Make a copy of the document if any changes are required for customer deliverables.

Please DO NOT edit this document.

Copy Instructions:

1. Click **File** -> **Make a Copy.**
2. Replace ‘Copy of ‘ from the **Name** text field with ‘<CUSTOMER> - ‘.
3. Replace ‘<SOLUTION_NAME> ’ with the Provider’s solution.
4. Select the desired folder to save a copy.
5. Select the **Copy comments and suggestions** box. This will also select the **Share it with the same people** box above it.

<CUSTOMER>
<SOLUTION_NAME>
Deployment Guide

Version 1
<AUTHOR>
<DATE>

Revision Summary

Date	Revision History	Comments
<REV_DATE>	1.0	Initial Version

snowflake

Table of Contents

Prerequisites

- The provider must accept the Snowflake Marketplace Provider Terms of Service.
- The consumer must accept the Snowflake Marketplace Client Terms of Service.
- The provider must determine which cloud regions the app will be available.
- The provider must create an account that serves as the main account for the ACF.
- The provider must create an account in each cloud region their native app will be available in. This account is used to collect events from consumer apps in each region. Events from this account are routed to the native app/ACF account via private data listings.
    - The Listing API is in Public Preview (PuPr). Visit https://other-docs.snowflake.com/en/progaccess/listing-progaccess-about for more information.
    - Once this account is created, the provider will set it as the Event account for the cloud region, by executing the following in the Snowflake Organization account, as `ORGADMIN`
        - `CALL SYSTEM$SET_EVENT_SHARING_ACCOUNT_FOR_REGION('<REGION>', 'PUBLIC', '<ACCOUNT_NAME>');`
            - <REGION> can be found by executing `SELECT CURRENT_REGION();`
            - <ACCOUNT_NAME> can be found by executing `SELECT CURRENT_ACCOUNT_NAME();`
        - `CALL SYSTEM$SYSTEM$ENABLE_GLOBAL_DATA_SHARING_FOR_ACCOUNT('<ACCOUNT_NAME>');`
            - <ACCOUNT_NAME> can be found by executing `SELECT CURRENT_ACCOUNT_NAME();`
- The user that will execute the scripts in each account must have either the `ACCOUNTADMIN` role granted or a role that can create roles and manage grants on the account to other roles.

Provider vs. Consumer

The Native Apps Framework term “provider” refers to <CUSTOMER>, as the app’s owner. The Native Apps Framework term “consumer” refers to <CUSTOMER> clients that install the app.

Consumption Tracking

Comments have been added to key framework objects to track Snowflake credit consumption of applications built using the Application Control Framework. **Please do not remove or modify comments.** Any modifications could result in the ability to properly track consumption.

Disclaimer

Any screenshots included in this guide are examples. Please refer to the text in the steps below when installing this app.

This code is not perfect, nor is it "production-ready". This is a template designed to demonstrate the concepts of creating an application on the Native Apps framework in Snowflake, with controls in place to monitor consumer usage and define access. This code is meant to kickstart client implementations.

Key Native App Components

Application Logic

Application logic is the critical component of the native app. It provides the intended functionality the consumer executes in their Snowflake account. Application logic is deployed in the form of one or more functions and/or stored procedures. Apps built by the ACF can support multiple functionalities and allow the provider the ability to control which consumers get access to what functionality.

Multiple Native App Modes

The framework comes with the ability to build a native app with custom functionality, depending on the version of the app. For example, the consumer can evaluate the “free” version of the app from the Snowflake Marketplace, without interaction from the provider. Once the consumer is interested in the one or more paid versions of the app, they can be granted access to the desired version.

By default, the ACF supports three app modes:

- **FREE**: a free version of the app that is publicly available in the Snowflake Marketplace. This version offers limited functionality, meant to entice the consumer to convert to a paid version of the app. Each consumer of this app version has the same entitlements/limits (i.e. five requests).

- **PAID**: a paid version of the app that is publicly available in the Snowflake Marketplace. This version offers more or complete app functionality. Each consumer of this app has the same entitlements/limits (if any) enforced (i.e. process 1MM records every 30 days).

- **ENTERPRISE**: a version of the app where unique entitlements/limits can be set for each consumer. The entitlements/limits are managed via the ACF’s App Control Manager. This is ideal for providers that want to create custom deals with consumers where the default entitlements/limits of the other app versions are not ideal for the consumer. Enterprise versions of the app should be listed privately and only made available to a single consumer.

