PassQueen
a.k.a dumpsqueen.com

Certified CloudBees Jenkins Engineer (CCJE)

CloudBees CCJE

Version Demo

Total Demo Questions: 13

Total Premium Questions: 131

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Jenkins Fundamentals	24
Topic 2, Jenkins Administration	53
Topic 3, Jenkins Security	11
Topic 4, Jenkins Pipeline	31
Topic 5, Jenkins Ecosystem	12
Total	131

QUESTION NO: 1

Which are ways to influence the number of build artifacts that can be archived? Choose 2 answers

A. unit build data using the job setting "Discard old builds".

B. Change the order of jobs in the pipeline.

C. Gear the Jenkins cache.

D. Increase the size of the JENKINS.HOME partition/logical volume.

E. Make sure you regularly delete unused views.

ANSWER: A D

Explanation:

The job setting "Discard old builds" directly controls artifact retention. Jenkins can apply a build discarder based on the maximum number of builds or days to retain and, independently, the maximum number of artifacts or artifact-retention days. Once the configured limit is exceeded, Jenkins removes artifacts from older retained runs or removes the runs themselves, reducing the amount and number of archived artifact data held by the controller. This is the standard mechanism for managing artifact retention at the job level.

Increasing the size of the JENKINS.HOME partition or logical volume increases the storage capacity available to Jenkins for build records and archived artifacts when they are stored on the controller filesystem. Greater capacity does not itself create artifacts, but it allows more archived artifacts to remain available before storage pressure requires retention policies or administrative cleanup. Capacity planning for `JENKINS_HOME`, together with explicit build-discarder settings, is therefore an appropriate way to influence how many artifact archives can be retained operationally. See the Jenkins documentation on discarding old builds and the Jenkins home directory.

QUESTION NO: 2

When you want to validate that your software produces the desired behavior for end users, you need to use_____.

A. non regression tests

B. acceptance tests

C. smoke tests

D. functional tests

ANSWER: B

Explanation:

Acceptance tests are used to validate that software delivers the behavior and outcomes expected by end users and business stakeholders. They evaluate the application from the perspective of agreed acceptance criteria: observable scenarios that demonstrate whether a feature solves the intended user need. For example, an acceptance test can confirm that a customer can complete a purchase, receive confirmation, and see the correct order status. This emphasis on user-facing outcomes makes acceptance testing an essential feedback mechanism before software is considered ready for release.

In a Jenkins or CloudBees CI delivery pipeline, acceptance tests are commonly automated and executed after the application has been built and deployed to an appropriate test environment. Pipeline automation makes the validation repeatable, visible, and enforceable, so teams receive fast evidence that a change still meets the required user behavior. Jenkins Pipeline is designed to model these build, test, and delivery stages as code, enabling consistent quality gates throughout delivery. See the Jenkins Pipeline documentation and CloudBees' overview of continuous delivery for context on automating validation in delivery workflows.

QUESTION NO: 3

Which practices help prevent secret exposure when using credentials in a Pipeline? Choose 3 answers

A. Bind credentials only for the steps that require them.

B. Use single-quoted Groovy strings when passing secret environment variables for shell expansion.

C. Avoid printing secrets or complete environment-variable listings in build logs.

D. Put credentials in the Jenkinsfile so that changes can be reviewed in source control.

E. Disable Jenkins security when a Pipeline needs to access a credential.

ANSWER: A B C

Explanation:

Credentials should be bound only around the commands that require them, reducing the duration and scope of secret availability. Shell commands should use shell expansion of environment variables where possible, rather than Groovy interpolation of secrets into command strings. Groovy interpolation can expose sensitive values in process arguments or Pipeline metadata before the command is sent to the agent.

Users should also avoid printing credentials, environment dumps, or commands containing secret values to the build log. Jenkins masks many supported credential values, but masking is not a guarantee that every possible transformation or encoding of a secret is concealed. See Jenkins guidance on handling credentials and the string interpolation documentation.

QUESTION NO: 4

You want to trigger a build when there K an SCM change. Which are reasons you might want to use a post commit trigger instead of polling the SCM frequently?

Choose 2 answers

A. Polling can introduce additional delays.

B. A post-commit trigger build includes all dependencies.

C. Polling may consume unnecessary resources.

D. Poling will not check out the latest revision.

E. A post commit trigger build is more reliable.

ANSWER: A C

Explanation:

“Polling can introduce additional delays.” is correct because SCM polling runs only on the schedule configured for the job. A commit made immediately after a polling interval begins may not be detected until the next interval, so the build start time can be delayed. A post-commit trigger, commonly implemented through an SCM webhook, notifies Jenkins or CloudBees CI when the change is made and can start evaluation and scheduling promptly.

