



PUBLIC
2021-08-26

SAP BTP Connectivity

Content

1	Connectivity.	3
1.1	Connectivity in the Cloud Foundry Environment.	7
	What Is SAP BTP Connectivity?.	8
	What's New for Connectivity.	14
	Initial Setup.	39
	Developing Applications.	165
	Security.	275
	Monitoring and Troubleshooting.	275
1.2	Cloud Connector.	276
	Installation.	282
	Configuration.	321
	Operations.	512
	Security.	567
	Upgrade.	578
	Update the Java VM.	580
	Uninstallation.	581
	Frequently Asked Questions.	582
1.3	Connectivity Proxy for Kubernetes.	590
	Concepts.	591
	Lifecycle Management.	612
	Verification and Testing.	629
	Monitoring.	631
	Using the Connectivity Proxy.	632
	Troubleshooting.	635
	Frequently Asked Questions.	641
1.4	Connectivity via Reverse Proxy.	643
1.5	Connectivity Support.	644
	Release and Maintenance Strategy.	646

1 Connectivity

SAP BTP Connectivity: overview, features, restrictions.

i Note

This documentation refers to SAP BTP, Cloud Foundry environment. If you are looking for information about the Neo environment, see [Connectivity for the Neo Environment](#).

Content

In this Topic

Hover over the elements for a description. Click an element for more information.

Overview

Features

Restrictions

- [Overview \[page 4\]](#)
- [Features \[page 5\]](#)
- [Restrictions \[page 5\]](#)

In this Guide

Hover over the elements for a description. Click an element for more information.

Cloud Foundry Environment

Cloud Connector

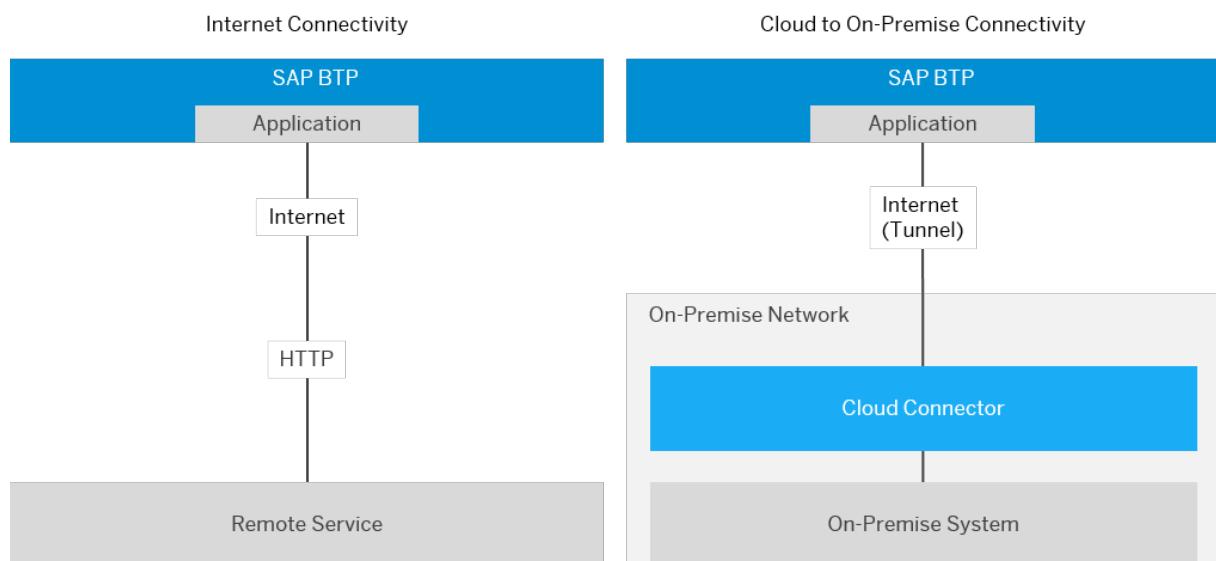
Support

- [Connectivity in the Cloud Foundry Environment \[page 7\]](#)
- [Cloud Connector \[page 276\]](#)
- [Connectivity Support \[page 644\]](#)

Overview

SAP BTP Connectivity allows SAP BTP applications to securely access remote services that run on the Internet or on-premise. This component:

- Allows subaccount-specific configuration of application connections via destinations.
- Provides a Java API that application developers can use to consume remote services.
- Allows you to make connections to on-premise systems, using the Cloud Connector.
- Lets you establish a secure tunnel from your on-premise network to applications on SAP BTP, while you keep full control and auditability of what is exposed to the cloud.
- Supports both the Neo and the Cloud Foundry environment for application development on SAP BTP.



A typical scenario for connecting your on-premise network to SAP BTP looks like this:

- Your company owns a global account on SAP BTP and one or more subaccounts that are assigned to this global account.
- Using SAP BTP, you subscribe to or deploy your own applications.
- To connect to these applications from your on-premise network, the Cloud Connector administrator sets up a secure tunnel to your company's subaccount on SAP BTP.
- The platform ensures that the tunnel can only be used by applications that are assigned to your subaccount.
- Applications assigned to other (sub)accounts cannot access the tunnel. It is encrypted via transport layer security (TLS), which guarantees connection privacy.

For inbound connections (calling an application or service on SAP BTP from an external source), you can use Cloud Connector [service channels \[page 492\]](#) (on-premise connections) or the respective API endpoints of your SAP BTP [region](#) (Internet connections).

Back to [Content \[page 3\]](#)

Features

SAP BTP Connectivity supports the following protocols and scenarios:

Protocol	Scenario
HTTP(S)	<p>Exchange data between your cloud application and Internet services or on-premise systems.</p> <ul style="list-style-type: none">• Create and configure HTTP destinations to make Web connections.• Connect to on-premise systems via HTTP, using the Cloud Connector.
RFC	<p>Invoke on-premise ABAP function modules via RFC.</p> <ul style="list-style-type: none">• Create and configure RFC destinations.• Make connections to back-end systems via RFC, using the Cloud Connector.
TCP	<p>Access on-premise systems via TCP-based protocols using a SOCKS5 proxy.</p>

Back to [Content \[page 3\]](#)

Restrictions

[General \[page 5\]](#)

[Protocols \[page 6\]](#)

[Cloud Foundry Environment \[page 6\]](#)

[Cloud Connector \[page 7\]](#)

i Note

For information about general SAP BTP restrictions, see [Prerequisites and Restrictions](#).

General

Topic	Restriction
Java Connector	To develop a Java Connector (JCo) application for RFC communication, your SDK local runtime must be hosted by a 64-bit JVM, on a x86_64 operating system (Microsoft Windows OS, Linux OS, or Mac OS X). On Windows platforms, you must install the Microsoft Visual Studio C++ 2013 runtime libraries (vcredist_x64.exe), see Visual C++ Redistributable Packages for Visual Studio 2013 .
Ports	For Internet connections, you are allowed to use any port > 1024 . For cloud to on-premise solutions there are no port limitations.
Destination Configuration	<ul style="list-style-type: none"> • You can use destination configuration files with extension .props, .properties, .jks, and .txt, as well as files with no extension. • If a destination configuration consists of a keystore or truststore, it must be stored in JKS files with a standard .jks extension.

Back to [Restrictions \[page 5\]](#)

Protocols

For the cloud to on-premise connectivity scenario, the following protocols are currently supported:

Protocol	Info
HTTP	HTTPS is not needed, since the tunnel used by the Cloud Connector is TLS-encrypted.
RFC	You can communicate with SAP systems down to SAP R/3 release 4.6C. Supported runtime environment is SAP Java Buildpack with a minimal version of 1.8.0.
TCP	You can use TCP-based communication for any client that supports SOCKS5 proxies.

Back to [Restrictions \[page 5\]](#)

Cloud Foundry Environment

Topic	Restriction
Service Channels	Service channels are supported only for SAP HANA database, see Using Service Channels [page 492] .
E-Mail	E-mail functions are not supported.

Back to [Restrictions \[page 5\]](#)

Cloud Connector

Topic	Restriction
Scenarios	To learn in which system landscapes you can set up the Cloud Connector, see Extended Scenarios [page 281] .
Installation	To check all software and hardware restrictions for working with the Cloud Connector, see Prerequisites [page 283] .

Back to [Restrictions \[page 5\]](#)

Back to [Content \[page 3\]](#)

Related Information

[Connectivity in the Cloud Foundry Environment \[page 7\]](#)

[Cloud Connector \[page 276\]](#)

[Connectivity via Reverse Proxy \[page 643\]](#)

[Connectivity Support \[page 644\]](#)

[Connectivity Proxy for Kubernetes \[page 590\]](#)

1.1 Connectivity in the Cloud Foundry Environment

Consuming SAP BTP Connectivity for your application in the Cloud Foundry environment: Overview.

i Note

This documentation refers to SAP BTP, Cloud Foundry environment. If you are looking for information about the Neo environment, see [Connectivity for the Neo Environment](#).

Hover over the elements for a description. Click an element for more information.

[What is SAP BTP Connectivity](#)

[What's New for Connectivity](#)

[Initial Setup](#)

[Developing Applications](#)

[Security](#)

[Monitoring and Troubleshooting](#)

- [What Is SAP BTP Connectivity? \[page 8\]](#)
- [What's New for Connectivity \[page 14\]](#)
- [Initial Setup \[page 39\]](#)
- [Monitoring and Troubleshooting \[page 275\]](#)
- [Security \[page 275\]](#)
- [Developing Applications \[page 165\]](#)

1.1.1 What Is SAP BTP Connectivity?

Use SAP BTP Connectivity for your application in the Cloud Foundry environment: available services, connectivity scenarios, user roles.

Content

Hover over the elements for a description. Click an element for more information.



- [Services \[page 8\]](#)
- [Scenarios \[page 9\]](#)
- [User Roles \[page 9\]](#)

Services

SAP BTP Connectivity provides two services for the Cloud Foundry environment, the Connectivity service and the Destination service.

The Destination service and the Connectivity service together provide virtually the same functionality that is included in the Connectivity service of the Neo environment.

In the Cloud Foundry environment however, this functionality is split into two separate services:

- The **Connectivity** service provides a connectivity proxy that you can use to access on-premise resources.
- Using the **Destination** service, you can retrieve and store the technical information about the target resource (destination) that you need to connect your application to a remote service or system.

You can use both services together as well as separately, depending on the needs of your specific scenario.

Back to [Content \[page 8\]](#)

Scenarios

- Use the **Connectivity** service to connect your application or an SAP HANA database to *on-premise systems*:
 - Set up on-premise communication via HTTP or RFC for your cloud application.
 - Use a service channel to connect to an SAP HANA database on SAP BTP from your on-premise system, see [Configure a Service Channel for an SAP HANA Database \[page 493\]](#).
- Use the **Destination** service:
 - To retrieve technical information about destinations that are required to consume the Connectivity service (optional), or
 - To provide destination information for connecting your Cloud Foundry application to any other *Web application* (remote service). This scenario does not require the Connectivity service.

Back to [Content \[page 8\]](#)

User Roles

In this document, we refer to different types of user roles – *responsibility roles* and *technical roles*. Responsibility roles describe the required user groups and their general tasks in the end-to-end setup process. Configuring technical roles, you can control access to the dedicated cloud management tools by assigning specific permissions to users.

[Responsibility Roles \[page 9\]](#)

[Technical Roles \[page 10\]](#)

Responsibility Roles

The end-to-end use of the Connectivity service and the Destination service requires these **user groups**:

- *Application operators* - are responsible for productive deployment and operation of an application on SAP BTP. Application operators are also responsible for configuring the remote connections (destination and trust management) that an application might need, see [Initial Setup \[page 39\]](#).
- *Application developers* - develop a connectivity-enabled SAP BTP application by consuming the Connectivity service and/or the Destination service, see [Developing Applications \[page 165\]](#).
- *IT administrators* - set up the connectivity to SAP BTP in your on-premise network, using the [Cloud Connector \[page 276\]](#).

Some procedures on the SAP BTP can be done by developers as well as by application operators. Others may include a mix of development and operation tasks. These procedures are labeled using icons for the respective task type.

Task Types

 Operator	 Developer	 Operator and/or Developer
--	---	---

Technical Roles

To perform connectivity tasks in the Cloud Foundry environment, the following **technical roles** apply:

[Technical Roles \[Feature Set A\] \[page 10\]](#)

[Technical Roles \[Feature Set B\] \[page 11\]](#)

i Note

To apply the correct technical roles, you must know on which cloud management tools feature set (A or B) your account is running. For more information on feature sets, see [Cloud Management Tools — Feature Set Overview](#).

Technical Roles [Feature Set A]

Technical Connectivity Roles and Operations [Feature Set A]

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
Subaccount	Connect a Cloud Connector to a sub-account (Cloud Connector) Disconnect a Cloud Connector (cockpit) Manage destinations (all CRUD operations) on subaccount level (cockpit) View destinations (read operations) on subaccount level (cockpit) Manage certificates (all CRUD operations) on subaccount level (cockpit) View certificates (read operations) on subaccount level (cockpit) Generate or renew the subaccount key pair for trust management (cockpit) Download the subaccount key pair for trust management (cockpit)	One of these roles: <ul style="list-style-type: none">• <i>Global Account</i> member See Add Members to Your Global Account.• <i>Security Administrator</i> (must be Global Account member or Cloud Foundry Org/Space member) See Managing Security Administrators in Your Subaccount [Feature Set A].

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
Subaccount	<p>View Cloud Connectors connected to a subaccount (cockpit)</p>	A Cloud Foundry org role containing the permission <code>readSCCTunnels</code> , for example, the role <code>Org Manager</code> .
Service instance	<p>Manage destinations (all CRUD operations) on service instance level (cockpit)</p> <p>View destinations (read operations) on service instance level (cockpit)</p> <p>Manage certificates (all CRUD operations) on service instance level (cockpit)</p> <p>View certificates (read operations) on service instance level (cockpit)</p>	One of these roles: <ul style="list-style-type: none"> • <code>Org Manager</code> • <code>Space Manager</code> • <code>Space Developer</code> <p>See User and Member Management.</p>

Back to [Technical Roles \[page 10\]](#)

Back to [User Roles \[page 9\]](#)

Technical Roles [Feature Set B]

Feature set B provides **dedicated roles** for specific operations. They can be assigned to **custom role collections**, but some of them are also available in **default role collections**.

[Technical Connectivity Roles and Operations \[Feature Set B\] \[page 11\]](#)

[Default Role Collections \[Feature Set B\] \[page 13\]](#)

i Note

To see the Destination editor, you must have at least the *Destination Viewer* role or both the *Destination Configuration Viewer* and the *Destination Certificate Viewer* roles.

