DEVOPS INTERVIEW TASK – END-TO-END AWS EKS DEPLOYMENT

Github repository: https://github.com/Cyril3012/hello-app.git

---

PROJECT GOAL AND IDEOLOGY

The goal of this task is to demonstrate an end-to-end DevOps workflow using AWS, Docker, Kubernetes (EKS), Terraform, Helm, and GitHub Actions.

The solution focuses on automated deployments, security, and cost optimization while deploying a simple application to Amazon EKS using Infrastructure as Code and container orchestration best practices.

This approach reflects real-world production environments where infrastructure provisioning, application deployment, access control, and CI/CD pipelines are fully automated, version-controlled, and reproducible.

SOLUTION OVERVIEW

• Infrastructure is provisioned using Terraform to ensure consistency, repeatability, and easy rollback
• The application is containerized using Docker and stored in a container registry (Docker Hub)
• Amazon EKS is used for container orchestration, self-healing, and service exposure
• Helm is used to manage Kubernetes manifests and application releases
• GitHub Actions is used for CI/CD automation
• AWS IAM and EKS access entries are configured for secure cluster access
• Resource limits and scaling are optimized to stay within AWS Free Tier constraints

---

FLOW OF TASK:

1. Aws account setup
2. Download dependency tools (Docker, Kubernetes, Docker Desktop, Terraform, Helm)
3. Configure AWS CLI
4. Sample node application
5. Containerize the application using Docker
6. Kubernetes deployment
7. IaC using Terraform
8. Deployment and troubleshooting
9. CI/CD Pipeline
10. Package Kubernetes using Helm
11. Conclusion

AWS ACCOUNT SETUP

• Create an AWS Free Tier account
• Enable MFA for the root user
• Configure a billing alarm to avoid unexpected charges
• Create an Admin IAM user with:
  – Programmatic access

– AWS Management Console access
• Attach AdministratorAccess policy to the IAM user

Purpose: Secure AWS account access and control costs.

DOWNLOAD AND INSTALL AWS CLI

msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi

```
C:\Users\DELL>aws --version
aws-cli/2.32.30 Python/3.13.11 Windows/11 exe/AMD64
```

aws configure
Enter:
AWS Access Key ID : <your-access-key>
AWS Secret Access Key : <your-secret-key>
Default region name : ap-south-1
Default output format : json

Test:
aws sts get-caller-identity

DOCKER IMAGE CREATION

OVERVIEW

Docker is used to package the application into a lightweight and secure container image.
A multi-stage build with a distroless image is used to reduce image size and improve security.

MULTI-STAGE BUILD

A multi-stage Docker build is used to separate the build process from the runtime environment.

• The first stage builds the application
• The final stage contains only the required runtime artifacts

This approach helps:
• Reduce final image size
• Exclude unnecessary build tools
• Improve deployment efficiency

DISTROLESS IMAGE

A distroless base image is used for the final container.

• Contains only application runtime dependencies
• No shell or package manager included
• Reduces attack surface and improves security

DOCKER COMMAND REFERENCE

docker build -t hello-app app
docker images
docker ps -a
docker stop hello-test
docker rm hello-test
docker run -d -p 3000:3000 --name hello-test hello-app
docker tag hello-app:latest chandrucyril/hello-app:latest
docker push chandrucyril/hello-app:latest

KUBERNETES APPLICATION DEPLOYMENT

OVERVIEW

Kubernetes is used to deploy and manage the containerized application on the EKS cluster.
The deployment follows a simple and controlled approach suitable for Free Tier and demo environments.

DEPLOYMENT COMPONENTS

• Secret
• Service – NodePort
• Deployment

These components handle configuration, service exposure, and application lifecycle management.

DEPLOYMENT EXECUTION (COMMAND REFERENCE)

kubectl apply -f kubernetes/secret.yaml
kubectl apply -f kubernetes/service.yaml
kubectl apply -f kubernetes/deployment.yaml

VERIFICATION AND MONITORING

kubectl config current-context
kubectl cluster-info
kubectl get pods
kubectl describe pod hello-app-85675d7844-dsljq

APPLICATION UPDATE

Restart the application deployment to apply changes:

kubectl rollout restart deployment hello-app

TERRAFORM INFRASTRUCTURE PROVISIONING

OVERVIEW

Terraform is used to provision the AWS infrastructure in a consistent and repeatable manner.
It manages the EKS cluster and related resources using Infrastructure as Code.

