DUMPSBOSS.

Salesforce Certified B2C Commerce Developer

Salesforce Certified-B2C-Commerce-Developer

Version Demo

Total Demo Questions: 24

Total Premium Questions: 244

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, B2C Commerce Setup	38
Topic 2, Work With a B2C Site	34
Topic 3, Data Management	32
Topic 4, Application Development	128
Topic 5, Application Security	12
Total	244

QUESTION NO: 1

A developer has created a Page Designer page with the ID `summer-landing-page`. The page must be rendered from a storefront controller.

Which method should the developer use?

A. PageMgr.renderPage('summer-landing-page')

B. ContentMgr.renderContent('summer-landing-page')

C. CatalogMgr.renderPage('summer-landing-page')

D. URLUtils.renderPage('summer-landing-page')

ANSWER: A

Explanation:

`PageMgr.renderPage()` is correct because it renders a Page Designer page by its page ID. The method allows a storefront controller to request the configured Page Designer page and return its rendered output for the current request context.

The page's type and component configuration are managed in Page Designer, while its rendering scripts and templates are supplied by the cartridge implementation. A controller can use `PageMgr.renderPage('summer-landing-page')` when it needs to render that managed page as the response for a route. See the PageMgr Script API reference and the Develop for Page Designer documentation.

QUESTION NO: 2

A merchant uploads an image using the Content Image Upload module of Business Manager.

Which three modules can the merchant or developer use to display the image on the Storefront?

Choose 3 answers

A. ISML templates

B. Content assets

C. Storefront catalogs

D. Content slots

E. Payment types

ANSWER: A B D

Explanation:

Images uploaded through Business Manager's Content Image Upload functionality are stored in the site's content library and can be referenced in several storefront-content mechanisms. **ISML templates** can render the image directly by resolving its content-library URL and placing that URL in an image element or another suitable HTML element. This supports developer-controlled presentation wherever the template is used. **Content assets** can include uploaded library images in rich content, allowing merchants to manage promotional, editorial, and informational content without changing storefront code. **Content slots** display content selected through Business Manager at defined locations on storefront pages; when a slot renders a content asset, the asset can in turn render its associated content-library image. Together, these mechanisms support both code-driven and merchant-managed use of uploaded content images, while retaining the image as a reusable library resource. Salesforce's content-management documentation describes content libraries, assets, and slot-based rendering, and the ISML documentation covers template-driven storefront output. See Salesforce B2C Commerce Content Management and Salesforce ISML documentation.

QUESTION NO: 3 - (HOTSPOT)

