**Interview Preparation**

1. **Could you briefly describe your past experience, your areas of expertise in DevOps, and what your daily tasks involve?**

* Good Evening!.
* Myself, Shivam Thakur. I have completed my MCA from LPU in 2024 and I live in Airoli, Navi Mumbai.I have around 4.5 years of experience as DevOps engineer including around 2 years of experience in Java at Infosys. Actually, I started my career as a Java Developer, but I was involved in DevOps tasks as well in my first project.
* There I had mainly worked on developing and maintaining microservices using Java and Spring Boot, and built CICD pipeline for microservices using Jenkins &Spinnaker.
* Handled real-time data using Kafka.
* For databases, depending on the need, we used Cassandra, PostgreSQL, and OpenSearch to store data through microservices.
* Later, I moved into another project as a full time DevOps Engineer.
* Right now, I mostly working with Kubernetes, Docker, Terraform, Ansible, Jenkins, AWS.
* My daily work includes building and managing CI/CD pipelines with Jenkins, and deployment of applications into EKS. I’ve also written a Docker / Docker Compose file.
* For monitoring, used Prometheus & Grafana for EKS, and use AWS CloudWatch for AWS instance and Nagios for onpremise server
* Created infrastructure using Terraform and configured the servers using Ansible playbooks
* And created some Shell scripts and Python scripts.
* Overall, I have experience on development, automation, cloud infrastructure management, and CI/CD.
* I’m now excited to take on a new opportunity where I can learn even more, solve bigger challenges, and contribute to a great team.

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

**Jenkins**

1. **How did you set up CI/CD pipelines?**

* We have a full CICD setup using Jenkins, SonarQube, Docker, ECR, EKS & Helm.
* Jenkins had a pipeline configured that would trigger automatically using webhooks whenever there was a code push.
* First, Jenkins pulls the latest code from the repository and runs a SonarQube scan to do static code analysis and check for any code quality issues.
* If the SonarQube quality gate passes, Jenkins moves to the next stage where it builds the Java application using Maven.
* After the build, Jenkins creates a Docker image for the application.
* Trivy scan to scname docker image for vulnability
* Then the Docker image is pushed to AWS ECR (Elastic Container Registry)
* Once the image is available in ECR, Jenkins uses Helm to deploy the application on EKS.

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

1. **Where do you store credentials in Jenkins, and how do you use them?**

* Storing Credentials:

We store sensitive data like API keys, passwords, and SSH keys in Jenkins using the Credentials plugin

We define credentials under Jenkins > Manage Jenkins > Manage Credentials.

Credentials can be stored globally or scoped to specific folders or pipelines.

* Types of Credentials:

Username and Password: For services like databases or third-party APIs.

Secret Text: For API tokens or other secrets.

SSH Keys: For Git repositories or other services requiring SSH access.

* Using Credentials:

We use the stored credentials in pipelines by referencing them with the credentials parameter in Jenkinsfiles or build steps.

Example in a Jenkinsfile:

pipeline {

agent any

environment {

MY\_SECRET = credentials('my-secret-id') // Fetch stored credentials

}

stages {

stage('Deploy') {

steps {

sh '''

echo "Using the secret: $MY\_SECRET"

# Use the secret in a deployment script

'''

}

}

}

}

* Access Control:

Only authorized users and jobs can access the stored credentials, ensuring security.

It's also important to avoid exposing credentials in logs or UI, so we use Jenkins' built-in mechanisms to mask sensitive data.

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

1. **How do you use or access the variables in Jenkins?**

* Environment Variables: Jenkins automatically sets some environment variables, like BUILD\_ID, BUID\_NUMBER, JOB\_NAME, etc. We can use them in your pipeline like this:

echo "Job Name: ${env.JOB\_NAME}"

* Global Variables: We can define global variables in Jenkins settings or in our pipeline script, and use them throughout our job. For example:

def myVar = 'Hello, Jenkins!'

echo "${myVar}"

* Pipeline Parameters: If we want to pass variables when triggering a job, we can use parameters. These can be defined in the job configuration and accessed like this:

echo "Param Value: ${params.MY\_PARAM}"

* Secrets and Credentials: We can access sensitive data (like passwords) securely using Jenkins' credentials plugin:

withCredentials([usernamePassword(credentialsId: 'my-credentials', usernameVariable: 'USER', passwordVariable: 'PASS')]) {

echo "User: ${USER}, Password: ${PASS}"

}

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

1. **How to define and access parameters?**

* Defining Parameters

Go to the Jenkins job configuration page

Under the Build Triggers section, check the "This project is parameterized" option >

Click Add Parameter and choose the type of parameter (e.g., String, Boolean, Choice).

