DUMPSARENA

Copado Developer Certification Exam

Copado Copado-Developer

Version Demo

Total Demo Questions: 11

Total Premium Questions: 111

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1

Who can see the Advanced button in button on the Commit Changes page?

A. Any user with access to Destructive Changes Git operation.

B. Anyone with the Copado User managed permission set assigned.

C. Any user with Edit User Story Commit Base Branch custom permission assigned.

D. Just the owner of the user story.

ANSWER: C

Explanation:

Any user with Edit User Story Commit Base Branch custom permission assigned can see the Advanced button on the Commit Changes page. This button exposes advanced commit settings, including the ability to control the base branch used for a User Story commit. Because changing a commit's base branch can affect the branch relationship and the destination of metadata changes, Copado protects this capability with a dedicated custom permission rather than making it generally available to all users who can commit changes.

An administrator grants this access by assigning the custom permission through an appropriate permission set or permission-set group. The user does not need to be the User Story owner; visibility of this advanced control is determined by possession of the specific custom permission. This permission-based design supports least-privilege access, allowing teams to reserve branch-level commit configuration for release managers, technical leads, or other users responsible for Git workflow governance. See Copado's documentation for its Commit Changes process and Salesforce guidance on custom permissions.

QUESTION NO: 2

Dave has been assigned the task of deleting a custom object that has not been used in

production for a long time. After committing the destructive changes, he notices that no

feature branch has been created. What could be the reason?

A. The base branch was set to master branch, and the component was not yet in production.

B. He did not have rights to delete the component in the source org.

C. The Re-Create Feature Branch checkbox was not flagged.

D. While adding a row, Dave misspelled the metadata API name of the component he wanted to delete in the metadata grid.

ANSWER: D

Explanation:

While adding a row, Dave misspelled the metadata API name of the component he wanted to delete in the metadata grid. is correct because Copado identifies destructive-change items through their exact Salesforce metadata type and API name, rather than through the object's display label. The metadata-grid entry must therefore match the component stored in source control precisely. If the API name is misspelled, Copado cannot resolve a valid metadata component for the destructive change. Consequently, there is no valid change to commit into the generated source-control work, and the expected feature-branch creation does not occur.

For a custom object, the entered value must use the object's actual API name, including any namespace or custom-object suffix where applicable. Before committing, Dave should confirm the API name in Salesforce Setup or in the repository and correct the destructive-change row. This ensures that Copado can include the intended deletion in the deployment package and create the associated branch workflow. Salesforce's Metadata API documentation describes destructive changes as metadata components identified by metadata type and full component name: Salesforce Metadata API: Deleting Components. Copado's documentation portal provides further product guidance for deployments and metadata operations: Copado Documentation.

QUESTION NO: 3

A team needs to deploy records using a data template, but a target-org validation rule can block the records during deployment. Which actions can help the team move the data deployment forward? Select all that apply!

A. Enable the Bypass Validation Rules checkbox in the data template.

B. Use a Manual Task step to coordinate temporary deactivation and reactivation of the validation rule.

C. Add the validation rule to the repository .gitignore file.

D. Change the promotion branch to the production branch before running the data deployment.

ANSWER: A B

Explanation:

Enabling *Bypass Validation Rules* in the data template can allow the data deployment to proceed without target-org validation rules preventing the record operation. This is appropriate only when the team has determined that bypassing the rules is acceptable for the deployment.

A Manual Task can also be used to coordinate temporary deactivation of the validation rule before the data deployment and reactivation after it is complete. This approach provides an auditable control point when validation logic must remain enabled for normal business operations. See Copado Documentation and Salesforce Validation Rules documentation.

QUESTION NO: 4

Rob would like to run Apex tests for all Apex metadata coming into the UAT environment and a

compliance check for profiles and permission sets coming into the same environment. How can he

accomplish this?

A. He needs to set up 3 connection behaviors in UAT with 1 quality gate each, 1 for Apex tests, 1 for the compliance check for permission sets and 1 for the compliance check for profiles.

B. He needs to set up 1 connection behavior in UAT with 2 quality gates, an Apex test with Validation for the Apex elements and a Compliance Check both for profiles and for permission sets.

C. He needs to set up 2 connection behaviors in UAT, one for the Apex tests and another one for the compliance check.

D. He needs to set up 1 connection behavior in DEV with 2 quality gates, an Apex test with Validation for the Apex elements and a Compliance Check both for profiles and for permission sets.

