**Weight & Biases Server Deployment on AWS EKS using Helm Charts.**

This doc walks you through the steps on how I deployed Weight & Biases in AWS EKS using Helm Charts.

First this First.

**What is Weight & Biases**

Weight & Biases is the AI Developer Platform with tools for training models, fine tuning models and leveraging foundational models.

It has 4 main core components - **Core, Models, Weave and Inference.**

It also has a Platform (Weight & Biases Platform) which is the foundational infrastructure, tooling, governance scaffolding which support the Weight & biases Products like Core, Weave and Model.

Weight & Biases Platform is available in 3 different deployment options.

1.**Multi-Teanant (SAAS)** - Fully managed service deployed in Weight & Biases Cloud Infrastructure which provides seamless access to Weight & Biases Products at a desired scale,

Cost efficient for pricing and continuous updates for latest features and functionalities.

2**. Dedicated Cloud** - Single Tenant. Fully managed service deployed in Weight & Biases Cloud Infrastructure. This is good for if you have a strong compliance and data residency requirements.

3**.Customer Managed** - Customers are responsible for provisioning and managing the infrastructure and deploying the Weight & Biases server. They also need to take care of maintenance, backups along with scaling, security and Performance characteristics.

Ways to Deploy the Weight & Server.

Weight & Biases offer 3 Options to deploy the Weight & Biases Server.

1.Terraform

2.Helm Chart

3.Operators.

Have explored all 3 options but decided to pick Helm Charts for this Assignments and plan to try the operators option next.

So here we go with the actual Steps.

**1. Prerequisites**

AWS Account with AdminPrivileges to Provision the AWS Resources (VPC, EKS etc.)

AWS CLI - Access AWS resources Programmatically.

Eksctl - Create the EKS cluster.

kubectl - Interact with the cluster.

Helm - Deploy W &B Server.

Python - Python script for log runs.

2.**EKS Cluster Creation.**

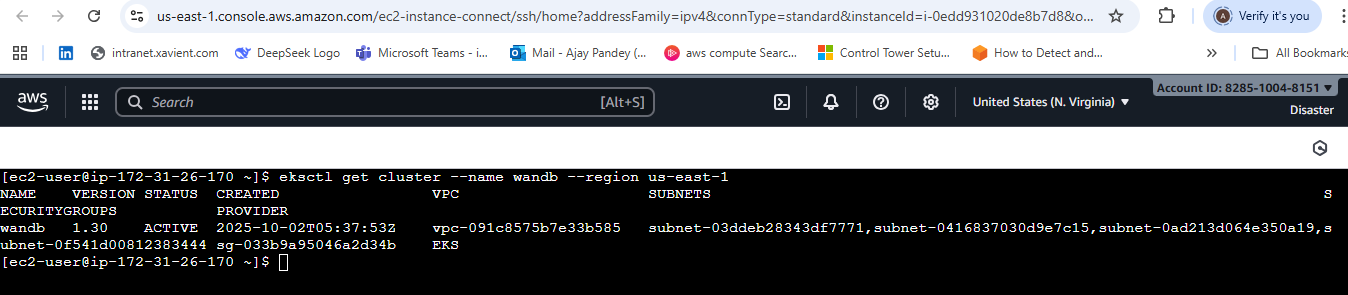
Eksctl create cluster –name wandb –region us-east-1

It creates a VPC with two public & Private Subnets.

