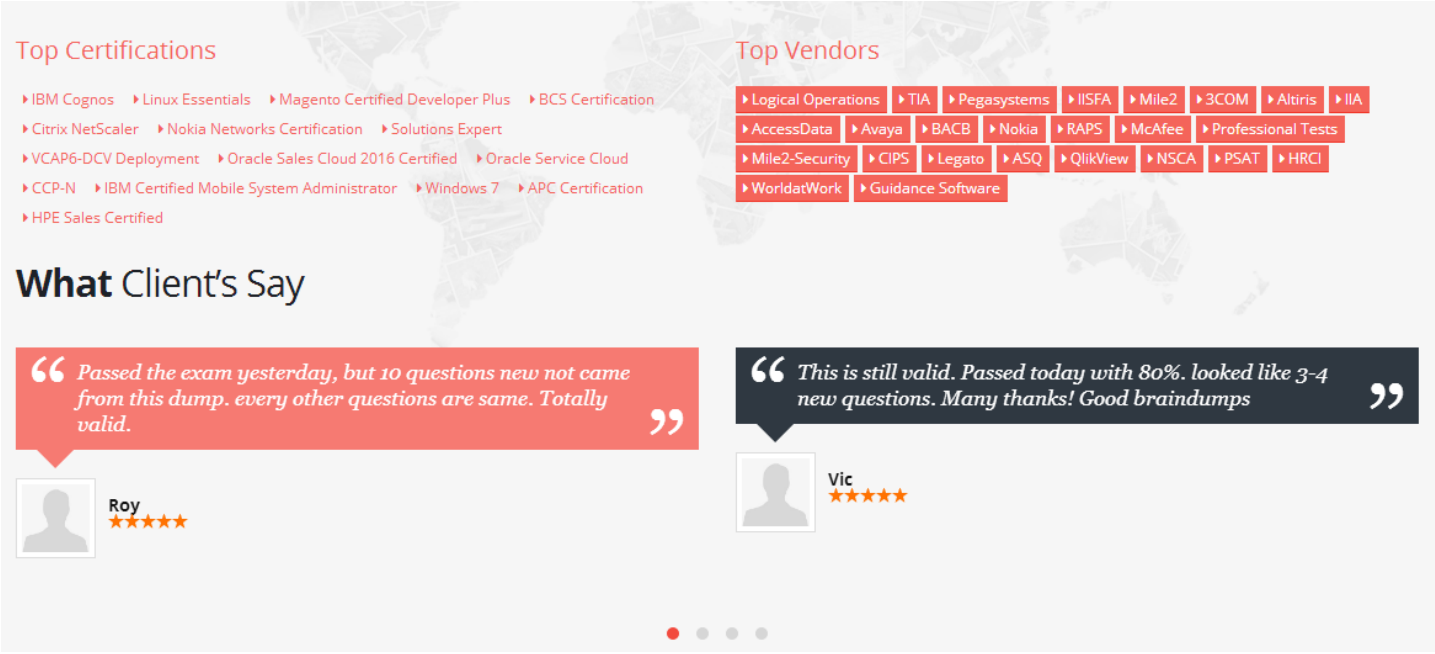
Lead2passExam



http://www.lead2passexam.com/

Available Exam Cram and Valid Dumps - Lead2Pass Exam

Exam : **NAS-C01**

Title : SnowPro Specialty - Native Apps

Vendor : Snowflake

Version : DEMO

NO.1 You are developing a Snowflake Native App that allows consumers to enhance their existing customer data with enrichment data provided by your app. The app relies on secure data sharing. To minimize data duplication and maximize query performance within the consumer's account, which of the following approaches offers the MOST optimized and secure way to deliver the enrichment data?

A. Create a staging table in the provider account, load the enrichment data into it, and grant the consumer account SELECT privileges on this staging table. The consumer then creates their own table and copies the data.

B. Utilize Snowflake Secure Data Sharing to directly share the enrichment data tables from the provider account to the consumer account. The consumer can then create views on the shared tables or query them directly.

C. Develop a Snowflake UDF (User-Defined Function) that, when called in the consumer account, retrieves the enrichment data from an external API endpoint (maintained in the provider account) and returns it.

D. Create a secure view within the application package itself, which joins the application's enrichment data with consumer-provided data via an API integration. The consumer queries this secure view.

E. Use Snowflake Data Marketplace listing to provide enrichment data.

Answer: B

Explanation:

Option B (Utilize Snowflake Secure Data Sharing) is the most optimized and secure approach. Secure Data Sharing avoids data duplication, minimizes latency, and allows the consumer to directly access the enrichment data without needing to copy or move it. It leverages Snowflake's native sharing capabilities, ensuring security and governance. Option A involves data duplication. Option C introduces external API dependency and potential performance bottlenecks. Option D mixes consumer data with the application code, which is not optimal and less secure. Option E is suitable for data discoverability and monetization, not for application-specific, customized data enrichment.

NO.2 A Snowflake Native App developer is building an application package that contains multiple versions. They want to ensure that a consumer upgrading from version 1.0 to version 2.0 MUST execute a specific set of SQL scripts as part of the upgrade process to migrate existing application dat a. Which Snowflake Native App feature should they leverage to achieve this?

A. Application Role-Based Access Control (RBAC)

B. A post-install script defined in the setup script.

C. A version change handler defined within the application package.

D. Snowflake Streams and Tasks to monitor the application version and trigger the migration scripts automatically.

E. Snowflake Pipes to load and transform the existing data.

Answer: C

Explanation:

Option C is the correct approach. A version change handler is a dedicated script that runs automatically when a consumer upgrades or downgrades the application. This handler can contain the necessary SQL scripts to migrate data, update schemas, or perform any other actions required during the version transition. Option B executes only during initial installation, not during upgrades. Option A, RBAC, manages permissions but doesn't execute scripts. Option D, Streams and Tasks,

could be used, but it's more complex than a version change handler. Option E is used for continuous data loading.

