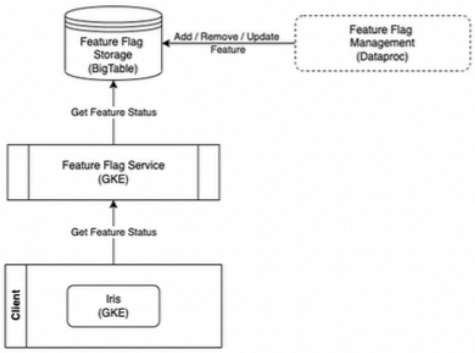
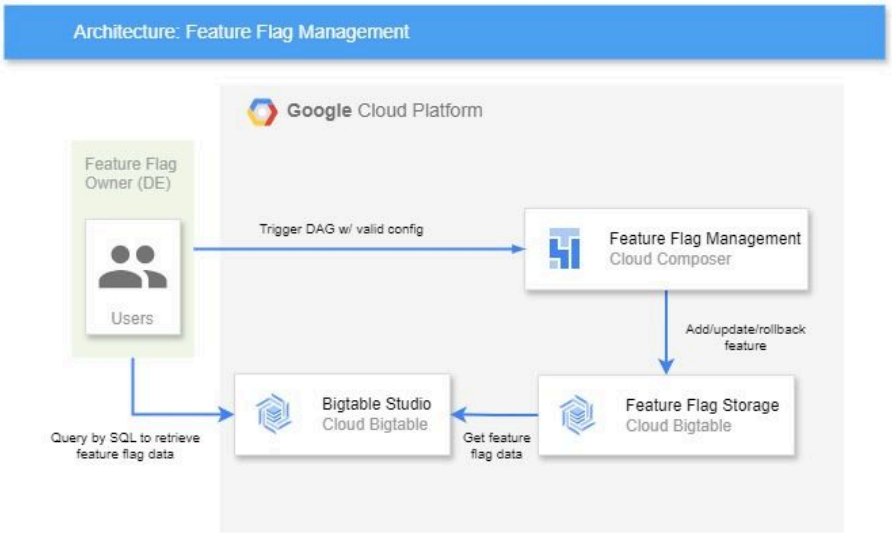
Feature Flag Framework Runbook

Architecture



Feature Flag Framework Architecture



Feature Flag Management Architecture

Reference:

- Feature flag design: doc link.
- Feature flag Bigtable sample data and access pattern: doc link.

📈 Process - Access Pattern

Feature Flag Management

	Procedure	Step instructions	Execution location	Run environments	Est. duration
1	Add a new feature flag	• Use DAG to run cmd. Click “Trigger DAG w/ config“. Sample input JSON configuration is, `{ "row_key_prefix_PROJECT": "csr-assistant", "row_key_prefix_ROWTYPE": "FLAG", "row_key_prefix_FLAGTYPE": "feature", "row_key_prefix_FLAGNAME": "multi_step_api_router", "row_cell_value": "{'Name': 'multi_step_api_router','Description': 'Enable multi step API router','Type': 'FEATURE'}", "add_new_feature_flag": true }` • Sample output is as below,  • Input: ◦ `row_key_prefix_ROWTYPE`: ▪ use “`FLAG`" to add a new feature flag. ▪ use "`REVISION`" to add a new version to an existing feature. ◦ `row_key_prefix_FLAGTYPE`: ▪ use “`data`“ if the flag holds a string configuration value. Ex: gpt-4o. ▪ use “`feature`" if flag value is Boolean, signaling feature is either enabled (`true`) or disabled (`false`). ◦ `row_key_prefix_FLAGNAME`: use underscore as delimiter for feature flag name. ◦ `row_cell_value`: should be dictionary format. ◦ `add_new_feature_flag`: ▪ use `true` to add a new feature flag. ▪ use `false` to add a new version to an existing feature.	Airflow DAG: • Prod DAG link	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • DAG ID: `csr-va-hcb-iris-feature-flag-management-bigtable-add-row` • DAG Link: TBD	00:00:05
2	Add a new version for an	• Use DAG to run cmd. Click “Trigger DAG w/ config“. Sample input JSON configuration is, `{`	Airflow DAG:	• Project ID: `anbc-hcb-dev`, `anbc-hcb-`	00:00:05

