Section 1: Core Concepts

1. Differentiate Continuous Integration, Continuous Delivery, Continuous Deployment.

   Here, CI, CD are basically devops automation tools we have.
   Continuous integration= developers frequently merge/pushes code into shared repos, and automated build/test run with CI so that any bugs will be detected early.
   Continuous Delivery= Every code change is automatically build,tested, and made ready for release/deployment. Here we get an option to manual deployment.
   Continuous Deployment = similar to Con delivery, but every change in code is automatically deployed to prod without manual intervention.

2. What is the purpose of a Jenkinsfile?

Basically, jenkinsfile defines the CICD pipeline as a code. It describes stages as build-test-deploy, ensure consistency across envirnments, version control . also makes pipeline as module for reuse or reproducible

3. Contrast declarative vs scripted pipelines.

Declarative Pipeline= it is having simple, structured syntax. Very easy to write and maintain. Best suited for microservices architecture for multipart pipeline.

Scripted Pipeline= it is basically Groovy lang based. Very flexible and useful for applications having complex logic. One drawback is hard to understand/read.

4. Difference between Freestyle job and Pipeline job.

These are mostly used job type for pipeline.
Freestyle job = It is basically having good User interface, has limited flexibility for deploying app. Mostly suited for simpler tasks.

Pipeline Job= it is executed via code i.e in most cases vis Jenkinsfile. Jenkinsfile will be stored in your git repo and called from there. It supports complex logics and used for version control also. Mostly used for complex applications.

5. How would your secure credentials (Git/Docker/Slack tokens) in Jenkins?

Credentials play very important role in any application development. best practice to follow is not to hardcode credentials. For securing credentials for Jenkins, use Jenkins credential store. When you want to access them, call them in pipeline using “withCredentials” block.

6. If a Jenkins job fails randomly, what are the first 3 things you'd check?

First thing I do is to go and check **console logs**. Will check any error message to understand root cause.

Secondly, will check for agent status, means if it is active or not.

Then I will check for memory/disk space.

And lastly, I will check for any issues with my git repo or docker registry i.e my external services.

Section 3: Debugging & Scenarios

1. Pipeline fails with error Exit code 137. What could cause this?

I think my pipeline is failing and giving error as exit code means there is issue with my nodes. And usually, nodes fail because of mostly memory related issues. So, I will check for "Out of Memory" (OOM) issue. So, my nodes must be out of memory or passed resource limits.

2. Jenkins has 2 agents but 5 long jobs. How do you optimize execution?

First thing I will do is check for application code and if possible, add more Agents because it will cause problem with limited node capacity.
And if adding more agents is not an option then will break jobs into smaller stages and segregates them on available agents
Also, will execute jobs parallel on available agents. And for this will prioritize jobs means which job is to run on which agent.

3. Jenkins master goes down. How do you recover?

First thing I will do is restart Jenkins server. Sometimes this happens because of overloading on Jenkins server as we know Jenkins will require more than 20 Jb of space.
Then, if it doesn't work, will restore it from backup.

4. A developer accidentally exposed a GitHub token in Jenkins logs. What actions do you take?

First thing I will do is to terminates respective token immediately.

Remove that token from logs also.

I will generate new token and reconfigure Jenkins credential (store)

Will also inform to developer about situations and brief him about future safe use.

