Adobe AD0-E134 Practice Questions

Shared by Townsend on 19-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

An AEM server is overloaded with too many concurrently running workflows. The developer decides to reduce the number of concurrent workflows.

What should be configured to reduce the number of concurrent workflows?

Options:

A- The number of threads in Scheduler
B- The number of threads in Apache Felix Jetty Http Service
C- Launchers for each workflow
D- Maximum Parallel Jobs in OSGI console

Answer:

D

Explanation:

Maximum Parallel Jobs is a configuration property that controls how many workflows can run concurrently on an AEM instance. Reducing this value can help to avoid overloading the server with too many workflows.

Question 2

Question Type: MultipleChoice

Where should an AEM Developer add a front end dependency?

Options:

A- config.json
B- settlngs.xml
C- package.json
D- vault.xml

Answer:

C

Explanation:

An AEM Developer should add a front-end dependency in the package.json file. The package.json file is a standard configuration file for managing dependencies in JavaScript projects, including those using npm or Yarn as package managers.

Here's how to add a front-end dependency:

Open the package.json File: This file is typically located at the root of your project.

Add the Dependency: Add the required front-end dependency under the dependencies or devDependencies section. For example, to add lodash as a dependency:

{

'name': 'my-aem-project',

'version': '1.0.0',

'dependencies': {

'lodash': '^4.17.21'

}

}

Install the Dependency: Run the following command to install the new dependency:

npm install

Verify Installation: Ensure that the dependency has been added to the node_modules directory and is listed in the package-lock.json file.

The package.json file is the central place to manage all front-end dependencies, making it easy to track, update, and share dependencies across the development team.

npm Documentation

Managing Dependencies in AEM Projects

These steps ensure that the front-end dependencies are managed efficiently and consistently within the AEM project.

Question 3

Question Type: MultipleChoice

The following anchor tag is not resolving:

{item.name}

Upon further inspection the developer notices that the link has no .html appended to the end of the URL

What could be a potential fix for the issue?

A)

{item.name}

B)

{item.name}

C)

{item.name}

D)

{item.name}

Options:

A- Option A
B- Option B
C- Option C
D- Option D

Answer:

B

Explanation:

Option B is a potential fix for the issue. Option B uses the data-sly-attribute block statement to

add the href attribute to the anchor tag. The data-sly-attribute statement uses an expression to append ''.html'' to the item.path value. This way, the link will have the correct extension and will resolve to the corresponding page. Reference: https://experienceleague.adobe.com/docs/experience-manager-htl/using-htl/htl-block-statements.html?lang=en#data-sly-attribute

Question 4

Question Type: MultipleChoice

Which notable change should the developer be aware of in order to distribute content in AEM as a Cloud Service?

Options:

A- Removal of publish content tree workflow
B- Removal of replication agents
C- Removal of tree activation
D- Removal of touch Ul

Answer:

B

Explanation:

In AEM as a Cloud Service, one notable change that developers should be aware of is the removal of replication agents. Instead of the traditional replication agents used in on-premise or AEM Managed Services environments, AEM as a Cloud Service uses a different mechanism to distribute content.

Key points about this change:

Content Distribution:

In AEM as a Cloud Service, content distribution is managed through automated processes that do not rely on manually configured replication agents.

The cloud infrastructure handles the replication and distribution of content to publish instances.

Publishing Model:

The publishing process is designed to be more seamless and integrated within the cloud environment, leveraging the underlying cloud services for scalability and efficiency.

Developers and administrators do not need to manually manage replication agents, simplifying the content deployment process.

Benefits:

This change reduces the complexity of content distribution in AEM projects.

It ensures higher reliability and performance by leveraging the capabilities of the cloud infrastructure.

AEM as a Cloud Service Documentation

Content Distribution in AEM as a Cloud Service

By understanding this change, developers can effectively manage content distribution in AEM as a Cloud Service and leverage the new mechanisms provided by the cloud platform.

Question 5

Question Type: MultipleChoice

An AEM application development team is assigned a task to create an Event-Driven Data Layer implementation for an Analytics solution.

Which Adobe recommended best practice should the developer choose?

Options:

A- Use Adobe Experience Platform's data layer to integrate with AEM.
B- Create a custom data layer and add each component template, and its properties to the data layer
C- Use Adobe Client Data Layer and integrate with Core components.
D- Create an Adobe Cloud Service configuration to use third-party tool's data layer.

Answer:

C

Explanation:

Adobe Client Data Layer is a JavaScript library that provides a standardized way to collect, structure, and manage data on a web page. It can be used to implement an event-driven data layer for analytics solutions. It integrates with Core components and allows authors to configure

data layer properties for each component. It also supports custom events and data sources. Reference: https://experienceleague.adobe.com/docs/experience-manager-core-components/using/developing/data-layer.html?lang=en https://github.com/adobe/adobe-client-data-layer

Question 6

Question Type: MultipleChoice

Which Maven plugin checks if all the requirements declarations made in OSGi bundles are satisfied by the capabilities declarations of other bundles included in the Maven project?

