PUBLIC
2020-09-24

SAP Cloud Platform Feature Flags

THE BEST RUN SAP

Content

1 What Is Feature Flags

The Feature Flags service on the Cloud Foundry environment allows enabling or disabling new features without redeploying or restarting the application.

The Feature Flags service implements a common agile development concept that allows:

- Delivery of latent code
- Synchronized rollout of features that require changes in several different components or microservices
- Fast rollback of features

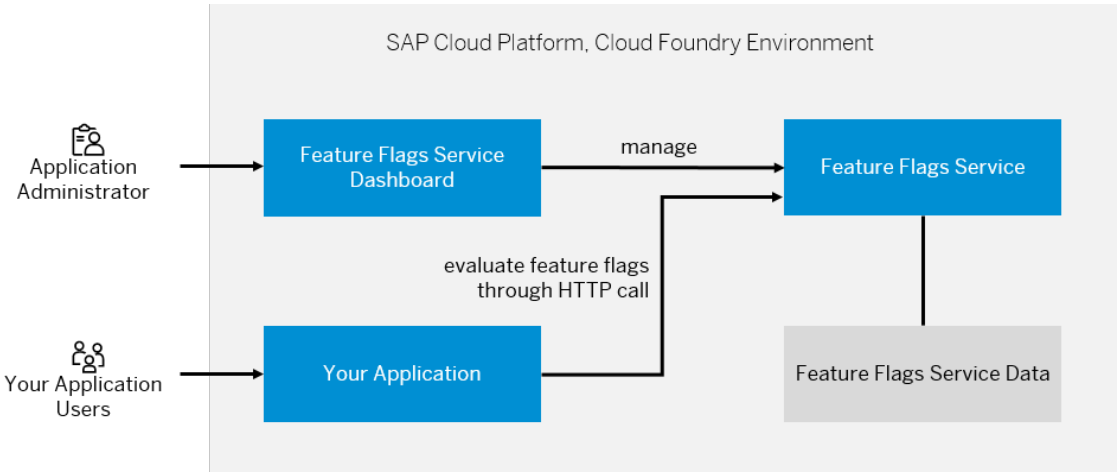
The concept of having a central synchronization point for releasing microservice-based features is common for cloud development best practices. The need comes from the fact that some business functionalities require development in several microservices that run in separate containers and need to be released in a synchronized way.

> **i Note**
>
> This service is for the Cloud Foundry environment only.

The Feature Flags service offers a dedicated dashboard for managing individual flags. Every Feature Flags service instance is displayed separately in the dashboard. Once you create a service instance in a space of your Cloud Foundry environment in the *Actions* section of the service instance, you will find the *Open Dashboard* button for accessing the Feature Flags dashboard for that service instance.

In order to start consuming the feature flags, that belong to a service instance, first, you have to bind your application to that service instance. The service instance binding is creating automatically new environment variables for your application that are holding credentials data and the URI for accessing the service instance.

To be able to switch on and off functional parts of your application, you need to wrap the respective source code in a toggle point (for example, an if-else statement) related to a particular feature flag. In the toggle point you access the service instance using the environment variables generated for your application and specify the feature flag name. Depending on the evaluation result of the feature flag (active, or inactive), your application serves the functionality in this specific toggle point.



Feature Flags Service Flow

1.1 Quick Start

Prerequisites

First, you need to go through the conceptual information about the Feature Flags service:

- Feature Flags service terms and their explanation. See Concepts [page 7].
- Delivery patterns and how you can benefit from using feature flags. See Delivery Techniques [page 12].
- The different approaches you can use to release your new functionality. See Release [page 15].

Procedure

These are the basic steps to start using the Feature Flags service:

1. Create a service instance of the Feature Flags service. See Create a Service Instance [page 20].
2. Bind the application to the Feature Flags service instance. The credentials and URI of this service instance appear in the environment variables of your application in the cloud cockpit. See Bind Your Application to the Feature Flags Service Instance [page 21].
3. Create your first feature flag. The simplest feature flag has a Boolean response type and no strategy is defined for it. See Create Feature Flags [page 23].
4. Implement toggle point inside your application. See Use APIs to Consume Feature Flags from Applications [page 31].

Example

If you want to try out a more detailed example, use the Feature Flags service demo application at https://github.com/SAP/cloud-cf-feature-flags-sample .

2 What's New for Feature Flags

2019

Technical Component	Capability	Environment	Title	Description	Type	Available as of
Feature Flags	Extension Suite - Development Efficiency	Cloud Foundry	Export Flags API	There is a new API that allows you to export all flags in a certain Feature Flags service instance without using the dashboard. See Export Flags API.	New	2019-06-06
Feature Flags	Extension Suite - Development Efficiency	Cloud Foundry	Overriding the State of the Feature Flags	When importing feature flags from a JSON file and updating the service instance, you can now preserve the state of these flags as part of this service update. To do that, you have to specify the parameter *overrideStatus* and set it to `false`. If you set it to `true`, then the service instance configuration will be exactly as specified in this JSON file. See Import Feature Flags.	New	2019-04-25
Feature Flags	Extension Suite - Development Efficiency	Cloud Foundry	Monitoring Feature Flags	When using the Feature Flags service, you can now: • View the history of a feature flag. • Check how much a feature flag has been evaluated for the past 10 days. • Check which feature flags have not been evaluated for a dedicated period of time. See Monitor Feature Flags.	New	2019-03-14

3 Concepts

These are the terms you need to understand to be able to use the Feature Flags service on the Cloud Foundry environment.

Feature Flag

A feature flag is associated with a functionality in your application that you want to deliver to your users. It has the following properties:

- Name: it has to be unique for feature flags in one service instance.
- Description: give a meaningful description of your feature flags, so that you can easily find out which feature flags to activate.
- Return type: Boolean, or String.
  - Boolean: the feature flag has only two variations, **true** and **false**.
  - String: the feature flag has two or more variations defined as a text. You can choose which variations to which tenants to deliver.
- State: active when you want a certain functionality to be available for your users, or inactive when you don't want this functionality to be available.
- Variations: the value returned when the feature flag is evaluated. Each feature flag has at least two variations. The variations are numbered in the dashboard starting from 1.
- Strategy: direct delivery, percentage delivery. The strategy adds the conditions to define which users would receive the new feature.
- Release information: released, or not released. The feature flag is released when the corresponding functionality is ready to reach all users.

You create the feature flags in a service instance. The maximum number of feature flags you can create depends on the service plan that you use.

Feature Flag Evaluation

You evaluate a feature flag when you want to check its state. To do this, you execute a GET request using the evaluation API. See Use APIs to Consume Feature Flags from Applications [page 31].

Batch Evaluation

You can evaluate the state of a subset of feature flags defined in a service instance with only one GET request using the batch evaluation API. See Batch Evaluation API [page 39].

Toggle Point

The toggle point wraps a piece of your application source code that you want to deliver in latent state, or to deliver using some of the supported deliveries in the feature flag strategy. An example of a toggle point is an `if-else` statement. See Use APIs to Consume Feature Flags from Applications [page 31].

Tenant

Multitenancy is a concept in the software architecture that allows an application to provide access to different groups of users. In this way the application guarantees the resources isolation for each separate group. The implementation of the multitenancy concept is application-specific. This means that for your application a tenant can be any of the following - subaccount, user, company name, and so on. For the direct delivery you can specify a list of entities that represent different tenants in your application.

Identifier Query Parameter

An `identifier` is a query parameter that is part of the evaluation request to the Feature Flags service. The value of the `identifier` represents the tenant of your application for whom you are performing a flag evaluation. The `identifier` is used when evaluating flags that have direct or percentage delivery as part of their strategy.