Technical Connectivity Roles and Operations [Feature Set B]

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
Subaccount	Connect a Cloud Connector to a subaccount (Cloud Connector)	Cloud Connector Administrator

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
	Manage destinations (all CRUD operations) on subaccount level (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Administrator• Destination Configuration Administrator
	View destinations (read operations) on subaccount level (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Viewer• Destination Configuration Viewer
	Manage certificates (all CRUD operations) on subaccount level (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Administrator• Destination Certificate Administrator
	View certificates (read operations) on subaccount level (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Viewer• Destination Certificate Viewer
	Generate or renew the subaccount key pair for trust management (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Administrator• Destination Subaccount Trust Administrator
	Download the subaccount key pair for trust management (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Viewer• Destination Subaccount Trust Viewer
Subaccount	View Cloud Connectors connected to a subaccount (cockpit)	A role containing the permission <code>readSCCTunnels</code> , for example, the predefined role <code>Cloud Connector Administrator</code> .
Service instance	Manage destinations (all CRUD operations) on service instance level (cockpit)	One of these roles: <ul style="list-style-type: none">• Destination Administrator• Destination Configuration Administrator <i>plus</i> one of these roles: <ul style="list-style-type: none">• Org Manager• Space Manager• Space Developer
		See User and Member Management .

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
	<p>View destinations (read operations) on service instance level (cockpit)</p> <p>Manage certificates (all CRUD operations) on service instance level (cockpit)</p> <p>View certificates (read operations) on service instance level (cockpit)</p>	<p>One of these roles:</p> <ul style="list-style-type: none"> • Destination Viewer • Destination Configuration Viewer <p><i>plus one of these roles:</i></p> <ul style="list-style-type: none"> • Org Manager • Space Manager • Space Developer <p>See User and Member Management.</p> <p>One of these roles:</p> <ul style="list-style-type: none"> • Destination Administrator • Destination Certificate Administrator <p><i>plus one of these roles:</i></p> <ul style="list-style-type: none"> • Org Manager • Space Manager • Space Developer <p>See User and Member Management.</p> <p>One of these roles:</p> <ul style="list-style-type: none"> • Destination Viewer • Destination Certificate Viewer <p><i>plus one of these roles:</i></p> <ul style="list-style-type: none"> • Org Manager • Space Manager • Space Developer <p>See User and Member Management.</p>

Back to [Technical Roles \[Feature Set B\] \[page 11\]](#)

Default Role Collections [Feature Set B]

Default Role Collection	Connectivity Roles Included
Subaccount Administrator	<ul style="list-style-type: none"> • Cloud Connector Administrator • Destination Administrator
Subaccount Viewer	<ul style="list-style-type: none"> • Cloud Connector Auditor • Destination Viewer
Cloud Connector Administrator	Cloud Connector Administrator

Default Role Collection	Connectivity Roles Included
Destination Administrator	Destination Administrator
Connectivity and Destination Administrator	<ul style="list-style-type: none">Cloud Connector AdministratorDestination Administrator

i Note

You can access subaccount-level destinations in two ways:

- Via the cockpit (as described above)
- Via the Destination service REST API 

If a user has access to the Destination service REST API (via service instance binding credentials or a service key), he has full access to the destination and certificate configurations managed by that instance of the Destination service.

For more information, see [About Roles in the Cloud Foundry Environment](#) and check the activity *Instantiate and bind services to apps* in the linked Cloud Foundry documentation (docs.cloudfoundry.org).

Additionally, applications have access to the REST API of the Destination service instance they are bound to.

Back to [Technical Roles \[Feature Set B\] \[page 11\]](#)

Back to [Technical Roles \[page 10\]](#)

Back to [User Roles \[page 9\]](#)

Back to [Content \[page 8\]](#)

Related Information

[What's New for Connectivity \[page 14\]](#)

[Initial Setup \[page 39\]](#)

[Developing Applications \[page 165\]](#)

[Security \[page 275\]](#)

[Monitoring and Troubleshooting \[page 275\]](#)

[Security Administration: Managing Authentication and Authorization](#)

1.1.2 What's New for Connectivity

Find the latest features, enhancements and bug fixes for SAP BTP Connectivity .

[What's New for Connectivity](#)

Related Information

[2020 Connectivity \(Archive\) \[page 15\]](#)

[2019 Connectivity \(Archive\) \[page 23\]](#)

[2018 Connectivity \(Archive\) \[page 31\]](#)

[2017 Connectivity \(Archive\) \[page 35\]](#)

1.1.2.1 2020 Connectivity (Archive)

2020

Technical Component	Capability	Environment	Title	Description	Type	Available as of
Connectivity	Integration Suite	Neo Cloud Foundry	Java Connector (JCo) - Client Certificates	JCo provides the new property <code>jco.client.tls_client_certificate_logon</code> to support the usage of a TLS client certificate for logging on to an ABAP system via WebSocket RFC. For more information, see: User Logon Properties (Cloud Foundry environment) User Logon Properties (Neo environment) For more information on WebSocket RFC, see also: WebSocket RFC	New	2020-12-17
Connectivity	Integration Suite	Cloud Foundry	HTTP Destinations - Authentication Types	Authentication type SAP Assertion SSO is deprecated. It will soon be removed as a feature from the Destination service. Use Principal Propagation SSO Authentication instead, which is the recommended mechanism for establishing single sign-on (SSO).	Deprecated	2020-12-17
Connectivity	Integration Suite	Neo	HTTP Destinations - Authentication Types	Authentication type SAP Assertion SSO is deprecated. Use Principal Propagation SSO Authentication instead, which is the recommended mechanism for establishing single sign-on (SSO).	Deprecated	2020-12-17

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Connectiv- ity	Inte- gration Suite	Cloud Foun- dry	HTTP Destina- tion Proper- ties	The destination property <code>SystemUser</code> for the authentication types: <ul style="list-style-type: none">• OAuth SAML Bearer Assertion Authentication• SAP Assertion SSO Authentication will be removed soon. More information on timelines and required actions will be published in the release notes at a later stage. See also: OAuth SAML Bearer Assertion Authentication SAP Assertion SSO Authentication	Announce- ment	2020-1-03
Connectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	JCo Run- time - Enhance- ment	JCo Runtime 3.1.3.0 introduces the following enhancement: If the backend is known to be new enough, JCo does not check for the existence of <code>RFC_METADATA_GET</code> , thus avoiding the need to provide additional authorizations for the repository user.	New	2020-1-05
Connectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	JCo Run- time - Bug Fix	JCo Runtime 3.1.3.0 provides the following bug fix: Up to JCo 3.1.2, the initial value for fields of type <code>STRING</code> and <code>XSTRING</code> was <code>null</code> . Since the initial value check in ABAP is different, JCo now behaves the same way and uses an empty string and an empty byte array, respectively.	Change- ed	2020-1-05
Connectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Auto- matic To- ken Re- trieval	The Destination service offers a new feature related to the automatic token retrieval functionality, which lets the destination administrator define HTTP headers and query parameters as additional configuration properties, used at runtime when requesting the token service to obtain an access token. See HTTP Destinations .	New	2020-1-05
Connectiv- ity	Inte- gration Suite	Cloud Foun- dry	Docu- menta- tion - Principal Propaga- tion Sce- narios	The documentation of principal propagation (user propagation) scenarios provides improved information on the basic concept and guidance on how to set up different scenarios. See Principal Propagation .	Change- ed	2020-1-022

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	Cloud Connec- tor 2.12.5 - En- hance- ments	<p>Release of Cloud Connector version 2.12.5 introduces the following improvements:</p> <ul style="list-style-type: none"> For principal propagation scenarios, custom attributes stored in <code>xs.user.attributes</code> of the JWT (JSON Web token) are now accessible for the subject pattern. See Configure a Subject Pattern for Principal Propagation. Improved resolving for DNS names with multiple IP addresses by adding randomness to the choice of the IP to use. This is relevant for many connectivity endpoints in SAP Cloud Platform, Cloud Foundry environment. 	New	2020-1 0-22
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Found- ry	Cloud Connec- tor 2.12.5 - Fixes	<p>Release of Cloud Connector version 2.12.5 provides the following bug fixes:</p> <ul style="list-style-type: none"> After actively performing a master-shadow switch for a disaster recovery subaccount, a zombie connection could cause a timeout of all application requests to on-premise systems. This issue has been fixed. When refreshing the subaccount certificate in an high availability setup, transferring the changed certificate to the shadow was not immediately triggered, and the updated certificate could get lost. This issue has been fixed. If many RFC connections were canceled at the same time, the Cloud Connector could crash in the native layer, causing the process to die. This issue has been fixed. The LDAP configuration test now supports all possible configuration parameters. 	Chang ed	2020-1 0-22
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	Connec- tivity Service - Service Instances - Quota Manage- ment	<p>When using service plan "lite", quota management is no longer required for this service. From any subaccount you can consume the service using service instances without restrictions on the instance count.</p> <p>Previously, access to service plan "lite" has been granted via entitlement and quota management of the application runtime. It has now become an integral service offering of SAP Cloud Platform to simplify its usage.</p> <p>See also Create and Bind a Connectivity Service Instance.</p>	Chang ed	2020-1 0-08

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	Destina- tion Service - Service Instances - Quota Manage- ment	<p>When using service plan “lite”, quota management is no longer required for this service. From any subaccount you can consume the service using service instances without restrictions on the instance count.</p> <p>Previously, access to service plan “lite” has been granted via entitlement and quota management of the application runtime. It has now become an integral service offering of SAP Cloud Platform to simplify its usage.</p>	Chang ed	2020-1 0-08
				See also Create and Bind a Destination Service Instance .		
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	SAP Java Build- pack - Java Connec- tor (JCo)	<p>The SAP Java Buildpack has been updated from 1.27.3. to 1.28.0.</p> <ul style="list-style-type: none"> TomEE Tomcat has been updated from 7.0.104 to 7.0.105. SAPJVM has been updated to 81.65.65. The <code>com.sap.cloud.security.xsuaa</code> API has been updated from 2.7.5 to 2.7.6. The SAP HANA driver has been updated from 2.5.49 to 2.5.52. JCo-corresponding libraries have been updated: <code>connectivity</code> to 3.3.3, <code>connectivity_apiext</code> to 0.1.37. The activation process for the JCo component in the SAP Java Buildpack has been changed. Starting with this release, it is activated by setting the following environment variable: <code><USE_JCO=true></code>. 	Chang ed	2020-0 9-24
				<p>i Note</p> <p>The previous activation process for the JCo component is deprecated and will expire after a transition period.</p>		
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	Destina- tion Service - Error Handling	Error handling has been improved for updating service instances via the Cloud Foundry CLI and the cloud cockpit when providing the configuration JSON data.	Chang ed	2020-0 9-10
Con- nectiv- ity	Inte- gration Suite	Neo	Connec- tivity Service - Bug Fix	A synchronization issue has been fixed on cloud side that in very rare cases could lead to a zombie tunnel from the Cloud Connector to SAP Cloud Platform, which required to reconnect the Cloud Connector.	Chang ed	2020-0 9-10

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Bug Fix	During <i>Check Connection</i> processing of a destination with basic authentication, the Destination service now uses the user credentials for both the HTTP HEAD and HTTP GET requests to verify the connection on HTTP level.	Change ed	2020-0 9-10
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Bug Fix	Using authentication type <code>OAuth2SAMLBearerAssertion</code> , an issue could occur when adding the user's SAML group attributes into the resulting SAML assertion that is sent to the target token service. This issue has been fixed.	Change ed	2020-0 8-13
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service REST API - Pagina- tion Fea- ture	The REST API pagination feature provides improved error handling in case of issues with the pagination, for example, if an invalid page number is provided.	Change ed	2020-0 8-13
Con- nectiv- ity	Inte- gration Suite	Neo	HttpDes- tination Library - New Ver- sion	The <code>HttpDestination v2</code> library has been officially released in the Maven Central Repository . It enables the usage in Tomcat and TomEE-based runtimes the same way as in the deprecated JavaWeb and Java EE 6 Web Profile runtimes. See also HttpDestination Library .	New	2020-0 7-30
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Bug Fix	An error handling issue has been fixed in the Destination service, which is related to the recently introduced SAP Assertion SSO authentication type. If a wrong input was provided, you can now see the error properly, and recover it.	Change ed	2020-0 7-30
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tions - Authenti- cation Types	You can use authentication type <code>OAuth2JWTBearer</code> when configuring a Destination. It is a simplified version of the authentication type <code>OAuth2UserTokenExchange</code> and represents the official OAuth grant type for exchanging OAuth tokens. See HTTP Destinations .	New	2020-0 7-02
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - HTTP Header	The Destination service provides a prepared HTTP header that simplifies application and service development. See HTTP Destinations (code samples).	New	2020-0 7-02

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Bug Fix	A concurrency issue in the Destination service, related to parallel auth token retrieval in the token cache functionality, could result in partial request failures. This issue has been fixed.	Chang ed	2020-0 7-02
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	HTTP Destina- tions - Authenti- cation Types	The Cloud Foundry environment supports SAP Assertion SSO as authentication type for configuring destinations in the Destination service. See HTTP Destinations .	New	2020-0 6-18
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service REST API	The "Find Destination" REST API now includes the scopes of the automatically retrieved access token in the response that is returned to the caller. See "Find Destination" Response Structure .	New	2020-0 6-04
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tions for Service Instan- ces	For subscription-based scenarios, you can use an automated procedure to create a destination that points to your service instance. See Managing Destinations .	New	2020-0 6-04
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Connec- tivity Service - Bug Fix	In rare cases, establishing a secure tunnel between Cloud Connector (version 2.12.3 or older) and the Connectivity service could cause an issue that requires to manually disconnect and connect the Cloud Connector. This issue has been fixed. The fix requires Cloud Connector version 2.12.4 or higher.	Chang ed	2020-0 5-21
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12.4 - Fea- tures	Release of Cloud Connector version 2.12.4 introduces the following features and enhancements: <ul style="list-style-type: none">You can activate the SSL trace in the Cloud Connector administration UI also for the shadow instance.	New	2020-0 5-07