Options:

A- maven-enforcer-plugin
B- femaven-assembly-plugin
C- content-package-maven-plugin
D- aemanalyser-maven-plugin

Answer:

D

Explanation:

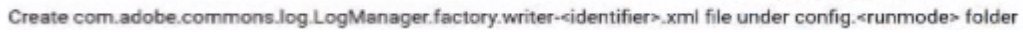
The aemanalyser-maven-plugin is a Maven plugin that checks if all the requirements declarations made in OSGi bundles are satisfied by the capabilities declarations of other bundles included in the Maven project. This plugin ensures that the OSGi bundles are consistent and can be resolved at runtime. The plugin also checks for other issues such as API compatibility, package versioning, and bundle start order. Reference: https://experienceleague.adobe.com/docs/experience-manager-cloud-service/implementing/developing/aem-project-content-package-structure.html?lang=en#build-analyzer-maven-plugin https://github.com/adobe/aemanalyser-maven-plugin

Question 7

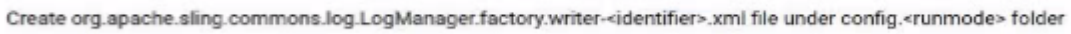
Question Type: MultipleChoice

How should a developer create a custom log configuration?

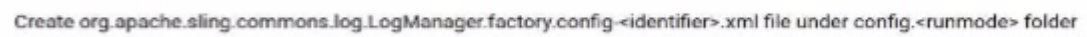
A)



B)



C)



Options:

A- Option A
B- Option B
C- Option C

Answer:

B

Explanation:

To create a custom log configuration in AEM, the developer should create an OSGi configuration file for the org.apache.sling.commons.log.LogManager.factory.writer component. This configuration file should be placed under the appropriate runmode-specific folder within the config directory.

Here is the detailed process:

Create Configuration File:

Create a new XML file named org.apache.sling.commons.log.LogManager.factory.writer-<identifier>.xml in the config.<runmode> folder.

The <identifier> can be any unique identifier for your custom log configuration.

Specify Configuration:

Add the necessary configuration properties within this XML file. For example:

<?xml version='1.0' encoding='UTF-8'?>

<jcr:root xmlns:sling='http://sling.apache.org/jcr/sling/1.0'

xmlns:jcr='http://www.jcp.org/jcr/1.0'

jcr:primaryType='sling:OsgiConfig'

org.apache.sling.commons.log.file='/var/log/custom.log'

org.apache.sling.commons.log.level='debug'

org.apache.sling.commons.log.pattern='\{0\}'

org.apache.sling.commons.log.names='[com.myproject.custom]' />

Deploy Configuration:

Save the configuration file and deploy it to the appropriate environment. The logging system in AEM will pick up this configuration and create a custom log file with the specified settings.

Adobe Experience Manager Logging

Apache Sling Logging

Question 8

Question Type: MultipleChoice

Which property under /cache on dispatcher.any file identifies the directory where cached files are stored?

Options:

A- /invalidate
B- /statfile
C- /docroot
D- /cacheroot

Answer:

D

Explanation:

The /cacheroot property under /cache in the dispatcher.any file identifies the directory where cached files are stored. It is a relative or absolute path to the cache root directory. The dispatcher creates a subdirectory for each virtual host under this directory and stores the cached files there. Reference:

https://experienceleague.adobe.com/docs/experience-manager-dispatcher/using/configuring/dispatcher-configuration.html?lang=en#cache

Question 9

Question Type: MultipleChoice

A developer needs to create an OSGI service that is able to read a list of spoken languages from the configuration of the service. The configuration file tanguageServicelmplefgjson' already exisls:

```
{
    "languages":[
      "French",
      "German",
      "Italian"
    ]
```

Which snippet should the developer use lo read the OSGi configurations?