For example, a String Parameter named MY\_PARAM can be added.

* Accessing Parameters in the Pipeline

Once defined, you can access the parameters within your pipeline script using the params object.

echo "The value of MY\_PARAM is: ${params.MY\_PARAM}"

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

1. **How do you create a user in Jenkins and give them specific access?**

* Install and Configure the "Matrix Authorization Strategy" (if not already done)

Go to Manage Jenkins > Security.

Under Authorization, select Matrix-based security.

This allows you to control user permissions more finely.

* Create a New User

Go to Manage Jenkins > Users

Click Create User and fill in the required details (username, password, email)

Click Save to create the user.

* Assign Specific Access Using Permissions

Once the user is created, go to Manage Jenkins > Manage and Assign roles

Under Matrix Authorization Strategy, you'll see a list of all users and groups.

Add the new user by entering their username.

Assign permissions by checking the appropriate boxes for each type of access.

For example:

Job: Allow user to view, configure, or build jobs.

Administer: Grant full administrative rights (be careful with this one!).

Overall: Permissions like managing Jenkins or managing users.

* Save the Changes

After assigning the correct permissions, click Save to apply the changes.

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

1. **How do you send an email if a pipeline fails? Do you use any plugin?**

* Install the Email Extension Plugin from the Jenkins plugin manage.
* In the post block of the Jenkinsfile, I can use emailext function to send an email depending on the status.
* In the Pipeline Script, add a post-build action using the emailext function to send an email if the pipeline fails.

post {

success {

emailext(

subject: "Pipeline Success",

body: "Your pipeline has completed successfully.",

to: "your-email@example.com"

)

}

failure {

emailext(

subject: "Pipeline Failed",

body: "Oops! Something went wrong with the pipeline.",

to: "your-email@example.com"

)

}

}

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

1. **How do you run stages in parallel during a pipeline execution?**

* Parallel stages allow us to run multiple tasks at the same time in a pipeline instead of one after another.
* Example: In my pipeline, I used parallel stages to run unit tests and static code analysis together.

stages {

stage('Parallel Tasks') {

parallel {

stage('Unit Test') {

steps {

echo 'Running Unit Tests'

}

}

stage('Code Analysis') {

steps {

echo 'Running Static Code Analysis'

}

}

}

}

}

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

1. **How do you update Docker images in Jenkins?**

* In your Jenkins pipeline, we can use the BUILD\_NUMBER variable to tag your Docker image.

This helps to keep track of the version of the image built for each Jenkins run.

* Example:

pipeline {

agent any

stages {

stage('Build Docker Image') {

steps {

script {

// Build Docker image

def imageName = "your-repo/your-image:${BUILD\_NUMBER}"

sh "docker build -t ${imageName} ."

}

}

}

}

}

**--------------------------------------------------------------------------------------------------------------------------------------------**

1. **Where do you store Docker images?**

* For AWS users, ECR is a managed registry service for storing Docker images

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

1. **How many master and slave Jenkins have in organization project?**

In our setup, we have 1 master and 5 agents.

2 agents for Linux builds,

1 for Windows builds,

1 for Docker-related tasks,

and 1 for heavy performance tests.

**Kubernetes:**

**AWS**

1. **Can you explain IAM user, group, role, and policy in AWS?**

* IAM (Identity and Access Management) helps us to control who can access what in our AWS account.

For example, if I want a user to only access S3 but not EC2, I can create an IAM user and give only S3 permissions. It also supports roles, policies, and groups so we can manage access in a secure and organized way.

* An IAM user is an individual entity like a person or service, who needs to access AWS resources. Each IAM user gets a unique name and security credentials (like passwords or access keys) to access AWS.

