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>
Native App Deployment Guide

Version 1
<AUTHOR>
<DATE>

Revision Summary

Date	Revision History	Comments
<REV_DATE>	1.0	Initial Version

snowflake

Table of Contents

Prerequisites **5**
Provider vs. Consumer **5**
Disclaimer **5**
Key Native App Components **6**
Environments 6
Dev to Prod Process Flow 6
Application Logic 6
Multiple Native App Modes 7
Setup Script 7
Manifest File 8
Readme 8
Sidecar Loader 8
App Control Manager **9**
Build and Deploy Native App **10**
Step 1: Create Application Logic 10
Custom Logging/Metrics 11
Step 2: Modify/Add Controls (optional) 14
Step 3: Create Custom Rules (optional) 16
Step 4: Test the Application Logic 18
Step 5: Build the Native App 18
Part 1: Create Test Application Package 18
Source Data Ownership 21
Part 2: Create/Patch a Version for the Application Package 22
Part 3: Release Patch 28
Step 6: Create Test Application Listing 29
Step 7: Onboard Consumer (ENTERPRISE only) 30
Step 8: Manage Consumer Controls (optional) 32
Step 9: QA/Test the Native App 34
Step 10: Promote Application Package to Prod 34
Step 11: Create Listing for Production-ready App 35
Removals **36**
Remove Consumer 36
Remove Listing 37
Drop Version 40
Remove Application Package 42
Remove ACF 43

Prerequisites

- The user that will build and deploy the native app must be granted the `P_<APP_CODE>_ACF_ADMIN` role.
- For testing purposes, a test consumer account should be created in the same organization as the provider's account.

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

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.

Disclaimer

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

Key Native App Components

Environments

The ACF supports having application code/objects in separate dev and prod environments. This allows the provider the ability to work on the various components of the native app (application logic, Streamlit, templates, etc.) separately in the dev environment, without impacting any of the production application packages. Once changes in the dev environment pass testing, they can be promoted to the prod environment.

For example, if the provider wants to make a change to only the production native app's Streamlit UI, the provider can make changes to the Streamlit code in the dev environment. Once ready to test, the provider will create a test application package and version, referencing the updated Streamlit code in the dev environment. The provider will reference the application logic and template files already in production, since those components are not affected.

Dev to Prod Process Flow

1. Create a Test application package for the dev environment code/objects.
2. Create/Patch a version for the Test application package
3. Create a private test Marketplace listing
4. In a QA/Test consumer account, install the application from the private Marketplace listing.
5. In a QA/Test consumer account, Test the application package version
6. Iterate until the app successfully passes testing.
7. Promote the application package's code/objects to prod
8. Create a Production application package
9. Create/Patch a version for the Production application package
10. Create a public Marketplace listing (unless the version of the app should be privately listed (i.e. enterprise)). See Multiple Native App Modes for more information.

This document will provide step-by-step instructions for promoting code/objects from dev to prod.

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 stored procedures. Apps built by the ACF can support multiple

functionality/procedures and allows 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 contains SQL statements, including application logic DDL, that are executed 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 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 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

A readme file is 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.

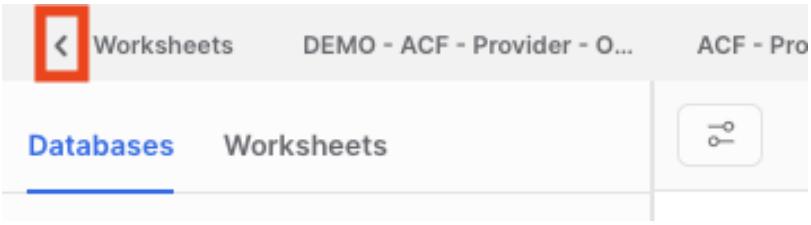
App Control Manager

The Application Control Framework includes the App Control Manager, a Streamlit UI available in the provider account. The App Control Manager allows the provider to easily build and manage an app built on the ACF, manage consumers, and remove the ACF if/when desired.

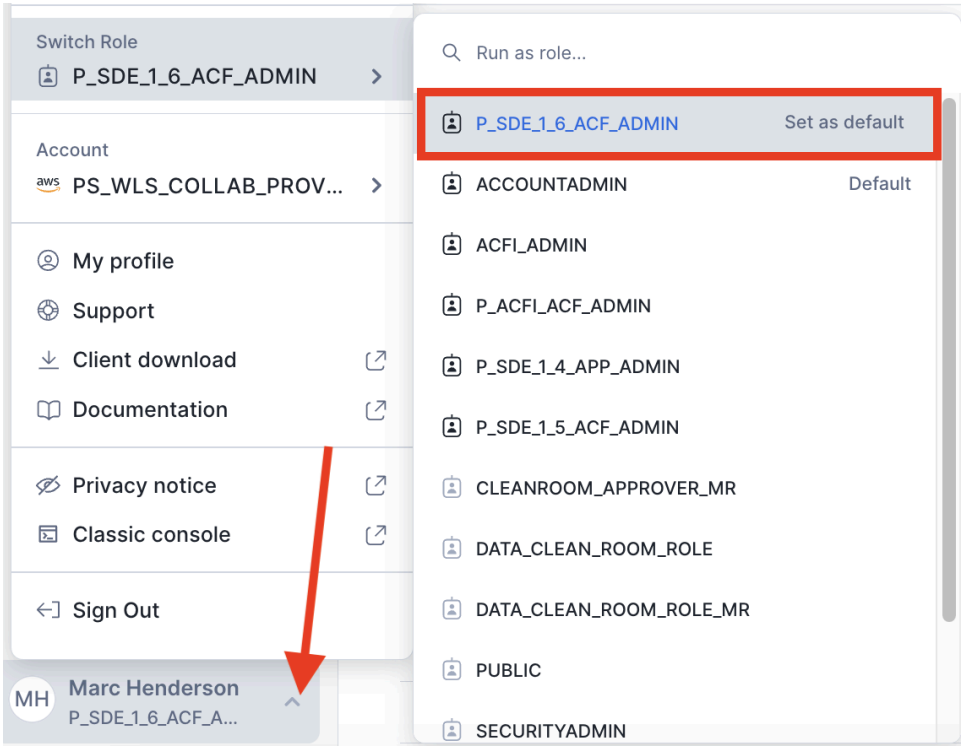
The App Control Manager can be accessed via the following steps:

Step 1: Log into **Snowsight**.

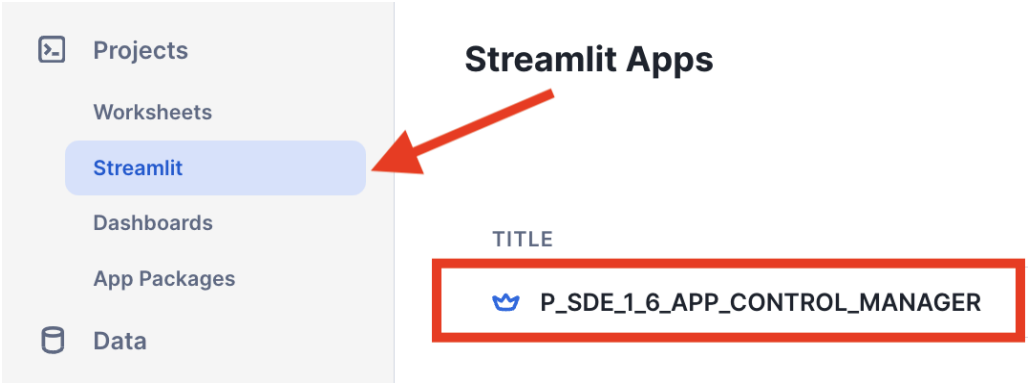
Step 2: Once logged in, if not at the Snowsight home screen, click the **Back** button, in the top left area of the UI, to open the left navigation menu.

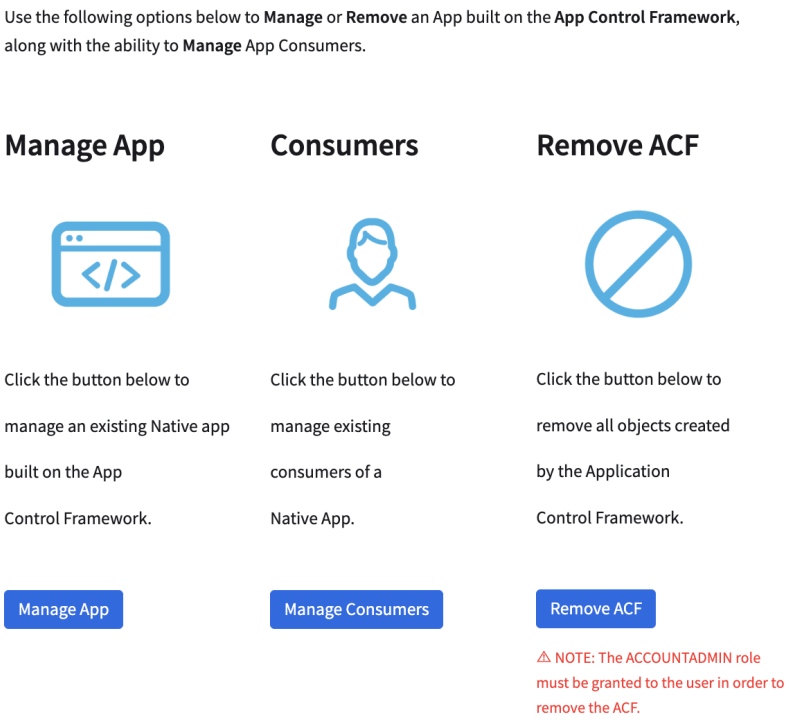


Step 3: Switch to the **P_<APP_CODE>_ACF_ADMIN** role, by clicking the drop-down in the bottom left area of the UI, then hovering over the Switch Role menu item.



Step 4: Click **Projects** >> **Streamlit**, then **P_<APP_CODE>_APP_CONTROL_MANAGER**.





(App Control Manager Home)

Build and Deploy Native App

Step 1: Create Application Logic

The Application Control Framework includes a dedicated dev environment, where the application logic can be created and tested. Application logic includes stored procedures and/or functions required for the native app to successfully execute the intended behavior.

Functions used by the native app are to be created in the `P_<APP_CODE>_SOURCE_DB_DEV.FUNCS_APP` schema. Stored procedures used by the native app are to be created in the `P_<APP_CODE>_SOURCE_DB_DEV.PROCS_APP` schema.

Custom Logging/Metrics

The application logic will likely need to collect additional logs/metrics. The ACF's `APP_LOGGER` stored procedure is used to log specific app events and collect stats about each request. `APP_LOGGER` is available to the provider to use to log relevant events and/or stats.

NOTE: the value passed to the `message` parameter is a JSON payload containing details of the metrics being collected. This can be any value, but if the value is a string, it must be enclosed in double quotes.

APP_LOGGER Signature

Parameter	Type	Description
account_locator	VARCHAR	The consumer's Snowflake Account Locator
consumer_name	VARCHAR	The consumer's name
app_key	VARCHAR	The consumer's app key for the installation.
app_mode	VARCHAR	The app's mode (i.e. FREE, PAID, or ENTERPRISE)
entry_type	VARCHAR	The type of entry (i.e. LOG or METRIC)
event_type	VARCHAR	The type of event being logged
event_attributes	VARCHAR	Any applicable attributes associated with the event type (can be NULL). **NOTE**: if the message is a string, it **should be enclosed in double quotes** (i.e. '"this is a test event"').
timestamp	TIMESTAMP_NTZ	The UTC timestamp for the event
status	VARCHAR	The status of the event (i.e. PROCESSING, COMPLETE, or ERROR)
message	VARCHAR	The message to log. **NOTE**: if the message is a string, it should be enclosed in double quotes (i.e. '"this is my message"').

