DUMPSBOSS.

Certified Jenkins Engineer (CJE)

CloudBees CJE

Version Demo

Total Demo Questions: 23

Total Premium Questions: 230

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Jenkins Fundamentals	112
Topic 2, Jenkins Administration	39
Topic 3, Jenkins Pipeline	58
Topic 4, Jenkins Security	21
Total	230

QUESTION NO: 1

You're at a job interview and the interviewer looks at you, trying to make you nervous. He looks down at his paper, looks up at you and asks, "How would you describe continuous integration?" You think to yourself. Which of the following answers is best?

A. Building in 60 minutes or less.

B. A software development discipline where software is built so that it can be released to production at any time.

C. A software development discipline where software is released continuously as part of an automated pipeline.

D. A software development practice where contributors are integrating their work very frequently.

ANSWER: D

Explanation:

A software development practice where contributors are integrating their work very frequently. is the best description of continuous integration (CI). CI is a development practice in which team members regularly merge their changes into a shared source-code repository, ideally several times per day. Each integration is then verified through an automated build and automated tests, allowing the team to identify integration defects close to the change that introduced them. This shortens feedback cycles, reduces the risk and difficulty of large end-of-branch merges, and helps maintain a dependable main code line. Jenkins is commonly used to implement this feedback loop by monitoring source control, triggering builds for incoming changes, and reporting build and test results to developers. CI does not itself require deploying releases to production; its essential focus is frequent integration and automated validation of the integrated work. CloudBees describes Jenkins as an automation server that supports continuous integration through building, testing, and delivering software, while the Jenkins documentation explains how pipelines automate these stages. See the Jenkins documentation and the CloudBees overview of continuous integration.

QUESTION NO: 2

DevOps teams can implement traceability of artifacts in a continuous delivery pipeline by using

A. the Downstream Builds plugin

B. manual recording

C. the Pipeline plugin to fingerprint files

D. Pipeline labels

ANSWER: C

Explanation:

The Pipeline plugin to fingerprint files provides artifact traceability by recording a fingerprint, typically an MD5 checksum, for files that a build produces or consumes. Jenkins stores this fingerprint metadata and associates it with the jobs and build numbers in which the file appeared. Consequently, a team can inspect an artifact’s fingerprint record to understand where that exact artifact originated and identify later builds or deployments that used the same file. This creates an auditable relationship across stages and jobs, even when artifacts move between independently configured pipelines.

In a Pipeline, the `fingerprint` step can target one or more files using a file pattern. Teams commonly fingerprint immutable deliverables such as JARs, WARs, ZIP archives, container-related metadata, or deployment packages immediately after they are created and before they are promoted. The resulting lineage helps support release investigation, impact analysis, and reliable promotion decisions because Jenkins tracks the identity of the actual artifact rather than relying solely on a build label or manually maintained record. Jenkins documents fingerprinting and its build-usage tracking behavior at Using Jenkins Fingerprints, and documents the Pipeline fingerprint step at Pipeline Steps: fingerprint.

QUESTION NO: 3

Which of the following can cause a build to remain in the Jenkins build queue? Choose 3 answers

A. No idle executor is available on an agent matching the job's label.

B. The only eligible agent is offline.

C. An earlier build of the same job is running while concurrent builds are disabled.

D. The job has archived artifacts from a previous build.

E. The job has a successful build in its Build History.

ANSWER: A B C

Explanation:

A build remains in the queue until Jenkins can satisfy all scheduling requirements. It may wait because no idle executor is available on an agent matching the job's assigned label. It can also wait when an eligible agent is offline or disconnected, since that agent cannot accept work. If a job or Pipeline is configured to disallow concurrent builds, a later build of that same job can remain queued while an earlier build is still running.

Queue status is therefore not necessarily an error. Jenkins displays the reason for a queued item in the user interface, which helps administrators identify whether additional capacity, an online agent, or a configuration change is required. See the Jenkins documentation on agents and executors and the documentation on managing nodes.

QUESTION NO: 4 - (SIMULATION)

What are the main advantages of using webhooks/post commit hooks from your Source Code Management system to trigger your Jenkins project rather than using SCM polling? Choose 2 answers A A. Builds are started on a defined a on schedule.