Example:

If I’m working in a DevOps team and need to access EC2 or S3, I’ll have an IAM user with a username and password (or access keys) to log in and perform actions.

* An IAM group is a collection of IAM users. Instead of assigning permissions to each user individually, we can assign them to a group, and all users in that group inherit the same permissions.

Example: We can have a group called "DevOpsTeam" that has access to EC2, S3, and CloudWatch. Anyone added to this group automatically gets those permissions.

* A role is like a temporary identity with permissions. Instead of assigning permissions directly to a user or service, we assign them to a role.

Example:

If an EC2 instance needs to access an S3 bucket, we create a role with S3 access and attach it to the EC2 instance. So no need to store any credentials, AWS handles that securely.

* A policy is a JSON file that defines what actions are allowed or denied on which AWS resources. It’s attached to users, groups, or roles.

Example:

If I want a user to only read from a specific S3 bucket, I’ll write a policy that says:"Allow s3:GetObject on that bucket" and attach it to the user or role.

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

1. **What is VPC?**

* A VPC(Virtual Private Cloud) is like a private network inside AWS where we can launch our resources like EC2 instances, databases, and more. It allows us to create an isolated environment where we can control things like IP addresses, subnets, routing, and security settings.
* We can configure it to suit our needs, such as creating public and private subnets, setting up security groups and network ACLs for better control over traffic, and connecting to on-premises networks via a VPN or Direct Connect.
* Let say, if we’re hosting a web application. We can create a VPC to set up EC2 instances and a database. We can define a public subnet for web servers (load balancers) and a private subnet for databases (RDS) which can only accessible from the web server.
* In short, VPC helps us organize and secure our cloud infrastructure by controlling how resources talk to each other and to the internet.

1. **What is NAT (Network Address Translation)?**

* NAT (Network Address Translation) is a method used to map private IP addresses to a public IP address so that devices in a private network can access the internet and vice versa.
* Example:
* Let’s say we have an EC2 instance in a private subnet in our VPC. This instance can’t access the internet directly because it doesn’t have a public IP.
* To fix this, we need to set up a NAT Gateway in the public subnet.
* When the EC2 instance wants to access the internet, it sends requests to the NAT Gateway.
* The NAT Gateway forwards the requests to the internet using its own public IP
* When the response comes back, the NAT Gateway sends it back to the EC2 instance.

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

1. **Difference between a security group and a network ACL.**

* Security Group is like a firewall for EC2 instances, it controls inbound and outbound traffic at the instance level. It’s stateful, which means if we allow inbound traffic, the outbound reply is automatically allowed.
* Network ACL (NACL) is like a firewall for the whole subnet, it controls traffic at the subnet level. It's stateless, so we have to allow both inbound and outbound rules separately.

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

1. **What is ELK (Elasticsearch, Logstash, Kibana)?**

* ELK stands for Elasticsearch, Logstash, and Kibana. It’s basically a stack used for log management and monitoring.
  + Logstash collects and processes the logs from different sources ((like servers or apps)).
  + Elasticsearch stores and indexes those logs, so we can search them quickly.
  + Kibana is the UI, where we can visualize the logs, create dashboards, and monitor issues easily

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

1. **For what purpose do you use AWS CloudWatch?**

* I use AWS CloudWatch for monitoring and alerting purposes. It helps us to track performance metrics, collect logs, and set alarms for all AWS resources like EC2, S3, and Lambda.
* For example, if CPU usage of an EC2 instance goes above 80%, CloudWatch can trigger an alarm and send a notification to Slack or email. That way, we can take action before the server crashes.

Also, we use CloudWatch Logs to check application logs, like if something went wrong in a Lambda function or ECS container.