NO.3 You are designing a Snowflake Native App that needs to store temporary state information specific to each consumer account. This state information should not be directly accessible to the consumer but must be persistent across application sessions. Which of the following approaches are valid and recommended for storing this data?

A. Create a public schema in the application package and store the state information in tables within that schema. Grant SELECT access to the consumer's account on these tables.

B. Utilize internal stages within the application package to store the state data as files.

C. Create secured views that expose the state data but control access through application roles.

D. Store the state information in a private schema within the application package, ensuring that consumers do not have direct access.

E. Use external stages connected to a provider controlled AWS S3 bucket or similar.

Answer: B,D

Explanation:

Options B and D are the correct approaches. Option D: Private schema, Consumers should not have direct access to the app's internal data. Storing state information in a private schema within the application package ensures that consumers cannot directly query or modify this data, preserving the application's internal state and security. Option B: Internal Stages, offer a way to store file-based data that is tightly coupled with the application package. It enables you to persist the state information for each consumer securely. Option A is incorrect as it gives consumers direct access. Option C is incorrect because the requirement says the data cannot be accessible, not that it should be controlled. Option E creates dependency on external infrastructure.

NO.4 You are developing a Snowflake Native App that leverages shared data from the consumer's account. The app needs to dynamically determine the existence of a specific table (e.g., 'CUSTOMER DATA) in the consumer's shared database before attempting to query it. Which of the following SQL commands or approaches within the application package provides the MOST reliable way to programmatically check for the existence of the table?

A. Use the ' SHOW TABLES LIKE 'CUSTOMER_DATA" command and check if any rows are returned.

B. Execute a 'SELECT 1 FROM CUSTOMER_DATA LIMIT 1' query within a 'TRY...CATCH' block and handle the exception if the table does not exist.

C. Use the 'INFORMATION SCHEMA.TABLES' view and query it to check if a table with the name 'CUSTOMER DATA' exists in the shared database.

D. Use the 'DESCRIBE TABLE CUSTOMER_DATX command within a 'TRY...CATCH' block and handle the exception if the table does not exist.

E. Attempt to grant SELECT privilege on the table and check if an error occurs

Answer: C

Explanation:

Option C (Use the 'INFORMATION SCHEMATABLES view) is the most reliable and recommended approach. The INFORMATION_SCHEMR provides metadata about database objects, and querying it to check for the existence of a table is a standard and efficient practice. Option A ('SHOW TABLES) might be affected by the session context or user's privileges. Options B and D (using 'TRY...CATCH' blocks) rely on exception handling, which can be less efficient and less readable. Also, consuming

exceptions can be more expensive than directly querying system views. Option E is inefficient since it requires an attempt to change a permission.

NO.5 A data science company is developing a Snowflake Native App that provides advanced fraud detection for financial institutions. The app requires secure access to sensitive transaction data within the consumer's Snowflake account. Which of the following approaches BEST aligns with Snowflake's recommended practices for managing data access and securing data within a Native App?

A. The app should request full ACCOUNTADMIN privileges in the consumer account to access all necessary data for fraud detection.

B. The app should use secure views created by the consumer that grant the app read-only access to only the specific transaction data needed for fraud detection.

C. The app should directly access tables containing transaction data using the consumer's own service account credentials, which are securely stored within the app's code.

D. The app should create and manage its own temporary tables in the consumer account and load transaction data into them for processing.

E. The app should directly access transaction data through data sharing from provider account.

Answer: B

Explanation:

Using secure views is the best approach. Secure views allow the consumer to control precisely whatdata the application can access without granting overly broad permissions. This minimizes the risk of unintended data exposure and adheres to the principle of least privilege. ACCOUNTADMIN is excessive and insecure. Storing credentials is also insecure and against best practices. Creating temporary tables and loading data duplicates data and can be inefficient.

NO.6 You are developing a Snowflake Native App that will be listed on the Snowflake Marketplace. The application requires a specific version of a Python library that may conflict with other libraries used by consumers. How can you ensure that your application uses the correct library version without causing conflicts in the consumer's environment?

A. Instruct consumers to manually install the required Python library version in their Snowflake environment before installing your app.

B. Bundle the required Python library with your application package and utilize Anaconda packages support to manage dependencies, ensuring isolation from the consumer's environment.

C. Modify the consumer's 'PYTHON PATH' environment variable upon installation to include the location of your application's Python libraries.

D. Rely on Snowflake's automatic dependency resolution to identify and install the correct version of the Python library in the consumer's environment.

E. Package all application logic into a single SQL UDF to avoid Python dependency issues entirely.

Answer: B

Explanation:

Bundling the required Python library within your application package and leveraging Anaconda is the recommended approach. This creates an isolated environment for your application's dependencies, preventing conflicts with other libraries in the consumer's environment. Manual installation by the consumer is inconvenient and prone to errors. Modifying the is not a supported or recommended practice. Snowflake's dependency resolution may not guarantee the correct version for all cases,

especially with conflicts. Limiting to SQL UDFs might not be feasible or efficient for complex Python-based logic.

NO.7 A Snowflake Native App is designed to perform complex data transformations on large datasets provided by the consumer. To optimize performance and manage resource consumption, you need to choose the appropriate execution context for your application logic. Which of the following options should be considered to accomplish this?

A. Executing all data transformations within a single stored procedure to minimize overhead.

B. Distributing the transformations across multiple SQL UDFs running in parallel using Snowflake's compute grid.

C. Leveraging Snowflake's external functions to offload data transformations to a separate cloud-based compute environment.

D. Using a combination of stored procedures, SQL UDFs, and external functions to orchestrate the data transformation process, depending on the specific requirements of each step.