You provide the `identifier` query parameter in the source code of your application where you specify the toggle point. It must match the tenant that you use in the feature flag properties in the Feature Flags dashboard. See Use APIs to Consume Feature Flags from Applications [page 31].

Service Plans

There are two plans available: **lite** and **standard**. The standard plan is available only in the enterprise accounts. For more details about the plans, see the table below:

Plan	Details	Account Type
Lite	• Is for development or testing purposes only • Allows up to 10 flags to be managed • Only flags with no strategy defined are supported, this means that the feature marked with this flag will be either active, or inactive. No special deliveries are applied.	Trial Enterprise
Standard	• Is for productive use-cases • Allows up to 500 flags to be managed • Provides additional functionality such as direct delivery and percentage delivery • Gives you better control on feature rollout. For example, activating a feature flag requires Space Manager role, and deleting it requires the feature flag to be deactivated first. • After the new feature is ready to be delivered, you can release the respective feature flag.	Enterprise

Variations

The value returned when the feature flag is evaluated. Each feature flag has at least two variations.

You always have one variation that would be delivered. If the flag is inactive, that variation will be **variation 1**. You also have a default variation that is delivered if none of the deliveries in the strategy are met.

Boolean Return Type

When using feature flags with Boolean return type, you have two variations: **true** and **false**.

- If the feature flag is inactive regardless whether you have a strategy defined or not, you get variation 1 (false), and no one receives the new feature.
- If the feature flag is active and no strategy is defined, you get variation 2 (true), and all users receive the new feature.

- If the feature flag is active and a strategy is defined:
  - You can have at most one direct delivery which describes the tenants who receive the functionality (variation 2: true). Everyone else receives variation 1: false.
  - You can spread the variations among your users using the percentage delivery.
  - You can use a combination of the already described deliveries. In this case, consider the evaluation order.

String Return Type

When you have a feature flag with String return type, you can have 2 or more variations defined as text.

- If the feature flag is inactive regardless whether you have a strategy defined or not, you get variation 1.
- If the feature flag is active and no strategy is defined, you get the default variation.
- If the feature flag is active and a strategy is defined:
  - You can have one or more direct deliveries which describe the tenants who receive the specified variation (each direct delivery has exactly one variation associated with it). Everyone else receives the default variation.
  - You can spread the variations among your users using the percentage delivery.
  - You can use a combination of the already described deliveries. In this case, consider the evaluation order.

You can have as many direct deliveries as the number of variations you have and you can specify which tenants would receive which variation.

If you want to spread the variations to a certain percentage of your users, you use the percentage delivery.

The benefit from using many variations is that you can customize your features and experiment with different values without changing the source code itself.

Default Variation

This is the variation that is returned if the String flag has no strategy defined, or none of the deliveries in the strategy is matched when evaluating the flag.

Error Variation

Using the error variation you can specify which would be the variation that is returned as part of the evaluation response if a problem occurs during the evaluation. See Evaluation API [page 36].

> **i Note**
>
> Setting an error variation is optional.

Strategy

The strategy defines the conditions that will be applied when a feature flag is evaluated. The strategy is optional and if you use it, according to your scenario you may specify:

- One or more direct deliveries
- One percentage delivery
- Combination between one or more direct deliveries and/or a percentage delivery

> **i Note**
>
> If a variation takes part in the strategy, it cannot be deleted.

Direct Delivery

You can use the direct delivery strategy when you want to define special conditions for your feature flag. For example, you want only certain tenants to access the new functionality you provide.

When you define a feature flag that uses direct delivery strategy, you have to provide a list of values that represent different tenants in your application and you want the feature flag to be active only for them. When evaluating this feature flag, you need to provide an `identifier` query parameter with tenant value from the current context. For example, if you have a Company entity and the company name is a tenant in your application, then you need to add the company names in the dashboard and you should construct the URL inside your application in the following way:

```
<uri>/api/v1/evaluate/<feature_flag_name>?identifier=<Company.getName()>
```

The direct delivery strategy is available in the standard plan.

Percentage Delivery

You can use the percentage delivery when you want to distribute the variations you have defined for a feature flag among your users without targeting specific ones. In the Feature Flags dashboard, you can define in a percentage format which users will receive the respective variation. For example, you have a String feature flag, and you have defined three variations: variation 1: blue, variation 2: red, variation 3: green. Using the percentage delivery, you can specify that a certain amount of users would see one and the same button in the application UI in a different color:

- 60% of your users will receive variation 1 and will see the button in blue;
- 30% of your users will receive variation 2 and will see the button in red;
- 10% of your users will receive variation 3 and will see the button in green.

The percentage delivery strategy is available in the standard plan.

> **i Note**
>
> The percentage values and their sum must be between 0 and 100.

Evaluation Order

The order of the deliveries in a strategy is prioritized considering first the direct deliveries, then the percentage delivery, and at the end, the default variation. The same order appears in the dashboard when you specify a strategy for a flag.

Release

A new or improved functionality is ready to be released when:

- It is fully implemented and tested.
- The different variations are evaluated and you have chosen the proper variation that will be delivered to all users.

You can manage the release of this functionality through the Feature Flags service by releasing the related feature flag. There are two release options you can choose from:

- Release at once (by default)
- Gradual release

Release schedule is supported for both release options.

> **i Note**
>
> By default, the release option is set to release at once.

Related Information

Delivery Techniques [page 12]
Release [page 15]

3.1 Delivery Techniques

The Feature Flags service for the Cloud Foundry environment allows enabling or disabling new features without redeploying or restarting the application. You can benefit from using feature flags in the following delivery techniques:

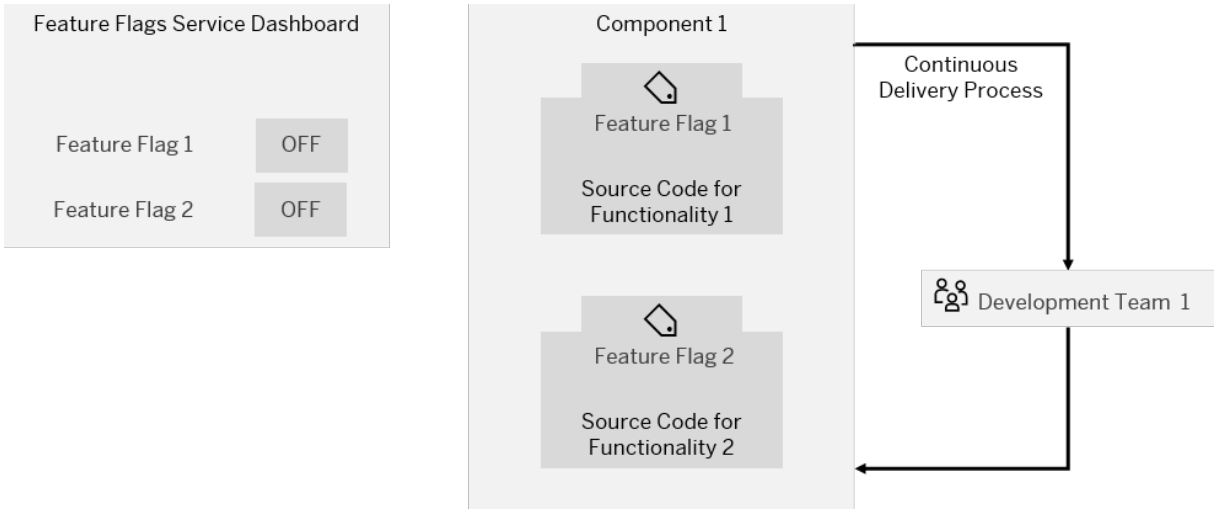
Latent Code Delivery

Latent code delivery is a technique when you deliver functionalities that you don't want to be accessed by your users. To do this, you need to:

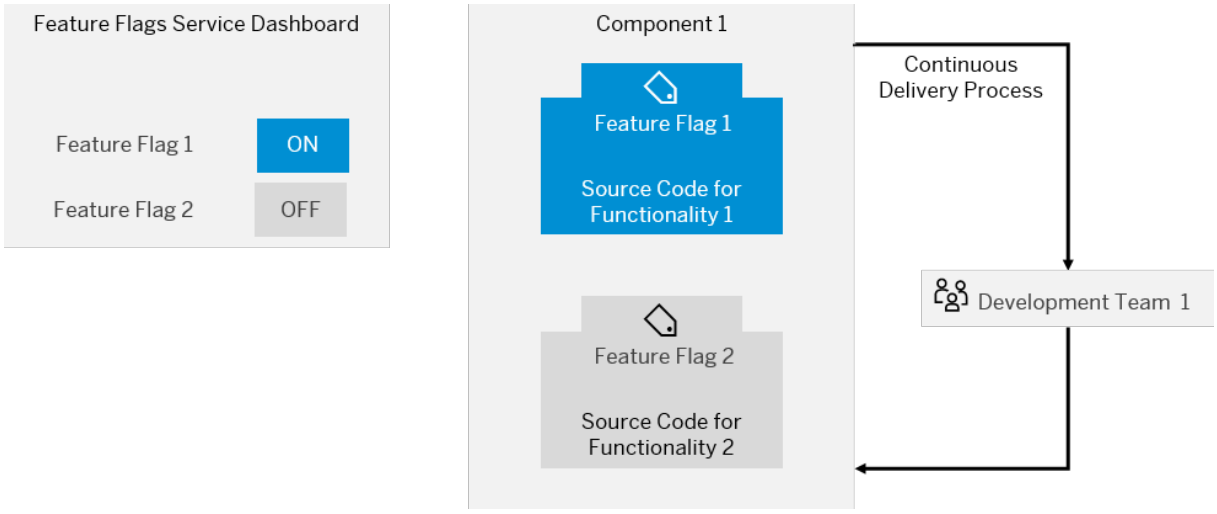
1. Implement a toggle point in your application code that wraps the latent code.
2. Associate this toggle point with an inactive feature flag.

By toggling this feature flag, you can switch this functionality on and off at runtime.

Benefits: The development and delivery of new features does not put the delivery of the whole product or service at risk.



Latent Code Delivery: Functionality 1 and Functionality 2 Are Not Available

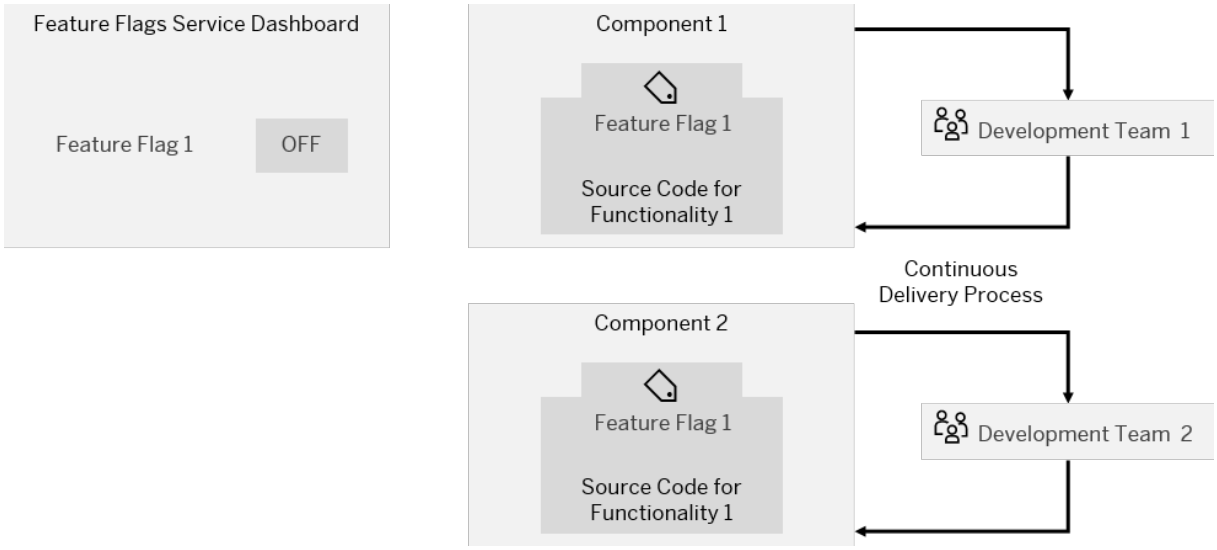


Latent Code Delivery: Functionality 1 Is Now Available

Synchronized Delivery

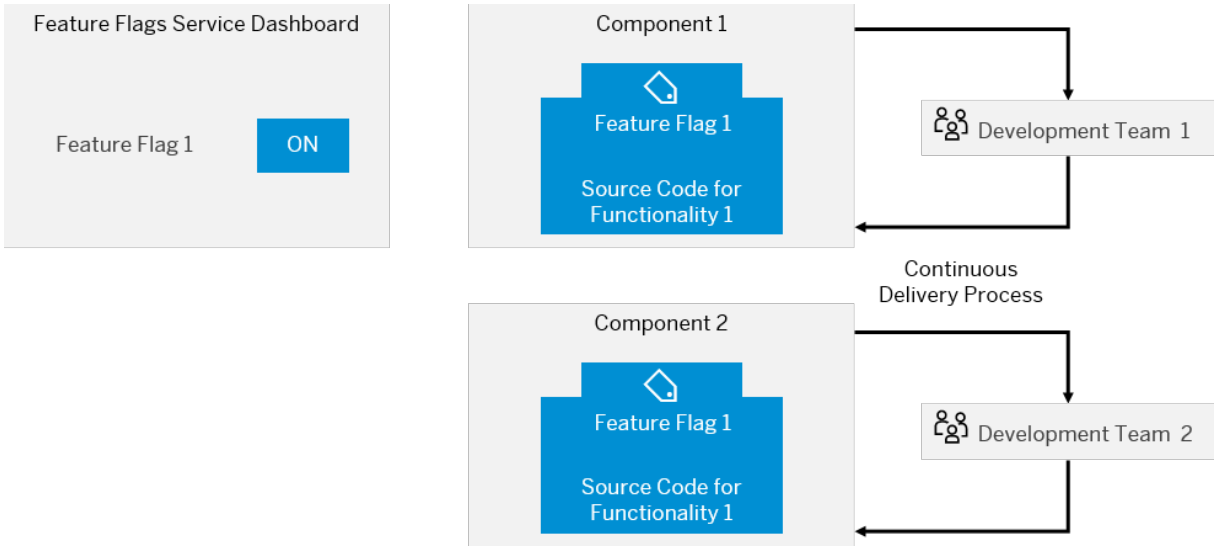
Simultaneous rollout of new functionalities in distributed systems is a challenge. When your solution is based on several components or microservices with different release cycles and developed by different teams this technique can help. Developers can wrap a code related to the new functionality in toggle points that are checking one and the same feature flag. All components deliver independently their part of the functionality as latent code. When all the components are ready, you can activate the feature flag associated with these toggle points. This will provide the new functionality to your users instantly.

Benefits: The development in the different modules is not blocked and the affected developers can plan their tasks better.



Synchronized Rollout: Functionality Not Available

The respective functionalities in Component 1 and Component 2 are fully implemented by both Development Team 1 and Development Team 2.