`APP_LOGGER` can be called directly from the application logic stored procedure(s):

Example APP_LOGGER Call - Log Event:

```
CALL UTIL_APP.APP_LOGGER(
      '<CONSUMER_ACCOUNT_LOCATOR>'
      ,'<CONSUMER_NAME>'
      ,'9224FFD050BE936E27DBE'
      ,'enterprise'
      ,'log'
      ,'install'
      ,'""'
      ,SYSDATE()
      ,'COMPLETE'
      ,'"install successful. app key generated: 9224FFD050BE936E27DBE"'
);
```

Example Log Event JSON Payload:

```
{
 "account": "<CONSUMER_ACCOUNT_LOCATOR>",
 "app_key": "9224FFD050BE936E27DBE",
 "app_mode": "enterprise",
 "consumer_name": "<CONSUMER_NAME>",
 "entry_type": "log",
 "event_attributes": "",
 "event_type": "install",
 "message": "install successful. app key generated: 9224FFD050BE936E27DBE",
 "status": "COMPLETE",
 "timestamp": "2023-05-03 15:51:58.748"
}
```

Example APP_LOGGER Call - Metrics:

```
      CALL UTIL_APP.APP_LOGGER(
      '<CONSUMER_ACCOUNT_LOCATOR>'
      ,'<CONSUMER_NAME>'
      ,'9224FFD050BE936E27DBE'
      ,'enterprise'
      ,'metric'
      ,'request'
      ,'[{"request_id":"afadsf-3431232"}, {"proc_name":"enrich"},
{"proc_parameters":"<APP_CODE>,C_<APP_CODE>_HELPER_DB.SOURCE.ENRICH_CRM_V,EMAIL
,<APP_CODE>.RESULTS_APP.ENRICH_CRM_RESULTS"}]'
      ,SYSDATE()
      ,'PROCESSING'
      ,'{
            "metric_type":"request_summary",
            "metrics":{
```

```
"input_table_name":"C_<APP_CODE>_HELPER_DB.SOURCE.ENRICH_CRM_V",
                    "input_record_count":5000000,
                    "results_table_name":"",
                    "results_record_count":0,
                    "results_record_count_distinct":0,
                    "comments":"",
                    "submitted_ts":"'||SYSDATE()||'",
                    "completed_ts":""
             }
      }'
);
```

NOTE: `request_summary` is the type of metric being collected. As many metric types as required can be collected.

Example Metric Event JSON Payload:

```
{
 "account": "<CONSUMER_ACCOUNT_LOCATOR>",
 "app_key": "9224FFD050BE936E27DBE",
 "app_mode": "enterprise"
 "consumer_name": "<CONSUMER_NAME>",
 "entry_type": "metric"
 "event_attributes": [
 {
  "request_id": "afadsf-3431232"
 },
 {
  "proc_name": "enrich"
 },
 {
  "proc_parameters":
"<APP_CODE>,C_<APP_CODE>_HELPER_DB.SOURCE.ENRICH_CRM_V,EMAIL,<APP_NAME>.RESULTS
_APP.ENRICH_CRM_RESULTS"
 }
 ],
 "event_type": "request",
 "message": {
 "metric_type": "request_summary",
 "metrics": {
  "comments": "",
  "completed_ts": "",
  "input_record_count": 5000000,
  "input_table_name": "C_<APP_CODE>_HELPER_DB.SOURCE.ENRICH_CRM_V",
  "results_record_count": 0,
```

```
  "results_record_count_distinct": 0,
  "results_table_name": "",
  "submitted_ts": "2023-05-03 15:55:39.846"
 }
 },
 "status": "PROCESSING",
 "timestamp": "2023-05-03 15:55:40.141"
}
```


Step 2: Modify/Add Controls (optional)

The Application Control Framework has built-in preset controls with default values, defined in the METADATA_DICTIONARY table. The control default values can be updated, as required. Additional controls can be added, as required, by the application logic. Additional controls can be used to track additional consumer metrics, enforce custom rules, etc. The controls and their default values are set for each onboarded consumer (unless overridden - which is covered later in this document). Controls can be modified/added via the ACF's App Control Manager.

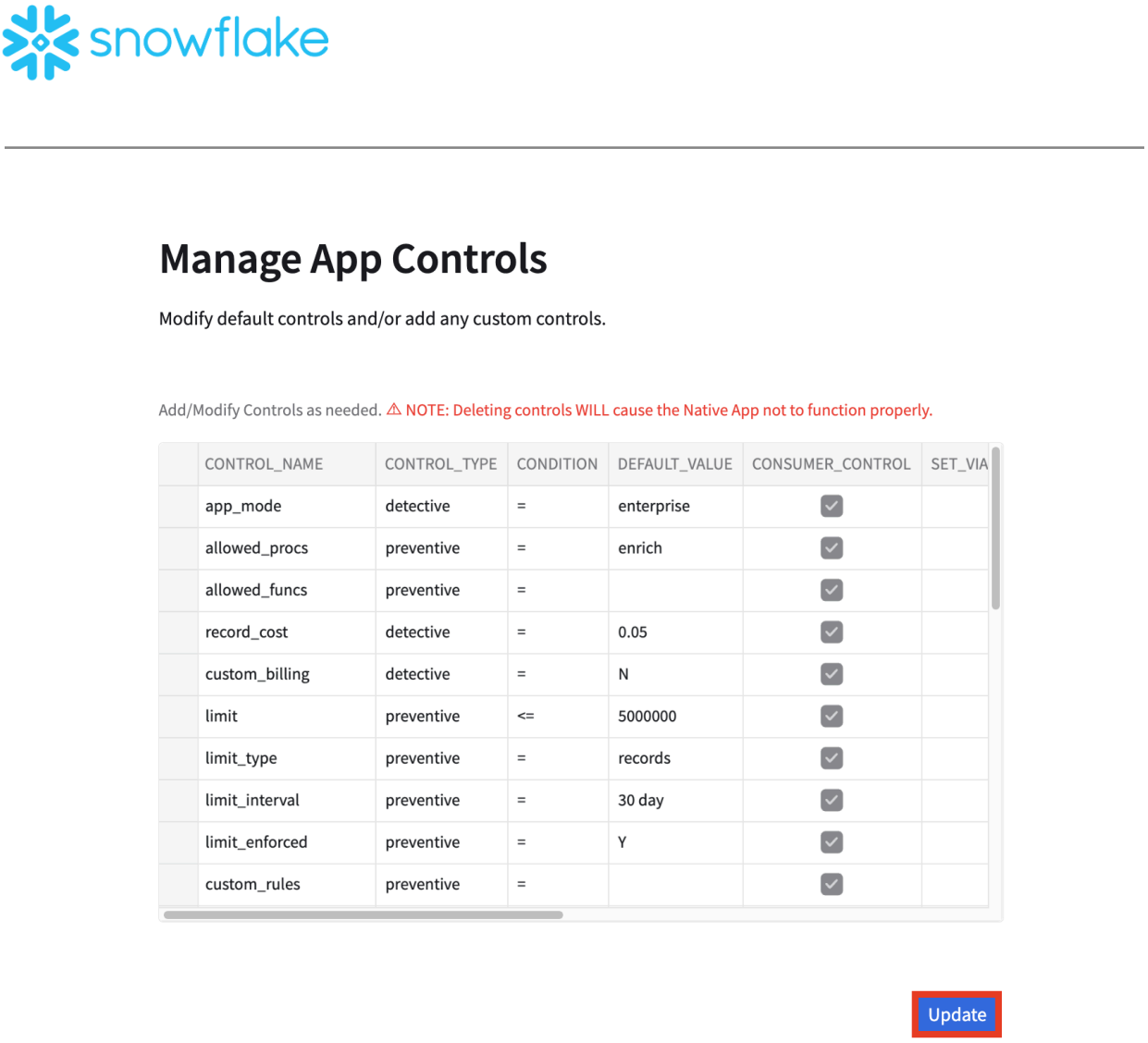
NOTE: existing controls should **NOT** be removed. This will result in app failure.

Step 1: In the App Control Manager, click **Manage App** >> **Controls**.

Step 2: Modify/Add controls as needed.

- **Field Definitions**:
  - **control_name** - the name of the control.
  - **control_type** - type of control (preventive, detective, deterrent, or corrective); optional.
  - **condition** - operand in relation to the default_value (=, >, >=, <, <=, etc.)
  - **default_value** - the control's default value, if applicable.
  - **consumer_control** - flag indicating whether the control is consumer-related. Some controls may not be consumer-related, but provider-related (i.e. the provider_secret control that stores the key used to encrypt logs/metrics).
  - **set_via_onboard** - flag indicating whether the control can be overridden when a consumer is onboarded. Not all controls should be available to be overridden (i.e. controls used to track certain types of events, such as installs, requests, etc.).
  - **consumer_visible** - flag indicating whether the control is visible to the consumer, when they install the app. The provider may want to hide some controls from the consumer.
  - **description** - the description of the control

Step 3: Click **Update**.



NOTE: If the custom control that is added is a metric based on the consumer's usage of the app, the ACF's PROCESS_CONSUMER_EVENTS stored procedure may need to be updated to process events to update the control accordingly. Please refer to the **Application Control Framework - Detailed Design** document for more details.

Step 3: Create Custom Rules (optional)

By default, the Application Control Framework enforces rules that regulate consumers' usage of the native app based on either the number of records processed or requests within a predefined interval (defined either in the METADATA_DICTIONARY or overridden when the consumer is onboarded).

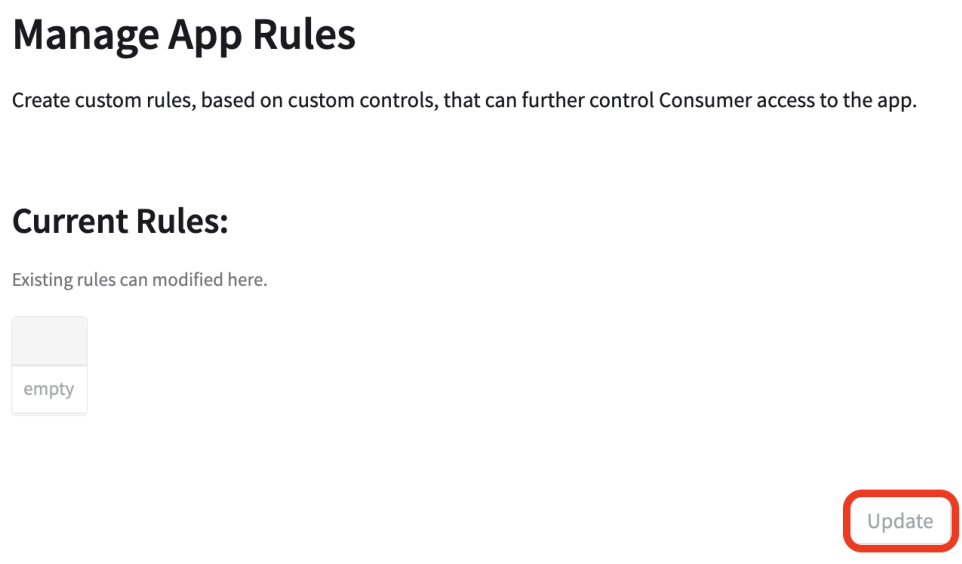
The provider can add additional custom rules, if applicable, to regulate usage of the native app, based on either controls defined in the METADATA_DICTIONATY or custom fields/conditions. The custom rules are created in the style of an if-statement and can be as complex as required, with multiple conditions joined together either by AND/OR within a single group, or each condition can be its own group. Each custom rule can be assigned to consumers, as required.

NOTE: the consumer can only be assigned one custom rule.

Custom roles can be created/updated via the ACF's App Control Manager:

Step 1: In the App Control Manager, click **Manage App** >> **Rules**.

Step 2: If updating an existing rule, modify the rule(s), as applicable, in the Current Rules section, then click **Update** (the **Update** button is only enabled when there is an existing rule).



Step 3: If creating a new rule, click **+ Rule**.

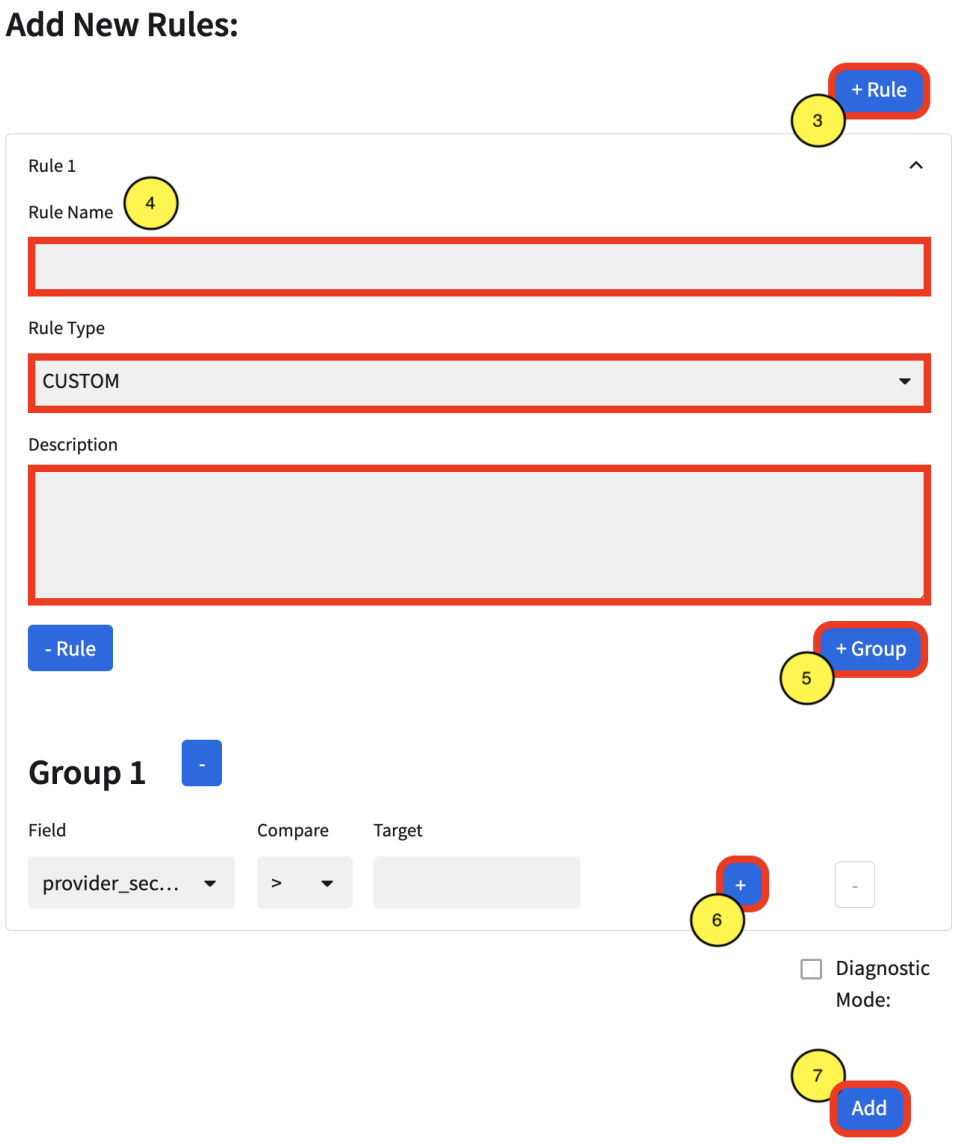
NOTE: multiple rules can be added/removed as required. Rules can be removed by clicking **- Rule**.

