DUMPSBOSS.

Adobe Commerce Developer Professional

Adobe AD0-E724

Version Demo

Total Demo Questions: 13

Total Premium Questions: 136

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture	70
Topic 2, Catalog	17
Topic 3, Sales	5
Topic 4, Checkout	5
Topic 5, Customer	3
Topic 6, Admin	24
Topic 7, APIs	12
Total	136

QUESTION NO: 1

Which two actions will the developer need to take to translate strings added in JS files? (Choose two.)

A. Add 'mage/translate' as a RequireJS dependency.

B. Use $trans('').

C. Use $.mage._('').

D. Use $.mage.__('') to translate the string.

ANSWER: A D

Explanation:

To make a JavaScript string translatable, the JavaScript module must load `mage/translate` as a RequireJS dependency. This module extends the jQuery object with Magento's client-side translation function and ensures that the locale-specific translation dictionary is available when the module executes. The dependency is commonly declared alongside `jquery` in the module's `define()` dependency array.

The string itself must then be passed to `$.mage.__('<string>')`. The double-underscore method is the JavaScript equivalent of Magento's translation helper: it uses the original phrase as the lookup key and returns the matching phrase for the active storefront locale when one is available. For example, a module can call `$.mage.__('Add to Cart')` when building user-facing UI text. The source phrase can subsequently be supplied in the module or theme CSV translation dictionary, allowing static-content deployment and storefront localization to apply the appropriate translated value. Adobe's frontend translation guidance and the implementation of the translation module are available in the Adobe Commerce frontend translation documentation and the mage/translate source.

QUESTION NO: 2

An Adobe Commerce developer is working on a Magento 2 instance which contains a B2C and a B2B website, each of which contains 3 different store views for English, Welsh, and French language users. The developer is tasked with adding a link between the B2C and B2B websites using a generic link template which is used throughout the sites, but wants these links to display in English regardless of the store view.

The developer creates a custom block for use with this template, before rendering sets the translate locale and begins environment emulation using the following code:

```
/** @var $this->_translate \Magento\Framework\TranslateInterface */
$this->_translate->setLocale($newLocaleCode);

/** @var $this->_emulation \Magento\Store\Model\App\Emulation */
$this->_emulation->startEnvironmentEmulation($storeId, \Magento\Framework\App\Area::AREA_FRONTEND);
```

They find that the template text is still being translated into each stores language. Why does this occur?

A. true), to override and lock the locale of the emulated store. If this is set and locked initially then the environment emulation will not be able to override this.

B. startEnvironmentEmulation() resets the translation locale to that of the emulated store, which overrides the locale the developer set when setLocale() is called before startEnvironmentEmulation(), as displayed above.

C. setLocale() does not change the translation locale after it has been initially set; the $this->_translate->emulate($newLocaleCode) method exists to temporarily modify this by pushing the new locale to the top of the current emulatedLocales stack.

ANSWER: B

Explanation:

`startEnvironmentEmulation()` resets the translation locale to that configured for the emulated store, overriding the locale that was set before environment emulation began. Environment emulation is designed to establish a consistent store-

specific execution context: it switches the current store and applies that store's configuration, design settings, and locale-related state. Consequently, when the selected B2C or B2B store view uses Welsh or French, starting its emulation loads that store view's locale for translation processing. The earlier call to set the translation locale to English therefore does not remain in effect when the template is rendered.

For template text that must remain English while operating in a particular store context, the translation component should temporarily emulate the English locale immediately around rendering, then revert it afterward. This keeps the intended store/design context while explicitly controlling the locale used for translated phrases. Adobe Commerce's environment-emulation implementation applies the emulated store's locale to the translator, as shown in the Magento_Store environment emulation source. Adobe's environment emulation documentation also describes its purpose as changing application context for a store.

QUESTION NO: 3

What folder would a developer place a custom patch on an Adobe Commerce Cloud project to have it automatically applied during the build phase?

A. Add the patch file to the m2-hotfixes/ directory

B. Add the patch file to the patches/ directory

C. Add the patch file to the m2-patches/ directory

ANSWER: A

Explanation:

"Add the patch file to the m2-hotfixes/ directory" is correct. In an Adobe Commerce on cloud infrastructure project, the `m2-hotfixes/` directory at the project root is the designated location for custom hotfix patch files that must be applied automatically during deployment. The `ece-patches` package applies patches found there during the build phase, before the deploy phase produces and releases the application artifact. This makes the directory appropriate for maintaining project-specific fixes in version control without manually applying them to each environment. Patch files in `m2-hotfixes/` are applied in lexicographical (alphabetical) filename order, so developers can control a required sequence by naming files accordingly, such as with numeric prefixes. The patch must use a supported patch-file format and be committed to the Cloud project repository so it is available to the build container. Adobe's Cloud documentation specifically identifies `m2-hotfixes/` as the location for custom patches and explains that these patches are applied during the build process. For implementation details and patch ordering guidance, see Apply patches in Adobe Commerce on cloud infrastructure and Cloud Patches for Commerce.