Setup Script

The setup script of an application contains SQL statements, including application logic, that are run when the consumer installs or upgrades an application or when a provider installs or upgrades an application for testing. Every application must contain a setup script. The location of the setup script is specified in the manifest file.

The setup script defines the objects that are created when an application is installed or upgraded. For more information, visit https://docs.snowflake.com/en/developer-guide/native-apps/creating-setup-script.

The ACF includes a setup script template that is used to construct each app version/patch's setup script. The ACF's App Control Manager UI automatically generates the setup script, based on the selected table/view, functions, and procedures required for each version/patch.

Manifest File

The Snowflake Native App Framework requires that every application package contains a manifest file. This file defines properties required by the application package, including the location of the setup script and version definitions.

- The manifest file has the following requirements:
  - The name of the manifest file must be manifest.yml.
  - The manifest file must be uploaded to a named stage so that it is accessible when creating an application package or Snowflake Native App.

The manifest file must exist at the root of the directory structure on the named stage. For more information, visit https://docs.snowflake.com/en/developer-guide/native-apps/creating-manifest.

The ACF includes a manifest template that is used to construct each app version/patch's setup script. The ACF's App Control Manager UI automatically generates the manifest file.

Readme

There is a corresponding readme file that gets included when the consumer installs the corresponding version. Each readme is slightly different due to setup steps required for each app version.

Sidecar Loader

Sidecar is a utility that allows the consumer to execute pre-set commands in their account that cannot be executed by the application during installation. The Sidecar loads commands into a table to be executed by the consumer, via the SidecarRunner stored procedure. The commands are visible and can be evaluated prior to execution.

Deploy Application Control Framework

The provider's Native App is built and deployed on the Native Apps framework via Snowflake's Application Control Framework (ACF). The ACF is a framework that allows native app providers to build apps that allow providers to manage and understand consumer usage and create custom rules and controls that govern consumer app experience. The native app controls, rules, and consumers can be managed via the Application Control Manager Streamlit UI, included with the framework.

Deployment of the ACF is done in three steps. Each step is deployed via a setup script that must be deployed via SnowSQL. Each setup script requires a configuration file, which is defined below.

Configuration Parameters

Executing the event account scripts requires a set of parameters to be passed via a config file. The config file must contain a value for the following parameters, unless otherwise noted:

- **DIR** - the URI where the `application_control_framework` root directory resides on the provider's local machine
    - **NOTE**: The URI formatting differs depending on the provider's local operating system:
        - Linux/MacOS:
            - The path must include the initial forward slash in the path (e.g. `/Users/mhenderson/Documents/GitHub`.
            - If the directory path and/or filename includes special characters, the entire URI must be enclosed in single quotation marks.
        - Windows:
            - You must include the drive and backslash in the path (e.g. `C:\Users\mhenderson\Documents\GitHub`).
            - If the directory path and/or filename includes special characters, the entire file URI must be enclosed in single quotes. Note that the separator character is a forward slash (/) in enclosed URIs (e.g. `'C:\Users\mhenderson\Documents\GitHub Repos'` for a path in Windows that includes a directory named GitHub Repos).
- **ORG_NAME** - the organization provider's Snowflake account belongs (this can be found by executing the `SELECT CURRENT_ORGANIZATION_NAME()` command)
    - **NOTE**: only required for the **config_events**.txt file.

- **ACF_ACCOUNT_NAME** - the name of the provider's Snowflake where the ACF will be deployed (this can be found by executing the `SELECT CURRENT_ACCOUNT_NAME()` command)
    - **NOTE**: only required for the **config_events**.txt file.
- **ACF_ACCOUNT_LOCATOR** - the account identifier for provider's Snowflake account where the ACF will be deployed (this can be found by executing the `SELECT CURRENT_ACCOUNT()` command)
- **EVENTS_ACCOUNT_LOCATOR** - the account identifier for provider's Snowflake account where ACF events will be collected. (this can be found by executing the `SELECT CURRENT_ACCOUNT()` command)
    - **NOTE**: only required for the **config_events**.txt file.
- **APP_CODE** - an abbreviated representation for the name of the app (e.g. `SDE` for Sample Data Enrichment)

The following sections detail the order in which to deploy the ACF.

Step 1: Event Account Setup

The first step is to set up event account(s) for each region the native app will be deployed. The scripts create an event table (if one has not already been created), streams, tasks, and data listings to share consumer native app events from the event account to the main ACF account.