Step 4: Add a Rule Name, and Description (currently the only Rule Type is **CUSTOM**).

Step 5: In Group 1, select the Field, the Compare operand, and the Target value for the first condition. Click the (**+**) button to repeat for additional conditions. Conditions can also be removed by clicking the **-** button, next to each condition.

Step 6: If Group 1 should be compared to another group, click **+ Group** and repeat Step 5. Groups can also be removed by clicking the (**-**)button next to each group name.

Step 7: Once the rule groups/conditions are set, click **Add**.



Step 4: Test the Application Logic

The Application Control Framework also includes scripts to build the provider's dev environment. This environment includes the source data, functions, and/or procedures that will be included in the provider's native app. Please refer to the **Application Control Framework - Detailed Design** document for a description of the objects created in this database.

Rather than building the application each time to test, the provider can test functions/procedures, as they will function in the native app, directly in their dev environment. The provider can call the functions/procedures directly from the dev environment.

Example:

```
--testing ENRICH stored procedure
USE ROLE P_<APP_CODE>_ACF_ADMIN;
USE WAREHOUSE P_<APP_CODE>_ACF_WH;

CALL P_<APP_CODE>_SOURCE_DB_DEV.PROCS_APP.ENRICH('<APP_CODE>'
,P_<APP_CODE>_SOURCE_DB_DEV.SOURCE_SCHEMA.SOURCE_TABLE,EMAIL,
,'P_<APP_CODE>_SOURCE_DB_DEV.RESULTS_APP.RESULTS_TABLE');
```

For the provider's convenience, the dev environment is created in the ***P_<APP_CODE>_SOURCE_DB_DEV*** database. Once each function/procedure passes testing, the objects are ready to be included in the native app. The ACF's App Control Manager streamlines the native app build and release process. The subsequent sections will outline the App Control Manager, and how to build and release a native app.

Step 5: Build the Native App

Once the Application Control Framework's Controls and Rules have been confirmed/updated and the application functions and/or procedures have been successfully tested, the provider can build the native app. The ACF's App Control Manager can be used to build the native app. The following sections outline how to use the App Control Manager to build the native app.

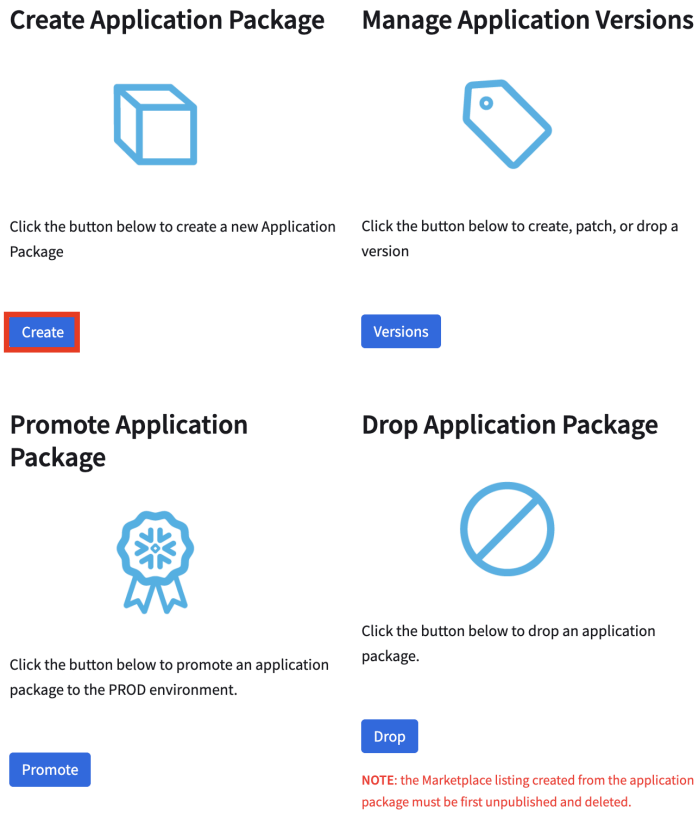
Part 1: Create Test Application Package

The native app's application package consists of views of the native app's source data, along with views of the required Application Control Framework tables used to enforce the rules defined in the ACF and collect logs/metrics. Each native app listed in the Marketplace can have only one application package.

The ACF's App Control Manager allows the provider to create an application package by choosing the source data object(s) required for the native app, manage its versions and releases, and drop the application package.

The following steps detail how to create an application package via the App Control Framework:

Step 1: In the App Control Manager, click **Manage App** >> **App Package** >> **Create**.



Step 2: Provide a name for the application package.

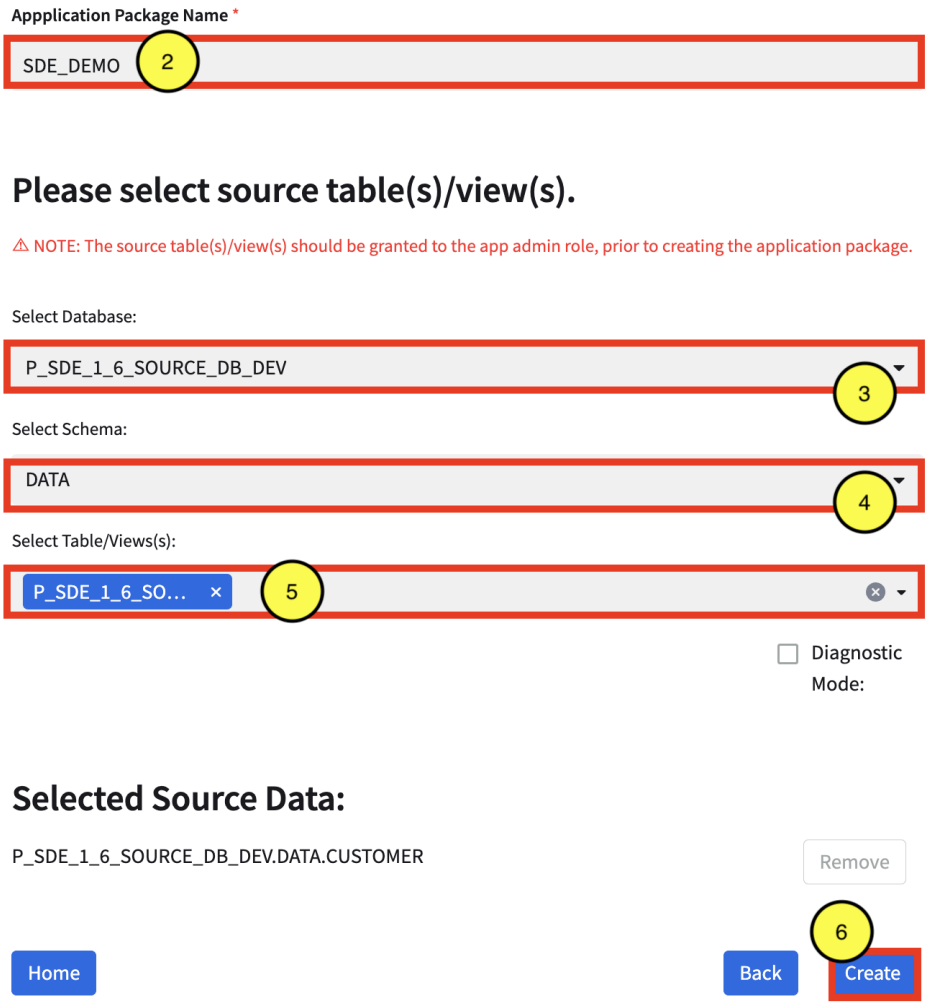
NOTE: This name will be prefixed by: **P_<APP_CODE>_APP_PKG_**. When creating the test application package, it is recommended to include a term the describes that this is a testing application package (i.e. `'TEST'` or `'QA'`)

Step 3: If the P_<APP_CODE>_ACF_ADMIN role created the source data, select the **P_<APP_CODE>_SOURCE_DB_DEV** database. If the role does not own the source data, skip to **Step 6**, then see the Source Data Ownership section.

Step 4: Select the **DATA** schema from the Select Schema drop-down

Step 5: Select the applicable tables/views.

Step 6: Click **Create**.



Source Data Ownership

A common occurrence when building native apps with the ACF is that the source data may not be owned by the **P_<APP_CODE>_ACF_ADMIN** role. When this occurs, the **P_<APP_CODE>_ACF_ADMIN** role must grant the data owner role privileges to the application package(s) and a stored procedure, **APP_PKG_SOURCE_VIEWS**, that manages the grants. This stored procedure can either **grant** or **revoke** privileges to source data. Once the **P_<APP_CODE>_ACF_ADMIN** role has granted privileges to the data owner role, the data owner must then create views of the source data and grant the views to the application package.

NOTE: The data owner role should also create a test copy of the source data in the **P_<APP_CODE>_SOURCE_DB_DEV.DATA** schema. This copy should consist of data that the **P_<APP_CODE>_ACF_ADMIN** can access and successfully works with the application logic.

Example (Data Owner gets privileges to app packages):

```
USE ROLE P_<APP_CODE>_ACF_ADMIN;
USE WAREHOUSE P_<APP_CODE>_ACF_WH;
CALL P_<APP_CODE>_ACF_DB.UTIL.GRANTS_TO_DATA_OWNER(TO_ARRAY('<PKG_LIST>'),
'<DATA_OWNER_ROLE>');
```

Example (Data Owner creates source data views and grants to app packages):

```
USE ROLE <DATA_OWNER_ROLE>;
USE WAREHOUSE <DATA_OWNER_WH>;
CALL P_<APP_CODE>_ACF_DB.UTIL.APP_PKG_SOURCE_VIEWS(TO_ARRAY('<TABLE_LIST>'),
TO_ARRAY('<PKG_LIST>'), '<ACTION>');
```

NOTES:

`<APP_CODE>` = the app's app code

`<PKG_LIST>` = the comma-separated list of application packages (i.e. `'pkg1,pkg2,pkg3'`)

`<DATA_OWNER_ROLE>` = the role that owns the source data

`<TABLE_LIST>` = the comma-separated list of source tables

`<ACTION>` = the action to take when calling the **APP_PKG_SOURCE_VIEWS** procedure. The only actions accepted are `GRANT` and `REVOKE`