E. Run entire process by calling API from provider side.

Answer: D

Explanation:

Choosing the correct execution context for your application logic requires a combination of stored procedures, SQL UDFs, and external functions to orchestrate the data transformation process, depending on the specific requirements of each step, offering the best balance of performance, scalability, and resource management. A single stored procedure may not be efficient for complex transformations. SQL UDFs in parallel can be effective but might have limitations depending on the complexity. External functions offer flexibility for offloading but introduce latency and complexity. API call will be more appropriate to transfer the data.

NO.8 You are developing a Snowflake Native App that needs to persist state information between different invocations. The app requires tracking of user preferences, processing progress, and other runtime dat a. Which of the following options are viable and secure methods for persisting this type of state information within the context of a Snowflake Native App?

A. Using temporary tables within the consumer's account to store state information. The tables are dropped when the app is uninstalled.

B. Using the application provider's own Snowflake account to store state information associated with each consumer.

C. Storing state information within the application package itself by updating the package version with each state change.

D. Using internal stages and secured views in consumer account to persist state information.

E. Using secure external stages managed by the application provider to store the consumer-specific state information.

Answer: B,D

Explanation:

Persisting state information requires careful consideration of security, scalability, and isolation. Using the application provider's own Snowflake account is a viable approach if designed carefully, but requires robust security measures to isolate consumer data. Internal stages and secure views offer a more secure and controlled way to persist state within the consumer's account. Temporary tables are ephemeral and unsuitable for long-term state persistence. Storing state within the application

package is impractical and would lead to versioning issues. External stages managed by the provider introduce complexity and potential security risks.

NO.9 You are developing a Snowflake Native App that requires accessing sensitive user data within the consumer's account. To ensure data privacy and security, you need to implement appropriate access controls. Which of the following approaches is the MOST secure and recommended way to grant your app access to this data, assuming the data is stored in a table named 'user_data' ?

A. Granting 'USAGE privilege on the 'user_data' table directly to the application role provided by Snowflake Native App Framework.

B. Creating a custom role in the consumer account with 'SELECT privilege on the 'user_data' table and granting that role to the application.

C. Defining a secure view that filters the table based on the application role and granting 'SELECT privilege on the secure view to the application role provided by Snowflake Native App Framework.

D. Using a stored procedure with 'EXECUTE AS CALLER that accesses the 'user_data' table. Grant 'EXECUTE privilege on the stored procedure to the application role.

E. Granting 'SELECT privilege directly on the 'user_data' table to the application role provided by Snowflake Native App Framework.

Answer: C

Explanation:

The most secure approach is to use a secure view. This allows you to control which rows and columns the application can access. Granting 'SELECT directly to the application role is generally discouraged as it exposes the entire table. Using 'EXECUTE AS CALLER has security implications and might not be desirable. Creating custom roles in the consumer account adds complexity to the application deployment.

NO.10 You're developing a Snowflake Native App. Part of your application logic involves using a UDF (User-Defined Function) to perform complex calculations on data residing in a table within the consumer's Snowflake account. Which of the following statements correctly describes how you would package and deploy this UDF within your Native App using the application package?

A. The UDF definition must be created directly within the consumer's account after the application is installed, as UDFs cannot be included in the application package.

B. You must create the UDF within a schema in your application package, and grant USAGE privilege on the schema to the application role. The consumer will automatically be granted EXECUTE privilege on the UDF.

C. You must create the UDF within a schema in your application package. No additional privileges need to be granted, as the application role automatically has all necessary privileges within the application's namespace.

D. You must create the UDF within a schema in your application package, and grant USAGE privilege on the schema to the application role. The application code can then execute the UDF without any further privilege grants within the consumer account, and the UDF can be called from SQL or other UDFs using unqualified names.

E. The UDF definition must be created as an external function, pointing to an external API endpoint that your application controls, enabling the UDF logic to be executed outside of the consumer's Snowflake environment.

Answer: D
Explanation:
The UDF needs to be created within a schema in the application package. The application role needs USAGE privilege on the schema containing the UDF, and the application can then execute the UDF.

NO.11 Consider the following 'manifest.yml' file for a Snowflake Native App:
A. The application will fail to install because the version property is missing. Each version must explicitly be specified in the manifest file.
B. The application version will default to '0.1.0' and the application can be installed successfully by any consumer account.
C. The application version will be automatically assigned by Snowflake during the installation process on the consumer side.
D. The application will fail to install because no 'privileges' are listed in the manifest file. All applications must declare the necessary privileges.
E. The application version will default to '0.0.1'. If the application includes a view, then the view needs access to the user data and must have a privilege declared to allow users to view data from the application.
Answer: A
Explanation:
A valid version is mandatory in 'manifest.yml' to install native apps. When version is missing, the installation will fail.

NO.12 You are developing a Snowflake Native App that processes data in the consumer's account and presents the results in a dashboard. You need to ensure that the dashboard automatically updates when the underlying data changes. Which of the following approaches, or combination of approaches, would be MOST effective in achieving this real-time or near real-time data refresh for the dashboard?
Choose TWO.
A. Schedule a Snowflake Task within the application package to refresh the data used by the dashboard every minute.
B. Use Snowflake Streams and Tasks to incrementally update the dashboard's underlying data based on changes in the source tables.
C. Implement a mechanism where the consumer's application triggers a refresh of the dashboard's data through a stored procedure whenever they detect changes in their data.
D. Rely solely on the consumer's existing data pipelines to update the data used by the dashboard.
E. Implement a serverless function using Snowflake's Snowpark Container Services to periodically query the consumer's data and update the dashboard within the Native App environment.
Answer: B,C
Explanation:
Streams and Tasks are designed for incremental data processing and are well-suited for automatically updating data in response to changes. Allowing the customer to trigger an update is useful, but it doesn't provide auto update. A task can be used, however Streams and Tasks should be used. Depending on the application architecture Container Services might be used to keep the dashboards updated.

NO.13 You are developing a Snowflake Native App. Your app needs to interact with the consumer's Snowflake account to retrieve dat a. You want to ensure that your app only has access to the necessary data and that the consumer can control the data access. Which of the following methods is the MOST secure and recommended way to achieve this?