ANSWER: B

Explanation:

He needs to set up 1 connection behavior in UAT with 2 quality gates, an Apex test with Validation for the Apex elements and a Compliance Check both for profiles and for permission sets. A connection behavior is configured on the target environment connection, so placing it on UAT ensures the checks are evaluated whenever a deployment is promoted into UAT. Copado permits a connection behavior to include multiple quality gates, allowing the required controls to be grouped under the same inbound deployment behavior rather than creating separate behaviors for each check.

The Apex Test quality gate can be configured with validation scoped to the Apex metadata in the deployment. This verifies that the relevant Apex changes meet the organization's test requirements before the deployment is accepted. The Compliance Check quality gate can be configured to assess the requested metadata categories together, including both profiles and permission sets, against the applicable compliance configuration. Combining these two gates in one UAT connection behavior gives Rob a single, consistent deployment policy: Apex-related changes are tested and access-control metadata is checked for compliance as part of the same promotion. Copado's quality-gate documentation is available at Copado Documentation, and Salesforce's Apex testing guidance is available at Apex Testing.

QUESTION NO: 5

You would like to create a YAML file to remove all references to Field Level Security from managed

fields in profiles. Which value should you use in the replace_values parameter?

A. packagePrefix__.*?.

B. packagePrefix__fieldName1, packagePrefix__fieldName2,

C. packagePrefix__fieldName3 and so on.
{!Environment.classes/packagePrefix__.*}.

D. This is not possible.

ANSWER: A B

Explanation:

`packagePrefix__.*?` is the scalable value for this requirement because it is a regular-expression pattern that matches field API names beginning with the managed-package namespace prefix. Managed-package custom fields have names in the `namespace__FieldName__c` form, so using the prefix followed by a wildcard lets the YAML replacement process target every matching managed field without requiring maintenance whenever the package adds or changes fields. This is particularly useful for Profile metadata because field-permission entries refer to fields by API name, and those entries can prevent deployment when the target org does not have the corresponding managed package or package version installed.

An explicit sequence such as `packagePrefix__fieldName1, packagePrefix__fieldName2,` can also be used when the intended cleanup is limited to a known, enumerated set of managed fields. In that approach, the list must be completed with every applicable namespace-qualified field name. The wildcard pattern is generally preferable when the objective is to remove *all* managed-field FLS references. Salesforce documents Profile metadata and its field-permission definitions in the Profile metadata reference. For the namespace convention used by managed packages, see Salesforce's Namespaces and Prefixes documentation.

QUESTION NO: 6

Which situations can cause Copado to apply a test level that is more restrictive than the Promotion Test Level selected on a user story? Select all that apply!

A. The deployment contains Apex components and is promoted to production with NoTestRun selected.

B. The destination Environment has the Run all local tests checkbox enabled.

C. The source Environment has a lower Minimum Apex Code Coverage value than the destination Environment.

D. The feature branch contains more than one Git commit.

ANSWER: A B

Explanation:

A production deployment containing Apex cannot use NoTestRun because Salesforce requires testing for Apex-containing production deployments. Copado must honor that platform requirement and applies the required production-safe test behavior.

Also, when the destination Environment has the *Run all local tests* setting enabled, Copado enforces local test execution for deployments targeting that environment. This destination-environment control takes precedence over a less restrictive User Story test-level setting.

See Copado Documentation and Salesforce Metadata API deployment test-level documentation.

QUESTION NO: 7

What test level is applied to the deployment when you check the Promote & Deploy checkbox?

A. The test level specified in the destination Org Credential record.

B. The test level specified in the Promotion Test Level field on the User Story record.

C. The test level specified in the Test Level field of the destination environment.

D. The test level specified in the source environment.

ANSWER: B

Explanation:

When *Promote & Deploy* is selected, Copado uses the test level configured in the *Promotion Test Level* field on the User Story record. This setting is specifically intended to control the validation and test-execution behavior for the promotion's deployment operation. It lets a team define the required testing policy as part of the work item being promoted, so the deployment follows the test strategy associated with that User Story rather than relying on a general environment-level default.

In Salesforce deployments, the selected test level determines whether tests are skipped where permitted, local tests are run, specified tests are run, or all tests are run. Copado passes the applicable deployment configuration into its promotion process, and the User Story's Promotion Test Level is the relevant configuration when the promotion is immediately deployed through *Promote & Deploy*. This supports consistent, traceable deployment governance because the test requirement is visible alongside the User Story and can be reviewed before the promotion proceeds. For Salesforce's definitions of deployment test levels, see Salesforce Metadata API deployment documentation. Copado's product documentation is available through the Copado documentation portal.