Synchronized Rollout: Functionality Available

Direct Delivery

Using the direct delivery technique, you can define which users will access a new feature. You build in your application logic different sets of users. Then, you configure in the feature flag related to the new functionality a direct shipment strategy including these sets of users as `identifier` query parameters. For more information, see Identifier Query Parameter [page 8], Direct Delivery [page 11] and Create Feature Flags [page 23].

Benefits: This approach is great for internal, or beta testing, where you can control which users will be the first to try out a new feature.

Percentage Delivery

Using the percentage delivery technique, you can release a new feature to a specified percentage of your users. You can also release multiple variations of this feature, each to a specified percentage of your users.

You define what percentage of your users would receive each variation of the feature. This approach is good for risk mitigation when releasing a new feature. It is also useful when you want to test how your users would react to the different variations of your feature. For more information, see Variations [page 9] and Percentage Delivery [page 11].

Benefits: You can release a feature to a specified percentage of your users to reduce risk and analyze performance. If you have several implementations of a certain feature, you can deliver each implementation to a specified percentage of your users and test which variation receives the most positive feedback.

Related Information

Concepts [page 7]
Developing Applications Using the Service [page 22]

3.2 Release

Feature Flags Release

Using feature flags you can set up different variations for a new feature to:

- Test different color schemes.
- Receive feedback by delivering it only to a small set of users.

When a new functionality is ready to be rolled out to all the users, and the proper variation is chosen, you can release the corresponding feature flag.

These are important aspects you need to consider:

- You can release feature flags only when they are active.
- The released variation will be delivered to all tenants.
- Once a feature is released, and the feature flag is in state *Released*, the flag can be deleted.

Release at Once

Using this option, you deliver the feature at once to all tenants starting immediately after releasing the flag. This is the default option.

Gradual Release

Using this option, you deliver the feature gradually spread in a defined number of days starting immediately after releasing the flag. You have to specify in the release settings of the flag the number of days that would take to spread the delivery. For example, if you define 5 days, this means that the new functionality will reach all users in 5 days. The number of users that would receive the functionality each day is automatically calculated in percentage format.

Scheduled Release

Both release options can be scheduled in time. You define the release option, release at once, or gradual release, and specify in advance the date when the release would start.

Cancel the Release

It is possible to cancel a release only if it has not reached all the users. This means that the only release that can be canceled is the Gradual release in which the percentage of users who received the functionality is less than 100%.

Related Information

Concepts [page 7]
Release Feature Flags [page 26]

3.3 JSON File Example

You can import feature flags in a JSON file:

- When creating a Feature Flags service instance. See Create a Service Instance [page 20].
- Explicitly to modify multiple flags at a time. See Import Feature Flags [page 29].

In this example, you import 5 feature flags:

- *flag-name-1*: Boolean, it has no strategy;
- *flag-name-2*: String, with 4 variations, equally distributed;
- *flag-name-3*: String, with 4 variations, the first two are distributed to 2 tenants;
- *flag-name-4*: String, with 3 variations, all of them are distributed to 3 tenants;
- *flag-name-5*: String, with 2 variations, distributed 25:75.

For all these flags, the property *overrideStatus* is set to **`true`**. This means that this JSON file will override the state of the feature flags described in it if these flags already exist in the service instance. If they do not exist, they will be created. For example, let's assume that the *flag-name-5* flag already exists in the service instance. In the JSON file example, the parameter *"enabled"* is set to *true*, which will activate the *flag-name-5* flag.

```
{
  "overrideStatus" : true,
  "flags":[
     {
        "id":"flag-name-1",
        "description":"description of flag-name-1",
        "directShipments":[
        ],
        "weightedChoices":[
        ],
        "variations":[
           "false",
           "true"
        ],
        "offVariationIndex":0,
        "variationType":"BOOLEAN",
        "defaultVariationIndex":1,
        "enabled":true
     },
     {
        "id":"flag-name-2",
        "description":"description of flag-name-2",
        "directShipments":[
        ],
        "weightedChoices":[
           {
              "variationIndex":0,
              "weight":25
           },
           {
              "variationIndex":1,
              "weight":25
           },
           {
              "variationIndex":2,
              "weight":25
           },
           {
              "variationIndex":3,
              "weight":25
           }
        ],
        "variationType":"STRING",
        "variations":[
           "red",
           "green",
           "blue",
           "yellow"
        ],
        "offVariationIndex":0,
        "defaultVariationIndex":3,
        "enabled":true
     },
     {
        "id":"flag-name-3",
        "description":"description of flag-name-3",
        "directShipments":[
           {
              "id":1,
              "variationIndex":0,
              "receivers":[
```

```
                "company-id-1"
            ]
         },
         {
            "id":2,
            "variationIndex":1,
            "receivers":[
               "company-id-2"
            ]
         }
      ],
      "weightedChoices":[
      ],
      "variationType":"STRING",
      "variations":[
         "red",
         "green",
         "blue",
         "yellow"
      ],
      "offVariationIndex":0,
      "defaultVariationIndex":3,
      "enabled":true
   },
   {
      "id":"flag-name-4",
      "description":"description of flag-name-4",
      "directShipments":[
         {
            "id":3,
            "variationIndex":0,
            "receivers":[
               "company-id-1"
            ]
         },
         {
            "id":4,
            "variationIndex":1,
            "receivers":[
               "company-id-2"
            ]
         },
         {
            "id":5,
            "variationIndex":2,
            "receivers":[
               "company-id-3"
            ]
         }
      ],
      "weightedChoices":[
      ],
      "variationType":"STRING",
      "variations":[
         "red",
         "green",
         "blue"
      ],
      "offVariationIndex":0,
      "defaultVariationIndex":1,
      "enabled":false
   },
   {
      "id":"flag-name-5",
      "description":"description of flag-name-5",
      "directShipments":[
      ],
      "weightedChoices":[
```

```
            {
               "variationIndex":0,
               "weight":25
            },
            {
               "variationIndex":1,
               "weight":75
            }
         ],
         "variationType":"STRING",
         "variations":[
            "red",
            "green"
         ],
         "offVariationIndex":0,
         "defaultVariationIndex":0,
         "enabled":true
      }
   ]
}
```

4 Initial Setup

These are the steps you need to follow to set up the Feature Flags service:

1. Create a Service Instance [page 20]
2. Bind Your Application to the Feature Flags Service Instance [page 21]

4.1 Create a Service Instance

Follow these steps to create a service instance.

Context

To use the Feature Flags service, first you need to create a service instance in the SAP Cloud Platform cockpit. There are two plans available: lite and standard. If you have a trial account, you can select only the lite plan. If you have an enterprise account, you can select either the lite plan, or the standard plan. For more information, see Service Plans [page 9].

When you are creating the service instance, you can import feature flags using a JSON file. You get this JSON file when you export all feature flags from another service instance, or you create it using this example: JSON File Example [page 16].

Depending on whether you are using Cloud Foundry or Kyma environment, you have to follow different steps to create a Feature Flags service instance:

- In the Cloud Foundry environment: Creating Service Instances
- In the Kyma environment: Creating Service Instances

4.2 Bind Your Application to the Feature Flags Service Instance

Follow these steps to bind your application to the Feature Flags service instance.

Prerequisites

- Have an application deployed in the Cloud Foundry environment.
- Have a Feature Flags service instance created.

Context

Before consuming any feature flags from your application, you need to bind your application to the Feature Flags service instance in the SAP Cloud Platform cockpit.

Depending on whether you are using Cloud Foundry or Kyma environment, you have to follow different steps to create a Feature Flags service instance:

- In the Cloud Foundry environment: Binding Service Instances to Applications

  > **i** Note
  >
  > In the cloud cockpit, in the environment variables of your application, under *VCAP_SERVICES* > *your_feature_flag_service_name* > *credentials* appears information about your Feature Flags service instance credentials and URI.

