DUMPSBOSS.

Adobe Experience Manager Sites Developer Expert

Adobe AD0-E137

Version Demo

Total Demo Questions: 7

Total Premium Questions: 70

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture	7
Topic 2, Development	42
Topic 3, Deployment	21
Total	70

QUESTION NO: 1

A developer needs to customize the handling of assets in a complex workflow model where different paths process assets based on their metadata and trigger specific external services.

Which approach is a best practice for implementing this solution?

A. Use out-of-the-box Adobe Experience Manager Workflow steps and configure them through the Workflow console to handle all metadata for asset processing, using conditions in the Workflow.

B. Write custom workflow process steps in Java to handle specific metadata conditions and integrate external services, using the Workflow API to manage dynamic branching logic.

C. Implement a content fragment model to pre-define asset metadata, using Workflows only for publishing the fragments after external service calls are completed.

ANSWER: B

Explanation:

Write custom workflow process steps in Java to handle specific metadata conditions and integrate external services, using the Workflow API to manage dynamic branching logic. This is the appropriate approach because AEM workflow process steps are designed to encapsulate business-specific processing that runs during execution of a workflow model. A custom OSGi service implementing AEM's `WorkflowProcess` interface can obtain the workflow payload, access the associated asset and its metadata through AEM APIs, apply the organization's routing or validation rules, and call an external service through a properly configured client. The process can then persist processing results or status information for subsequent workflow steps to use. Keeping this logic in a dedicated process step makes the workflow model responsible for orchestration while Java code owns reusable, testable integration and metadata-evaluation behavior. For multiple potential routes, the workflow model can use its routing constructs in combination with the results produced by the custom process, rather than embedding service-integration logic in the model itself. Adobe documents custom workflow process implementations through the `WorkflowProcess` interface and recommends deploying them as OSGi components so they are available as workflow process steps. See AEM Workflow Process Reference and WorkflowProcess API reference.

QUESTION NO: 2

A replication agent from Author to Publish has a blocked queue. The agent log records an HTTP 401 response from the configured Publish transport endpoint.

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

A. Correct the transport user credentials configured for the replication agent.

B. Use Test Connection after updating the agent configuration.

C. Increase the Retry Delay until the blocked queue clears.

D. Delete all queue entries before investigating the transport configuration.

E. Change the Dispatcher cache invalidation rules on Publish.

ANSWER: A B

Explanation:

An HTTP 401 response indicates that the transport endpoint rejected the credentials supplied by the replication agent. The developer should correct the configured transport user credentials and then use the agent's Test Connection capability to verify that Author can authenticate to the configured Publish endpoint.

Increasing Retry Delay does not fix invalid credentials, and deleting queue entries discards pending replication requests without resolving the delivery failure. Dispatcher cache invalidation settings do not control authentication between an Author replication agent and its Publish transport endpoint.

QUESTION NO: 3

A new Publish environment is deployed behind Dispatcher. Authors activate pages successfully, but visitors continue to receive stale cached HTML. The project must invalidate the relevant cached pages when content is activated without disabling normal Dispatcher caching.

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

A. Configure a Dispatcher Flush agent on Author that targets the Dispatcher endpoint.

B. Configure the Dispatcher farm `/invalidate` rules to match the cached page files that must be invalidated.

C. Increase the Dispatcher cache TTL for all HTML files.

D. Configure an Author-to-Publish replication agent with a shorter Retry Delay.

E. Allow Dispatcher to cache POST responses for page activation requests.

ANSWER: A B

Explanation:

A Dispatcher Flush agent on Author sends invalidation requests to the Dispatcher endpoint when content is activated. In addition, the Dispatcher farm configuration must contain appropriate `/invalidate` rules that permit invalidation of the cached page files, such as matching the relevant `*.html` content. Together, these settings allow activated content to invalidate stale cache entries while preserving cached delivery for subsequent requests.

Increasing the cache TTL does not invalidate a page on activation. A replication agent delivers repository content to Publish but does not invalidate Dispatcher by itself. Caching POST responses is not an appropriate solution for page activation.

QUESTION NO: 4

A team has created Content Fragment Models under `/conf/wknd`. A headless application needs to query fragments based on those models through AEM GraphQL, but no schema is available for the project configuration.

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

A. Enable the Content Fragment Models used by the headless application.

B. Configure a GraphQL endpoint for the project configuration in Configuration Browser.

C. Publish an individual Content Fragment before enabling its model.

D. Add the Content Fragment Models to an editable template policy.

E. Create a resource-type Sling servlet to generate the GraphQL schema.

ANSWER: A B

Explanation:

Content Fragment Models must be enabled before AEM can use them for fragment authoring and GraphQL schema generation. The project configuration also requires a GraphQL endpoint, configured through the Configuration Browser, so that the endpoint exposes schemas for the enabled models in that configuration.