B. Avoid unnecessary overhead from polling.

C. Builds are started immediately after changes are committed.

D. The entire repository Is scanned so no commits are missed.

ANSWER: See the explanation for the answer

Explanation:

Correct answers: B and C.

B. Avoid unnecessary overhead from polling. Webhooks notify Jenkins only when a relevant SCM event occurs, rather than Jenkins repeatedly querying the SCM server on a schedule.
C. Builds are started immediately after changes are committed. A post-commit hook/webhook can trigger Jenkins as soon as the SCM system sends the commit event.

A describes scheduled triggering, which is polling behavior rather than a webhook advantage. **D** is not a webhook advantage; Jenkins does not need to scan the entire repository merely because a webhook is used.

QUESTION NO: 5

What types of notification integrations are there?

A. Email

B. All of these

C. Slack

D. HipChat

ANSWER: B

Explanation:

All of these is correct because Jenkins supports notifications through its plugin ecosystem, including email, Slack, and HipChat integrations. Email notifications are commonly configured through the Mailer or Email Extension plugins, allowing builds and pipeline jobs to send status messages to configured recipients. Slack integrations enable Jenkins jobs and pipelines to publish build results, links, and other updates to Slack channels. Jenkins has also provided a HipChat notification plugin, enabling build messages to be delivered to HipChat rooms where that integration is in use.

These integrations represent different delivery channels rather than mutually exclusive notification types. A Jenkins administrator can install and configure the appropriate plugins to match the organization's communication tools, and pipeline code can invoke supported notification steps as part of post-build handling. This extensibility is a core Jenkins capability: integrations are supplied and managed as plugins, with each plugin adding its own configuration and pipeline features. The Jenkins documentation describes plugin management and the availability of plugins through the Update Center, while the Slack Notification plugin documentation details Jenkins-to-Slack notifications. See Jenkins: Managing Plugins and Slack Notification plugin.

QUESTION NO: 6

You are using a base Jenkins master in production with the recommended set of plugins.

The administrators have configured a Global Pipeline Library named "common-libs", stored in a git repository, with the configuration shown in the exhibit above.

You have a Pipeline job at the root of the Jenkins dashboard, whose script starts with the annotation @LibraryCcommon-libs') _

You want to test this Pipeline job with a beta version of the Global Pipeline Library "common-libs", stored on the branch named ''edge'' the git repository.

Which of the following statements are TRUE? Choose 2 answers

A. Replace the `@Library('common-libs')` annotation with `@Library('common-libs@edge')` and run the job again.

B. Duplicate the Pipeline job in a folder with a local Global Pipeline Library referencing the Git repository on the `edge` branch, and replace the annotation with `@Library('common-libs:edge')`.

C. Duplicate the Pipeline job in a folder with a local Global Pipeline Library referencing the Git repository on the `edge` branch, and change its annotation to `@Library('common-libs@edge')`.

D. A Jenkins test instance with the same configuration as the production Jenkins master can be used to achieve this, giving yourself administrator rights to configure the `edge` branch in the Global Configuration.

E. You cannot do this on this Jenkins instance.

ANSWER: A D

Explanation:

Replace the `@Library('common-libs')` annotation with `@Library('common-libs@edge')` and run the job again. This is the standard Pipeline Shared Library version-selection syntax: the portion after `@` requests a specific library version, which can be a Git branch such as `edge`. This approach depends on the global library configuration allowing its default version to be overridden, as indicated by the relevant library setting in Jenkins. It lets the Pipeline load the beta branch for that run without changing the configured default version used by other jobs. Jenkins documents versioned library references and the requirement that overriding be enabled in the library definition at Jenkins Pipeline Shared Libraries: Library Versions.

A Jenkins test instance with the same configuration as the production Jenkins master can be used to achieve this, giving yourself administrator rights to configure the `edge` branch in the Global Configuration. A separate test controller is an appropriate way to validate a library branch in an environment that mirrors production while avoiding changes to the production controller's global library configuration. An administrator can set the library's default version to the beta branch on that test instance, then execute the copied or equivalent Pipeline there. CloudBees also describes centrally configured shared libraries and their use by Pipelines at CloudBees CI Shared Libraries documentation.

QUESTION NO: 7

You are using the Jenkins CLI to communicate with a remote Jenkins master. Which are valid ways to authenticate your identity gain access?