- In the Kyma environment: Binding Service Instances to Applications

Next Steps

Create Feature Flags [page 23]

5 Developing Applications Using the Service

The Feature Flags service for Cloud Foundry environment allows enabling or disabling new features without redeploying or restarting the application.

These are the steps you need to follow, to start using the feature flags for your application. To learn more about the feature flags concept and basics, see:

- What Is Feature Flags [page 3]
- Concepts [page 7]

Prerequisites

- Have a subaccount, org, and space in the Cloud Foundry environment. See Get Started.
- Have your application deployed on Cloud Foundry environment.
- Have at least Space Developer role in this space. This is a detailed list of the roles in the Cloud Foundry environment that allow you to manage feature flags depending on the plan.

Operations in the Feature Flags Dashboard	Lite Plan		Standard Plan	
Access the dashboard	Space Developer	Space Manager	Space Developer	Space Manager
Create feature flags	Space Developer	Space Manager	Space Developer	Space Manager
Activate/deactivate feature flags	Space Developer	Space Manager	N/A	Space Manager
Update feature flags' properties	Space Developer	Space Manager	Space Developer	Space Manager
Delete feature flags	Space Developer	Space Manager	Space Developer	Space Manager

Using the Feature Flags Service

To use the Feature Flags service, you have to:

- Create and configure feature flags. See Configure the Feature Flags Service [page 23].
- Evaluate these feature flags using the APIs. See Use APIs to Consume Feature Flags from Applications [page 31].

5.1 Configure the Feature Flags Service

Context

The Feature Flags service has a dedicated dashboard in the SAP Cloud Platform cockpit. You can use this dashboard to perform the following tasks:

- Create Feature Flags [page 23]
- Edit Features Flags [page 25]
- Release Feature Flags [page 26]
- Monitor Feature Flags [page 27]
- Export and Import Feature Flags [page 28]
- Delete Feature Flags [page 30]

Related Information

Use APIs to Consume Feature Flags from Applications [page 31]

5.1.1 Create Feature Flags

Follow these steps to create a feature flag.

Prerequisites

- Have an instance of the Feature Flags service created. See Create a Service Instance [page 20].
- Have your application bound to this instance of the Feature Flags service. See Bind Your Application to the Feature Flags Service Instance [page 21].
- Have at least Space Developer role in this space. This is a detailed list of the roles in the Cloud Foundry environment that allow you to manage feature flags depending on the plan. See Developing Applications Using the Service [page 22].

Context

Follow these steps to create a feature flag. Have in mind that you can define some of the fields, such as the feature flag's name and type, only when creating the flag. You cannot modify them later on.

Procedure

1. Open the SAP Cloud Platform cockpit and navigate to your space in the Cloud Foundry environment.
2. Choose *Services* > *Service Instances*.
3. In the row of the instance of your Feature Flags service, choose *(Open Dashboard)* from the *Actions* column. The *Feature Flags Dashboard* opens after successful authentication.
4. Choose *New Flag*.
   a. Fill in the feature flag basic information: *Name* (mandatory field) and *Description* (mandatory field). Once you save the flag for the first time, you cannot change its name anymore. The description of the flag can be changed in edit mode.

      > **i Note**
      >
      > The name can contain only small Latin letters, numbers, underscores, dots, and dashes. It can be up to 255 symbols long. You cannot have two feature flags with the same name in one service instance.

   b. Choose between the two flag types *Boolean* (default) and *String*. When using the String flag type, you can define up to five variations as a String. The flag type can be selected only when creating the feature flag and cannot be changed in edit mode.
   c. Choose the state *Active* or *Inactive* (default) for the flag. This state is applied after you complete and save the configuration.
   d. Define the feature flag variations for String flag type.

      > **i Note**
      >
      > The variations for Boolean flag type are predefined and cannot be changed: *Var. 1* is with value **false** and will be returned when the flag is inactive and *Var. 2* is with value **true** and will be returned when the flag is active.

      - Define String values for the *Var. 1* and *Var. 2* variations. The String values have to be different. You can change these String values in edit mode but you cannot delete these two variations.
      - You can add up to three more variations (Var. 3, Var .4 and Var 5) by choosing + *(Add)*. The String values for these variations also have to be unique for the flag. When editing the flag, you can change the String values of these variations. If there is no strategy defined for these three additional variations, you can also delete them by using *(Delete)*.

      > **i Note**
      >
      > When the flag is inactive, the String value defined for Var. 1 will be returned. When the flag is active, you will get the String of the variation that is evaluated based on the strategy defined (see below the instruction for strategy definitions).

   e. Define the feature flag delivery strategy.

      > **i Note**
      >
      > You can choose a strategy and variations only in the standard plan. The strategy is considered only when the flag is active.

You can either choose one of the strategies, or combine them according to your delivery purposes. The strategies are prioritized considering first the direct delivery, then the percentage delivery, and at the end, the default variation.

- *Direct Delivery*: you define which variation to what tenants to be delivered directly. These tenants are specific for your application and may be subaccounts, user IDs, user emails, or e-mail domain, depending on your application logic. You have to list the String values to define the tenants for the specific variation.

  > **i Note**
  >
  > For Boolean feature flags, you define the list with tenants for direct delivery only for Var. 2 (true).

- *Percentage Delivery*: you can choose to ship the different variations to a predefined percentage of your users. To change the default percentage values, click the respective part of the donut chart.

  > **i Note**
  >
  > The percentage values and their sum must be between 0 and 100.

f. Define a default variation. This is the variation that is delivered if none of the rules in the strategies are met.

> **i Note**
>
> When creating a feature flag, consider the evaluation order. See Evaluation Order [page 11].

g. Choose *Save*.

Next Steps

Use APIs to Consume Feature Flags from Applications [page 31]

5.1.2 Edit Features Flags

Follow these steps to change the properties of a feature flag.

Prerequisites

Have a feature flag created. See Create Feature Flags [page 23].

Context

You can modify some of the properties of a feature flag, such as:

- Description
- String values of the variations of String feature flag

> **i** Note
>
> You cannot change the feature flag's name and type. You can define them only when creating the flag.

Procedure

1. Open the SAP Cloud Platform cockpit and navigate to your space in the Cloud Foundry environment.
2. Choose *Services* > *Service Instances*.
3. In the row of the instance of your Feature Flags service, choose *(Open Dashboard)* from the *Actions* column. The *Feature Flags Dashboard* opens after successful authentication.
4. In the *Feature Flags Dashboard* select the row of a feature flag, and choose *(Edit)* from the *Actions* column.
5. Make the necessary changes in the properties that are available for editing.
6. Choose *Save*.

5.1.3 Release Feature Flags

Context

When a new functionality is ready to be rolled out to all the users, you have to release the corresponding feature flag. When you do that, the flag becomes in state *Released* and can be deleted from the Feature Flags dashboard.

You can choose from on of the following release options:

- Release at once
- Gradual release
- Scheduled release

For more information, see Release [page 15].

Procedure

1. To release a feature flag, first you need to activate it.
2. Select this flag from the dashboard and choose *Release*.

> **i Note**
>
> Have in mind, that after releasing the feature flag and the release is completed, you cannot cancel the release.

3. In the *Deliver* field, select the variation you want to deliver.
4. In the *Starting* field, you can define whether to start delivering the variation immediately, or to schedule it.
5. In the *Reaching all users* field, you can define whether to deliver the variation to all users at once, or to do it gradually. To do that, you have to specify the number of days that would take the functionality to reach all the users. The number of users that will receive this functionality each day is calculated automatically in percentage format.
6. Choose *Release*.

Related Information

Delete Feature Flags [page 30]

5.1.4 Monitor Feature Flags

Follow these steps to monitor a feature flag.

Prerequisites

Have a feature flag created. See Create Feature Flags [page 23].