A. Use a Snowflake account administrator role to execute all queries within the app.

B. Create a service account within the consumer's Snowflake account and grant the necessary privileges to this account.

C. Utilize Snowflake's secure data sharing feature combined with granular access control policies (row access policies, masking policies) within the provider's app, granting access to specific views or tables.

D. Expose an API endpoint within the consumer's account that grants access to the data without any restriction.

E. Create a shared secret between the provider and consumer accounts and use this secret to authenticate API calls for data access.

Answer: C

Explanation:

Snowflake's secure data sharing feature, coupled with granular access control policies, provides the MOST secure and controlled way to access data within a consumer's Snowflake account. This approach allows the provider to define which data the app can access and the consumer to control the access through policies.

NO.14 A Native App provider wants to use a parameter in their MANIFEST.yml to dynamically control the number of worker threads a stored procedure uses. The desired behavior is to allow the consumer to configure this value. Considering security best practices, which of the following parameter definitions in the 'MANIFEST.yml' is the MOST appropriate?

A.

```yaml parameters: WORKER_THREADS: type: INT default: 4 secure: false ```

B.

```yaml parameters: WORKER_THREADS: type: STRING default: '4' secure: true ```

C.

```yaml parameters: WORKER_THREADS: type: INT default: 4 secure: true ```

D.

```yaml parameters: WORKER_THREADS: type: VARCHAR default: '4' secure: false ```

E.

```yaml parameters: WORKER_THREADS: type: INT default: 4 secure: true validation_rules: - type: range min: 1 max: 16 ```

Answer: E

Explanation:

Option E is the MOST appropriate. While 'secure: true' is important to encrypt and protect the parameter value, 'validation_rules' provide an extra layer of security and data integrity by ensuring the consumer provides a valid value within a defined range. It prevents potentially harmful or unexpected values from being used, enhancing the app's robustness. Type INT also correctly defines the expected data type.
```

NO.15 You are developing a Snowflake Native App that performs data transformations. The core logic is implemented in a series of stored procedures. You need to implement versioning for your app to allow for future updates and rollbacks. Which of the following steps are REQUIRED to properly implement versioning for a Snowflake Native App?

A. Increment the 'version' attribute in the 'MANIFEST.yml' file for each new release.

B. Create a separate branch in your version control system for each version of the app.

C. Update the 'version' attribute in the 'setup.sqr file, if one exists.

D. Use the 'ALTER APPLICATION' command to update the app's version in the consumer's account.

E. Tag each release in your version control system with the corresponding version number.

Answer: A

Explanation:

The 'version' attribute in the 'MANIFEST.yml' file is the core mechanism Snowflake uses to identify different versions of your Native App. Incrementing this value for each release is essential for proper versioning. The other options, while potentially part of a good software development lifecycle, are not directly related to how Snowflake handles Native App versions. ALTER APPLICATION is used to update the PATCH version, not the main version declared in MANIFEST.yml. No ALTER APPLICATION command is used in provider account

NO.16 Your team is building a Snowflake Native App that uses a UDF (User-Defined Function) for data enrichment. The UDF needs to access an external API key. Which of the following approaches are valid and secure methods for providing the API key to the UDF within the Native App framework?

A. Store the API key directly in the UDF's code.

B. Pass the API key as a parameter to the UDF when it's called.

C. Use a secure parameter in the MANIFEST.yml and access it within the UDF.

D. Store the API key in a database table within the consumer's account and grant the UDF access to that table.

E. Store the API key using Snowflake's Secret Management feature and grant the UDF access to the secret.

Answer: C,E

Explanation:

Storing the API key directly in the UDF's code (A) is highly insecure. Passing it as a parameter (B) exposes the key. Storing the API Key in consumer Account (D) can be done, but its not ideal as it asks to store the API Key outside of the Native APP Package by consumer. Using a secure parameter in the MANIFEST.yml (C) is a secure way to provide configuration values to the app. Storing the API Key with Snowflake Secret Management (E) is another secure method, allowing to store secrets encrypted and control access. So, options C and E are the valid and MOST secure methods.

NO.17 You are developing a Snowflake Native App that requires specific privileges to be granted to the consumer account. These privileges are necessary for the app to access and process data within the consumer's Snowflake environment. Which section of the manifest file is primarily used to declare these required consumer-side privileges?

A.

```
application.privileges
```

B.

```
application.behavior
```

C.

```
application.resources
```

D.

```
application.setup
```

E.

```
application.initialization
```

Answer: A

Explanation:

The application.privileges section in the manifest file is specifically designed to declare the privileges required by the Native App in application . privileges the consumer's account. This ensures that the app has the necessary permissions to function correctly within the consumer's Snowflake environment. The other options are related to different aspects of the app's behavior, resources, and setup but not the declaration of required privileges.

NO.18 A Snowflake Native App developer wants to externalize certain configuration values so that a consumer can adjust the app's behavior without modifying the app's code directly. Which of the following methods is the MOST appropriate for achieving this using the Snowflake Native App Framework?

A. Hardcoding the configuration values directly into the SQL code.

B. Using Snowflake variables and setting them at the session level.

C. Defining parameters in the manifest file and using them within the application code.

D. Storing the configuration values in a separate Snowflake table within the consumer account.

E. Using dynamic SQL to modify the application code at runtime based on environment variables.

Answer: C

Explanation:

Defining parameters in the manifest file is the most appropriate way to externalize configuration values in a Snowflake Native App. This allows the consumer to set these parameters during installation or upgrade, effectively tailoring the app's behavior without needing to modify the underlying code. Hardcoding values is inflexible, session variables are not persistent for the app, using a separate table adds unnecessary complexity, and dynamic SQL for code modification is generally discouraged for security and maintainability reasons.

NO.19 You are packaging a Snowflake Native App for distribution. Which of the following artifacts are essential components of the application bundle that must be included?