	existing feature flag	<pre>"row_key_prefix_PROJECT": "csr-assistant", "row_key_prefix_ROWTYPE": "REVISION", "row_key_prefix_FLAGTYPE": "feature", "row_key_prefix_FLAGNAME": "multi_step_api_router", "row_cell_value": "{'Name': 'multi_step_api_router','Type': 'FEATURE','FormatVersion': 1,'ControlValue': False,'Comment': 'Rollout to 40%','OwnerID': 'John Smith','Treatments': [{'OverrideList': ['A12345'], 'PercentRollout':40, 'Value': True}]}", "add_new_feature_flag": false }</pre> • Sample output is as below,  • Input: ◦ `row_key_prefix_ROWTYPE`: ▪ use "`FLAG`" to add a new feature flag. ▪ use "`REVISION`" to add a new version to an existing feature. ◦ `row_key_prefix_FLAGTYPE`: ▪ use "`data`" if the flag holds a string configuration value. Ex: gpt-4o. ▪ use "`feature`" if flag value is Boolean, signaling feature is either enabled (`true`) or disabled (`false`). ◦ `row_key_prefix_FLAGNAME`: use underscore as delimiter for feature flag name. ◦ `row_cell_value`: should be dictionary format. ◦ `add_new_feature_flag`: ▪ use `true` to add a new feature flag. ▪ use `false` to add a new version to an existing feature.	• Prod DAG link	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • DAG ID: `csr-va-hcb-iris-feature-flag-management-bigtable-add-row` • DAG Link: TBD	
3	Rollback to a historical version (like the second last one) of a given feature flag (different from above)	1. Go to Bigtable Studio, search for the target feature flag and pull all its version history. Refer to "*Get the version history of a given feature flag*" for sample query. 2. Copy the exact entire row value (not row key value!) of the version to rollback to. 3. Follow the same procedure to add a new version to the target feature flag, but with this copied row value. Refer to "*Add a new version for an existing feature flag*" for how-to. 4. **Note**: As soon as DAG being triggered, FF data in Bigtable becomes available. It is up to the config in FF Service and FF Client for the total duration for a rollback version becoming available for user to use it in the application.	• Bigtable Studio • Airflow DAG: ◦ Prod DAG link	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • DAG ID: `csr-va-hcb-iris-feature-flag-management-bigtable-add-row` • DAG Link: TBD • Bigtable instance: `csr-va-db-hcb-dev`, `csr-va-db-hcb-test`, `csr-va-db-hcb-prod`. • Feature Flag Bigtable name: `csr_va_feature_flag_bigtable`. • Bigtable Studio link: https://console.cloud.google.com/bigtable/instances/csr-va-db-hcb-dev/studio/query?project=anbc-hcb-dev	00:03:00 (check **Note** for more explanation)
4	Update an existing feature flag	The same procedure to add a new feature flag. In backend, FF service will always pick the latest metadata (by using `metadata['CreatedTime']` or order by `_key` value DESC)	Airflow DAG: • Prod DAG	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`.	00:00:03