Context

The following options can help you improve the way you are using the Feature Flags service:

- Monitor the evaluation calls of the feature flag for the past 10 days
- Check which feature flags have not been evaluated for a dedicated period of time
- View the history of a feature flag
- Filter the feature flags by their state

Procedure

1. Open the SAP Cloud Platform cockpit and navigate to your space in the Cloud Foundry environment.
2. Choose *Services* > *Service Instances*.
3. In the row of the instance of your Feature Flags service, choose *(Open Dashboard)* from the *Actions* column. The *Feature Flags Dashboard* opens after successful authentication.
4. To check the evaluation graphic with evaluation calls of the feature flag for the past 10 days, select the feature flag in the dashboard and choose *Monitoring* from the navigation area.
5. To check which feature flags have not been evaluated for a dedicated period of time, choose *Unused Flags* from the navigation area in the dashboard. A list of all feature flags that have not been evaluated for the past 90 days appears. You can change the default period. When a feature flag has been evaluated, it is removed from the *Unused Flags* list.
6. To view the history of a feature flag, select this feature flag, and choose *History* from the navigation area. A list of all the changes related to this feature flag appears staring from the most recent ones. You can filter these changes by the person that made them, or by other keyword.
7. To filter the feature flags by their state, choose the dropdown menu next to the *Search* field.

5.1.5 Export and Import Feature Flags

You can export the feature flags using the Feature Flags dashboard. To import feature flags, you have to update the Feature Flags service instance using the Cloud Foundry Command Line Interface (cf CLI).

Related Information

Export Feature Flags [page 28]
Import Feature Flags [page 29]

5.1.5.1 Export Feature Flags

Procedure

1. To export all feature flags, go the Feature Flags dashboard.
2. Choose *Export Flags*. You get a JSON file that you can use to import these feature flags in another service instance.

> **i Note**
>
> You can import this JSON file only when creating a new service instance.

Related Information

Export Flags API [page 36]

5.1.5.2 Import Feature Flags

You can import feature flags in a JSON file and as a result you update the service instance.

Prerequisites

- Have the cf CLI downloaded and installed.
- Have access to the space where your Feature Flags service instance resides through the cf CLI.

Context

By updating the service instance, you can import feature flags. This allows you to:

- Update the metadata of existing feature flags
- Add new feature flags
- Remove feature flags that you no longer need

> **i Note**
>
> You cannot change the plan by updating the service instance. See Service Plans [page 9]

To import feature flags you need to prepare a JSON file with a specific structure. This is a sample JSON file that you can modify for your case:

> **i Note**
>
> The JSON structure is the same as the JSON file that you get when exporting a feature flag from the Feature Flags dashboard. See *Step 8. Exporting Feature Flags*.

Procedure

1. Prepare a JSON file with the feature flags you want to import. See JSON File Example [page 16].

> **Note**
>
> Additionally, you can add the parameter *overrideStatus* with value **`true`** or **`false`**. If you set it to **`true`**, then the service instance configuration will be exactly as specified in this JSON file. This is also the default behavior, if you don't specify this parameter.

2. Log in to the cf CLI.
3. Navigate to the space where your Feature Flags service instance resides.

   ```
   cf target -s <space_name>
   ```

4. Run the following command:

   ```
   cf update-service <service_instance_name> -c <path_to_JSON>
   ```

Results

The the service instance will be updated as follows based on the value of the *"overrideStatus"* parameter in the JSON file:

- *true*, then:
  - Existing feature flags that are provided again in the JSON file, will be updated with the metadata specified in the file.
  - Existing feature flags that are **not** provided again in the JSON file will be deleted.
  - New feature flags that are provided in the JSON file, will be created.
- *false*, then:
  - Existing feature flags that are provided again in the JSON file, will be updated with the metadata specified in the file except for the flag's state (represented by the *enabled* parameter in the JSON file). The flag's state will remain the same as specified previously.
  - Existing feature flags that are **not** provided again in the JSON file will remain unchanged.
  - New feature flags that are provided in the JSON file, will be created.

5.1.6 Delete Feature Flags

Follow these steps to delete a feature flag.

Prerequisites

Have at least Space Developer role in this space. This is a detailed list of the roles in the Cloud Foundry environment that allow you to manage feature flags depending on the plan. See Developing Applications Using the Service [page 22].

> **Note**
>
> In the standard plan you cannot delete active feature flags.

Context

When the feature is released, and the feature flag is in state `Released`, you can delete the flag.

Procedure

1. Open the cloud cockpit and navigate to your space in the Cloud Foundry environment.
2. Choose *Services* > *Service Instances*.
3. To delete a feature flag, use *(Delete)* in the *Actions* column.

5.2 Use APIs to Consume Feature Flags from Applications

Follow these steps to create toggle points in your application source code.

Prerequisites

- Have a Feature Flags service instance created. See Create a Service Instance [page 20].
- Have your application bound to this service instance. See Bind Your Application to the Feature Flags Service Instance [page 21].
- Have a feature flag created. See Create Feature Flags [page 23].

Context

To switch on and off functional parts of your application, you need to create a feature flag in the Feature Flags dashboard in the SAP Cloud Platform cockpit. After that you have to add a toggle point in your application source code for that specific flag. To do this, you need:

- The flag name from the Feature Flags dashboard in the cloud cockpit. See Create Feature Flags [page 23].
- The URI, username and password generated for this Feature Flags service instance from the application environment variables. See Bind Your Application to the Feature Flags Service Instance [page 21].

Apply the necessary changes in your application source code so that you can use the feature flags:

Procedure

1. Extract all the needed configuration data from the environment. This is information about the Feature Flags service instance such as the URI and the credentials that are supplied at runtime as part of the VCAP application property. See https://docs.cloudfoundry.org/devguide/deploy-apps/environment-variable.html .

   The Feature Flags service provides:

   - Root for the service REST API.
   - Username and password for this service instance.
2. Evaluate the feature flag value in the current context using the evaluation API. See Evaluation API [page 36].

   > **Note**
   >
   > If you need to evaluate the value of several feature flags at a time, to avoid multiple HTTP calls, you can use the batch evaluation API. See Batch Evaluation API [page 39].

   1. Use the respective address for the evaluation API or the simplified evaluation API (only for Boolean feature flags) endpoint.

      > **Note**
      >
      > If your feature flag uses any strategy you also have to provide an `identifier` query parameter. This parameter specifies the application specific tenant, for which the evaluation is performed. The tenant depends on your application logic. For example, if you have a Company entity and the company name is as a tenant in your application, then you need to add the company names in the dashboard and you should construct the URL inside your application the following way: `<uri>/api/v2/evaluate/<feature_flag_name>?identifier=<Company.getName()>`.

   2. Provide an authentication for the evaluation of the evaluation API endpoint. Add a Basic Authentication header with credentials from the VCAP application property as a base64-encoded String to the HTTP request:
3. Create a code snippet for each one of the possible evaluation results and wrap them in a toggle point. The toggle point is a call to the evaluation API and includes either `if-else` statements or a switch statement over the parsed result from the evaluation call. See Evaluation API [page 36].

Related Information

Feature Flags Service Demo Application
Monitor Feature Flags [page 27]

5.2.1 Simplified Evaluation API

When you have a feature flag with **Boolean** return type, you can use the simplified evaluation API. The response for successful evaluation requests does not contain a body. Instead, different response status codes are returned.

Routes

GET /api/v1/evaluate/{feature_flag_name}

Request Parameters

Path

Parameter	Required	Data Type	Description
feature_flag_name	Yes	String	Name of the feature flag, it has to be unique for a particular service instance.

Query

Parameter	Required	Data Type	Description
identifier	No	String	Specifies the application specific tenant, for which the evaluation is performed.