A. The application's source code (e.g., SQL scripts, Python files).

B. The manifest file (snowflake.yml).

C. Binary executables compiled for a specific operating system.

D. The 'setup.sql file containing the application's initialization logic.

E. Compiled Java JAR files if the app includes Java UDFs.

Answer: B,D,E

Explanation:

The manifest file (snowflake.yml) is crucial as it defines the app's metadata and dependencies. The

'setup.sqr file is also essential as it contains the necessary SQL statements to initialize the application environment in the consumer's account. If the app consists of any Java UDFs, the corresponding compiled JAR files are required to be packaged along with the application. Binary executables for a specific operating system is incorrect as snowflake app excutes on snowflake computing.

NO.20 A Snowflake Native App developer encounters an issue where the application fails to install on a consumer's account due to insufficient privileges. The developer has verified that the necessary privileges are declared in the application. privileges section of the manifest file. However, the installation still fails. What could be the most likely cause of this issue, and how can it be resolved?

A. The consumer's account is not on the same Snowflake region as the provider account. Verify and align the regions.

B. The consumer's account does not have the SNOWFLAKE.APP_INSTALLER role enabled. Ensure this role is granted to the appropriate user.

C. The privileges declared in the manifest file are not supported by the consumer's Snowflake edition. Review and adjust the required privileges.

D. The consumer account has object level grants conflicting with the required grants. Revoke the object level grants and try again.

E. The privileges were declared in the manifest file, but were not actually granted to the application role in the provider account before creating the version. Grant the declared privileges to the application role using GRANT ON ACCOUNT TO APPLICATION ROLE and create a new version.

Answer: E

Explanation:

Even if the privileges are declared in the manifest file, they must be explicitly granted to the application role in the provider account before a new version is created. The consumer's Snowflake edition might not support all privileges (but the error would be slightly different), SNOWFLAKE.APP INSTALLER role does not exist , regional difference does not matter as such if provider created version after granting the privileges. The application's role in provider account needs required permission using GRANT ON ACCOUNT TO APPLICATION ROLE .

NO.21 You are developing a Snowflake Native App that needs to access a secured external API to enrich customer dat a. The API requires an OAuth 2.0 token stored securely. Which of the following approaches would you use within the application manifest and Snowflake objects to securely store and access this token, minimizing exposure and adhering to best practices?

A. Store the OAuth 2.0 token directly in the application's manifest file as a PARAMETER. This simplifies access within the application code.

B. Store the OAuth 2.0 token as a cleartext secret in a Snowflake stage and access it using external functions.

C. Use Snowflake's SECRET object type to securely store the OAuth 2.0 token. Reference the SECRET object within the application using PARAMETERS in the manifest and utilize function in application code to retrieve it at runtime.

D. Hardcode the OAuth 2.0 token within the application's stored procedures. This ensures the token is readily available.

E. Store the OAuth 2.0 token as a masked comment within the application's SQL code. This will

provide sufficient obfuscation.

Answer: C

Explanation:

Storing secrets like OAuth tokens directly in the manifest or hardcoding them is a significant security risk. Using Snowflake SECRET objects is the recommended approach for securely managing sensitive information. The 'SYSTEM$GET SECRET function provides a secure way to retrieve the secret at runtime without exposing it in the code or manifest. Snowflake stages shouldn't be used to store secrets in clear text.

NO.22 You're packaging a Snowflake Native App and need to define the minimum Snowflake version required for installation. You also want to specify an optional PARAMETER that users can configure during installation to customize the app's behavior. Which section of the manifest file allows you to achieve this?

A. The 'setup.sqrl' file, where you define SQL commands to be executed during app installation.

B. The 'application.yaml' file under the 'configuration' section, where you can specify and define PARAMETERS with their types and default values.

C. The 'metadata.json' file, which only contains descriptive information about the app.

D. The 'privileges.sql' file, where you define the privileges required by the application.

E. The 'deployment.sql' file where you define the app's deployment settings.

Answer: B

Explanation:

The 'application.yaml' file is the core configuration file for a Snowflake Native App. It's where you define the to ensure compatibility and declare PARAMETERS with their data types, descriptions, and optional default values, enabling customization during app installation.

NO.23 Consider the following snippet from a Snowflake Native App manifest file (application.yaml):

```
configuration:
  minimum_snowflake_version: 7.0
  parameters:
    API_ENDPOINT:
      type: STRING
      default: 'https://default.api.example.com'
      description: 'The API endpoint to use.'
    DATA_RETENTION_DAYS:
      type: INT
      default: 30
      description: 'Number of days to retain data.'
    ENABLE_LOGGING:
      type: BOOLEAN
      default: false
      description: 'Enable verbose logging.'
```

Which of the following statements are TRUE regarding how these parameters can be accessed and used within the application's SQL code?

- ☐ Parameters can be directly accessed using the parameter name as a variable within SQL statements (e.g., `SELECT FROM my_table WHERE retention_date < CURRENT_DATE() - DATA_RETENTION_DAYS;`).
- ☐ Parameters must be accessed using the `SYSTEM$GET_PARAMETER` function, providing the parameter name as an argument (e.g., `SELECT SYSTEM$GET_PARAMETER('DATA_RETENTION_DAYS');`).
- ☐ The app package consumer can override the default values of these parameters during installation.
- ☐ The parameters are only accessible within the manifest file and cannot be used in the application's SQL code directly.
- ☐ These parameters can only be accessed during the app's installation process and not during runtime.

A. Option A
B. Option B
C. Option C
D. Option D
E. Option E
Answer: B,C
Explanation:
Snowflake Native Apps use 'SYSTEM$GET PARAMETER to access parameter values defined in the manifest. This function retrieves the current value of the parameter, which can be either the default value defined in the manifest or a value overridden by the app package consumer during installation. Thus C is also correct. Direct variable access (A) is incorrect and Parameters are not restricted only to manifest file. They are accessible at runtime. So D and E are also incorrect

NO.24 You are designing the 'privileges.sqr file for a Snowflake Native App. The app requires the ability to create and manage internal stages within the consumer's account to store temporary dat a. Which of the following SQL statements in 'privileges.sqr would correctly grant the necessary privileges to the application role (assuming the application role is named 'APP ROLE)?