NOTE: Repeat this step for each event account created.

Prerequisites

The event account must have been created for each region the native app will be deployed, along with setting each account as the event account for the region. Each event account must be enrolled in the Listing API Private Preview. Please see the Prerequisites section for more information.

Script Execution

These scripts can be deployed either via SnowSQL or either in Snowflake Worksheets or an Integrated Development Environment (IDE) of choice.

NOTE: `<REGION>` = the cloud/region where the event account is being configured.

SnowSQL:

The event account scripts can be executed via SnowSQL, by calling the `setup.sql` script:

Execution: `./event_acct/setup/setup.sh config/config_events_<REGION>.txt`

non-SnowSQL

The ACF includes a python utility, `param_sub.py`, to replace parameter macros in the code with the values set in the config_events.txt file. The python script requires at least Python 2.7 and works with Python 3.

Execution: `python3 util/param_sub.py ../config/config_events_<REGION>.txt ../event_acct/scripts/ ../event_acct/scripts/`

Once complete, the following scripts are to be executed in order (either in a Snowflake worksheet or IDE of choice):

- `event_acct/scripts/01_account_setup_sub.sql`
- `event_acct/scripts/02_create_stream_events_procedure_sub.sql`
- `event_acct/scripts/03_create_remove_app_events_procedure`
- `event_acct/scripts/04_create_remove_all_events_procedure`

Supporting Events for Multiple Apps in the Same Event Account

The event account setup supports the ability to have multiple ACF-created apps' events located in the same event account. This avoids the provider having to create an account per application per region.

In the event the provider wants to leverage an existing event account to stream and share events to the main ACF account, the provider can execute the following command in the event account:

```
USE ROLE ACCOUNTADMIN;
USE WAREHOUSE <APP_CODE>_EVENTS_WH;
CALL EVENTS.EVENTS.STREAM_EVENTS(TO_ARRAY('<APP_CODES>'));
```

NOTES:

- `<APP_CODES>` = a comma-separated string of app code(s) to create streams/tasks for.
- While multiple app's events can reside in the same event account, the events must be shared to the same main ACF account. The ACF supports multiple deployments, for different applications, in the same account.

Step 2: ACF Account Setup

The second step is to deploy the ACF to the “main” account. This account is different from an event account and will be where the ACF will reside and be managed.

The ACF includes all of the objects required to manage ACF functionality. In addition, this script also deploys the App Control Manager, the Streamlit UI that is used to perform such actions as define custom controls and rules, create app package versions, onboard consumers, etc.

Prerequisites

It’s recommended that event accounts are created for each region the native app will be deployed in before deploying the ACF. The ACF can be deployed without the event accounts created. Please see the Step 1: Event Account Setup section for more information.

Script Execution

The ACF scripts can only be deployed via SnowSQL due to the requirement to use the `PUT` command to place files onto a stage.

SnowSQL:

The framework scripts can be executed via SnowSQL, by calling the `setup.sql` script:

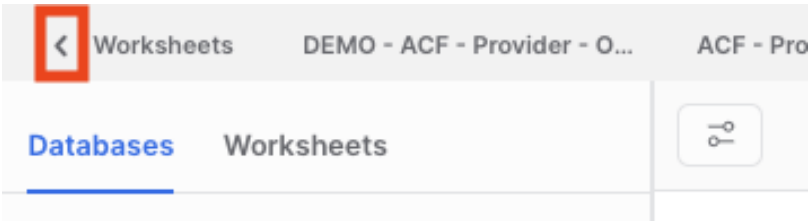
Example: `./main_acct/setup/setup.sh config/config_acf.txt`

Step 3: Mount the Event Shares

Once the ACF has been deployed, the shares from the event account(s) can be mounted in the ACF account. The following step details how to mount the share from each event account set up from Step 1.