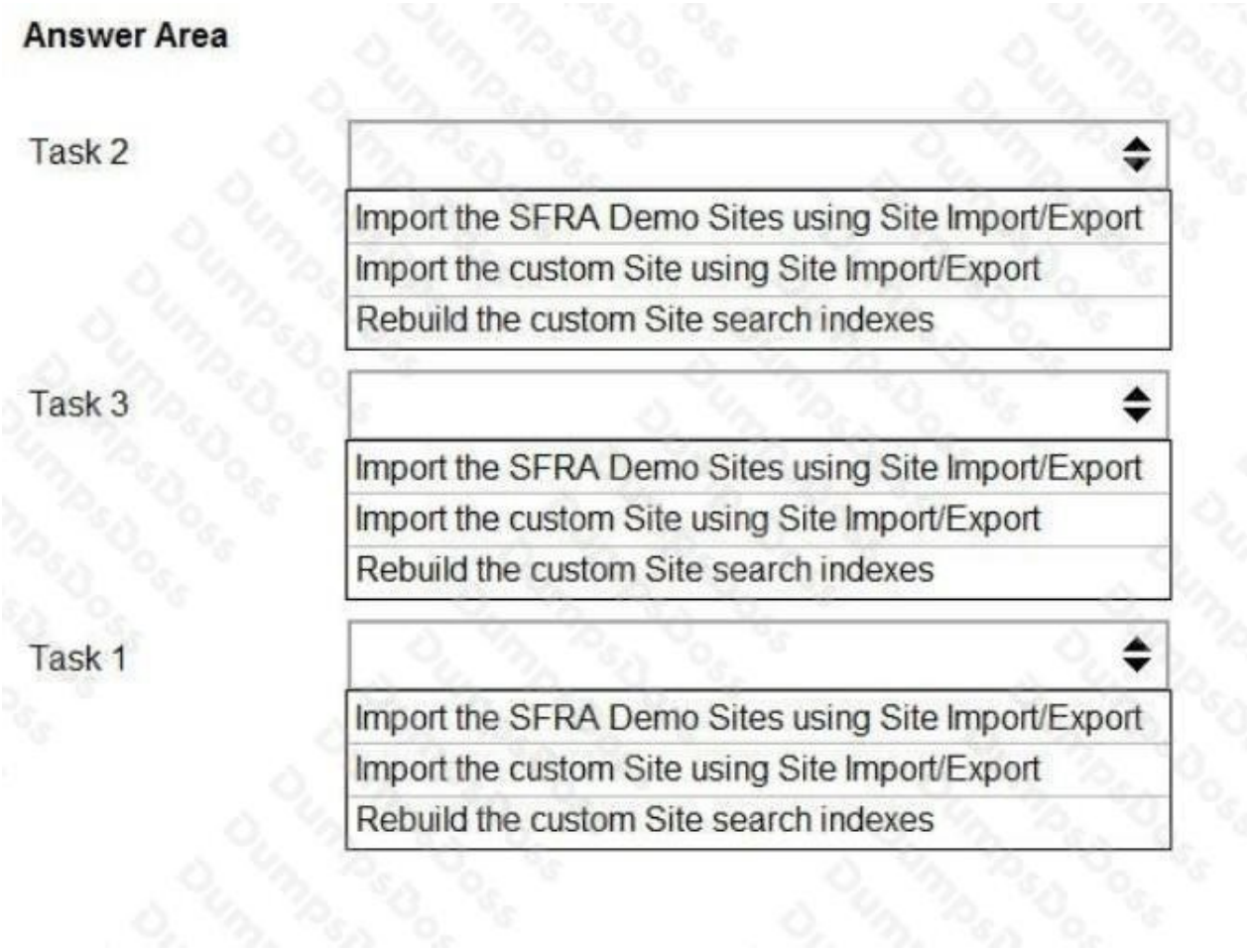
HOTSPOT

Given the following conditions:

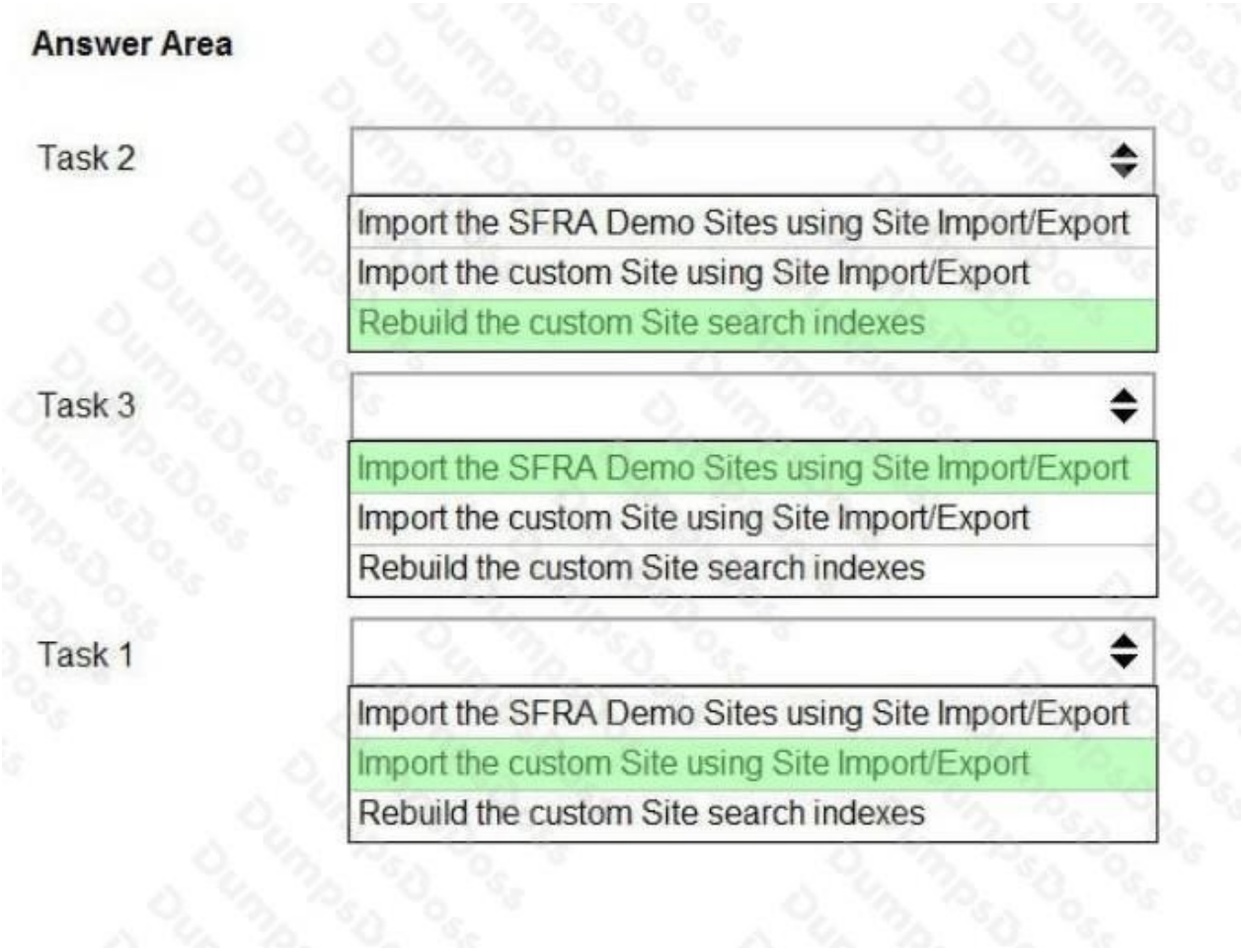
- A site export file with a copy of the Storefront data for a custom site
- A sandbox with the custom site code, but no Storefront data
- A requirement for a working copy of SFRA for development reference

A developer is assigned three tasks to perform in Business Manager. In what sequence should the developer perform the tasks, so that the custom site displays the products as intended?