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Connec- tivity	Inte- gration Suite	Neo Cloud Found- ry	Cloud Connec- tor 2.12.4 - Fixes	<p>Release of Cloud Connector version 2.12.4 provides the following bug fixes:</p> <ul style="list-style-type: none"> • You can edit and delete domain mappings in the Cloud Connector administration UI correctly. • The REST API does no longer return an empty configuration. • REST API DELETE operations do not require setting a content-type application/json to function properly. • If more than 2000 audit log entries match a selection, redefining the search and getting a shorter list now works as expected. • A potential leak of HTTP backend connections has been closed. 	Changed	2020-05-07
Connec- tivity	Inte- gration Suite	Cloud Found- ry	Connec- tivity Service - Bug Fix	A fix has been applied in the Connectivity service internal load balancers, enabling the sending of TCP keep-alive packets on client and server side. This change mainly affects SOCKS5-based communication scenarios.	Changed	2020-03-26
Connec- tivity	Inte- gration Suite	Cloud Found- ry	Destina- tion Service - Service Instan- ces	You can create a service instance specifying an update policy. This allows you to avoid name conflicts with existing destinations. See Create and Bind a Destination Service Instance .	New	2020-03-26
Connec- tivity	Inte- gration Suite	Cloud Found- ry	Cockpit - Destina- tion Manage- ment	The Destinations editor in the cockpit is available for accounts running on the cloud management tools feature set B. See Managing Destinations .	New	2020-03-12
Connec- tivity	Inte- gration Suite	Neo	Connec- tivity Service - Bug Fix	When creating or editing a destination with authentication type OAuth2ClientCredentials in the cockpit, the parameter Audience could not be added as additional property. This issue has been fixed.	Changed	2020-03-12

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12.3 - Fea- tures	<p>Release of Cloud Connector version 2.12.3 introduces the following features and enhancements:</p> <ul style="list-style-type: none"> When using the SAP JVM as runtime, the thread dump includes additional information about currently executed RFC function modules. The hardware monitor includes a Java Heap history, showing the usage in the last 24 hours. If you are using the file <code>scc_daemon_extension.sh</code> to extend the daemon in a Linux installation, the content is included in the initialization section of the daemon. This lets you make custom extensions to the daemon that survive an upgrade. See Installation on Linux OS, section <i>Installer Scenario</i>. 	New	2020-0 2-27
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12.3 - Fixes	<p>Release of Cloud Connector version 2.12.3 provides the following bug fixes:</p> <ul style="list-style-type: none"> When switching roles between master and shadow instance in a high availability setup, the switch is no longer blocked by active RFC function module invocations. A fix in the backend HTTP connection handling prevents issues when the backend tries to send the HTTP response before completely reading the HTTP request. When sending large amounts of data to an on-premise system, and using RFC with a network that provides large bandwidth, the Cloud Connector could fail with the error message <i>Received invalid block with negative size</i>. This issue has been fixed. The Cloud Connector admin UI now shows the correct user information for installed Cloud Connector instances in the <i>About</i> window. Fixes in the context of disaster recovery: <ul style="list-style-type: none"> The location ID is now handled properly when setting it <i>after</i> adding the recovery subaccount. Application trust settings and application-specific connections are applied in the disaster case. Principal propagation settings are applied in the disaster case 	Chang ed	2020-0 2-27

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	JCo Run- time - Web- Socket RFC	The JCo runtime in SAP Cloud Platform lets you use WebSocket RFC (RFC over Internet) with ABAP servers as of S/4HANA (on-premise) version 1909. In the RFC destination configuration, this is reflected by new configuration properties and by the option to choose between different proxy types. See Target System Configuration (Cloud Foundry environment), or Target System Configuration (Neo environment).	New	2020-0 2-13
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Connec- tivity Service for Trial Accounts - Bug Fix	The Connectivity service is operational again for trial accounts. A change in the Cloud Foundry Core component caused the service not be accessible by applications hosted in DiegoCell that are dedicated for trial usage in a separate VPC (virtual private cloud) account. This issue has been fixed.	Changed	2020-0 1-30

1.1.2.2 2019 Connectivity (Archive)

2019

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12.2 - Fea- tures	Release of Cloud Connector version 2.12.2 introduces the following features and enhancements: <ul style="list-style-type: none"> • You can turn on the TLS trace from the Cloud Connector administration UI instead of modifying the <code>props.ini</code> file on OS level. See Troubleshooting. • The status of the used subaccount certificate is shown on the Subaccount overview page of the Cloud Connector administration UI, in addition to expiring certificates shown in the Alerting view. See Establish Connections to SAP Cloud Platform. 	New	2019-1 2-05

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Found- ry	Cloud Connec- tor 2.12.2 - Fixes	<p>Release of Cloud Connector version 2.12.2 provides the following bug fixes:</p> <ul style="list-style-type: none"> Subject values for certificates requiring escaping are treated correctly. Establishing a connection to the master is now possible when being logged on to the shadow with a user that has a space in its name. Performance statistics could show too long total execution times. This issue has been fixed. IP address changes for the connectivity service hosts are recognized properly. The Cloud Connector could crash on Windows, when trying to enable the payload trace with 4-eyes-principle without the required user permissions. This issue has been fixed. 	Chang ed	2019-1 2-05
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	Connec- tivity Service - Bug Fix	<p>Applications sending a significant amount of data payload during OAuth authorization processing could cause an out-of-memory error on the Connectivity service side. This issue has been fixed.</p>	Chang ed	2019-11 -21
Con- nectiv- ity	Inte- gration Suite	Neo	Region Europe (Frank- furt) - Change of Con- nectivity Service Hosts	<p>The following IP addresses of the Connectivity service hosts for region Europe/Frankfurt (eu2.hana.ondemand.com) will change on 26 October 2019:</p> <ul style="list-style-type: none"> connectivitynotification.eu2.hana.ondemand.com: from 157.133.70.140 (current) to 157.133.206.143 (new) connectivitycertsigning.eu2.hana.ondemand.com: from 157.133.70.132 (current) to 157.133.205.174 (new) connectivitytunnel.eu2.hana.ondemand.com: from 157.133.70.141 (current) to 157.133.205.233 (new) <p>If you have allowed the current addresses or IP ranges in your firewall rules, make sure you also include the new values before 26 October 2019.</p> <p>See also: Prerequisites: Network.</p>	An- nounce ment	2019-1 0-03

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Connectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Connec- tion Check	<p>Using the Destinations editor in the cockpit, you can check connections also for on-premise destinations.</p> <p>See Check the Availability of a Destination.</p>	Changed	2019-09-26
Connectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor - Java Runtime	<p>The support for using Cloud Connector with Java runtime version 7 will end on December 31, 2019. Any Cloud Connector version released after that date may contain Java byte code requiring at least a JVM 8.</p> <p>We therefore strongly recommend that you perform fresh installations only with Java 8, and update existing installations running with Java 7, to Java 8 as of now.</p> <p>See SAP Cloud Connector – Java 7 support will phase out and Update the Java VM.</p>	Announce	2019-09-13
Connectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12.1 - Fea- tures	<p>Release of Cloud Connector version 2.12.1 introduces the following features and enhancements:</p> <ul style="list-style-type: none"> Subject Alternative Names are separated from the subject definition and provide enhanced configuration options. You can configure complex values easily when creating a certificate signing request. See Exchange UI Certificates in the Administration UI. In a high availability setup, the master instance detection no longer switches automatically if the configuration between the two instances is inconsistent. Disaster recovery switch back to main subaccount is periodically checked (if not successful) every 6 hours. Communication to on-premise systems supports SNI (Server Name Indication). 	New	2019-08-15

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Found- ry	Cloud Connec- tor 2.12.1 - Fixes	<p>Release of Cloud Connector version 2.12.1 provides the following bug fixes:</p> <ul style="list-style-type: none"> The communication between master and shadow instance no longer ends up in unusable clients that show 403 results due to CSRF (Cross-Site Request Forgery) failures, which could cause undesired role switches. When restoring a backup, the administrator password check works with all LDAP servers. The LDAP configuration test utility properly supports secure communication. The Refresh Subaccount Certificate dialog is no longer hanging when the refresh action fails due to some authentication or authorization issue. 	Changed	2019-08-15
Con- nectiv- ity	Inte- gration Suite	Cloud Found- ry	Destina- tion Service - Scope Attribute for OAuth- based Authenti- cation Types	<p>You can use the scope destination attribute for the OAuth-based authentication types <code>OAuth2ClientCredentials</code>, <code>OAuth2UserTokenExchange</code> and <code>OAuth2SAMLBearerAssertion</code>. This additional attribute provides flexibility on destination configuration level, letting you specify what scopes are selected when the OAuth access token is automatically retrieved by the service.</p> <p>See HTTP Destinations.</p>	New	2019-08-15
Con- nectiv- ity	Inte- gration Suite	Neo SAP Cloud Platform - Fea- tures	JCo Run- time for SAP Cloud Platform - Fea- tures	<ul style="list-style-type: none"> Additional APIs have been added to <code>JCoBackgroundUnitAttributes</code>. See API documentation for details. If a structure or table contains only char-like fields, new APIs let you read or modify all of them at once for the structure or the current table row. <p>See API documentation of <code>JCoTable</code> and <code>JCoStructure</code>.</p>	New	2019-07-18
Con- nectiv- ity	Inte- gration Suite	Neo SAP Cloud Platform - Fixes	JCo Run- time for SAP Cloud Platform - Fixes	<ul style="list-style-type: none"> qRFC and tRFC requests sent to an ABAP system by JCo can be monitored again by AIF. Structure fields of type STRING are no longer truncated if there is a white space at the end of the field. 	Changed	2019-07-18

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Connec- tivity	Inte- gration Suite	Cloud Foun- dry	Connec- tivity Service - JCo Mul- itenancy	<p>The Connectivity service supports multitenancy for JCo applica- tions.</p> <p>This feature requires a runtime environment with SAP Java Buildpack version 1.9.0 or higher.</p> <p>See Scenario: Multitenancy for JCo Applications (Advanced).</p>	New	2019-0 6-20
Connec- tivity	Inte- gration Suite	Cloud Foun- dry	Cloud Cockpit - Cloud Connec- tor View	The Cloud Connector view is available also for Cloud Foundry re- gions. It lets you see which Cloud Connectors are connected to a subaccount.	New	2019-0 4-25

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12 - Features	<p>Release of Cloud Connector version 2.12 introduces the following features and enhancements:</p> <ul style="list-style-type: none"> The administration UI is now accessible not only with an administrator role, but also with a display and a support role. See Configure Named Cloud Connector Users and Use LDAP for Authentication. For HTTP access control entries, you can <ul style="list-style-type: none"> allow a protocol upgrade, e.g. to WebSockets, for exposed resources. See Limit the Accessible Services for HTTP(S). define which host (virtual or internal) is sent in the host header. See Expose Intranet Systems, Step 8. A disaster recovery subaccount in disaster recovery mode can be converted into a standard subaccount, if a disaster recovery region replaces the original region permanently. See Convert a Disaster Recovery Subaccount into a Standard Subaccount. A service channel overview lets you check at a glance, which server ports are used by a Cloud Connector installation. See Service Channels: Port Overview. Important subaccount configuration can be exported, and imported into another subaccount. See Copy a Subaccount Configuration. An LDAP authentication configuration check lets you analyze and fix configuration issues before activating the LDAP authentication. See Use LDAP for Authentication. You can use different user roles to access the Cloud Connector configuration REST APIs. See Configuration REST APIs. REST APIs for shadow instance configuration have been added. See Shadow Instance Configuration. You can define scenarios for resources. Such a scenario can be exported, and imported into other hosts. See Configure Accessible Resources. 	New	2019-04-25

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.12 - Fixes	<p>Release of Cloud Connector version 2.12 provides the following bug fixes:</p> <ul style="list-style-type: none"> The SAN (subjectAlternativeName) usage in certificates can be defined in a better way and is stored correctly in the certificate. See Exchange UI Certificates in the Administration UI. <code>IllegalArgumentException</code> does not occur anymore in HTTP processing, if the backend closes a connection and data are streamed. DNS caching is now recognized in reconnect situations if the IP of a DNS entry has changed. SNC with load balancing now works correctly for RFC SNC-based access control entries. A master-master situation is also recognized if, at startup of the former master instance, the new master (the former shadow instance) is not reachable. Solution management model generation works correctly for a shadow instance. The daemon is started properly on SLES 12 standard installations at system startup. 	Chang ed	2019-0 4-25
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tion Service - Authenti- cation Types	<p>Authentication type OAuth2SAMLBearerAssertion provides two different types of Token Service URL:</p> <ul style="list-style-type: none"> Dedicated: used in the context of a single tenant, or Common: used in the context of multiple tenants. <p>For type Common, the tenant subdomain is automatically set to the target Token Service URL.</p> <p>In addition, cloud applications can use the <code>x-user-token</code> HTTP header to propagate the user access token to the external target service at runtime. By default, the user principal is processed via the authorization HTTP header.</p> <p>See SAML Bearer Assertion Authentication.</p>	New	2019-0 4-11
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Connec- tivity Service - Fix	<p>When an on-premise system closed a connection that uses an RFC or SOCKS5 proxy, the Connectivity service kept the connection to the cloud application alive.</p> <p>This issue has been fixed. The connection is now always closed right after sending the response.</p>	Chang ed	2019-0 4-11

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Connec- tivity	Inte- gration Suite	Cloud Found- ry	Connec- tivity Service - Proto- cols	The Connectivity service supports TCP connections to on-prem- ise systems, exposing a SOCKS5 proxy to cloud applications. This feature follows the concept of binding the credentials of a Connectivity service instance. See Using the TCP Protocol for Cloud Applications .	New	2019-0 3-14
Connec- tivity	Inte- gration Suite	Neo	Connec- tivity Service - Fix	After receiving an on-premise system response with HTTP header <i>Connection: close</i> , the Connectivity service kept the HTTP connection to the cloud application alive. This issue has been fixed. The connection is now always closed right after sending the response.	Change ed	2019-0 3-14
Connec- tivity	Inte- gration Suite	Neo	Cloud Connec- tor - Cer- tificate Update	For the Connectivity service (Neo environment), a new, region- specific certificate authority (X.509 certificate) is being intro- duced. If you use the Cloud Connector for on-premise connections to the Neo environment, you must import the new certificate au- thority into your trust configuration. <ul style="list-style-type: none"> After the next month (concrete notification will be rolled out), the current certificate authority will no longer be used to issue client certificates for Cloud Connector deploy- ments, and only the new one will be used. The Connectivity service will still trust client certificates of Cloud Connector deployments that were already issued. After a three-month period (concrete notification will be rolled out), that trust will be removed and your Cloud Con- nector deployment must be configured to use the new client certificates. 	An- nounce ment	2019-0 2-28
				See Update the Certificate for a Subaccount .		
Connec- tivity	Inte- gration Suite	Cloud Found- ry	Destina- tion Service - Authenti- cation Types	The new authentication type OAuth2UserTokenExchange lets your applications use an automated exchange of user access tokens when accessing other applications or services. The fea- ture supports single-tenant and multi-tenant scenarios. See OAuth User Token Exchange Authentication .	New	2019-0 2-14
Connec- tivity	Inte- gration Suite	Neo	RFC - Stateful Sequen- ces	You can make a stateful sequence of function module invoca- tions work across several request/response cycles. See Invoking ABAP Function Modules via RFC .	Change ed	2019-0 1-31

