jan 30 07:42 \_helpers.tpl

-rw-r--r--. 1 ec2-user ec2-user 2159 Jan 30 07:42 deployment.yaml

-rw-r--r--. 1 ec2-user ec2-user 985 Jan 30 07:42 hpa.yaml

-rw-r--r--. 1 ec2-user ec2-user 2075 Jan 30 07:42 ingress.yaml

-rw-r--r--. 1 ec2-user ec2-user 355 Jan 30 07:42 service.yaml

-rw-r--r--. 1 ec2-user ec2-user 385 Jan 30 07:42 serviceaccount.yaml

drwxr-xr-x. 2 ec2-user ec2-user 34 Jan 30 07:42 tests

[ec2-user@ip-172-31-9-225 templates]$

Now we can modify this files based on our requirements.

We will remove all files and create deploy.yml and load.yml files.

[ec2-user@ip-172-31-9-225 templates]$ cat deploy.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: python-flask

spec:

replicas: 1

selector:

matchLabels:

app: python-flask

template:

metadata:

labels:

app: python-flask

spec:

containers:

- name: python-app

image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"

resources:

limits:

memory: "2Gi"

cpu: "500m"

requests:

memory: "1Gi"

cpu: "200m"

[ec2-user@ip-172-31-9-225 templates]$ cat load.yml

apiVersion: v1

kind: Service

metadata:

name: python-load

spec:

selector:

app: python-flask

type: LoadBalancer

ports:

- port: 5001

targetPort: 5001

protocol: TCP

We have moved the same chart structure to our repo.

Below is our full workflow file.

name: Build and push image to ECR

on:

# push:

workflow\_dispatch:

inputs: # user input for tag

tag:

description: "Enter image tag"

required: true

type: string

jobs:

build:

name: Build Image

runs-on: ubuntu-latest # our workflow will run on ubuntu server

steps:

- name: Check out code

uses: actions/checkout@v2

- name: Install kubectl # installing kubectl to connect to EKS cluster

uses: azure/setup-kubectl@v2.0

with:

version: 'v1.27.0' # default is latest stable

id: install

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ secrets.AWS\_REGION }}

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v2

- name: Build, tag, and push image to Amazon ECR

env:

ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

ECR\_REPOSITORY: python-flask-demo

IMAGE\_TAG: ${{ github.event.inputs.tag }}

run: |

cd blue/flask-demo

docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG .

docker images

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

- name: Update kube config # to update kubeconfig file in our local runner and to connect to our cluster.

run: |

aws eks --region ap-south-1 update-kubeconfig --name eks\_cluster\_demo

kubectl get nodes

- name: check chart exist

env:

ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

ECR\_REPOSITORY: python-flask-demo

IMAGE\_TAG: ${{ github.event.inputs.tag }}

run: |

echo "x = $(( `helm list|grep -w test|wc -l` ))"

if [ $x >= 1 ]; then

echo " Chart exist then upgrade"

helm upgrade test ./python-flask --set image.repository=$ECR\_REGISTRY/$ECR\_REPOSITORY --set image.tag=$IMAGE\_TAG

else

echo "Chart not exist. Create it"

helm install test ./python-flask --set image.repository=$ECR\_REGISTRY/$ECR\_REPOSITORY --set image.tag=$IMAGE\_TAG

fi

To run the job click on Action and our workflow name.

A screenshot of a computer

Description automatically generated

Click on “Run Workflow” -> Enter tag name.(We have defined this as input variable). Then click on green button.

A screenshot of a computer

Description automatically generated

Now job will be started.

A screen shot of a computer

Description automatically generated

Click on the job. We can see all the steps we have created.

A screenshot of a computer

Description automatically generated

Now our app is deployed.

A screenshot of a computer

Description automatically generated

Now we update our demo.py and rerun the job. Dockerfile exist on same repo under blue/flask-demo and updating python file.

A screenshot of a computer

Description automatically generated

Now we rerun the job.

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

<https://www.fosstechnix.com/build-and-push-docker-image-to-aws-ecr-using-github-actions/>

<https://www.fosstechnix.com/deploy-nodejs-helm-chart-on-eks-using-github-actions/>

<https://rtfm.co.ua/en/github-actions-docker-build-to-aws-ecr-and-helm-chart-deployment-to-aws-eks/>

<https://www.youtube.com/watch?v=cYrq4fwB1Gg>