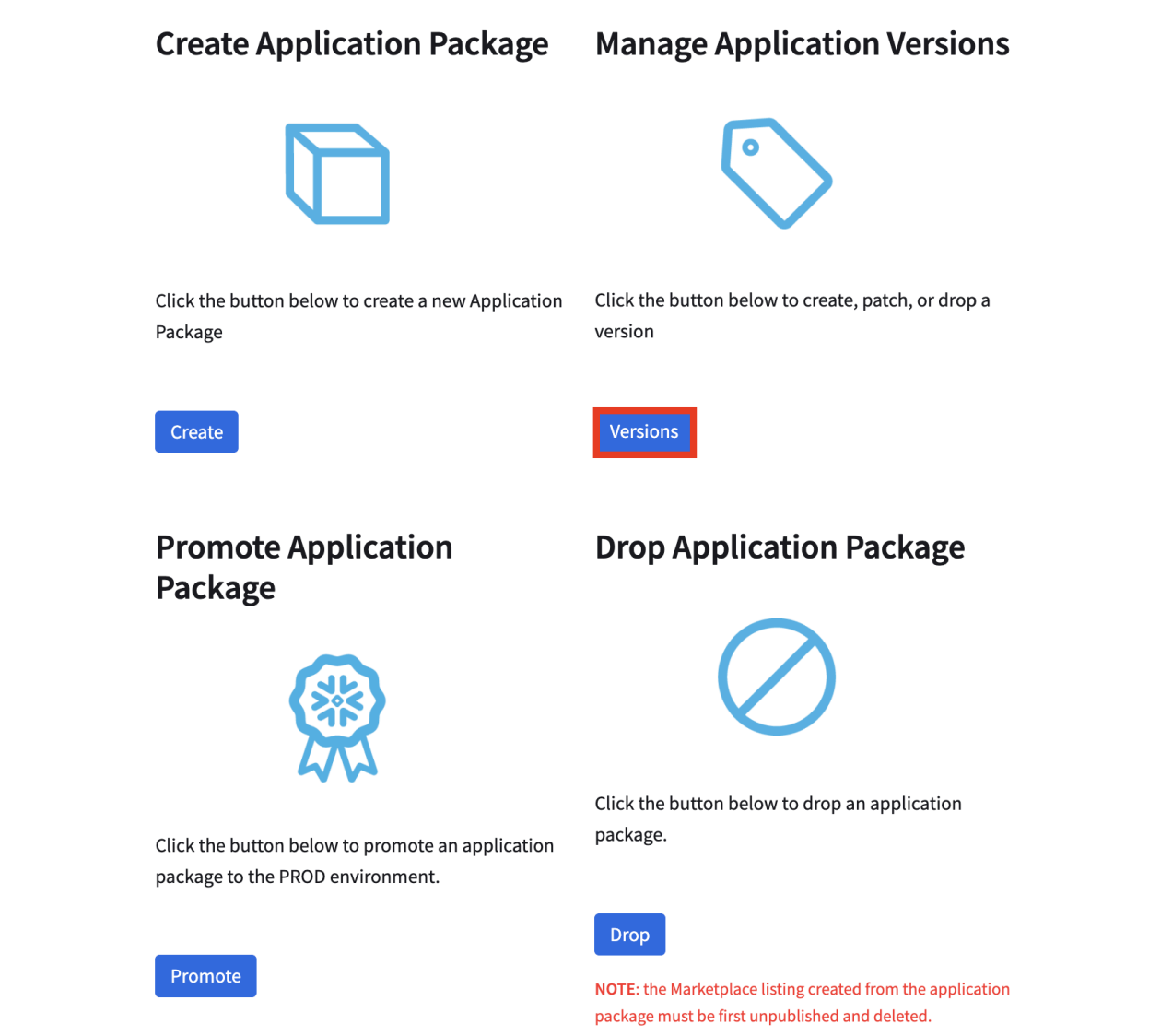
Part 2: Create/Patch a Version for the Application Package

Once the application package is created, the application functions/procedures are tied to the application package by creating a version and a patch. When creating a new version, its initial patch number is 0. Each version can have up to 130 patches. Each patch has a setup script that contains the objects to be created in the consumer account, along with a manifest file that defines the privileges the app needs in the consumer account. For each version, a stage is created to store the setup script and the manifest files. When new versions or patches are created, the consumer upgrades the installed native app to get access.

The App Control Manager streamlines version management by allowing the provider to create, patch, or drop a version for each application package. When creating/patching a version, the provider can select the previously-tested functions/procedures directly from the dev environment. The function/procedure's DDL are dynamically added to the setup script, allowing them to be created in the consumer's account, during installation. An application package can only have 2 active versions at any given time.

The following steps detail how to create, patch, or drop a version and promote an application package to prod:

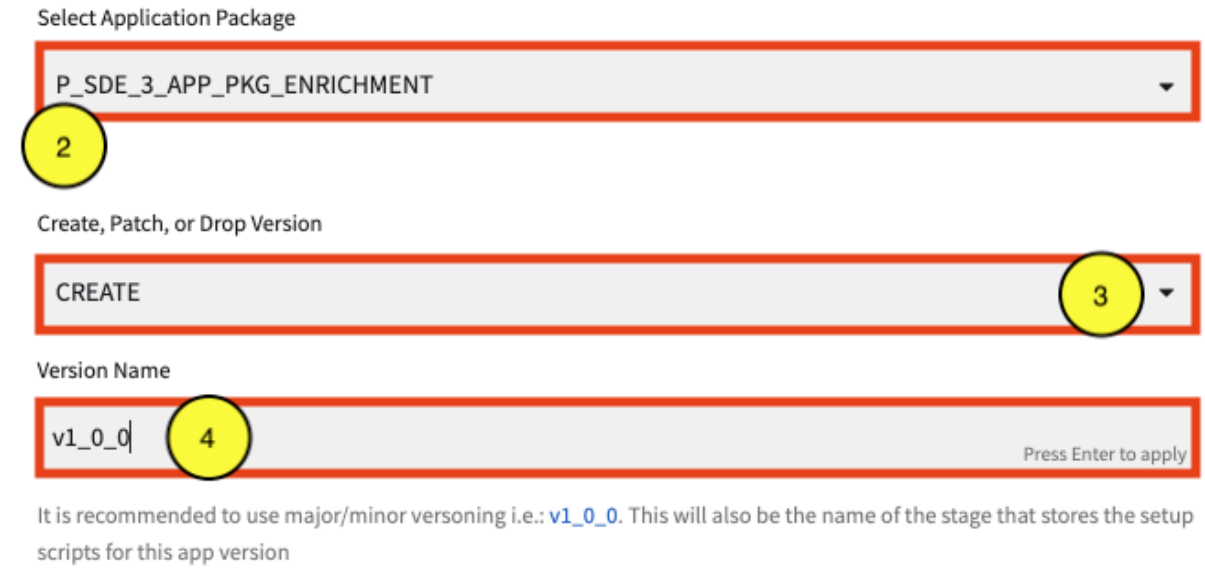
Step 1: In the App Control Manager, click **Manage App** >> **App Package** >> **Versions**.



Step 2: Select the **Application Package** from the Select Application Package drop-down.

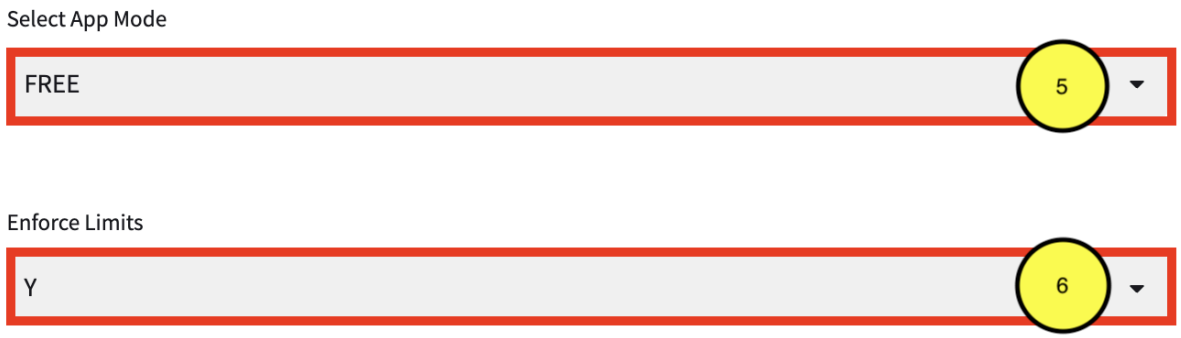
Step 3: Select **CREATE** or **PATCH** from the Create, Patch, or Drop Version drop-down, depending on desired action.

Step 4: If **CREATE**, specify a Version Name. If **PATCH**, select the existing version.

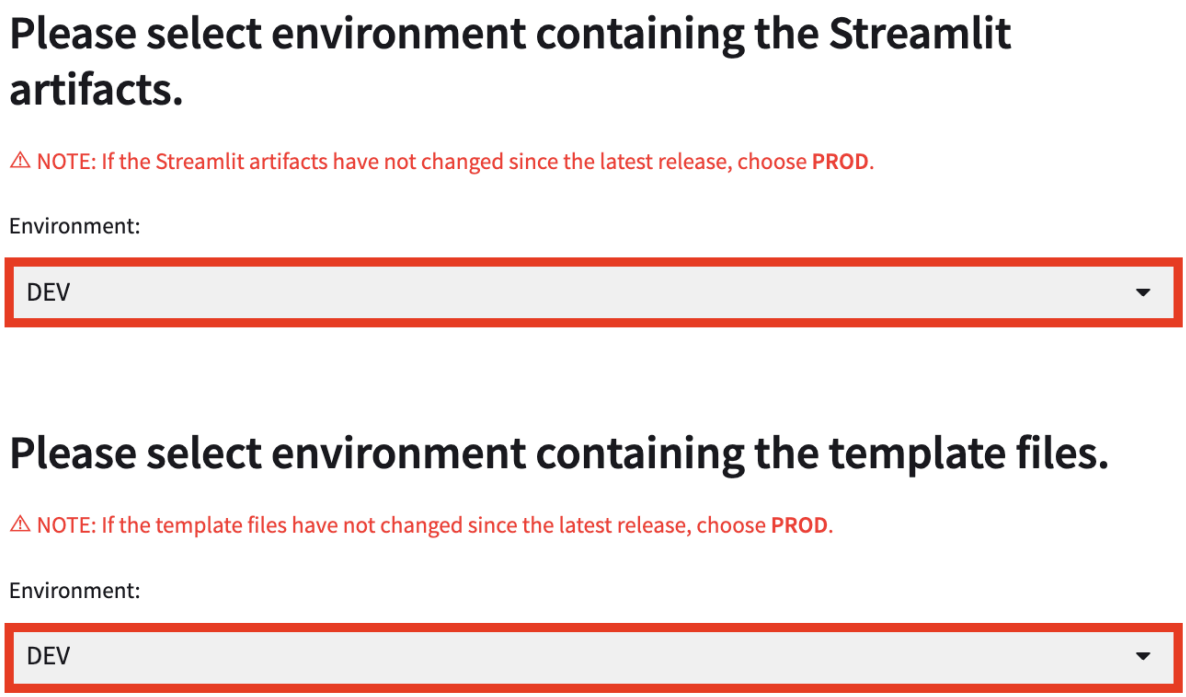


Step 5: Select **FREE**, **PAID**, or **ENTERPRISE**, from the Select App Mode drop-down, depending on desired app version.

Step 6: Select **Y** or **N** from the Enforce Limits drop-down. By default, limits should be enforced, but this can be set to N if limit enforcement should be turned off (i.e. during testing).



Step 6: Select the environment to pull the **Streamlit** and **template** files from.



Step 7: Select the **Database** and **Schema**, then the source **Functions** required by the native app from the drop-down menus.

If any of the functions have been updated or yet to have been promoted to prod, the functions should be located in the ***P_<APP_CODE>_SOURCE_DB_DEV.FUNCS_APP*** schema. If there aren't any changes to the functions since being promoted to prod, reference the prod versions, in the ***P_<APP_CODE>_SOURCE_DB_PROD.FUNCS_APP*** schema.

NOTE: If these functions are to be made available to the consumer, select the **Accessible** checkbox next to the function under Selected Functions:.

snowflake

Please select source function(s).

⚠ NOTE 1: The source functions should be created by/granted to the app admin role, prior to creating the application package.