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Inte- gration Suite	Neo Cloud Foun- dry	Cloud Connec- tor 2.11.3	A security note for Cloud Connector version 2.11.3 has been issued. See SAP note 2696233 .	Changed	2019-0 1-15
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Proto- cols - RFC Communi- cation	You can use the RFC protocol to set up communication with on-premise ABAP systems for applications in the Cloud Foundry environment. This feature requires a runtime environment with SAP Java Buildpack version 1.8.0 or higher. See Invoking ABAP Function Modules via RFC .	New	2019-0 1-17
Con- nectiv- ity	Inte- gration Suite	Cloud Foun- dry	Destina- tions - Renew Certifi- cates	A button in the Destinations editor lets you update the validity period of an X.509 certificate. See Set up Trust Between Systems .	New	2019-0 1-17

1.1.2.3 2018 Connectivity (Archive)

2018

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Integra- tion	Neo	Con- nectiv- ity Service - Per- for- mance	A change in the SAP Cloud Platform Connectivity service improves performance of data upload (on-premise to cloud) and data download (cloud to on-premise) up to 4 times and 15-30 times respectively.	Changed	2018-1 2-20

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Integra- tion	Neo	Connec- tivity Service - Resil- ience	The Connectivity service has a better protection against zombie connections, which improves resilience and overall availability for the cloud applications consuming it.	Changed	2018-1 2-20
Con- nectiv- ity	Integra- tion	Neo	Password Storage Service	A Password Storage REST API is available in the SAP API Business Hub, see Password Storage (Neo Environment) .	New	2018-1 2-06
Con- nectiv- ity	Integra- tion	Neo	Desti- nation Config- uration Service	A Destination Configuration service REST API is available in the SAP API Business Hub.	New	2018-1 2-06
Con- nectiv- ity	Integra- tion	Cloud Foun- dry	Desti- nation Service	A Destination service REST API is available in the SAP API Business Hub.	New	2018-1 2-06
Con- nectiv- ity	Integra- tion	Neo	JCo Run- time for SAP Cloud Plat- form - Fixes	<ul style="list-style-type: none"> When using <code>JCoRecord.fromJSON()</code> for a structure parameter, the data is now always sent to the backend system. Also, you do not need to append the number of provided rows for table parameters before parsing the JSON document anymore. Depending on the configuration of certain JCo properties, an internally managed connection pool could throw a <code>JCoException</code> (error group <code>JCO_ERROR_RESOURCE</code>). In a thread waiting for a free connection from this pool, an error message then erroneously reported that the pool was exhausted. <p>This error situation could occur if the used destination was not configured with the property <code>jco.destination.max_get_client_time</code> set to 0 and the destination's <code>jco.destination.peak_limit</code> value was set higher than the <code>jco.destination.pool_capacity</code>. This issue has been fixed.</p>	Changed	2018-1 2-06

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Integra- tion	Neo Run- time for SAP Cloud Plat- form - Fea- tures	JCo	<p>Support of the RFC fast serialization. Depending on the exchanged parameter and data types, the performance improvements for RFC communication can reach multiple factors.</p> <p>See SAP note 2372888 (prerequisites) and Parameters Influencing Communication Behavior (JCo configuration in SAP Cloud Platform).</p>	Changed	2018-1 2-06
Con- nectiv- ity	Integra- tion	Neo Run- time for SAP Cloud Plat- form - Infor- mation	JCo	<p>Local runtimes on Windows must install the VS 2013 redistributables for x64, instead of VS 2010.</p>	Changed	2018-1 2-06
Con- nectiv- ity	Integra- tion	Neo Cloud Found- ry	Cloud Con- nector Fixes	<p>Release of Cloud Connector 2.11.3:</p> <ul style="list-style-type: none"> An issue in RFC communication could cause the trace entry <code>com.sap.scc.jni.CpicCommunicationException: no SAP ErrInfo available</code> when the network is slow. This issue has been fixed. The Windows service no longer runs in error 1067 when stopped by an administrator. In previous releases, the connection between a shadow and a master instance occasionally failed at startup and produced an empty error message. This issue has been fixed. The Cloud Connector does not cache Kerberos tokens in the protocol handler any more, as they are one-time tokens and cannot be reused. For HTTP access control entries, you can configure resources containing a # character. 	Changed	2018-1 2-06

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Integra- tion	Neo Cloud Found- ry	Cloud Con- nector En- hance- ments	<p>Release of Cloud Connector 2.11.3:</p> <ul style="list-style-type: none"> If the user <code>sapadm</code> exists on a system, the installation on Linux assigns it to the <code>sccgroup</code>, which is a prerequisite for solution management integration to work properly, see Configure Solution Management Integration [page 504]. Restoring a backup has been improved. See Configuration Backup [page 508]. The HTTP session store size has been reduced. You can handle higher loads with a given heap size. Cipher suite configuration has been improved. Also, there is a new security status entry for cipher suites, see Recommendations for Secure Setup [page 309]. 	Chang ed	2018-1 2-06
Con- nectiv- ity	Integra- tion	HTTP Desti- nations	Neo	<p>The OAuth2 Client Credentials grant type is supported by the Destinations editor in the SAP Cloud Platform cockpit as well as by the client Java APIs ConnectivityConfiguration, AuthenticationHeaderProvider and HttpDestination, available in SAP Cloud Platform Neo run-times.</p> <p>See OAuth Client Credentials Authentication.</p>	Chang ed	2018-1 0-11
Con- nectiv- ity	Integra- tion	Cloud Found- ry	User Propaga- tion	<p>The connectivity service supports the SaaS application subscription flow and can be declared as a dependency in the get dependencies subscription callback, also via MTA (multi-target)-bundled applications.</p> <p>See Consuming the Connectivity Service (Cloud Foundry Environment) and Configure Principal Propagation via User Exchange Token (Cloud Foundry Environment).</p>	Chang ed	2018-0 9-27
Con- nectiv- ity	Integra- tion	Neo Cloud Found- ry	Cloud Con- nector 2.11.2	<p>Release of Cloud Connector 2.11.2</p> <ul style="list-style-type: none"> SNC configuration now provides the value of the environment variable SECUDIR, which you need for the usage of the SAP Cryptographic Library (SAPCRYPTOLIB). See Initial Configuration (RFC). On Linux, the RPM (Red Hat Package Manager) now ensures that the configuration of the interaction with the SAP Host Agent (used for the Solution Manager integration) is adjusted. See Configure Solution Management Integration. The Cloud Connector shadow instance now provides a configuration option for the connection and request timeout that may occur during health check against the master instance. See Master and Shadow Administration. 	Chang ed	2018-0 8-16

Techni- cal Com- ponent	Capa- bility	Envi- ron- ment	Title	Description	Type	Availa- ble as of
Con- nectiv- ity	Integra- tion	Neo Cloud Foun- dry	Cloud Con- nector 2.11.2	<p>Fixes of Cloud Connector 2.11.2</p> <ul style="list-style-type: none"> In a high availability setup, the switch from the master instance to the shadow instance occasionally caused communication errors towards on-premise systems. This issue has now been fixed. You can now import multiple certificates with the same subject to the trust store. Details about expiration date and issuer are displayed in the tool tip. See Set Up Trust, section <i>Trust Store</i>. You can now configure also the MOC (Multiple Origin Composition) OData service paths as resources. The <code>Location</code> header is now adjusted correctly according to your access control settings in case of a redirect. Principal propagation now also works with SAML assertions that contain an empty attribute element. SAP Cloud Platform applications occasionally got an HTTP 500 (internal server error) response when an HTTP connection was closed. The applications are now always informed properly. 	Chang ed	2018-0 8-16
Con- nectiv- ity	Integra- tion	HttpDe stina- tion Li- brary	HttpDe stina- tion Li- brary	<p>The SAP <code>HttpDestination</code> library (available in the SDK and cloud runtime "Java EE 6 Web Profile") now creates Apache <code>HttpClient</code> instances which work with strict SNI (Server Name Indication) servers.</p> <p>Use cases with strict SNI configuration on the server side will no longer get the error message <i>Failure reason: "peer not authenticated"</i>, that was raised either at runtime or while performing a connection test via the SAP Cloud Platform cockpit Destinations editor (Check Connection function).</p>	Chang ed	2018-0 8-16

1.1.2.4 2017 Connectivity (Archive)

Archived release notes for 2017 and older.

28 September 2017 - Connectivity

New

The destination service (Beta) is available in the Cloud Foundry environment. See [Consuming the Destination Service \[page 191\]](#).

3 August 2017 - Connectivity

Enhancement

Cloud Connector

Release of SAP Cloud Platform Cloud Connector 2.10.1.

- The URLs of HTTP requests can now be longer than 4096 bytes.
- SAP Solution Manager can be integrated with one click of a button if the host agent is installed on a Cloud Connector machine. See the *Solution Management* section in [Monitoring \[page 526\]](#).
- The limitation that only 100 subaccounts could be managed with the administration UI has been removed. See [Managing Subaccounts \[page 338\]](#).

Fix

Cloud Connector

- The regression of 2.10.0 has been fixed, as principal propagation now works for RFC.
- The cloud user store works with group names that contain a backslash (\) or a slash (/).
- Proxy challenges for NT LAN Manager (NTLM) authentication are ignored in favor of Basic authentication.
- The back-end connection monitor works when using a JVM 7 as a runtime of Cloud Connector.

25 May 2017 - Connectivity

Enhancement

Cloud Connector

Release of SAP HANA Cloud connector 2.10.0.1

- Support of connectivity to an SAP Cloud Platform Cloud Foundry environment.
- Support of direct connectivity with S/4HANA Cloud systems. You can open a Service Channel to an S/4HANA Cloud system in order to use Communication Scenarios requiring RFC communication to S/4HANA Cloud. See [Configure a Service Channel for RFC \[page 497\]](#).
- Support of arbitrary protocols via the possibility to configure a TCP access control entry. SAP Cloud Platform Connectivity is offering a SOCKS5 proxy, with which you can address such exposed hosts. See [Using the TCP Protocol for Cloud Applications](#).
- Support for disaster recovery events of SAP Cloud Platform regions. For each subaccount you can configure a disaster recovery subaccount for a disaster region. In case of a disaster, the disaster recovery account can be switched active immediately using the exact same configuration. See [Configure a Disaster Recovery Subaccount \[page 346\]](#).
- In the access control settings you can add further constraints for RFC based communication to ABAP systems: an administrator can configure, which clients shall be exposed and can define which users should not be able to access the system via Cloud Connector. See [Configure Access Control \(RFC\) \[page 387\]](#).
- You can generate self-signed certificates for CA and system certificate so that you can setup demo scenarios with principal propagation without the need of using lengthy openSSL or keytool command sequences. See [Configure a CA Certificate for Principal Propagation \[page 354\]](#) and [Initial Configuration \(HTTP\) \[page 329\]](#).
- A first set of monitoring HTTP APIs have been introduced: The state of all subaccount connections, a back-end connection monitor and a performance overview monitor. See [Monitoring APIs \[page 534\]](#).
- On Windows platforms, Cloud Connector 2.10 now requires Visual Studio 2013 runtime libraries.

Fix

Cloud Connector

- There is no longer a bottleneck that could lengthen the processing times of requests to exposed back-end systems, after many hours under high load when using principal propagation, connection pooling, and many concurrent sessions.
- Session management is no longer terminating early active sessions in principal propagation scenarios.
- On Windows 10 hardware metering in virtualized environments shows hard disk and CPU data.

11 May 2017 - Connectivity

New

In case the remote server supports only TLS 1.2, use this property to ensure that your scenario will work. As TLS 1.2 is more secure than TLS 1.1, the default version used by HTTP destinations, consider switching to TLS 1.2.

30 March 2017 - Connectivity

Enhancement

The release of SAP Cloud Platform Cloud Connector 2.9.1 includes the following improvements:

- UI renovations based on collected customer feedback. The changes include rounding offs, fixes of wrong/odd behaviors, and adjustments of controls. For example, in some places tables were replaced by `sap.ui.table.Table` for better experience with many entries.
- You can trigger the creation of a thread dump from the [Log and Trace Files](#) view.
- The connection monitor graphic for idle connections was made easier to understand.

Fix

- When configuring authentication for LDAP, the alternate host settings are no longer ignored.
- The email configuration for alerts is processing correctly the user and password for access to the email server.
- Some servers used to fail to process HTTP requests when using the HTTP proxy approach ([HTTP Proxy for On-Premise Connectivity](#)) on the SAP Cloud Platform side.
- A bottleneck was removed that could lengthen the processing times of requests to exposed back-end systems under high load when using principal propagation.
- The Cloud Connector accepts passwords that contain the '\$' character when using authentication-mode password.

16 March 2017 - Connectivity

Enhancement

Update of JCo runtime for SAP Cloud Platform. See [Connectivity \[page 3\]](#).

Fix

A `java.lang.NullPointerException` might have occurred when using a `JCoRepository` instance in roundtrip optimization mode (will be used if the JCo property `jco.use_repository_roundtrip_optimization` was set to 1 at its creation time). The `NullPointerException` was either thrown when trying to execute a `JCoFunction` object, which has been created by such a repository instance, or even earlier when querying the meta data for a `JCoFunction`, `JCoFunctionTemplate` or a `JCoRecordMetaData` object from an AS ABAP back-end system. Only certain complex data structures and table parameter definitions were affected by this bug.

Older Release Notes

- [2016](#)
- [2015](#)
- [2014](#)

- 2013 

1.1.3 Initial Setup

Manage destinations and authentication for applications in the Cloud Foundry environment.

Task	Description
Managing Destinations [page 39]	Manage HTTP destinations for Cloud Foundry applications in the SAP BTP cockpit.
HTTP Destinations [page 64]	You can choose from a broad range of authentication types for HTTP destinations, to meet the requirements of your specific communication scenario.
RFC Destinations [page 111]	Use RFC destinations to communicate with an on-premise ABAP system via the RFC protocol.
Principal Propagation [page 122]	Use principal propagation (user propagation) to securely forward cloud users to a back-end system (single sign-on).
Set up Trust Between Systems [page 126]	Download and configure X.509 certificates as a prerequisite for user propagation from the Cloud Foundry environment to the Neo environment or to a remote system outside SAP BTP, like S/4HANA Cloud, C4C, Success Factors, and others.
Multitenancy in the Connectivity Service [page 157]	Manage destinations for multitenancy-enabled applications that require a connection to a remote service or on-premise application.
Create and Bind a Connectivity Service Instance [page 160]	To use the Connectivity service in your application, you must first create and bind an instance of the service.
Create and Bind a Destination Service Instance [page 162]	To use the Destination service in your application, you must first create and bind an instance of the service.

1.1.3.1 Managing Destinations

To manage destinations for your application, choose a procedure that fits best your requirements.