QUESTION NO: 8

Within the same sprint, Debbie added, a defaultPhone method and Dave added a defaultUrl method, both to the account operations class. Since, Debbie and Dave are working in their

individual sandboxes, after promoting the second user story, a Git conflict arose. They leamed up to solve the conflict using Copado's online conflict resolution feature. What

happens when both user stories are mass back-promoted?

A. The defaultPhone method overwrites the other method.

B. Copado uses semantic conflict resolution.

C. The defaultUrl method overwrites the other method.

D. Copado uses the conflict solution that was previously created.

ANSWER: D

Explanation:

Copado uses the conflict solution that was previously created. When the team resolves the Git conflict through Copado's online conflict-resolution capability, that resolution becomes the selected, merged version for the affected metadata component. In this case, the Account operations class contains independently added methods from the two user stories, and the saved resolution preserves the intended combined content rather than treating one developer's version as automatically authoritative.

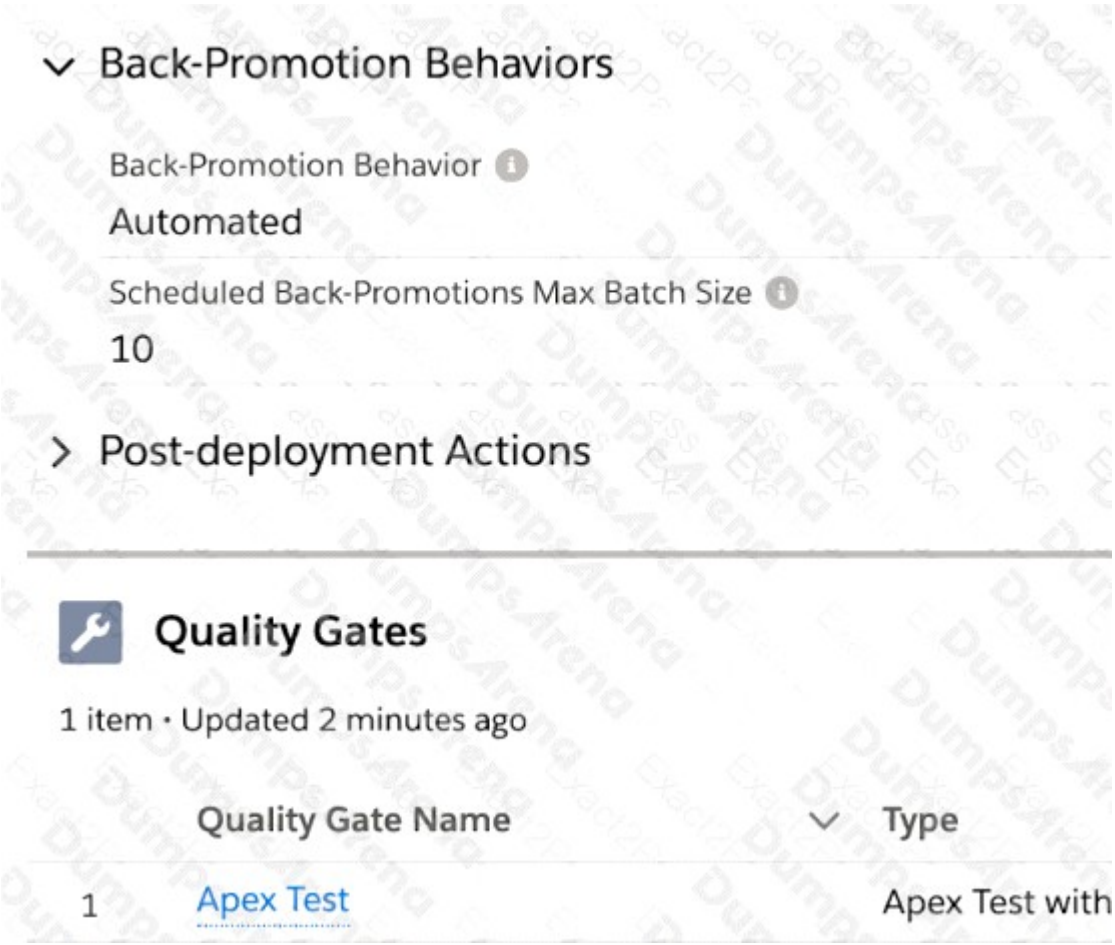
During mass back-promotion, Copado applies the established conflict-resolution result when propagating the promoted user stories back through the relevant environments. This gives the team a consistent source-controlled version of the class and avoids requiring the same manual merge decision again merely because the changes are being back-promoted together.

The key point is that conflict handling is based on the resolution already created for the Git conflict, so the class state used in the promotion and back-promotion flow reflects that accepted solution.