QUESTION NO: 4

A message queue currently has queue/consumer-wait-for-messages set to true, which allows the consumer process to run until a message is inserted into the queue. A piece of functionality is driven by data stored in the model

\Magento\variable\Model\variable and this value is only loaded once during the consumer run. If the variable is updated we want the consumer to restart so that the new value is loaded into memory without having to reload the variable on each message consumed.

The Adobe Commerce developer has created an after plugin on the \Magento\Variable\Model\variable:: save() function.

How would the developer use the plugin to trigger the consumer restart?

A. Call the function \Magento\Framework\MessageQueue\PoisonPill\PoisonPillPutInterface::put().

B. Call the function \Magento\Framework\MessageQueue\Consumers\TriggerRe5tartInterface:trigger().

C. Set the global Cache key trigger_consumer_restart t0 1.

ANSWER: A

Explanation:

Call the function `\Magento\Framework\MessageQueue\PoisonPill\PoisonPillPutInterface::put()`. Adobe Commerce provides the poison-pill mechanism specifically to invalidate currently running message-queue consumers when a change means their in-memory state is no longer valid. The developer should inject `PoisonPillPutInterface` into the plugin and invoke `put()` after the variable has been successfully saved. This records a new poison-pill value that active consumers detect while processing messages. A consumer that detects the changed value exits cleanly rather than continuing to use the value loaded when its process began. When the consumer is started again through the configured consumer-running mechanism, its initialization code loads the updated variable value into memory. This approach avoids adding a database or model reload to every individual message while still ensuring configuration-like data changes are applied promptly. The interface is part of the framework's message-queue poison-pill implementation; its source defines `put()` as the operation used to publish the changed poison-pill state. See the PoisonPillPutInterface source and Adobe's message queue operations documentation.

QUESTION NO: 5

An Adobe Commerce developer is asked to change the tracking level on a custom module for free downloading of pdf and images.

The module contains following models:

Vendor\FreeDownload\Model\Download

Vendor\FreeDownload\Model\DownloadPdf extends Vendor\FreeDownload\Model\Download

Vendor\FreeDownload\Model\DownloadImage extends Vendor\FreeDownload\Model\Download

Download class has a parameter for tracking_level.

How will the developer configure the tracking_level parameter, in di.xml.to have a value of 4 for Download class and all classes that extend Download?

A)

B)

C)

A. 4

B. 4

C. 4

ANSWER: C

Explanation:

The appropriate configuration is to declare the constructor argument for `Vendor\FreeDownload\Model\Download` in the module's `etc/di.xml` and assign the numeric value `4`. Adobe Commerce merges dependency-injection configuration and applies argument configuration declared for a class to its descendant classes. Consequently, `DownloadPdf` and `DownloadImage`, which extend `Download`, receive the same resolved `tracking_level` value unless a more specific configuration overrides that argument for one of those classes.

The `<type>` node identifies the PHP class whose construction settings are being configured. Within it, `<arguments>` supplies constructor parameters by name. Since the required value is an integer, the argument uses `xsi:type="number"` with a value of `4`. This keeps the setting centralized at the inheritance root and ensures a consistent tracking level throughout the download-model hierarchy. Adobe's dependency-injection documentation specifically states that configuration arguments for a class are inherited by its subclasses, which is the behavior required here. See Adobe Commerce dependency injection configuration and parameter configuration inheritance.

QUESTION NO: 6

In which two directories are third-party modules located by default? (Choose two.)

A. vendor/

B. app/packages/

C. app/modules/

D. app/code/

ANSWER: A D

Explanation:

The correct default locations are `vendor/` and `app/code/`. Adobe Commerce uses `vendor/` for modules delivered as Composer packages. When a merchant or development team installs a third-party extension through Composer, Composer places the package beneath `vendor/` and records the dependency in `composer.json` and `composer.lock`. This makes the extension reproducible across environments when dependencies are installed through Composer.

