



User Guide | PUBLIC
2021-08-23

SAP BTP, Serverless Runtime

Content

1	What Is Serverless Runtime?	4
1.1	Scope and Limitations.	6
2	What's New for Serverless Runtime.	11
2.1	2020 What's New for Serverless Runtime (Archive).	11
3	Concepts.	14
3.1	Functions Programming Model.	17
3.2	Supported Trigger Types.	23
4	Initial Setup.	30
4.1	Create a Serverless Runtime Service Instance.	31
4.2	Create a Serverless Runtime Service Instance Using Cloud Foundry Command Line Interface.	32
4.3	Assign Roles to Users.	32
5	Development.	35
5.1	Create Extensions Using Extension Center.	35
	Add Functions to an Extension.	36
	Add Triggers to an Extension.	37
5.2	Create Extensions Using SAP BTP, serverless runtime CLI.	38
5.3	Create Extensions Using SAP Business Application Studio.	41
	Create Extensions with Serverless Extensions Project Template.	41
	Create Extensions with Yeoman Project Template.	43
	Add Functions or Triggers to an Extension.	45
	Creating Run Configurations for an Extension.	46
6	Using Extension Center.	48
6.1	Manage Extensions.	48
6.2	Register OData Services.	49
6.3	Monitor Errors.	51
6.4	Manage Metadata Validation and Cache Settings.	52
6.5	Manage Extension Credentials.	53
7	Security.	54
7.1	Technical System Landscape.	54
7.2	User Roles.	55
7.3	Authentication and Authorization.	55
7.4	Transport Encryption.	56
7.5	Data Protection and Privacy.	57

8	Monitoring and Troubleshooting	58
----------	---	-----------

1 What Is Serverless Runtime?

Develop extensions using a serverless architecture.

SAP BTP, serverless runtime lets you build, run, and manage serverless applications that extend your digital core and can react to the latest business changes.

You get the following benefits when you go serverless:

- As the cloud provider is responsible for auto-scaling, managing, and operating the infrastructure, you're left with the time and freedom to concentrate on building the application logic.
- Serverless architectures do comprise of servers, but these servers are implemented and scaled by the cloud provider.
- The less in serverless means less cost for you, with pay-per-use there's less complexity, less time spent on operations, and less dependency on infrastructure.

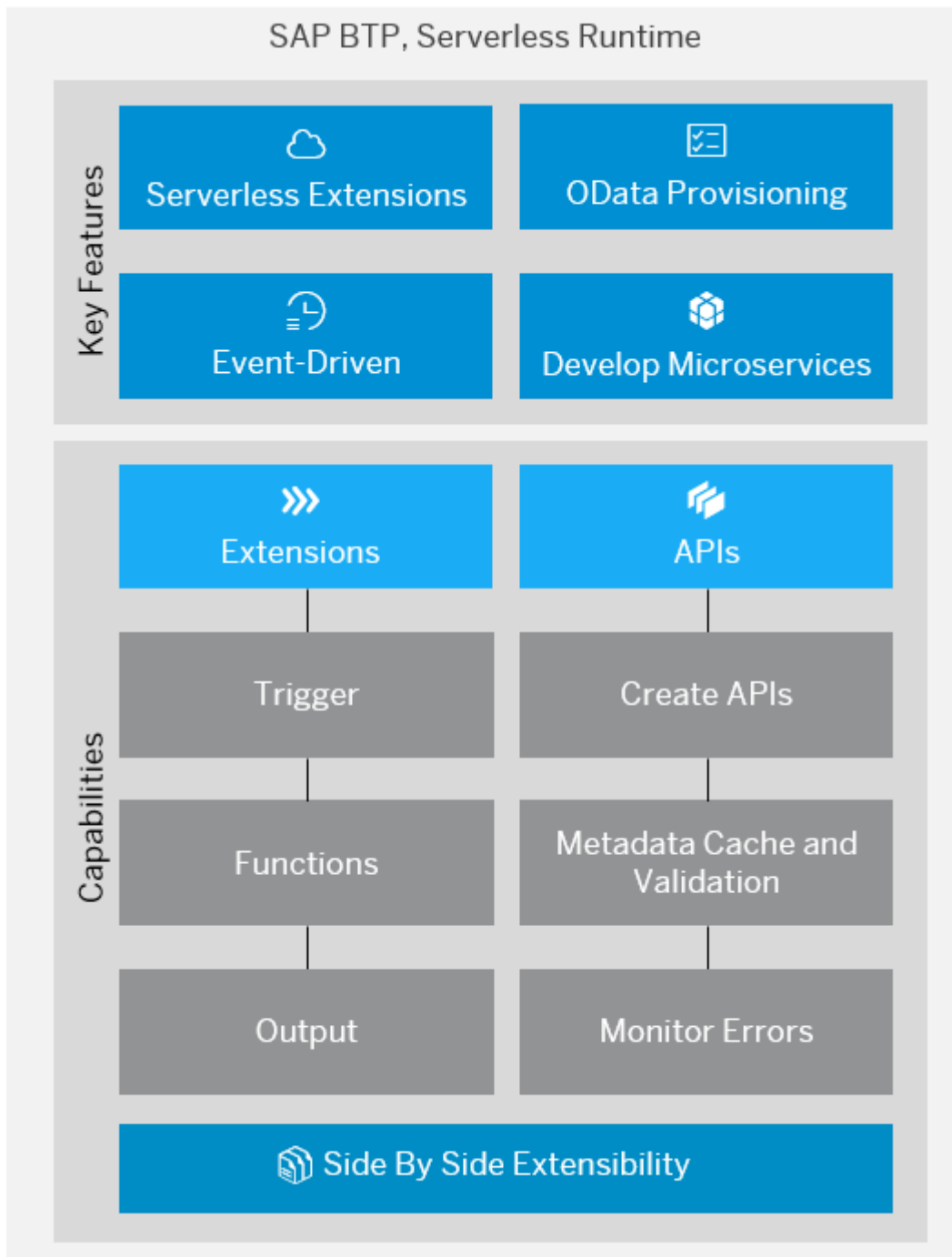
Environment

This service runs in the Cloud Foundry environment.

Features

Holistic user experience	The Extension Center business application provides a full-fledged user interface to build and manage your serverless extensions.
Fully-managed environment	Run your cloud-native extensions in a fully-managed, serverless environment decoupled from your digital core.
Extend business logic	Create serverless functions that can be triggered on demand as defined in your extension.
Connected systems	Connect your SAP systems to access highly available, scalable, and resilient OData APIs.

Overview



Tools

Tools	Description
SAP BTP Cockpit	The central point for managing all activities associated with your subaccount.
Extension Center	A web user interface that allows you to create and run your extensions.
SAP BTP, serverless runtime CLI	A command-line interface tool that allows you create extensions and deploy it to the service.
Serverless Runtime Development Tools	An additional extension on SAP Business Application Studio that allows you create extensions and deploy it to the service.

Scope and Limitations

For information on technical limitations, see [Scope and Limitations \[page 6\]](#).

Related Information

[Concepts \[page 14\]](#)

[Using Extension Center \[page 48\]](#)

1.1 Scope and Limitations

Find information on the service availability and technical limitations for SAP BTP, serverless runtime:

Regional Availability

Get an overview on the availability of Serverless Runtime according to region, infrastructure provider, and release status in the [Service Catalog](#) under [Service Plan](#). The extensions capability is available only in the following regions:

Regional Availability for Extensions

Region & Region Name	Technical Key
ap21 Singapore	cf-ap21
eu10 Europe (Frankfurt)	cf-eu10
eu20 Europe (Netherlands)	cf-eu20
jp20 Japan (Tokyo)	cf-jp20
us10 US East (VA)	cf-us10
us20 US West (WA)	cf-us20
us21 US East (VA)	cf-us21
ap10 Australia (Sydney)	cf-ap10
ap11 Singapore	cf-ap11
ca10 Canada (Montreal)	cf-ca10
br10 Brazil (São Paulo)	cf-br10
jp10 Japan (Tokyo)	cf-jp10

Limitations

General

The maximum number of service instances allowed for a subaccount is 1.

Extensions

Allowed Limits

- The maximum number of extensions per service instance is 5.
- The maximum number of functions per project is 5.
- The maximum number of HTTP triggers per project is 10.
- The maximum number of timer triggers per project is 10.
- The maximum number of AMQP triggers per project is 2.
- The maximum number of secrets per project is 10.
- The maximum number of config maps per project is 10.
- The maximum size of secrets and config maps per project is 1 MB.
- The maximum size of the source code for a project is 50 MB.
- The default value for reconcile timeout is 30 secs.
- The inbound rate limit is 100 requests per second.
- The memory limit for a function is 512Mi.

APIs

The following tables contain the features that OData APIs registered through SAP Business Suite support:

Legend

Notation	Definition
X	Not Supported
0	Not Available
1	Supported without batch calls
2	Supported with batch calls and normal flow

Feature Matrix for OData APIs created using SAP Business Suite

Features	OData (V2)
Query	2
Read	2
Read with property (\$value)	0
Navigation	2
Multilevel Navigation	2
Expand	2
Expand with Navigation	2
Create with Navigation	0
Filter Logical operators	2
Filter Arithmetic operators	0
Filter Binary operator	1

Features	OData (V2)
Filter with String function	2
Filter with grouping ()	X
Filter with Datetime	2
\$top \$skip	2
\$orderby	2
\$select	2
\$inlinecount	2
\$count	2
\$count=true	X
Media support	2
E-tag	2
Function Import (bound)	X
Function import (unbound)	2
Action import (bound)	X
Action import (unbound)	X
Format type	2
Post	2
Deep Insert	2
Put	2
Patch	2
Delete	2
\$apply with groupby	X
\$apply with aggregate	X
\$apply with compute	X
\$apply with filter	X
Cross Service Navigation	X
Search	X
Create with Non-Unicode characters	2

- \$link and custom query options are not supported.
- Concurrency control using ETags not supported only for properties of type TIMESTAMPL. If properties of type TIMESTAMPL are set as Etag properties, then the if-match header is ignored for PUT or DELETE requests. This means that irrespective of the Etag value set in the header, the corresponding operation (PUT or DELETE) goes through, which is not an expected behaviour.
- Push notifications scenarios are not supported.
- Soft state feature is not supported.
- Rule based routing is not supported.