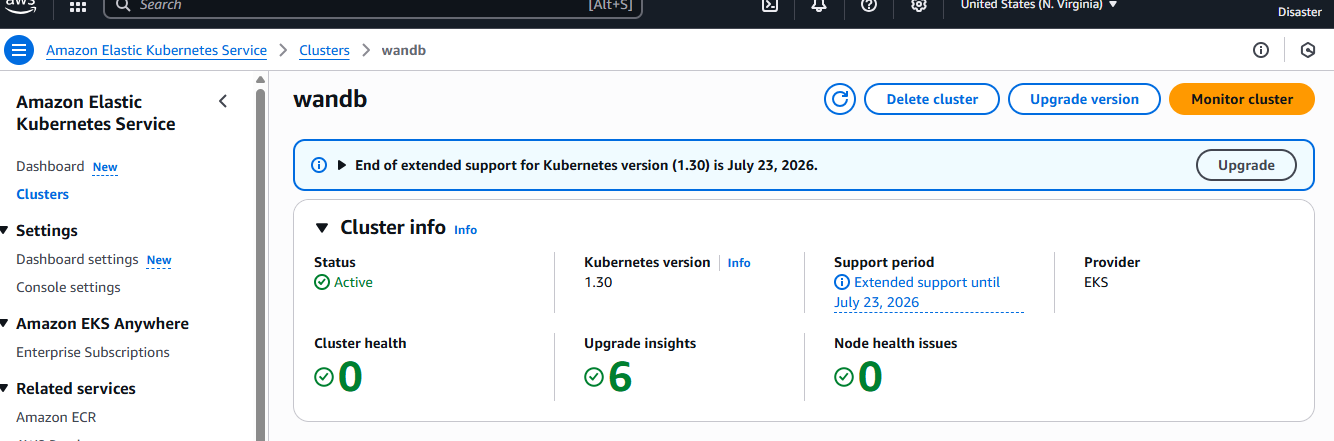
Creates a Managed Node Group with two nodes and adds to an autoscaling group.

Creates control Plane

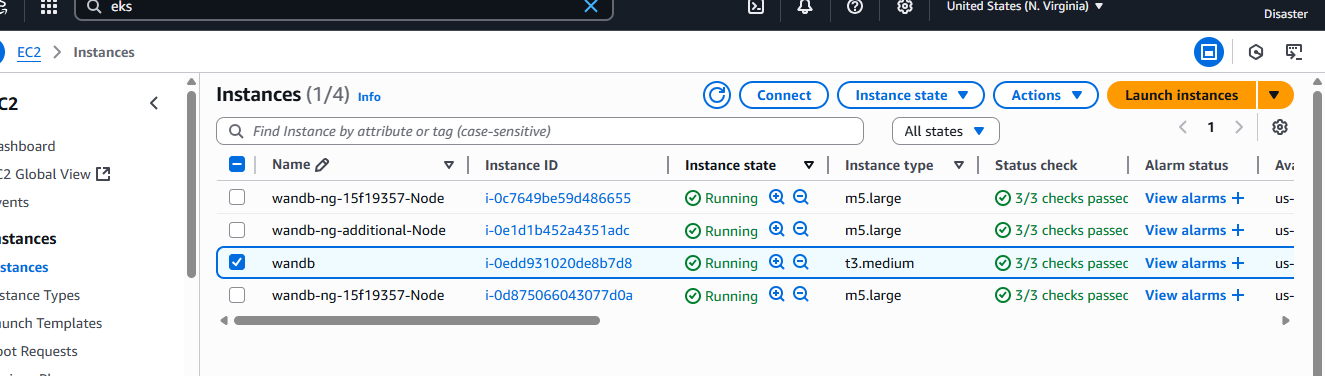
Install AWS Managed add-ons (VPC CNI, Core DNS, Kube-Proxy, Metrics Server).



EKS Cluster



Nodes



3.**Install add on - EBS CSI drivers**

It Supports the requirements of MySQL for Persistent Storage.

1. IAM OIDC Provider - The EBS CSI driver uses IAM roles for service accounts (IRSA), which requires an OIDC.

eksctl utils associate-iam-oidc-provider \

--region us-east-1 \

--cluster wandb \

--approve

1. Create a Service account and attach an IAM Policy.

eksctl create iamserviceaccount \

--cluster wandb \

--namespace kube-system \

--name ebs-csi-controller-sa \

--attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \

--approve

1. Install the EBS CSI Driver ( using helm chart)

helm repo add aws-ebs-csi-driver https://kubernetes-sigs.github.io/aws-ebs-csi-driver

