DUMPSARENA

Adobe Commerce Developer Expert

Adobe AD0-E725

Version Demo

Total Demo Questions: 5

Total Premium Questions: 57

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture	37
Topic 2, EAV/Database	3
Topic 3, Layout/UI	4
Topic 4, Checkout	3
Topic 5, Sales	1
Topic 6, Catalog	4
Topic 7, Admin	5
Total	57

QUESTION NO: 1

A client wants to calculate tax differently when an order is billed for a range of particular postcodes. They want to implement their own system instead of using a third-party system. The Developer is asked to assist in this task by creating a conditional webhook.

What must be done to achieve this?

A. Create an after plugin on the webhook class and check for a list of postcodes at the time of processing and return true for certain postcodes.

B. Create a webhooks.xml file and add the rules wanted in the form of a rule node element using the regex operator to catch a range of postcodes.

C. Implement a new webhook with a list of postcodes and link it to the original webhook, then advise the client to monitor for the new webhook instead.

ANSWER: B

Explanation:

Create a webhooks.xml file and add the rules wanted in the form of a rule node element using the regex operator to catch a range of postcodes. Adobe Commerce webhooks are declared in a module's `etc/webhooks.xml` configuration file, which lets a developer define the webhook endpoint and the conditions that determine whether Commerce invokes it. A rule can inspect data available in the event payload, such as the billing address postcode, before the outbound request is sent. Using the `regex` operator provides an appropriate declarative way to match a defined postcode pattern or range without embedding postcode-selection logic in the webhook-processing flow.

When the configured billing postcode satisfies the regular expression, the conditional webhook can call the client's tax-calculation service. That service can return the tax information required by the webhook contract, allowing Commerce to apply the client's custom tax behavior for the matching addresses. Keeping the condition in declarative webhook configuration makes the behavior module-managed, reviewable, and aligned with Adobe Commerce's webhook extension mechanism. Adobe's webhook configuration guidance is available in the Adobe Commerce Webhooks configuration documentation; the broader implementation model is described in the Adobe Commerce Webhooks overview.

QUESTION NO: 2

A Developer has created a service contract interface and a concrete implementation for a custom order-export operation. The operation must be exposed through a new REST endpoint without modifying Commerce core code.

Which two configuration steps are required? (Choose two.)

A. Declare the REST route, service interface, and service method in etc/webapi.xml.

B. Configure a preference from the service interface to its implementation in etc/di.xml.

C. Declare the REST route as a field in etc/schema.graphqls.

D. Declare the service interface as an extension attribute in etc/extension_attributes.xml.

E. Add the endpoint URL as a router in routes.xml.

ANSWER: A B

Explanation:

The REST route must be declared in `etc/webapi.xml`. The route specifies the URL, HTTP method, service interface, service method, and authorization resource used by the Web API framework.

The service interface must resolve to its concrete implementation through dependency injection, normally by declaring a preference in `etc/di.xml`. Extension attributes extend API data interfaces, but do not register a REST route. A GraphQL schema file does not expose a REST endpoint.

QUESTION NO: 3

A Developer adds an observer for a storefront-only event. The observer must not execute when the same event is dispatched from the Admin area, REST API, or cron process.

Where should the Developer declare the observer?

A. In `etc/events.xml`

B. In `etc/frontend/events.xml`

C. In `etc/adminhtml/events.xml`

ANSWER: B

Explanation:

The observer should be declared in `etc/frontend/events.xml`. Area-specific event configuration is loaded only when that application area is active, which scopes the observer to storefront execution.

Declaring the observer in `etc/events.xml` makes it global and can cause it to execute in multiple application areas. Declaring it in `etc/adminhtml/events.xml` would instead scope it to the Admin area.

QUESTION NO: 4

A module must execute an observer only for storefront requests when the `customer_login` event is dispatched.

Which two actions should the Developer take? (Choose two.)

A. Create `etc/frontend/events.xml`.

B. Configure an observer for the `customer_login` event in that file.

C. Configure the observer in global `etc/events.xml`.

D. Add the observer class as a job in `crontab.xml`.

ANSWER: A B

Explanation:

The Developer should create `etc/frontend/events.xml` so the event configuration is loaded only in the storefront area. Within that file, the Developer must add an `event` node named `customer_login` and configure an `observer` node with the observer class.

Adding the observer in global `etc/events.xml` would make it available in every area where the event is dispatched. A cron job definition is unrelated to synchronous event-observer execution.

QUESTION NO: 5

A Developer is working on an Adobe Commerce Cloud project and needs to upgrade a Redis service on production to the latest version for improved performance and security.

Which step should the Developer follow to upgrade the installed service version in Adobe Commerce Cloud?

A. Submit a ticket to Adobe support.

B. Edit the services.yaml file to specify the new service version.

C. Use the magento-cloud service:update command.

ANSWER: B

Explanation:

Edit the services.yaml file to specify the new service version. Adobe Commerce on cloud infrastructure defines the services available to an environment, including Redis, in the `.magento/services.yaml` configuration file. Each service definition declares its type and version, so changing the Redis version in that file is the configuration change that requests the platform to provision the desired supported service version.

The developer should make the version change in source control, validate that the selected version is supported for the project's Adobe Commerce and cloud-infrastructure configuration, and deploy the change through the normal environment deployment workflow. For production, this ordinarily means merging or pushing the tested configuration change through the project's deployment process. The deployment applies the updated service configuration to the target environment; application configuration should also continue to reference the Redis relationship exposed by the platform.

Adobe's service configuration documentation describes `.magento/services.yaml` as the location for declaring service versions and configuration. The Redis service documentation provides the Redis-specific configuration context and supported usage on cloud infrastructure. See Configure services with services.yaml and Redis service.