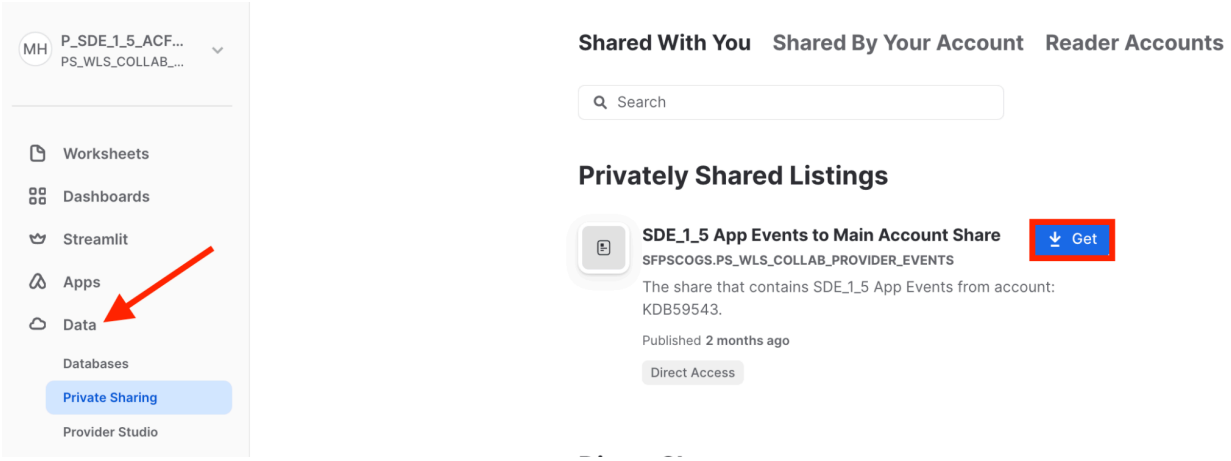
Step 1: Log into **Snowsight**.

Step 2: Once logged in, if not at the Snowsight home screen, click the **Back** button, in the top left area of the UI, to open the left navigation menu.

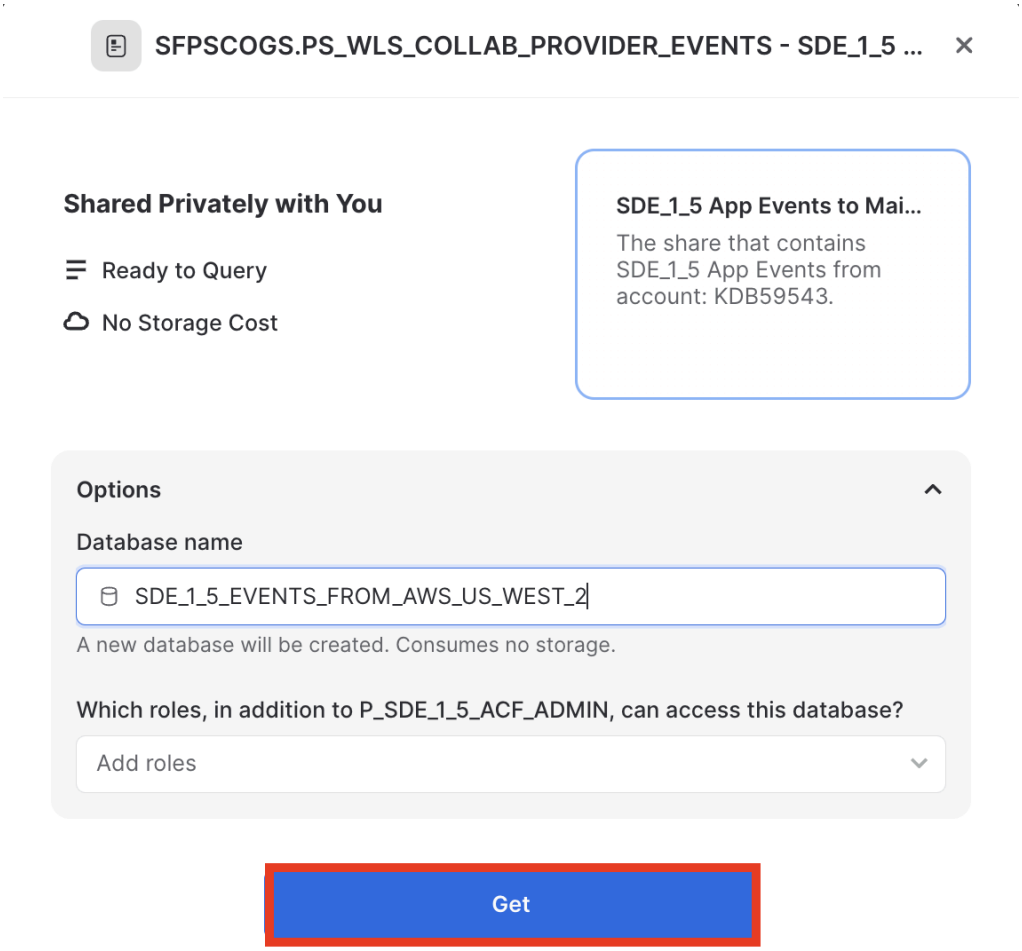


Step 3: Switch to the **P_<APP_CODE>_ACF_ADMIN** role, by clicking the drop-down in the bottom left area of the UI, then hovering over the Switch Role menu item.

