DUMPSARENA

Adobe Commerce Developer Expert

Adobe AD0-E716

Version Demo

Total Demo Questions: 9

Total Premium Questions: 93

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture	63
Topic 2, Catalog	9
Topic 3, Sales	7
Topic 4, Checkout	1
Topic 5, API	13
Total	93

QUESTION NO: 1

An Adobe Commerce developer added a new API method to search and retrieve a list of Posts for a custom Blog functionality. This is the content of the module ' s etc/webapi.xml file:

```
<?xml version="1.0" ?>
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Webapi:etc/webapi.xsd">
    <route url="/V1/myvendor-blog/post/search" method="GET">
        <service class="MyVendor\Blog\Api\PostRepositoryInterface" method="getList"/>
        <resources>
            <resource ref="MyVendor_Blog::Post_view"/>
        </resources>
    </route>
</routes>
```

The new code has been deployed to production and the merchant is using https: //merchant. domain. com/swagger to review the new endpoint, but it is not visible in swagger.

What would be a reason for this?

A. The webapi.xml file should be moved into the etc/webapi_rest/webapi.xml file.

B. Since the new endpoint is not anonymous, the merchant needs to enter a valid integration token in swagger in order to see the new method.

C. The @return annotation is missing in the MyVendor\Blog\Api\PostRepositoryInterface class.

ANSWER: C

Explanation:

The `@return` annotation is missing in the `MyVendor\Blog\Api\PostRepositoryInterface` class is correct. Adobe Commerce uses the API service contract—particularly its PHPDoc annotations—to build the metadata used by the Web API framework and Swagger/OpenAPI documentation. For a repository method such as `getList`, the interface should declare the returned service-contract type, commonly a custom search-results interface, in an `@return` PHPDoc annotation. For example, `@return \MyVendor\Blog\Api\Data\PostSearchResultsInterface` tells Commerce the response data type and lets it derive the API schema for the operation. Without return-type metadata, Commerce cannot reliably describe the service response for the generated API documentation, which can prevent the operation from appearing properly in Swagger. The developer should add the appropriate `@return` declaration to the interface method, deploy the updated code, and clear generated metadata/cache as appropriate for the deployment process. Adobe's guidance on defining service interfaces and exposing them as web APIs describes the importance of service-contract method annotations and web API route configuration: Web API services and Web APIs overview.

QUESTION NO: 2

How would a developer turn on outgoing emails on an Adobe Commerce Cloud Staging environment?

A. ece-tools enable_smtp true

B. magento-cloud environment:info -p -e enable_smtp true

C. In the Cloud Console, select Configure environment, turn Outgoing emails On, and save the configuration.

ANSWER: C

Explanation:

Use the Adobe Commerce Cloud Console to open the Staging environment's configuration, turn **Outgoing emails** on, and save the change. Adobe Commerce Cloud controls outbound email separately for each environment because non-production environments commonly suppress email delivery to prevent test transactions, account notifications, and other development activity from reaching real recipients. The environment configuration in the Cloud Console provides the supported administrative control for enabling this capability on Staging. A user with the necessary project access selects the Staging environment, chooses **Configure environment**, changes the outgoing-email setting to the enabled state, and saves

the configuration. Once enabled, the application can send email from that environment using the Cloud platform's configured mail service; application-level email configuration and recipient behavior still apply. This approach is an environment setting rather than an ece-tools command or a Magento application CLI setting. Adobe documents environment administration through the project and environment views in the Cloud Console; see the Adobe Commerce Cloud project overview and the environment management documentation.

QUESTION NO: 3

A developer is working on an Adobe Commerce Cloud project and wants to get connection data for the environment ' s deployed services. The developer has all of the necessary permissions to do this.

Which two options would the developer take to get the connection credentials? (Choose Two.)

A. Run the magento-cloud relationships CLI command.

B. Get the data from the Project Web Interface dedicated section.

C. Execute ece-tools env:config:show services command.

D. Connect to the server via SSH and read the $_ENV['MAGENTO_CLOUD_RELATIONSHIPS'] variable.

ANSWER: A D

Explanation:

Running the `magento-cloud relationships` CLI command is a supported way to retrieve the relationship information for services deployed in a selected Adobe Commerce Cloud environment. The returned data contains the connection details made available to the application, including items such as host, port, path or database name, username, and password where applicable. The command can be scoped to the target environment when needed.