TERRAFORM EXECUTION FLOW (COMMAND REFERENCE)

terraform init
terraform validate
terraform plan
terraform apply

ERROR HANDLING AND RE-APPLY

If an error occurs during apply:
• Fix the configuration
• Remove partially created resources
• Reinitialize and reapply

terraform destroy
terraform init -reconfigure
terraform apply

STATE AND OUTPUT VERIFICATION

terraform state list
terraform output

CONNECT TO EKS CLUSTER

Configure kubectl access to the EKS cluster:

aws eks update-kubeconfig --region ap-south-1 --name hello-eks

---

DEPLOYMENT ACCESS, OPTIMIZATION, AND TROUBLESHOOTING

EKS ACCESS CONFIGURATION (AWS CONSOLE)

To allow secure access to the EKS cluster, IAM-based access is configured using the AWS Console.

1. Open AWS Console → Amazon EKS
2. Select cluster: hello-eks
3. Navigate to the Access tab
4. Click Create access entry

Configuration:
• IAM principal: IAM User ARN
• Type: Standard

Access Policy:
• Add policy: **AmazonEKSClusterAdminPolicy**
• Scope: **Cluster**

Save the configuration.

RESOURCE OPTIMIZATION AND COST CONTROL

To stay within AWS Free Tier and reduce unnecessary resource usage, the following optimizations are applied.

Reduce CoreDNS replicas:

kubectl scale deployment coredns -n kube-system --replicas=1

This limits system pod usage while maintaining cluster functionality.

Additional optimizations:
• Reduce EKS worker node count using Terraform
• Minimize IAM policies to least privilege
• Restrict worker node security group rules to required ports only

---

DEPLOYMENT MANAGEMENT (CI/CD FRIENDLY)

Due to Free Tier resource limits:

• Pod limit maintained at 5–6 pods
• Rolling updates are avoided
• Recreate deployment strategy is preferred for CI/CD

Reason:
• Prevents pod over-allocation
• Ensures predictable deployments
• Avoids node resource exhaustion

APPLICATION DEPLOYMENT VERIFICATION

kubectl get pods
kubectl rollout restart deployment hello-app
kubectl get svc
kubectl get nodes -o wide
kubectl get deployment hello-app

SERVICE AND APPLICATION TROUBLESHOOTING

Describe service to verify exposure:

kubectl describe svc hello-service

Common checks:
• Service selector matches pod labels
• Correct port mapping
• Pod is in Running state

SELF-HEALING VALIDATION

Delete a pod manually to validate Kubernetes self-healing:

kubectl delete pod hello-app-54fc5ff74f-865lk

Expected behaviour:
• Kubernetes automatically creates a new pod
• Application remains available

TROUBLESHOOTING SUMMARY

This section helps to:
• Validate cluster access
• Control resource usage
• Monitor deployments
• Debug application issues
• Demonstrate Kubernetes self-healing

VERIFY APPLICATION USING NODEPORT

OVERVIEW

NodePort is used to verify application accessibility by exposing the service on a static port of each worker node.
This method is simple and suitable for testing and demo environments.

VERIFY SERVICE TYPE

Check the service configuration:

kubectl get svc

Confirm that the service type is **NodePort** and note the assigned port.

GET NODE DETAILS

Retrieve worker node IP addresses:

kubectl get nodes -o wide

Use the **EXTERNAL-IP** or **INTERNAL-IP** of any worker node.

ACCESS THE APPLICATION

Access the application using a browser or curl:

http://<NODE-IP>:<NODEPORT>

Example:
• Node IP: Worker node external IP
• Port: Assigned NodePort from service

TROUBLESHOOTING CHECKS

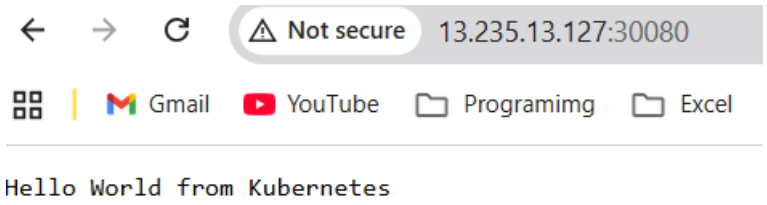
If the application is not accessible:

• Verify pod status:
kubectl get pods

• Verify service mapping:
kubectl describe svc hello-service

• Ensure node security group allows NodePort range (30000–32767)
• Confirm application container is listening on the correct port

Vefify Security group of woker node is open to specifiect TCP port



GITHUB ACTIONS CI/CD PIPELINE

OVERVIEW

GitHub Actions is used to automate the build, containerization, and deployment process.
The pipeline ensures consistent, repeatable, and secure application deployments to the EKS cluster.