There are various ways to manage destinations. Each method is characterized by different prerequisites and limitations. Before choosing a method, you should evaluate them and decide which one is the most appropriate for your particular scenario. The following table compares the available methods:

Method	Create from Scratch	Create from Template	Create from File (Import)	Save to File (Export)	Update	Delete	Clone	Check Connection
Using the Destinations Editor in the Cockpit [page 40]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Destination Service REST API [page 59]	Yes	No	Yes	Yes	Yes	Yes	No	No
Create Destinations Using the MTA Descriptor [page 60]	Yes	Limited	No	No	Yes	No	No	No
Create Destinations on Service Instance Creation [page 64]	Yes	No	No	No	Yes	No	No	No

1.1.3.1.1 Using the Destinations Editor in the Cockpit

Use the *Destinations* editor in the SAP BTP cockpit to configure HTTP, RFC or mail destinations in the Cloud Foundry environment.

The *Destinations* editor lets you manage destinations on subaccount or service instance level.

You can use a destination to:

- Connect your Cloud Foundry application to the **Internet** (via HTTP), as well as to an **on-premise system** (via HTTP or RFC).
- Send and retrieve e-mails, configuring a mail destination.
- Create a destination for subscription-based scenarios, pointing to your service instance. For more information, see [Destinations Pointing to Service Instances \[page 52\]](#).

Prerequisites

1. You are logged into the SAP BTP cockpit.
2. You have the required authorizations. See [User Roles \[page 9\]](#).

3. Make sure the following is fulfilled:

- Service instance level – you must have created a Destination service instance, see [Create and Bind a Destination Service Instance \[page 162\]](#).
- Subaccount level – no specific prerequisites.

For more information, see [Access the Destinations Editor \[page 41\]](#).

Restrictions

- A destination name must be unique for the current application. It must contain only alphanumeric characters, underscores, and dashes. The maximum length is 200 characters.
- The currently supported destination types are **HTTP**, **RFC** and **MAIL**.
 - [HTTP Destinations \[page 64\]](#) - provide data communication via the HTTP protocol and are used for both Internet and on-premise connections.
 - [RFC Destinations \[page 111\]](#) - make connections to ABAP on-premise systems via RFC protocol using the Java Connector (JCo) as API.
 - Mail destinations - specify an e-mail provider for sending and retrieving e-mails.

Tasks

- [Create Destinations from Scratch \[page 43\]](#)
- [Create Destinations from a Template \[page 51\]](#)
- [Check the Availability of a Destination \[page 53\]](#)
- [Clone Destinations \[page 55\]](#)
- [Edit and Delete Destinations \[page 55\]](#)
- [Use Destination Certificates \[page 56\]](#)
- [Import Destinations \[page 58\]](#)
- [Export Destinations \[page 59\]](#)

Related Information

[Destination Examples \[page 48\]](#)

1.1.3.1.1.1 Access the Destinations Editor

Access the Destinations Editor in the SAP BTP cockpit to create and manage destinations in the Cloud Foundry environment.

You can edit destinations at two different levels:

- Subaccount level
- Service instance level

On subaccount level, you can specify a destination for the entire subaccount, defining the used communication protocol and more properties, like authentication method, proxy type and URL.

On service instance level, you can reuse this destination for a specific space and adjust the URL if required. You can also create a new destination only on service instance level that is specific to the selected service instance and its assigned applications.

Prerequisites

- You are logged into the SAP BTP cockpit.
- You have the required authorizations. See [User Roles \[page 9\]](#).

Procedure

Access on Subaccount Level

1. In the cockpit, select your [Global Account](#) and your subaccount name from the [Subaccount](#) menu in the breadcrumbs.
2. From the left-side panel, choose [Connectivity](#) [Destinations](#).

Access on Service Instance Level

Note

To perform these steps, you must have created a Destination service instance in your space, see [Create and Bind a Destination Service Instance \[page 162\]](#). On service instance level, you can set destinations only for Destination service instances.

1. In the cockpit, choose your [Global Account](#) from the [Region Overview](#) and select a [Subaccount](#).

Note

The Cloud Foundry Organization must be enabled.

2. From the [Spaces](#) section, select a space name.
3. From the left-side menu, choose [Services](#) [Service Instances](#).
4. Choose the [Actions](#) icon for a Destination service instance and select [View Dashboard](#).

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has 'Service Instances' selected under 'Services'. The main area is titled 'Space: dev - Service Instances' and lists one instance named 'testfm'. A context menu is open for this instance, with the 'View Dashboard' option highlighted.

- On the *Destinations* screen, you can create new destinations or edit existing ones.

The screenshot shows the SAP BTP Cockpit interface with 'Destinations' selected in the sidebar. The main area is titled 'Service Instance: testfm2 - Destinations' and shows a table with no destinations defined. The 'New Destination' button is highlighted.

See also section *Create and Bind a Service Instance from the Cockpit* in [Create and Bind a Destination Service Instance \[page 162\]](#).

Related Information

- [Create Destinations from Scratch \[page 43\]](#)
- [Create Destinations from a Template \[page 51\]](#)
- [Check the Availability of a Destination \[page 53\]](#)
- [Clone Destinations \[page 55\]](#)
- [Edit and Delete Destinations \[page 55\]](#)
- [Use Destination Certificates \[page 56\]](#)
- [Import Destinations \[page 58\]](#)
- [Export Destinations \[page 59\]](#)

1.1.3.1.1.2 Create Destinations from Scratch

Use the *Destinations* editor in the SAP BTP cockpit to configure destinations from scratch.

Configuring destinations from scratch provides the complete set of editing functions. While this requires deeper technical knowledge of the scenario and the required connection configuration, it is the most flexible procedure and lets you create any type of supported destination.

To [Create Destinations from a Template \[page 51\]](#), in contrast, you do not need this deeper knowledge for configuration, but editing options are limited by the available templates.

Related Information

[Create HTTP Destinations \[page 44\]](#)

[Create RFC Destinations \[page 45\]](#)

[Create Mail Destinations \[page 47\]](#)

[Destination Examples \[page 48\]](#)

1.1.3.1.1.2.1 Create HTTP Destinations

Create HTTP destinations in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have logged into the cockpit and opened the *Destinations* editor.

Procedure

1. Choose *New Destination*.

i Note

In section **Destination Configuration**, do not change the default tab *Blank Template*, unless you want to create a destination for a specific service instance in a subscription-based scenario. For more information, see [Destinations Pointing to Service Instances \[page 52\]](#).

2. Enter a destination name.
3. From the *<Type>* dropdown menu, choose **HTTP**.
4. The *<Description>* field is optional.
5. Specify the destination URL.
6. From the *<Proxy Type>* dropdown box, select **Internet** or **OnPremise**, depending on the connection you need to provide for your application.

i Note

For more information, see also [HTTP Destinations \[page 64\]](#).

- From the <Authentication> dropdown box, select the authentication type you need for the connection.

For details, see [HTTP Destinations \[page 64\]](#).

i Note

If you set an **HTTPS** destination, you need to also add a Trust Store. For more information, see [Use Destination Certificates \[page 56\]](#).

- (Optional) If you are using more than one Cloud Connector for your subaccount, you must enter the <Location ID> of the target Cloud Connector.
See also [Managing Subaccounts \[page 338\]](#) (section **Procedure**, step 4).
- (Optional) You can enter additional properties.
 - In the *Additional Properties* panel, choose *New Property*.
 - Enter a key (name) or choose one from the dropdown menu and specify a value for the property. You can add as many properties as you need.
 - To delete a property, choose the  button next to it.

i Note

For a detailed description of specific properties for SAP Business Application Studio (formerly known as SAP Web IDE), see [Connecting to External Systems](#).

- When you are ready, choose the *Save* button.

Related Information

[Edit and Delete Destinations \[page 55\]](#)

[Destination Examples \[page 48\]](#)

1.1.3.1.1.2.2 Create RFC Destinations

How to create RFC destinations in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have logged into the cockpit and opened the *Destinations* editor.

Procedure

1. Choose [New Destination](#).

i Note

In section **Destination Configuration**, do not change the default tab **Blank Template**. Tab **Service Instance** only applies for HTTP destinations.

2. Enter a destination name.
3. From the **<Type>** dropdown menu, choose **RFC**.
4. The **<Description>** field is optional.
5. From the **<Proxy Type>** dropdown box, select **Internet** or **OnPremise**, depending on the connection you need to provide for your application.

i Note

Using **<Proxy Type> Internet**, you can connect your application to any target service that is exposed to the Internet. **<Proxy Type> OnPremise** requires the Cloud Connector to access resources within your on-premise network.

6. Enter credentials for **<User>** and **<Password>**.
7. (Optional) Enter an **<Alias User>** name. See also [User Logon Properties \[page 112\]](#).
8. (Optional) Enter credentials for **<Repository User>** and **<Repository Password>**, if you want to use a different user for repository lookups. See also [Repository Configuration \[page 116\]](#).
9. (Optional) If you are using more than one Cloud Connector for your subaccount, you must enter the **<Location ID>** of the target Cloud Connector.
See also [Managing Subaccounts \[page 338\]](#) (section **Procedure**, step 4).
10. Depending on the proxy type of your RFC destination, specify at least the following JCo properties in section **Additional Properties**.
 - a. In the **Additional Properties** panel, choose [New Property](#).
 - b. Add each required property from the dropdown menu and specify its value:

Proxy Type	Property	Description
OnPremise	Load Balancing Connections jco.client.r3name jco.client.mshost jco.client.group	Three-letter system ID of your backend ABAP system (as configured in the Cloud Connector). Message server host (as configured in the Cloud Connector). (Optional) The group of application servers that is used (logon group). If not specified, the group PUBLIC is used.

Proxy Type	Property	Description
	jco.client.client	Three-digit ABAP client number (defines the client of the target ABAP system).
Direct Connections		
	jco.client.ashost	Application server name of your target ABAP system (as configured in the Cloud Connector).
	jco.client.sysnr	Instance number of the application server (as configured in the Cloud Connector).
	jco.client.client	Three-digit ABAP client number (defines the client of the target ABAP system).
Internet	jco.client.wshost	WebSocket RFC server host on which the target ABAP system is running. The system must be exposed to the Internet.
	jco.client.wsport	WebSocket RFC server port on which the target ABAP system is listening.
	jco.client.client	Three-digit ABAP client number (defines the client of the target ABAP system).

For a detailed description of RFC-specific properties (JCo properties), see [RFC Destinations \[page 111\]](#).

11. Press **Save**.

Related Information

[Edit and Delete Destinations \[page 55\]](#)

[Destination Examples \[page 48\]](#)

[Cloud Connector \[page 276\]](#)

1.1.3.1.1.2.3 Create Mail Destinations

Create mail destinations in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have logged into the cockpit and opened the *Destinations* editor.

Procedure

1. Choose [New Destination](#).

i Note

In section **Destination Configuration**, do not change the default tab **Blank Template**. Tab **Service Instance** only applies for HTTP destinations.

2. Enter a destination name.
3. From the **Type** dropdown menu, choose **MAIL**.
4. The **Description** field is optional.
5. From the **Proxy Type** dropdown box, select **Internet** or **OnPremise**, depending on the connection you need to provide for your application.

i Note

To access a mail server located in your own network (via Cloud Connector), choose **OnPremise**. To access an external mail server, choose **Internet**.

6. Optional: You can enter additional properties.
 - a. In the **Additional Properties** panel, choose [New Property](#).
 - b. Enter a key (name) or choose one from the dropdown menu and specify a value for the property. You can add as many properties as you need. Each key of an additional property must start with "mail".
 - c. To delete a property, choose the  button next to it.
7. When you are ready, choose the [Save](#) button.

Related Information

[Edit and Delete Destinations \[page 55\]](#)

[Destination Examples \[page 48\]](#)

[Cloud Connector \[page 276\]](#)

1.1.3.1.1.2.4 Destination Examples

Find configuration examples for HTTP and RFC destinations in the Cloud Foundry environment, using different authentication types.

Content

[HTTP Destination \(Internet, Client Certificate Authentication\) \[page 49\]](#)

[HTTP Destination \(Internet, OAuth2SAMLBearerAssertion\) \[page 49\]](#)

[HTTP Destination \(On-Premise\) \[page 50\]](#)

[RFC Destination \[page 50\]](#)

[Mail Destination \(Internet\) \[page 51\]](#)

[Mail Destination \(On-Premise\) \[page 51\]](#)

Example: HTTP Destination (Internet, Client Certificate Authentication)

The screenshot shows the SAP BTP Connectivity interface for configuring an HTTP destination. The destination is named 'MyInternet' and is of type 'HTTP'. It is configured to proxy to 'https://google.com' and use 'ClientCertificateAuthentication'. The key store location is 'MyKeyStore.jks'. In the 'Additional Properties' section, there is a checkbox for 'Use default JDK truststore' which is unchecked. The 'Trust Store Location' is set to 'MyTrustStore.jks'. The 'Upload and Delete Certificates' section shows a password field for the key store. At the bottom, there are 'Save' and 'Cancel' buttons.

[Back to Content \[page 48\]](#)

Example: HTTP Destination (Internet, OAuth2SAMLBearerAssertion)

The screenshot shows the SAP BTP Connectivity interface for configuring an HTTP destination. The destination is named 'MyDestination' and is of type 'HTTP'. It is configured to proxy to 'https://sample.server.com' and use 'OAuth2SAMLBearerAssertion' authentication. The key store location is 'MyKeyStore.jks'. In the 'Additional Properties' section, there is a checked checkbox for 'Use default JDK truststore'. The 'Token Service URL Type' is set to 'Dedicated'. The token service user and password fields are both set to '<optional>'. At the bottom, there are 'Save' and 'Cancel' buttons.

[Back to Content \[page 48\]](#)

Example: HTTP Destination (On-Premise)

The screenshot shows the SAP BTP Connectivity interface for creating a new destination. The top navigation bar includes links for 'New Destination', 'Import Destination', 'Certificates', 'Download Trust', and 'Renew Trust'. The main area has tabs for 'Type' (selected), 'Name', 'Basic Properties', and 'Actions'. A message 'No destinations defined' is displayed. Under 'Destination Configuration', there is a form with the following fields:

▪Name:	MyOnPremiseDestination	Additional Properties	New Property	
Type:	HTTP	▼		
Description:	test			
Location ID:	<optional>			
▪URL:	https://mycompany.corp:443			
Proxy Type:	OnPremise	▼		
Authentication:	BasicAuthentication	▼		
▪User:	SomeUser			
Password:	*****			

At the bottom left are 'Save' and 'Cancel' buttons.