Step 4: Click **Data Products**, then **Private Sharing,** then **Get**.



Step 5: Click **Get** (**NOTE**: do not rename the database).



Step 6: Repeat **Steps 1-5** for each remaining event listing.

Step 7: Create a stream and tasks to stream events from each mounted event database to the EVENTS_MASTER table.

```
USE ROLE P_<APP_CODE>_ACF_ADMIN;
USE WAREHOUSE P_<APP_CODE>_ACF_WH;
CALL
P_<APP_CODE>_ACF_DB.EVENTS.STREAM_TO_EVENT_MASTER(TO_ARRAY('<EVENT_DBS>'));
```

<EVENT_DBS> = a comma-separated list of each event database created in **Steps 1-6**.

Step 4: Dev Environment Setup

The ACF also comes with a set of scripts that creates the provider's dev environment. This dev environment includes the source data, application logic functions, and/or procedures to be included in the Native App. In addition, the dev environment also includes the objects required to test the application functions/procedures locally, to ensure desired functionality, before building the native app.

The dev environment mimics the native app database structure. See the Application section in the **Application Control Framework - Detailed Design** document for more information.

Once deployed, the provider can create any required functions in the `P_<APP_CODE>_SOURCE_DB_DEV` database's `FUNCS_APP` schema and any required procedures can be created in the database's `PROCS_APP` schema.

NOTE: `<APP_CODE>` = an abbreviated representation for the name of the app (e.g. `SDE` for Sample Data Enrichment)

Script Execution

These scripts can only be deployed via SnowSQL due to the requirement to use the `PUT` command to place files onto a stage.

SnowSQL:

The framework scripts can be executed via SnowSQL, by calling the `setup.sql` script:

Execution: `./main_acct/dev_env/setup/setup.sh config/config_acf.txt`

Objects Created

Refer to the **Application Control Framework - Detailed Design** document for a list of objects created via the scripts.

Step 5: Demo App Setup (Optional)

The ACF also comes with a set of scripts that creates objects in the provider's dev environment to build a demo data enrichment app. The demo app includes sample source graph data, a function that can be shared with the consumer, and a stored procedure that enriches consumer data with data from the source graph.

This step is optional, but could prove valuable to providers wanting to practice building native apps using the ACF, before building their own app.

Script Execution

These scripts can only be deployed via SnowSQL due to the requirement to use the `PUT` command to place files onto a stage.

SnowSQL:

The framework scripts can be executed via SnowSQL, by calling the `setup.sql` script:

Execution: `./main_acct/demo_app/setup/setup.sh config/config_acf.txt`

Objects Created

Refer to the **Application Control Framework - Detailed Design** document for a list of objects created via the scripts.

Remove Application Control Framework

The following steps detail the removal process.

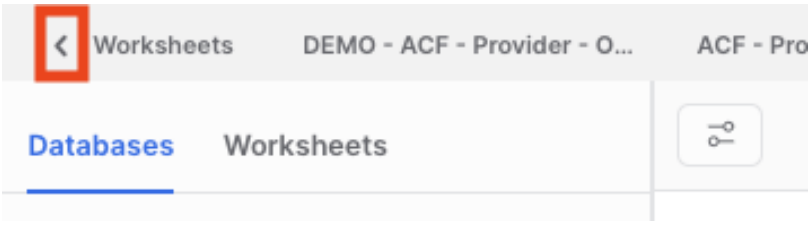
Step 1: Remove Listing from the ACF account

The following details how to remove the listing from the ACF account. This is required in order to remove the native app.

