**Job-Ready High-Level Project:**

**Mastering Three-Tier Architecture**

***with***

**EKS, CloudFormation, and Monitoring: Complete Deployment Guide**

**Project roadmap**

* Create an EC2 server
* Install and configure AWS CLI in the EC2 server
* Install eksctl and kubectl
* Create a cluster using eksctl
* Install Helm
* Install MySQL using Helm and mount EBS volume
* Configure the **MySQL** database
* Build the Two-tier Application v2 (with Database) Backend
* Push the Docker image in the Dockerhub
* Build the **Two-tier Application** v2 (with Database) Frontend
* Push the Docker image in the Dockerhub
* Create a deployment and a service for the **backend** in the K8s cluster
* Create a deployment and a service for the **frontend** in the K8s cluster
* View the application running
* Incorporate Prometheus and Grafana in the stack for logging and monitoring

**Create a Bastion Server:** Create ec2 named **bastion**, ubuntu AMI, security group named EKS (allow all traffic) Connect with mobaxterm

**sudo apt update && sudo apt -y full-upgrade**

Get AWS cli installation command from [*https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html*](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)

**sudo apt install unzip**

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

**unzip awscliv2.zip**

**sudo ./aws/install**

IAM >users **eks-user >** Attach policies directly > AdministratorAccess > create access key

**aws configure**

**aws s3 ls**

Create an EKS K8S Cluster

Get the eksctl installation commands from

[**https://eksctl.io/installation/**](https://eksctl.io/installation/)

# for ARM systems, set ARCH to: `arm64`, `armv6` or `armv7`

ARCH=amd64

PLATFORM=$(uname -s)\_$ARCH

curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$PLATFORM.tar.gz"

# (Optional) Verify checksum

curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_checksums.txt" | grep $PLATFORM | sha256sum --check

tar -xzf eksctl\_$PLATFORM.tar.gz -C /tmp && rm eksctl\_$PLATFORM.tar.gz

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

**eksctl version**

Get kubectl latest release and the installation command

[***https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/***](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)

Get the latest release

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

Install kubectl

**sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl**

**kubectl version**

Create an EKS Cluster

Create a cluster.yaml file to declare cluster configuration

**vim cluster.yaml**

**apiVersion: eksctl.io/v1alpha5**

**kind: ClusterConfig**

**metadata:**

**name: dec-24-eks**

**region: us-east-1**

**nodeGroups:**

**- name: ng-1**

**instanceType: t2.medium**

**desiredCapacity: 2**

**eksctl create cluster -f cluster.yaml**

**kubectl get nodes**  (see 4 nodes in the cluster)

Go to the CloudFormation dashboard in the AWS. You'll see two stacks have been created.

You can go to the Events tab to view the history of resources creation

**vim ~/.bashrc**

Add the alias command at the tail of the file.

**alias k='kubectl'**

**alias kgp='kubectl get pods'**

**alias kgs='kubectl get svc'**

Apply the changes

**source ~/.bashrc**

---

Install **MySQL** using **Helm** andmount **EBS volume**

Copy the Helm installation ubuntu commands from <https://helm.sh/docs/intro/install/>

Use this command:

**curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null sudo apt-get install apt-transport-https --yes echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list sudo apt-get update sudo apt-get install helm**

**Add Bitnami Chart**

Add bitnami charts into the list of Helm repo and update the repositories.

**helm repo add bitnami https://charts.bitnami.com/bitnami**

**helm repo list**

**helm repo update**

Managing the Amazon **EBS** CSI driver as an Amazon **EKS** add-on

Visit the following documentation URL to manage the Amazon EBS CSI driver as an Amazon EKS add-on <https://docs.aws.amazon.com/eks/latest/userguide/managing-ebs-csi.html>

Copy the command > Open the Creating an IAM OIDC provider in a new tab

Open the Creating the Amazon EBS CSI driver IAM role in a new tab

To see the required platform version, run the previously copied command.