Hot Area:



ANSWER:



Explanation:

The required sequence is to import the SFRA Demo Sites first, import the custom site second, and rebuild the custom-site search indexes last. Importing the SFRA Demo Sites creates the working SFRA reference storefront in the sandbox, which satisfies the development-reference requirement before custom storefront data is layered into the environment. The SFRA reference data is supplied as a site archive and is imported through Business Manager's site import/export capability, as described in Salesforce's Importing and Exporting Data documentation.

Once the SFRA demo data is available, the developer imports the custom-site export. That archive contains the custom site's storefront data, including the catalog-related content needed by the custom site. Importing it after the SFRA demo sites ensures the sandbox has both the requested SFRA reference and the custom-site data that the custom storefront is intended to use.

The search index rebuild must be the final action because the index needs to reflect the data after all imports are complete. Search indexes are generated from the current catalog and product-search configuration; rebuilding after the custom-site archive is imported makes the newly imported products searchable and available for storefront product listings and search results. Salesforce documents the role of search indexing and its administration in B2C Commerce Search. Completing the rebuild last therefore produces a sandbox where the custom site has its imported product data, the SFRA reference is available, and the storefront search index corresponds to the final imported catalog state.

QUESTION NO: 4

There are three logging categories: category1, category1.eu, and category1.us.

In Business Manager, category1 is enabled for WARN level and no other categories are configured. All custom log targets are enabled. The code segment below executes

var logger = Logger.getLogger("loggerFile", "category1.eu"); logger.warn("This is a log message");

What is the result?

A. Logs will be written to the log file with a prefix loggerFile.

B. Logs will not be written.

C. Logs will be written to the log file with a prefix customwarn.

D. Logs will be written to the log file with a prefix custom-loggerFile.

ANSWER: D

Explanation:

Logs will be written to the log file with a prefix custom-loggerFile. The logger is created with `loggerFile` as its file prefix and `category1.eu` as its category. Commerce Cloud logging categories are hierarchical, so configuration for the parent category `category1` applies when no more-specific configuration exists for `category1.eu`. Since the configured level is WARN and the code calls `logger.warn()`, the message meets the enabled severity threshold and is emitted.

Because this is a custom logger obtained through `Logger.getLogger(prefix, category)`, Commerce Cloud writes the entry to a custom log target rather than a standard application log. Custom log files use the `custom-` naming convention together with the supplied prefix, producing a file name beginning with `custom-loggerFile` (with additional platform-generated naming components such as date or instance information). The category controls whether the message is enabled, while the prefix controls the custom log file name. See the Salesforce B2C Commerce Script API documentation for dw.system.Logger and the dw.system.Log API.

QUESTION NO: 5

Recent code changes to an existing cartridge do not appear correctly on a Storefront. The developer confirms that the code is uploaded in the IDE and ensures that the cartridge is associated with the sandbox.

Which two additional steps should the developer take to troubleshoot this problem? (Choose two.)

A. Check that the search index was recently rebuilt.

B. Check the Business Manager site cartridge path.

C. Check that the correct code version is selected.

D. Check the Storefront site cartridge path.

ANSWER: C D

Explanation:

The developer should check that the correct code version is selected and check the Storefront site cartridge path. In B2C Commerce, uploading code to a sandbox makes the code version available, but it does not by itself ensure that the sandbox is running that version. The active code version must be selected for the instance; otherwise, the storefront continues to execute the previously active version even though the updated cartridge files are present.

In addition, cartridge assignment is evaluated through the storefront site's cartridge path. The cartridge containing the changed controllers, pipelines, templates, scripts, or other storefront resources must be included in that site's path. Its position is also significant because B2C Commerce resolves cartridge resources according to cartridge-path order, allowing cartridges earlier in the path to override resources supplied by later cartridges. Confirming both the active code version and the relevant storefront cartridge path verifies that the runtime is using the uploaded cartridge and resolving its updated resources. Salesforce documents code-version activation and cartridge-path behavior in the B2C Commerce code deployment guide and the B2C Commerce cartridges guide.

QUESTION NO: 6

A client sells its products in North America, Europe, and Asia, and has a B2C Commerce Site for each of these markets. The client receives three area-specific snippets of analytics code by a third-party provider to insert in the sites.

How should the developer configure an instance to allow the merchant to independently insert and update these snippets?