The `app/code/` directory is the application code pool for modules that are maintained directly in the project, including manually installed third-party modules or modules copied from a repository rather than brought in as Composer dependencies. Modules in this location follow the vendor/module namespace structure, such as `app/code/VendorName/ModuleName`. Adobe's component-location guidance identifies both locations as module code locations, while its extension-development documentation describes the standard module directory structure. See Adobe Commerce component locations and Adobe Commerce component file structure.

QUESTION NO: 7

Which file should a developer use to set the default value when creating configuration fields for admin?

A. etc/adminhtml/config.xml

B. etc/config.xml

C. etc/adminhtml/system.xml

ANSWER: B

Explanation:

etc/config.xml is the module configuration file used to declare default values for Magento configuration paths. Within its `<default>` node, the module defines values using the same section, group, and field path structure that is exposed in the Admin configuration UI. For example, a value for a field at `vendor_section/general/enabled` is declared as nested XML beneath `<default>`. Magento merges this module configuration with configuration from other modules and uses it as the default when no value has been saved for the relevant scope. This gives a module predictable initial behavior without requiring an administrator to save its settings first. The default declaration is separate from the Admin UI definition: configuration fields themselves are defined in `etc/adminhtml/system.xml`, while `etc/config.xml` supplies the configuration's default data. Adobe's configuration overview documents that module defaults are declared in `config.xml` and describes the configuration hierarchy and scope resolution used to retrieve them. See the Adobe Commerce configuration guide and the Admin system configuration tutorial.

QUESTION NO: 8

A developer is adding a custom summary component to the checkout. The component must be rendered through the standard checkout UI component framework without modifying core checkout files.

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

A. Add the component configuration to the checkout page `jsLayout` in layout XML.

B. Create the JavaScript component under the module or theme `view/frontend/web/js` directory.

C. Add the component only as a PHP block in the checkout layout.

D. Create the component in `view/adminhtml/ui_component`.

ANSWER: A B

Explanation:

The checkout page exposes its JavaScript component tree through the `jsLayout` argument. The developer should extend the appropriate checkout layout, commonly `checkout_index_index.xml`, and add the component configuration to the relevant `jsLayout` node.

The configured component must also exist as a JavaScript UI component in the module or theme, normally under `view/frontend/web/js`, and return a component that extends the expected UI component behavior. A PHP block alone does not register a Knockout UI component, and adding an Admin UI component XML file does not affect the storefront checkout.

QUESTION NO: 9

An Adobe Commerce developer wants to generate a list of products using ProductRepositorylnterf ace and search for products using a supplier_id filter for data that is stored in a standalone table (i.e., not in an EAV attribute).

Keeping maintainability in mind, how can the developer add the supplier ID to the search?

A. Write a before plugin on `\Magento\Catalog\Model\ProductRepository::getList()` and register the search criteria passed. Write an event observer to listen for the `catalog_product_collection_load_before` event. Iterate through the registered search criteria, and if found, apply the needed join and filter to the event's collection.

B. Add a custom filter to the virtual type `Magento\Catalog\Model\Api\SearchCriteria\CollectionProcessor\ProductFilterProcessor` for the `supplier_id` field. In the custom filter, apply the needed join and filter to the passed `$collection`.

C. Write a before plugin on `\Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface::process()`. Iterate through the `$searchCriteria` provided for `supplier_id`, and if found, apply the needed join and filter to the passed `$collection`.

ANSWER: B

Explanation:

Add a custom filter to the virtual type `Magento\Catalog\Model\Api\SearchCriteria\CollectionProcessor\ProductFilterProcessor` for the `supplier_id` field, and apply the required join and condition to the supplied collection. This integrates the standalone-table field into the established product repository search-criteria pipeline. When `ProductRepositoryInterface::getList()` processes a `SearchCriteria` object, the product collection processor delegates recognized custom fields to their registered custom filters. The custom filter can join the supplier-data table using the appropriate product key and then constrain the joined data to the requested supplier ID.

This design keeps the behavior scoped to product search criteria rather than coupling it to generic collection-load events or repository interception. It is also reusable: any caller that supplies a `supplier_id` filter through the repository receives the same consistent behavior. Registering the filter through dependency injection against the product filter processor follows Adobe Commerce's extensibility model and avoids modifying framework or core code. Adobe's repository guidance describes using search criteria for repository list operations, while the catalog implementation provides the product-specific collection-processing extension point for custom field filtering. See Adobe Commerce: Searching with repositories and ProductFilterProcessor source.