CI/CD FLOW (HIGH LEVEL)

1. Developer pushes code to the GitHub repository
2. GitHub Actions workflow is triggered automatically
3. Application is built and tested
4. Docker image is built using a multi-stage and distroless approach

5. Docker image is pushed to Docker Hub
6. Kubernetes deployment is updated
7. Application pods are restarted to apply the new version

GITHUB ACTIONS STEPS

1. Checkout source code
2. Set up Docker build environment
3. Build Docker image
4. Authenticate to Docker Hub
5. Push image to container registry
6. Update Kubernetes deployment
7. Restart application deployment

DEPLOYMENT STRATEGY

• Recreate strategy is used instead of rolling updates
• Ensures predictable deployments under limited node and pod capacity
• Prevents resource exhaustion in Free Tier environments

SECRET MANAGEMENT IN GITHUB ACTIONS

Secrets required for CI/CD are stored securely in GitHub and never exposed in code.

Secrets Used:
• Docker Hub username
• Docker Hub access token
• AWS access key and secret key
• EKS cluster configuration values

How Secrets Are Managed:
• Stored in GitHub Repository Secrets
• Accessed only at runtime during workflow execution
• Not logged or printed in pipeline output

KUBERNETES SECRET HANDLING

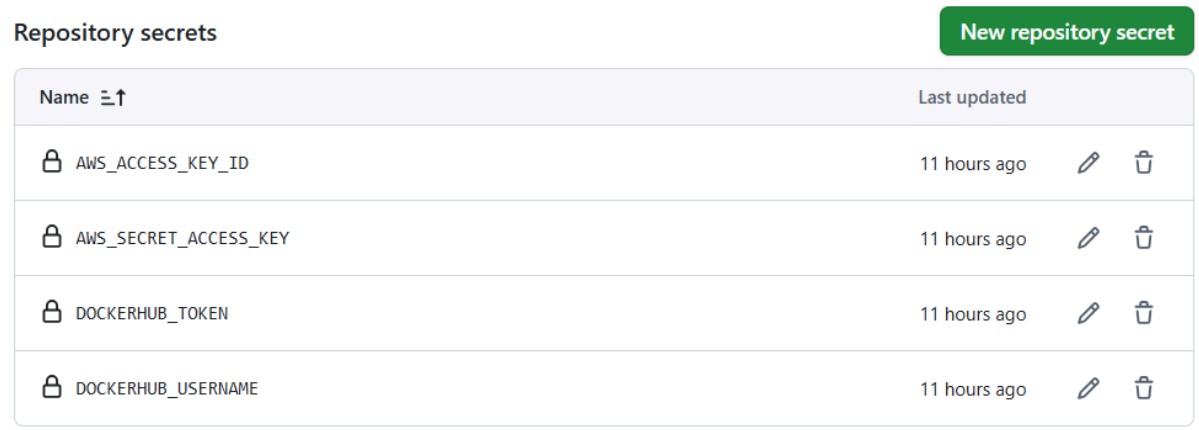
• Application secrets are managed using Kubernetes Secrets
• Secrets are referenced in deployment manifests or Helm charts
• No sensitive values are committed to GitHub

