DevOps — Jenkins (25 Questions)

Q1: A Jenkins pipeline intermittently fails during `docker build` with “no space left on device” on the agent node.

Answer:
Check workspace cleanup policies — old builds might be consuming disk. Enable `Workspace` Cleanup post-build, prune unused Docker images/volumes (`docker system prune`), and consider moving Docker storage to a larger disk. For multi-tenant agents, enforce quota per build.
Sample Points:

- Prune Docker artifacts periodically.
- Workspace cleanup plugin.
- Larger disk or separate volume for `/var/lib/docker`.

Example Code:

```
post {
    always {
        cleanWs()
        sh 'docker system prune -af'
    }
}
```

Q2: Jenkins shared library update isn’t reflecting in new pipeline runs.

Answer:
If library is cached, ensure @Library('my-lib@main') _ is used with version/tag. In Jenkins global config, set “Load implicitly” to false for manual control, and clear the SCM cache in `$JENKINS_HOME/caches`.
Sample Points:

- Explicit version tag avoids stale refs.
- Clear cache when updating.
- Avoid master default for stability.

Example Code:

```
@Library('my-lib@v1.2.3') _
```

Q3: Multi-branch pipeline triggers twice on a single commit.

Answer:
Likely duplicate webhooks or SCM polling enabled alongside webhooks. Remove extra webhook in repo settings, disable polling triggers in pipeline config.
Sample Points:

- Avoid mixed triggers (webhook + polling).
- Audit SCM webhook configuration.
- Use a single trigger source.

Example Code:

```
triggers { githubPush()}
```

Q4: Pipeline using `parallel` stages sometimes skips a branch without error.

Answer:
Ensure `failFast: false` if you want all branches to run independently. Also wrap each branch in `catchError` to prevent one failure stopping others.
SamplePoints:

- Use `failFast` intentionally.

- Catch errors in each parallel branch.
- Aggregate results after all run.

  Example Code:

```
parallel(
  branch1: { catchError { sh 'run-tests.sh' } },
  branch2: { catchError { sh 'lint.sh' } },
  failFast: false
)
```

Q5: Credentials binding to environment variables leaks in console output.

Answer:
Use `withCredentials` block to mask secrets and avoid echoing variables. Enable "Mask Passwords" plugin, set `echo false`.
Sample Points:

- Never echo secrets.
- Use binding plugins to mask automatically.
- Store in Jenkins Credentials store only.

  Example Code:

```
withCredentials([usernamePassword(credentialsId: 'dockerhub',
usernameVariable: 'USER', passwordVariable: 'PASS')]) {
    sh 'docker login -u $USER -p $PASS'
}
```

Q6: Distributed builds fail on a new agent with "Permission denied" when accessing workspace.

Answer:
Check that agent user has write permissions to Jenkins home and workspace dirs. Align file

ownership between master and agents, and consider using chown post-workspace creation.

Sample Points:

- File ownership consistency.
- Use dedicated Jenkins user on agents.
- Avoid running as root unless required.

Example Code:

```
sudo chown -R jenkins:jenkins /var/lib/jenkins/workspace
```

Q7: You must dynamically provision agents for Kubernetes builds.

Answer:
Install Kubernetes plugin, define pod templates with container specs for build environments, use podTemplate in pipelines. Limit pod idle timeout for cost savings.

Sample Points:

- Pod templates per job type.
- Idle timeout to control costs.
- Avoid privileged containers unless needed.

Example Code:

```
podTemplate(label: 'maven-agent', containers: [
  containerTemplate(name: 'maven', image: 'maven:3.8', ttyEnabled:
true, command: 'cat')
]) {
    node('maven-agent') { sh 'mvn clean install' }
}
```

Q8: Pipeline artifacts aren't available to downstream jobs.

Answer:
Use archiveArtifacts in upstream and copyArtifacts plugin in downstream, matching build numbers or tags. Ensure retention policy keeps artifacts until downstream completes.
Sample Points:

- Archive artifacts in upstream.
- Match build IDs in copy step.
- Adjust retention policies.

Example Code:

```
archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
```

Q9: Need to fail pipeline if code coverage drops below 80%.

Answer:
Parse coverage report (e.g., Jacoco XML) in a stage, compare value to threshold, fail build with error() if below.
Sample Points:

- Automate coverage checks.
- Fail early to enforce quality.
- Use coverage plugins or scripts.

Example Code:

```
def coverage = readFile('coverage.txt').trim().toInteger()
if (coverage < 80) error "Coverage below 80%: $coverage%"
```

Q10: Jenkins master CPU spikes due to concurrent builds.

Answer:
Move heavy jobs to agents, reduce master's executor count to 0, and use the master only for orchestration.
Sample Points:

- Master for orchestration only.
- Offload builds to agents.
- Reduce master executor count.

Example Code:

```
node('build-agent') { sh 'mvn clean install'}
```

Q11: Pipeline must rollback to previous app version on deployment failure.

Answer:
Use a post-deploy stage to check status and call rollback script or redeploy last stable artifact from `archiveArtifacts`.

Sample Points:

- Rollback script as part of pipeline.
- Keep last stable artifact.
- Automate detection of failure.

Example Code:

```
post {
    failure {
        sh './rollback.sh'
    }
}
```

Q12: Git shallow clone causes missing tags in Jenkins pipeline.

Answer:
Increase depth or disable shallow clone for builds requiring tags. Use `checkout` with `depth: 0`to fetch all history.

Sample Points:

- Shallow clone saves time but limits history.

- Fetch full history when tags are needed.
- Configure per-pipeline.

Example Code:

```
checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions:
[[$class: 'CloneOption', depth: 0]]])
```

Q13: Need to pass parameters from one pipeline to another triggered job.

