# Real world solution for setting up an environment that is using DevOps technologies & practices for deploying apps and cloud services/ cloud infrastructure to AWS

<https://drive.google.com/drive/folders/1cfNxho3zpyZtaZWqyosUaMZXDmktWutn?usp=sharing> - Contains the recordings.

After installing AWS CLI, Terraform, Docket Desktop, Visual Studio Code

Head over to the aws console and create a user for programmatic access. Make sure to make note of the secret access key otherwise it cannot be retrieved you will have to create another user. Make sure to give the user admin access.

Username: Ray

Enter your access key id

Enter your secret key id

Section 3. AWS configure credentials to access aws at a programmatic level.

Once you have created your user with the admin access then do the following;

To login into your aws user;

In terminal:

#aws configure:

#enter access key id > secret access key id > enter the region (eu-west-2 (londons) > json as it’s a json key pair maybe?)

To verify if its working you can do > in terminal

#ws sts get-caller-identity

Can verify we are logged in.

<https://github.com/AdminTurnedDevOps/DevOps-The-Hard-Way-AWS/blob/main/README.md>  
this is the github we cloned > in the labs

Skipping 2 > creating cluster will be charged if we forget to shut it down.

So we will have a local Kubernetes and not the cloud one.

When running EKS it means running Kubernetes on the cloud - it requires creating a EKS VPC which we will ignore for this time

We are also skipping section 4 terraform - as we use terraform to create the EKS cluster.

Section 5

We will move onto section 5 - Docker - the purpose is to create a docker image from the app that the organisation is running on-prem (the uber app), containerise it, and store the container inside of a container repository. For the container repo, we will use AWS ECR.

So the guy created a uber app already for us hence we don’t need to create one.

The application we have is running on our machine, so say we want to run it on another machine docker over the issue and packages the application in something called container. You can then move this container onto another machine that has docker installed and it can be ran over in there also.

Graphical user interface, text, application

Description automatically generated

App and dockerfile is ready for us. Now we have to enter into the CLI.

#cd DevOps-The-Hard-Way-AWS

#cd Docker/

To run docker file ;

#docker build -t uberapp .

-t is for the console, the dot at the end is to say where the docker file is since we are already in the docker folder we specify dot meaning here.

#docker image ls > can see the image we just created.

#docker run -it uberapp > -it is interactive and whatever you called your docker image.

Now do same command with -itd it should return a long string

#docker run -itd uberapp > a96a46bbae23936287e0b6fc59f89d2408d4a190a8560620571030f9ab646e5c this is the container id string.

#docker container ls. > shows you your container is running

So what we have done till now is we had an application we containerised it into a image. Now we can push this image to any other directory or online. You wouldn’t need python to run it having docker should be enough. Next we will push it to a docker hub registry. Normally we would use a ECR amazon elastic container registry and this is where you can store images, share them deploy anywhere. We will use our own docker hub. But first login to docker hub

#docker login

#enter your id

#enter your password

#docker commit a96a46bbae23936287e0b6fc59f89d2408d4a190a8560620571030f9ab646e5c rihan97/uberapp:v1.1   
> we are commiting this image to our docker hub same way we push code to git we do git commit etc. so use container id, docker hub id and name of app and can add a tag.

#docker push rihan97/uberapp:v1.1 > now push the container/ image into your docker hub whatever you called it earlier   
Then verify in docker hub if its come through. Done

So far we had the app ready called uber app, we needed to containerise it, which we did, the dockerFile is already created for us, then we built the image in docker build meaning it creates a container as a image. Then when we do docker run it creates it as a container and we can then push it, anyone can pull and use it without having any dependencies on your pc. The next step from here is Kubernetes Step 6.

Step 6   
Kubernetes - The purpose of the Kubernetes section is to connect to EKS locally and to write the Kubernetes manifest to deploy the Python Uber app.

#kubectl get nodes > shows all the nodes we have

#kubectl config get-contexts   
> his shows us the list of our clusters we have installed for us it should only show our local cluster docker-desktop at this stage, so if it shows it means its enabled and working.   
Docker allows us to run one container, if we get more traffic on our app or something and requires scaling up, each container contains a specific amount of ram etc we can keep increasing the number of containers but its not really the solution. Kubernetes is the solution it lets you have multiple containers and can put these containers into pods and the whole process is much easier. The Kubernetes infrastructure consists of the control pane which is the master and controls everything and contains the most important parts of the Kubernetes cluster, you also have the worker nodes which are the machines/ servers that do the most work (i.e. running the pods etc), You can have more services in the Kubernetes cluster such as the scheduler (making sure pods are scheduled on the right nodes etc), api server etc and these each do specific things.

Diagram

Description automatically generated

#cd ..

#cd kubernetes\_manifest/ > go into this file

Run the containerised application as a Kubernetes pod, the guy already created one in the git hence we can use his, can see it in the deployment.yml

In Kubernetes whenever you want to create something you use a yaml file, it’s a language, you don’t need to know how cerate the codes for i.e. a pod etc you can go to the following site copy the code and paste it. <https://kubernetes.io/>

So a deployment is a high level of like a pod where you can have more then one pod inside. Some important info about deployment is;

appVersion, Kind, Metadata, spec, replicas, template what contains the pod, then at the end you have important info lik the spec, container: name, image, ports.

#touch uberdeploy.yaml > creates a new yaml file that is currently empty. Copy the code from the deployment.yml file and paste into here.

We didn’t copy the load balancer because in Kubernetes we have different types of services (load balancer (which is world wide open), cluster ip (local) and node port (local) , it’s a way to expose your application to the world, with a load balancer we get a link that we can use and since we are not running it in the cloud but locally we don’t need it.

We will use this uberdeploy file so edit the details in the file with your image name rihan97/uberapp:v1.1 (docker hub id followed by the name of the app)

We will try to expose our app live on Kubernetes locally. But first run the app to see if it runs

#kubectl apply -f uberdeploy.yaml > says its created

#kubectl get deployments.apps > shows our uberapp is running

#kubectl get pods > can see all the pods within the deployment.yaml file are all running

Copy one of the pods id

Sicne we are not using the cloud we will next use a shortcut to run it

#sudo kubectl port-forward uber-ui-5bc7864996-bqf55 85:5000 > this show us the app is running fine we are opening port 85 for this

The right port is the container you opened 5000, and the left ports are random which are local ones which are not used.

So it says we are forwarding from local host 85 -> 5000. we can go to localhost port 85 from the browser and check this.

Localhost:85 > it works

So we have got our containerised app to be shown running locally in a pod. If we used the load balancer service then it would be on the internet running automatically on each of the three pod.

Graphical user interface, application, website

Description automatically generated

End of lab - thank you for reviewing.