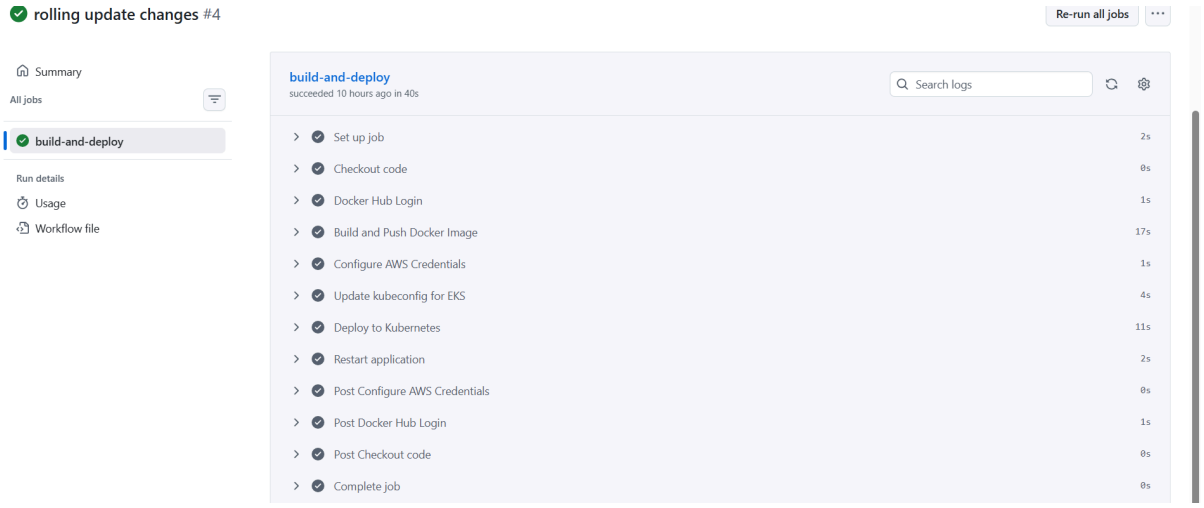
SECURITY BEST PRACTICES

• Secrets are encrypted at rest in GitHub
• Least-privilege IAM permissions are applied
• Separate secrets for CI/CD and runtime
• Credentials are rotated periodically

PURPOSE

This CI/CD setup ensures:
• Automated and reliable deployments
• Secure handling of credentials
• Faster release cycles
• Reduced manual intervention





HELM DEPLOYMENT (PACKAGE & RELEASE MANAGEMENT)

Helm is used to **package, template, version, and manage releases** of the application.
The underlying Kubernetes architecture remains unchanged.

HELM CHART STRUCTURE

helm/

└── hello-app/

├── Chart.yaml

├── values.yaml

└── templates/

├── deployment.yaml

├── service.yaml

└── secret.yaml

WHY HELM

- Single deployable unit for multiple Kubernetes manifests
- Configuration driven via values.yaml
- Built-in support for:
    - Versioning
    - Rollbacks
    - Safe upgrades
- CI/CD friendly and production standard

DEPLOY USING HELM

helm install hello-app ./helm/hello-app

helm upgrade hello-app ./helm/hello-app

helm list

kubectl get pods

kubectl get svc

HELM VERSIONING

- Each Helm upgrade creates a **new release revision**
- View release history:

helm history hello-app

HELM ROLLBACK

helm rollback hello-app 1

kubectl rollout status deployment hello-app

```
D:\Projects\Interview tasks\WebApp\hello-app>helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           A
PP VERSION
hello-app       default         1               2026-01-11 10:12:35.9794039 +0530 IST   deployed        hello-app-0.1.01
.0.0

D:\Projects\Interview tasks\WebApp\hello-app>kubectl get all
NAME                             READY   STATUS    RESTARTS   AGE
pod/hello-app-5cd85f457-vxhlp   1/1     Running   0          7m32s

NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/hello-app-service   NodePort    10.106.247.53   <none>        80:30080/TCP   7m32s
service/kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP        2d12h

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-app   1/1     1            1           7m33s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-app-5cd85f457   1         1         1       7m33s

D:\Projects\Interview tasks\WebApp\hello-app>kubectl get secret
NAME                                TYPE                 DATA   AGE
hello-app-secret                    Opaque               2      9m29s
sh.helm.release.v1.hello-app.v1   helm.sh/release.v1   1      9m29s
```

REFERENCE:

KUBERNETES (OFFICIAL DOCUMENTATION)

Kubernetes Documentation: https://kubernetes.io/docs/

Kubectl Installation: https://kubernetes.io/docs/tasks/tools/

Kubernetes Concepts: https://kubernetes.io/docs/concepts/

AWS EKS DOCUMENTATION

Amazon EKS User Guide :https://docs.aws.amazon.com/eks/latest/userguide/

CLI installation : https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

---

TERRAFORM (OFFICIAL DOCUMENTATION)

Terraform Documentation: https://developer.hashicorp.com/terraform/docs

Terraform Installation: https://developer.hashicorp.com/terraform/install

Terraform AWS Provider: https://registry.terraform.io/providers/hashicorp/aws/latest/docs

---

HELM (OFFICIAL DOCUMENTATION)

Helm Documentation: https://helm.sh/docs/

Helm Installation Guide: https://helm.sh/docs/intro/install/

Helm Charts: https://helm.sh/docs/topics/charts/