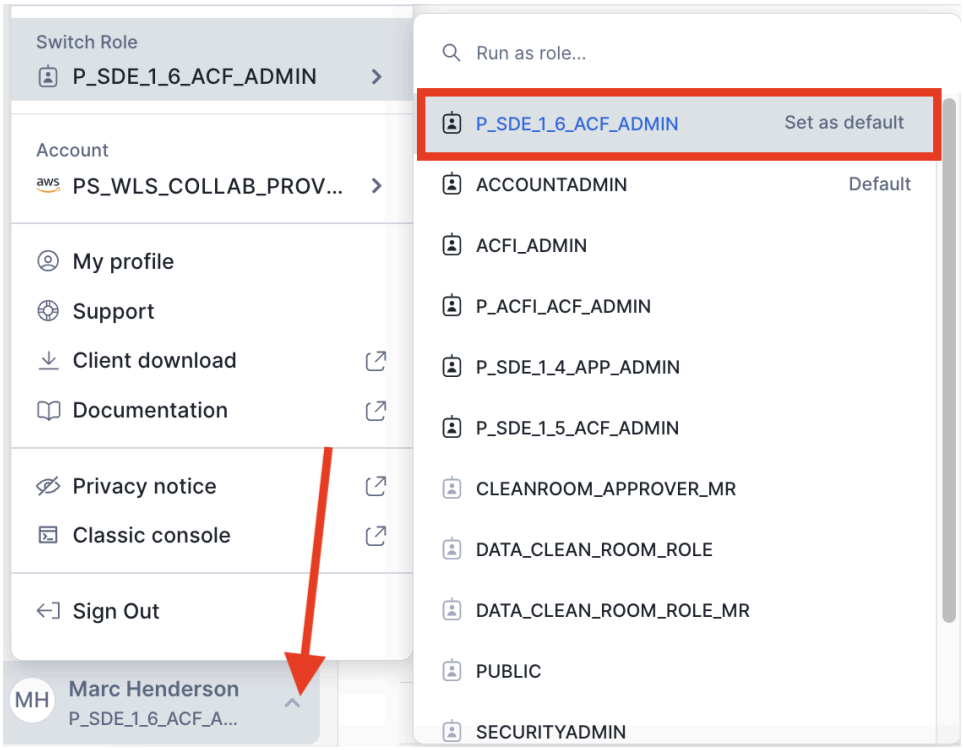
NOTE: This does not remove the application from the provider's account. Consumers will no longer have access to the application.

Step 1: Log into **Snowsight**.

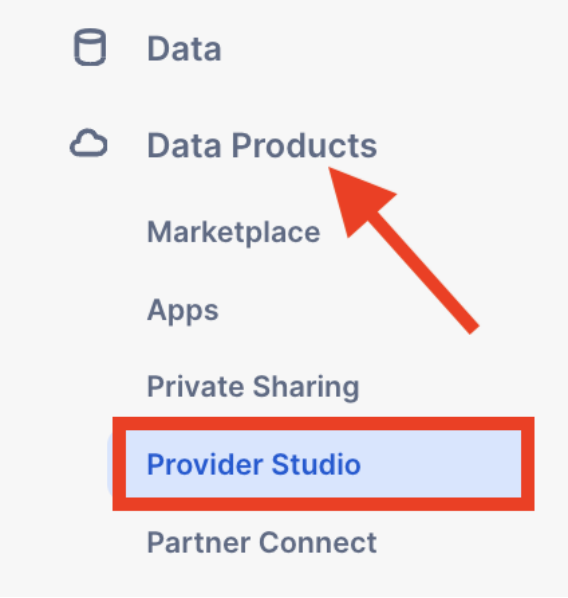
Step 2: Once logged in, if not at the Snowsight home screen, click the **Back** button, in the top left area of the UI, to open the left navigation menu.



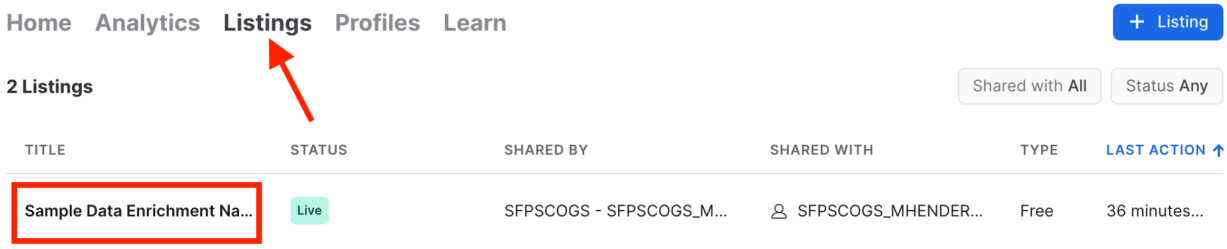
Step 3: Switch to the **P_<APP_CODE>_ACF_ADMIN** role, by clicking the drop-down in the bottom left area of the UI, then hovering over the Switch Role menu item.