- A single HTTP request and its response must not exceed a total size of 50 MB.
- It is recommended that the backend processing time doesn't exceed 5 min for the corresponding OData request to be processed as expected.

2 What's New for Serverless Runtime

2021

2.1 2020 What's New for Serverless Runtime (Archive)

2020

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Server- less Run- time	Exten- sion Suite - Devel- opment Effi- ciency	Cloud Foun- dry	AMQP and Clou- dE- vents Trigger	You can now add trigger types - AMQP and CloudEvents when you create an extension using SAP Business Application Studio. See Supported Trigger Types .	New	2020-10-08
Server- less Run- time	Exten- sion Suite - Devel- opment Effi- ciency	Cloud Foun- dry	Add Au- thenti- cation for HTTP Trigger	You can now add a JWT authentication for an HTTP trigger through the SAP BTP, serverless runtime CLI. See Supported Trigger Types .	Chang ed	2020-10-08
Server- less Run- time	Exten- sion Suite - Devel- opment Effi- ciency	Cloud Foun- dry	New Version of SAP BTP, serverl ess runtim e CLI	SAP BTP, serverless runtime CLI 2.0.6 is now available for down- load. We recommend that you update to this version of the CLI that has an improved project validation, new commands and so on. See Create Extensions Using SAP Cloud Platform Serverless Run- time CLI .	Chang ed	2020-10-08

Technical Component	Capability	Environment	Title	Description	Type	Available as of
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	Service Re-named	As of September 10, 2020, SAP Cloud Platform Extension Factory, serverless runtime has been renamed to SAP Cloud Platform Serverless Runtime, in short, Serverless Runtime. All delivered functionality remains the same.	Changed	2020-09-10
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	Add a Function or Trigger	You can now add a function or trigger to an existing project through the command palette. See Add Functions or Triggers to an Extension .	New	2020-08-06
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	Project Creation	When you create an extension, a test folder is added to the project folder automatically. It can be used while executing run configurations. See <ul style="list-style-type: none"> Create Extensions with Serverless Extensions Project Template. Create Extensions with Yeoman Project Template. 	Changed	2020-08-06
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	Run Configurations	You can now run your extensions from the Run Configurations view in SAP Business Application Studio and bind it to a Cloud Foundry service. See Creating Run Configurations for an Extension .	New	2020-08-06
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	New Version of SAP BTP, serverless runtime CLI	SAP BTP, serverless runtime CLI 2.0.2 is now available for download. We recommend that you update to this version of the CLI that has an improved authentication flow, new commands and so on. See Create Extensions Using SAP Cloud Platform Serverless Runtime CLI .	Changed	2020-06-18

Technical Component	Capability	Environment	Title	Description	Type	Available as of
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	Additional Extension on SAP Business Application Studio	<p>You can now create and deploy cloud extensions using Serverless Runtime Development Tools on SAP Business Application Studio.</p> <p>See Create Extensions Using SAP Business Application Studio.</p>	New	2020-05-24
Serverless Runtime	Extension Suite - Development Efficiency	Cloud Foundry	Service Instance	<p>When you create a service instance, you must provide a JSON file with the features that you want to enable with the service.</p> <p>See</p> <ul style="list-style-type: none"> • Create an Serverless Runtime Service Instance. • Create an Serverless Runtime Service Instance Using Cloud Foundry Command Line Interface. 	Changed	2020-02-28

3 Concepts

SAP BTP, serverless runtime makes it simpler to run solutions using your business logic. The service provides the capability of building extensions and managing APIs by supporting the following concepts, expand each section for more information:

Extensions

An extension allows you to write loosely coupled business logic that is quick to build and scales automatically. The functions that represent your business logic processes, transform, and store data on the cloud. You can also use functions to receive and respond to events across the SAP BTP and external event sources to validate data, process files, and send notifications. Along with these functions, you can create triggers that invoke the function, for example, a time-based trigger can schedule recurring activities such as reporting and cleanup jobs.

An extension has a well-defined structure that is described in the following table. You can customize the files in an extension after creation.

File or Folder Name	Description
faas.json	This file is created under the root folder of the extension and contains all the functions, triggers, secrets, and configurations used in the extension.
lib	This folder contains the source code for all the functions in the extension.
data	This folder defines all the secret configurations in the extension.
deploy	This folder contains the value for the properties of the secret to be used during deployment. For example, the values of different properties of a secret can be changed during deployment. It allows you to edit the values instead of changing the content of data folder during each deployment.

What Does a faas.json File Contain?

An extension has a faas.json file. The file contains information about the extension in a JSON format along with its properties. It's described in the following table:

Property	Description
project	Name of the project
version	Current version of the project
runtime	The runtime used for the function. The supported runtimes are node.js 8 and node.js 10.
Library	This folder contains the function's source code. The value of this property must be maintained as a folder in the project.

Property	Description
secrets	<p>Defines the list of secrets in the project. Use the following format:</p> <pre> ≡, Sample Code { "<secret_name>" : { "source": <path_to_secret's_data_folder> } } </pre> <p>The path must be relative and within the project. The secret's properties are file names and the values of these properties are contained in the files under the folder specified for the source property. When a project is created, a folder named data gets created for defining all secrets, its properties, and values.</p>
functions	<p>Defines all the functions in the project. Each function is defined as a JSON object in the following format:</p> <pre> ≡, Sample Code { "<function_name>" : { "module": <path_to_source_which_contains_function_module>, "handler": <name_of_handler_method_in_module> } } </pre> <p>The path must be relative to the library folder and within the project. See, Functions Programming Model [page 17].</p>

Property	Description
triggers	<p>Defines all the triggers in the project. Each trigger is defined as a JSON object in the following format:</p> <ul style="list-style-type: none"> HTTP trigger - <div> <pre> { "<trigger_name>" : { "type": "HTTP", "function": "<function_name_that_should_be_triggered>" } } </pre> </div> Timer trigger - <div> <pre> { "<trigger_name>" : { "type": "Timer", "function": "<function_name_that_should_be_triggered>", "timezone": "<enter TZ_database_name, for example, Europe/Berlin>", "timerCronExpression": "<cronexpression_schedule>" } } </pre> </div> AMQP trigger - <div> <pre> { "dial": { "uri": "amqp://<local host>" }, "sasl": { "mechanism": "PLAIN", "user": "<myuser>", "password": "<mypassword>" }, "amqp": { "incoming": { "inp01": { "sourceAddress": "<queue:test01>" } } }, "bind": { "rules": [{ "filter": { "incoming": "<inp01>" }, "action": { "function": "<fnc01>", "failure": "accept" } }] } } </pre> </div>

See, [Supported Trigger Types \[page 23\]](#).

APIs

SAP BTP, serverless runtime exposes business data and business logic as OData services on SAP BTP, thus enabling you to run user-centric applications. The service supports service registration for SAP Business Suite. Service registration to register, activate, and maintain SAP Business Suite where connectivity towards data sources is done via back-end enablement.

The service provides access to back-end services that are built based on the concepts of SAP Gateway and ensures it's published through the Cloud. With the service, you can gain access without having to install an on-premise SAP Gateway hub system. The installation of the nonmodifying back-end enablement components of SAP Gateway is sufficient. The services residing in the back end are enabled and published on the Cloud.

The Open Data Protocol (OData) has become the standard protocol for releasing data stored in SAP Business Suite applications. It makes the data available for user-centric consumption on any device. The service caters to the need for a simplified user experience and uses OData services to extract specific data that you can consume through applications.

While the SAP S/4HANA can publish OData natively, classic SAP Business Suite applications rely on SAP Gateway OData services to extract data from back-end systems. SAP Gateway consists of two parts:

- SAP Gateway back-end enablement - SAP Gateway component in SAP Business Suite systems
- SAP Gateway hub (front-end server) - SAP Gateway component in a dedicated SAP Gateway system

The SAP Gateway component in SAP Business Suite back-end systems gathers data from the back-end system. Whereas the SAP Gateway hub component converts the data from an SAP proprietary format to the OData standard. The conversion allows easy data consumption through the user interface.

The service streamlines the way in which SAP Gateway OData services are exposed to SAP BTP. It replaces the need for an SAP Gateway hub (front-end server) in the cloud and reduces the total cost of ownership and shifts the responsibility for maintenance and security tasks from you to the cloud provider.


Related Information

[Functions Programming Model \[page 17\]](#)

[Supported Trigger Types \[page 23\]](#)

3.1 Functions Programming Model

The Functions programming model explains the style of programming you need to adopt for developing functions using the service. Serverless Runtime supports node.js 8 and node.js 10.

A function is part of an extension. When you create a function in an extension using the service, you must define a module and a handler method. The module should be created locally. The handler method must contain parameters that call the function such as event, context, and so on. Also, it can have additional configuration data. The following sections describe how you can write function code using node.js 8 and node.js 10. For further information on how to structure a function project, see [Samples on GitHub](#) .

Note

When you create functions in the user interface, make sure you write the node.js modules entirely in JavaScript without native dependencies.

Handler

Write a handler method as a JavaScript function in a node.js module. It receives the event and context parameters from the runtime. The handler is called for each function invocation. It can return a simple value directly, undefined or a promise to handle the asynchronous execution.

You can export the function directly as shown in the following example:

Sample Code

```
'use strict';
/**
 * @typedef {import("@sap/faas").Faas.Event} Faas.Event
 * @typedef {import("@sap/faas").Faas.Context} Faas.Context
 */
/**
 * @param {Faas.Event} event
 * @param {Faas.Context} context
 * @return {Promise<*>|*}
 */
module.exports = function (event, context) {
  if (event.ce) {
    return event.sendResponseEvent(event.ce);
  } else {
    return event.data;
  }
};
```

You can also create a module that exports one object with multiple handlers, each identified by name. You can assign any name to a single exported handler. Later, you can provide this name in addition to the module file with the function declaration in the faas.json file.