helm repo update

helm install aws-ebs-csi-driver aws-ebs-csi-driver/aws-ebs-csi-driver \

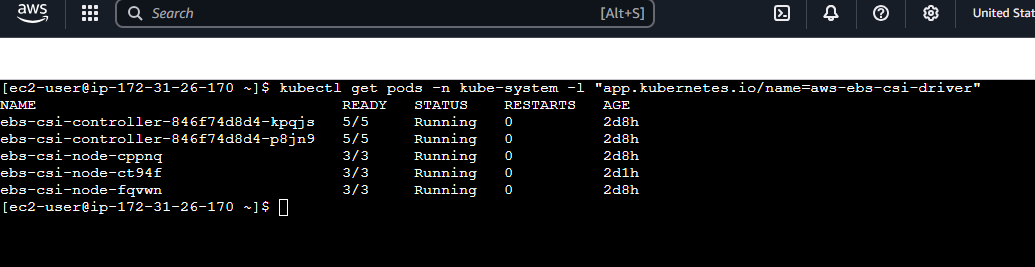
-namespace kube-system \

--set controller.serviceAccount.create=false \

--set controller.serviceAccount.name=ebs-csi-controller-sa

1. Verify the Installation

kubectl get pods -n kube-system -l "[app.kubernetes.io/name=aws-ebs-csi-driver](http://app.kubernetes.io/name=aws-ebs-csi-driver)"



.

4.**Defining the Storage Class to use in the Cluster.**

Created & deployed Storage Class and PVC manifest yaml for defining how storage will be provisioned and requested.

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ebs-sc

provisioner: ebs.csi.aws.com

parameters:

type: gp3

fsType: ext4

reclaimPolicy: Delete

volumeBindingMode: WaitForFirstConsumer

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: ebs-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 10Gi

storageClassName: ebs-sc

Now since all the infrastructure was set up.

5. **Generated the temporary License from Weight & Biases Website.**

6.. **Install Weight & Biases App using Helm Charts**

helm repo add wandb https://wandb.github.io/helm-charts/

helm repo update

created a Vaues.yaml to pass the values to templates.

license: XXXXXXXXXXXXXXXXXXXXXX

service:

type: LoadBalancer

ingress:

enabled: false

mysql:

persistence:

enabled: true

size: "10Gi"

storageClass: "ebs-sc"

resources:

limits:

cpu: 2

memory: 4Gi

requests:

cpu: 1

memory: 2Gi

**Note** : Since i don’t have a domain name so instead of Ingress used Service type : Load balancer to expose the Weight & Biases App and defined the storage class needed for database (MySql for Metadata) with Resource request and limits.

I could have used Amazon S3 or Minio for Object storage to store artifacts but kept it simple.

Install Weight & Biases using Helm

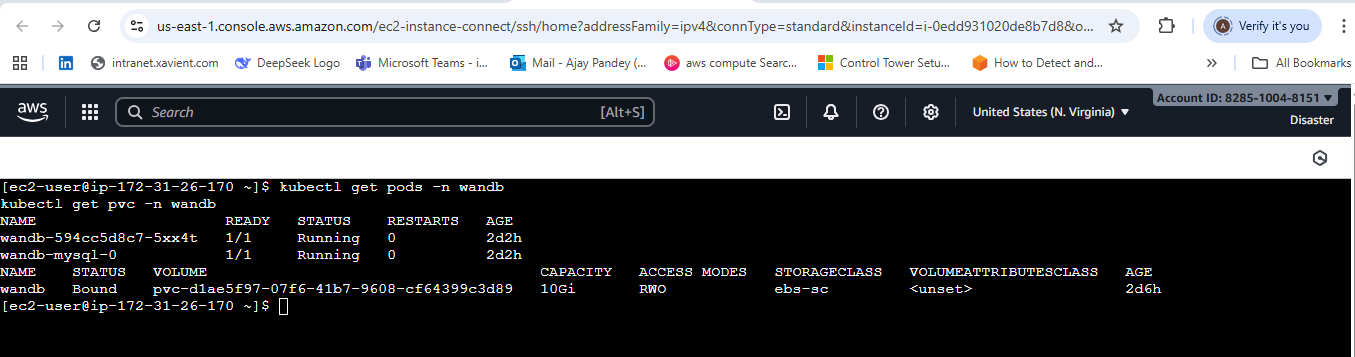
First it creates a namespace -wandb and deploy the app following the values in values.yaml