A. Create a new "HTML" attribute in the SitePreference object type.

B. Use ISML conditional tags to add the snippet into the codebase.

C. Configure a new Service Profile with the provided snippet of code.

ANSWER: A

Explanation:

Create a new "HTML" attribute in the SitePreference object type. Site preferences are configuration values that Business Manager users can maintain separately for each site, which directly supports distinct analytics snippets for the North American, European, and Asian storefronts. A custom site-preference attribute with the HTML data type provides an appropriate merchant-editable field for storing markup such as a third-party analytics script or tracking pixel. The developer can then retrieve the preference in the relevant ISML template or controller rendering context so the configured value is emitted for that site. This approach separates merchant-managed, market-specific content from deployed cartridge code: analytics providers can change their snippet, and the merchant can update the corresponding site preference without requiring a code release. Site preferences are administered in Business Manager and are scoped at the site level when configured as site-specific preferences, enabling each market site to retain its own independent value. Salesforce documents custom preferences and their use in site configuration in the B2C Commerce Custom Preferences guide. The platform's Business Manager documentation also describes Business Manager as the administrative interface for managing site configuration and business data.

QUESTION NO: 7

Given a customer environment configured with only the en_CA locale and the following new requirements:

- To add a new locale for fr_CA
- To localize the address form with the new locale
- To make the localization usable even for new possible French locales, such as fr_FR

And given the portion of form XML definition contained in the form file cartridge/forms/ default/address.xml:

```
<?xml version="1.0"?>
<form xmlms="http://www.demandware.com/xml/form/2008-04-19">

      <field formid="country" label="label.input.country" type="string" mandatory="true"/>
      ...
</form>
```

What is the right place to add the fr_CA translation for the country field label?

A. /cartridge/templates/resources/address_fr_CA.properties

B. /cartridge/templates/resources/forms_fr.properties

C. /cartridge/forms/resources/address_fr.properties

D. /cartridge/templates/resources/fr/forms.properties

ANSWER: B

Explanation:

`/cartridge/templates/resources/forms_fr.properties` is the correct location because labels referenced by an SFCC form definition are resolved from the cartridge's resource bundles under `templates/resources`. A form label such as the country-field label uses a resource key, and the corresponding translated value belongs in the applicable `forms` resource bundle.

Using the language-level suffix `_fr`, rather than a region-specific `_fr_CA` suffix, provides the requested reuse. SFCC locale resource resolution can use a language bundle as a fallback for locales sharing that language. Therefore, the French translation in `forms_fr.properties` can be used for `fr_CA` and can also serve a future French locale such as `fr_FR`, unless a locale-specific bundle is later added to override it. This keeps common French form text centralized while still allowing Canadian- or France-specific wording when necessary.

Salesforce documents that localized storefront resources are maintained as properties files in a cartridge's `templates/resources` directory and that locale-specific resource bundles support fallback behavior. See Salesforce B2C Commerce localization documentation and Salesforce B2C Commerce forms documentation.

QUESTION NO: 8

A Digital Developer is asked to optimize controller performance by lazy loading scripts as needed instead of loading all scripts at the start of the code execution.

Which statement should the Developer use to lazy load scripts?

A. importPackage () method

B. $.ajax () jQuery method

C. local include

D. require () method

ANSWER: D

Explanation:

The `require()` method is correct because Salesforce B2C Commerce script modules are loaded on demand when execution reaches a `require()` call. This lets a controller defer loading a dependency until the code path actually needs it, rather than eagerly loading every potentially needed module when the controller begins. A developer supplies the cartridge-relative module path, such as `require('*/cartridge/scripts/helpers/exampleHelper')`, and assigns the returned module exports to a variable for use in the current script. This modular approach improves organization and can reduce unnecessary work for requests that do not use a particular feature or helper.

In SFRA controllers, required modules are commonly declared near the top of the file for essential dependencies. However, when a module is relevant only to a conditional branch or infrequently used action, placing `require()` within that branch implements the requested lazy-loading behavior. B2C Commerce resolves CommonJS modules through its cartridge path mechanism, including the `*/` syntax, which searches cartridges in the active cartridge path. This ensures the appropriate module implementation is located at runtime while preserving reusable, maintainable server-side code.

See Salesforce's Storefront Reference Architecture documentation and the B2C Commerce cartridge documentation for module and cartridge-path context.

QUESTION NO: 9

A developer is asked to write a log containing the ID and name of the product with a variable named myProduct.

Which snippet of code should be used?

A. Logger.warn('The current product is ${myProduct.getID()} with name ${myProduct.getName()}');

B. Logger.warn('The current product is {0} with name {1}', myProduct.getID(), myProduct.getName());

C. Logger.warn('The current product is %s with name %s').context
(myProduct.getID(), myProduct.getName());

D. Logger.warn('The current product is {0} with name {1}'}.context
(myProduct.getID(), myProduct.getName());

ANSWER: B

Explanation:

`Logger.warn('The current product is {0} with name {1}', myProduct.getID(), myProduct.getName());` is correct because B2C Commerce Script API logger methods accept a message pattern followed by replacement arguments. The placeholders `{0}` and `{1}` identify the first and second supplied arguments, respectively. At runtime, `myProduct.getID()` supplies the product identifier and `myProduct.getName()` supplies its display name, producing a single warning-level log entry with both values included in the message.