“Polling may consume unnecessary resources.” is also correct. Frequent polling requires Jenkins controllers and SCM integrations to repeatedly contact repositories and determine whether changes have occurred, including during periods with no commits. Across many jobs, repositories, branches, and controllers, these repeated checks create avoidable load on Jenkins and the SCM server. Post-commit notifications are event-driven: the SCM sends a notification only when a relevant change occurs, reducing routine repository queries while providing timely build triggering. Jenkins documents SCM polling as a configurable trigger and supports webhook-style repository notifications through SCM integrations. See the Jenkins Pipeline trigger documentation and the Jenkins Git plugin documentation on push notifications.

QUESTION NO: 5

What does CAP do?

A. Specifies the set of features, feature versions, and feature dependencies that are either verified or trusted, depending on the degree of testing they have received.

B. Runs security metrics toots on the duster to ensure that no installed plugins have security vulnerabilities.

C. Specifies the set of plugins, plugin versions, and plugin dependencies that are either verified or trusted, depending on the degree of testing they have received.

D. Tests all installed plugins to ensure that they are compatible with all other the CloudBees Core or Jenkins release you are running.

E. Specifies the set of apps, app versions, and app dependencies that are either verified or trusted, depending on the degree of testing they have received.

ANSWER: C

Explanation:

CloudBees Assurance Program (CAP) specifies a curated set of Jenkins plugins, plugin versions, and their dependencies that CloudBees has tested and supports at a defined assurance level. CAP is intended to give administrators a predictable, validated plugin ecosystem rather than requiring them to independently determine whether arbitrary plugin combinations are suitable for their CloudBees CI environment. The program's plugin catalog identifies plugins that are either verified or trusted according to the level of testing and support they have received. This helps teams select compatible, maintainable plugins while planning upgrades and operating controllers in production. CAP therefore concerns the Jenkins plugin ecosystem: it is not a security-scanning tool, a process that dynamically tests every plugin installed on a cluster, or a catalog of generic features or applications. CloudBees documents the program and its assurance levels in its plugin management documentation. The associated catalog of supported and assured plugins is available through the CloudBees Plugin Catalog documentation.

QUESTION NO: 6

Which is an advantage of using a replaceable build node?

A. It allows jobs to have dedicated machines tailored to specialized needs.

B. It improves the throughput of master executors.

C. It reduces the impact on productivity if there is an outage of a build node.

D. It decreases the load on the network.

ANSWER: C

Explanation:

"It reduces the impact on productivity if there is an outage of a build node." is correct because replaceable build nodes are designed so that an individual agent is not a unique, irreplaceable dependency. If an agent becomes unavailable because of hardware failure, operating-system issues, maintenance, or an infrastructure outage, it can be removed and replaced with another node that has the required agent configuration, labels, tools, and capacity. Jenkins can then schedule queued builds onto available replacement agents rather than leaving work blocked until the failed machine is repaired.

This approach supports resilient and scalable build infrastructure. In particular, ephemeral or dynamically provisioned agents embody the replaceable-node model: they can be created for workloads and discarded or recreated when they are no longer healthy. The important operational benefit is reduced disruption from the loss of any one build node, since build capacity is treated as a pool rather than tied permanently to a specific machine. Jenkins guidance on distributed builds and agents is available in the Jenkins Scaling documentation, and CloudBees documents the use of dynamically managed agents in its ephemeral agents documentation.

QUESTION NO: 7

Which of the following are true about the structure of a Declarative Pipeline?Choose 2 answers

A. Complex computational logic should be placed outside stage blocks, such as in a shared library.

B. All stages in a Pipeline must execute on the same type of agent.

C. Steps are the logical segmentation of a Pipeline; they contain stages that define actual tasks.

D. Each pipeline must have a global agent specification.

E. Stages are the logical segmentation of a Pipeline; they contain steps that define actual tasks.

ANSWER: A E

Explanation:

Stages are the primary logical units used to organize a Declarative Pipeline into meaningful parts of the delivery process, such as building, testing, and deploying. A `stage` contains a `steps` block, and that block holds the individual Jenkins actions that perform the work. This hierarchy makes Pipeline execution easier to read, visualize, and maintain. Jenkins documents the `stages` section as required in a Declarative Pipeline and describes stages as sequential phases containing the work to be performed.