⚠ NOTE 2: If **NONE** of the source functions have changed since the latest release, choose **P_SDE_1_6_SOURCE_DB_PROD**, otherwise choose **P_SDE_1_6_SOURCE_DB_DEV**

Select Database:

P_SDE_1_6_SOURCE_DB_DEV

Select Schema:

FUNCS_APP

Select Function(s):

P_SDE_1_6_SO... ×

Diagnostic Mode:

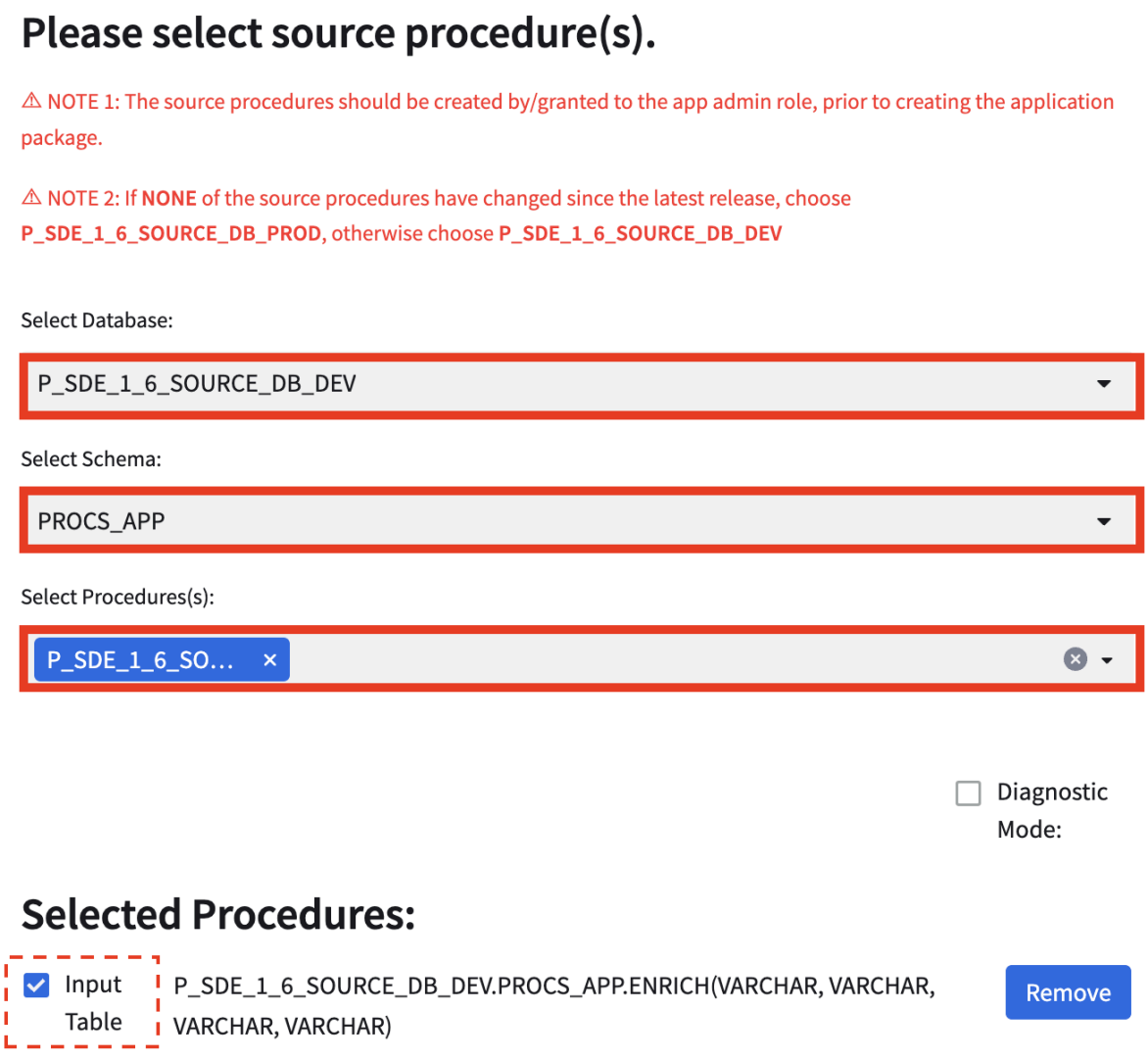
Selected Functions:

Accessible P_SDE_1_6_SOURCE_DB_DEV.FUNCS_APP.JS_FACTORIAL(FLOAT) Remove

Step 8: Select the **Database** and **Schema**, then the source **Procedures** required by the native app from the drop-down menus.

If any of the procedures have been updated or yet to have been promoted to prod, the procedures should be located in the ***P_<APP_CODE>_SOURCE_DB_DEV.PROCS_APP*** schema. If there aren't any changes to the procedures since being promoted to prod, reference the prod versions, in the ***P_<APP_CODE>_SOURCE_DB_PROD.PROCS_APP*** schema.

NOTE: If these procedures require an input table as a parameter, select the **Input Table** checkbox next to the function under Selected Procedures:.

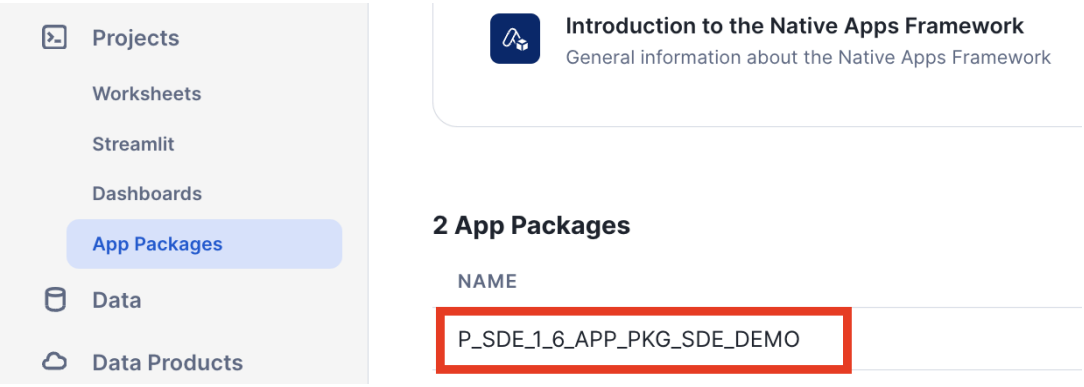


Step 9: Click the **CREATE** or **PATCH** button, depending on desired action to take.

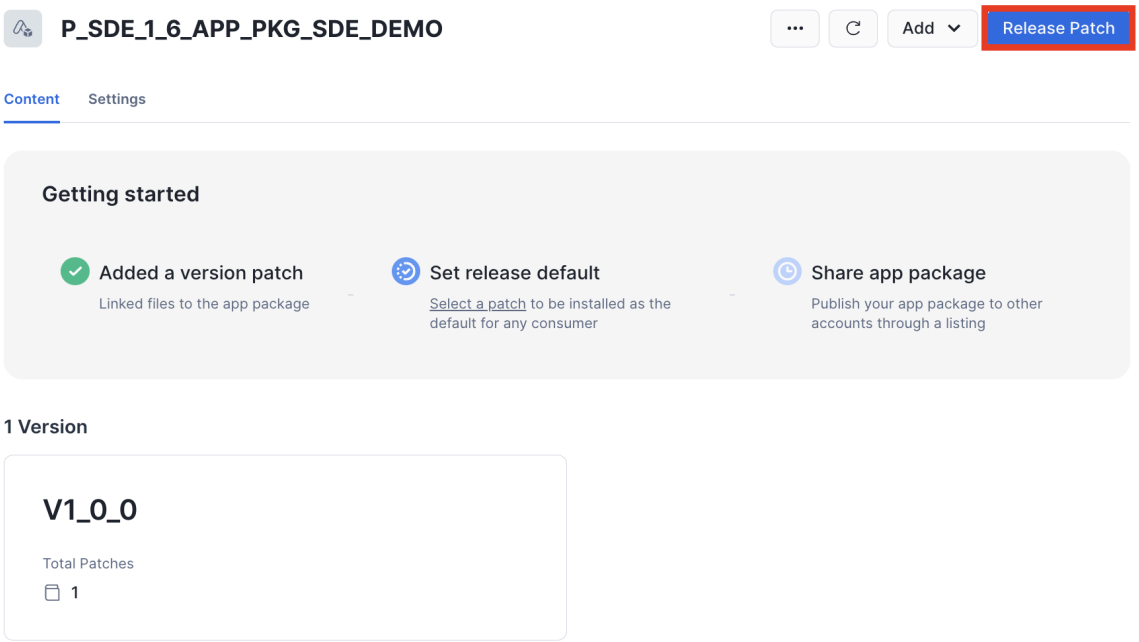
Part 3: Release Patch

In order to grant consumers access to the latest version/patch, the provider must create a release for the application package. The following steps detail how to set/unset a Release Directive:

Step 1: In Snowsight, click **Apps** >> **Packages**. Select the application package created in Part 2.



Step 2: Click **Release Patch**.

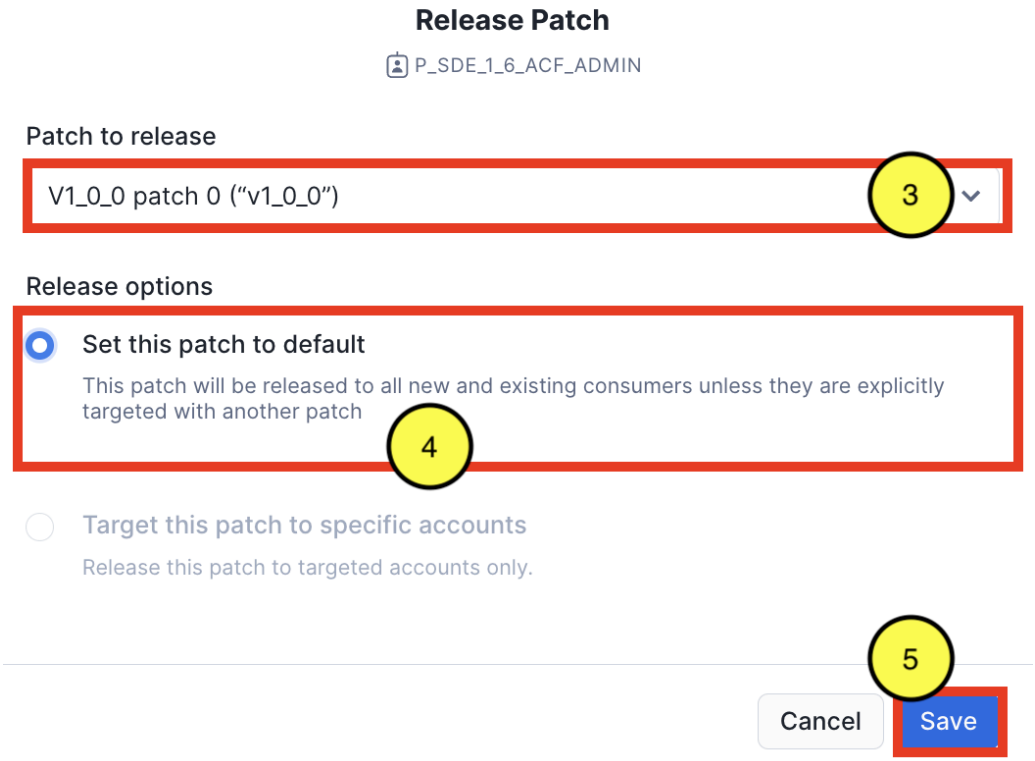


Step 3: Select the appropriate patch from the Patch to release drop-down.

Step 4: Choose either **Set this patch to default** (which allows all new and existing customers to get the update) or **Target this patch to specific consumers**.

- If **Target this patch to specific consumers** is selected:
  - Either create or select a targeted release group
  - Add account(s) to the group using the `<organization_name>.<account_name>` format.

Step 5: Click **Save**.



Step 6: Create Test Application Listing

Once the application package has a released version and patch, the native app is ready to be privately listed. For instructions on how to create a Private Listing for the native app and add a test consumer account, visit the **Create a Listing for Your Application** section: https://docs.snowflake.com/en/developer-guide/native-apps/tutorials/getting-started-tutorial#publish-and-install-your-application.

Step 7: Onboard Consumer (ENTERPRISE only)

Prior to installation, a consumer of the **ENTERPRISE** version of the app must first be onboarded to use the native app. This can be done via the App Control Manager.

When a consumer is onboarded, metadata is generated for the consumer, which regulates how they can use the app and tracks key metrics, such as number of installs, requests, etc. Default values are applied for each consumer, as defined in the METADATA_DICTIONARY table. However, certain default values can be overridden, if desired.

The following steps detail how to onboard a consumer:

Step 1: In the App Control Manager, click **Manage Consumers** >> **Onboard**.

Step 2: Click **+ Consumer**

Step 3: Enter the **Consumer Account** (Snowflake Account Locator), **Consumer Name**, and if desired, **Select Controls to Override Defaults**. If overriding defaults, enter the New Value for each control.