Copado's documentation on resolving merge conflicts and promoting user stories provides the relevant workflow context: Resolve Conflicts and Back Promote User Stories.

QUESTION NO: 9

Rob, has set up the following connection behavior in UAT:



Dave, the lead developer, has committed an Apex class in dev1 and is going to deploy it to UAT.

What will happen once the user story is successfully deployed to UAT?

A. The user story will be displayed in the user stories behind count on the Pipeline page, and Dave will have to manually back-promote it by clicking on Back Promote & Deploy.

B. Copado will re-run the quality checks and automatically back-promote the user story to the other lower environments connected with UAT only if the relevant Apex tests pass. (This might be correct, still need to check)

C. Copado will send Dave a notification to let him know that the user story is ready to be backpromoted to other lower environments connected with UAT.

D. Copado will automatically back-promote the user story to other lower environments connected with UAT.

ANSWER: D

Explanation:

Copado will automatically back-promote the user story to other lower environments connected with UAT. The configured connection behavior on the UAT environment controls what Copado does after a deployment to that environment completes

successfully. With automatic back-promotion enabled, Copado identifies the lower environments that are connected to UAT and propagates the user story's promoted changes back to them. This keeps development environments synchronized with the version of the metadata that has been successfully deployed and validated in UAT.

Back-promotion is particularly useful in a pipeline where development work may occur in multiple lower environments. Once UAT has received the user story, automatically returning the deployed changes to its connected lower environments reduces configuration drift and avoids requiring a developer or release manager to perform the synchronization as a separate manual activity. The behavior occurs as part of the successful promotion flow; it is not merely a notification or an item added to a manual back-promotion queue. Copado's official documentation describes its CI/CD pipeline and environment-promotion capabilities at Copado Documentation and provides product guidance through Copado Success Community.

QUESTION NO: 10

In a sandbox, you have made some changes in the System Admin profile. When you try to deploy the

profile, you are getting the error "Unknown User Permission: ManageSandboxes." What can you do in

order to move your changes forward? Select all that apply!

A. Create a YAML file to exclude the ManageSandboxes permission, add it to the pipeline and redeploy.

B. In the Git Promotion deployment step, use the Find and Replace feature to exclude the ManageSandboxes permission and redeploy.

C. Manually remove the ManageSandboxes permission from the destination branch and redeploy.

D. Update the .gitignore file in the uat branch to exclude the user permissions introduced in the higher API version and redeploy.

ANSWER: A B

Explanation:

Creating a YAML file to exclude the ManageSandboxes permission, adding it to the pipeline, and redeploying is appropriate because it makes the handling of the unsupported metadata explicit, repeatable, and source-controlled. The deployment can continue while preserving the rest of the intended System Admin profile changes. This is particularly useful when the permission was introduced in a Salesforce API version or release that is not supported by the target organization or by the metadata API version used for deployment.

Using the Find and Replace feature in the Git Promotion deployment step to exclude the ManageSandboxes permission is also a valid way to move the deployment forward. It applies a targeted transformation during promotion, allowing Copado to remove the incompatible profile permission from the deployment payload without requiring the team to discard other profile updates. Both approaches address the deployment-time metadata incompatibility at the pipeline level and allow the promotion process to proceed in a controlled manner. Salesforce profile metadata includes user-permission settings, and deployment compatibility depends on the receiving org's supported metadata and feature set. See the Salesforce Profile metadata reference and Copado documentation for deployment configuration guidance.

QUESTION NO: 11

To which of the following components does Copado apply the resolution strategy "branch A wins over branch B"? Select all that apply!

A. Apex classes

B. Custom objects

C. Profiles

D. Layouts

ANSWER: A D

Explanation:

Copado applies the “branch A wins over branch B” resolution strategy to **Apex classes** and **Layouts**. In a Copado merge operation, branch A represents the source side whose selected changes take precedence when Copado encounters conflicts for component types governed by this rule. For Apex classes, this provides a deterministic result for source-code conflicts: the class version from branch A is retained rather than attempting to combine conflicting code automatically. For Layouts, Copado likewise prioritizes branch A when overlapping layout metadata cannot be reconciled under its merge-resolution handling. This predictable behavior is important because both source code and page-layout metadata can contain mutually dependent edits where an automatic field-by-field choice could produce an inconsistent or invalid result. Teams should therefore review branch-A changes before completing the merge and validate the resulting deployment in the target environment. Copado’s documentation provides its CI/CD and merge-management guidance at Copado Documentation. For background on the Salesforce metadata types involved in these deployments, see Salesforce’s Metadata Coverage Report.