**aws eks describe-addon-versions --addon-name aws-ebs-csi-driver**

Go to the **Creating an IAM OIDC provider**and run the three commands in the screenshot

[*https://docs.aws.amazon.com/eks/latest/userguide/enable-iam-roles-for-service-accounts.html*](https://docs.aws.amazon.com/eks/latest/userguide/enable-iam-roles-for-service-accounts.html)

**export cluster\_name=dec-24-eks**

**oidc\_id=$(aws eks describe-cluster --name dec-24-eks --query "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)**

**aws iam list-open-id-connect-providers | grep $oidc\_id | cut -d "/" -f4**

**echo $oidc\_id**

**eksctl utils associate-iam-oidc-provider --cluster dec-24-eks --approve**

Visit the following URL to get the command for Creating the Amazon EBS CSI driver IAM role

[**https://docs.aws.amazon.com/eks/latest/userguide/csi-iam-role.html**](https://docs.aws.amazon.com/eks/latest/userguide/csi-iam-role.html)

Run the copied commands. Remember to insert your own cluster name after the **--cluster**argument

**eksctl create iamserviceaccount \**

**--name ebs-csi-controller-sa \**

**--namespace kube-system \**

**--cluster dec-24-eks \**

**--role-name AmazonEKS\_EBS\_CSI\_DriverRole \**

**--role-only \**

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

**--approve**

Go to the IAM dashboard > Roles > new role **AmazonEKS\_EBS\_CSI\_DriverRole** appeared on top. Go inside that role. **Copy** **ARN**

Copy the command to create add-on from  [*https://docs.aws.amazon.com/eks/latest/userguide/managing-ebs-csi.html*](https://docs.aws.amazon.com/eks/latest/userguide/managing-ebs-csi.html)

Replace the arn with your own IAM Role ARN and run the command

**eksctl create addon --name aws-ebs-csi-driver --cluster dec-24-eks --service-account-role-arn arn:aws:iam::**<Your\_ARN>:**role/AmazonEKS\_EBS\_CSI\_DriverRole --force**

See that multiple pod appear in the kube-system namespace

**kubectl get pods -n kube-system**

In the CloudFormation, there should be a new stack of **ebs-csi-controller**

**vim storage.yaml**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: ebs-sc

annotations:

storageclass.kubernetes.io/is-default-class: "true"

provisioner: ebs.csi.aws.com

parameters:

type: gp2

reclaimPolicy: Retain

volumeBindingMode: WaitForFirstConsumer

**kubectl apply -f storage.yaml**

Prepare the MySQL database

**vim mysql-values.yaml**

*Write the following config*

**auth:**

**rootPassword: "hello123"**

**createDatabase: true**

**database: "twotier"**

**username: "twotieruser"**

**password: "secretpass"**

**primary:**

**persistence:**

**enabled: true**

**storageClass: "ebs-sc"**

**accessModes:**

**- ReadWriteOnce**

**size: 20Gi**

*Install MySQL in the mysql namespace and with the mysql-values.yaml configuration*

**helm install mysql oci://registry-1.docker.io/bitnamicharts/mysql -n mysql --create-namespace -f mysql-values.yaml**

See Mysql pod running

**kubectl get pods -n mysql**

kubectl get pvc -n mysql

kubectl get pv -n mysql

Get inside the mysql pod

**kubectl exec -n mysql -it mysql-0 -- bash**

Go to the mysql monitor where you can execute mysql queries.

**mysql -u root -p**

Password **hello123**

See that the **twotier**database has been created.  
Run the following commands to create the tables in the database

**show databases;**

**use twotier;**

CREATE TABLE users (id int NOT NULL AUTO\_INCREMENT, email varchar(100) NOT NULL, password varchar(100) NOT NULL, primary key(id));

CREATE TABLE history (id int NOT NULL AUTO\_INCREMENT, number varchar(100) NOT NULL, response varchar(100) NOT NULL, type varchar(30) NOT NULL, time timestamp default current\_timestamp, primary key(id));

**show tables;**

Create a user in the users table and view the table

**insert into users (email, password) values ("hello@example.com", "hello123");**

**select \* from users;**

Exit from MySQL and the pod.

**exit**

**exit**

**Create Two-tier Application v2 (with Database) Backend Docker Image**

--------------------------------------------------------------------------------------------------------------------

Clone the two tier application git repo

**git clone** [**https://github.com/azizulmaqsud/two-tier-app.git**](https://github.com/azizulmaqsud/two-tier-app.git)

**cd two-tier-application**

**git pull**

**ls**

**cd backend**

**vim .env**

**MYSQL\_HOST='mysql.mysql.svc.cluster.local'**

**MYSQL\_USER='twotieruser'**

**MYSQL\_PASSWORD='secretpass'**

**MYSQL\_DB='twotier'**

Install docker

**sudo apt install docker.io -y**

Login into docker with your Dockerhub username and password

**sudo docker login**

Write a Dockerfile for the backend

**vim Dockerfile**

Write the following config in the Dockerfile, then save it.