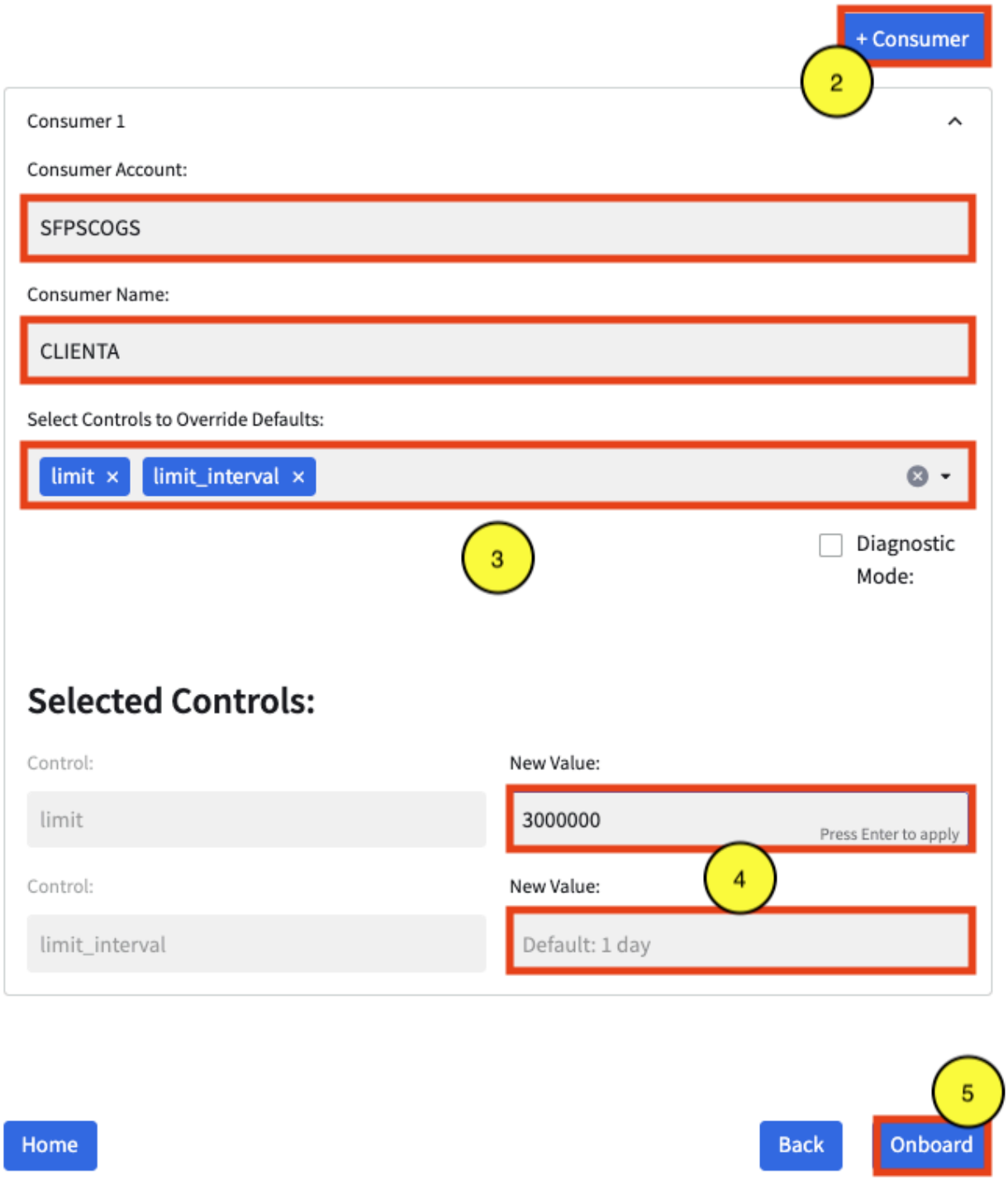
Step 4: If adding multiple consumers, click **+ Consumer** to add as many consumers as desired, repeating Step 3 for each. To remove a consumer from being onboarded, click **- Consumer**.

Step 5: Click **Onboard**.

NOTES:

- Consumers of the FREE and PAID version of the app are automatically onboarded, when the consumer shares events with the provider.
- Once the consumer is onboarded and events are received, they're automatically able to use the app.

Onboard a new Consumer, specifying the Consumer's Snowflake Account, Name, and any control default values to override.



Step 8: Manage Consumer Controls (optional)

If desired, the App Control Manager can update values stored in a consumer's metadata:

Step 1: In the App Control Manager, click **Manage Consumers** >> **Manage**.

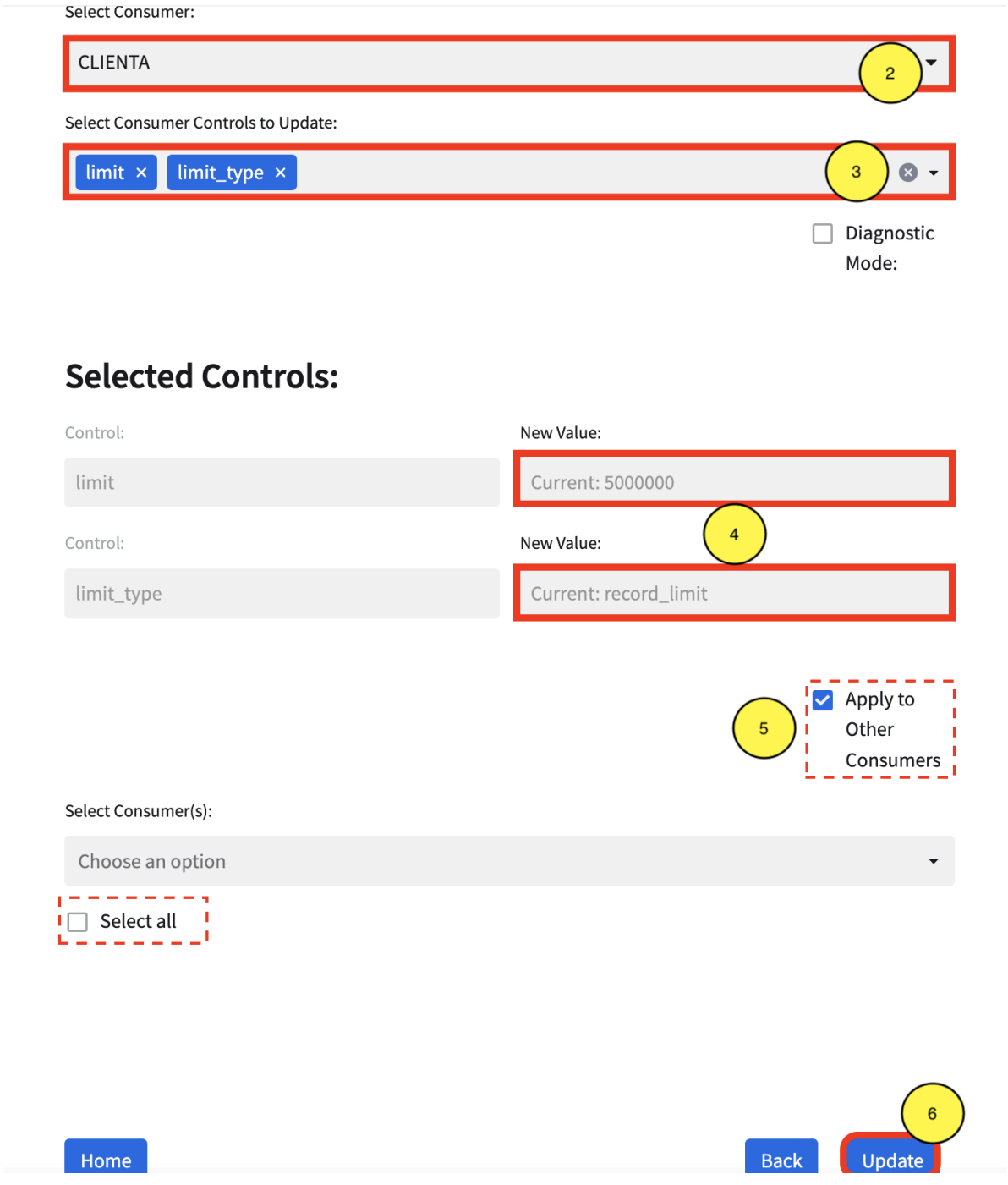
Step 2: Select **Consumer** from the Select Consumer drop-down.

Step 3: Select the **control(s)** to update from the Select Consumer Controls to Update drop-down

Step 4: In the **New Value** field, enter the updated value for each selected control.

Step 5: If the updated values should be applied to other or all consumers, click **Apply to Other Consumers**, then select either the applicable consumers or click **Select all**.

Step 6: Click **Update**.


Select Consumer:
CLIENTA
2
Select Consumer Controls to Update:
limit
limit_type
3
Diagnostic Mode:
Selected Controls:
Control:
limit
New Value:
Current: 5000000
Control:
limit_type
New Value:
Current: record_limit
4
5
Apply to Other Consumers
Select Consumer(s):
Choose an option
Select all
6
Home
Back
Update

Step 9: QA/Test the Native App

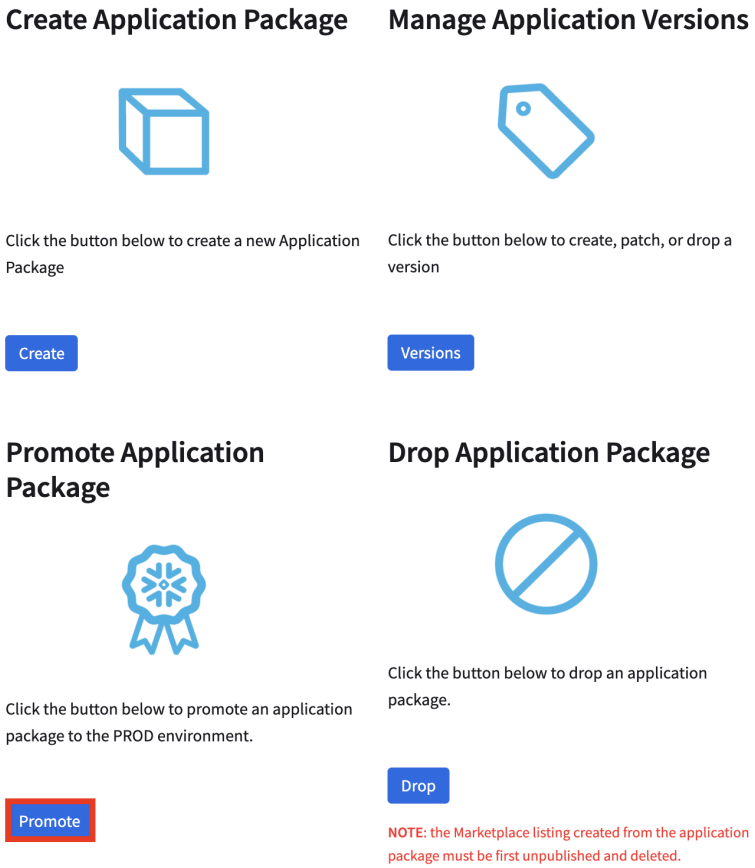
Install the native app available from the private listing. For instructions on how to install a native app, visit the **Install the Application** section https://docs.snowflake.com/en/developer-guide/native-apps/tutorials/getting-started-tutorial#id6.

Once installed, test the application as required.

Step 10: Promote Application Package to Prod

Once the application has passed testing, it is ready to be promoted to the prod environment. This process creates the prod environment, **P_<APP_CODE>_SOURCE_DB_PROD**, cloning the **P_<APP_CODE>_SOURCE_DB_DEV** database and all of its objects. The **P_<APP_CODE>_SOURCE_DB_PROD** database will serve as the source for any production-ready application packages.

Step 1: In the App Control Manager, click **Manage App** >> **App Package** >> **Promote**.

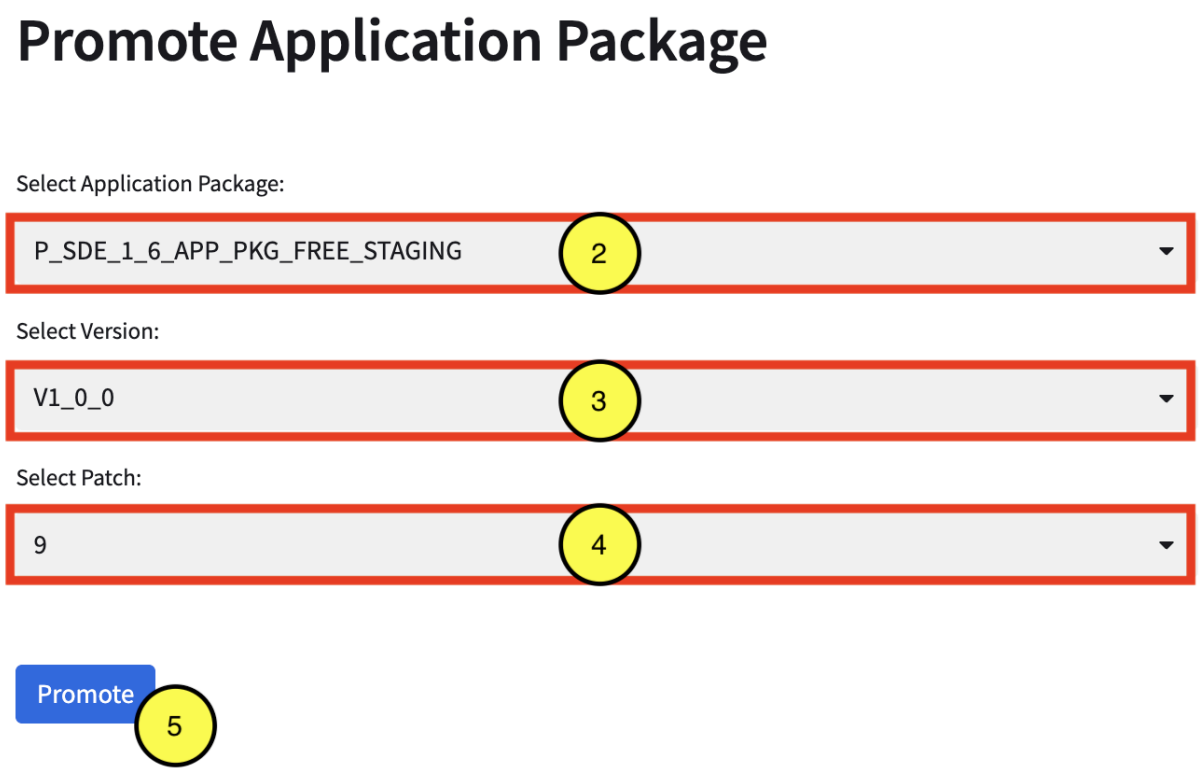


Step 2: Select the applicable **application package** from the Select Application Package drop-down.