Sample Code

```
module.exports = {
  handler: function (event, context) {
  }
}
```

Context Parameter

The context parameter is a JSON object that contains information corresponding to the runtime and is independent from a single call. Use the following attributes and utility methods for the context parameter:

Attributes	Type	Description	Example
funcName	string	The name of the function	function1
timeoutMS	number	The timeout value to be used while deploying the function. The default value is 180 seconds.	120

Methods	Description
getServiceCredentials(faas-json-alias)	provides service credentials as binary data
getServiceCredentialsString(faas-json-alias)	provides service credentials as text
getServiceCredentialsJSON(faas-json-alias)	provides service credentials as parsed JSON data
getSecretValueStream(name, key)	provides the secret value stream
getSecretValue(name, key)	provides the secret value as binary data
getSecretValueString(name, key)	provides the secret value as text
getSecretValueJSON(name, key)	provides the secret value as parsed JSON data
getSecretValueYAML(name, key)	provides the secret value as parsed YAML data
getConfigValueStream(name, key)	provides the config value stream
getConfigValue(name, key)	provides the config value as binary data
getConfigValueString(name, key)	provides the config value as text
getConfigValueJSON(name, key)	provides the config value as parsed JSON data
getConfigValueYAML(name, key)	provides the config value as parsed YAML data
callFunction(name, content, callback): response	Calls another function by name within the same instance. Here, content.data provides the request payload, content.type provides the request content type. If a callback is provided it's called to handle the end of the call by providing the parameters error and result. Otherwise, a Promise is returned which resolves it with the result. result.data contains the received payload, result.type the received content type.
getFunctionEndpoint(name):string	If you have enabled HTTP API, provide a function name within the same instance and it returns the corresponding HTTP endpoint.

i Note

The object is reused for all function calls within an instance, but attributes can't be changed. Use the prefix `async` when you use node.js 10 as the runtime.

Event Parameter

The event parameter is a JSON object used to process data. It provides information related to a single function call. Use the following attributes and utility methods:

Attributes	Mandatory	Variable Name	Type	Description	Example
auth	If available	type	string	Initial part of the HTTP authorization header	Bearer
		credentials	string	Data corresponding to the type provided	
ce (CloudEvents)	If available	type	string	The type of occurrence. The format can be defined by you and can include information such as the version of the event type	com.github.pull.create
		specVersion	string	The version of the CloudEvents specification that the event uses	0.2
		source	URI-reference	The event producer. Includes information such as the type of the event source, the organization publishing the event, the process that produced the event, and some unique identifiers.	/cloudevents/spec/pull/123
		id	string	The ID of the event	The database commit ID
		time	Timestamp	Timestamp of when the event occurred	1985-04-12T23:20:50.52Z It represents 20 minutes and 50.52 seconds after the 23rd hour of April 12, 1985 in UTC.
		schemaUrl	URI	A link to the schema to which the data attribute adheres	
		contentType	string	The data attribute value	application/XML
data	Yes	data	string, buffer, or object	The payload (HTTP body) of an event that is specific to the event source	
http	If available	http	HTTP response/HTTP request	Provides access to the HTTP request and response	

Methods	Description
<code>decodeJsonWebToken():{ header: object, payload: object, signature: string }</code>	Decodes event credentials as a JSON WebToken (JWT). It returns null if the mechanism isn't 'Bearer' or if the token has no valid JWT structure. It doesn't validate the token signature. The token itself is provided in the field <code>event.auth.credentials</code> .
<code>decodeUserPassword():{ user: string, password: string }</code>	Decodes event credentials as basic authentication data. It returns null if the mechanism isn't 'Basic' or if the credentials don't match the expected structure. It doesn't validate the credentials against any provider, as it is in the function logic already.
<code>setBadRequest()</code>	Sets the event status to bad request. The event won't be processed and the handler can still add data to the response, for example as a hint. In contrast, throwing any error would be treated as an internal error.
<code>setUnauthorized()</code>	Sets the event status to unauthorized. Event won't be processed, handler can still add data to the response, for example as hint. In contrast, throwing any error would be treated as an internal error.
<code>getContentType():string</code>	Provides the received content type. If there are cloud events, it's taken from the event itself. Data is provided accordingly, it means that the object can be in JSON format, string or binary.
<code>setResponseType(string)</code>	Defines the response content type explicitly. Otherwise, the call's accepted type is compared with the return data type and best matches are used. A fallback strategy can be formulated based only on the returned data type.
<code>getResponseStream(content-Type):WritableStream</code>	The response content type is defined and the corresponding stream is returned. It's possible to write data directly to the stream or to pull data from other streams. The handler returns a Promise to indicate the asynchronous end of processing.
<code>sendResponseEvent(ce)</code>	The cloud event is returned as result of the function execution. HTTP triggers return it to the caller whereas AMQP or cloud events trigger can send the event based on its configuration. The source is adjusted.

Note

- If the function returns a simple value without defining the content type, a matching response content type is selected automatically. Also, it's considered if the client sends an HTTP request with the Accept header.
- If your development environment supports comments (JSDoc annotations), add it to enable code proposals for event and context parameter.

Dependencies

Dependencies are JSON objects that contain node.js modules used in package.json file. You can add only development dependencies, particularly to `@sap/faas: ">=0.7.3"` to support, for example, unit tests for functions. The following code snippet shows how dependencies can be defined in the package.json:

Sample Code

package.json

```
"dependencies": {},
"devDependencies": {
  "@sap/faas": ">=0.7.3",
  "jsdoc": "=3.6.2",
  "mocha": "=7.0.0",
  "sinon": "=7.3.2"
},
```

Secrets and Config maps

You can provide additional configuration data such as secrets and config maps in the `faas.json` file. Use a secret to input configuration data into a key value map and bind it to a function. The secret's configuration data is then made available within the function handler and added to the function's file system. A collection of secrets is used to store sensitive information such as credentials, API keys and so on. Add config maps to provide configuration-like settings for storing unencrypted configuration information. Config maps must not be hard-coded. The following example shows how secrets and config maps can be provided in the `faas.json` file:

Sample Code

`faas.json`

```
{
  "project": "example",
  "version": "0.0.1",
  "runtime": "nodejs8",
  "library": "./lib",
  "secrets": {
    "sec1": {
      "source" : "../data/sec1"
    }
  },
  "configs": {
    "cfg1": {
      "source" : "../data/cfg1"
    }
  }
}
```

Note

In addition to the unique object name, for example, `sec1` or `cfg1`, the attribute `source` is required. The source contains the relative path to the data directory.

Service Reference


You can use a service reference as a simpler way to provide credentials of services within SAP BTP to functions or triggers. Use the alias `my-ems` inside your `faas.json` for trigger definitions or within your function code. Also, specific values of the reference can be provided during deployment in the `values.yaml` file. Use the SAP BTP, serverless runtime CLI to transfer the service keys in a secure manner. For example, you can define a service reference to an Event Mesh service instance. The following example shows a service reference defined in the `faas.json` file:

Sample Code

`faas.json`

```
{
  "services": {
    "my-ems": {
      "type": "enterprise-messaging",
      "instance": "01234567-0123-0123-0123-0123456789ab",
      "key": "myservicekey"
    }
  }
}
```

Functions Calling Other Functions

A function can call another function. Functions that are deployed to the same service instance can call each other. You can use the context parameter utility method `callFunction(name, content, callback)` to do it. Functions that are deployed to different service instances are isolated by network policies such as tenant isolation, see [sample on GitHub](#) .

3.2 Supported Trigger Types

The service allows you to define a trigger that invokes the function during runtime.

The following trigger types are supported by the service:

- HTTP: Executes the function on an HTTP request. An HTTP trigger exposes your code to the public internet.
- Timer: Executes the function based on a schedule. The schedule can be written using a cron expression.
- AMQP: Executes the function based on incoming links, outgoing links, and connection rules to the Event Mesh service using the AMQP 1.0 over WebSocket messaging protocol.
- CloudEvents: Executes the function based on the rule defined in the subscribed CloudEvents specification.

How can you configure a trigger?

Each trigger must be assigned a trigger name, along with the name of the function that must be invoked. The following sections describe how you can configure each trigger type:

HTTP Trigger

Specify the function that the trigger invokes as per the HTTP request. There's a one-to-one relationship between a function and an HTTP trigger. If you're configuring multiple functions, specify an HTTP trigger for each function separately.

The following example shows the `faas.json` file for an extension with an HTTP trigger:

❖ Example

```
{
  "project": "http-trigger",
  "version": "0.0.1",
  "runtime": "nodejs8",
  "library": "./lib",
  "secrets": {},
  "configs": {},
  "functions": {
    "http-trigger-func": {
      "module": "module.js",
      "handler": "handler",
      "secrets": [],
      "configs": []
    }
  },
  "triggers": {
    "httptrigger1": [
      {
        "type": "HTTP",
        "function": "http-trigger-func"
      }
    ]
  }
}
```

```
}
```

You can use JSON Web Token (JWT) authentication when you create an HTTP trigger through the SAP BTP, serverless runtime CLI. Do the following to authenticate the trigger:

- Create a service instance of the Authorization and Trust Management service for the application service plan. See
 - [Create a Service Instance from the xsuaa Service](#)
 - [Create Service Instances Using the Cockpit](#). Select the service Authorization & Trust Management in Service Marketplace, application service plan, and provide the `xs-security.json` in the parameters section.
- Obtain the client credentials for authentication using the service key and store it in your `values.yaml` file. See [Create Service Keys Using the Cloud Foundry Command Line Interface](#).
- Get a service reference for xsuaa. See [Create Extensions Using SAP BTP, serverless runtime CLI \[page 38\]](#). Choose the xsuaa instance that you want to authenticate.