A)

```
@Component(
    service = { LanguageService.class }
)
@Designate(ocd = LanguageServiceImpl.Config.class)
public class LanguageServiceImpl implements LanguageService {

...

@ObjectClassDefinition(
        name = "Sample - Language Service",
        description = "OSGi Service providing information about languages"
    )
    @interface Config {

        @Property(
                label = "Sample Languages",
...                                          lan
}                                            h",
```

B)

```
@Component(
    service = { LanguageService.class }
)
@Designate(ocd = LanguageServiceImpl.Config.class)
public class LanguageServiceImpl implements LanguageService {

...

@ObjectClassDefinition(
        name = "Sample - Language Service",
        description = "OSGi Service providing information about languages"
    )
    @interface Config {

        @AttributeDefinition(
                name = "Sample Languages",
                description = "List of spoken languages"
        )
        String[] languages() default { "English", "Japanese" };

    }

    private String[] languages;

    @Activate
    protected void activate(Config config) {
        this.languages = config.languages();
```

Options:

A- Option A
B- Option B

Answer:

B

Explanation:

To read a list of spoken languages from the configuration of an OSGi service, the correct snippet to use is Option B. This snippet demonstrates how to define and read the configuration properties using the OSGi R7 annotations (@ObjectClassDefinition and @AttributeDefinition), which are the recommended way for defining OSGi configurations in modern AEM projects.

Here is the detailed explanation of the snippet:

Option B Snippet Explanation:

Component Definition:

@Component(

service = { LanguageService.class }

)

@Designate(ocd = LanguageServiceImpl.Config.class)

public class LanguageServiceImpl implements LanguageService {

This defines an OSGi component and designates a configuration class for it.

Configuration Interface:

@ObjectClassDefinition(

name = 'Sample - Language Service',

description = 'OSGi Service providing information about languages'

)

@interface Config {

@AttributeDefinition(

name = 'Sample Languages',

description = 'List of spoken languages'

)

String[] languages() default { 'English', 'Japanese' };

}

This defines the configuration interface with annotations to describe the configuration properties. The languages attribute is defined with a default value of {'English', 'Japanese'}.

Activate Method:

private String[] languages;

@Activate

protected void activate(Config config) {

this.languages = config.languages();

}

The activate method reads the configuration values and assigns them to the instance variable languages when the service is activated.

Here is the complete Option B code:

@Component(

service = { LanguageService.class }

)

@Designate(ocd = LanguageServiceImpl.Config.class)

public class LanguageServiceImpl implements LanguageService {

@ObjectClassDefinition(

name = 'Sample - Language Service',

description = 'OSGi Service providing information about languages'

)

@interface Config {

@AttributeDefinition(

name = 'Sample Languages',

description = 'List of spoken languages'

)

String[] languages() default { 'English', 'Japanese' };

```
}

private String[] languages;

@Activate

protected void activate(Config config) {

this.languages = config.languages();

}

// Additional methods to use the languages array

}
```

By using this approach, you ensure that your OSGi service can dynamically read and use the list of spoken languages specified in its configuration, making it adaptable to different environments and requirements.

OSGi R7 Annotations

Adobe Experience Manager - OSGi Configuration

Question 10

Question Type: MultipleChoice

What is the correct order of resolution of OSGi configuration at Runtime?

Options:

A- 1. Modifying a configuration in /apps will take immediate effect
2. Modifying a configuration in the Web console will take immediate effect as it takes precedence at runtime.
3. Modifying a configuration in /libs will take immediate effect, unless it is masked by a configuration in /apps.

B- 1. Modifying a configuration in the Web console will take immediate effect as it takes precedence at runtime.
2. Modifying a configuration in /apps will take immediate effect
3. Modifying a configuration in /libs will take immediate effect, unless it is masked by a configuration in /apps.

C- 1. Modifying a configuration in /libs will lake immediate effect, unless it is masked by a configuration in /apps O 2. Modifying a configuration in /apps will take immediate effect
3. Modifying a configuration in the Web console will take immediate effect as it takes

precedence at runtime.
D- 1. Modifying a configuration in the Web console will take immediate effect as it takes precedence at runtime.
2. Modifying a configuration in /libs will take immediate effect, unless it is masked by a configuration in /apps.
3. Modifying a configuration in /apps will lake immediate effect.

Answer:

B

Explanation:

The order of resolution for OSGi configurations at runtime in AEM is as follows:

Web Console Configuration:

Changes made through the OSGi Web Console take immediate effect and have the highest precedence. This is because these configurations are considered the most direct way to adjust OSGi settings at runtime.

/apps Configuration:

Configurations in the /apps directory take precedence over those in the /libs directory. This allows custom configurations to override the default configurations provided by AEM.

/libs Configuration:

Configurations in the /libs directory are the default configurations provided by AEM. They are overridden by any configurations in the /apps directory.

By understanding this order, developers can effectively manage and prioritize their OSGi configurations to ensure the desired behavior of their AEM instances.

Question 11

Question Type: MultipleChoice

Which OSGi configuration values can be used in an AEM as a Cloud Service Implementation?

Options:

A- Inline, secret, runmode-specific

B- Inline.restricted,runmode-specific
C- Inline, restricted. environment-specific
D- Inline, secret environment-specific

Answer:

D

Explanation:

In AEM as a Cloud Service, the OSGi configuration values that can be used include Inline, secret, and environment-specific. These configurations provide flexibility and security for managing environment-specific settings and sensitive information.