Connecting through SSH and reading `$_ENV['MAGENTO_CLOUD_RELATIONSHIPS']` is also valid because Adobe Commerce Cloud exposes service relationship data to the application through this environment variable. Its value is Base64-encoded JSON, so the developer decodes and parses it to obtain the individual service credentials and endpoints. In an SSH session, the corresponding shell variable can likewise be decoded for inspection. This approach is particularly useful when examining precisely the relationship payload available to the deployed application. Adobe documents both the relationships command and the environment relationship variable as mechanisms for accessing service connection information. See Adobe Commerce Cloud CLI reference: relationships and Adobe Commerce Cloud service relationships documentation.

QUESTION NO: 4

A developer is reviewing possible plugin targets before designing an extension. The implementation requires a plugin that must reliably execute.

Which two target declarations prevent Adobe Commerce from intercepting the method? (Choose Two.)

A. A public method declared as `final`.

B. A protected non-final method.

C. A public non-final method on an interceptable concrete class.

D. A public non-final method declared by a service interface.

ANSWER: A B

Explanation:

A final method cannot be overridden by the generated interceptor, and a protected method is not eligible for plugin interception because plugins apply to public methods. Either target therefore requires a different extension point.

A public non-final method is generally interceptable. Targeting an interface in a plugin declaration can also be valid when the concrete implementation is resolved through dependency injection and otherwise eligible for interception.

QUESTION NO: 5

An Adobe Commerce developer creates an Admin controller action that allows authorized users to export records from a custom entity. Users without the custom permission must be denied even if they enter the controller URL directly.

Which two actions are required? (Choose Two.)

A. Add the custom permission as a resource in `etc/webapi.xml`.

B. Declare the custom ACL resource in `etc/acl.xml`.

C. Hide the export menu item for all roles except Administrators.

D. Set the controller's `ADMIN_RESOURCE` constant to the custom ACL resource ID.

ANSWER: B D

Explanation:

The module must declare a custom ACL resource in `etc/acl.xml`, normally beneath an appropriate Administration resource. The controller must define its `ADMIN_RESOURCE` constant with that resource ID so the backend authorization framework checks it before the action executes.

Adding a menu item alone only controls navigation visibility and does not protect direct URL access. A web API route resource applies to REST and SOAP routes, not to an Admin controller action.

QUESTION NO: 6

An Adobe Commerce developer has created a module that adds a product attribute to all product types via a Data Patch- According to best practices, how would the developer ensure this product attribute is removed in the event that the module is uninstalled at a later date?

A. Add an Uninstall.php file extending \Magento\Framework\Setup\UninstallInterface to the module's Setup directory and implement the uninstall method.

B. Add instructions to the module 's README.md file instructing merchants and developers that they must manually remove this attribute if they want to uninstall the module.

C. Make the Data Patch implement \Magento\Framework\Setup\Patch\PatchRevertableInterface and implement the revert method to remove the product attribute.

ANSWER: C

Explanation:

Make the Data Patch implement `\Magento\Framework\Setup\Patch\PatchRevertableInterface` and implement the `revert()` method to remove the product attribute. This is the purpose-built lifecycle mechanism for making changes performed by a data patch reversible. The patch should continue to implement `DataPatchInterface` for its installation behavior, while `PatchRevertableInterface` supplies the corresponding uninstall behavior. During module uninstallation, Adobe Commerce can call the patch's `revert()` method, allowing the module to cleanly undo the data it introduced.

For an EAV product attribute, `revert()` should create an EAV setup instance using the module data setup dependency and call `removeAttribute()` with the product entity type and the attribute code originally used in `apply()`. Keeping the creation and removal logic together in the same patch makes the module self-contained and ensures that its database changes are managed consistently through its lifecycle. Adobe's data-patch documentation specifically notes that a patch can implement `PatchRevertableInterface` and use `revert()` to reverse its operations. See Adobe Commerce: Data and schema patches and the PatchRevertableInterface source.

QUESTION NO: 7

An Adobe Commerce developer is trying to create a custom table using declarative schema, but is unable to do so.

What are two errors in the snippet above? (Choose two.)