The following example shows the `faas.json` file for an extension with an HTTP trigger that has authentication:

❖ Example

```
{
  "project": "httptrigger-jwt-auth",
  "version": "0.0.1",
  "runtime": "nodejs10",
  "library": "./lib",
  "services": {
    "xsuaa-srv": {}
  },
  "functions": {
    "hello-world": {
      "module": "call-fn.js"
    },
    "hello-world": {
      "module": "index.js"
    }
  },
  "triggers": {
    "jwt-auth": {
      "type": "HTTP",
      "function": "hello-world",
      "auth": {
        "type": "xsuaa",
        "service": "xsuaa-srv"
      }
    }
  },
}
```

See [Code Samples on GitHub](#) for more information.

Timer Trigger

Specify a cron expression that defines the intervals at which the trigger invokes the function. A cron expression is a string with subexpressions that describes the time schedule. The subexpressions are separated by a space. It contains the allowed values and special characters mentioned in the following table:

Allowed Values for Cron Expression

Name	Allowed Values	Allowed Special Characters	Mandatory
Seconds	0–59	, - * /	No
Minutes	0–59	, - * /	Yes
Hours	0–23	, - * /	Yes
Month	0–11 or JAN–DEC	, - * /	Yes
Day - Month	1–31	, - * ? / L W C	Yes
Day - Week	1–7 or SUN–SAT	, - * ? / L C #	Yes

❖ Example

15 8 * * * schedules the trigger each day at 08:15, Berlin time zone.

i Note

You can also provide a simple cron expression that contains hours, minutes, or seconds alone.

The following example shows the faas.json file for an extension with a timer trigger:

❖ Example

```
{
  "project": "timer-trigger",
  "version": "0.0.1",
  "runtime": "nodejs8",
  "library": "./lib",
  "secrets": {},
  "configs": {},
  "functions": {
    "timer-trigger-func": {
      "module": "module.js",
      "handler": "handler",
      "secrets": [],
      "configs": []
    }
  },
  "triggers": {
    "timertrigger1": [
      {
        "type": "Timer",
        "function": "timer-trigger-func",
        "schedule": "5s",
        "timezone": ""
      }
    ]
  }
}
```

AMQP Trigger

Specify the incoming links, outgoing links (optional), service reference, and the connection rules that invoke the function. Use this trigger to connect to any service endpoint that supports the AMQP 1.0 over WebSocket messaging protocol. The following example shows the faas.json file for an extension with an AMQP trigger:

❖ Example

```
{
```

```

    "project": "amqp-trigger",
    "version": "0.0.1",
    "runtime": "nodejs8",
    "library": "./lib",
    "secrets": {
      "amqp-trigger": {
        "source": "../data/folder/secret",
      }
    },
    "configs": {
      "amqp-trigger": {
        "source": "../data/folder/configs",
      }
    },
    "functions": {
      "amqp-trigger": {
        "module": "module.js"
      }
    },
    "triggers": {
      "amqp-trigger": {
        "type": "AMQP",
        "service": "em-amqp",
        "config": "amqp-trigger"
      }
    },
    "services": {
      "em-amqp": {
        "type": "enterprise-messaging",
        "instance": "em-serviceinstance",
        "key": "em-service-key"
      }
    }
  },
}

```

→ Tip

When you provide the service reference (type, instance, and key) for a service running on SAP BTP in the Cloud Foundry environment; the service reference is retained in the subaccount to facilitate simpler access and easier deployment. The value for instance is a UUID after final value replacement. For information on how to create a service key, see [creating service keys](#).

Based on the provided rules the trigger calls one selected function for each incoming message. For each message, the trigger selects the function based on following connection rules:

- the link name,
- message property `to`,
- message property `subject`,
- cloud event source
- cloud event type

The connection rules define the function that must be called, error handling and the response. You can provide filter conditions that contain the cloud event source and its type. These filter conditions must be used to refine the connection rules. The following list describes the behaviour of rules and filter criteria:

- A rule matches if all the defined filter criteria match the corresponding message property or the content.
- If there are multiple messages, the first rule that matches is taken.
- If no rule matches, the messages are accepted without further processing.
- A rule without any filter values always matches.

The following example shows how connection rules can be defined:

❁ Example

```
{
  "dial": {
    "uri": "enterprisemessaging/amqp"
  },
  "sasl": {
    "mechanism": "PLAIN",
    "user": "myuser",
    "password": "mypassword"
  },
  "amqp": {
    "incoming": {
      "inp01": { "sourceAddress": "queue:que01" }
    }
  },
  "bind": {
    "rules": [
      {
        "filter": { "incoming": "inp01" },
        "action": { "function": "fnc01", "failure": "accept" }
      }
    ]
  }
}
```

Each rule allows an explicit decision whether to `accept`, to `reject`, or to `release` the message in case of function failure. The default value is set to `accept`. A message can transport AMQP values as a payload instead of binary data. In this case, the trigger calls a function, encodes the value as a JSON string, and defines the content-type as an `application/json`.

You can use this trigger type to create a simple messaging application. Apart from message consumption, you can also reply to messages. If your request is successful with any code 2XX, except 204 (No Content) and 208 (Already Reported), depending on the configuration the response data is sent as a reply through a new message. The following points must be considered:

- To reply to a message, the target must be configured as `rule.replyTo`.
- If the incoming message contains `message.properties.replyTo` then, this address is preferred over `rule.replyTo`.
- If the request fails multiple times or the response does not indicate success then, the message can be forwarded. This target must be configured with `rule.errorTo`.

i Note

When you create an AMQP trigger in SAP Business Application Studio:

- The source address and target address for Event Mesh must follow the pattern `queue:<queue name>`. For example, `queue:q1` or `topic:t1`.
- At least one incoming link must be provided and multiple incoming links can be added during creation. You can add an outgoing link if required.
- In the Trigger Rule section, provide information on the function that the trigger must invoke. When you create the trigger for the first time you can only add one trigger for a function. You can change this by editing the `faas.json` after creation.
- In the Binding Confirmation section, you can select if you want to use an existing service reference to the Event Mesh service or a secret. A secret allows you to manually enter service credentials like client

ID, client secret, AMQP URL, or token endpoint obtained from the service key directly. This automatically creates a deploy folder with the values.yaml file. If you want to add this information later; ensure the information is added manually before deployment in the values.yaml file and run all the deployment commands.

CloudEvents Trigger

Specify a rule that allows you to subscribe to cloud events to invoke the function. It works similar to an AMQP trigger, but doesn't require secret or config map. Instead, it's fully defined within faas.json. A configuration to the Event Mesh service is done automatically along with project deployment. During deployment, the referenced service instance is called to create a queue or queue subscription according to the provided rules. For one message, only one rule, the first one that matches is applied.

The following example shows the faas.json file for an extension with a CloudEvents trigger:

❁ Example

```
{
  "functions": {
    "ce-trigger": {
      "module": "index.js",
      "services": []
    }
  },
  "triggers": {
    "my-ce": {
      "type": "CloudEvents",
      "service": "my-ems",
      "rules": [
        {
          "ce-source": "",
          "ce-type": "com.sap.cloudevents.abc",
          "function": "ce-trigger",
          "failure": "accept"
        },
        {
          "ce-source": "",
          "ce-type": "com.sap.cloudevents.xyz",
          "function": "ce-trigger",
          "failure": "accept"
        },
        {
          "ce-source": "",
          "ce-type": "com.sap.cloudevents.ijk",
          "function": "ce-trigger",
          "failure": "accept"
        }
      ]
    }
  }
}
```

i Note

When you create a CloudEvents trigger in SAP Business Application Studio:

- The event source must follow the pattern /<region>/<application>[/<instance>]. For example, /us2/sap.grc.ap/EC130CAC or /eu10/sap.billing.sb.
- The event type must follow the pattern shown in the examples. Example - sap.dsc.FreightOrder.Arrived.v1 or sap.odm.workforce.WorkforcePerson.EmailsAdded.v1.

- You can provide multiple rules for a trigger.
- When you create the trigger for the first time you can only add one trigger for a function. You can change this by editing the faas.json after creation.
- In the Binding Information section, you can provide the service reference to the Event Mesh service. This automatically creates a deploy folder with the values.yaml file. If you want to add this information later; ensure the information is added manually before deployment in the values.yaml file and run all the deployment commands.

Related Information

[What Is SAP Event Mesh?](#)

[Functions Programming Model \[page 17\]](#)

[Development \[page 35\]](#)

4 Initial Setup

After you purchase the Serverless Runtime default plan, you must perform the following steps in the SAP BTP cockpit to start using the service.

Prerequisites

- You have a customer account with SAP BTP and an Serverless Runtime default service plan, see [getting started with a customer account: workflow in the cloud foundry environment](#).
- You have a global account that has the entitlement to use Serverless Runtime, see [get a paid global account \[AWS, Azure, or GCP\]](#).
- You've created a subaccount. See [create a subaccount in the cloud foundry environment](#).
- You've created a space within the subaccount in which Cloud Foundry is enabled, see [managing orgs and spaces using the cockpit](#).

Procedure

1. In the SAP BTP cockpit, navigate to your global account and assign an entitlement to the default service plan for Serverless Runtime to your subaccount.
 - a. Open your global account and choose ► [Entitlements](#) ► [Subaccount Assignments](#) ►.
 - b. In the dropdown list, select your subaccount and choose [Go](#).
 - c. Choose [Configure Entitlements](#) and then, [Add Service Plans](#).
 - d. In the [Subaccount Entitlements](#) dialog box, select the Serverless Runtime service.
 - e. In the [Service Details: Serverless Runtime](#) window, select the default service plan.
 - f. Choose [Add 1 Service Plans](#) to add this entitlement for the Serverless Runtime service for your subaccount.
 - g. Choose [Save](#).
2. Create a service instance for Serverless Runtime using either the cockpit or cf CLI, see
 - [Create a Serverless Runtime Service Instance \[page 31\]](#).
 - [Create a Serverless Runtime Service Instance Using Cloud Foundry Command Line Interface \[page 32\]](#).
3. [Assign Roles to Users \[page 32\]](#).

Related Information

[Configure Entitlements and Quotas for Subaccounts](#)

[Subscribe to Multitenant Business Applications in the Cloud Foundry Environment Using the Cockpit](#)

4.1 Create a Serverless Runtime Service Instance

Use the SAP BTP cockpit to create a service instance for Serverless Runtime.