QUESTION NO: 10

A child theme must customize a template and a JavaScript file supplied by `Vendor_Module` without changing the module source code.

Which two theme locations should the developer use? (Choose two.)

A. `/Vendor_Module/templates/...`

B. `/Vendor_Module/web/...`

C. `/web/Vendor_Module/...`

D. `/view/adminhtml/Vendor_Module/...`

ANSWER: A B

Explanation:

To override a module template, the child theme places the replacement below `Vendor_Module/templates` in the theme directory, preserving the template's relative path. Adobe Commerce resolves the theme file before the module file.

To override a module static asset, such as a JavaScript file, the child theme places the replacement below `Vendor_Module/web`, again preserving the relative path. Placing the file only in the theme's generic `web` directory does not identify it as an override for the module asset, and `view/adminhtml` applies to Admin assets rather than storefront theme fallback.

QUESTION NO: 11

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

```
<table name="student_details" resource="default" engine="innodb" comment="Students Detail Table">
        <column xsi:type="int" name="entity_id" padding="10" unsigned="true" nullable="false" identity="false"
                comment="Entity Id"/>
        <column xsi:type="smallint" name="roll_no" padding="2" unsigned="true" nullable="false"
                identity="true" default="null" comment="Student Roll No"/>
        <column xsi:type="text" name="student_name" nullable="false" comment="Student Name"/>
        <column xsi:type="varchar" name="student_class" length="5" nullable="false" comment="Student Class"/>
    </table>
```

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 numeric value. In the underlying MySQL table, an AUTO_INCREMENT column must be defined as a key, so the schema must provide the required key/index definition for the identity column. This allows Adobe Commerce to generate a valid table definition and permits the database to maintain the next generated value correctly.

Column (student_name) does not have attribute length. A string value stored as a variable-length character column needs an explicit length in the declarative schema definition so that Adobe Commerce can generate the corresponding database column with a valid maximum size. Supplying a length, such as 255 where appropriate for the application’s data requirements, makes the column definition complete and deterministic during schema processing.

Adobe Commerce declarative schema uses `db_schema.xml` to describe table columns and database constraints/indexes, then compares that declaration with the current database state during setup. The platform documentation describes column attributes, including identity and length, and the required schema structures for keys and indexes. See the Adobe Commerce declarative schema configuration documentation and the MySQL AUTO_INCREMENT documentation.

QUESTION NO: 12

A module must expose a REST endpoint that allows an administrator to retrieve data from a custom service. The endpoint must be available only to integrations or administrators that have been granted the module's ACL resource.

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

A. Declare the route, service operation, and resource in `etc/webapi.xml`.

B. Declare the module resource hierarchy in `etc/acl.xml`.

C. Declare the endpoint path in `etc/frontend/routes.xml`.

D. Declare the endpoint authorization in `etc/di.xml`.

ANSWER: A B

Explanation:

The module must declare the REST route, service class, service method, and required ACL resource in `etc/webapi.xml`. The resource referenced by the route must also be declared in `etc/acl.xml`. Commerce evaluates the authenticated user's or integration's assigned ACL permissions against that resource before allowing the request.

`routes.xml` defines frontend or admin controller routing, not Web API service routes. `di.xml` can configure dependency injection but does not expose a service operation as a REST endpoint or define its authorization requirement.

QUESTION NO: 13

What is the command used to upgrade ECE-Tools on an Adobe Commerce Cloud platform?

A. php ./vendor/bin/ece-tools upgrade

B. composer update magento/ece-tools --with-all-dependencies

C. magento-cloud ece-tools:upgrade

ANSWER: B

Explanation:

`composer update magento/ece-tools --with-all-dependencies` is the appropriate command because ECE-Tools is delivered as the Composer package `magento/ece-tools` in an Adobe Commerce on cloud infrastructure project. Updating it through Composer changes the package version recorded in the project's dependency definition and lock file, rather than attempting to perform an in-place upgrade through a standalone executable or Cloud CLI command. The `--with-all-dependencies` flag permits Composer to update packages that ECE-Tools requires, including dependencies that may otherwise be held at versions specified by the existing lock file. This is important when the selected ECE-Tools release has compatible dependency-version requirements that differ from the currently installed release. After committing the resulting `composer.json` and `composer.lock` changes, the Cloud build and deployment process installs the resolved ECE-Tools version for the environment. Adobe documents ECE-Tools as a Composer package and provides guidance for managing and upgrading it in cloud projects. See the ECE-Tools package overview and Adobe's ECE-Tools update guidance.