A. Column (roll_no) does not have index. It is needed since attribute identity is set to true.

B. Column (entity_id) does not have index. It is needed since attribute identity is set to false.

C. Column (student_name) does not have attribute length.

D. null is not a valid value for column (roll_no).

ANSWER: A C

Explanation:

Column (roll_no) does not have index. It is needed since attribute identity is set to true. An identity column represents an auto-incrementing database value. Auto-increment columns must be part of a key; in practice, the declarative schema should define the appropriate primary-key or index constraint for that column. This lets the database enforce uniqueness and efficiently maintain the next generated value. In Adobe Commerce declarative schema, column definitions describe attributes such as type and identity, while constraints and indexes are declared separately in the table schema.

Column (student_name) does not have attribute length. A character-based field intended to hold a bounded value, such as a student name, must declare its supported maximum length in the declarative schema. Supplying the length allows Adobe Commerce to generate the intended database column definition consistently during schema installation or upgrade. For example, a name field is commonly defined as a varchar-style column with a length such as 255. Adobe documents declarative schema column attributes and separate constraints/index definitions at Adobe Commerce declarative schema configuration. Database requirements for auto-increment columns and keys are also described in the MySQL auto-increment documentation.

QUESTION NO: 8

A merchant of an Adobe Commerce Cloud project wants to setup one of their websites using a subdomain. The merchant is considering the domain to be set as secondstore.example.com.

In addition to editing the magento-vars.php file, and apply a domain check and set $_SERVER["MAGE_RUN_CODE"] and $_SERVER["MAGE_RUN_TYPE"].

What file is required to perform this action?

A. Configure secondstore.example.com subdomain route in NGINX virtual-host configuration file.

B. Configure secondstore.example.com subdomain route in .magento/services.yaml.

C. Configure secondstore.example.com subdomain route in .magento/routes.yaml.

ANSWER: C

Explanation:

Configure secondstore.example.com subdomain route in .magento/routes.yaml. Adobe Commerce on cloud infrastructure uses the `.magento/routes.yaml` configuration file to declare which incoming HTTP hostnames are accepted and how each hostname is routed to the application upstream. Therefore, adding a route for `secondstore.example.com` is required for requests to that subdomain to reach the deployed Commerce application. The route maps the public domain to the application container, typically using an upstream such as `"mymagento:http"` or the applicable application name and port defined for the project.

The runtime domain check in `magento-vars.php` has a separate but complementary role. Once the request reaches Magento, that logic can inspect the host and assign `MAGE_RUN_CODE` and `MAGE_RUN_TYPE`, allowing Magento to select the intended website or store configuration. A route must exist first so that Adobe Commerce cloud infrastructure directs the

subdomain request to the application. After committing the route configuration and deploying it, the merchant must also ensure the domain's DNS record and Adobe Commerce Cloud domain/TLS configuration are in place for the hostname to resolve securely. Adobe documents route declarations and upstream mappings in its routes.yaml configuration guide, and documents multi-site implementation considerations in the multiple websites or stores guide.

QUESTION NO: 9

An Adobe Commerce developer is writing an integration test. They checked some Integration Tests for Magento core modules for reference and noticed that they use data fixtures initialized by adding annotations to test classes. For example:

The developer wants to add their own fixture to test a MyVendor_MyModule they created. Which steps will make this possible?

A. Create my_fixture.php in MyVendor/MyModule/Test/Integration/_files/ and add the annotation @magentoDataFixture MyVendor_MyModule::Test/Integration/_files/my_fixture.php to the test method.

B. 2. Add the following annotation to the test method:

C. 2. Add the following annotation to the test method:

D. Option A

E. Option B

F. Option C

ANSWER: A

Explanation:

Create the fixture PHP script in the module's integration-test fixture directory, `MyVendor/MyModule/Test/Integration/_files/my_fixture.php`, and reference it from the integration test with `@magentoDataFixture MyVendor_MyModule::Test/Integration/_files/my_fixture.php`. The `Module_Name::` notation lets the integration-test framework resolve the path from the registered module directory rather than relying on a project-specific absolute path. This keeps the fixture with the module that owns both the test and its required setup data, so the module remains portable across Adobe Commerce installations. The fixture script is executed before the annotated test and can use the object manager or application APIs to create the required entities and state. Adobe Commerce integration tests support test isolation and fixture annotations for repeatable test setup; placing fixture files under `Test/Integration/_files` is the module-oriented convention used for this purpose. See Adobe's integration testing guide and the Adobe Commerce module development documentation for module structure and testing guidance.