Prerequisites

- You have the administrator role for your global account.
- Access to a Cloud Foundry space.

Context

To use the service, you need to create a service instance in the cockpit.

Procedure

1. Open the cockpit.
2. Navigate to your subaccount. Expand Cloud Foundry and choose [Spaces](#).
3. Open your space. Expand Services and choose [Instances](#)
4. Choose [Create](#).
5. Select Serverless Runtime as the service and the default plan.
6. Enter an instance name and choose [Next](#).
7. Specify the following parameters using a JSON file. To create extensions, you must set the parameter extensions as true. Similarly, to provision OData services, you must set the parameter apis as true.

Sample Code

```
{
  "version": "1.1.0",
  "capabilities": {
    "extensions": true,
    "apis": true
  }
}
```

8. Review the details you've provided and choose [Create](#).

If you want to delete an Serverless Runtime service instance, choose [Delete](#) under Actions.

4.2 Create a Serverless Runtime Service Instance Using Cloud Foundry Command Line Interface

Create an Serverless Runtime service instance using the Cloud Foundry Command Line Interface (cf CLI).

Prerequisites

- You have the administrator role for your global account.
- Access to a Cloud Foundry space.

Procedure

1. [Log on to the Cloud Foundry Environment using the Command Line Interface.](#)
2. Select your org, then the space.
3. Enter `cf marketplace` to verify the availability of the Serverless Runtime service in the Cloud Foundry marketplace.
4. Enter `cf create-service <service plan> <service instance name> -c <features you want to enable>` to create a new service instance.

Sample Code

```
cf create-service xfs-runtime default extension -c {  
  "version": "1.1.0",  
  "capabilities": {  
    "extensions": true,  
    "apis": true  
  }  
}
```

4.3 Assign Roles to Users

The service provides standard roles for typical user profiles. You can configure application roles and then assign users to these roles using the SAP BTP cockpit.

Prerequisites

- You have the administrator role for your global account.

- You've created a service instance.
- You're using one or both of the following trust configurations:
 - Default trust configuration (SAP ID service), see [Default Identity Federation with SAP ID Service in the Cloud Foundry Environment](#).
 - Custom trust configuration (Identity Authentication service or any SAML 2.0 identity provider), see [Establish Trust and Federation with UAA Using Any SAML Identity Provider](#).
- (Optional) [Create a new user group](#). You can also assign roles to individual platform users.
- (Optional) [Create a role collection](#). Use a role collection if you want to add multiple roles to a user or user group.

Context

You can assign roles to users based on the type of trust configuration you've enabled in SAP BTP. The following options are available:

- Directly assign role collections to users.
- Map role collections to user groups defined in the identity provider. You initially maintain the mapping between user groups and role collections once in SAP BTP, and maintain group memberships of users in the identity provider.

If you're using the default trust configuration with SAP ID service, you can directly assign users to role collections. However, if you're using a custom trust configuration, for example, with the Identity Authentication service, you can use both options.

The roles that can be assigned include the following:

- FunctionsRead (developer) - View extensions, functions, triggers, config maps, and secrets.
- FunctionsManage (administrator) - Create, edit, and delete extensions, functions, triggers, config maps, and secrets along with the FunctionsRead role.
- ODPAPIAccess - Provides access to the service document from the Services tab in the user interface.
- APIFullAccess - Provides runtime access to the registered OData services.
- ODPMange and APIMange - View and register services from the Services tab in the user interface.

Procedure

1. Navigate to your subaccount.
2. In the left pane, choose [Subscriptions](#).
3. Choose the [Extension Center](#) tile then, [Manage Roles](#) to view, create, and modify the application roles.
4. Go back to your subaccount and choose [Security > Role Collections](#) in the left pane to add the standard roles provided with the service to a role collection.
5. Choose [Security > Trust Configuration](#).
6. Choose your active trust configuration.
7. Select one of the following methods to proceed:

Trust Configuration Used	Steps
Default trust configuration (SAP ID service)	<ol style="list-style-type: none"> 1. Choose Role Collection Assignment. 2. Enter the business user's name, for example <code>john.doe@example.com</code>. 3. Enter the user name, for example <code>john.doe@example.com</code>. <div> <p>→ Tip</p> <p>If the user identifier you've entered has never logged on to an application in this subaccount, SAP BTP can't automatically verify the user name. To avoid mistakes, you must check and confirm that the user name is correct.</p> </div> <ol style="list-style-type: none"> 4. To see the role collections that are currently assigned to this user, choose Show Assignments. 5. To assign a role collection, choose Assign Role Collection. 6. Choose the name of the role collection you want to assign. 7. Save your changes.
Custom trust configuration (Identity Authentication service or any SAML 2.0 identity provider)	<p>You can perform the preceding steps or</p> <ol style="list-style-type: none"> 1. Choose Role Collection Mappings. 2. Choose New Role Collection Mapping. 3. Choose the role collection you want to map and enter the name of the user group in the <i>Value</i> field. <div> <p>❖ Example</p> <p>In the Identity Authentication service, you can find user groups in the Administration console of your Identity Authentication service tenant under Users & Authorizations > User Groups. Open the Administration console of, for example, the Identity Authentication service using <code>https://<Identity_Authentication_tenant>.accounts.ondemand.com/admin</code>.</p> </div> <ol style="list-style-type: none"> 4. Save your changes.

Related Information

[Assign role collection to users/user groups](#)

[Add roles to role collections](#)

5 Development

Use the following methods to create and deploy an extension:

- Extension Center allows you to create extensions through its user interface. See [Create extensions using Extension Center](#).
- SAP BTP, serverless runtime CLI is a command-line interface tool that allows you to create extensions. See [Create extensions using SAP BTP, serverless runtime CLI](#).
- SAP Business Application Studio provides an additional extension to create extensions. See [Create extensions using SAP Business Application Studio](#).

For information on how to create functions using node.js, see [Samples on GitHub](#) ➡

Related Information

[Concepts](#)

[Initial setup](#)

[Blog Series: Writing Function-as-a-Service: Overview of Blogs](#) ➡

5.1 Create Extensions Using Extension Center

An extension contains functions that define your business logic and triggers that invoke the function.

Prerequisites

- You've completed the [Initial setup](#) [page 30].
- You have the FunctionsManage role assigned. See [Assign roles to users](#) [page 32].

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.
3. Choose [New Extension](#).
4. In the [Create Extension](#) dialog box, select a template and choose [Step 2](#).

5. Enter an extension name and select the runtime you want to use.
6. If you chose a [Template with a function and a trigger](#), perform these additional steps:
 - a. Choose [Step 3](#).
 - b. Enter a function name and choose [Step 4](#).
 - c. Enter a trigger name, select the trigger type, and choose [Step 5](#).
 - d. Review the information you've provided.

i Note

HTTP, Timer, and AMQP triggers are supported.

7. Choose [Create](#).

Related Information

[Concepts](#)

[Functions programming model](#)

[Supported trigger types](#)

5.1.1 Add Functions to an Extension

After creating an extension, you can add functions to enhance your business logic.

Prerequisites


- You've created an extension. See [Create extensions using the extension center \[page 35\]](#).
- You have the FunctionsManage role assigned. See [Assign roles to users \[page 32\]](#).

Context

For information on how to develop functions, see [Functions programming model \[page 17\]](#).

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.

3. Open an extension that you've created.
4. In the *Functions* tab, choose *Add Function*.
5. Enter a function name.
6. Provide a module name and a handler name.
7. Choose .

i Note

You can add multiple functions and add them at one go in the end.

8. Choose *Add*.

5.1.2 Add Triggers to an Extension

After creating an extension, you can add a trigger that invokes the function during runtime.


Prerequisites





- You've created an extension. See [Create extensions using the extension center \[page 35\]](#).
- You have the FunctionsManage role assigned. See [Assign roles to users \[page 32\]](#).
- If you're adding an AMQP trigger, you've generated a service key for the Event Mesh service, see [Creating service keys](#).

Context

HTTP, Timer, and AMQP triggers are supported by Extension Center, see [Supported trigger types \[page 23\]](#).

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose *Go to Application* in the Subscriptions tab.
3. Open an extension that you've created.
4. In the *Triggers* tab, select the type of trigger you want to add.
5. For HTTP trigger, perform the following steps:
 - a. Enter a trigger name.
 - b. Select the function you want to invoke and choose .

6. For timer trigger, perform the following steps:
 - a. Enter a trigger name.
 - b. Select the function you want to invoke and the schedule type.
 - c. Provide an expression for the schedule and choose .
7. For AMQP trigger, perform the following steps:
 - a. Enter a trigger name and choose [Step 2](#).
 - b. Provide a queue or topic name from which you want to receive messages and choose .
 - c. Choose [Step 3](#), provide a queue or topic name to which you want to send messages (optional), and choose .
 - d. Choose [Step 4](#), provide connection rules for the trigger, and choose .

i Note

Rules define the behavior of the AMQP trigger in accordance to the incoming and outgoing links you've provided, see [Supported trigger types \[page 23\]](#).

- e. Choose [Step 5](#) and provide credentials to the Event Mesh service obtained from the service key.

i Note

You can add multiple triggers and add them at one go in the end.

8. Choose [Add](#).

You can view details of the trigger that you've added by choosing [View Trigger Details](#).

Related Information

[What is Enterprise Messaging](#)

5.2 Create Extensions Using SAP BTP, serverless runtime CLI

The service allows you to use SAP BTP, serverless runtime CLI to create an extension and deploy it to the service.