Choose 2 answers

A. A Jenkins user's username and GitHub API token.

B. An SSH key matching an entry in the authorized_keys file of the user account that the Jenkins master process runs "as".

C. A Jenkins user's username and Kerberos token.

D. An SSH key matching a Jenkins user's public key.

E. A Jenkins user's username and password or API token.

ANSWER: D E

Explanation:

Both **An SSH key matching a Jenkins user's public key.** and **A Jenkins user's username and password or API token.** are supported Jenkins CLI authentication methods. When the CLI uses Jenkins' SSH transport, Jenkins identifies the caller by matching the private key used by the client with a public key registered in that Jenkins user's profile. The public key is managed within Jenkins under the user's configured SSH public keys, allowing Jenkins to associate the SSH connection with the appropriate authenticated user and apply that user's permissions.

For CLI commands sent over HTTP or HTTPS, Jenkins supports the `-auth` mechanism with credentials in the form `username:password` or, preferably, `username:API-token`. API tokens are intended for programmatic access and can be created and revoked per user, avoiding the need to expose an account password in automation. After authentication, Jenkins authorization still determines whether that identity may execute the requested CLI command. Jenkins' CLI documentation describes both HTTP(S) authentication and SSH public-key authentication, while the API-token documentation explains token-based access for scripts and remote clients. See the Jenkins CLI documentation and Jenkins API token documentation.

QUESTION NO: 8

What do you need to configure an SSH agent?

A. A username and password configured for the slave node

B. SSH keys with the Master ' s pub key set as an authorized key on the agent node

C. No special authorization setup

D. None of these

ANSWER: B

Explanation:

SSH keys with the Master ' s pub key set as an authorized key on the agent node is correct because Jenkins connects to an SSH-launched agent by authenticating to the remote machine with an SSH credential. In the usual key-based setup, the Jenkins controller holds the private key (either directly or through a configured credential), while the corresponding public key is installed in the target agent account's `~/.ssh/authorized_keys` file. This establishes that the controller is permitted to open an SSH session as that account without interactive password entry. After the connection is established, Jenkins transfers and starts the agent runtime remotely, allowing the machine to accept build work.

Current Jenkins terminology uses *controller* rather than *master*, but the key requirement described in the answer remains the same: the public half of the authentication key pair must be authorized for the remote user on the agent node, and Jenkins must have access to the matching private key. The SSH Build Agents plugin documents this launch method and its required

host, credentials, and remote filesystem configuration. See the SSH Build Agents plugin documentation and the Jenkins credentials management documentation.

QUESTION NO: 9

Which statements are true about labels assigned to Jenkins agents? Choose 3 answers

A. An agent can be assigned more than one label.

B. A Pipeline can use a label expression to select a suitable agent.

C. Labels can describe capabilities such as an operating system or installed tool.

D. Each label can be assigned to only one agent.

E. A label determines the number of executors on an agent.

ANSWER: A B C

Explanation:

An agent can have more than one label, allowing it to advertise multiple capabilities such as its operating system, installed tools, location, or hardware characteristics. Jobs and Pipelines can request an agent using a label expression, and Jenkins schedules the work only on an agent whose labels satisfy that expression.

Labels can also be combined with logical operators. For example, an expression can require an agent with both a Linux label and a particular tool label. This lets administrators define flexible scheduling rules without creating a separate agent definition for every combination of capabilities. See the Jenkins documentation on using agents and the Declarative Pipeline agent documentation.

QUESTION NO: 10

What's "Infrastructure as Code?"

A. None of these

B. Infrastructure is defined and managed through version-controlled code.

C. There's no hardware

D. Source code is hosted in your Infrastructure

ANSWER: B

Explanation:

Infrastructure as Code is the practice of defining, provisioning, and managing infrastructure through machine-readable configuration files or code rather than making manual changes through a graphical interface or individual commands. These definitions can describe resources such as virtual machines, networks, load balancers, permissions, and Kubernetes resources. Keeping the definitions in version control makes infrastructure changes reviewable, repeatable, auditable, and easier to reproduce consistently across environments. In a Jenkins-centered delivery workflow, a Jenkinsfile is more precisely an example of Pipeline as Code: it stores a build, test, and delivery pipeline as versioned code alongside an application. The same core principle—declaring operational behavior in source-controlled files—also underpins Infrastructure as Code, but infrastructure itself is commonly managed with tools such as Terraform, CloudFormation, Ansible, Puppet, or Chef. AWS describes Infrastructure as Code as managing and provisioning infrastructure through code instead of manual processes, and Jenkins documents the Jenkinsfile as the foundation of Pipeline as Code. See AWS: Infrastructure as Code and Jenkins: Using a Jenkinsfile.