Jenkinsfile

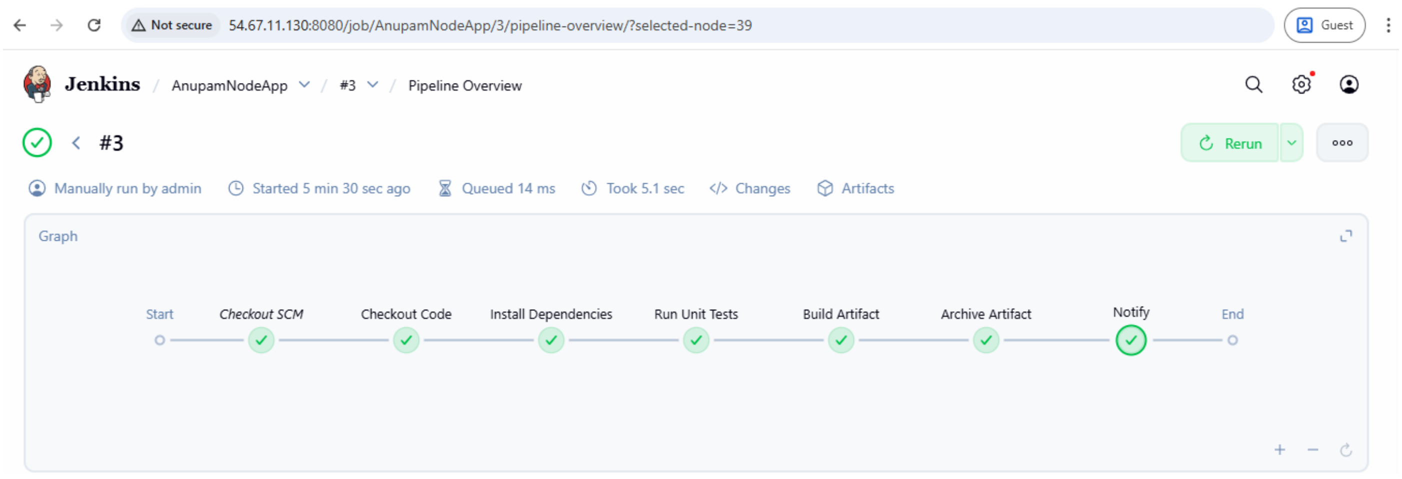
```
pipeline {
    agent any

    stages {
        stage('Checkout Code') {
            steps {
                git branch: 'main', url: 'https://github.com/anupam0312/Devops.git'
            }
        }

        stage('Install Dependencies') {
            steps {
                sh 'npm install'
            }
        }

        stage('Run Unit Tests') {
            steps {
                sh 'npm test'
            }
        }

       stage('Build Artifact') {
    steps {
        sh 'tar --exclude=node_modules --exclude=.git --exclude=Jenkinsfile -czf build.tar.gz .'
    }
}
```

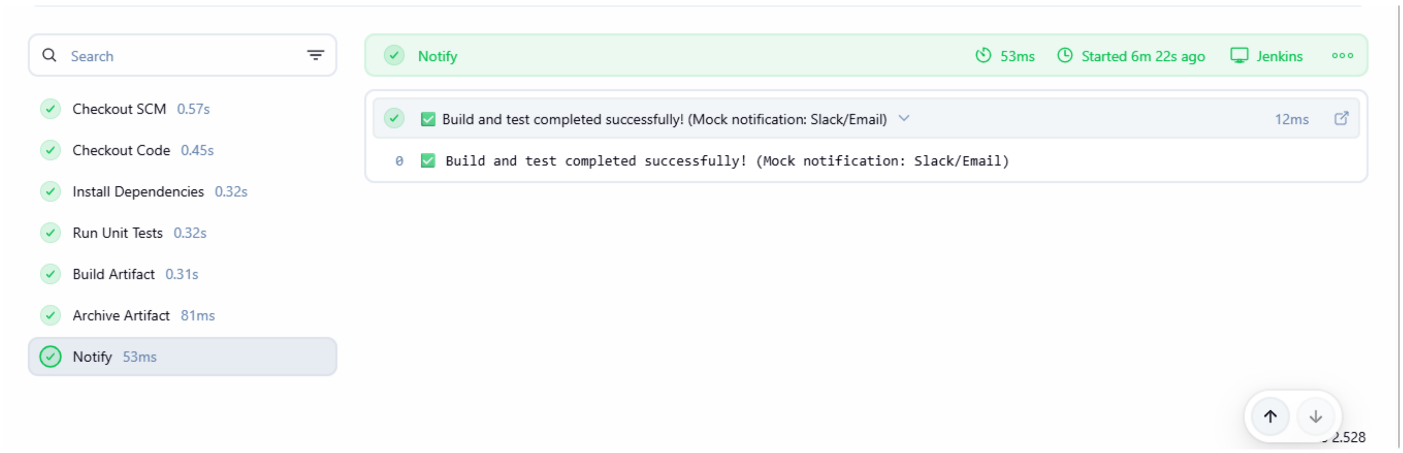
```
        stage('Archive Artifact') {
      steps {
        archiveArtifacts artifacts: 'build.tar.gz', followSymlinks: false
      }
    }



        stage('Notify') {
          steps {
            echo '✅ Build and test completed successfully! (Mock notification: Slack/Email)'
          }
        }
      }

      post {
        failure {
          echo "❌ Build failed! (Mock notification: Slack/Email)"
        }
      }
    }
```

Disclaimer-

The task required producing build.zip, but I initially used tar to create build.tar.gz because tar is native to Linux and doesn’t require extra installation. It served the same purpose of creating a compressed artifact. However, I understand the requirement was specific to .zip, and I can easily adjust the pipeline to use zip once the utility is installed.

Successful Run with All Stages =





✅ Build and test completed successfully! (Mock notification: Slack/Email)

```
✅ Build and test completed successfully! (Mock notification: Slack/Email)
```