			link	• DAG ID: `csr-va-hcb-iris-feature-flag-management-bigtable-add-row` • DAG Link: TBD	
5	Get the metadata of a given feature flag	```sql SELECT * FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) WHERE _key LIKE 'PROJECT#csr-assistant#ROWTYPE#FLAG#FLAGTYPE#feature#FLAGNAME#multi_step_api_router%' ; # To retrieve the latest metadata # method 1 SELECT * FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) WHERE _key LIKE 'PROJECT#csr-assistant#ROWTYPE#FLAG#FLAGTYPE#feature#FLAGNAME#multi_step_api_router%' ORDER BY _key DESC LIMIT 1; # method 2 SELECT * FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) WHERE _key LIKE 'PROJECT#csr-assistant#ROWTYPE#FLAG#FLAGTYPE#feature#FLAGNAME#multi_step_api_router%' ORDER BY metadata['CreatedTime'] DESC LIMIT 1; ```	Bigtable Studio	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • Bigtable instance: `csr-va-db-hcb-dev`, `csr-va-db-hcb-test`, `csr-va-db-hcb-prod`. • Feature Flag Bigtable name: `csr_va_feature_flag_bigtable`. • Bigtable Studio link: https://console.cloud.google.com/bigtable/instances/csr-va-db-hcb-dev/studio/query?project=anbc-hcb-dev	00:00:03
6	Get the most recent version of a given feature flag	• Monitor Dashboard: https://csr-va-iris-feature-flag-monitoring.hcb-dev.aig.aetna.com/ • Or, Query on Bigtable Studio ```sql SELECT * FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) WHERE _key LIKE ' ' ORDER BY metadata['CreatedTime'] DESC LIMIT 1; # Or SELECT * ```	• Monitor dashboard • Bigtable Studio	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • Bigtable instance: `csr-va-db-hcb-dev`, `csr-va-db-hcb-test`, `csr-va-db-hcb-prod`. • Feature Flag Bigtable name: `csr_va_feature_flag_bigtable`. • Bigtable Studio link: https://console.clou	00:00:10

		12 FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) 13 WHERE _key LIKE 'PROJECT#csr-assistant#ROWTYPE#REVISION#FLAGTYPE#feature#FLAGNAME#multi_step_api_router%' 14 ORDER BY _key DESC 15 LIMIT 1;		d.google.com/bigtable/instances/csr-va-db-hcb-dev/studio/query?project=anbc-hcb-dev	
7	Get all feature flags within a given project (not with their most recent version)	1 SELECT 2 * 3 -- metadata['Name'] AS feature_name 4 FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) 5 WHERE _key LIKE 'PROJECT#csr-assistant#ROWTYPE#FLAG%' 6 ;	Bigtable Studio	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • Bigtable instance: `csr-va-db-hcb-dev`, `csr-va-db-hcb-test`, `csr-va-db-hcb-prod`. • Feature Flag Bigtable name: `csr_va_feature_flag_bigtable`. • Bigtable Studio link: https://console.cloud.google.com/bigtable/instances/csr-va-db-hcb-dev/studio/query?project=anbc-hcb-dev	00:00:10
8	Get all feature flags within a given project (with their most recent version)	• TBD: Feature Flag dashboard.			
9	Get the version history of a given feature flag	1 SELECT 2 * 3 FROM `csr_va_feature_flag_bigtable`(WITH_HISTORY=>FALSE) 4 WHERE _key LIKE 'PROJECT#csr-assistant#ROWTYPE#REVISION#FLAGTYPE#feature#FLAGNAME#multi_step_api_router%' 5 ORDER BY metadata['CreatedTime'] DESC 6 -- Or: 7 -- ORDER BY _key DESC 8 ;	Bigtable Studio	• Project ID: `anbc-hcb-dev`, `anbc-hcb-test`, `anbc-hcb-prod`. • Bigtable instance: `csr-va-db-hcb-dev`, `csr-va-db-hcb-test`, `csr-va-db-hcb-prod`. • Feature Flag Bigtable name: `csr_va_feature_flag_bigtable`. • Bigtable Studio link: https://console.clou	00:00:10

				d.google.com/bigtable/instances/csr-va-db-hcb-dev/studio/query?project=anbc-hcb-dev	
10	~~Remove an existing feature flag~~	• Not supported			
11	~~Remove a version of a feature flag~~	• Not supported			

Feature Flag Service

Endpoints

API Endpoint: `https://csr-va-iris-feature-flag-service.hcb-prod.aig.aetna.com/get-feature-flag`

Request Parameters: `project_name`, `instance_id`, `flag_type`, `flag_name`

Response Format: `request_id`, `instance_id`, `flag_type`, `flag_name`, `flag_value`, `rollout_group`