QUESTION NO: 11

What type of views can be configured in Jenkins?

A. email

B. notice

C. emailext

D. alert

E. List view

ANSWER: E

Explanation:

List view is correct because Jenkins uses views to organize and present jobs on the dashboard. A list view is the standard built-in Jenkins view type: it displays selected jobs in a configurable table and can include job status, health indicators, build history, and other columns. Administrators and users with the required permissions can create a new view, give it a name, select *List View*, and define which jobs it contains, either by manually selecting jobs or by applying a regular-expression filter. This makes it useful for grouping jobs by team, application, environment, release stream, or any other naming convention. Jenkins may offer additional view types when plugins are installed, but List View is the core view type available in Jenkins itself. The terms *email*, *notice*, *emailext*, and *alert* relate to notifications or do not identify Jenkins view types. In particular, the Email Extension plugin provides the `emailext` Pipeline step for sending email, rather than configuring a dashboard view. See the Jenkins documentation on using views and the Email Extension Pipeline step.

QUESTION NO: 12

Which statements are true about the `stash` and `unstash` steps in a Pipeline? Choose 3 answers

A. `unstash` restores files saved by a named `stash`.

B. Stashes are normally available only for the current Pipeline run.

C. `preserveStashes()` can retain stashes for Declarative Pipeline restart.

D. A stash is automatically shared with all jobs on the controller.

E. Stashes are the preferred long-term repository for large release artifacts.

ANSWER: A B C

Explanation:

The `stash` step saves files from the current workspace for later use in the same Pipeline run, and `unstash` restores files saved under a specified stash name into the current workspace. These steps are commonly used when separate stages run on different agents and a small set of files must be transferred between their workspaces.

Stashes are normally discarded at the end of a Pipeline run. The Declarative `preserveStashes()` option retains stashes from completed builds to support restarting a Declarative Pipeline from a later stage. Stashes are intended for relatively small files; Jenkins recommends using an external artifact repository for large files or files that must be retained beyond a Pipeline run. See the stash step reference and the Declarative Pipeline options reference.

QUESTION NO: 13

Your organization has been carefully and painstakingly performing builds on specific commits which the development team deems as releases or release candidates and subsequently only testing major releases. You have been placed in charge of a new project in which continuous integration is the primary goal. Which part of your organization's existing process do you need to modify in the furtherance of the goal of continuous integration?

A. Building every commit.

B. Maintaining a single source repository.

C. Building everything manually with great care.

D. Making Builds Self-Testing.

ANSWER: C

Explanation:

Building everything manually with great care. is the process element that must change to support continuous integration. CI depends on an automated, repeatable build and verification process that provides rapid feedback whenever changes are integrated. A manually performed build, even when executed carefully, is too slow, difficult to run consistently, and inherently limits how often the team can validate changes. Automation allows the same build steps, dependency retrieval, compilation, packaging, and test execution to be performed reliably without waiting for a person to initiate and oversee each build.

In Jenkins, this automation is normally expressed as a Pipeline stored alongside the application source in a `Jenkinsfile`. The Pipeline can run automatically when source changes are detected, execute the build in a defined environment, and publish the resulting status to the team. This makes integration feedback timely and dependable, which is essential when teams integrate changes frequently rather than reserving builds and tests for selected release candidates. Jenkins documentation describes Pipeline as a durable, automated model for delivering software, while CloudBees describes CI as a practice centered on frequent integration and automated verification. See the Jenkins Pipeline documentation and the CloudBees CI Pipeline documentation.

QUESTION NO: 14

You are in charge of a new development process and find that the current branching strategies may have room for improvement. Which types of common strategies might you recommend?