This approach uses the formatting mechanism supported by `dw.system.Logger`, which documents logger methods such as `warn` as taking a message and optional arguments. The message pattern follows Java `MessageFormat`-style placeholder notation, allowing values to be passed separately from the text rather than assembled through interpolation. This is appropriate for diagnostic logging because it makes the intended substitutions explicit and uses the platform logger's standard API. See the B2C Commerce Logger Script API reference and the Java MessageFormat documentation.

QUESTION NO: 10

A developer is creating a custom job step that must process a large import file one record at a time and write the processed records in batches.

Which type of job step should the developer implement?

A. A chunk-oriented job step

B. A task-oriented job step

C. A pipeline job step

D. A controller job step

ANSWER: A

Explanation:

A chunk-oriented job step is correct because it is designed to process a large data set incrementally. A chunk step can implement lifecycle functions such as `beforeStep`, `read`, `process`, `write`, and `afterStep`. The framework repeatedly reads and processes items, then writes them in configured chunks, which helps control memory use and supports efficient batch processing.

A task-oriented step is intended for a single unit of work and exposes an `execute` function. It does not provide the item-by-item read, process, and write lifecycle required for this import scenario. See the B2C Commerce Job Framework documentation for job-step implementation details.

QUESTION NO: 11

Universal Containers wants to change a content slot that is currently configured to display a content asset. Now they want the slot to display the top five selling boxes for the week.

Which two changes need to be made for this to occur? (Choose two.)

A. Change the slot's configuration content type to "products."

B. Change the slot's configuration content type to "recommendations."

C. Change the slot's configuration template to the appropriate rendering template.

D. Delete the existing content asset.

ANSWER: B C

Explanation:

To render a changing set of merchandise, such as the five best-selling boxes for a given week, the slot must be configured as a recommendations slot. A recommendations content type allows the slot to receive a product-recommendation result rather than a single, fixed content asset. The recommendation source can supply the relevant ranked products, while the slot placement remains available for page rendering.

The slot configuration must also use an appropriate rendering template. The template is responsible for iterating through the recommendation result and producing the required storefront markup for the products, such as product names, images, prices, and links. Content-slot configuration therefore combines the type of content being supplied with the template that knows how to render that content. After these configuration changes, the business can associate the applicable recommendation data with the slot and the storefront can display the weekly top-selling products.

Salesforce documents content-slot configuration, including content types and rendering templates, in the B2C Commerce Content Slots guide. See also the B2C Commerce recommendations documentation for the role of recommendation data in storefront experiences.

QUESTION NO: 12

Given the customer basket described below:

- A customer has an existing basket that consists of multiple items.
- One of the items is identified as a gift item by an attribute at the product line item.

The developer needs to write custom code to fetch the customer basket and then modify the basket based upon the items in the cart. If the basket contains any gift items, modify the basket and create a separate shipment for the gift item.

Four hooks are required to make the modification, beginning with modifyGETResponse and ending with validateBasket.

1. dw.ocapi.shop.basket.modifyGETResponse

2. -- missing hook -3. -- missing hook --

4. dw.ocapi.shop.basket.validateBasket

What are the two missing hooks on the middle? (Choose two.)

A. dw.ocapi.shop.basket.shipment.afterDELETE

B. dw.ocapi.shop.basket.shipment.beforePOST

C. dw.ocapi.shop.basket.shipment.beforePATCH

D. dw.ocapi.shop.basket.shipment.beforeDELETE

ANSWER: B D

Explanation:

The appropriate middle hooks are `dw.ocapi.shop.basket.shipment.beforePOST` and `dw.ocapi.shop.basket.shipment.beforeDELETE`. The `modifyGETResponse` hook runs when the basket GET response is being prepared, enabling the customization to identify product line items marked as gifts and to expose or initiate the required basket-handling logic. Creating a distinct shipment for those items is a shipment creation operation, so `shipment.beforePOST` is the relevant extension point. It runs before OCAPI creates the shipment and allows the implementation to apply the required custom shipment behavior before the operation is completed.

Shipment removal is also part of maintaining the intended shipment structure when basket contents change. The `shipment.beforeDELETE` hook provides the point to perform custom handling before a shipment deletion is processed, such as maintaining the separation and consistency of gift-item shipment assignments as shipment data is updated. Finally, `validateBasket` validates the resulting basket state before it is accepted, ensuring the custom shipment arrangement remains valid. Salesforce documents OCAPI hook names and their operation-specific lifecycle points in the B2C Commerce OCAPI Hooks documentation and the OCAPI for B2C Commerce guide.

QUESTION NO: 13

A developer needs to create a custom object instance that stores an external loyalty identifier for a shopper.

Which two actions are required to create the persistent custom object correctly?

Choose 2 answers

A. Call `CustomObjectMgr.createCustomObject()`.

B. Perform the creation inside `Transaction.wrap()`.

C. Store the loyalty identifier in session privacy data.

D. Use `ProductMgr.createProduct()`.

E. Add the identifier directly to the request object.

ANSWER: A B

Explanation:

The developer must use `CustomObjectMgr.createCustomObject()` to create the instance for the configured custom object type and key. Because the object is persistent data, its creation and any custom-attribute updates must occur within a transaction, such as `Transaction.wrap()`.