Inline Configurations: Inline configurations are directly embedded within the code and are typically used for straightforward configurations.

Secret Configurations: Secret configurations are used to securely store sensitive information, such as passwords and API keys. These configurations are managed separately to ensure security.

Environment-Specific Configurations: Environment-specific configurations allow you to tailor settings for different environments (e.g., development, staging, production) without changing the underlying codebase.

Example of using these configurations:

Inline Configuration:

{

'service.url': 'https://api.example.com'

}

Secret Configuration: Managed through Adobe IMS and not directly embedded in the code.

Environment-Specific Configuration:

{

'runmode': 'dev',

'service.timeout': '30'

}

Managing OSGi Configurations in AEM

Adobe IMS for Secrets Management

Question 12

Question Type: MultipleChoice

A new component called Page Headline needs to be implemented. The only difference to the title component delivered by Adobe's WCM core components is that the text needs to be taken from the current page title instead of jcr.title.

How should a developer implement this request?

A)

1. Create custom component

2. Implement Sling Modal as follows

```
@Model(adaptables = SlingHttpServletRequest.class, adapters = Title.class, resourceType = "myproject/components/pageHeadline")
public class PageHeadline  {

    @ScriptVariable
    private Page currentPage;

    @Override
    public String getText() {
        return currentPage.getTitle();
    }
```

B)

1. Create proxy component from the core title component

2. Implement sling Model as follows

```
@Model(adaptables = Resource.class, adapters = Title.class, resourceType = "myproject/components/pageHeadline")
public class PageHeadline implements Title {

    @ScriptVariable
    private Page currentPage;

    @Self @Via(type = ResourceSuperType.class)
    private Title title;

    @Override
    public String getText() {
        return currentPage.getTitle();
```

C)

```
@Model(adaptables = SlingHttpServletRequest.class, adapters = Title.class, resourceType = "myproject/components/pageHeadline")
public class PageHeadline implements Title {

    @ScriptVariable
    private Page currentPage;

    @Self @Via(type = ResourceSuperType.class)
    private Title title;

    @Override
    public String getText() {
        return currentPage.getTitle();
    }

    @Override
    public String getType() {
        return title.getType();
    }
}
```

1. Create proxy component from the core title component

2. Implement Sling Model as follows

```
@Model(adaptables = SlingHttpServletRequest.class, adapters = Title.class, resourceType = "myproject/components/pageHeadline")
public class PageHeadline implements Title {

    @ScriptVariable
    private Page currentPage;

    @Self @Via(type = ResourceSuperType.class)
    private Title title;

    @Override
    public String getText() {
        return currentPage.getTitle();
    }

    @Override
    public String getType() {
        return title.getType();
    }
}
```

Options:

A- Option A
B- Option B
C- Option C

Answer:

C

Explanation:

To implement a new component called 'Page Headline' which takes the text from the current page title instead of jcr:title, you should create a proxy component from the core title component and implement a Sling Model accordingly. Option C demonstrates the correct approach to achieve this functionality.

Here is a detailed explanation of Option C:

Create Proxy Component:

Create a new component in your project that will act as a proxy to the core title component. This involves creating a new component node in the repository that inherits from the core title component.

Example path: /apps/myproject/components/pageHeadline with sling:resourceSuperType set to core/wcm/components/title/v2/title.

Implement Sling Model:

Implement a Sling Model that adapts from SlingHttpServletRequest and Title.class, ensuring it overrides the text fetching logic to retrieve the title from the current page's title.

The model will use @ScriptVariable to inject the current page and @Self to access the core title component's logic.

@Model(adaptables = SlingHttpServletRequest.class, adapters = Title.class, resourceType = 'myproject/components/pageHeadline')

public class PageHeadline implements Title {

```
@ScriptVariable

private Page currentPage;

@Self @Via(type = ResourceSuperType.class)

private Title title;

@Override

public String getText() {

return currentPage.getTitle();

}

@Override

public String getType() {

return title.getType();

}

}
```

@Model Annotation: Specifies that this class is a Sling Model and adapts from SlingHttpServletRequest. It is also an adapter for Title.class and applies to the myproject/components/pageHeadline resource type.

@ScriptVariable: Injects the current Page object, which allows access to the current page's properties.

@Self @Via(type = ResourceSuperType.class): Injects the core title component, allowing the reuse of existing logic.

getText() Method: Overrides the getText() method to return the title from the current page instead of the jcr:title.

getType() Method: Delegates to the core title component's getType() method to maintain existing functionality.

This approach leverages the power of Sling Models and the core components to extend functionality while keeping the implementation clean and maintainable.

Adobe Experience Manager - Core Components

Sling Models Documentation