Back to [Content \[page 48\]](#)

Example: RFC Destination

The screenshot shows the SAP BTP Connectivity interface for creating a new destination. The top navigation bar includes links for 'New Destination', 'Import Destination', 'Certificates', 'Download Trust', and 'Renew Trust'. The main area has tabs for 'Type' (selected), 'Name', 'Basic Properties', and 'Actions'. A message 'No destinations defined' is displayed. Under 'Destination Configuration', there is a form with the following fields:

▪Name:	MyJCoSystem	Additional Properties	New Property
Type:	RFC	▼	
Description:			
Location ID:			
User:	MyJCoUser	jco.client.lang	EN
Password:	*****	jco.client.ashost	abapserver.hana.cloud
Repository User:		jco.client.client	000
Repository Password:		jco.client.sysnr	42

At the bottom left are 'Save' and 'Cancel' buttons.

The following main properties correspond to the relevant additional properties:

User → jco.client.user

Password → jco.client.passwd

Repository password → jco.destination.repository.passwd

i Note

For security reasons, do not use these additional properties but use the corresponding main properties' fields.

Back to [Content \[page 48\]](#)

Example: Mail Destination (Internet)

Destination Configuration

Name:*	mail_internet
Type:	MAIL
Description:	
Proxy Type:	Internet
User:	user
Password:	*****

Additional Properties	
mail.smtp.host	my-mail-server.com
mail.transport	smtp

New Property

Back to [Content \[page 48\]](#)

Example: Mail Destination (On-Premise)

Destination Configuration

Name:*	mail_onprem
Type:	MAIL
Description:	
Proxy Type:	OnPremise
User:	user
Password:	*****
Location ID:	Paris

Additional Properties	
mail.smtp.host	my-mail-server.com
mail.transport	smtp

New Property

Back to [Content \[page 48\]](#)

Related Information

[HTTP Destinations \[page 64\]](#)

[RFC Destinations \[page 111\]](#)

1.1.3.1.1.3 Create Destinations from a Template

Use a template to configure destinations with scenario-specific input data in the SAP BTP cockpit.

If you want to create several destinations for a common scenario, you can use a template that provides the scenario-specific input data. The Destination service uses the template to configure the destinations accordingly.

Currently, the following template is available for destination configuration:

[Destinations Pointing to Service Instances \[page 52\]](#)

1.1.3.1.3.1 Destinations Pointing to Service Instances

Create a destination for subscription-based scenarios that points to your service instance.

i Note

This feature is applicable for a selected set of the most commonly used services (from a Destination service perspective). If you would like to use this feature for a service which is not yet supported, let us know by opening a support ticket, see [Connectivity Support \[page 644\]](#).

In the meantime, you can follow the steps described in [Create Destinations from Scratch \[page 43\]](#).

Usually, in the Cloud Foundry environment, you consume service instances by binding them to your applications. However, in subscription-based scenarios this is not always possible. If you have purchased a subscription to an SaaS application that runs in a provider's subaccount, you cannot bind your service instance to this application.

In this case, you must create a destination that points to your service instance. Applications can consume this destination through a subscription to gain access to your service instance.

If you create such a destination from scratch, you must provide a service key for your instance, look up the credentials, and enter these values in the newly created destination.

Using the *Destinations Pointing to Service Instances* template, you only have to select the corresponding service instance.

i Note

This procedure only applies for HTTP destinations on subaccount level.

Prerequisites

- You have a service instance which you want to make accessible to applications you are subscribed to.
- You have the Space Developer role in the space where this service instance resides.
- You have logged in to the cockpit and opened the *Destinations* editor on subaccount level. See [Access the Destinations Editor \[page 41\]](#).

Procedure

1. Choose *New Destination*.
2. Select the tab *Service Instance* in the **Destination Configuration** section.
3. In the <Service Instance> dropdown list, you find all the service instances, grouped by space, where you have the role Space Developer.

4. Select a service instance.
5. Give the destination configuration a name and, optionally, a description.
6. (Optional) You can specify additional properties.
7. Choose *Next*.
A service key for that service instance is automatically generated, using the naming convention <service_instance_name>-service-key. If the key name already exists, it is reused. A new destination with pre-filled fields is previewed, using the given service instance data. Do not change the values of these fields.
8. If you want to create a destination with these values, choose *Save*. Otherwise, choose *Cancel*.

Result

You have a destination pointing to your service instance. If you delete this service instance or its service key, the destination stops working.

⚠ Caution

If you delete this service instance or its service key, the destination will stop working.

1.1.3.1.1.4 Check the Availability of a Destination

How to check the availability of a destination in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have logged into the cockpit and opened the *Destinations* editor.

Context

You can use the *Check Connection* button in the *Destinations* editor of the cockpit to verify if the [URL](#) configured for an HTTP Destination is reachable and if the connection to the specified system is possible.

ℹ Note

This check is available with *Cloud Connector version 2.7.1 or higher*.

For each destination, the check button is available in the destination detail view and in the destination overview list (icon *Check availability of destination connection* in section *Actions*).

i Note

The check does not guarantee that the target system or service is operational. It only verifies if a connection is possible.

This check is supported for destinations with `<Proxy Type> Internet` and `OnPremise`:

- For `Internet` destinations:
 - If the check receives an **HTTP status code above or equal to 500** from the targeted URL, the check is considered **failed**.
 - Every **HTTP status code below 500** is treated as **successful**.
- For `OnPremise` destinations:
 - If the targeted backend is reached and returns an **HTTP status code below 500** the check is considered **successful**.

Error Messages for OnPremise Destinations

Error Message	Reason	Action
<i>Backend status could not be determined.</i>	<ul style="list-style-type: none">● The Cloud Connector version is less than 2.7.1.● The Cloud Connector is not connected to the subaccount.● The backend returns an HTTP status code above or equal to 500 (server error).● The Cloud Connector is not configured properly.	<ul style="list-style-type: none">● Upgrade the Cloud Connector to version 2.7.1 or higher.● Connect the Cloud Connector to the corresponding subaccount.● Check the server status (availability) of the backend system.● Check the basic Cloud Connector configuration steps: Initial Configuration [page 322]
<i>Backend is not available in the list of defined system mappings in Cloud Connector.</i>	The Cloud Connector is not configured properly.	<p>Check the basic Cloud Connector configuration steps: Initial Configuration [page 322]</p>
<i>Resource is not accessible in Cloud Connector or backend is not reachable.</i>	The Cloud Connector is not configured properly.	<p>Check the basic Cloud Connector configuration steps: Initial Configuration [page 322]</p>
<i>Backend is not reachable from Cloud Connector.</i>	Cloud connector configuration is ok but the backend is not reachable.	Check the backend (server) availability.

1.1.3.1.1.5 Clone Destinations

How to clone destinations in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have previously created or imported an HTTP destination in the *Destinations* editor of the cockpit.

Procedure

1. In the *Destinations* editor, go to the existing destination which you want to clone.
2. Choose the  icon.
3. The editor automatically creates and opens a new destination that contains all the properties of the selected one.
4. You can modify some parameters if you need.
5. When you are ready, choose the *Save* button.

Related Information

[Export Destinations \[page 59\]](#)

[Destination Examples \[page 48\]](#)

1.1.3.1.1.6 Edit and Delete Destinations

How to edit and delete destinations in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have previously created or imported an HTTP destination in the *Destinations* editor of the cockpit.

Procedure

- Edit a destination:
 1. To edit a created or imported destination, choose the  button.
 2. You can edit the main parameters as well as the additional properties of a destination.
 3. Choose the **Save** button. The changes will take effect in up to five minutes.

→ Tip

For complete consistency, we recommend that you first stop your application, then apply your destination changes, and then start again the application. Also, bear in mind that these steps will cause application downtime.

- Delete a destination:

To remove an existing destination, choose the  button. The changes will take effect in up to five minutes.

Related Information

[Export Destinations \[page 59\]](#)

[Destination Examples \[page 48\]](#)

1.1.3.1.1.7 Use Destination Certificates

Maintain trust store and key store certificates in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have logged into the cockpit and opened the *Destinations* editor. For more information, see [Access the Destinations Editor \[page 41\]](#).

Context

You can upload, add and delete certificates for your connectivity destinations. Bear in mind that:

- You can use JKS, PFX, PEM and P12 files.
- You can add certificates only for **HTTPS** destinations. The trust store can be used for all authentication types. The key store is available only for ClientCertificateAuthentication and OAuth2SAMLBearerAssertion.
- An uploaded certificate file should contain the entire certificate chain.

Procedure

Uploading Certificates

1. Choose the *Certificates* button.
2. Choose *Upload Certificate*.
3. Browse to the certificate file you need to upload.
The certificate file is added.

i Note

You can upload a certificate during creation or editing of a destination, by clicking the *Upload and Delete Certificates* link.

Adding Certificates to Destinations

1. Create a new destination, or open an existing one for editing.
2. In the *URL* field, enter an HTTPS address.
3. You can use the default JDK trust store or select another one from the dropdown menu. If the menu is empty, you can upload a certificate on the fly. To omit this property, you can set `TrustAll=true`.
4. If you choose *Authentication = ClientCertificateAuthentication*, you need to also provide a key store.

Deleting Certificates

1. Choose the *Certificates* button or click the *Upload and Delete Certificates* link.
2. Select the certificate you want to remove and choose *Delete Selected*.
3. Upload another certificate, or close the *Certificates* window.

More Information

[Create HTTP Destinations \[page 44\]](#)

[Edit and Delete Destinations \[page 55\]](#)

[Import Destinations \[page 58\]](#)

1.1.3.1.1.8 Import Destinations

How to import destinations in the *Destinations* editor (SAP BTP cockpit).

Prerequisites

You have previously created an HTTP destination.

i Note

The *Destinations* editor allows importing destination files with extension `.props`, `.properties`, `.jks`, and `.txt`, as well as files with no extension. Destination files must be encoded in **ISO 8859-1** character encoding.

Procedure

1. Log into the cockpit and open the *Destinations* editor.
2. Choose *Import from File*.
3. Browse to a configuration file that contains destination configuration.
 - If the configuration file contains valid data, it is displayed in the *Destinations* editor with no errors. The *Save* button is enabled so that you can successfully save the imported destination.
 - If the configuration file contains invalid properties or values, under the relevant fields in the *Destinations* editor are displayed error messages in red which prompt you to correct them accordingly.

Related Information

[Edit and Delete Destinations \[page 55\]](#)

[Destination Examples \[page 48\]](#)

1.1.3.1.1.9 Export Destinations

Export destinations from the *Destinations* editor in the SAP BTP cockpit to backup or reuse a destination configuration.

Prerequisites

You have created a destination in the *Destinations* editor.

Procedure

1. Log into the cockpit and open the *Destinations* editor.
2. Select a destination and choose the  button.
3. Browse to the location on your local file system where you want to save the new destination.
 - If the destination does not contain client certificate authentication, it is saved as a single configuration file.
 - If the destination provides client certificate data, it is saved as an archive, which contains the main configuration file and a JKS file.

Related Information

[Edit and Delete Destinations \[page 55\]](#)

[Destination Examples \[page 48\]](#)

1.1.3.1.2 Destination Service REST API

Destination service REST API specification for the SAP Cloud Foundry environment.

The Destination service provides a REST API that you can use to read and manage destinations and certificates on all available levels. This API is documented in the [SAP API Business Hub](#).

It shows all available endpoints, the supported operations, parameters, possible error cases and related status codes, etc. You can also execute requests using the credentials (for example, the service key) of your Destination service instance, see [Create and Bind a Destination Service Instance \[page 162\]](#).

1.1.3.1.3 Create Destinations Using the MTA Descriptor

Use the multitarget-application (MTA) descriptor to manage destinations for complex deployments.

Content

- [Concept \[page 60\]](#)
- [Content Deployment \[page 60\]](#)
- [Create a Destination on Service Instance Creation \[page 64\]](#)

Concept

When modeling a multitarget application (MTA), you can create and update destinations from your MTA descriptor.

For more information on MTA, see [Multitarget Applications in the Cloud Foundry Environment](#).

Back to [Content \[page 60\]](#)

Content Deployment

When modeling MTAs, you can configure content deployments (for more information, see [Content Deployment](#)). The Destinations service supports such content deployments, which lets you create or update destinations by modeling them in the MTA descriptor. Other operations, like deleting a destination, are not supported by this method.

Parameters

The parameters of the content deployment have the following structure:

```
content:  
  subaccount:  
    existing_destinations_policy: <policy> # optional, default value is "fail".  
    See "Existing destinations policy" for more details  
    destinations:  
      - <destination descriptor 1> # See "Modeling options" to learn about the  
        structure of this descriptor  
      ...  
      - <destination descriptor N> # See "Modeling options" to learn about the  
        structure of this descriptor  
    instance:  
      existing_destinations_policy: <policy> # optional, default value is "fail".  
      See "Existing destinations policy" for more details  
    destinations:
```

```
- <destination descriptor 1> # See "Modeling options" to learn about the  
structure of this descriptor  
...  
- <destination descriptor N> # See "Modeling options" to learn about the  
structure of this descriptor
```

i Note

Both the `subaccount` and `instance` sections are optional. They can both be present at the same time, or only one of them. They define the level on which the resulting destination is created.

Existing Destinations Policy

The `existing_destinations_policy` setting allows you to control what happens if a destination with the same name already exists. The possible values are:

- `fail`: Treat it as an error situation and fail the deployment. This is the default value of the setting.
- `ignore`: Keep what is currently saved in the Destination service, and skip deployment for this destination.
- `update`: Override what is currently saved in the Destination service.

Modeling Options

The `destinations` section represents an array of destination descriptors. Each of these array elements is converted to a destination and saved in the service on the respective level, based on the existing destination policy. The following options are available for modeling a destination descriptor via content deployment. They can be combined:

Destination Pointing to a Service Instance

This option lets you:

- Reference a service instance and a service key
- Specify a destination name
- Enter a description for the resulting destination (optional)
- Add additional properties and override default properties (optional)

As a result, a destination is created, based on the properties in the referenced service key.

i Note

This function is equivalent to the [Destinations Pointing to Service Instances \[page 52\]](#) template.

⚠ Caution

This feature is applicable for a selected set of the most commonly used services (from a Destination service perspective). If you would like to use this feature for a service which is not yet supported, let us know by opening a support ticket, see [Connectivity Support \[page 644\]](#).

In the meantime, you can follow the steps described in [Create Destinations from Scratch \[page 43\]](#).