**# Use the official Python base image**

**FROM python:3.9-slim**

**# Set the working directory**

**WORKDIR /app**

**# Install dependencies and clean up to reduce image size**

**RUN apt-get update -y && \**

**apt-get install -y --no-install-recommends \**

**python3-pip python3-dev libmariadb-dev pkg-config && \**

**apt-get clean && \**

**rm -rf /var/lib/apt/lists/\***

**# Copy application files to the container**

**COPY . .**

**# Install Python dependencies**

**RUN pip install --no-cache-dir -r requirements.txt**

**# Expose the application port**

**EXPOSE 5000**

**# Set the default command to run the application**

**CMD ["flask", "run", "--host", "0.0.0.0"]**

Build the backend docker image

**sudo docker build -t azizul2go/n-tier-backend:v1 .**

Push the docker image into Dockerhub

**sudo docker push azizul2go/n-tier-backend:v1**

**Create Two-tier Application v2 (with Database) Frontend Docker Image**

Go to the frontend directory and write a Dockerfile

**cd ../frontend**

**vim Dockerfile**

Write the following config in the Dockerfile, then save it

**# Build Stage**

**FROM node:11.10.0-alpine AS build-stage**

**# Install dependencies**

**RUN apk add --no-cache \**

**python \**

**make \**

**g++ \**

**&& rm -rf /var/cache/apk/\***

**# Set working directory and copy necessary files**

**WORKDIR /src**

**COPY . .**

**COPY ./package\* ./**

**# Install Node.js dependencies and build the application**

**RUN npm install**

**RUN yarn build --frozen-lockfile**

**# Production Stage**

**FROM nginx:latest**

**# Clean existing HTML content and copy the built application**

**RUN rm -rf /usr/share/nginx/html**

**COPY --from=build-stage /src/build/ /usr/share/nginx/html/**

**# Copy custom Nginx configuration**

**COPY default.conf /etc/nginx/conf.d/**

**# Expose port 80 and set the default command**

**EXPOSE 80**

**CMD ["nginx", "-g", "daemon off;"]**

**vim default.conf**

Add the **/login**and **/get-history** endpoints.

Change the proxy pass into **http://backend-svc:5000** for all the endpoints.

**vim .env**

Keep the BACKEND\_API\_URL **blank**

Build the frontend docker image

**sudo docker build -t azizul2go/n-tier-frontend:v1 .**

Push the docker image into Dockerhub

**sudo docker push azizul2go/n-tier-frontend:v1**

**Create Backend Deployment**

Create twotier namespace

**kubectl create ns twotier**

**cd (**go to home dir)

Create a backend deployment file and view it

**kubectl create deployment backend --image azizul2go/n-tier-backend:v1 -o yaml --dry-run=client -n twotier --port 5000 > backend.yaml**

**cat backend.yaml**

Apply the deployment and see the running pods

**kubectl apply -f backend.yaml**

**kubectl get pods -n twotier**

Create a service for the deployment and copy the configurations

**kubectl expose deployment backend --name backend-svc --type NodePort --port 5000 --dry-run=client -n twotier -o yaml**

Copy the output configuration

**vim backend.yaml**

Put**separator**(**---**) and below that paste the copied configuration of the backend service.

Run the service and view the running pods and services

**kubectl apply -f backend.yaml**

**kubectl get pods -n twotier**

**kubectl get svc -n twotier**

**Create Frontend Deployment**

Create a frontend deployment file and run it

**kubectl create deployment frontend --image azizul2go/n-tier-frontend:v1 -o yaml --dry-run=client -n twotier --port 80 > frontend.yaml**

**kubectl apply -f frontend.yaml**

**kubectl get pods -n twotier**

Create a service for the deployment and copy the configurations

**kubectl expose deployment frontend --name frontend-svc --type NodePort --port 80 --dry-run=client -n twotier -o yaml**

Copy the output configuration

**vim frontend.yaml**

Put**separator**(**---**) and below that paste the copied configuration of the frontend service.