helm install wandb wandb/wandb \

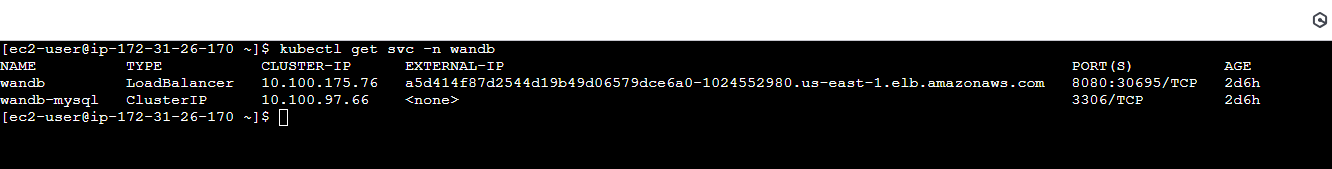
-f values.yaml \

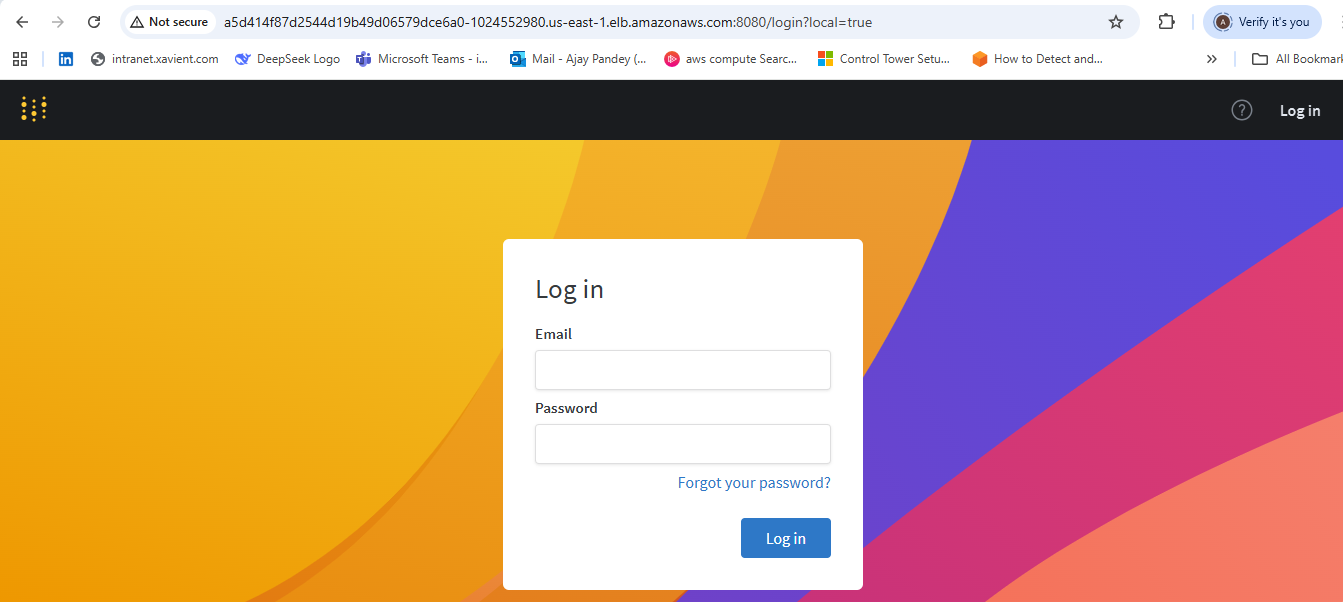
--namespace wandb --create-namespace

7. **Verify the Deployment.**

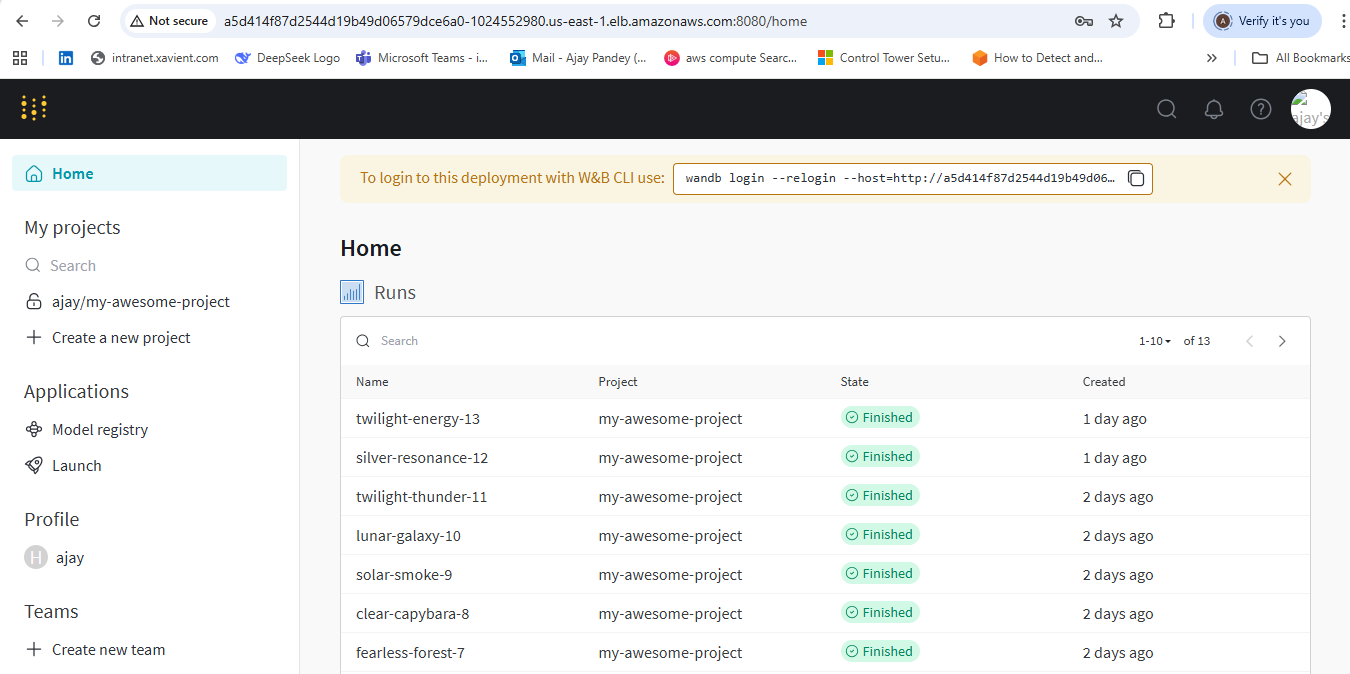


**8.Access the Web App using Load balancer url**



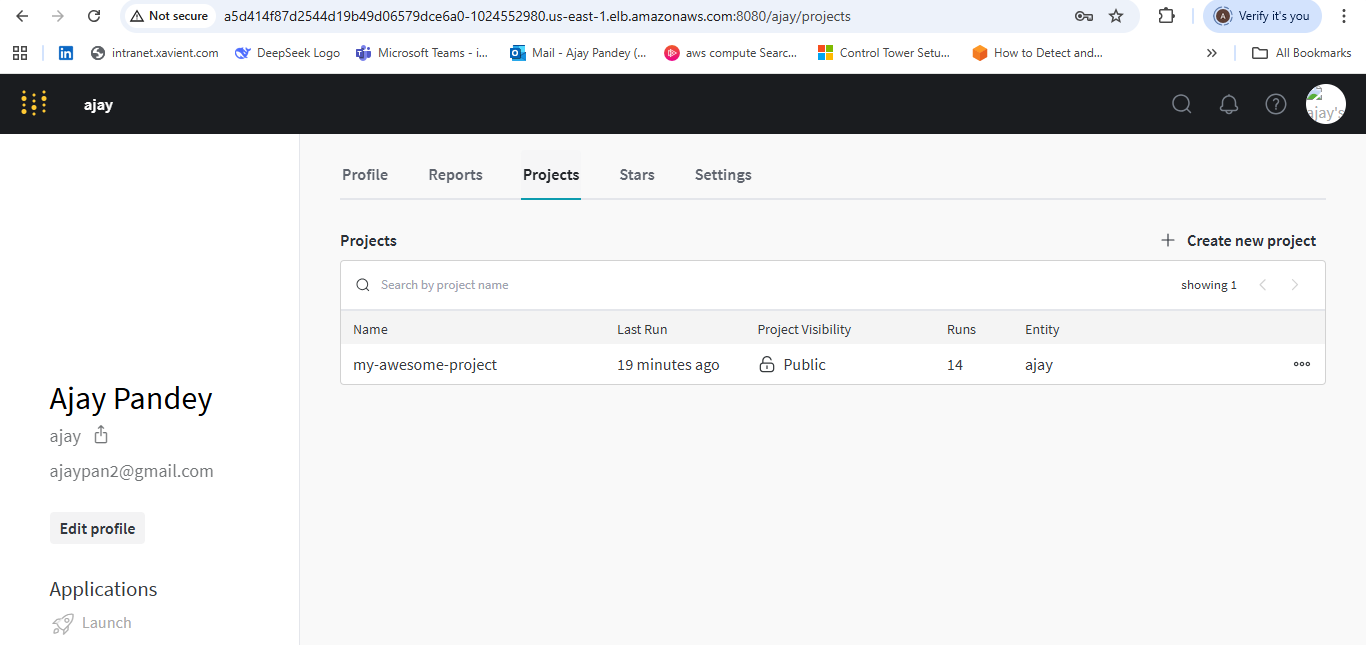


Created a user Id and password to log in.



9.**Next Task was to Create a Project with 10 runs.**