Request Example: `https://csr-va-iris-feature-flag-service.hcb-prod.aig.aetna.com/get-feature-flag/?project_name=csr-assistant&flag_name=llm_version&flag_type=data&instance_id=A123456`

Environment Variables / Configuration Parameters

Name	Description	Default
`environment`	Name of the environment	hcb-dev
`TENANT_NAME`	Name of the tenant	csr-va
`INSTANCE_ID`	Bigtable instance ID	csr-va-db-hcb-dev, csr-va-db-hcb-test, csr-va-db-hcb-prod
`PROJECT_ID`	Project ID	anbc-hcb-dev, anbc-hcb-test, anbc-hcb-prod
`BT_TABLE_ID`	Feature Flag Bigtable	csr_va_feature_flag_bigtable
`FF_LOG_TABLE_ID`	Feature Flag Operation Log Bigtable	csr_va_feature_flag_service_operation_log
`PARTIAL_ROLLOUT_HASH_VAL_CACHE_TTL_DAYS`	TTL in days for caching instance ids hash values used during partial rollout	2
`FEATURE_FLAG_CACHE_TTL_MINUTES`	TTL in minutes for caching flag values	5

Feature Flag Retrieval Process

1. Async Retrieval: The function `get_feature_flag` uses the asynchronous `get_latest_flag_data` method to retrieve feature flag data from Bigtable.
2. Data Transformation: The retrieved data is transformed into `FeatureFlagResponse` Pydantic model, which includes: `request_id`, `instance_id`, `flag_name`, `flag_type`, `flag_value` and `rollout_group`.

Caching

Caching is performed in two levels:

1. Cache the hash values of the `instance_id` that is used during partial rollout.
2. Cache the feature flag values that is retrieved from Bigtable. Force eviction after TTL.

Partial Rollout

The partial rollout module uses the hash values of `instance_id` and use the position of the hash values on the hash space to determine if the treatment should be rolled out to the user or not.

The module currently supports three hashing functions:

1. `md5` (128 bit) (default)
2. `sha1` (160 bit)
3. `sha256` (256 bit)

The module is implemented in a way such that it would support any hashing function that is implemented using the `hashlib` interface.

Implementation Details:

1. The modules receives the `instance_id` and generates a hash value in hex.
2. The hex value is then converted to decimal.
3. The decimal value is normalized to a [0, 1] range using the hash space.
4. This normalized value is compared to the input rollout percentage to determine if the treatment should be rolled out to the user.

The module would raise an invalid rollout percentage exception if the rollout_percentage is invalid or the invalid hashing algorithm exception if the hashing algorithm name is invalid.

Response Logging Process

Logging Functionality

1. Background Tasks: After retrieval, a background task is created using FastAPI's `BackgroundTasks`.
2. Synchronous Logging Endpoint: The background task triggers the synchronous logging endpoint .
3. Logged Data: The following information is stored in Bigtable:

- `flag_name`, `flag_type`, `instance_id`, `flag_value`, `rollout_group`
- `timestamp` (UTC time)
- `request_id` (for traceability)
- `cache_flag` (indicates whether data came from cache)

Bigtable Schema for Logs

- Row Key Format: `PROJECT#{project_name}#FLAGTYPE#{flag_type}#FLAGNAME#{flag_name}#INSTANCEID#{instance_id}#REQUESTID#{request_id}#TIME#{timestamp}`
- Column Family: metadata

- Columns:
  - `RequestID`
  - `Name`
  - `CreatedTimestamp`
  - `Type`
  - `InstanceID`
  - `AssignedValue`
  - `AssignedGroup`
  - `CacheFlag'`