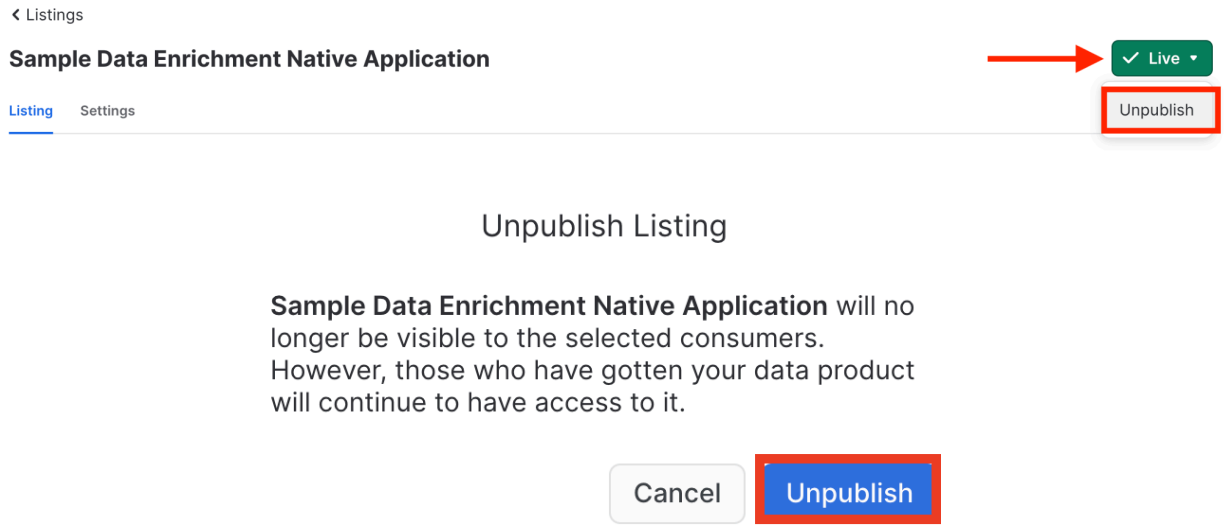
Step 4: Click **Data Products**, then **Provider Studio**.



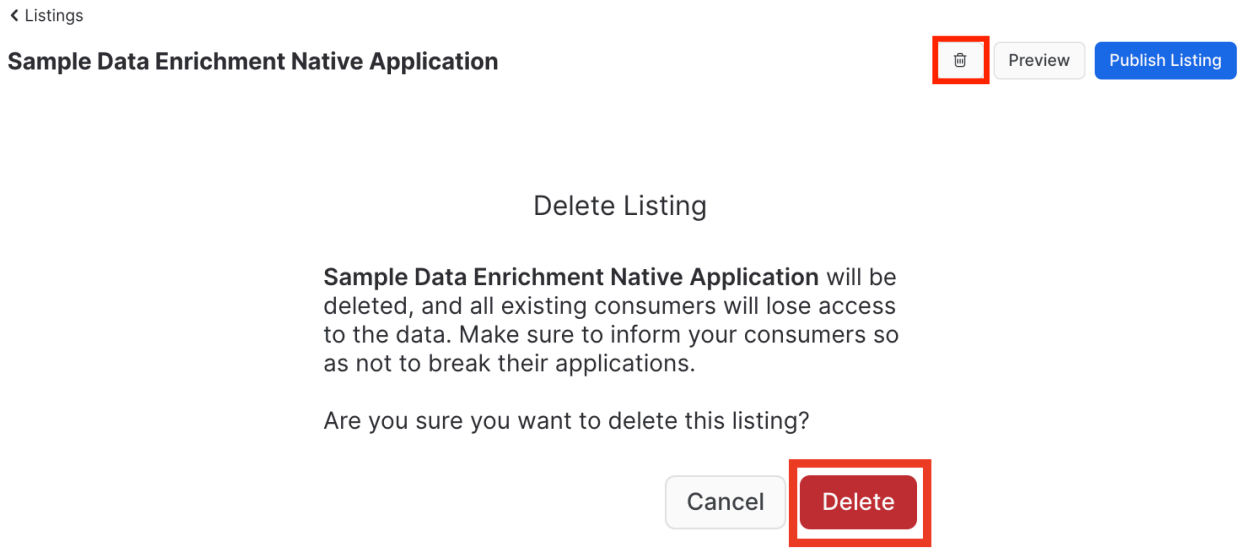
Step 5: Click the **Listings** tab and select the listing for this application.