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

1. **How do you implement an RDS database in AWS? Where would you set up a cluster or instance?**

* In the development environment, a single instance is sufficient for testing and small workloads:
* I'd define the RDS instance in Terraform using the aws\_db\_instance resource.

resource "aws\_db\_instance" "dev\_db" {

allocated\_storage = 20 # Storage in GB

engine = "mysql" # Database engine (e.g., MySQL, PostgreSQL)

instance\_class = "db.t3.micro" # Instance type (e.g., t3.micro for dev)

name = "devdb" # DB name

username = "admin" # DB username

password = "yourpassword" # DB password

db\_subnet\_group\_name = aws\_db\_subnet\_group.dev\_subnet\_group.name # Subnet for DB

multi\_az = false # Disable Multi-AZ for Dev (not needed for low availability)

storage\_type = "gp2" # General Purpose SSD

publicly\_accessible = true # Set to true if you need to access from outside VPC

skip\_final\_snapshot = true # Skip snapshot on DB deletion (for dev, not critical)

}

* In a production environment, high availability and scalability are crucial. An RDS Aurora Cluster provides this by supporting multiple instances (read replicas) and automatic failover.

I’d use the aws\_rds\_cluster resource for creating the cluster and aws\_rds\_cluster\_instance for individual instances in the cluster

resource "aws\_rds\_cluster" "prod\_cluster" {

cluster\_identifier = "prod-cluster"

engine = "aurora" # Aurora database engine

master\_password = "yourpassword"

master\_username = "admin"

database\_name = "proddb" # DB name

skip\_final\_snapshot = true # Don't take a snapshot on deletion (optional)

storage\_encrypted = true # Enable encryption for security

backup\_retention\_period = 7 # Retain backups for 7 days

}

resource "aws\_rds\_cluster\_instance" "prod\_instance" {

cluster\_identifier = aws\_rds\_cluster.prod\_cluster.id

instance\_class = "db.r5.large" # Instance type (db.r5.large for prod workloads)

engine = "aurora" # Must match the engine of the cluster

publicly\_accessible = false # Set to false for internal access only

db\_subnet\_group\_name = aws\_db\_subnet\_group.prod\_subnet\_group.name

}

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

1. **How can you automatically shut down & start an EC2 instance on weekends?**

* We can AWS Lambda in combination with CloudWatch Events.
* Write a simple Lambda function in Python using boto3 to stop the EC2 instance:

import boto3

def lambda\_handler(event, context):

ec2 = boto3.client('ec2')

ec2.stop\_instances(InstanceIds=['i-xxxxxxxxxxxxxxxxx'])

return 'EC2 Instance Stopped'

* Create a CloudWatch Event rule to trigger the Lambda function at our desired time:

Go to CloudWatch > Events and create a new rule.

Choose Schedule, and use a cron expression or rate expression to specify when to trigger the Lambda.

Example: to stop the EC2 instance every Saturday at 6 PM, use the cron expression: cron(0 18 \* \* 6)

Set the target as the Lambda function we created

* To start the instance at a specific time, create another Lambda function that starts the EC2 instance and set up another CloudWatch Event rule to trigger it.

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

1. **How would you migrate a Kubernetes cluster to EKS (Elastic Kubernetes Service)?**

* We had an on-prem K8s cluster. We created the same setup in EKS using Terraform, backed up the app configs, and applied them in EKS.
* Create a VPC, subnets, and a security group for EKS.

resource "aws\_vpc" "main" {

cidr\_block = "10.0.0.0/16"

}

resource "aws\_subnet" "subnet\_a" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "us-west-2a"

map\_public\_ip\_on\_launch = true

}

resource "aws\_subnet" "subnet\_b" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.2.0/24"

availability\_zone = "us-west-2b"

map\_public\_ip\_on\_launch = true

}

resource "aws\_security\_group" "eks\_security\_group" {

vpc\_id = aws\_vpc.main.id

}

* Now, define the EKS cluster with the necessary configurations.

resource "aws\_eks\_cluster" "my\_cluster" {

name = "my-eks-cluster"

role\_arn = aws\_iam\_role.eks\_cluster\_role.arn

vpc\_config {

subnet\_ids = [aws\_subnet.subnet\_a.id, aws\_subnet.subnet\_b.id]

}

}

* Need AM role to give EKS the necessary permissions.

resource "aws\_iam\_role" "eks\_cluster\_role" {

name = "eks-cluster-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Action = "sts:AssumeRole"

Principal = {

Service = "eks.amazonaws.com"

}

Effect = "Allow"

Sid = ""

},

]

})