Created a Project in Weight & Biases Console by the name my-awesome-project.



Install Weight & Biases SDK on my local machine

pip install wandb

Created a Python Script to execute the runs from the command line from my local machine.

import wandb

import random

# start a new wandb run to track this script

wandb.init(

# set the wandb project where this run will be logged

project="my-awesome-project",

# track hyperparameters and run metadata

config={

"learning\_rate": 0.02,

"architecture": "CNN",

"dataset": "CIFAR-100",

"epochs": 10,

}

)

# simulate training

epochs = 10

offset = random.random() / 5

for epoch in range(2, epochs):

acc = 1 - 2 \*\* -epoch - random.random() / epoch - offset

loss = 2 \*\* -epoch + random.random() / epoch + offset

# log metrics to wandb

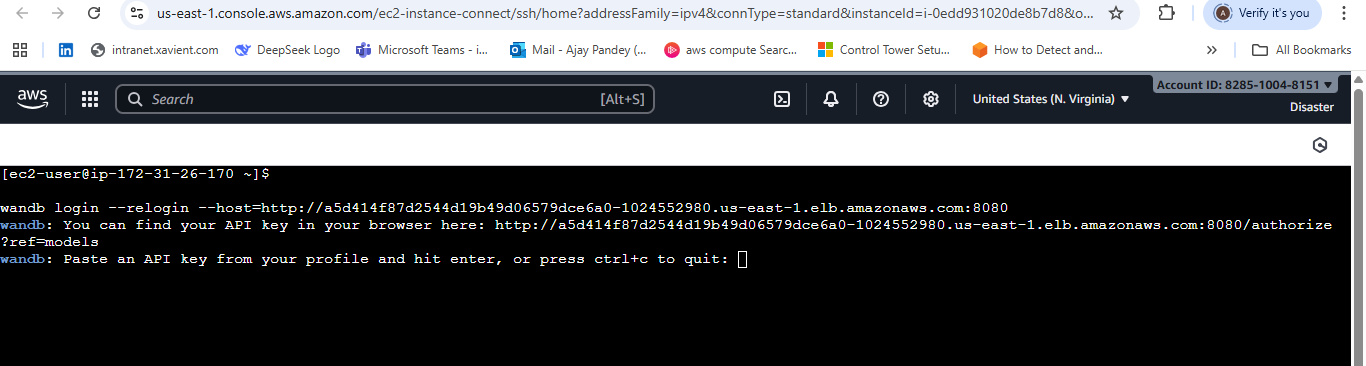
wandb.log({"acc": acc, "loss": loss})