Complex computational logic should be kept outside the stage definition itself, preferably in a Jenkins Shared Library when it grows beyond a small amount of Pipeline scripting. This keeps the Jenkinsfile declarative and focused on orchestration, while reusable implementation details are versioned, tested, and shared independently. When limited scripted behavior is required within a Declarative Pipeline, Jenkins provides the `script` step; however, its documentation recommends moving non-trivial logic into a Shared Library. See the Jenkins Pipeline Syntax reference and the guidance on the Shared Libraries feature.

QUESTION NO: 8

Which is an element that CANNOT be promoted from a Custom Update Center?

A. Custom-developed plugins.

B. Jenkins core.

C. Build tool installers.

D. Open source plugins.

ANSWER: A

Explanation:

Custom-developed plugins cannot be promoted from a Custom Update Center because promotion is a mechanism for selecting and making available artifacts that already exist in an upstream update center. A custom-developed plugin is an organization's own plugin artifact, so it must first be packaged and published to the custom update center through the appropriate plugin distribution and update-site publication process. Once it is published and represented in that update center's metadata, controllers can discover and install it according to the organization's governance process.

This distinction is important for release management: promotion controls the availability of known upstream content, while internally developed plugins require an explicit publishing workflow that supplies the plugin file, version information, dependencies, and update-center metadata. This lets teams validate, sign where appropriate, and distribute internally built extensions deliberately before they are offered to Jenkins controllers. CloudBees' update-center administration guidance describes managing the update sources available to controllers, and the Jenkins developer documentation explains how custom update sites publish plugin metadata and artifacts. See CloudBees CI update center management and Jenkins custom update sites documentation.

QUESTION NO: 9

Which Jenkins job type automatically discovers branches and pull requests in an SCM repository and creates Pipeline jobs for them?

A. Multibranch Pipeline

B. Freestyle project

C. Folder

D. External monitor job

ANSWER: A

Explanation:

A Multibranch Pipeline is designed to discover repository branches, tags, and change requests through an SCM source. Jenkins creates and manages a separate Pipeline job for each discovered item that contains a Jenkinsfile. When branches are removed or change requests are closed, Jenkins can also apply orphaned-item retention policies to the corresponding jobs.

This model lets a team maintain its Pipeline definition with the source code for each branch while avoiding manual creation of a separate Jenkins job for every branch. See the Jenkins Multibranch Pipeline documentation.

QUESTION NO: 10

You are the administrator of a base Jenkins master with the recommended set of plugins installed. You want to protect the Jenkins master against malicious (or bad) usages of

Groovy methods. You also want your users to be able to share their Pipeline code via Globa Pipeline Libraries housed on a git repository under your company's Github

Organization. Which of the following statements is TRUE?

A. You should configure Global Pipeline Libraries at the Github Organization level: The libraries are running as "untrusted" code, allowing developer to run code on the Groovy sandbox.

B. You should not configure any Global Pipeline Libraries at the folder level: The libraries are running as "trusted" code, allowing all developers to execute privileged methods.

C. You should configure Global Pipeline Libraries at the Pipeline level: The libraries are running as "untrusted" code, allowing developers to run cod on the Groovy sandbox.

D. You should not configure any Global Pipeline Libraries at all: The libraries are running as "trusted" code, allowing all developers to execute privileged methods.

ANSWER: A

Explanation:

You should configure Global Pipeline Libraries at the Github Organization level: The libraries are running as "untrusted" code, allowing developer to run code on the Groovy sandbox. A GitHub Organization item is a Jenkins folder, so a shared library configured in that folder is available to Pipelines beneath it while being scoped to that organization rather than the entire controller. Jenkins treats folder-level shared libraries as untrusted. Consequently, their Groovy code is subject to the Pipeline Groovy sandbox, and calls requiring permissions beyond the sandbox must be explicitly approved by a Jenkins administrator. This provides a practical way for teams to reuse Pipeline functions from a centrally managed Git repository without granting every contributor the unrestricted controller-level privileges that trusted library code has. The library repository should still be protected with appropriate GitHub organization permissions and branch protections, since its contents are executable Pipeline-related code. Jenkins documents that folder-level libraries are always untrusted and are intended for libraries managed by less-trusted users, while trusted global libraries should be limited to highly trusted

administrators. See the Jenkins Shared Libraries documentation and the Jenkins Script Security and in-process approval documentation.