}

resource "aws\_iam\_role\_policy\_attachment" "eks\_cluster\_policy\_attachment" {

policy\_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"

role = aws\_iam\_role.eks\_cluster\_role.name

}

* Use managed node groups to create EC2 instances that run your pods

resource "aws\_eks\_node\_group" "node\_group" {

cluster\_name = aws\_eks\_cluster.my\_cluster.name

node\_role\_arn = aws\_iam\_role.node\_group\_role.arn

subnet\_ids = [subnet-1, subnet-2]

scaling\_config {

desired\_size = 2

max\_size = 3

min\_size = 1

}

}

* Export YAML from old cluster:

$ kubectl get all --all-namespaces -o yaml > backup.yaml

Then apply it to EKS:

$ kubectl apply -f backup.yaml

* Check if pods, services, and deployments are running fine.

$ kubectl get pods -a

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

1. **If an organization is migrating from on-premises infrastructure to AWS, how would you ensure high availability, fault tolerance, and cost optimization?**

* First, I would start by assessing the existing infrastructure like checking what apps, servers, databases, and storage are there.
* Then, I would plan the migration decide what services map to AWS services. For example, moving VMs to EC2, databases to RDS or DynamoDB, storage to S3, and so on.
* For high availability, I would make sure resources deploy across multiple Availability Zones, use Auto Scaling Groups, and Elastic Load Balancers to handle traffic efficiently.
* For fault tolerance, I would use multi-AZ deployments for databases and critical services, and design stateless applications wherever possible.
* For cost optimization, I would right-size the instance, use Reserved Instances or Savings Plans for long-term workloads, Spot Instances for non-critical tasks, and set up monitoring and alerts with CloudWatch and Trusted Advisor to keep optimizing resources over time.
* Finally, I would use a proper migration service like AWS Migration Hub or Server Migration Service (SMS) to migrate the workloads smoothly and minimize downtime.

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

1. **Which metric provide by CloudWatch?**

* CloudWatch automatically provides default metrics for AWS services like:

EC2 Instances – CPUUtilization, DiskReadBytes, DiskWriteBytes, NetworkIn, NetworkOut, StatusCheckFailed

EBS (Volumes) – VolumeReadBytes, VolumeWriteBytes, VolumeReadOps, VolumeWriteOps, VolumeIdleTime

S3 (Buckets) – NumberOfObjects, BucketSizeBytes, AllRequests

* By default, CloudWatch does NOT provide MemoryUtilization or DiskUsage for EC2 instances

To monitor memory (RAM), you need to: Install CloudWatch Agent on the EC2 instance > Configure it to collect memory, disk, and other OS-level metrics > Push custom metrics to CloudWatch.

Example:

sudo yum install amazon-cloudwatch-agent

sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard

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

1. **Did you worked on ECS?**

* ECS (Elastic Container Service) is a fully managed container service by AWS to run Docker containers. It helps us run and manage containers without setting up our own servers.
* We can use Fargate (serverless) or EC2 launch type (we manage EC2 instances).
* Example:

In one of my projects, we used ECS with Fargate to run microservices without managing servers. We created a task definition, defined the image URL from ECR, memory, CPU, and port., and ECS handled the rest.

**Git**

1. **What branching strategy do you use?**

* We follow the GitFlow branching strategy in our project. We use branches like feature, develop, release, sit, preprod, and main.
* When developers work on a new feature, they create a feature branch from develop, e.g., feature/user-login
* Once the feature is done, it gets merged back into the develop branch.
* After all features are ready, we create a new release branch from develop branch e.g release/1.0 for final bug fixes.
* After release testing in sit and preprod, the code is merged into the main branch for production deployment.

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

1. **How do you manage conflicts in Git?**

* Run "git status" command to see which files have conflicts.
* Open the conflicted files in a text editor — Git marks the conflicts with <<<<<<<, =======, and >>>>>>>.
* Discuss with developers/TL to manually resolve the conflicts by deciding which changes to keep or combine both.
* After resolving, I stage the resolved files with "git add <file>" command.
* Finally, I commit the changes with "git commit", and Git creates a merge commit automatically.
* Example:

git status # See which files have conflicts

# Edit the conflicted files manually

git add <file> # Stage the resolved files

git commit # Commit the merge

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

1. **What command do you use to merge branches in Git?**

* I use "git merge <branch-name>" to merge a branch into the current branch.