Exception Handling

Exception Type	Status Code	Log Message	Who Should Handle	How to Handle
`InvalidFlagTypeException`	400	`Invalid flag type: {flag_type}.`	Client/User	Return a JSON response with details of the invalid flag. Update feature flag data with valid flag type in feature flag management.
`InvalidRolloutPercentageException`	400	`Rollout percentage: {rollout_percentage} is invalid. Value should be an integer in range [0, 100].`	Client/User	Return a JSON response with details of the invalid rollout percentage. Update feature flag data with valid rollout percentage in feature flag management.
`EmptyFlagDataException`	404	`No Flag Data found: {exc.detail}`	Client/User	Return a JSON response indicating that the flag data is empty or missing. Ensure feature flag data exists in the system.
`ValidationError`	500	`An error of type %s caught: %s.", type(exc).__name__, exc`	Development Team	Log the validation errors and return a 500 response with details of the error. Debug and fix the schema validation issue (Pydantic response model).
`TimeoutError` `asyncio.exceptions.TimeoutError` `concurrent.futures.TimeoutError`	504	`Timeout error occured for : %s.", exc`	Development Team	Investigate the issue. Retry or optimize the backend calls.

`HTTPException`	400-499	`A client-side error of type %s caught: %s.", type(exc).__name__, exc`	Client/User	Return appropriate HTTP status code and detailed error message indicating the client side issue.
`HTTPException` (Unhandled)	503	`An error of type %s caught: %s.", type(exc).__name__, exc`	Development Team	Log the error for debugging. Investigate the issue.
Generic `Exception` (Unexpected)	500	`An unexpected error of type %s caught: %s.", type(exc).__name__, exc`	Development Team	Log the error for debugging and return an internal server error response.

Auto-scaling Policy

Our current auto-scaling configurating for the feature flag service starts with two replicas. When the CPU utilization exceeds 75%, the system automatically scales up to handle the increased load, allowing for a maximum of 3 replicas to be deployed as needed. Resource limits are set at 1000m (1 vCPU) and 1GB of memory, capping the maximum resources each replica can use. Resource requests are set at 800m (0.8 vCPU) and 1GB memory, ensuring a baseline for steady performance.

Load Test

Feature Flag Service can handle up to 750 requests/second with P95 < 50 ms. This capacity should meet the goal of IRIS to serve 11 requests per second and reliably serve the volume of requests without being an overhead, even under load conditions.

More details are available here: Feature Flag Load Tests.

Feature Flag Client Library

iris-feature-flag-client is a python client library to read feature flag from Feature Flag service. The library exposes `FeatureFlagClient` and `AsyncFeatureFlagClient` classes to support synchronous and asynchronous execution. Both classes implements `get_flag_boolean(flag_name, instance_id, default_value)` and `get_flag_string(flag_name, instance_id, default_value)` functions to query from feature flag service.

Key Features

- **Synchronous Client** - Run tasks sequentially, where each tasks waits for the previous one to complete, ensuring predictable, blocking behavior.
- **Asynchronous Client** - Run tasks concurrently without waiting for others to finish, enabling non-blocking, high-performance execution using `async` and `await` keywords.
  Note: Synchronous code can be executed within an asynchronous Python environment; but not vice versa.
  More details about Python asyncio library.
- **TTL based In-memory cache** - Stores feature flag value for a specific duration, enabling faster retrieval while ensuring stale data is automatically evicted.

Requirements

- Python version `3.10.*`
- `pipenv`

Installation

Run the following command to install the python library

```
pipenv install csr-va-iris-feature-flag-client --index https://nexus.mgmt.aig.aetna.com/repository/pypi-hosted/simple
```