Step 3: Select the applicable **version** from the Select Version drop-down.

Step 4: Select the applicable **patch** from the Select Patch drop-down.

Step 5: Click **Promote**.



Step 11: Create Listing for Production-ready App

Repeat Step 5 to create an application package for the production-ready code/objects and Step 7 to onboard consumers (if privately listed).

Removals

The following sections detail how to remove the various components of the native app, from removing a consumer's access to the application, to removing the entire ACF from the provider's account.

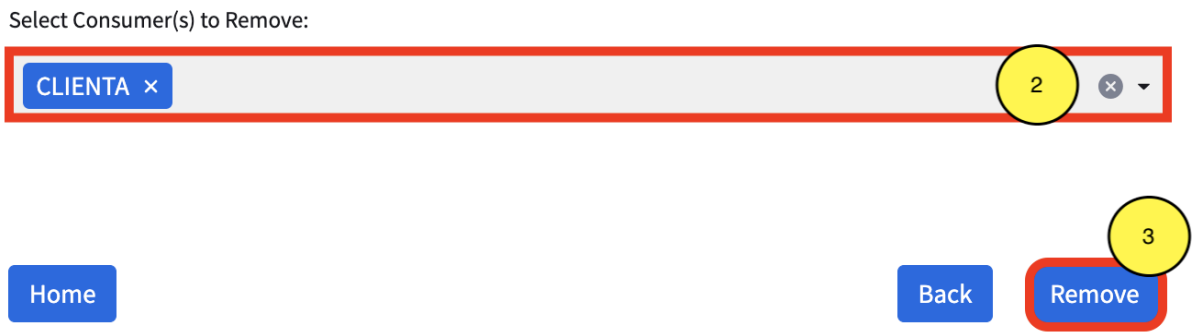
Remove Consumer

In the event the provider should remove the consumer, the following steps detail how to remove a consumer:

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

Step 2: Select **Consumer(s)** from the drop-down.

Step 3: Click **Remove**.



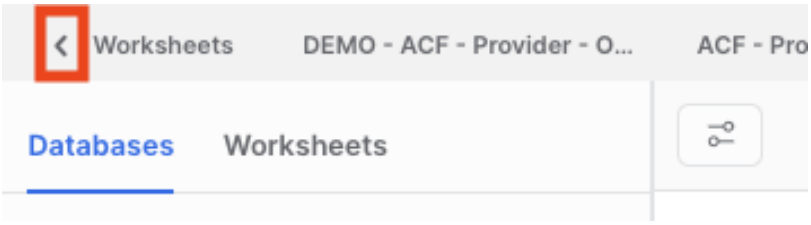
Remove Listing

The following details how to remove the private listing, in the event the provider wants to remove the private listing, in order to remove the native app.

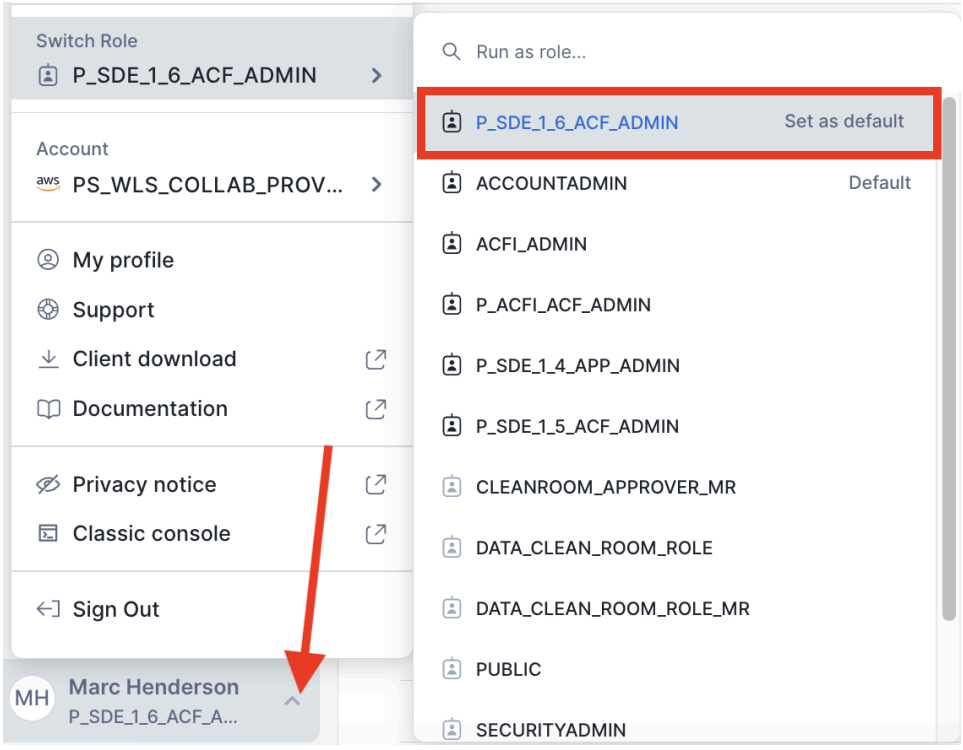
NOTE: This does not remove the application from the provider's account. Consumers will no longer have access to the application.

Step 1: Log into **Snowsight**.

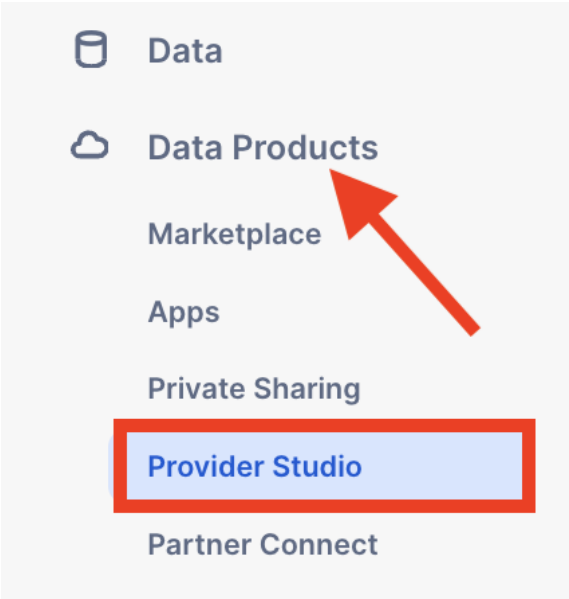
Step 2: Once logged in, if not at the Snowsight home screen, click the **Back** button, in the top left area of the UI, to open the left navigation menu.



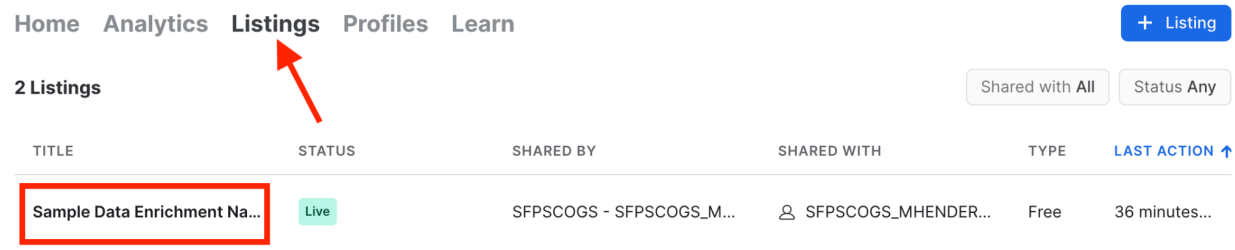
Step 3: Switch to the **P_<APP_CODE>_ACF_ADMIN** role, by clicking the drop-down in the bottom left area of the UI, then hovering over the Switch Role menu item.



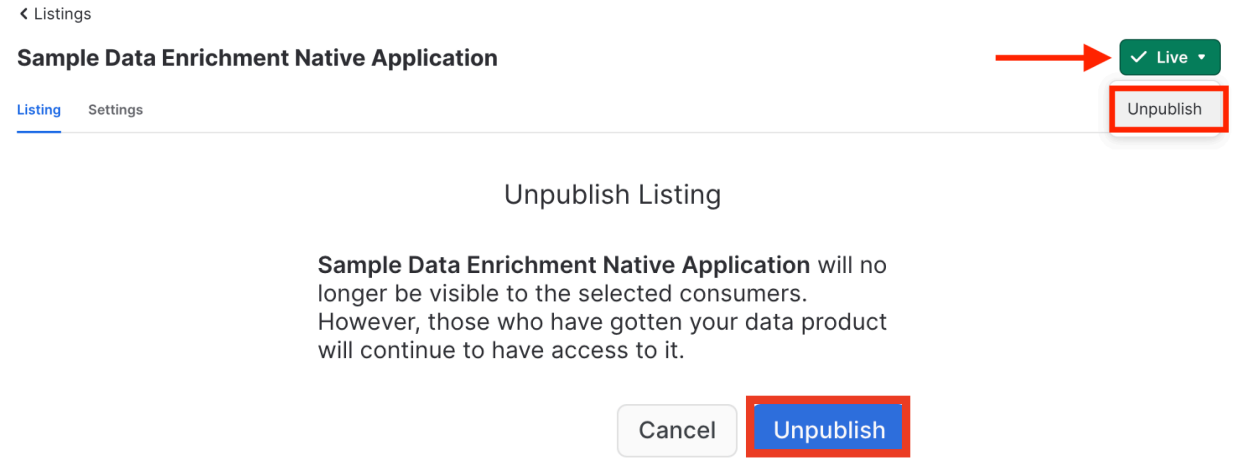
Step 4: Click **Data Products**, then **Provider Studio**.



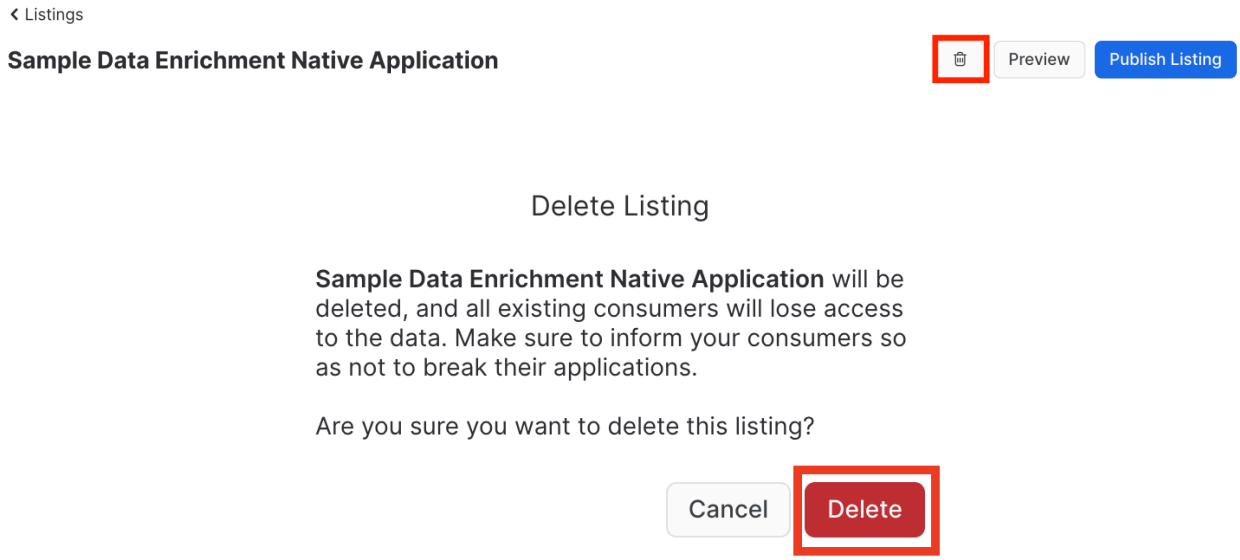
Step 5: Click the **Listings** tab and select the listing for this application.