**kubectl apply -f frontend.yaml**

**kubectl get pods -n twotier**

**kubectl get svc -n twotier**

See that the frontend service is running at port NodePort in this case, it may be different for you. Take note of it as you will need the port number to run the frontend.

**Change Security Rules**

Select **one** of the cluster nodes. In the security tab, you will find two different security groups. We need to change the inbound rules of both of them. Open the security groups in new tabs.

All traffic rule for both.

**View the application running**

Hit one of the cluster node's IP Address and the frontend port **<Cluster\_Node\_IP>:<Frontend\_PORT>**

You should see that the **Two-tier Application v2** is running. The backend, frontend, and all the endpoints work.

**Incorporate Prometheus and Grafana in the stack for logging and monitoring**

------------------------------------------------------------------------------------------------------

Create a namespace for monitoring

**kubectl create ns monitoring**

Add Prometheus charts in the helm repo. You can get the command from here **https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack**

Run the command

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

**helm repo update**

**helm install my-kube-prometheus-stack prometheus-community/kube-prometheus-stack --version 62.6.0**

Go to

[***https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html***](https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html)

[*https://docs.aws.amazon.com/eks/latest/userguide/lbc-helm.html*](https://docs.aws.amazon.com/eks/latest/userguide/lbc-helm.html)

use this command:

**curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.7.2/docs/install/iam\_policy.json**

**aws iam create-policy \**

**--policy-name AWSLoadBalancerControllerIAMPolicy \**

**--policy-document** [**file://iam\_policy.json**](file://iam_policy.json)

**eksctl create iamserviceaccount \**

**--cluster=<name> \**

**--namespace=kube-system \**

**--name=aws-load-balancer-controller \**

**--role-name AmazonEKSLoadBalancerControllerRole \**

**--attach-policy-arn=arn:aws:iam::<111122223333>:policy/AWSLoadBalancerControllerIAMPolicy \**

**--approve**

**---**

**helm repo add eks** [**https://aws.github.io/eks-charts**](https://aws.github.io/eks-charts)

**helm repo update eks**

Next, we need to add the eks-charts repo. Go back to the **Installing the AWS Load Balancer Controller add-on**page and copy these commands

[***https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html***](https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html)

Add, update the repositories and view the repo list

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

**helm repo update**

**helm repo list**

**Install the ALB Ingress Controller:**

**helm install aws-load-balancer-controller eks/aws-load-balancer-controller \**

**--namespace kube-system \**

**--set clusterName=<**your-cluster-name**> \**

**--set serviceAccount.create=false \**

**--set region=<your-region> \**

**--set vpcId=<**your-vpc-id**> \**

**--set serviceAccount.name=aws-load-balancer-controller**

**Verify deployment:**

**kubectl get pods -n kube-system -l app.kubernetes.io/name=aws-load-balancer-controller**

Find this command and copy it. Replace my-cluster with the name of your cluster. Run the command. Now see that there are aws-load-balancer-controller pods running in the kube-system namespace

**kubectl get pods -n kube-system**

If **not** found then

**helm uninstall aws-load-balancer-controller -n kube-system**

helm install aws-load-balancer-controller eks/aws-load-balancer-controller \

--namespace kube-system \

--set clusterName=<name> \

--set serviceAccount.create=true \

--set region=us-east-1 \

--set vpcId=vpc-11111111111111

kubectl get pods -n kube-system

**vim prom-values.yaml** add following….

**grafana:**

**ingress:**

**enabled: true**

**ingressClassName: alb**

**annotations:**

**alb.ingress.kubernetes.io/scheme: internet-facing**

**alb.ingress.kubernetes.io/target-type: ip**

Release prometheus and grafana with the following command

**helm install prom-graf prometheus-community/kube-prometheus-stack -n monitoring -f prom-values.yaml**

See that multiple prom-graf pods have started to run in the monitoring namespace

**kubectl get pods -n monitoring**

Visit your AWS Console and you should see a Load Balancer created there

View the ingress running

**kubectl get ingress -n monitoring**

Copy the address of the running ingress. Visit ingress address

Change security group at ALB, A record, all traffic inbound

The Grafana login screen should pop up.   
Log in with the username **admin** and password **prom-operator**

Go to **dashboards,** Expand **General**

Go inside the **Cluster dashboard**

See the visualization of the entire cluster including all the namespaces, services, and the pods running. You can play around in the dashboard to view different data

**Thank You**