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

**Network**

1. What is a load balancer? What are the different types of load balancers, and what load balancers can be used for web applications?

* A load balancer distributes incoming traffic across multiple servers, so no single server gets overloaded. It helps improve performance, reliability, and ensures high availability.
* There are mainly two types of load balancers:

Layer 4 (Transport Layer) Load Balancer:

Works at TCP/UDP level, routes traffic based on IP address and port number. Example: AWS Network Load Balancer (NLB).

* + Layer 7 (Application Layer) Load Balancer: Works at the HTTP/HTTPS level, routes traffic based on content like URL path, hostname, or headers. Example: AWS Application Load Balancer (ALB).
  + For web applications, we mainly use Layer 7 load balancers because they can make smart decisions based on URLs, headers, cookies.
  + Some common load balancers for web apps are:
  + ALB – great for web traffic, L7
  + NGINX – very popular open-source option, works as both L4 and L7 load balancer.
  + HAProxy – another open-source one, very reliable for high-traffic sites.

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

1. **What load balancing strategies have you implemented?**

* We implemented round-robin and least connections (when some servers handle more traffic).
* Round-Robin: Distributes traffic evenly across all servers. It’s simple and works best when all servers have similar capacity.
* Least Connections: Sends new traffic to the server with the fewest active connections. It’s useful when some servers are handling heavier loads.
* For microservices, we route traffic based on things like URL paths — for example, /payments requests go to the payment service.

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

**Linux**

1. **What is the difference between awk and sed?**

* Awk is mainly used for analyzing and processing text, like working with columns in a. Use awk when you want to work with fields/columns.
* Sed is mainly used for finding and replacing text, like editing text streams on the fly. Use sed when you want to search and replace text.

**--------------------------------------------------------------------------------------------------------------------------------------------**

1. **How to build image in pipeline**

**DMZ non DM**

1. **IAM policies for EKS** Explain the deployment workflow of your first project

* Developer pushes code to Bitbucket.
* Jenkins pipeline trigger to builds & tests the code, builds Docker images and pushes them to ECR.
* Spinnaker Pipeline fetches the new image from ECR.
* Updates the Kubernetes manifest (Deployment, Service, etc.).
* Applies deployment strategy blue-green strategy.
* Spinnaker shifts traffic from the old version to the new one.
* Spinnaker integrates with Nagios to monitor deployments. If errors occur, it can roll back automatically.

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

1. How have you used Jenkins and Spinnaker in your projects?

I used Jenkins to automate build, test, and deployment processes, ensuring continuous integration and delivery. Spinnaker was used for advanced deployment strategies like blue-green deployments and canary releases, making deployments more reliable and reducing downtime.

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

1. What issues did you face in CI/CD and how did you solve them?

* Build Failures – Caused by dependency conflicts or incorrect configurations.

Solution: Used version locking in dependencies and maintained proper environment configurations.

* Slow Pipeline Execution – Due to unnecessary steps and large artifacts.

Solution: Optimized the pipeline by caching dependencies and running parallel jobs.

* Deployment Failures – Due to misconfigured Kubernetes manifests or Spinnaker pipelines.

Solution: Implemented proper Helm charts, version control, and rollback strategies.

* Secrets Management – Exposing credentials in pipelines.

Solution: Used Azure Key Vault and Kubernetes Secrets for secure management.

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

1. What AWS services have you worked with?

* S3 for storage, IAM for access control, CloudWatch for monitoring, and SNS for notifications.

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

1. How did you use Terraform?

* Used Terraform to provision AWS resources like EC2, S3, and EKS clusters.

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

1. What challenges did you face with Kubernetes?

* Faced pod scheduling issues and fixed them using resource limits and affinity rules.

1. What automation scripts have you written?

* Wrote Shell and Python scripts for log analysis, monitoring, and backup, deployment tasks.

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

1. How did you use PowerShell?

* Used PowerShell for Windows server automation and Active Directory management.