A. Feature branching and release branching, but not developer branching, are common branching strategies.

B. Developer branching and feature branching, but not release branching, are common branching strategies.

C. Developer branching, feature branching, and release branching are common branching strategies.

D. Release branching and developer branching, but not feature branching, are common branching strategies.

ANSWER: C

Explanation:

Developer branching, feature branching, and release branching are common branching strategies. A development team can use developer branches as short-lived working areas for an individual developer's changes, helping isolate work before it is integrated into a shared branch. Feature branches similarly isolate work, but organize it around a specific capability or story; they are typically merged when the feature is complete and validated. Release branches provide a stable line for preparing, testing, maintaining, and, where necessary, patching a release while work toward later versions continues elsewhere.

These strategies are particularly relevant to Jenkins because Multibranch Pipeline automatically discovers branches and can create and run a separate Pipeline for each branch. This lets teams apply continuous integration to feature, developer, and release work without requiring a single shared pipeline execution path. The appropriate lifespan, merge policy, naming convention, and protection rules should be defined by the team's delivery and release practices, but each branch type is a recognized and useful way to organize parallel development. See the Jenkins Multibranch Pipeline documentation and the Jenkins Pipeline documentation.

QUESTION NO: 15

Which of the following are true about an orphaned item strategy in a Multibranch Pipeline project? Choose 3 answers

A. It can remove child jobs for branches that no longer exist in the source repository.

B. It can retain orphaned child jobs for a configured number of days.

C. It can retain a configured maximum number of orphaned child jobs.

D. It deletes branches from the source-control repository.

E. It prevents Jenkins from indexing new branches.

ANSWER: A B C

Explanation:

An orphaned item strategy controls retention of child jobs that were previously discovered but no longer correspond to an active branch, pull request, or other repository head. For example, when a branch is deleted from source control, its associated child Pipeline job can be identified as orphaned during indexing.

The strategy can discard orphaned child jobs according to age and/or a maximum number of retained items. This prevents obsolete branch jobs, their build history, and related metadata from accumulating indefinitely. It does not delete the source-control branch itself; source-control lifecycle remains the responsibility of the SCM system. See the Jenkins Multibranch Pipeline documentation.

QUESTION NO: 16

Which TIIKII of the following are considered best practices when setting up rules for notifications?

A. Send notifications only when direct intervention is needed.

B. Do not send developers too many email notifications.

C. Make sure the notification's target Is the right person.

D. Send notifications by multiple channels (email, chat room, PagerDuty, etc.) to make sure they are received.

E. Periodically change the recipient of emails, to make sure they are not classified as spam.

ANSWER: A B C

Explanation:

Effective Jenkins and CloudBees CI notifications should be actionable, relevant, and routed to an accountable audience. “Send notifications only when direct intervention is needed.” is a sound alerting principle because recipients should receive a notification when there is an action they can take, such as investigating a failed build, addressing a blocked deployment, or responding to an operational incident. Restricting alerts to actionable events preserves the signal-to-noise ratio and helps teams respond promptly when attention is genuinely required.

“Do not send developers too many email notifications.” is also essential. Excessive build emails create alert fatigue: people begin ignoring or filtering messages, including messages that may require timely action. Notification rules should therefore be scoped to meaningful build states, relevant branches, and appropriate severity levels.

“Make sure the notification's target Is the right person.” ensures that a failure or status change reaches the person or team able to own the response. Jenkins supports configuring recipients and notification behavior in jobs and Pipelines, allowing teams to direct information to the relevant maintainers. See the Jenkins email documentation and the Jenkins Pipeline syntax documentation for notification-related configuration patterns.

QUESTION NO: 17

A unit test_____.

A. tests an individual unit or component

B. verifies cross-functionalities

C. verifies that the complete software matches the specifications it was written to fulfill

D. is written when an Integration or multi-environment bog is fixed

ANSWER: A

Explanation:

tests an individual unit or component is correct because unit testing focuses on the smallest practical, independently testable part of an application, such as a method, class, function, or module. The test supplies controlled inputs and checks the unit's observed result against its expected behavior. Its purpose is to give developers fast, repeatable feedback that a localized piece of code behaves as intended while it is being developed or changed.