Prerequisites

- [Download and install the cloud foundry command line interface](#)

- [Download and install the SAP BTP, serverless runtime CLI](#)
- [Create a Serverless Runtime service instance using the cloud foundry command line interface \[page 32\]](#)

Context

You can see the available commands and flags by entering `--help` or `-h`. Each sub command requires the following flags:

Command	Description
faas project delete	Deletes a project
faas project deploy	Deploys a project
faas project download	Downloads currently deployed project
faas project get	Retrieves metadata, status information, and links of the project
faas project list	Lists project names available in the service instance
faas service list	Lists all the registered services
faas service register	Registers a service
faas service delete	Deletes the service reference stored in the subaccount
faas project logs	Retrieves logs of the project
faas project metrics	Retrieves metrics of the project, for example, number of invocations, GB seconds
faas tenant list	Lists the service instances in your subaccount
faas project check	Validates a project
faas service tidy	Reads the service index and checks whether the referenced service still exists. If it is missing, a cleanup is proposed. Otherwise, the service key gets updated.
faas project run	Starts a local test run and checks devDependencies. Changes are proposed if required.
faas project init values	Initialize a deployment file based on specific secret definitions contained inside the project:

Procedure

1. Open the command-line interface or terminal.
2. Log in to the Cloud Foundry space where your Serverless Runtime service instance is present, see [Log on to the Cloud Foundry environment using the Cloud Foundry command line interface](#).
3. Run the following command to create a service key:

```
cf create-service-key <SERVICE_INSTANCE_NAME> <SERVICE_KEY_NAME>
```

4. Change your directory path to the folder where you've extracted the SAP BTP, serverless runtime CLI using the following command:

```
cd <path to xfsrt-cli folder>
```

→ Tip

Use WinRAR to extract the SAP BTP, serverless runtime CLI executable file. Skip this step if you've stored the path to the folder in your environment variables.

5. Log in to the SAP BTP, serverless runtime CLI using the following command:

```
xfsrt-cli login
```

- A service instance and key in the space from which you logged in are selected. Check the parameters to change the value that is selected by default.
- The credentials from the service key are used to run a `client_credentials` flow for authentication.
- All service instances and service keys in your space are listed for selection when you try to log in.

6. Create an extension in your local system, see:

- [Concepts \[page 14\]](#)
- [Functions programming model \[page 17\]](#)
- [Supported trigger types \[page 23\]](#)

7. (Optional) Perform the following steps if there are no external node.js module dependencies:

- a. Change your directory path to the project folder using the following command:

```
cd <path to project folder>
```

- b. Run `npm install` locally using the following command:

```
npm install
```

8. (Optional) If you want to store a service reference, use the following command:

```
xfsrt-cli service register
```

i Note

Ensure you have entitlements to use the service you want to register. All the services in your space are listed for selection.

9. Deploy your project to the service using the following command:

```
xfsrt-cli faas project deploy -p <path to project folder>
```

→ Remember

- A project must be self-contained during deployment. All dependent libraries must be resolved before deployment.
- The CLI builds an archive of the specified project folder and uploads it to the service.
- The size limit for a project is 50 MB.
- For confidential data, use a deployment file (`values.yaml`) instead of adding the data directly to your project.

Related Information

[Samples on GitHub](#) 

5.3 Create Extensions Using SAP Business Application Studio

Use SAP Business Application Studio to create and deploy your extensions.

SAP Business Application Studio is provided as a business application. After you have created a subaccount in your Cloud Foundry space, subscribe to SAP Business Application Studio, see [Subscribe to SAP Business Application Studio](#). Choose the [Go to Application](#) link to access the user interface.

Related Information

[What is SAP Business Application Studio](#)

[Developing an SAP Fiori application](#)

5.3.1 Create Extensions with Serverless Extensions Project Template

Create an extension that contains your function code using the project template on SAP Business Application Studio.

Prerequisites

- You've subscribed to SAP Business Application Studio.
- You've the `Developer` role assigned, see [Manage authorizations](#).
- In SAP Business Application Studio, you've created a dev space, see [Managing your dev spaces](#). Choose an application type of your choice then, [Serverless Runtime Development Tools](#) under [Additional SAP Extensions](#).
- You've created a service instance and a service key, see
 - [Create a Serverless Runtime service instance](#).
 - [Creating service keys](#)
- You have permissions to deploy to your Cloud Foundry space.

Context

The SAP BTP, serverless runtime CLI is provided as an in-built dependency to make project deployment easier.

Procedure

1. Open SAP Business Application Studio.
2. Navigate to your dev space.
3. Select [Create project from template](#), then [Serverless Extensions Project](#).
4. Provide a name for your project and select the runtime that you want to use.
5. Provide a name for your function and handler, then enter a timeout value, see [Functions programming model \[page 17\]](#).
6. Provide a name for your trigger and select the function it to be invoked. Then, select the trigger type. See [Supported trigger types \[page 23\]](#).
7. Choose [Open in New Workspace](#).
8. (Optional) Make additional changes to your function code in your workspace. See [Add Functions or Triggers to an Extension \[page 45\]](#).

Note

- A test file with .http extension is created in the project under the folder `Test > Integration` for each function.
- When you add more functions to the `faas.json`, a test file is added to the `Integration` folder automatically.
- After [creating a run configuration for the extension \[page 46\]](#), you can test the function locally. Choose the [Run](#) icon in the run configuration that you've created for the extension then, choose [Send Request](#) in the file for the respective function in the `Integration` folder.
- The results are displayed in the side panel of the editor.

9. Open the command palette and enter `CF login` and select `CF: Log in to Cloud Foundry`.
Alternately, choose the message `The organization and space in Cloud Foundry haven't been set.` at the bottom of the page. Then, press `Enter` to confirm your Cloud Foundry endpoint.
10. Enter your login credentials and provide details of the Cloud Foundry organization and space to which the deployment must be made.
11. Choose [Terminal > New Terminal](#).
12. Log in to the SAP BTP, serverless runtime CLI using the following command:

```
xfsrt-cli login
```

13. Select the required service instance and service key in your space from the list.
14. Change the directory to the project folder using the following command:

```
cd <project name>
```

15. Deploy your project to the service using the following command:

```
xfsrt-cli faas project deploy
```

16. Retrieve metadata, status information, and links to the project using the following command:

```
xfsrt-cli faas project get
```

i Note

After you deploy an extension with an HTTP trigger, the trigger URL is displayed.

5.3.2 Create Extensions with Yeoman Project Template

Create an extension that contains your function code using the Yeoman template in SAP Business Application Studio.




Prerequisites

- You've subscribed to SAP Business Application Studio.
- You have the `Developer` role assigned, see [Manage authorizations](#).
- In SAP Business Application Studio, you've created a dev space, see [Managing your dev spaces](#). Choose an application type of your choice then, *Serverless Runtime Development Tools* under *Additional SAP Extensions*.
- You've created a service instance and a service key, see
 - [Create a Serverless Runtime service instance](#).
 - [Creating service keys](#)
- You have permissions to deploy to your Cloud Foundry space.

Context

The Yeoman generator is provided as an in-built dependency to make project creation easier. You can create an extension by answering the questions from the Yeoman generator.

Procedure

1. Open SAP Business Application Studio.
2. Navigate to your dev space and select  [Terminal](#)  [New](#) .
3. In the Terminal that gets opened, enter `yo`.

4. Select `@xfsrt/serverlessextensions`.
5. Enter a project name and select the runtime that you want to use.
6. Provide a name for your function, module and handler method, then enter a timeout value. See [Functions programming model \[page 17\]](#).
7. Select the trigger type you want to use, provide a name for your trigger, and select the function to be invoked. See [Supported trigger types \[page 23\]](#).

The project gets added to your workspace.

8. (Optional) Make additional changes to your function code in your workspace. See [Add Functions or Triggers to an Extension \[page 45\]](#).

i Note

- A test file with `.http` extension is created in the project under the folder `Test > Integration` for each function.
- When you add more functions to the `faas.json`, a test file is added to the Integration folder automatically.
- After [creating a run configuration for the extension \[page 46\]](#), you can test the function locally. Choose the [Run](#) icon in the run configuration that you've created for the extension then, choose [Send Request](#) in the file for the respective function in the Integration folder.
- The results are displayed in the side panel of the editor.

9. Open the command palette and enter `CF login` and select `CF: Log in to Cloud Foundry`.
Alternately, choose the message *The organization and space in Cloud Foundry haven't been set.* at the bottom of the page. Then, press `Enter` to confirm your Cloud Foundry endpoint.
10. Enter your login credentials and provide details of the Cloud Foundry organization and space to which the deployment must be made.
11. Navigate to the Terminal and login to the SAP BTP, serverless runtime CLI using the following command:

```
xfsrt-cli login
```

12. Select the required service instance and service key in your space from the list.
13. Change the directory to the project folder using the following command:

```
cd <project name>
```

14. Deploy your project to the service using the following command:

```
xfsrt-cli faas project deploy
```

15. Retrieve metadata, status information, and links to the project using the following command:

```
xfsrt-cli faas project get
```

i Note

After you deploy an extension with an HTTP trigger, the trigger URL is displayed.

5.3.3 Add Functions or Triggers to an Extension

After creating a project for your extension, you can add more functions or trigger to the project.

Prerequisites

You've created an extension. See

- [Create Extensions with Serverless Extensions Project Template \[page 41\]](#).
- [Create Extensions with Yeoman Project Template \[page 43\]](#).

Procedure

1. Open SAP Business Application Studio.
2. Navigate to your dev space and open the command palette.
3. Enter **Serverless Runtime: Add Module** and select it
4. Based on your requirement, perform one of the following:

To add a function:

- a. Select [Add Function](#) then, choose [Next](#).
- b. Choose [Browse for folder](#) and select an existing extension project to which you want the function to be added.

i Note

The extension project that you select must contain a faas.json.

- c. Choose [Next](#) then, provide details of the function that you want to add.
- d. Choose [Next](#). The function gets added to the respective extension project.

To add a trigger:

- a. Select [Add Trigger](#) then, choose [Next](#).
- b. Choose [Browse for folder](#) and select an existing extension project to which you want the trigger to be added.

i Note

The extension project that you select must contain a faas.json.

- c. Choose [Next](#) then, provide details of the trigger that you want to add.
- d. Choose [Next](#). The trigger gets added to the respective extension project.

5.3.4 Creating Run Configurations for an Extension

Create configuration settings for running your extensions.

Prerequisites

- Your extension was created in an SAP Cloud Business Application dev space. See [Create Extensions Using SAP Business Application Studio \[page 41\]](#).
- Your extension must include the `faas.json` file in your root folder.