A.
```
GRANT USAGE ON WAREHOUSE TO ROLE APP_ROLE;
```
B.
```
GRANT CREATE STAGE ON DATABASE TO ROLE APP_ROLE;
```
C.
```
GRANT ALL PRIVILEGES ON DATABASE TO ROLE APP_ROLE;
```
D.
```
GRANT CREATE STAGE ON SCHEMA TO ROLE APP_ROLE; GRANT USAGE ON SCHEMA TO ROLE APP_ROLE;
```
E.
```
GRANT OWNERSHIP ON DATABASE TO ROLE APP_ROLE;
```
Answer: D
Explanation:
To create stages, the application role needs the 'CREATE STAGE' privilege. This privilege needs to be granted on the schema where the stages will be created, and usage privilege is also needed. Granting it at the database level is too broad (B, C, E), and only granting warehouse usage (A) doesn't address the requirement of creating stages.

NO.25 You are developing a Snowflake Native App that requires specific privileges to access data in the consumer's account. In your manifest file, you want to request these privileges. Which of the following statements best describes the correct way to request these privileges within the 'privileges' section of the manifest file?
A. Privileges are automatically granted to the application based on the roles specified in the application code. No explicit declaration is needed in the manifest file.
B. Use the 'allowed_roles' section in the manifest to list the roles that are allowed to access data in the consumer's account, Snowflake automatically grants necessary privileges to these roles.
C. Specify a list of required privileges under the 'privileges' section, detailing the specific objects (databases, schemas, tables) and actions (SELECT, INSERT, UPDATE) the application needs to perform .
D. Privileges are managed entirely through the application's setup scripts. The manifest file does not play a role in privilege management.
E. You can only request privileges for future grants. Existing object grants are not supported in manifest file.
Answer: C
Explanation:
The 'privileges' section of the manifest file is used to declare the specific privileges required by the application. This includes specifying the objects (databases, schemas, tables) and actions (SELECT, INSERT, UPDATE) that the application needs to perform in the consumer's account. Snowflake uses this information to manage access control and ensure that the application has the necessary permissions to function correctly.

NO.26 You are building a Snowflake Native App that needs to store configuration settings specific to each consumer account. These settings will influence the behavior of the application. Which of the following methods is the MOST secure and recommended approach for managing these

configuration settings within the Snowflake Native App Framework?

A. Store the configuration settings directly within the application's code. This allows for easy access and modification during development.

B. Use external configuration files stored in an Amazon S3 bucket and accessed via external functions. This provides flexibility and scalability.

C. Utilize parameters defined in the 'setup.sql' script and accessible via SQL commands like 'SYSTEM$GET APP PARAMETER. This allows consumers to configure the application during installation.

D. Store the configuration settings in a public database table within the application's share. This makes the settings easily accessible to all consumers.

E. Hardcode the configuration settings directly into the manifest file as environment variables.

Answer: C

Explanation:

Using parameters defined in the 'setup.sqr script and accessed via 'SYSTEM$GET APP PARAMETER is the most secure and recommended approach. It allows consumers to configure the application during installation, provides a controlled and secure mechanism for accessing configuration settings, and prevents sensitive information from being directly embedded in the application code.

NO.27 You are tasked with designing a Snowflake Native App that requires multiple roles with different levels of access to its functionalities. The app needs to define which roles can perform specific actions, such as viewing data, modifying data, or managing app settings. Which of the following methods are CORRECT ways to control role-based access within a Snowflake Native App? (Select all that apply)

A. Granting specific privileges to roles directly within the application's stored procedures and functions, checking the current role using and conditionally executing code based on the role.

B. Using the 'allowed_roleS section in the manifest file to restrict access to certain functionalities based on the consumer's roles. This ensures that only authorized roles can interact with those components.

C. Creating separate versions of the application for each role, with each version containing only the functionalities that the role is allowed to access.

D. Defining application roles within the provider account and mapping them to consumer roles during the installation process. These application roles can then be used for fine-grained access control.

E. Defining roles within the setup script with specific privileges and instructing the consumer to grant their existing roles to these newly created roles.

Answer: A,D

Explanation:

Option A is correct because using within stored procedures and functions allows you to conditionally execute code based on the current role, providing fine-grained access control. Option D is also correct because defining application roles within the provider account and mapping them to consumer roles during installation allows for a structured and manageable approach to role-based access control within the app.

NO.28 You are developing a Snowflake Native App that requires specific Snowflake features and

functionalities. To ensure compatibility and proper execution across different consumer environments, you need to define the minimum required Snowflake version. Which section in the manifest file is used to specify the minimum Snowflake version required by the application and what is the correct syntax?

A. The 'dependencies' section. Syntax: '"dependencies": [{"snowflake_version":">=7.0"}]

B. The 'application' section. Syntax: '"application": {"min_snowflake_version": "7.0"}'

C. The 'configuration' section. Syntax: '"configuration": {"required_snowflake_version": "7.0"}'

D. The 'setup' section. Syntax:

```
CREATE PROCEDURE setup() RETURNS STRING LANGUAGE SQL AS $$
  SET snowflake_version = CURRENT_VERSION();
  IF snowflake_version < '7.0' THEN
    RETURN 'Minimum Snowflake version requirement not met.';
  END IF;
$$;
```

E. The 'api_version' section. Syntax: '"api_version": along with in the 'applications section.

Answer: B

Explanation:

The 'application' section within manifest file is used to specify minimum snowflake version required by Snowflake Native Application with the syntax '"application": {"min_snowflake_version": "7.0"}'. This attribute ensures that app installed is compatible with consumer account.