Step 6: Click ✓ **Live**, then **Unpublish**. When the dialog box appears, click **Unpublish**.



Step 7: Delete the listing, by clicking the **trashcan** icon. When the dialog box appears, click **Delete**.

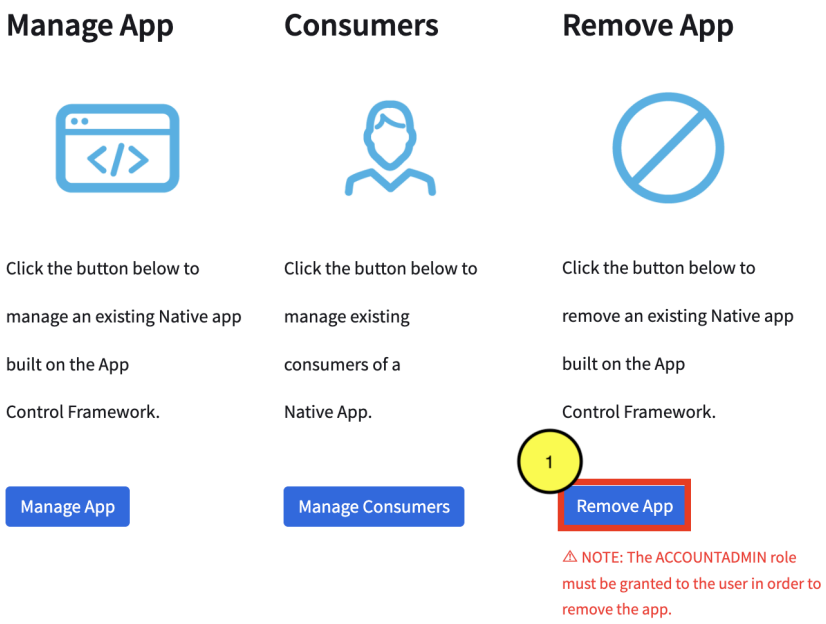


Step 2: Remove App and ACF

The following steps detail how to remove In the event the provider should remove the native app, and the Application Control Framework itself (including all consumer logs/metrics and metadata):

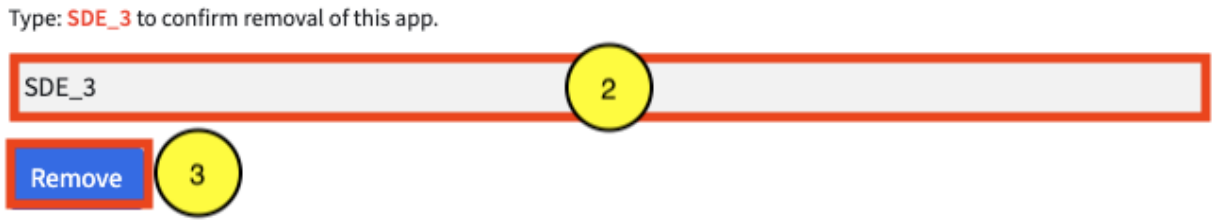
NOTE: a user granted `ACCOUNTADMIN` is required to perform this action

Step 1: In the App Control Manager, click **Remove App**.



Step 2: Type the **App Code** (in **red**) to confirm removal.

Step 3: Click **Remove**.



Step 3: Remove Events from Event Account

The next step is to remove the app's events and the objects used to stream and share events to the main account from the event account.

On a worksheet in the event account, execute the following:

```
USE ROLE ACCOUNTADMIN;
USE WAREHOUSE <APP_CODE>_EVENTS_WH;
CALL EVENTS.EVENTS.REMOVE_APP_EVENTS(TO_ARRAY('<APP_CODE>'));
```

NOTES:

- `<APP_CODES>` = a comma-separated string of app code(s) to create streams/tasks for.
- Events for multiple apps can be removed at once, using this command.

Step 4: Remove Events from Event Account (Optional)

To fully remove the event table, shares, listings, etc created in the event account, on a worksheet, execute the following:

```
USE ROLE ACCOUNTADMIN;
USE WAREHOUSE <APP_CODE>_EVENTS_WH;
CALL EVENTS.EVENTS.REMOVE_ALL_EVENTS();
```