Procedure

1. From the left-side menu, choose **Run Configurations** > **Create Configuration**.
2. From the command palette, select the runnable object for which you want to create the configuration. Select the deployment file (if available) and enter a name for the run configuration.

i Note

A run configuration name with the convention `Run <project name>-<faas.json>-<deployment file name>` is shown in the command palette. You can edit this name if necessary.

A configuration tree appears in the **RUN CONFIGURATIONS** view containing the run configurations that were created for the runnable objects. A new configuration is added to your `launch.json` file.

3. In the **Run Configurations** view, you can see the available services as dependencies defined in the `faas.json` file. You can bind or unbind these dependencies to a specific Cloud Foundry service instance as defined in your deployment file.

To bind the dependency to a Cloud Foundry service:

- a. Open the **Run Configurations** view.
- b. Select the required configuration and dependency. Choose the **Bind** icon.

If not already logged in, you must log in to Cloud Foundry to proceed. A list of all available service instances that match the service you've defined are shown in the command palette.

- c. Select the required service.

i Note

After the dependency is bound, information for the service instance and service key is updated in the available or selected deployment file (`values.yaml`). If you want to unbind the service, choose the **Unbind** icon.

4. To run a configuration:
 - a. Select the required configuration and choose the **Run** icon. The Debug Console opens.
 - b. Open the **Debug** view to stop a configuration that is already running.

i Note

Stopping a configuration from the *Debug* view doesn't stop any running tasks.

5. To edit a run configuration:

a. Right-click a configuration to do the following:

- *Configure Environment* - View the binding configuration.
- *Rename* - Provide a new name for the selected run configuration.
- *Show in File* - Open the JSON file containing the set of configuration properties, with the name highlighted.
- *Delete* - Delete the set of configuration properties from the JSON file. You can also select a run relevant configuration and choose *Delete* to delete the run configuration itself.

i Note

- Configuring the environment or renaming the run configuration can change the existing binding configuration. We recommend that you use it only if necessary.
- If you delete the launch configuration, it's removed from the `launch.json` file but the tasks remain.

Related Information

[Creating Run Configurations](#)

6 Using Extension Center

Use the Extension Center business application to manage your extensions.

The Extension Center is provided as a business application. After you've created a service instance for Serverless Runtime and subscribed to the Extension Center. Choose the [Go to Application](#) link in the subscriptions tab to access the user interface.

Related Information

[Create extensions using Extension Center](#)

[Manage extensions](#)

6.1 Manage Extensions

After you've deployed your extension, you can manage your extensions using the Extension Center.

Prerequisites

- You've created an extension. See [Development](#).
- You have the FunctionsRead role assigned. See [Assign roles to users](#).

Context

In the Extension Center, you can view a list of extensions and information about the secrets, config maps, and execution logs for an extension.

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.
3. Open an extension that you've created.

4. The extension code is displayed in an editor. You can do the following in the editor:

Actions

Action	Description
Create File	Add a new file to the extension
Create Folder	Add a new folder to the extension
Delete File or Folder	Select a file or folder you want to delete and choose this action. You can't retrieve a file or folder that you've deleted.
Delete	Delete the extension. You can't retrieve an extension that you've deleted.
Download	Download the extension as a zip file to your local system
Save and Deploy	Save the changes you've made and deploy the extension to the service
Add Functions to an Extension	See, Add functions to an extension .
Add Triggers to an Extension	See, Add triggers to an extension .

Tabs

Tab	Action
Secrets	<ul style="list-style-type: none">View a list of secrets present in the extension.View details for a secret under Actions.
ConfigMaps	<ul style="list-style-type: none">View a list of ConfigMaps present in the extension.View details for a ConfigMap under Actions.
Logs	<ul style="list-style-type: none">Select a function and view the associated execution logs.Refresh the page to retrieve the latest execution logs.

6.2 Register OData Services

You can provision OData services using the service.

Prerequisites

- You've completed the [Initial setup \[page 30\]](#).
- You have the ODPAPIAccess and ODPMange role assigned, see [Assign roles to users \[page 32\]](#).
- [Create an HTTP destination](#) to the service you want to provision. While adding the destination, ensure you choose [New Property](#) and provide an additional property `odc` with value `true`. Destinations are part used for the outbound communication between a cloud application and an SAP system. The destination contains connection details for remote communication of an application. Use the Destinations editor in the cockpit to configure HTTP destinations to connect your Web application to the Internet or consume an on-premise back-end system via HTTP(S).

i Note

The destination URL must be of the SAP Business Suite system where the service implementations are present. The generic URL is `https://<hostname>:<port>/sap/iwbep?sap-client=<client number>`. Hostname, port, and client number depend on the system you're using.

To get the destination URL:

1. In transaction SICF, choose Execute (**F8**) to display the service tree hierarchy.
2. Expand the default host and navigate to the node IWBEF (► [default_host](#) ► [sap](#) ► [iwbeep](#) ►).
3. You can register services by exposing SAP Business Suite data as an OData service on SAP Cloud. In the context menu of the node [iwbeep](#), choose ► [Test Service](#) ► [Allow](#) ►. The URL you get in the address bar of the browser is the destination URL.

Context

When you make a service error tolerant, its behavior changes in the following ways:

- An error is returned only if the data retrieved from all the back-end systems cause an error. If at least one back-end system doesn't cause an error, then a feed is returned.
- The feed contains the data retrieved from the back-end systems that didn't cause an error.
- The data from the back-end systems that caused an error can be requested again using the OData skip token available in feed.

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.
3. Choose ► [Services](#) ► [Register](#) ►.
4. Select a destination you've created and look for the service name.
5. Select the service and choose [Register](#).

Open the [Service Name](#) URL to view details of the registered service.

6. In the service details page, choose [Add](#) to add another back-end system.

i Note

- You can delete a registered service only if the [Status](#) is set to [OFF](#).
- You can't delete a destination that has been set as [Default](#).

7. Choose [Open Service Document](#) to view the service metadata.

Related Information

[Create destinations \(Cockpit\)](#)

[Concepts \[page 14\]](#)

6.3 Monitor Errors

Analyze the root cause for errors and where they originated.

Prerequisites

- You've completed the [Initial setup \[page 30\]](#).
- You have the APIFullAccess and APIManage role assigned, see [Assign roles to users \[page 32\]](#).

Context

You can view monitoring and error data for respective data sources in SAP Business Suite.

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.
3. Expand [Monitoring](#) then, choose [Services](#).
4. When there's an error, you can start troubleshooting under [Error Log Entries](#).
5. Select [From](#) and [To](#) dates. Choose [Search](#).

The errors that occurred during the selected time frame are shown.

6.4 Manage Metadata Validation and Cache Settings

Enable or disable metadata validation for a registered service. You can also allow caching of metadata, which significantly improves performance and cleans up the metadata cache for APIs created with SAP Business Suite as the backend type.

Prerequisites

- You've completed the [Initial setup \[page 30\]](#).
- You have the ODPAPIAccess and ODPManage role assigned, see [Assign roles to users \[page 32\]](#).

Context

Metadata validation is inactive for registered services by default. For example, consider a scenario where you have activated metadata validation. If the ABAP type defined in the back end for a property supports a length greater than the length of the edm type defined in the metadata for the same property. Then, modify requests like PUT or POST result in a metadata validation error. To avoid such errors, you can deactivate metadata validation. In this scenario, if metadata validation is inactive but, the back end validates the input, a modify request can still result in an error. Hence, it's recommended to have the same length for the edm properties and the corresponding back end (ABAP) properties.

Metadata cache is active by default. If you have activated metadata cache, then by default, [Destination-based metadata caching](#) is activated. Enabling this setting fetches and caches the service metadata from a default or specific destination on the first request. All subsequent calls to the service for this destination use the cached metadata.

Disable destination-based caching only if the metadata for a registered service is the same across all the destinations. It's applicable for all the registered services. Disabling this setting fetches and caches the service metadata from any available destination on the first request. All subsequent calls to the service use the cached metadata. Disabling this setting becomes useful when you have users who can't access all destinations, especially those destinations that are set as the default or the metadata default. Enabling or disabling this setting clears any existing cache first.

Metadata cache clean up requires the metadata cache to be active.

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.
3. Expand [Configurations](#) then, choose [Services](#).
4. Use any of the following options based on your requirement:

User Interface Element	Action
Metadata Validation	Use the toggle button to select whether to keep the metadata validation inactive or to activate it.
Cache Metadata	Use the toggle button to activate or deactivate metadata cache. When you deactivate metadata cache existing metadata is cleared and the latest metadata is retrieved from the destination.
Destination Based Metadata Caching	Use the toggle button to select whether to keep the destination-based metadata caching active or to deactivate it.
Clear under Actions	Use to clear metadata cache for the corresponding API.
Clear All	Use to clear metadata cache for the all APIs listed in the table.
Refresh	Use to refresh metadata cache for all destinations.

6.5 Manage Extension Credentials

You can store credentials used for authentication in your extensions.

Prerequisites

- You've created an extension. See [Development](#).
- You have the FunctionsRead and FunctionsManage role assigned. See [Assign roles to users](#).

Procedure

1. In the cockpit, navigate to your subaccount.
2. Choose [Go to Application](#) in the Subscriptions tab.
3. Expand [Configurations](#) then, choose [Extensions](#).
4. Choose [New Credentials](#).
5. Select the service to which your extension needs to connect.
6. Provide the service instance name, service key name and the service key credentials (client ID and client secret) for the service you've selected.
7. Choose [Create](#).
8. Under [Actions](#), choose [Delete](#) to delete the corresponding extension credential.

7 Security

SAP BTP, serverless runtime security information describes the policies, technologies, and controls that are applicable specifically to the service to protect data, applications, and their associated infrastructure within SAP BTP.