Example

```
GET /api/v1/evaluate/boolean_feature_flag?identifier=<tenant>
```

Responses

Status Code 200 OK

The feature flag is active.

Status Code 204 NO CONTENT

The feature flag is inactive.

Status Code 400 BAD REQUEST

The feature flag has direct delivery or percentage delivery as part of its strategy and no `identifier` query parameter is provided.

Status Code 404 NOT FOUND

The feature flag is missing.

Status Code 422 UNPROCESSABLE ENTITY

The feature flag that is evaluated is not Boolean. The simplified evaluation API supports only Boolean feature flags.

Status Code 500 INTERNAL SERVER ERROR

An error has occurred within the Feature Flags service.

Code Snippet Examples

These are examples for implementing a toggle point in a Java application. The best practice is to execute the new logic only in case the feature flag is active. For all other cases, execute the fallback scenario.

The first code snippet contains a basic HTTP client:

```
// String evaluateUrlFormat = "%s/api/v1/evaluate/%s?identifier=<tenant_ID>"; //
in case of direct shipment flags, identifier is mandatory
String evaluateUrlFormat = "%s/api/v1/evaluate/%s"; // in case of unconditional
flags
String uri = <URI_FROM_VCAP_SERVICES>; // get URI for Feature Flags service
instance from VCAP_SERVICES
String username = <USERNAME_FROM_VCAP_SERVICES>; // get username for Feature
Flags service instance from VCAP_SERVICES
String password = <PASSWORD_FROM_VCAP_SERVICES>; // get password for Feature
Flags service instance from VCAP_SERVICES
String feature = "myFeatureFlagName"; // the name of the feature flag defined in
the dashboard in the cloud cockpit
String evaluationUri = String.format(evaluateUrlFormat, uri, feature);
String credentials = Base64.encodeBase64String(new String(username + ":" +
password).getBytes());
HttpClient httpClient = HttpClientBuilder.create().build();
HttpUriRequest evaluateRequest = new HttpGet(evaluationUri);
evaluateRequest.addHeader("Authorization", "Basic " + credentials);
HttpResponse response = httpClient.execute(evaluateRequest);
int statusCode = response.getStatusLine().getStatusCode();
switch (statusCode) {
    case 200: // active
        // logic when flag is active
        ...
        break;
    case 204: // inactive
        break;
    case 404: // missing
        break;
    case 400: // input error
        break;
    default: // in all other cases
          // logic when flag is inactive
        ...
        break;
}
```

This code snippet uses `OkHttpClient` and a custom Authenticator:

```
OkHttpClient okHttpClient = new OkHttpClient.Builder().authenticator(new
Authenticator() {
```

```
    @Override
    public Request authenticate(Route route, Response response) throws
IOException {
        String credentials = Credentials.basic("<username>", "<password>");
        return response.request().newBuilder().header("Authorization",
credentials).build();
    }
}).build();
//try (Response response = okHttpClient.newCall(new
Request.Builder().url("https://<uri>/api/v1/evaluate/<flag_name>?
identifier=<tenant_id>").build()).execute()) {
try (Response response = okHttpClient.newCall(new Request.Builder().url("https://
<uri>/api/v1/evaluate/<flag_name>").build()).execute()) {
    switch (response.code()) {
        case 200:
            break;
        case 204:
            break;
        case 400:
            break;
        case 404:
            break;
        default:
            break;
    }
}
```

This code snippet uses `OkHttpClient`, a custom Authenticator and URL builder:

```
OkHttpClient okHttpClient = new OkHttpClient.Builder().authenticator(new
Authenticator() {
    @Override
    public Request authenticate(Route route, Response response) throws
IOException {
        String credentials = Credentials.basic("<username>", "<password>");
        return response.request().newBuilder().header("Authorization",
credentials).build();
    }
}).build();
URL url = new HttpUrl.Builder().
    scheme("https").
    host("<uri>").
    addPathSegment("api/v1/evaluate").
    addPathSegment("<flag_name>").
    //addQueryParameter("identifier", "<tenant_id>"). // in case of direct
delivery strategy
    build().url();
try (Response response = okHttpClient.newCall(new
Request.Builder().url(url).build()).execute()) {
    switch (response.code()) {
        case 200:
            break;
        case 204:
            break;
       case 400:
            break;
        case 404:
            break;
        default:
            break;
    }
```

5.2.2 Export Flags API

You can export the feature flags using the Export Flags API.

When exporting the feature flags from a Feature Flags service instance, you get a JSON file containing all the flags from that service instance, together with their metadata. Then, you can:

- Import this JSON file in another Feature Flags service instance.
- Modify the metadata of the flags when updating the service instance.

See:

- JSON File Example [page 16]
- Export and Import Feature Flags [page 28]

Routes

GET /api/v1/features/export

Example

```
GET /api/v1/features/export
```

Responses

Status Code 200 OK

You get a JSON file containing all feature flags that are part of a Feature Flags service instance.

If there are no flags in that service instance, you get an empty JSON file, such as:

```
{
    "flags": []
}
```

5.2.3 Evaluation API

You can evaluate the feature flags using the evaluation API with both Boolean and String return types.

> **i** Note
>
> The simplified evaluation API is applicable only for Boolean feature flags. See Simplified Evaluation API [page 33].

Routes

GET /api/v2/evaluate/{feature_flag_name}

Request Parameters

Path

Parameter	Required	Data Type	Description
feature_flag_name	Yes	String	Name of the feature flag, it has to be unique for a particular service instance.

Query

Parameter	Required	Data Type	Description
identifier	Only if the evaluated feature flag has either direct deliveries or a percentage delivery as part of its strategy.	String	Specifies the application specific tenant, for which the evaluation is performed.
errorVariation	No	Integer	Specifies the index of the variation that will be returned (indexing starts from 1) as part of the evaluation response if an error occurs during the evaluation. In case this parameter is not specified, variation 1 returned.

Examples

```
GET /api/v2/evaluate/<feature_flag_name>?identifier=company_id
```

```
GET /api/v2/evaluate/<feature_flag_name>?errorVariation=2
```

```
GET /api/v2/evaluate/<feature_flag_name>?identifier=company_id&errorVariation=2
```

> **i Note**
>
> The `identifier` query parameter is mandatory only if the evaluated feature flag has either direct deliveries or a percentage delivery as part of its strategy.
>
> The `errorVariation` is optional and the variation it refers to will be returned in case of the error.

Responses

Status Code 200 OK

When the evaluation request is successful. The response body contains the *featureName, type, variation* fields.

```
{
    "featureName": "<feature_flag_name>",
    "type": "<feature_flag_response_type>",
    "variation": "<variation_value_returned_from_evaluation>"
}
```

For example, in the simplest case you have an **active Boolean** feature flag called *sidemenu-flag* with no strategy. The evaluation call looks like this:

```
GET /api/v2/evaluate/sidemenu-flag
```

```
{
    "featureName": "sidemenu-flag",
    "type": "BOOLEAN",
    "variation": "true"
}
```

Status Code 400 BAD REQUEST

When you have an active feature flag with a direct delivery strategy defined and you haven't provided an identifier:

> **i** Note
>
> The `variation`, `type` and `featureName` fields are still part of the response body even when the response is not `200 OK`. In such cases additional fields are added to the response body that describe the error.

```
{
    "httpStatus": 400,
    "message": "Input validation error",
    "errors": [
        "request :: Query parameter [identifier] must be provided in order to
evaluate the specified feature"
    ],
    "featureName": "contacts-button-color-flag",
    "type": "STRING",
    "variation": "blue"
}
```

Status Code 404 NOT FOUND

When the feature flag that is evaluated does not exist in the particular service instance.

```
{
    "httpStatus": 404
}
```

Status Code 500 INTERNAL SERVER ERROR

When an error occurs within the Feature Flags service.