# [optional] finish the wandb run, necessary in notebooks

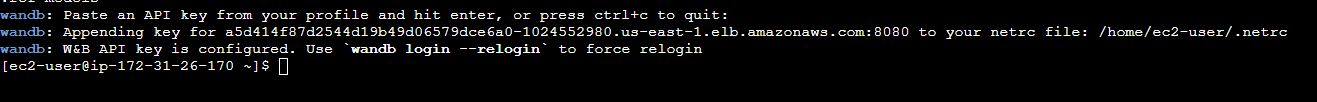
wandb.finish()

Command to connect to Weight & Biases Server from command line.

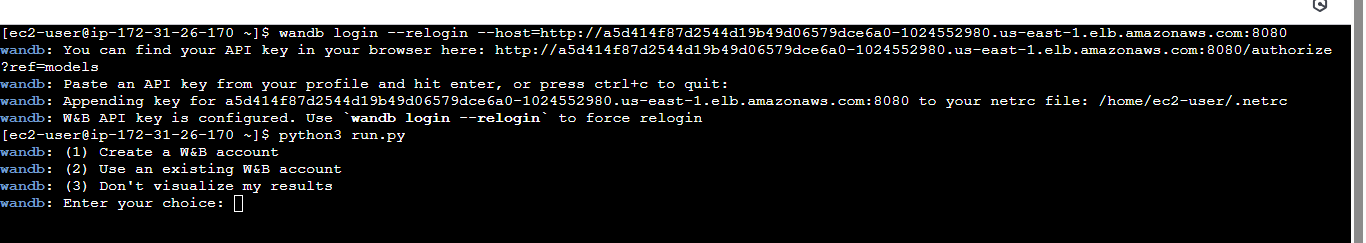
wandb login --relogin --host=http://a5d414f87d2544d19b49d06579dce6a0-1024552980.us-east-1.elb.amazonaws.com:8080