Session privacy data is not a replacement for a custom object because session data is temporary and cannot provide the required persistent record. ProductMgr is used for catalog product operations, not for creating custom object instances. See the CustomObjectMgr Script API reference and the Transaction Script API reference.

QUESTION NO: 14

A Digital Developer is requesting product information for an external integration. The following Open Commerce API (OCAPI) request is NOT functioning correctly:

```
POST /dw/shop/v18_3/products/(creative-zen-v,namco-we-ski-wii)?client_id=aaaa...
HTTP/1.1 Host: example.com
```

How should the Developer change the request?

A. Change the URI to /dw/shop/v18_3/products/creative-zen-v.

B. Change the HTTP method to PUT.

C. Change the HTTP method to GET.

D. Include an authentication token in the request.

ANSWER: C

Explanation:

Change the HTTP method to GET. The OCAPI Shop API product resource is a read operation when a developer needs information for one specific product. A request to the product endpoint, using the product ID in the URI, retrieves the product representation and applicable product details for the current site and shopper context. HTTP GET is the REST method intended for retrieving an existing resource without modifying it, making it the appropriate method for an external integration that is requesting product information.

In OCAPI, product retrieval is exposed through the Products resource. The resource documentation identifies the single-product endpoint and its retrieval operation, including the product ID path parameter and optional expansion parameters for requesting related data. Using GET ensures the request is routed to that resource operation and returns the requested product data in the response body. The integration must still be configured with the appropriate OCAPI permissions and any required client identification, but the necessary change to the nonfunctioning request shown is to use the retrieval method. See the Salesforce OCAPI Shop API Products resource reference and the B2C Commerce OCAPI reference.

QUESTION NO: 15

A Digital Developer must resolve a performance issue with product tiles. The Developer determines that the product tiles are NOT being cached for a long enough period.

Which two methods can the Developer use to verify the cache settings for the product tiles?

(Choose two.)

A. Enable cache information in the storefront toolkit and view the cache information for the product tile.

B. View the cache information provided by the Merchant Tools > Technical Reports Business Manager module.

C. View the product list page cache settings provided in the Administration > Manage Sites Business Manager module.

D. Enable the template debugger to verify the cache times for the producttile.isml template.

ANSWER: A C

Explanation:

Enabling cache information in the storefront toolkit and inspecting the product tile is a direct way to verify the effective cache behavior seen by a storefront request. The cache-information display exposes details such as cache status and expiration-related information, allowing the developer to determine whether the tile response is cached and whether its lifetime aligns with the intended configuration. This is particularly useful because product tiles can be rendered as separately cached fragments within a product-list page.

The product-list page cache settings in Business Manager under Administration > Manage Sites provide the site-level configuration used to control caching for product-list content. Reviewing these settings lets the developer validate the configured cache lifetime that applies to product-list pages and their tile content, then adjust the configuration if it does not meet performance requirements. Together, storefront cache information verifies the runtime result, while the site configuration confirms the applicable cache policy. Salesforce documents response expiration APIs and cache-control behavior in the Response API reference and provides broader implementation guidance in the B2C Commerce caching documentation.

QUESTION NO: 16

In order to build the SFRA code to a developer sandbox for the first time, which build steps should the developer perform for the site to appear and function as designed?

A. npm run compile:js, npm run compile:html, npm run clean

B. npm run compile:js, npm run compile:scss, npm run compile:fonts

C. npm run compile:js, npm run compile:scss, npm run compile:html

D. npm run compile:scss, npm run compile:html, npm run clean

ANSWER: B

Explanation:

`npm run compile:js, npm run compile:scss, npm run compile:fonts` is correct because an initial SFRA deployment needs the storefront's client-side assets to be generated from their source files. The JavaScript compilation task

bundles and transpiles the client-side scripts used for interactive storefront behavior. The SCSS compilation task converts the SFRA stylesheets into browser-ready CSS, which provides the intended visual layout and styling. The font compilation task copies the font assets into the static output so typography and icon fonts can load correctly in the sandbox.

These build tasks create the static files in the cartridge locations that SFRA serves as part of the storefront. After generating them, the developer uploads the cartridges and ensures the appropriate cartridges are included in the site cartridge path. This initial asset build is particularly important after obtaining a fresh SFRA codebase because generated static assets are not necessarily present or current in source control. SFRA's build tooling and project setup use npm scripts to produce these deployable client-side resources. See the SFRA repository's build guidance in the Storefront Reference Architecture repository and Salesforce's B2C Commerce cartridge documentation.

QUESTION NO: 17

A developer is asked to periodically create a CSB file in a WebDAV folder to hold the orders information of the last 30 days.

What are the appropriate actions to implement such a requirement?

