DUMPSBOSS.

Adobe Experience Manager Developer Exam

Adobe AD0-E134

Version Demo

Total Demo Questions: 9

Total Premium Questions: 94

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture	10
Topic 2, Development	60
Topic 3, Deployment	21
Topic 4, Maintenance	3
Total	94

QUESTION NO: 1

A Sling Model adapts from `SlingHttpServletRequest`. It must render an authored component property named `ctaLabel` and use the title of the current page to construct a tracking value.

Which two injections are appropriate for this model? (Choose two.)

A. `@ValueMapValue(name = "ctaLabel") private String ctaLabel;`

B. `@ScriptVariable private Page currentPage;`

C. `@OSGiService private Page currentPage;`

D. `@Self private String ctaLabel;`

E. `@ChildResource private Page currentPage;`

ANSWER: A B

Explanation:

`@ValueMapValue(name = "ctaLabel")` is appropriate for obtaining the authored property from the component resource associated with the request. The ValueMap injector reads persisted component properties and converts them to the requested Java type when possible.

`@ScriptVariable Page currentPage` is appropriate for accessing the current page from the HTL script context. It gives the model the page object needed to obtain values such as the page title. `@OSGiService` is intended for OSGi services, not page context, and a self-injected request does not itself provide the current page title or authored property.

QUESTION NO: 2

A developer is creating an editable template for a site. Every page created from the template must contain the same Header component, and page authors must not remove or edit that Header. A separate Text component should be present when a page is created, but authors must be able to edit or remove it on individual pages.

Which two template authoring approaches should the developer use? (Choose two.)

A. Place the Header component in the template structure.

B. Place the Header component in the template initial content.

C. Place the Text component in the template initial content.

D. Place the Text component in the template structure.

E. Place both components in the template policy.

ANSWER: A C

Explanation:

The Header belongs in the template structure. Components placed in the structure are inherited by pages created from the editable template and are locked for page authors. This is appropriate for common site elements whose presence and configuration must be controlled centrally.

The Text component belongs in the template initial content. Initial content is copied to a page when the page is created, after which it becomes normal page content that authors can edit, move, or remove. Placing the Text component in structure would incorrectly lock it on every page.

QUESTION NO: 3

Where should an AEM Developer add a front end dependency?

A. config.json

B. settlngs.xml

C. package.json

D. vault.xml

ANSWER: C

Explanation:

package.json is the correct location for declaring a front-end dependency in an AEM project. Modern AEM Maven archetype projects commonly include a `ui.frontend` module that uses Node.js tooling, such as npm, to build client-side JavaScript, CSS, and related assets. Within that module, `package.json` is the npm project manifest: it records the packages required to build or develop the front end and the scripts used for tasks such as compilation, linting, and production builds.

A runtime package needed by the browser application belongs in the `dependencies` section, while tooling used only during development or the build process—such as linters, bundlers, or test frameworks—typically belongs in `devDependencies`. Developers normally add a package with an npm command, for example `npm install package-name` or `npm install --save-dev package-name`; npm then updates `package.json` and its lock file. Keeping dependencies declared in this manifest makes front-end builds reproducible for local development and CI pipelines.

Adobe's AEM Project Archetype documentation describes the front-end module and its Node-based build workflow. See the AEM Project Archetype usage documentation and the npm package.json documentation.

QUESTION NO: 4

A project requires sharing information between SPA components. What is the least complex approach to achieve that objective?

A. Utilize model props to drill down and access or set the state on desired components.

B. Customize and extend the container component to leverage the object hierarchy.

C. Implement a state library in order to share component states.

D. Centralize the logic and broadcast to the necessary components.

ANSWER: A

Explanation:

Utilize model props to drill down and access or set the state on desired components. is the least complex approach when the components that need shared information have a practical parent-child relationship. In an SPA, data can be held by the nearest common parent and passed down through component props. When a nested component needs to request a state change, the parent can pass a callback through props as well. This keeps ownership of the state explicit, makes the data flow predictable, and avoids adding application-wide infrastructure before it is needed.

For AEM SPAs, mapped components receive model data and properties as part of their rendering integration with the AEM SPA Editor. Passing the relevant values and handlers through the existing component hierarchy fits naturally with this model and keeps the implementation small. This approach is especially appropriate for localized sharing among a limited number of related components; it can be refactored later if the component tree or sharing requirements become substantially more complex. Adobe's SPA Editor documentation describes the component-model integration, while React's documentation explains passing props through a component hierarchy: Adobe Experience Manager SPA Editor overview and React: Passing Props to a Component.

QUESTION NO: 5

A site must render a call-to-action label in the language selected by the visitor. Content authors and translators must be able to maintain the translated labels without changing component code.

Which approach should the developer use in the component HTL?

A. Use `${'Read more' @ i18n}` and maintain the key in an AEM i18n dictionary.

B. Hard-code every translated label in the component HTL file.

C. Store the translated label only as a property on each page.

D. Create a Content Fragment for every translated interface label.

ANSWER: A

Explanation:

The developer should use the HTL internationalization expression, for example `${'Read more' @ i18n}`. Sling resolves the key against the applicable AEM i18n dictionaries for the request locale, allowing translators to maintain language-specific values independently from the component implementation.