Sample Usage

Synchronous execution example

```python
##########################################
# File: params.py
##########################################
from iris_feature_flag_client import FeatureFlagClient
from enum import Enum

# Create an Enum and keep track of all the feature flags used in the application
# along with the JIRA link next to it.
class FeatureFlags(Enum):
  LLM_PROVIDER_DATA_FLAG: "llm_provider"  # JIRA link for flag removal
  MULTI_STEP_API_ROUTER_FEATURE_FLAG = "multi_step_api_router"   # JIRA link for flag removal

# Create client object once and use it across the application
# In this example, creating the client object in params.py where we configure
# application based on environments
ff_client = FeatureFlagClient(project_name="csr-assistant")

##############################################################################
# File: app.py
##############################################################################
# Import client object from params.py to call get flag functions
from params import ff_client, FeatureFlags

def simple_function():

  # TODO: JIRA link to follow-up feature flag removal
  # Reading a feature flag of boolean type
  flag_name = FeatureFlags.MULTI_STEP_API_ROUTER_FEATURE_FLAG.name
  instance_id = user_id_from_context   # user_id from request which is stored in context variable
  default_value = False
  # check if the multi_step_api_router enabled for user_id
  flag_value = ff_client.get_flag_boolean(flag_name, instance_id, default_value)
  if flag_value:
    # new behavior
    print(f"{flag_name} is enabled.")
  else:
    # old behavior
    print(f"{flag_name} is disabled.")

  # TODO: JIRA link to follow-up feature flag removal
  # Reading a feature flag of string type
  flag_name = FeatureFlags.LLM_PROVIDER_DATA_FLAG.name
  instance_id = request.user_id   # user_id from the request
  default_value = "openai"
  str_flag_value = ff_client.get_flag_string(flag_name, default_value, default_value)
  # custom logic to process string flag value.


```

```
simple_function()

```

Asynchronous execution example

```
################################################################################
# File: params.py
################################################################################
from iris_feature_flag_client import AsyncFeatureFlagClient
from enum import Enum

# Create an Enum and keep track of all the feature flags used in the application
# along with the JIRA link next to it.
class FeatureFlags(Enum):
  LLM_PROVIDER_DATA_FLAG: "llm_provider"  # JIRA link for flag removal
  MULTI_STEP_API_ROUTER_FEATURE_FLAG = "multi_step_api_router"   # JIRA link for flag removal

# Create client object once and use it across the application
# In this example, creating the client object in params.py where we configure
# application based on environments
ff_client_async = AsyncFeatureFlagClient("csr-assistant")

################################################################################
# File: app_async.py
################################################################################
# Import client object from params.py to call get flag functions
from params import ff_client_async, FeatureFlags

async def simple_function():

  # TODO: JIRA link to follow-up feature flag removal
  # Reading a feature flag of boolean type
  flag_name = FeatureFlags.MULTI_STEP_API_ROUTER_FEATURE_FLAG.name
  instance_id = user_id_from_context   # user_id from request which is stored in context variable
  default_value = False
  # check if the multi_step_api_router enabled for user_id
  boolean_flag_value = await ff_client_async.get_flag_boolean(flag_name, instance_id, default_value)
  if boolean_flag_value:
    print(f"{flag_name} is enabled.")
  else:
    print(f"{flag_name} is disabled.")

  # TODO: JIRA link to follow-up feature flag removal
  # Reading a feature flag of string type
  flag_name = FeatureFlags.LLM_PROVIDER_DATA_FLAG.name
  instance_id = request.user_id   # user_id from the request
  default_value = "openai"
  str_flag_value = await ff_client_async.get_flag_string(flag_name, default_value, default_value)
  # custom logic to process string flag value.


await simple_function()
```

Default Configurations

Config Name	Value
HTTP Client Timeout	5 seconds

In-memory Cache Size	100
In-memory Cache Timeout	300 seconds

Wheel file location

https://nexus.mgmt.aig.aetna.com/service/rest/repository/browse/pypi-hosted/csr-va-iris-feature-flag-client/

Support (TODO)

- **L1** support is the first line of assistance for common or basic issues (e.g., service restart), usually provided through phone, chat, or email; these may be dedicated support staff or junior engineers.
- **L2** support involves more qualified engineers capable of analyzing issues and providing solutions to problems that cannot be resolved by tier 1; these may be senior engineers.
- **L3** support is the final line of support and and should be capable of dealing with the most complicated or nuanced technical problems; these may be lead or principal engineers.

NOTE: Applications with a high *Criticality* or *Risk Tier* likely require 24/7 support; less critical applications (e.g., a data pipeline that runs once each week) may only need L1-L3 support during business hours.

Link to OpsGenie schedules: here