A. Develop and configure a steptype and corresponding CommonJS job step script.

B. Implement and configure a recurring task using the cron command in Business Manager.

C. Configure a new custom OCAPI endpoint and use the Customers resource type.

ANSWER: A

Explanation:

Develop and configure a steptype and corresponding CommonJS job step script is correct because Salesforce B2C Commerce jobs are the platform-supported mechanism for executing recurring, server-side operational work. A custom job step is declared through a `steptypes.json` definition and invokes a CommonJS script module. That script can query the relevant order records for the required 30-day period, build the export content, and write the resulting file to an IMPEX/WebDAV-accessible location by using the platform file APIs. After the custom step is available, a Business Manager administrator can add it to a job and configure the job's schedule, allowing the export to run periodically without an external client initiating it.

This design also keeps the business logic versioned in the cartridge, provides job execution history and status reporting, and uses the intended B2C Commerce operational framework for batch processing. The job step can accept parameters when needed, such as a target directory or lookback duration, while the schedule remains configurable independently in Business Manager. Salesforce documents custom job-step definitions and script implementation in its Custom Job Steps documentation, and describes configuration and scheduling through the B2C Commerce job scheduling documentation.

QUESTION NO: 18

Business Manager has the configuration:

▪ Active Log category is "root" ▪ Log level of WARN The code below is executing:

var log = Logger.getLogger("products");

Using this information, which two logs will be written? (Choose two.)

A. log.warn("This is a warn message");

B. log.error("This is an error message");

C. log.info("This is an info message");

D. log.debug("This is a debug message");

ANSWER: A B

Explanation:

The calls `log.warn("This is a warn message");` and `log.error("This is an error message");` will be written. The logger obtained through `Logger.getLogger("products")` is associated with the `products` category. Because the active Business Manager log category is `root`, its configured level applies as the effective level for this logger when no more-specific active category configuration overrides it.

A configured threshold of `WARN` records messages at WARN severity and all more severe levels. Therefore, the WARN call meets the threshold directly, while the ERROR call is also emitted because ERROR is more severe than WARN. This behavior lets a production environment retain actionable warnings and failures without producing the higher-volume informational and debugging output.

Salesforce's Script API identifies the logger methods and their severity levels, including `warn` and `error`: Logger Script API. For configuration and category-level behavior in Business Manager, see Salesforce's B2C Commerce logging documentation.

QUESTION NO: 19

A developer is configuring Open Commerce API access for an external application that must read product data through the Data API.

Which two items must be configured in the OCAPI settings?

Choose 2 answers

A. A client ID for the external application.

B. The permitted API resources and methods.

C. A storefront cartridge path.

D. A content library assignment.

E. A customer group for the API user.

ANSWER: A B

Explanation:

The OCAPI configuration must identify the client application by client ID and define the API resources and HTTP methods that client can access. The client ID associates incoming API requests with the applicable OCAPI permission configuration. Resource permissions determine which Data API resources, such as product resources, can be accessed and whether operations such as GET are permitted.

OCAPI access is controlled through the OCAPI settings rather than through a storefront cartridge path or a customer group. The configuration can also restrict access by site and version as required. See the Open Commerce API documentation.

QUESTION NO: 20

A job executes a pipeline that makes calls to an external system.

Which two actions prevent performance issues in this situation? (Choose two.)

A. Use synchronous import or export jobs

B. Configure a timeout for the script pipelet.

C. Disable multi-threading.

D. Use asynchronous import or export jobs.

ANSWER: B C

Explanation:

Configuring a timeout for the script pipelet prevents a stalled or slow external-system request from occupying a job execution thread indefinitely. An explicit timeout establishes a bounded failure window, allowing the job to handle an unavailable dependency rather than accumulating blocked executions and potentially delaying other scheduled processing. This is particularly important when a pipeline makes network calls, because remote response times and availability are outside Commerce Cloud's control.

Disabling multi-threading is also appropriate when a pipeline-based job makes external calls. It limits concurrent requests generated by that job, avoiding contention for the external system and reducing the risk that parallel executions exhaust connection capacity, trigger throttling, or cause cascading delays. Running the integration work in a controlled, single-threaded manner makes its load predictable and protects overall job performance when the downstream system is slow.

Salesforce's job framework guidance describes how jobs execute scheduled processing and should be designed with operational reliability in mind. For integration calls, Salesforce's service framework documentation also covers defining service behavior, including timeouts, for external HTTP and other service connections. See B2C Commerce Job Framework and B2C Commerce Web Services and Service Framework.

QUESTION NO: 21

In Log Center, a developer notes a number of Cross Site Request Forgery (CSRF) log entries. The developer knows that this happens when a CSRF token is either not found or is invalid, and is working to remedy the situation as soon as possible.

Which two courses of action might solve the problem? (Choose two.)

A. Extend the CSRF token validity to avoid timeouts

B. Delete the existing CSRF whitelists in Business Manager

C. Add the token in the ISML template

D. Add csrfProtection.generateToken as a middleware step in the controller

ANSWER: C D

Explanation:

CSRF validation succeeds only when the request includes a valid token that was generated for the shopper's current session. Therefore, adding the token in the ISML template is an appropriate corrective action. A form that submits to a protected endpoint must render the token as a hidden field, using the token name and value supplied to the template. This ensures the browser returns the token with the form submission, allowing server-side CSRF validation to find and validate it.