For reliable unit tests, dependencies outside the unit's responsibility are generally isolated or replaced with test doubles, allowing the test result to identify behavior in that specific unit rather than behavior from external services, databases, networks, or other application components. This scope makes unit tests well suited for frequent execution in continuous integration pipelines, where quick feedback helps detect regressions soon after a change is introduced. Jenkins can execute unit-test tooling and publish test reports as part of a Pipeline, helping teams make these checks visible and actionable. JUnit's documentation describes tests as automated checks of code behavior, while Jenkins documents publication of JUnit-formatted test results. See the JUnit User Guide and Jenkins JUnit Pipeline step documentation.

QUESTION NO: 18

Which are commonly referenced as key points of CI?

Choose 3 answers

A. Automated deployment to the production environment.

B. Collaboration among Dev, QA and Ops.

C. Automated tests after each commit

D. Automated builds after each commit.

E. Frequent commits to source code repository.

ANSWER: C D E

Explanation:

Continuous Integration centers on integrating small code changes into a shared source repository frequently, then validating every integration through an automated build and automated tests. "Frequent commits to source code repository." is therefore a core CI practice: keeping changes small and integrating them regularly reduces the cost and risk of merging work. "Automated builds after each commit." is also fundamental because the CI server must build the integrated revision consistently and provide rapid feedback about whether it remains usable. "Automated tests after each commit" completes that feedback loop by checking that the build continues to satisfy the team's defined automated quality checks. Together, these practices detect integration defects near the time they are introduced, allowing the team to fix them while the relevant change is still small and well understood. Jenkins supports this workflow through Pipeline automation and source-control triggers, so builds and tests can run whenever repository changes are received. Martin Fowler's foundational CI description likewise identifies frequent integration, automated builds, and self-testing builds as central CI elements. See Jenkins Pipeline documentation and Continuous Integration by Martin Fowler.

QUESTION NO: 19

Which of the following practices are recommended for a Declarative Pipeline? Choose 3 answers

A. Use the pipeline DSL to implement intricate networking and computational tasks that your Pipeline needs to do.

B. Simplify the test/debug process and improve performance of your pipeline by defining separate steps for each Important task performed by the pipeline.

C. Encapsulate common Jenkins logic within shared libraries when leveraging Declarative Pipelines.

D. Call scripts written in Shell, Batch, Groovy, or Python to implement complex logic required for your pipeline; call these scripts as steps in your pipeline.

E. Use tools such as Maven, Gradle, npm, Ant, and Make to define most of the build work; call these executables as steps in your pipeline.

ANSWER: C D E

Explanation:

Declarative Pipeline is intended to provide a clear, maintainable description of continuous-delivery workflow while keeping substantial implementation logic outside the Jenkinsfile. Encapsulating common Jenkins logic within shared libraries is recommended because it avoids duplicating trusted, reusable pipeline functionality across repositories and keeps individual Jenkinsfiles focused on the application-specific workflow. Jenkins supports loading such libraries globally or per pipeline, including use from Declarative Pipeline.

For complex logic, invoking scripts written in Shell, Batch, Groovy, or Python from pipeline steps is recommended. This makes the logic easier to develop, test, lint, and run independently of Jenkins, while the Jenkinsfile remains concise and orchestrates the work. Similarly, build systems such as Maven, Gradle, npm, Ant, and Make should perform the bulk of build, test, and packaging work. The pipeline should invoke these established tools rather than reproduce their behavior in Pipeline DSL. This separation uses each tool for its intended responsibility and improves portability and maintainability. Jenkins specifically advises keeping Pipeline code focused on orchestration and placing nontrivial build logic in external build tools or scripts. See the Jenkins Shared Libraries documentation and the Pipeline Best Practices guide.

QUESTION NO: 20

In order to send email notifications on build completion using Jenkins' built in mail functionality, which TWO of the following must be true?

A. The job must be configured to send email.

B. Jenkins must be successfully configured to point to a mail server.

C. Jenkins must be directly connected to the Internet.

D. Jenkins must have a unique email address.

E. Sendmail must be Installed and running on the same machine as Jenkins.

ANSWER: A B

Explanation:

Jenkins' built-in mail functionality requires both job-level notification configuration and a working global SMTP configuration. The job must be configured to send email because Jenkins needs an explicit post-build email notification setting, including recipient information and the build results that should trigger notification. This configuration is normally added in the job's post-build actions using Jenkins' Mailer functionality.