Answer:
Use `build`step with `parameters` block. Ensure downstream is parameterized.
Sample Points:

- Upstream triggers with parameters.
- Downstream must accept parameters.
- Match parameter names exactly.

Example Code:

```
build job: 'downstream', parameters: [string(name: 'VERSION', value:
'1.0')]
```

Q14: Pipeline fails due to Maven repo corruption in agent cache.

Answer:
Clean local repo (`~/.m2/repository`) on failure, consider mounting a clean volume per build or using a cache restoration plugin.
Sample Points:

- Cache can corrupt over time.
- Periodic clean reduces flakiness.
- Use immutable cache artifacts.

Example Code:

```
sh 'rm -rf ~/.m2/repository && mvn clean install'
```

Q15: Secret text credential must be injected into a Docker build without leaking in image history.

Answer:
Use --build-arg with ARG in Dockerfile, then unset inside build step. Avoid embedding secrets in final layers.
Sample Points:

- Build args not persisted in layers.
- Avoid ADDing secret files.
- Mask secrets in Jenkins logs.

Example Code:

```
withCredentials([string(credentialsId: 'api-key', variable:
'API_KEY')]) {
    sh "docker build --build-arg API_KEY=$API_KEY ."
}
```

Q16: Long-running pipeline stage must auto-abort after 30 minutes.
Answer:

Wrap stage with `timeout`block. This prevents stuck builds from hogging agents.
Sample Points:

- Timeout for runaway stages.
- Auto-abort frees agents.
- Adjust per stage.

Example Code:

```
timeout(time: 30, unit: 'MINUTES') { sh './run-tests.sh'}
```

Q17: Blue Ocean view missing some pipeline stages.

Answer:
Ensure stages are defined with `stage('name')` and not inside raw script without stage. Blue Ocean only renders defined stages.
Sample Points:

- Define all logical steps as stages.
- Avoid script blocks swallowing stages.
- Use parallel stages for clarity.
  Example Code:

```
stage('Build') { sh 'mvn clean package'}
```

Q18: Must conditionally skip deployment stage if commit message contains `[skip deploy]`.

Answer:

Read `env.GIT_COMMIT` message, skip stage with `when` directive.
Sample Points:

- Use regex in `when` for commit messages.
- Prevent accidental prod deploys.
- Lightweight safeguard.
  Example Code:

```
stage('Deploy') {
  when { not { expression { sh(script: "git log -1 --pretty=%B | grep
-q '\\[skip deploy\\]'", returnStatus: true) != 0 } } }
  steps { sh './deploy.sh' }
}
```

Q19: Jenkins agent on Kubernetes fails due to missing Docker socket.

Answer:
Use DinD (Docker in Docker) container or mount host socket if security allows. Alternatively, build with Kaniko/Buildah in rootless mode.
Sample Points:

- Host socket mount vs DinD trade-offs.
- Kaniko avoids privileged mode.
- Security vs compatibility choice.

Example Code:

```
volumeMounts:
- mountPath: /var/run/docker.sock
  name: docker-sock
```

Q20: Pipeline must only run if specific files changed.

Answer:
Use `when { changeset pattern: 'src/**', comparator: 'ANT' }` to limit execution.
SamplePoints:

- Changeset condition reduces wasted builds.
- Pattern matching supports ANT/regex.
- Define in Jenkinsfile for visibility.

Example Code:

```
when { changeset pattern: "src/**", comparator: "ANT"}
```

Q21: Jenkins master restarted mid-build, and job didn't resume.

Answer:
Ensure Pipeline is configured with `pipeline durability hint: MAX_SURVIVABILITY`

and Checkpoint plugin or external workspace to resume.

Sample Points:

- Durable pipelines survive restarts.
- Checkpoint allows resume points.
- Requires persistent workspace.

Example Code:

```
options { durabilityHint('MAX_SURVIVABILITY')}
```

Q22: Need to store pipeline logs in S3 for long-term audit.

Answer:

Use S3Log Publisher plugin or post-build script to upload logs to S3 with build metadata in the path.

SamplePoints:

- Long-term retention offloads from master.
- Metadata in S3 keys aids search.
- Secure bucket with IAM policy.

Example Code:

```
post {
    always {
        sh 'aws s3 cp ${BUILD_LOG}
s3://jenkins-logs/${JOB_NAME}/${BUILD_NUMBER}.log'
    }
}
```

Q23: Groovy sandbox prevents running certain methods in pipeline.

Answer:

Approve scripts in Script Approval page or run pipeline in “trusted” shared library. Limit who can

bypass sandbox.
Sample Points:

- Sandbox protects from unsafe calls.
- Script Approval grants specific methods.
- Use shared libs for trusted code.
  Example Code:

```
@Library('trusted-lib') _
```

Q24: Pipeline must deploy to multiple environments sequentially with approval in between.

Answer:
Use `input`step between stages for manual approval.
Sample Points:

- Input pauses for human gate.
- Sequential deploys per environment.
- Timeout input to avoid hangs.
  Example Code:

```
stage('Deploy Staging') { steps { sh './deploy-staging.sh'}}
stage('Approval') { steps { input 'Deploy to Prod?'}}
stage('Deploy Prod') { steps { sh './deploy-prod.sh'}}
```

Q25: Jenkinsfile in monorepo must build only the affected service.

Answer:
Parse `git diff` to detect changed directories, set env var for service path, run targeted build.
Sample Points:

- Detect changed paths with git diff.

- Targeted build saves time/cost.
- Works in monorepo CI/CD.
  Example Code:

```
def changed = sh(script: "git diff --name-only HEAD~1 | cut -d/ -f1|
sort -u", returnStdout: true).trim()
if (changed.contains('service-a')) { buildServiceA() }
```