Adding `csrfProtection.generateToken` as middleware in the controller is also appropriate because it generates a token and places the required CSRF data in the response view data before the page is rendered. In SFRA, this middleware is normally applied to the route that renders the form, while validation middleware is applied to the route that processes the submission. Used together, these steps establish the expected lifecycle: generate a session-bound token, render it into the form, submit it with the request, and validate it at the protected endpoint. Salesforce's CSRF API documents token generation and request validation, and the SFRA middleware implementation shows how generated values are exposed for templates. See the CSRFProtection Script API and the SFRA CSRF middleware.

QUESTION NO: 22

Consider the following information:

- A merchant has this three-tier category structure setup in the Storefront catalog. - New Arrivals > Women > Clothing

- The category named Clothing has all the clothing items for Women and is merchandised. - A Search Refinement named Newness is correctly configured for the Clothing category.

When a merchandiser views the Clothing category, the Search Refinement appears and works as expected. However, the merchandiser does not see the Search Refinement when searching for Clothing via the Storefront search.

What is the reason?

A. The Search Refinement definition is not set up for the New Arrivals category

B. There are conflicting Search Refinement definitions for Clothing and one of its parent categories

C. The Search Refinement definition is not set up for the Women category

D. The Search Refinement definition is not set up for the Root category

ANSWER: D

Explanation:

The Search Refinement definition is not set up for the Root category is correct. A storefront category-navigation request has category context. When the shopper or merchandiser views the Clothing category directly, Commerce Cloud can apply the refinement configuration assigned to Clothing, which is why the Newness refinement appears and functions on that category page.

A keyword search, including a search for “Clothing,” is handled as a general storefront search rather than as navigation within the Clothing category. It therefore does not automatically inherit the refinement configuration from the category whose name happens to match the search phrase. For search-result pages without a selected category context, the applicable search refinements must be configured at the Storefront catalog’s root category. Configuring Newness at the root makes the refinement available for those general search results, while category-level configuration continues to govern category browsing behavior.

This distinction is important when designing the refinement experience: place refinements that should be available broadly in keyword search at the root, and use category assignments for refinements intended for specific browsing paths. See Salesforce’s Search Refinements documentation and the catalog documentation.

QUESTION NO: 23

A developer plans to use the /search_suggestion (Shop API) in a Storefront application and the following property must be set to do so

suggestion.product.image:view_type

What consideration should the developer in keep in mind to ensure that image data is returned correctly as part of search suggestions?

A. If the view_type is not set or if the view_type is unknown, the image properties are not part of the response.

B. If the view_type is not set or if the view_type is unknown, the image size of 'small' is used by default

C. If the view_type is not set or if the view_type is unknown, the image size of 'large' is used by default

ANSWER: A

Explanation:

If the view_type is not set or if the view_type is unknown, the image properties are not part of the response. The `suggestion.product.image:view_type` setting controls which configured product-image view type the Shop API uses when it builds product search-suggestion results. A view type must correspond to an image view configured for the site’s catalog. When a valid view type is supplied, the API can resolve the appropriate image data and include the image fields in each relevant product suggestion. This makes the setting particularly important for storefront type-ahead search, where the client may need a specific rendition such as a thumbnail-oriented image view. Developers should therefore configure the desired image view in Business Manager and ensure that the OCAPI configuration references that exact view-type identifier. The setting does not select a generic fallback image size; image data depends on successful resolution of the named view type. Salesforce’s OCAPI documentation describes Shop API configuration and endpoint behavior, while the B2C

Commerce developer documentation provides the broader catalog and image-model context. See Salesforce B2C Commerce OCAPI documentation and B2C Commerce Product Script API documentation.

QUESTION NO: 24

A merchant is selling a new product line of televisions. In order to deliver a good customer experience, the merchandising team wants the screen size to be incorporated into the search and navigation journey.

Which two things can the developer do to facilitate this for them?

Choose 2 answers

A. Create a new search refinement for a Boolean value true or false and label it "big screen."

B. Define a new searchable attribute for Screen Size.

C. Configure catalog-level search refinement definition for Screen Size.

D. Configure Screen Size threshold search refinement bucket definitions.

ANSWER: B C

Explanation:

Screen Size should first be represented by a product attribute that is enabled as searchable. This makes the attribute available to the B2C Commerce search index, allowing its values to participate in product-search behavior. The attribute should be assigned appropriate values on the television products, and the relevant search index should be rebuilt after the configuration or product data changes so the new data is reflected in storefront search results.

The developer must also configure a catalog-level search refinement definition for Screen Size. A search refinement definition exposes the indexed product attribute as a navigational facet, so shoppers can narrow a television result set by the available screen-size values. The catalog-level configuration is important because it controls how the refinement is presented and made available within the catalog's search and navigation experience. Together, a searchable product attribute and its catalog search-refinement definition provide both the indexed data and the storefront filtering capability needed by merchandisers.

Salesforce documents product search, indexing, and refinements in the B2C Commerce Search guide and the Search Refinements documentation.