Jenkins must also be successfully configured to point to a mail server. In *Manage Jenkins → Configure System*, the Mailer settings identify the SMTP server and, when required, its port, authentication credentials, and encryption settings. Jenkins submits the notification to that configured SMTP service; the service may be internal to an organization or externally hosted, so Jenkins itself does not need a direct Internet connection. These two configurations work together: the job determines when and to whom mail is sent, while the system configuration provides the delivery mechanism. See the Jenkins documentation on system configuration and the Mailer plugin documentation for SMTP and job notification details.

QUESTION NO: 21

What's the difference between pushing and pulling code from a CI perspective?

A. Pulling is more efficient.

B. When the source informs the build system of a code change, that's pushing. When the build system asks if there are changes to the source code, that's pulling

C. Pushing uses more resources.

D. When the source informs the build system of a code change, that's pulling. When the build system asks if there are changes to the source code, that's pushing

ANSWER: B

Explanation:

When the source informs the build system of a code change, that's pushing. When the build system asks if there are changes to the source code, that's pulling. This accurately describes the two common CI triggering models. In a push-based model, the source-control platform sends an event notification—commonly a webhook—to Jenkins when a relevant repository event occurs, such as a commit or pull-request update. Jenkins can then promptly schedule the appropriate job or Pipeline. This event-driven approach lets the CI system react to actual repository activity rather than repeatedly querying the source-control system.

In a pull-based model, Jenkins periodically polls the configured source-control repository to discover whether revisions have changed since the last build. Jenkins supports polling through its SCM trigger mechanisms, while integrations such as the GitHub plugin can receive repository webhook notifications and trigger builds from those events. The distinction is therefore defined by which system initiates the notification or check: the source system initiates it for pushing, while the CI system initiates it for pulling. See the Jenkins documentation for SCM polling and the GitHub plugin's webhook-trigger support.

QUESTION NO: 22

What's an example of a cloud-based SCM?

A. GitHub

B. Atlassian BitBucket

C. All of these

D. AWS CodeCommit

ANSWER: C

Explanation:

"All of these" is correct because GitHub, Atlassian Bitbucket, and AWS CodeCommit are services designed to host and manage source-code repositories through cloud-accessible infrastructure. In Jenkins environments, cloud-based source control management systems are commonly used as the central repository location that Jenkins connects to for cloning repositories, polling for changes, or receiving webhooks that trigger builds. GitHub provides hosted Git repositories and collaboration features such as pull requests and code review. Bitbucket Cloud likewise provides Git repository hosting integrated with Atlassian collaboration tools. AWS CodeCommit is AWS's managed, secure source-control service for private Git repositories; although AWS stopped accepting new CodeCommit customers in 2024, it remains an example of a cloud-based SCM service in the context of this question. Each named service therefore meets the defining characteristic: source code is stored and managed remotely by a cloud service rather than solely on a locally administered SCM server. Jenkins supports integration with Git-based repository providers through Git and provider-specific plugins or webhook configurations. See GitHub's repository documentation and AWS CodeCommit documentation.

QUESTION NO: 23

Which of the following are recommended practices when using secrets in a Pipeline? Choose 3 answers

A. Store secrets in Jenkins credentials rather than in the Jenkinsfile.

B. Limit credential scope to the folder or jobs that require it.

C. Use single-quoted Groovy strings when passing shell references to secret environment variables.

D. Print a secret value in the console log to verify that masking is enabled.

E. Commit a credential ID and its secret value together in source control.

ANSWER: A B C

Explanation:

Credentials should be stored in Jenkins and exposed to Pipeline steps only through an appropriate credential binding, such as `withCredentials` or the Declarative `credentials()` helper. This avoids placing passwords, tokens, or private keys directly in a Jenkinsfile or source repository. Credentials should also be scoped as narrowly as practical, for example to the folder containing the jobs that need them.

Shell commands should use single-quoted Groovy strings when referencing secret environment variables, allowing the shell on the agent to expand the variable. This avoids Groovy interpolation of the secret before the command is passed to the shell. Jenkins attempts to mask known credential values in logs, but masking is not a substitute for careful handling of secrets. See the Jenkinsfile credentials documentation and the Credentials Binding Plugin reference.