The descriptor has the following structure:

```
Name: <name> # name of the destination
Description: <description> # optional, a description for the destination
Authentication: <auth> # optional for some services (the default is service
specific). Some services require the Authentication to be specified, like the
workflow service. The allowed authentication types are also service-specific
ServiceInstanceName: <instance name> # the name of the service instance to which
the destination will be created
ServiceKeyName: <service key name> # the service key of the instance targeted by
ServiceInstanceName which will be used as the source for the destination values
AdditionalProp1: value1 # optional
...
AdditionalPropN: valueN # optional
```

Destination Pointing to a Resource Protected by an XSUAA Service Instance

This option lets you:

- Reference a service instance and a service key
- Specify a destination name
- Enter a description for the resulting destination (optional)
- Specify the URL of the target resource
- Add additional properties and override default properties (optional)

As a result, a destination is created with the token service configuration based on the properties in the referenced service key, while the URL will be the one specified when modeling the destination.

The descriptor has the following structure:

```
Name: <name> # name of the destination
Description: <description> # optional, a description for the destination
Authentication: <auth> # optional for some services (the default is service
specific). Some services require the Authentication to be specified, like the
workflow service. The allowed authentication types are also service-specific
URL: <url> # the URL of the target resource
TokenServiceInstanceName: <instance name> # the name of the service instance
used for protecting the target resource
TokenServiceKeyName: <service key name> # the service key of the instance
targeted by ServiceInstanceName which will be used as the source for the token
service values in the destination
AdditionalProp1: value1 # optional
...
AdditionalPropN: valueN # optional
```

Example:

```
_schema-version: "3.2"
ID: example
version: 0.0.1
modules:
- name: myapp
  path: ./myapp
  type: javascript.nodejs
  requires:
  - name: xsuaa_service
  provides:
```

```

- name: myapp-route
  properties:
    url: ${default-url} #generated during deployment
- name: destination-content
  type: com.sap.application.content
  requires:
  - name: xsuaa_service
    parameters:
      service-key:
        name: xsuaa_service-key
- name: destination-service
  parameters:
    content-target: true
- name: myapp-route
  build-parameters:
    no-source: true
  parameters:
    content:
      subaccount:
        existing_destinations_policy: update
      destinations:
        - Name: myappOauth
          URL: ~{myapp-route/url}
          Authentication: OAuth2ClientCredentials
          TokenServiceInstanceName: xsuaa_service
          TokenServiceKeyName: xsuaa_service-key
          myAdditionalProp: myValue
        - Name: workflowOauthJwtBearer
          Authentication: OAuth2JWTBearer
          ServiceInstanceName: workflow_service
          ServiceKeyName: workflow_service-key
      instance:
        existing_destinations_policy: update
      destinations:
        - Name: workflowBasicAuthentication
          Authentication: BasicAuthentication
          ServiceInstanceName: workflow_service
          ServiceKeyName: workflow_service-key
          myAdditionalProp: myValue
  resources:
  - name: xsuaa_service
    type: org.cloudfoundry.managed-service
    parameters:
      service: xsuaa
      service-name: xsuaa_service
      service-plan: application
      config:
        xsappname: "myApp"
  - name: workflow_service
    type: org.cloudfoundry.managed-service
    parameters:
      service: workflow
      service-name: workflow_service
      service-plan: lite
  - name: destination-service
    type: org.cloudfoundry.managed-service
    parameters:
      service: destination
      service-name: my-destination-service
      service-plan: lite

```

[Back to Content \[page 60\]](#)

Create a Destination on Service Instance Creation

The MTA descriptor lets you create service instances and provide a JSON configuration for this operation. You can use this functionality to create a Destination service instance with a JSON, and include the required data to create or update destinations.

For more details, see [Use a Config.JSON to Create or Update a Destination Service Instance \[page 164\]](#).

Back to [Content \[page 60\]](#)

1.1.3.1.4 Create Destinations on Service Instance Creation

Use a JSON to create or update a destination when creating a Destination service instance.

When creating or updating a service instance of the Destination service, you can provide a JSON object with various configurations. One of the sections of this JSON lets you create or update destinations. Other operations, like deleting a destination, are not supported by this method.

For more information, see [Use a Config.JSON to Create or Update a Destination Service Instance \[page 164\]](#).

1.1.3.2 HTTP Destinations

Find information about HTTP destinations for Internet and on-premise connections (Cloud Foundry environment).

Destination Levels

The runtime tries to resolve a destination in the order: *Subaccount Level* → *Service Instance Level*.

Destinations for Subscribed Applications

In subscription-based scenarios, it is not always possible to consume a service instance by binding it to your application. In this case, you must create a destination pointing to your service instance. For more information, see [Destinations Pointing to Service Instances \[page 52\]](#).

Proxy Types

The proxy types supported by the Connectivity service are:

- **Internet** - The application can connect to an external REST or SOAP service on the Internet.
- **OnPremise** - The application can connect to an on-premise back-end system through the Cloud Connector.

The proxy type used for a destination is specified by the destination property `ProxyType`. The default value is **Internet**.

Proxy Settings for Your Local Runtime

If you work in your local development environment behind a proxy server and want to use a service from the Internet, you need to configure your proxy settings on JVM level. To do this, proceed as follows:

1. On the [Servers](#) view, double-click the added server and choose [Overview](#) to open the editor.
2. Click the [Open Launch Configuration](#) link.
3. Choose the [\(x\)=Arguments](#) tab page.
4. In the [VM Arguments](#) box, add the following row:

```
-Dhttp.proxyHost=yourproxyHost -Dhttp.proxyPort=yourProxyPort -  
Dhttps.proxyHost=yourproxyHost -Dhttps.proxyPort=yourProxyPort
```
5. Choose [OK](#).
6. Start or restart your SAP HANA Cloud local runtime.

Configuring Authentication

When creating an HTTP destination, you can use different authentication types for access control:

- [Server Certificate Authentication \[page 66\]](#)
- [Principal Propagation SSO Authentication for HTTP \[page 68\]](#)
- [OAuth SAML Bearer Assertion Authentication \[page 71\]](#)
- [Client Authentication Types for HTTP Destinations \[page 78\]](#)
- [OAuth Client Credentials Authentication \[page 81\]](#)
- [OAuth User Token Exchange Authentication \[page 87\]](#)
- [SAP Assertion SSO Authentication \[page 92\]](#)
- [OAuth Password Authentication \[page 97\]](#)
- [OAuth JWT Bearer Authentication \[page 101\]](#)
- [SAML Assertion Authentication \[page 106\]](#)

Related Information

[OAuth with X.509 Client Certificates \[page 110\]](#)

[Create HTTP Destinations \[page 44\]](#)

1.1.3.2.1 Server Certificate Authentication

Create and configure a *Server Certificate* destination for an application in the Cloud Foundry environment.

Context

The server certificate authentication is applicable for all client authentication types, described below.

i Note

TLS 1.2 became the default TLS version of HTTP destinations. If an HTTP destination is consumed by a java application the change will be effective after restart. All HTTP destinations that use the HTTPS protocol and have ProxyType=Internet can be affected. Previous TLS version can be used by configuring an additional property TLSVersion=TLSv1.0 or TLSVersion=TLSv1.1.

Properties

Property	Description
TLSVersion	Optional property. Can be used to specify the preferred TLS version to be used by the current destination. Since TLS 1.2 is not enabled by default on the older java versions this property can be used to configure TLS 1.2 in case this is required by the server configured in this destination. It is usable only in HTTP destinations. Example: <code>TLSVersion=TLSv1.2</code> .
TrustStoreLocation	Path to the JKS file which contains trusted certificates (Certificate Authorities) for authentication against a remote client. <ol style="list-style-type: none">When used in local environmentWhen used in cloud environment<ol style="list-style-type: none">The relative path to the JKS file. The root path is the server's location on the file system.The name of the JKS file.
TrustStorePassword	Password for the JKS trust store file. This property is mandatory in case TrustStoreLocation is used.

i Note

If the TrustStoreLocation property is not specified, the JDK trust store is used as a default trust store for the destination.

Property	Description
TrustAll	<p>If this property is set to TRUE in the destination, the server certificate will not be checked for SSL connections. It is intended for test scenarios only, and should not be used in production (since the SSL server certificate is not checked, the server is not authenticated). The possible values are TRUE or FALSE; the default value is FALSE (that is, if the property is not present at all).</p> <p>In case <code>TrustAll = TRUE</code>, the <code>TrustStoreLocation</code> property is ignored so you can omit it.</p> <p>In case <code><TrustAll> = FALSE</code>, the <code><TrustStoreLocation></code> property is mandatory to be used.</p>
HostnameVerifier	<p>Optional property. It has two values: Strict and BrowserCompatible. This property specifies how the server hostname matches the names stored inside the server's X.509 certificate. This verifying process is only applied if TLS or SSL protocols are used and is not applied if the <code>TrustAll</code> property is specified. The default value (used if no value is explicitly specified) is Strict.</p> <ul style="list-style-type: none"> • Strict <code>HostnameVerifier</code> works in the same way as Oracle Java 1.4, Oracle Java 5, and Oracle Java 6-rc. It is also similar to Microsoft Internet Explorer 6. This implementation appears to be compliant with RFC 2818 for dealing with wildcards. A wildcard such as "<code>*.foo.com</code>" matches only subdomains at the same level, for example "<code>a.foo.com</code>". It does not match deeper subdomains such as "<code>a.b.foo.com</code>". • BrowserCompatible <code>HostnameVerifier</code> works in the same way as Curl and Mozilla Firefox. The hostname must match either the first common name (CN) or any of the subject-alts. A wildcard can occur in the CN and in any of the subject-alts. <p>The only difference between BrowserCompatible and Strict is that a wildcard (such as "<code>.foo.com</code>") with BrowserCompatible matches all subdomains, including "<code>a.b.foo.com</code>".</p> <p>For more information about these Java classes, see Package <code>org.apache.http.conn.ssl</code>.</p> <p>In case <code><TrustAll> = TRUE</code>, the <code><HostnameVerifier></code> property is ignored so you can omit it.</p>

i Note

You can upload trust store JKS files using the same command as for uploading destination configuration property files. You only need to specify the JKS file instead of the destination configuration file.

i Note

Connections to remote services which require *Java Cryptography Extension (JCE) unlimited strength jurisdiction policy* are not supported.

Configuration

- [Managing Destinations \[page 39\]](#)

Related Information

[Client Authentication Types for HTTP Destinations \[page 78\]](#)

1.1.3.2.2 Principal Propagation SSO Authentication for HTTP

Forward the identity of a cloud user from a Cloud Foundry application to a backend system via HTTP to enable single sign-on (SSO).

Context

A *PrincipalPropagation* destination enables single sign-on (SSO) by forwarding the identity of a cloud user to the Cloud Connector, and from there to the target on-premise system. In this way, the cloud user's identity can be provided without manual logon.

i Note

This authentication type applies only for on-premise connectivity.

Configuration Steps

You can create and configure a *PrincipalPropagation* destination by using the properties listed below, and deploy it on SAP BTP. For more information, see [Managing Destinations \[page 39\]](#).

Properties

The following credentials need to be specified:

Property	Description
Name	Destination name. Must be unique for the destination level.

Property	Description
Type	Destination type. Use HTTP for all HTTP(S) destinations.
URL	Virtual URL of the protected on-premise application.
Authentication	Authentication type. Use PrincipalPropagation as value.
ProxyType	You can only use proxy type OnPremise .
CloudConnectorLocationId	As of Cloud Connector 2.9.0, you can connect multiple Cloud Connectors to a subaccount as long as their location ID is different. The location ID specifies the Cloud Connector over which the connection is opened. The default value is an empty string identifying the Cloud Connector that is connected without any location ID. This is also valid for all Cloud Connector versions prior to 2.9.0.
URL.headers.<header-key>	A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:
<div style="border: 1px solid #ccc; padding: 10px;"> <p>↳ Sample Code</p> <pre>{ ... "URL.headers.<header-key-1>" : "<header-value-1>", ... "URL.headers.<header-key-N>" : "<header-value-N>", }</pre> </div>	
<div style="border: 1px solid #ccc; padding: 10px;"> <p>ⓘ Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p> </div>	
URL.queries.<query-key>	A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:

Property	Description
	<p>↳ Sample Code</p> <pre>{ ... "URL.queries.<query-key-1>" : "<query-value-1>", ... "URL.queries.<query-key-N>" : "<query-value-N>", }</pre>
	<p>i Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>

Example

```
Name=OnPremiseDestination  
Type=HTTP  
URL= http://virtualhost:80  
Authentication=PrincipalPropagation  
ProxyType=OnPremise
```

Related Information

[Principal Propagation \[page 122\]](#)

1.1.3.2.3 OAuth SAML Bearer Assertion Authentication

Create and configure an *OAuth SAML Bearer Assertion* destination for an application in the Cloud Foundry environment.

Context

You can call an OAuth2-protected remote system/API and propagate a user ID to the remote system by using the `OAuth2SAMLBearerAssertion` authentication type. The Destination service provides functionality for automatic token retrieval and caching, by automating the construction and sending of the SAML assertion. This simplifies application development, leaving you with only constructing the request to the remote system by providing the token, which is fetched for you by the Destination service. For more information, see [User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#).

Properties

The table below lists the destination properties for *OAuth2SAMLBearerAssertion* authentication type. You can find the values for these properties in the provider-specific documentation of OAuth-protected services. Usually, only a subset of the optional properties is required by a particular service provider.

Property	Description
Required	
Name	Name of the destination. Must be unique for the destination level.
Type	Destination type. Choose <code>HTTP</code> for all HTTP(S) destinations.
URL	URL of the target endpoint.
ProxyType	Choose <code>Internet</code> . <code>OnPremise</code> is not supported for this authentication type.
Authentication	Authentication type. Use <code>OAuth2SAMLBearerAssertion</code> as value.

Property	Description
KeyStoreLocation	<p>Path to the JKS file that contains the client certificate(s) for authentication against a remote server.</p> <ol style="list-style-type: none"> 1. When used in local environment 2. When used in cloud environment <p>1. The relative path to the JKS file. The root path is the server's location on the file system. 2. The name of the JKS file.</p> <p>This property is optional. If not specified, the subaccount key pair is used.</p> <div style="background-color: #f0f0f0; padding: 10px;"> <p>i Note</p> <p>You can upload KeyStore JKS files using the same command for uploading destination configuration property file. You only need to specify the JKS file instead of the destination configuration file.</p> </div> <div style="background-color: #f0f0f0; padding: 10px;"> <p>i Note</p> <p>You can configure the keystore properties only in the Destination editor on subaccount level.</p> </div>
KeyStorePassword	The password for the key storage. This property is mandatory in case KeyStoreLocation is used.
audience	Intended audience for the assertion, which is verified by the OAuth authorization server. For more information, see SAML 2.0 Bearer Assertion Profiles for OAuth 2.0 .
clientKey	Key that identifies the consumer to the authorization server.