Hard-coding translated strings in HTL requires code changes for translation updates. Reading a page property does not provide the normal dictionary-based translation workflow, and using a Content Fragment is unnecessary for a reusable interface label.

QUESTION NO: 6

On package install content that is already present in the repos must not be overwritten and if not present in the repos it must not be removed.

Which import mode should the developer use?

A. update

B. replace

C. merge

ANSWER: C

Explanation:

The correct import mode is **merge**. In AEM package installation, merge is intended for additive, non-destructive imports: it adds content from the package where it does not yet exist, while leaving repository content that already exists unchanged. It also does not delete existing repository content merely because that content is absent from the package. This behavior directly meets the requirement to protect pre-existing repository content from being overwritten and to retain repository content that the package does not contain.

This mode is useful when a package supplies baseline or supplemental content but the target environment may already contain authored, environment-specific, or manually maintained content. A developer can therefore install the package without replacing those repository values or removing nodes that are outside the package's current contents. Adobe documents package import modes in its Package Manager guidance, and the underlying FileVault import behavior is documented by Apache Jackrabbit. See Adobe Experience Manager Package Manager documentation and Apache Jackrabbit FileVault import modes.

QUESTION NO: 7

A developer creates an editable template with a Layout Container. A custom component is deployed successfully and appears in the Components browser, but authors cannot add it inside the Layout Container when editing pages created from the template.

Which two actions can make the component available to authors in that container? (Choose two.)

A. Add the component explicitly to the Layout Container policy's allowed components.

B. Add a component group containing the component to the Layout Container policy.

C. Add additional fields to the component dialog.

D. Publish the editable template again without changing its policy.

E. Set the component's `componentGroup` property without updating the policy.

ANSWER: A B

Explanation:

The developer can add the custom component to the Layout Container policy's allowed components. The component can also be made available by selecting a component group in the policy that includes the custom component. The policy controls which components authors can place in a particular container.

Adding only a `componentGroup` property to the component does not automatically allow it in every container; the applicable policy must allow that group or component. Editing the component dialog affects authoring fields rather than container eligibility, and publishing the template does not change its allowed-components policy.

QUESTION NO: 8

The OSGi configuration is added to a runmode specific configuration "config.author.staging" in AEM as a Cloud Service. The application fails to read the configuration.

What is a possible cause of this issue?

A. The custom OSGi configuration runmode used (i.e., "config.author.staging") is not supported in AEM as a Cloud service.

B. OSGi configuration runmodes cannot be installed automatically on AEM as a Cloud Service. We need to install them as a package using the Package manager.

C. AEM as a Cloud service does not support OSGi configuration runmodes.

D. Only < service > specific OSGi configuration runmodes like "config.author" or "config.publish" are supported in AEM as a Cloud service.

ANSWER: A

Explanation:

The custom OSGi configuration runmode used (i.e., "config.author.staging") is not supported in AEM as a Cloud service. In AEM as a Cloud Service, OSGi configurations are deployed as part of the application code and their configuration-folder names must use the Cloud Service run-mode conventions. The service run mode is expressed through supported values such as `author`, `publish`, or `preview`; environment targeting uses the documented environment names, including `dev`, `stage`, and `prod`. Therefore, `staging` is not a recognized environment run mode. A configuration under `config.author.staging` does not match the expected naming convention and may not be selected for the target deployment. The configuration should instead use the applicable supported folder name, such as `config.author.stage` when targeting the Stage author service, or a service-only folder when no environment-specific variation is required. Adobe also supports environment variables and Cloud Manager configuration mechanisms for values that vary by environment. See Adobe's OSGi configuration run-mode documentation and its environment-specific configuration guidance.

QUESTION NO: 9

A developer needs to create a project based on AEM Project Archetype with a specific AEM as a Cloud Service SDK version on the local environment.

Which two properties must be defined when creating this project? (Choose two.)

A. aemVersion=cloud

B. sdkVersion=2022.5.7575.20220530T152407Z-220401

C. sdkVersion=latest

D. aemVersion=latest

E. aemVersion=2022.5.7575.20220530T152407Z-220401

ANSWER: A B

Explanation:

`aemVersion=cloud` identifies Adobe Experience Manager as a Cloud Service as the intended AEM target for the generated project. This setting ensures that the AEM Project Archetype applies the Cloud Service-oriented project configuration and compatibility choices rather than targeting an on-premise AEM version. The archetype's `aemVersion` property accepts `cloud` for this purpose.

`sdkVersion=2022.5.7575.20220530T152407Z-220401` explicitly pins local development to the requested AEM as a Cloud Service SDK release. Supplying the complete SDK build version is appropriate when a project must be created and tested against a specific local SDK level, helping developers align local behavior and dependencies with that selected release. The SDK version is a distinct concern from the Cloud Service target selection, so both properties are needed: one declares the deployment model and the other selects the exact local SDK build.

Adobe documents the archetype properties, including the Cloud Service target value, in the AEM Project Archetype documentation. Adobe's AEM as a Cloud Service SDK documentation describes using SDK releases for local development.