Security aspects include the following:

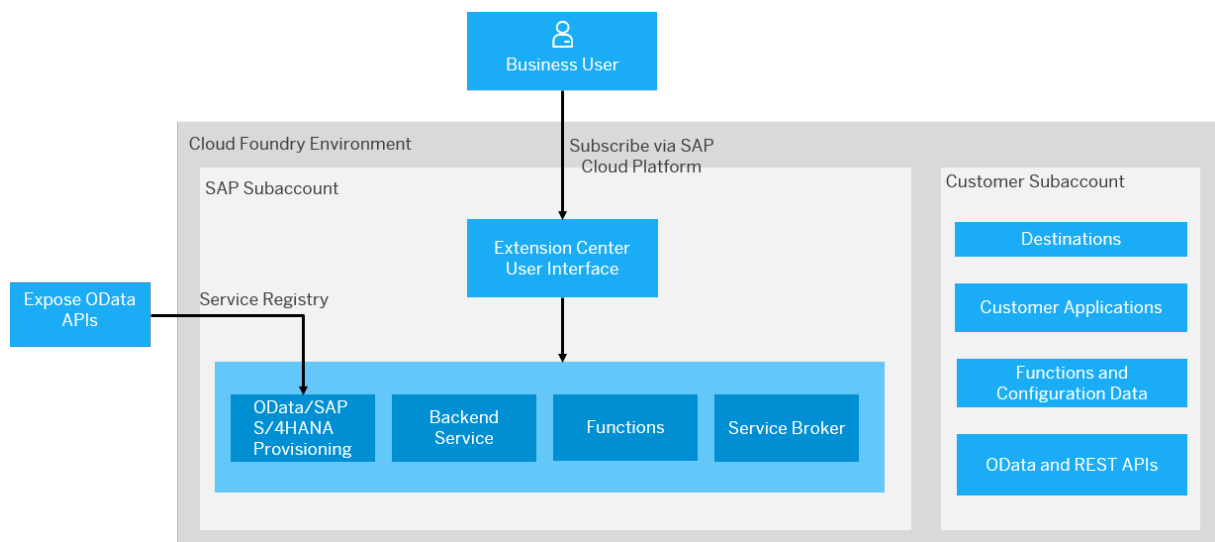
- [Technical System Landscape \[page 54\]](#)
- [User Roles \[page 55\]](#)
- [Authentication and Authorization \[page 55\]](#)
- [Transport Encryption \[page 56\]](#)
- [Data Protection and Privacy \[page 57\]](#)

SAP BTP, serverless runtime runs on SAP BTP, for this reason; all security requirements for SAP BTP are also applicable to the service.

7.1 Technical System Landscape

Use SAP BTP, serverless runtime in the Cloud Foundry environment with a web browser.

The following figure illustrates the technical system landscape for the service:



7.2 User Roles

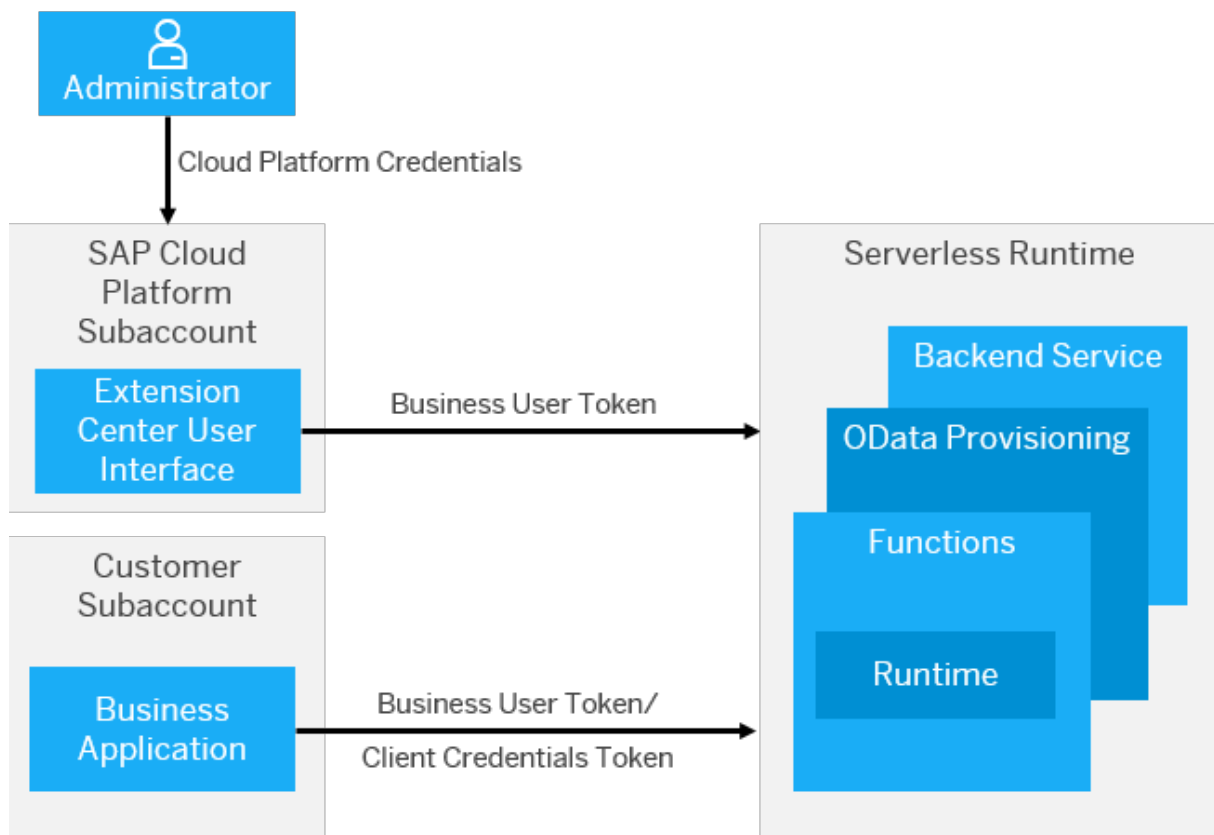
User authentication in SAP BTP, serverless runtime uses the authentication methods supported by SAP BTP. It uses the Security Assertion Markup Language (SAML) 2.0 protocol for both authentication and single sign-on.

User Type	Usage	Authentication	Authorization/Scope
XSUAA Shared (Business User)	To access Extension Center and manage its capabilities	OAuth2 (Authorization Code Grant Flow)	<ul style="list-style-type: none">• FunctionsManage• FunctionsRead• ODPAPIAccess• ODPMManage and API- Manage
XSUAA Shared (Business User)	To create and manage extensions	OAuth2 (Authorization Code Grant Flow)	<ul style="list-style-type: none">• FunctionsManage• FunctionsRead
XSUAA Shared (Business User)	To access the registered OData services to fetch data from SAP Business Suite	OAuth2 (Authorization Code Grant Flow)	APIFullAccess and ODPA- PIAccess

7.3 Authentication and Authorization

Authorization in the SAP BTP, serverless runtime determines access to applications. Administrators generally authorize users.

Assign authorizations to specify the actions users are allowed to perform.



Component	Description	Mechanism
Extension Center	For Functions source code, configuration data, OData APIs from SAP Business Suite and SAP S/4HANA on-premise systems, and full-stack APIs from data models.	OAuth 2.0 authorization code grant flow
Capability Runtime	Data defined by the user	Business user or OAuth 2.0 client credentials token

7.4 Transport Encryption

All outside communication channels are encrypted via TLS (Transport Layer Security). External communication is solely based on HTTPS.

7.5 Data Protection and Privacy

Data protection is associated with numerous legal requirements and privacy concerns. In addition to compliance with general data protection and privacy acts, it's necessary to consider compliance with industry-specific legislation in different countries.

SAP provides specific features and functions to support compliance with regard to relevant legal requirements, including data protection. SAP doesn't give any advice on whether these features and functions are the best method to support company, industry, regional, or country-specific requirements. Furthermore, this information must not be taken as advice or a recommendation regarding additional features that would be required in specific IT environments. Decisions related to data protection must be made on a case-by-case basis, considering the given system landscape and the applicable legal requirements.

SAP BTP, serverless runtime doesn't store any private data.

→ Remember

- Function source code and configuration data must be stored in the source code repository. The extension code in the Extension Center stores only the latest version and is stored until it's deleted by you. You can't retrieve older versions of the extension code. It's recommended that you don't add any personal data to the function source code or configuration data.
- For provisioning an OData API, enable Read Access Logs for sensitive data in the back-end system.

i Note

SAP doesn't provide legal advice in any form. SAP software supports data protection compliance by providing security features and specific data protection-relevant functions. In many cases, compliance with applicable data protection and privacy laws will not be covered by a product feature. Definitions and other terms used in this document aren't taken from a particular legal source.

⚠ Caution

The extent to which data protection is supported by technical means depends on secure system operation. Network security, security note implementation, adequate logging of system changes, and appropriate usage of the system are the basic technical requirements for compliance with data privacy legislation and other legislation.

Related Information

[Security](#)

8 Monitoring and Troubleshooting

If you encounter an issue with this service, we recommend that you do the following:

Announcements and Subscriptions

You can follow the availability of SAP BTP at [SAP Trust Center](#). See,

- the availability by service on the [SAP Serverless Runtime service](#) tile of the [Cloud Status](#) tab page;
- the availability by region on the [Data Center](#) tab page.

To get notifications for updates and downtimes, subscribe at the [Cloud System Notification Subscriptions](#) application. Create a subscription by specifying Cloud Product, Cloud Service, and Notification Type. For more information, see [Cloud System Notification Subscriptions User Guide](#).

Contact SAP Support

You can report an incident or error through the [SAP Support Portal](#).

Use the following components for your incident:

Component Name	Component Description
BC-CP-XF-SRT	Issues on Serverless Runtime
BC-CP-XF-SRT-FUN	Issues on Functions runtime
BC-CP-XF-SRT-ODP	Issues on OData Provisioning
BC-CP-XF-SRT-EC	Issues on the Extension Center business application



When submitting the incident, we recommend including the following information:

- Region information (Canary, EU10, US10)
- Subaccount technical name
- The URL of the page where the incident or error occurs
- The steps or clicks used to replicate the error
- Screenshots, videos, or the code entered

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.

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.