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



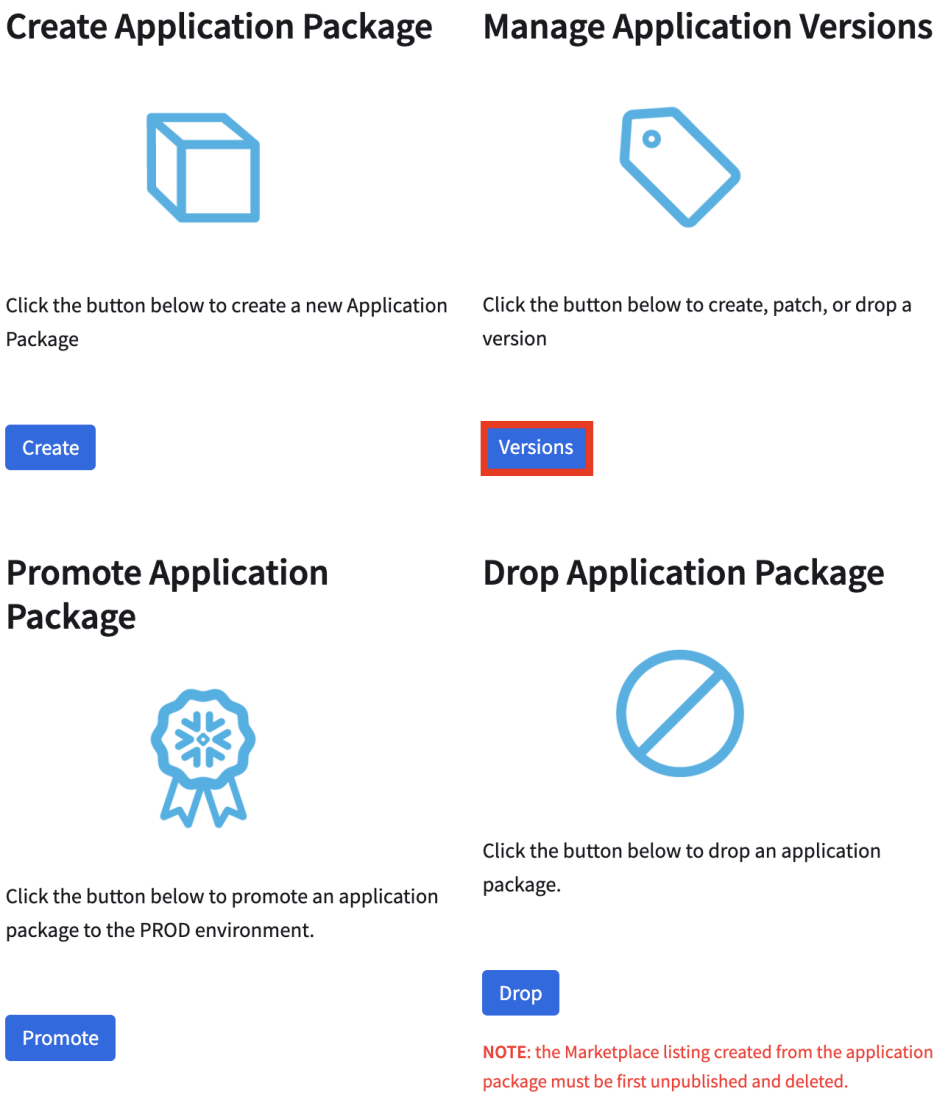
Step 7: Delete the listing, by clicking the **trashcan** icon. When the dialog box appears, click **Delete**.



Drop Version

The following steps detail how to remove a version for a particular application package:

Step 1: In the App Control Manager, click **Manage App** >> **App Package** >> **Versions**.

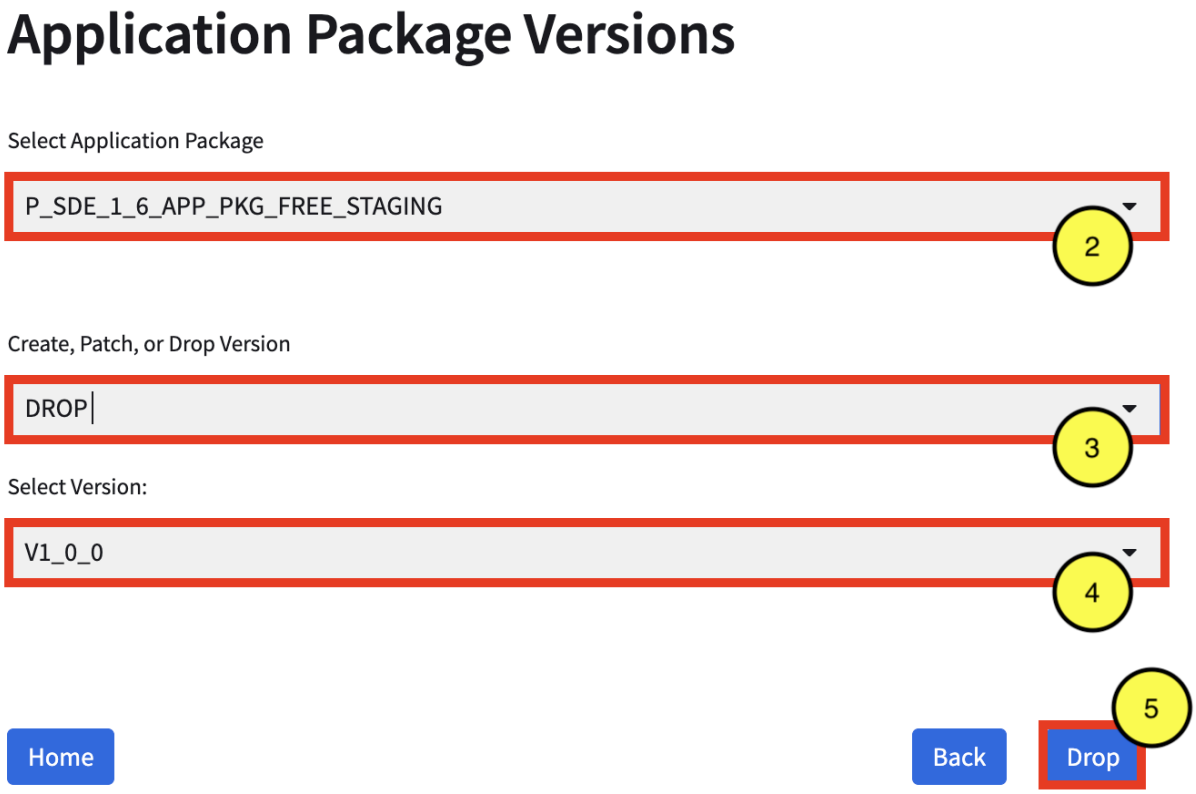


Step 2: Select the **application package** from the Select Application Package drop-down.

Step 3: Select **DROP** from the Create, Patch, or Drop Version drop-down, depending on desired action.

Step 4: Select the **version** from the Select Version drop-down, depending on desired action.

Step 5: Click **DROP**.

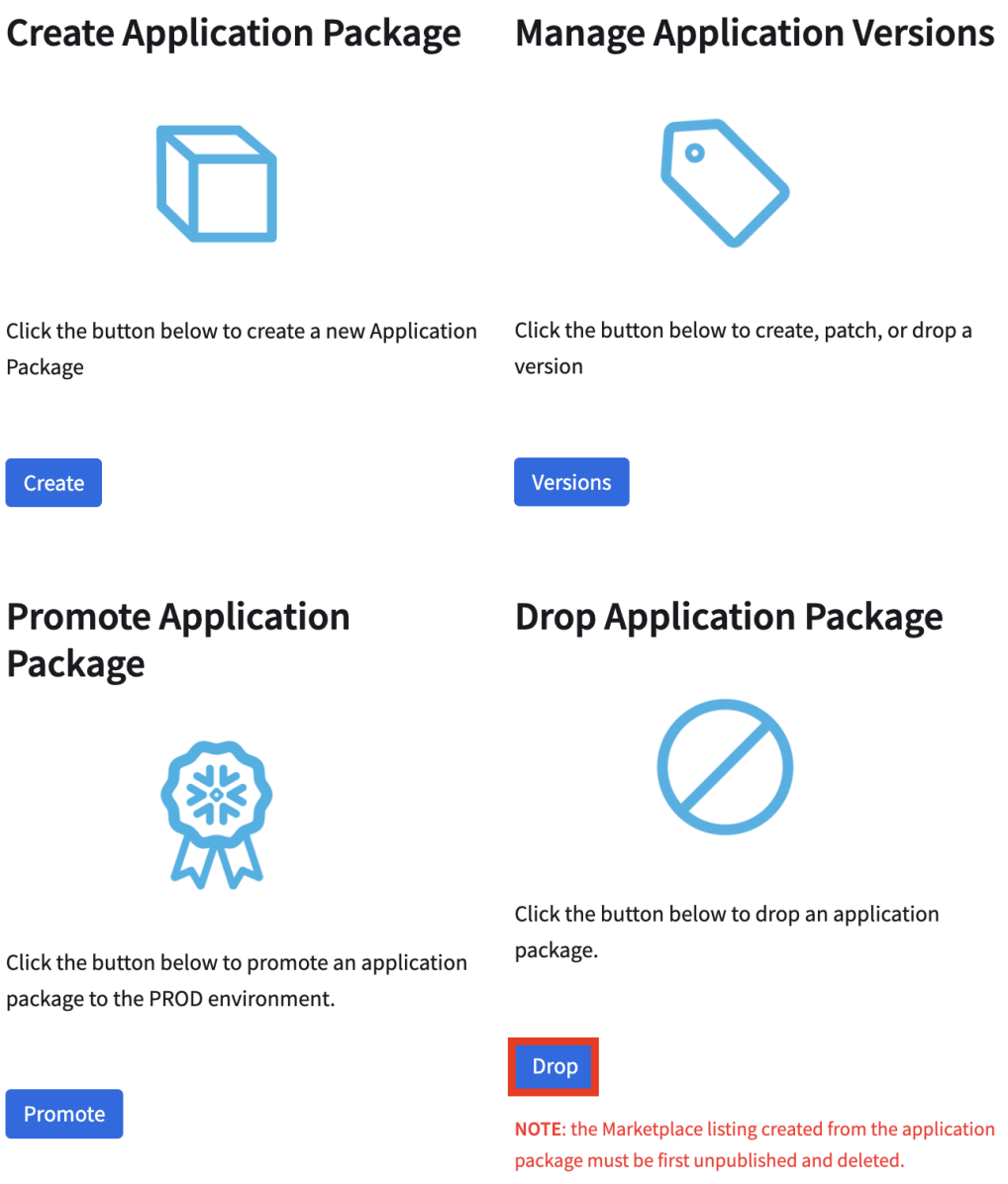


Remove Application Package

The following steps detail how to remove an application package.

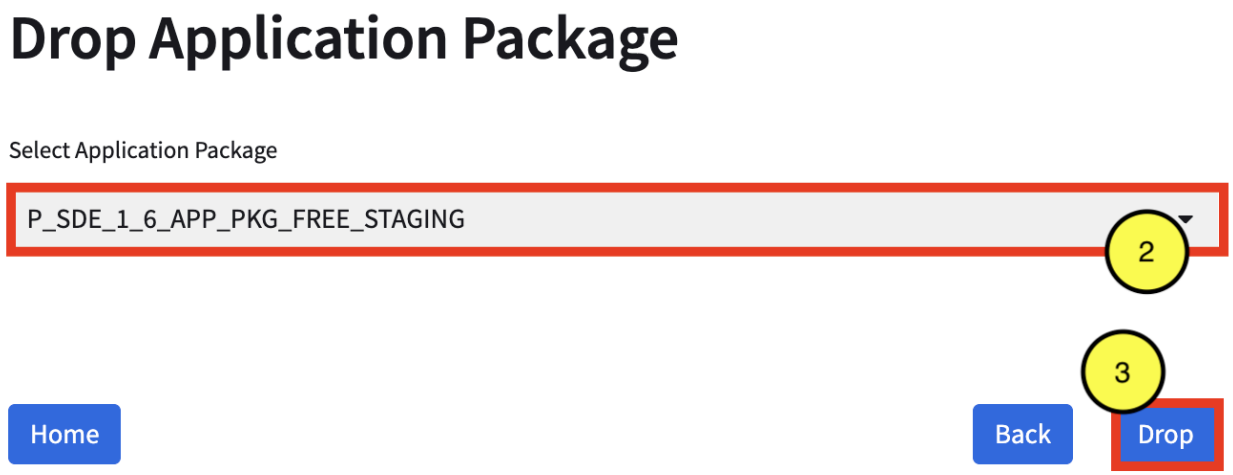
NOTE: Any listing using the application package should be unpublished and deleted prior to removing the application package.

Step 1: In the App Control Manager, click **Manage App** >> **App Package** >> **Drop**.



Step 2: Select the **application package** from the Select Application Package drop-down.

Step 3: Click **DROP**.



Remove 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 ACF**.

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

Step 3: Click **Remove**.