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

1. How did you use Prometheus and Grafana?

* Configured Prometheus to collect Kubernetes metrics and visualized them using Grafana.

1. **Explain CICD flow of your organization**

* Yeah, sure. In my last project, we have a full CICD setup using Jenkins, SonarQube, Docker, ECR, Helm, ArgoCD, and EKS.
* CICD flow is like:
* Code Commit:

Developers push Java code to GitHub (or Bitbucket).

* Jenkins Trigger:

A webhook triggers the Jenkins pipeline automatically.

* Code Quality Check (SonarQube):

Jenkins first runs a SonarQube scan to check for code quality and vulnerabilities using Maven commands (mvn clean verify sonar:sonar).

* Build Java Application:

If SonarQube passes, Jenkins builds the Java app (mvn clean package) and creates the .jar file.

* Docker Image Build and Push:

Jenkins then builds a Docker image, tags it with the build version, and pushes it to AWS ECR.

* Update Helm Values:

After pushing the image, Jenkins updates the Helm values file with the new Docker image tag.

* Push to Git (Helm Repo):

Jenkins pushes the updated Helm values file to the Git repository where ArgoCD is watching.

* Deployment via ArgoCD:

ArgoCD automatically detects the change in the Git repo, pulls the latest Helm chart, and deploys the new version to the EKS cluster.

* Deployment Verification:

We verify if the new pods are up and running, and ArgoCD makes sure the actual deployment matches the desired state.

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

1. What microservices have you developed?

* I have developed microservices for real-time data streaming using Kafka.
* These microservices process and store data in Apache Cassandra, PostgreSQL, and OpenSearch.

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

1. What microservices have you developed?

* I built REST APIs for data processing, user management, and real-time event handling. These APIs interacted with Kafka for messaging.

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

1. How did you handle API security?

* Used JWT for authentication and IAM roles for access control in AWS.

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

1. How did you integrate Kafka with your microservices?

* Used Kafka producers and consumers in Spring Boot for real-time data processing.

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

1. Explain your roles & responsibilities

I help developers with Git issues and manage Git repositories based on client requirements

I create and manage Jenkins jobs, set permissions, and schedule builds to support continuous integration..

I launch and configure EC2 instances on AWS, using AMIs and setting up servers as needed. I also manage Elastic IPs as per client requests.

I write Dockerfiles to create customized container images and work with Maven for automated builds and deployments.

I handle code merging between development and main branches and ensure code quality by enabling SonarQube quality checks.

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

1. Which branching strategy you are using

We use feature branch strategy. In our development process, the master branch always points to the latest production commit. The full codebase is maintained in the integration branch.

If I need to work on a new feature, I create a branch from the integration branch.

While developing, every commit goes through the CI/CD pipeline, where we track whether the build and tests are successful

Developers raise a Pull Request (PR) only if the code passes CI/CD checks in the development environment. Before merging, the PR is reviewed by other team members to verify the changes.

Once merged, the changes go through multiple environments: QA → PreProd → Production.

If the code passes all stages successfully, it is deployed to production. If any stage fails, the CI/CD process stops, and we fix the issues before proceeding.

After a successful production deployment, the master branch is updated, and we tag the commit with a version like like V1.1, V1.2, etc., making it easy to manage and rollback if needed. If conflicts arise, I resolve them and then I can push back into integration branch and release a model based on requirement.

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

1. What kinds of conflicts in github do you get how do you resolve them?
2. Merge Conflicts (Most Common)

Occurs when two branches modify the same line in a file.

Fix:

$ git pull origin main

$ git merge feature-branch # Resolve conflicts manually in the file

$ git add .

$ git commit -m "Resolved merge conflict"

$ git push origin feature-branch

1. Rebase Conflicts

Happens when rebasing a branch with changes from another branch.

Fix:

$ git rebase main

# Resolve conflicts manually

$ git add .

$ git rebase --continue

1. Push Conflicts (Rejected Push)

Occurs when pushing to a remote branch that has newer commits.

Fix:

git pull --rebase origin main

git push origin main

1. Stash Conflicts

Happens when applying stashed changes that conflict with the working directory.

Fix:

git stash pop

# Resolve conflicts

git add .

git commit -m "Resolved stash conflict"