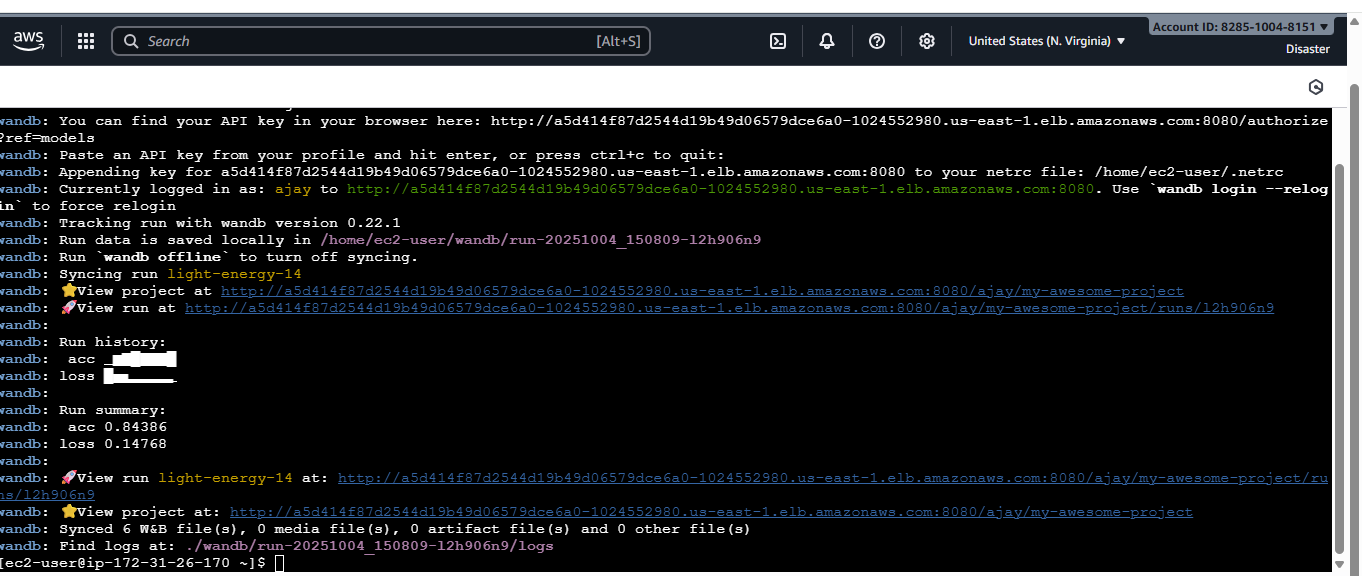


It asked for APiI key…Used the Url in the message to generate the API Key and Provided.

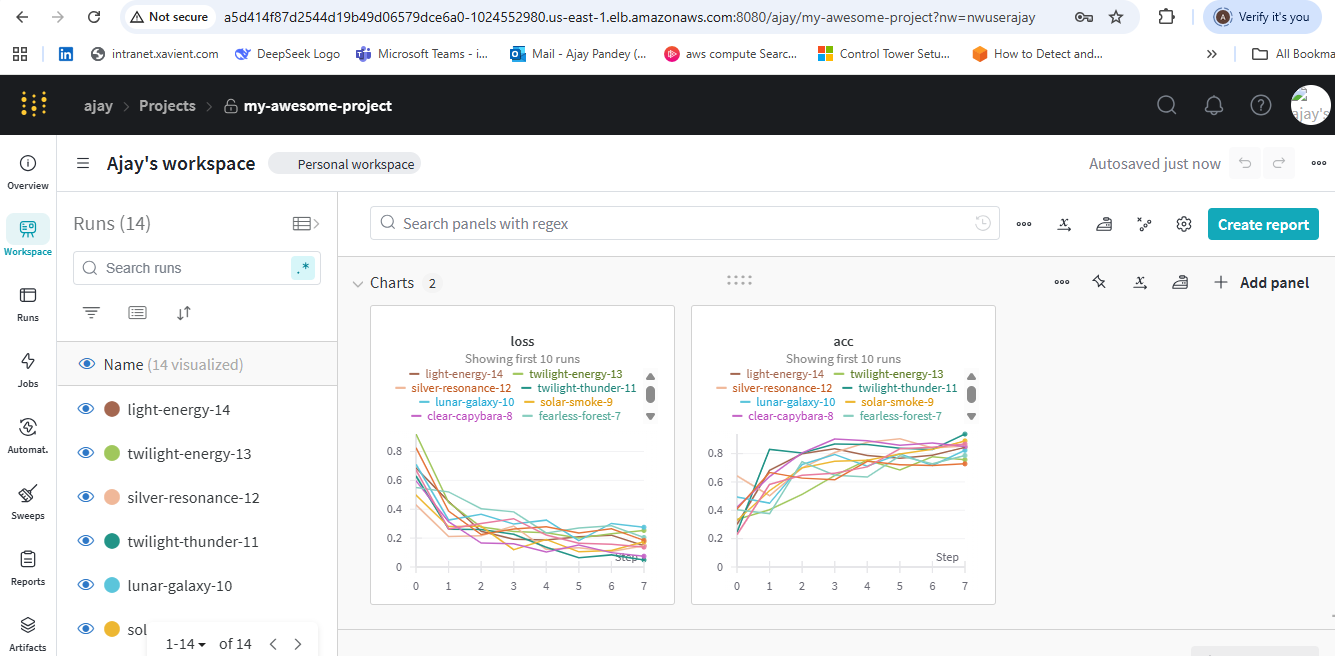


Once Key was configured, ran the Python Script to execute the runs and sync to Weight & Biases Console.





Shows the runs in the Weight & Baises console under my Project - my-awesome-project.



Note: I decided to take this approach to install a Weight & Baise server using helm charts becoz i wanted to explore and experiment. It would have been easy to do either with Terraform modules or Operators and get away quickly.

But using Helm charts was a great learning experience and do I ran into issues which I enjoyed troubleshooting and fixing. Please see the challenges file for the issues.

Next I will try the Operator method and will share my feedback.