SNOWFLAKE NATIVE APPS: DEVELOPER CHEAT SHEET

Key Concepts of Snowflake Native Apps

What Providers Do

- Author application source code and bundle it into an *application package*
- Publish and monetize application packages on Snowflake Marketplace
- Maintain applications via staged rollouts of updates

What Consumers Do

- Discover applications on Snowflake Marketplace
- Install applications in their Snowflake account
- Run applications using their compute resources
- Maintain full control of the application's permissions

Common Patterns: Installation Scripts

An installation script is required; it's commonly named **setup.sql**.

The script contains SQL that is executed when a consumer installs the app.

The installation script typically includes definitions for:

- Roles (app roles)
- Permissions (for each role)
- Logic (procedures, UDFs)
- New objects (schemas, tables, etc.)

Sample **setup.sql:**

```
CREATE APPLICATION ROLE app_user;

CREATE OR ALTER VERSIONED SCHEMA code;
GRANT USAGE ON SCHEMA code TO APPLICATION
ROLE app_user;

CREATE OR REPLACE PROCEDURE code.do_thing() ... ;


GRANT USAGE ON PROCEDURE core.do_thing() TO
APPLICATION ROLE app_user;

CREATE SCHEMA config IF NOT EXISTS;
CREATE TABLE config.params IF NOT EXISTS (
    param_name STRING,
    param_value STRING,
    change_time TIMESTAMP,
    comment STRING,
);
```

Sharing Provider Data With App Consumers

When an application package depends on an external database in the provider's account, providers use *reference usage* to share it with the consumer via the app.

```
GRANT REFERENCE_USAGE ON DATABASE provider_data TO
SHARE IN APPLICATION PACKAGE app_pkg;
```

Next, create a schema, create necessary views using provider data, and add views to the schema.

Finally, grant usage on the schema and relevant permissions on the views and share in the application package.

```
GRANT USAGE ON SCHEMA schema_shared TO SHARE IN
APPLICATION PACKAGE app_pkg;

GRANT SELECT ON VIEW schema_shared.sensor_types_
view TO SHARE IN APPLICATION PACKAGE app_pkg;
```

Key Components of the Snowflake Native Application Framework

This is the core set of tools used to build a Snowflake Native App:

- **Snowflake accounts:** Where data is stored
- **SQL:** Used for the application installation script and/or logic
- **Snowpark:** Used for data transformations, UDFs and stored procedures
- **Streamlit:** Used to build the frontend of the application

UDFs, stored procedures, tasks and other Snowflake primitives are available to use alongside this core set of tools.

Common Patterns: Manifest File

A manifest file is required and must be named **manifest.yml.**

It contains metadata about the app and references to objects required by the application.

The manifest file typically includes:

- App version
- Artifact references (references to the installation script and main UI file)
- Log configuration
- Object references (tables, warehouses, etc.) and permission levels required by the app

Sample **manifest.yml:**

```
manifest_version: 1

version:
    name: v1
    label: Version One
    comment: The first version of the application

artifacts:
    setup_script: scripts/setup.sql
    default_streamlit:
app_instance_schema.streamlit

configuration:
    log_level: debug
    trace_level: off

references:
    - order_table:
         label: "Orders Table"
         description: "Select table"
         privileges:
             - SELECT
         object_type: Table
         multi_valued: false
         register_callback: app_instance_schema.
update_reference
```

Testing Your Application

Install the app in your account to test it.

Sample code:

```
CREATE APPLICATION my_new_app FROM APPLICATION
PACKAGE app_pkg USING VERSION v1 PATCH 0;
```

Open and run the application.

Mechanics Behind a Snowflake Native App

App source code lives in a Snowflake-managed stage; an application package is created from the source code in the stage.

An application package typically includes:

- Installation script (**setup.sql**)
- App metadata (**manifest.yml**)
- References to objects required by the app (**manifest.yml**)
- Application logic with SQL and/or Snowpark (such as **udfs.py**)
- Application frontend with Streamlit (such as **ui.py**)

Sample directory structure of an app:

```
my_app/
    setup.sql
    manifest.yml
    src/
        udfs.py
        ui.py
```

Common Patterns: App Logic and Frontend

Application Logic

- Flexible directory structure can be organized according to your preferences.
- UDFs and stored procedures are defined in the application script and bound to logic defined in files in the application source code.
- Logic can be written in SQL or any supported language.

Frontend

- Defined in a Python file (for example, **ui.py**).
- App can have one or more UI files, but one must be designated as the default in the manifest file.

Sample **UDF** (with binding):

udf.py

```
def add(a,b):
    return a + b
```

setup.sql

```
create or replace function my_schema.add(a int,
b int)
returns int
language python
runtime_version = '3.8'
packages = ('snowflake-snowpark-python')
imports = ('/some_path/udf.py')
handler = 'udf.add';
```

Application Management

Only two major versions of an application may exist at any point. Major versions may contain multiple patches (up to 130).

Use a release directive (specify major version and patch) to set the default version of the app that consumers will install.

Update the release directive when a new version of the app is ready to be rolled out. Updating the release directive initiates an automated upgrade that will update all installed instances of the previous version.

Sample code for default release directive:

```
ALTER APPLICATION PACKAGE hello_snowflake_package
       SET DEFAULT RELEASE DIRECTIVE
       VERSION = v1_0
       PATCH = 2;
```

Run again with updated version or patch values to initiate the rollout.