5.2.4 Batch Evaluation API

You can evaluate a set of multiple feature flags using the batch evaluation API with both Boolean and String return types. To evaluate multiple feature flags at a time, you need to provide in the endpoint a subset of the feature flag names defined in one service instance and a set of the identifiers defined for these flags.

Routes

GET /api/v2/evaluateset

Request Parameters

Query

Parameter	Required	Data Type	Description
flag	Yes	String	Specifies the name of the feature flag, created in a particular service instance.
identifier	Only if the evaluated feature flag has either direct deliveries or a percentage delivery as part of its strategy.	String	Specifies the application specific tenant, for which the evaluation is performed.

Examples

```
GET /api/v2/evaluateset?
flag=<feature_flag_name_1>&...&flag=<feature_flag_name_N>&identifier=<company_id_1>
&...&identifier=<company_id_N>
```

Where:

- <feature_flag_name_1> to <feature_flag_name_N> are a subset of the feature flags defined in the service instance.
- <company_id_1> to <company_id_N> are also a set of identifiers that will be used for all flags that require an identifier.

The whole set of identifiers will be used for evaluating each flag.

```
GET /api/v2/evaluateset?
flag=<feature_flag_name_1>&flag=<feature_flag_name_2>&identifier=<company_id_1>&ide
ntifier=<company_id_2>
```

> **i Note**
>
> The `identifier` query parameter is mandatory only if the evaluated feature flag has either direct deliveries or a percentage delivery as part of its strategy. The feature flag will be evaluated in the evaluation set even if you don't specify the respective identifier in the endpoint.

Responses

Status Code 200 OK

The evaluateset request is successful for all the feature flags that are part of the evaluation set. The response body contains the *httpStatus, type, variation* fields for every feature flag the has been part of the evaluation set and its respective identifier.

> **i** Note
>
> If you don't specify any identifier in the endpoint, then, to keep the response structure the same, you will get *undef* as an identifier in the response body for each of the evaluated flags.

```
GET /api/v2/evaluateset?flag=sidemenu-flag&flag=footer-flag
{
    "sidemenu-flag": {
        "undef": {
            "httpStatus": 200,
            "type": "BOOLEAN",
            "variation": "true"
        }
    },
    "footer-flag": {
        "undef": {
            "httpStatus": 200,
            "type": "BOOLEAN",
            "variation": "true"
        }
    }
}
```

```
GET /api/v2/evaluateset?flag=sidemenu-flag&flag=footer-
flag&identifier=<company_id_1>&identifier=<company_id_2>
{
    "sidemenu-flag": {
        "company_id_2": {
            "httpStatus": 200,
            "type": "BOOLEAN",
            "variation": "true"
        },
        "company_id_1": {
            "httpStatus": 200,
            "type": "BOOLEAN",
            "variation": "true"
        }
    },
    "contacts-button-color-flag": {
        "company_id_2": {
            "httpStatus": 200,
            "type": "STRING",
            "variation": "green"
        },
        "company_id_1": {
            "httpStatus": 200,
            "type": "STRING",
            "variation": "blue"
        }
    }
}
```

Status Code 207 MULTI-STATUS

The evaluateset request returns a combination of different HTTP statuses for each of the evaluated feature flags that are part of the evaluation set. The response body contains again the *httpStatus, type, variation* fields

for every feature flag that has been part of the evaluation set and its respective identifier. In case there is a feature flag that returns *400 BAD REQUEST*, additional fields are added to the response body that describe the error.

> **Note**
>
> If you don't specify any identifier in the endpoint, then, to keep the response structure the same, you will get *undef* as an identifier in the response body for each of the evaluated flags.

```
GET /api/v2/evaluateset?flag=sidemenu-flag&flag=footer-flag
{
    "sidemenu-flag": {
        "company_id_2": {
            "httpStatus": 400,
            "type": "BOOLEAN",
            "variation": "true"
        },
        "company_id_1": {
            "httpStatus": 200,
            "type": "BOOLEAN",
            "variation": "true"
        }
    },
    "contacts-button-color-flag": {
        "company_id_2": {
            "httpStatus": 200,
            "type": "STRING",
            "variation": "green"
        },
        "company_id_1": {
            "httpStatus": 200,
            "type": "STRING",
            "variation": "blue"
        }
    }
}
```

Status Code 400 BAD REQUEST

In case you have a set of active feature flags with a direct delivery strategy defined and you haven't provided an identifier for any of these flags:

> **Note**
>
> The `httpStatus`, `variation`, and `type` fields for every feature flag are still part of the response body. In such cases additional fields are added to the response body that describe the error.

> **Note**
>
> If you don't specify any identifier in the endpoint, then, to keep the response structure the same, you will get *undef* as an identifier in the response body for each of the evaluated flags.
>
> ```
> GET /api/v2/evaluateset?flag=sidemenu-flag&flag=footer-flag
> {
>     "sidemenu-flag": {
>         "undef": {
>             "httpStatus": 200,
>             "type": "BOOLEAN",
>             "variation": "true",
>             "href": "https://feature-
> flags.cfapps.sap.hana.ondemand.com/api/v1/feature/sidemenu-flag"
> ```

```
            }
        },
        "footer-flag": {
            "undef": {
                "httpStatus": 400,
                "message": "Invalid validation error",
                "errors": [
                    "Query parameter [identifier] must be provided in order to
 evaluate the specified feature"
                ],
                "type": "BOOLEAN",
                "variation": "false",
                "href": "https://feature-
 flags.cfapps.sap.hana.ondemand.com/api/v1/feature/footer-flag"
            }
        }
 }
```

```
GET /api/v2/evaluateset?flag=sidemenu-flag&flag=footer-flag
{
    "sidemenu-flag": {
        "company_id_2": {
            "httpStatus": 400,
            "message": "Input validation error",
            "errors": [
                "request :: Query parameter [identifier] must be provided in
order to evaluate the specified feature"
            ],
            "type": "BOOLEAN",
            "variation": "true"
        },
        "company_id_1": {
            "httpStatus": 400,
            "message": "Input validation error",
            "errors": [
                "request :: Query parameter [identifier] must be provided in
order to evaluate the specified feature"
            ],
            "type": "BOOLEAN",
            "variation": "true"
        },
    "contacts-button-color-flag": {
        "company_id_2": {
            "httpStatus": 400,
            "message": "Input validation error",
            "errors": [
                "request :: Query parameter [identifier] must be provided in
order to evaluate the specified feature"
            ],
            "type": "STRING",
            "variation": "green"
        },
        "company_id_1": {
            "httpStatus": 400,
            "message": "Input validation error",
            "errors": [
                "request :: Query parameter [identifier] must be provided in
order to evaluate the specified feature"
            ],
            "type": "STRING",
            "variation": "blue"
        }
    }
}
```

Status Code 404 NOT FOUND

If the feature flags that are evaluated do not exist in the particular service instance.

> **i** Note
>
> If you don't specify any identifier in the endpoint, then, to keep the response structure the same, you will get *undef* as an identifier in the response body for each of the evaluated flags.

```
GET /api/v2/evaluateset?flag=sidemenu-flag&flag=footer-
flag&identifier=<company_id_1>&identifier=<company_id_2>
{
    "sidemenu-flag": {
        "company_id_2": {
            "httpStatus": 404
        },
        "company_id_1": {
           "httpStatus": 404
        }
    },
    "contacts-button-color-flag": {
        "company_id_2": {
            "httpStatus": 404
        },
        "company_id_1": {
            "httpStatus": 404
        }
    }
}
```

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
    - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
    - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

THE BEST RUN SAP