NO.29 You are developing a Snowflake Native App that requires specific database privileges to be granted to the application role during installation. Which sections of the 'manifest.yml' file are primarily responsible for defining these required privileges, and how does Snowflake interpret and enforce them during the installation process?

A. The 'application.initial_version.privileges' section defines the privileges needed. Snowflake automatically grants these privileges to the application role defined in 'application.initial_version.role' before the setup script runs. This ensures the application has the necessary permissions to create objects and execute code within the consumer's account.

B. The 'setup.script' section defines the privileges needed. The application setup code must explicitly grant these privileges using 'GRANT statements to a specific application role. Snowflake does not automatically grant any privileges, so you have full control over permission management.

C. The 'application.initial_version.artifacts' section is used to define the privileges. Snowflake parses this section to identify required privileges and grants them dynamically to the application role whenever a function or procedure is called that requires them.

D. A combination of the 'application.initial_version.privileges' and 'setup.script' sections. 'application.initial_version.privileges' defines initial static privileges, while 'setup.script' can grant additional privileges dynamically during installation if needed. Snowflake ensures that all specified privileges are granted before the application is fully operational.

E. The 'application.initial_version.parameters' section. While parameters can control behavior, they do not define or manage privileges. Privileges must be manually configured by the consumer post-installation.

Answer: D
Explanation:
The correct answer is D. The 'application.initial_version.privileges' section is used to define static privileges that the application role should have from the start. The 'setup.script' allows for more dynamic privilege granting, enabling the application to adjust permissions based on the consumer's environment or configuration. Snowflake ensures these privileges are in place to enable the application to function correctly.

NO.30 You are packaging a Snowflake Native App using the 'snowflake nativeapp package' command. The app relies on a specific external Python library that isn't available by default in Snowflake. Which methods can you use to include this library in your application package and ensure it's accessible within your application's procedures and functions?

A. Upload the Python library as a stage object and reference it in the 'manifest.ymr file using the 'artifacts' section. Snowflake will automatically include it in the application package and make it available during runtime using the 'IMPORT' statement in the handler code.

B. Bundle the Python library as a ZIP file along with your application code. When creating the function using 'CREATE FUNCTION' , specify the ZIP file as part of the 'IMPORTS' clause. Snowflake automatically extracts and makes the library available to the function.

C. Use the Anaconda channel integration. Specify the required package in the 'manifest.ymr file, and Snowflake automatically downloads and installs the library during application installation. This only works for packages available through Anaconda.

D. Combine options B and C. Use the Anaconda channel integration when possible, and for custom or unavailable packages, use the ZIP file import method for individual functions. Prioritize Anaconda for easier dependency management.

E. Upload the Python library using the 'PUT' command and explicitly reference it in the 'manifest.ymr under the 'artifacts' section, pointing to the internal stage location. Use the 'IMPORT statement within your stored procedure or UDF handler code to make the library available during execution.

Answer: D
Explanation:
The correct answer is D. Snowflake allows using both Anaconda channel for readily available packages and ZIP file imports for custom or unavailable ones. Utilizing Anaconda streamlines dependency management where possible, while the ZIP file method provides flexibility for including any Python library directly with the function/procedure deployment.

NO.31 You are designing a Snowflake Native App that processes sensitive customer dat a. To comply with data residency requirements, you need to ensure the application processes data exclusively within the consumer's Snowflake account and does not transmit data outside.
Which features of the Snowflake Native App Framework support this requirement? (Select all that apply)

A. Secure Data Sharing: Allows the application to access data directly within the consumer's account without copying or moving it.

B. External Functions: Enables the application to call external services for data processing and enrichment, ensuring all data remains within the Snowflake environment.

C. Snowflake Marketplace: Provides a secure and governed platform for distributing the application without requiring direct access to the consumer's account.

D. Stored Procedures: Allows the application to execute custom logic within the consumer's Snowflake environment, processing data locally.

E. Application Packages: Package the entire application including code and configuration, ensuring data locality and not data transfer outside environment

Answer: A,D,E

Explanation:

The correct answers are A, D and E. Secure Data Sharing enables direct access to data without movement. Stored Procedures execute logic within the consumer's environment. Application packages ensures everything within the consumer's environment. External Functions (B) inherently involve sending data outside of Snowflake, violating the residency requirement. Snowflake Marketplace (C) is about distribution and governance, not data processing location.

NO.32 You are developing a Snowflake Native App that needs to be configurable by the consumer during installation. You define parameters in the 'manifest.ymr file. During application installation, the consumer provides values for these parameters. How can you access these parameter values within your application's setup script ('setup.sql')?

A. Parameter values are automatically available as global variables within the 'setup.sqr script. You can reference them directly by their name as defined in the 'manifest.yml'.

B. You must use the 'GET_DDL' function to retrieve the parameter values from the application object definition. The returned DDL string will contain the parameter values, which you need to parse.

C. Parameter values are passed as arguments to the 'setup.sqr script. You need to define input parameters in the script declaration and then access them by their position or name.

D. Use the ' SYSTEM$GET PARAMETER function within the 'setup.sqr script, passing the parameter name as an argument. This function retrieves the current value of the specified parameter for the application.

E. Parameter values are stored in a dedicated table automatically created by Snowflake during application installation. You can query this table using standard SQL to retrieve the parameter values.

Answer: D

Explanation:

The correct answer is D. The 'SYSTEM$GET_PARAMETER function is the designated way to access the parameter values specified by the consumer during installation from within the application's setup script. This function allows you to dynamically configure the application based on the consumer's input.

NO.33 You are developing a Snowflake Native App that requires granular access control to its underlying data for consumer accounts. You want to implement a secure pattern that limits direct access to the data while allowing specific operations through your app's procedures. Which of the following approaches represent the MOST secure and recommended practices regarding access control in the setup script, assuming you also want to grant USAGE on future schema?