Publishing individual Content Fragments does not create a schema. An editable template policy controls page authoring and does not enable GraphQL. A custom Sling servlet is unnecessary because AEM provides the GraphQL endpoint capability.

QUESTION NO: 5

A developer is adding a reference script to a third-party analytics tool. The script needs to be editable since it is being modified on a schedule outside of the team's development cycle.

How should the developer complete this task?

A. Add a new component with the script in the HTL.

B. Add the script to the page policy on the template.

C. Add a **Generic Analytics Snippet**.

ANSWER: C

Explanation:

Add a Generic Analytics Snippet. is correct because AEM provides the Generic Analytics Snippet mechanism specifically for adding third-party analytics markup or script references without embedding the changing script directly in application code. The snippet is managed as content/configuration in AEM, so an authorized author or administrator can update it when the analytics provider changes its required implementation. This separates externally maintained analytics code from the development and deployment lifecycle, avoiding a code release merely to replace a tracking tag or script URL.

The snippet can be configured for the appropriate site context and is rendered by AEM where analytics scripts are included. This makes it suitable for vendor tags that must be adjusted independently, while retaining AEM governance over who can edit and publish the change. Adobe documents Generic Analytics Snippets as a supported capability for including custom analytics code and managing it through the AEM interface. See Adobe's Integrating with Adobe Analytics and Analytics providers documentation and the Page Properties documentation for the author-managed site configuration model.

QUESTION NO: 6

A developer needs to define **vanity URLs** in the dispatcher configuration file.

Under which property would this be defined?

A. /vanity_url

B. vanityUrls

C. /vanityUrls

D. /vanity_urls

ANSWER: D

Explanation:

The correct property is **/vanity_urls**. In an AEM Dispatcher farm configuration, this section configures Dispatcher's vanity-URL handling. AEM authors define vanity paths on page properties, while Dispatcher obtains the published vanity-path list from AEM and stores it in a local file. This lets Dispatcher automatically allow requests for the configured vanity paths while retaining a restrictive filter configuration for paths that are not registered as vanities.

The **/vanity_urls** block is placed in the relevant farm and includes settings such as **/url**, which identifies the AEM vanity URL service endpoint; **/file**, which specifies where Dispatcher saves the downloaded vanity URL list; and **/delay**, which controls how often Dispatcher refreshes that list. The Dispatcher process must have permission to write to the configured local file. Adobe documents this configuration as the Dispatcher mechanism for caching and auto-allowing AEM vanity URLs, with the service typically exposed through `/libs/granite/dispatcher/content/vanityUrls.html`.

See Adobe's Dispatcher configuration documentation for the **/vanity_urls** section and its supported child properties: Configure Dispatcher. Additional implementation guidance for vanity URLs is available in AEM Vanity URLs.

QUESTION NO: 7

An Adobe Experience Manager team is using an additional DEV environment in Adobe Experience Manager as a Cloud Service as their UAT environment. An AEM architect is asked to configure a dedicated URL endpoint to be used as a preview service for the same environment.

Which configuration will accomplish this task?

A. /apps//osgiconfig/author.uat/.cfg.json in ui.apps.structure project

B. /apps//osgiconfig/config.publish.uat/.cfg.json in ui.config project

C. /apps//osgiconfig/config.uat.author/.cfg.json in ui.config project

D. /apps//osgiconfig/config.preview.dev/.cfg.json in ui.config project

ANSWER: D

Explanation:

The configuration path `/apps/<project name>/osgiconfig/config.preview.dev/<persistent identity>.cfg.json` in the `ui.config` project is correct because AEM as a Cloud Service assigns a distinct `preview` run mode to the Preview Service. OSGi configurations intended for that service must therefore be packaged in a `config.preview` run-mode folder. Because the environment being used for UAT is an additional environment of the DEV type, the environment-specific run mode is `dev`, producing the combined folder name `config.preview.dev`. AEM resolves the applicable run-mode configuration at deployment and applies it to the Preview Service endpoint for that DEV environment. The configuration belongs in the `ui.config` module, which is the standard AEM project module for deployable OSGi configuration content. This setup lets the team apply settings specifically to the preview endpoint without treating “UAT” as an AEM Cloud Service environment run mode. Adobe documents the supported OSGi run-mode folder conventions, including Preview Service and environment-type combinations, in its OSGi configuration documentation. Environment types and their Cloud Manager role are described in the Manage environments documentation.