QUESTION NO: 11

The main purpose of the Beekeeper Upgrade Assistant is to_____.

A. Automatically install new CloudBees Core releases as soon as they are available.

B. Provide information on the current status and configuration options of the CloudBees Assurance Program.

C. Enforce and lock a specific configuration in the CloudBees Core instance defined by the administrator.

D. Notify the user when a new CloudBees Core release is available.

ANSWER: D

Explanation:

The main purpose of the Beekeeper Upgrade Assistant is to notify the user when a new CloudBees Core release is available. It is an upgrade-management feature designed to make administrators aware of applicable CloudBees CI/Core releases and to support a controlled, informed upgrade process. Rather than silently changing a production installation, the assistant presents upgrade availability to an administrator, who can then review the release and plan the update according to their organization's maintenance, testing, and change-control practices. This behavior is especially important for Jenkins-based platforms, where an upgrade should be assessed alongside the installed plugin set, operational requirements, and any supported-version guidance.

CloudBees documentation describes upgrade management as an administrator-led activity and provides guidance for keeping CloudBees CI current. The Beekeeper Upgrade Assistant plugin is specifically associated with informing administrators about available CloudBees releases, allowing them to take action deliberately rather than applying an unreviewed upgrade automatically. See the Beekeeper Upgrade Assistant plugin page and the CloudBees CI upgrade guide for upgrade-related documentation and practices.

QUESTION NO: 12

When using SSH to launch an agent, which are required? Choose 2 answers

A. OpenSSL must be instated on the agent.

B. The agent's SSH Public Key must be in the master's authorized keys.

C. The pubic key of the credential that is used for the agent must be installed on the agent.

D. You must log into the agent with SSH and start the agent.

E. SSHD must be installed on the agent.

ANSWER: C E

Explanation:

SSH Build Agents authenticate from the Jenkins controller to the remote agent host, then Jenkins starts the agent process remotely over that SSH connection. Therefore, the remote agent must run an SSH server: **SSHD must be installed on the agent.** The controller needs an SSH endpoint to contact, and the SSH daemon provides that endpoint and accepts the connection on the configured port.

Public-key authentication also requires the matching public key for the private-key credential configured in Jenkins to be authorized for the target account on the agent. Thus, **The pubic key of the credential that is used for the agent must be installed on the agent.** In practice, this means placing the public key in the remote account's `~/.ssh/authorized_keys` file, with suitable ownership and permissions. Jenkins then uses the corresponding private key stored in its SSH credential to establish the session and launch the inbound agent automatically. Jenkins' documentation describes configuring an SSH

credential and host details for an SSH Build Agent, while the SSH protocol documentation explains the `authorized_keys` mechanism for public-key authentication. See the Jenkins node-management documentation and the OpenSSH authorized_keys documentation.

QUESTION NO: 13

In a Freestyle fob, the tests are being executed within a Shell build step and then published vid the "Publish JUnit results" post build action. Which conditions must Alt be met In order for the JUnit publisher to display the job status .is "Unstable''?

Choose 3 answers

A. The "Publish JUnR results" reporter finds the test reports.

B. All build steps are successful.

C. There is at least one SKIPPED test case.

D. All publishers before "Publish JUnit results" assign the SUCCESS status.

E. The overall "stability report amplification factor" is greater than 1.00.

F. There is at least one FAILED test case.

ANSWER: A B F

Explanation:

The required conditions are that the JUnit publisher locates the XML test report files, the shell build step itself completes successfully, and the located reports contain at least one failed test case. The publisher can only evaluate and publish test outcomes when its configured report pattern matches actual JUnit-format report files. Once it reads those reports, a failed test case is the test-result condition that causes the JUnit publisher to set the build result to `UNSTABLE`. This status communicates that Jenkins successfully ran the build and collected test results, but the test suite did not pass completely.

The shell step must also be successful for the overall job to display `UNSTABLE`. Jenkins build results worsen monotonically: if the shell command exits with a nonzero code and has already made the build a failure, publishing failing JUnit tests does not replace that more severe result with `UNSTABLE`. Jenkins documents that the JUnit plugin publishes test reports and marks builds unstable when tests fail; its documentation also describes the report-file pattern used to locate results. See the JUnit plugin documentation and the Jenkins guide to publishing JUnit test results.