A. Granting SELECT privilege directly on the underlying tables in the app's data schema to the consumer's role.

```
GRANT SELECT ON ALL TABLES IN SCHEMA app_db.data_schema TO APPLICATION ROLE app_role;
```

B. Creating secure views on top of the underlying tables and granting SELECT privilege on these views to the consumer's role, while ensuring consumers can only access the exposed view.

```
CREATE OR REPLACE SECURE VIEW app_db.public.secure_view AS SELECT  FROM app_db.data_schema.my_table; GRANT SELECT ON SECURE VIEW app_db.public.
secure_view TO APPLICATION ROLE app_role;
```

C. Creating stored procedures with the 'EXECUTE AS CALLER privilege and granting USAGE on the schema to the consumer's role. The stored procedures handle all data access logic.

```
CREATE OR REPLACE PROCEDURE app_db.public.get_data() RETURNS VARIANT LANGUAGE SQL EXECUTE AS CALLER AS $$
 SELECT   FROM app_db.data_schema.my_table;
$$; GRANT USAGE ON SCHEMA app_db.public TO APPLICATION ROLE app_role; GRANT EXECUTE ON PROCEDURE app_db.public.get_data() TO APPLICATION ROLE app_role;
```

D. Granting OWNERSHIP on the app's data schema to the consumer's role, allowing them full control over the data.

```
GRANT OWNERSHIP ON SCHEMA app_db.data_schema TO APPLICATION ROLE app_role;
```

E. Creating stored procedures with the 'EXECUTE AS OWNER privilege and granting USAGE on the schema to the consumer's role. The stored procedures handle all data access logic.

```
CREATE OR REPLACE PROCEDURE app_db.public.get_data() RETURNS VARIANT LANGUAGE SQL EXECUTE AS OWNER AS $$
 SELECT   FROM app_db.data_schema.my_table;
$$; GRANT USAGE ON SCHEMA app_db.public TO APPLICATION ROLE app_role; GRANT EXECUTE ON PROCEDURE app_db.public.get_data() TO APPLICATION ROLE app_role;
```

Answer: C,E

Explanation:

Granting SELECT directly is insecure (A). Secure views provide some abstraction but still allow direct SELECT (B). Granting OWNERSHIP gives excessive privileges (D). Stored procedures with EXECUTE AS OWNER or 'EXECUTE AS CALLER , combined with limited schema privileges, offer the best control and security. 'EXECUTE AS OWNER' allows the procedure to run with the privileges of the app itself, bypassing consumer privileges, and 'EXECUTE AS CALLER' uses consumer's priviledges, but still contains the data logic. Both combined, gives the best secure and recommended access control.

NO.34 You are tasked with developing a Snowflake Native App that leverages a Partner provided secret using an external function which will send the processed data to their system. You plan to store the OAuth information to be used by the external function in a secret object managed by Snowflake. Which of the following steps are NECESSARY to ensure secure and correct setup of this app? Select all the options that apply.

A. Store the OAuth token directly within the external function's definition as a variable, ensuring it is encrypted using AES ENCRYPT before deployment.

B. Create a Secret object in Snowflake to store the OAuth token securely and reference it within the external function using 'SYSTEM$GET_SECRET function. Ensure proper access rights on the Secret object.

C. Grant the 'IMPORTED PRIVILEGES privilege on the database containing the Secret object to the application role. GRANT IMPORTED PRIVILEGES ON DATABASE TO APPLICATION ROLE '

D. Grant the 'USAGE privilege on the API integration to the application role if the external function requires an API integration. GRANT USAGE ON API INTEGRATION TO APPLICATION ROLE

E. Bypass storing the OAuth token in Snowflake and instead rely on passing it as a parameter to the external function from the consumer account, documented clearly for the consumer's security responsibilities.

Answer: B,C,D

Explanation:

Storing the OAuth token directly in the function (A) is insecure. Passing the OAuth token from the consumer (E) is not a recommended practice. The correct approach is to use Snowflake's Secret object (B) to store the token securely. The application role needs 'IMPORTED PRIVILEGES on the database containing the secret (C) to access the secret. The 'USAGE privilege on the API integration(D) is also needed if the external function leverages an API integration.

NO.35 You are developing a Snowflake Native App that performs complex data transformations and offers different pricing tiers based on usage. You plan to use Snowflake Usage Metering Events to track consumer usage. Which of the following is the MOST appropriate approach to track usage and associate it with the correct pricing tier in your setup script and application code?

A. Call 'SYSTEM$METER USAGE with a fixed event name and quantity for each transformation, regardless of complexity or pricing tier.

B. Call 'SYSTEM$METER USAGE with a dynamic event name that encodes the pricing tier and complexity of the transformation. Store the mapping of event names to pricing in a configuration table within your app.

```
SELECT SYSTEM$METER_USAGE('tier_premium_complex_transform', 1);
```

C. Call 'SYSTEM$METER USAGE without specifying an event name. Aggregate usage data based on query history in the consumer account to determine pricing.

D. Avoid using 'SYSTEM$METER USAGE altogether and rely on Snowflake's built-in resource monitoring to track usage for billing.

E. Create a view on the APP USAGMETERING_HISTORY view to capture usage data. Call 'SYSTEM$METER USAGE from with in the View.

```
CREATE OR REPLACE VIEW app_db.public.usage_view AS SELECT CURRENT_ACCOUNT(), CURRENT_DATABASE(), CURRENT_SCHEMA(), SYSTEM$METER_USAGE('transformation_event', 1);
```

Answer: B

Explanation:

Using a fixed event name (A) doesn't allow for differentiation between pricing tiers. Relying on query history (C) or resource monitoring (D) is not reliable or precise for application-specific usage. Calling 'SYSTEM$METER_USAGE from a view (E) isn't the intended way to track usage. The best approach is to use dynamic event names that reflect the pricing tier and complexity, enabling accurate tracking and billing (B).