Property	Description
tokenServiceURL	<p>The URL of the token service, against which the token exchange is performed. Depending on the Token Service URL type, this property is interpreted in different ways during the automatic token retrieval:</p> <ul style="list-style-type: none"> For Dedicated, the token service URL is taken as is. For Common, the token service URL is searched for the tenant placeholder {tenant}. {tenant} is resolved as the subdomain of the subaccount on behalf of which the caller is performing the call. If the placeholder is not found, {tenant} is inserted as a subdomain of the token service URL. The subaccount subdomain is mandated during creation of the subaccount, see Create a Subaccount. <p>Examples of interpreting the token service URL for the token service URL type Common, if the call to the Destination service is on behalf of a subaccount subdomain with value mytenant:</p> <ul style="list-style-type: none"> https://authentication.us10.hana.ondemand.com/oauth/token → https://mytenant.authentication.us10.hana.ondemand.com/oauth/token https://{tenant}.authentication.us10.hana.ondemand.com/oauth/token → https://mytenant.authentication.us10.hana.ondemand.com/oauth/token https://authentication.myauthserver.com/tenant/{tenant}/oauth/token → https://authentication.myauthserver.com/tenant/mytenant/oauth/token https://oauth.{tenant}.myauthserver.com/token → https://oauth.mytenant.myauthserver.com/token
tokenServiceURLType	Either Dedicated - if the token service URL serves only a single tenant, or Common - if the token service URL serves multiple tenants.
tokenServiceUser	User for basic authentication to OAuth server (if required).
tokenServicePassword	Password for tokenServiceUser (if required).

Property	Description
(Deprecated) SystemUser	<p>User to be used when requesting an access token from the OAuth authorization server. If this property is not specified, the currently logged-in user is used.</p> <div style="border-left: 2px solid red; padding-left: 10px;"> <p>⚠ Caution</p> <p>This property is deprecated and will be removed soon. We recommend that you work on behalf of specific (named) users instead of working with a technical user. As an alternative for technical user communication, we strongly recommend that you use one of these authentication types:</p> <ul style="list-style-type: none"> • Basic Authentication (see Client Authentication Types for HTTP Destinations [page 78]) • Client Certificate Authentication (see Client Authentication Types for HTTP Destinations [page 78]) • OAuth Client Credentials Authentication [page 81] <p>To extend an OAuth access token's validity, consider using an OAuth refresh token.</p> </div>
authnContextClassRef	Value of the AuthnContextClassRef tag, which is part of generated OAuth2SAMLBearerAssertion authentication. For more information, see SAML 2.0 specification .
Additional	
nameQualifier	Security domain of the user for which access token is requested.
companyId	Company identifier.
assertionIssuer	Issuer of the SAML assertion.
nameIdFormat	Value of the NameIdFormat tag, which is part of generated OAuth2SAMLBearerAssertion authentication. For more information, see SAML 2.0 specification .
userIdSource	When this property is set, the generated SAML2 assertion uses the currently logged-in user as a value for the NameId tag. See User Propagation via SAML 2.0 Bearer Assertion Flow [page 202] .
scope	The value of the OAuth 2.0 scope parameter, expressed as a list of space-delimited, case-sensitive strings.

Property	Description
tokenServiceURL.headers.<header-key>	<p>A static key prefix used as a namespace grouping of the tokenServiceUrl's HTTP headers. Its values will be sent to the token service during token retrieval. For each HTTP header's key you must add a 'tokenServiceURL.headers' prefix separated by dot delimiter. For example:</p>
	<p>↳ Sample Code</p> <pre data-bbox="825 617 1318 875">{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-value-1>", ... "tokenServiceURL.headers.<header-key-N>" : "<header-value-N>", }</pre>
tokenServiceURL.queries.<query-key>	<p>A static key prefix used as a namespace grouping of tokenServiceUrl's query parameters. Its values will be sent to the token service during token retrieval. For each query parameter's key you must add a 'tokenServiceURL.queries' prefix separated by dot delimiter. For example:</p>
	<p>↳ Sample Code</p> <pre data-bbox="825 1246 1302 1504">{ ... "tokenServiceURL.queries.<query-key-1>" : "<query-value-1>", ... "tokenServiceURL.queries.<query-key-N>" : "<query-value-N>", }</pre>

Property	Description
URL.headers.<header-key>	A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:

↳ Sample Code

```
{  
    ...  
    "URL.headers.<header-key-1>" :  
    "<header-value-1>,  
    ...  
    "URL.headers.<header-key-N>":  
    "<header-value-N>",  
}
```

i Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

Property	Description
URL.queries.<query-key>	A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:
	<p>↳ Sample Code</p> <pre>{ ... "URL.queries.<query-key-1>" : "<query-value-1>", ... "URL.queries.<query-key-N>" : "<query-value-N>", }</pre>
	<p>i Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>
x_user_token.jwks	Base64-encoded JSON web key set, containing the signing keys which are used to validate the JWT provided in the X-User-Token header.
	For more information, see JWK Set Format .
x_user_token.jwks_uri	URI of the JSON web key set, containing the signing keys which are used to validate the JWT provided in the X-User-Token header.
	For more information, see OpenID Connect Discovery .

Example

The connectivity destination below provides HTTP access to the OData API of the SuccessFactors Jam.

```
URL=https://demo.sapjam.com/OData/OData.svc
Name=sap_jam_odata
TrustAll=true
ProxyType=Internet
Type=HTTP
Authentication=OAuth2SAMLBearerAssertion
tokenServiceURL=https://demo.sapjam.com/api/v1/auth/token
clientKey=<unique_generated_string>
```

```
audience=cubetree.com  
nameQualifier=www.successfactors.com  
apiKey=<apiKey>
```

The response for "find destination" contains an authTokens object in the format given below. For more information on the fields in authTokens, see "["Find Destination" Response Structure \[page 199\]](#)".

↳ Sample Code

```
"authTokens": [  
    {  
        "type": "Bearer",  
        "value": "eyJhbGciOiJSUzI1NiIsInR5cC...",  
        "http_header": {  
            "key": "Authorization",  
            "value": "Bearer eyJhbGciOiJSUzI1NiIsInR5cC..."  
        }  
    }  
]
```

Related Information

[Create HTTP Destinations \[page 44\]](#)

[Destination Examples \[page 48\]](#)

[Exchanging User JWTs via OAuth2UserTokenExchange Destinations \[page 207\]](#)

1.1.3.2.4 Client Authentication Types for HTTP Destinations

Find details about client authentication types for HTTP destinations in the Cloud Foundry environment.

Context

This section lists the supported client authentication types and the relevant supported properties.

No Authentication

This is used for destinations that refer to a service on the Internet or an on-premise system that does not require authentication. The relevant property value is:

Authentication=**NoAuthentication**

i Note

When a destination is using HTTPS protocol to connect to a Web resource, the JDK truststore is used as truststore for the destination.

Basic Authentication

This is used for destinations that refer to a service on the Internet or an on-premise system that requires basic authentication. The relevant property value is:

Authentication=**BasicAuthentication**

The following credentials need to be specified:

⚠ Caution

Do not use your *own personal credentials* in the <User> and <Password> fields. Always use a *technical user* instead.

Property	Description
User	User name of the technical user to be used.
Password	Password of the technical user to be used.
Preemptive	If this property is not set or is set to TRUE (that is, the default behavior is to use preemptive sending), the authentication token is sent preemptively. Otherwise, it relies on the challenge from the server (401 HTTP code). The default value (used if no value is explicitly specified) is TRUE . For more information about preemptiveness, see http://tools.ietf.org/html/rfc2617#section-3.3 .

i Note

When a destination is using the HTTPS protocol to connect to a Web resource, the JDK truststore is used as truststore for the destination.

i Note

Basic Authentication and **No Authentication** can be used in combination with **ProxyType=OnPremise**. In this case, also the **CloudConnectorLocationId** property can be specified. As of Cloud Connector 2.9.0, you can connect multiple Cloud Connectors to a subaccount as long as their location ID is different. The value defines the location ID identifying the Cloud Connector over which the connection shall be opened. The default value is the empty string identifying the Cloud Connector that is connected without any location ID. This is also the case for all Cloud Connector versions prior to 2.9.0.

The response for "find destination" contains an authTokens object in the format given below. For more information on the fields in authTokens, see "["Find Destination" Response Structure \[page 199\]](#)".

↳ Sample Code

```
"authTokens": [
  {
    "type": "Basic",
    "value": "dGVzdDpwYXNzMTIzNDU=",
    "http_header": {
      "key": "Authorization",
      "value": "Basic dGVzdDpwYXNzMTIzNDU="
    }
  }
]
```

Client Certificate Authentication

This is used for destinations that refer to a service on the Internet. The relevant property value is:

Authentication=[ClientCertificateAuthentication](#)

The following credentials need to be specified:

Property	Description
KeyStore.Source	Optional. Specifies the storage location of the certificate to be used by the client. Supported values are: <ul style="list-style-type: none">• ClientProvided: The key store is managed by the client (the application itself).• DestinationService: The key store is managed by the Destination service. If the property is not set, the key store is managed by the Destination service (default).
KeyStoreLocation	The name of the key store file that contains the client certificate(s) for client certificate authentication against a remote server. This property is optional if KeyStore.Source is set to ClientProvided.
KeyStorePassword	Password for the key store file specified by KeyStoreLocation. This property is optional if KeyStoreLocation is used in combination with KeyStore.Source, and KeyStore.Source is set to ClientProvided.

i Note

You can upload KeyStore JKS files using the same command for uploading destination configuration property file. You only need to specify the JKS file instead of the destination configuration file.

Configuration

- [Managing Destinations \[page 39\]](#)

Related Information

[Server Certificate Authentication \[page 66\]](#)

1.1.3.2.5 OAuth Client Credentials Authentication

Create and configure an *OAuth2ClientCredentials* destination to consume OAuth-protected resources from a Cloud Foundry application.

SAP BTP supports applications to use the OAuth client credentials flow for consuming OAuth-protected resources.

The client credentials are used to request an access token from an OAuth authorization server.

i Note

The retrieved access token is cached and auto-renewed. When a token is about to expire, a new token is created shortly before the expiration of the old one.

Configuration Steps

You can create and configure an *OAuth2ClientCredentials* destination using the properties listed below, and deploy it on SAP BTP. To create and configure a destination, follow the steps described in [Managing Destinations \[page 39\]](#).

Properties

The table below lists the destination properties required for the *OAuth2ClientCredentials* authentication type.

Property	Description
Required	
Name	Destination name. Must be unique for the destination level.

Property	Description
Type	Destination type. Use HTTP as value for all HTTP(S) destinations.
URL	URL of the protected resource on the called application.
ProxyType	You can only use proxy type Internet .
Authentication	Authentication type. Use OAuth2ClientCredentials as value.
clientId	Client ID used to retrieve the access token.
clientSecret	Client secret for the Client ID.
tokenServiceURL	<p>URL of the token service, against which token retrieval is performed. Depending on the <code>tokenServiceURLType</code>, this property is interpreted in different ways during automatic token retrieval:</p> <ul style="list-style-type: none"> For Dedicated, the <code>tokenServiceURL</code> is used as is. For Common, the <code>tokenServiceURL</code> is searched for the tenant placeholder <code>{tenant}</code>. It is resolved as subdomain of the subaccount on whose behalf the caller is performing the call to the Destination service API for fetching the destination. <p>If the placeholder is not found, <code>{tenant}</code> is processed as a subdomain of the <code>tokenServiceURL</code>.</p> <p>See the Destination service REST API to learn how the subaccount's subdomain is specified. The subaccount subdomain is mandated during creation of the subaccount, see Create a Subaccount (SAP BTP Core documentation).</p> <p>Examples of interpreting of the <code>tokenServiceURL</code> for <code>tokenServiceURLType Common</code>, if the call to the Destination service is done on behalf of a subaccount with subdomain value <code>mytenant</code>:</p> <ul style="list-style-type: none"> • <code>https://authentication.us10.hana.ondemand.com/oauth/token</code> → <code>https://mytenant.authentication.us10.hana.ondemand.com/oauth/token</code> • <code>https://{tenant}.authentication.us10.hana.ondemand.com/oauth/token</code> → <code>https://mytenant.authentication.us10.hana.ondemand.com/oauth/token</code> • <code>https://authentication.myauthserver.com/tenant/{tenant}/oauth/token</code> → <code>https://authentication.myauthserver.com/tenant/mytenant/oauth/token</code> • <code>https://oauth.{tenant}.myauthserver.com/token</code> → <code>https://oauth.mytenant.myauthserver.com/token</code>

Property	Description
tokenServiceURLType	Either Dedicated (if the tokenServiceURL serves only a single tenant), or Common (if the tokenServiceURL serves multiple tenants).
tokenServiceUser	User for basic authentication to OAuth server (if required).
tokenServicePassword	Password for tokenServiceUser (if required).
Additional	
scope	The value of the OAuth 2.0 scope parameter expressed as a list of space-delimited, case-sensitive strings.
tokenServiceURL.headers.<header-key>	A static key prefix used as a namespace grouping of the tokenServiceUrl's HTTP headers. Its values will be sent to the token service during token retrieval. For each HTTP header's key you must add a 'tokenServiceURL.headers' prefix separated by dot delimiter. For example:
<p>Sample Code</p> <pre>{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-value-1>", ... "tokenServiceURL.headers.<header-key-N>" : "<header-value-N>", }</pre>	
tokenServiceURL.queries.<query-key>	A static key prefix used as a namespace grouping of tokenServiceUrl's query parameters. Its values will be sent to the token service during token retrieval. For each query paramester's key you must add a 'tokenServiceURL.queries' prefix separated by dot delimiter. For example:
<p>Sample Code</p> <pre>{ ... "tokenServiceURL.queries.<query-key-1>" : "<query-value-1>", ... "tokenServiceURL.queries.<query-key-N>" : "<query-value-N>", }</pre>	

Property	Description
tokenService.KeyStoreLocation	Contains the name of the certificate configuration to be used. This property is required when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .
tokenService.KeyStorePassword	Contains the password for the certificate configuration (if one is needed) when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .
URL.headers.<header-key>	A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:

↳ Sample Code

```
{
  ...
  "URL.headers.<header-key-1>" :
  "<header-value-1>",
  ...
  "URL.headers.<header-key-N>":
  "<header-value-N>",
}
```

i Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

Property	Description
URL.queries.<query-key>	<p>A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> ↳ Sample Code <pre>{ ... "URL.queries.<query-key-1>" : "<query-value-1>", ... "URL.queries.<query-key-N>" : "<query-value-N>", }</pre> </div> <p>i Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>

i Note

When the OAuth authorization server is called, it accepts the trust settings of the destination, see [Server Certificate Authentication \[page 66\]](#).

When using an SAP BTP Neo OAuth service (`https://api.{landscape-domain}/oauth2/apitoken/v1?grant_type=client_credentials` or `oauthasservices.{landscape-domain}/oauth2/apitoken/v1?grant_type=client_credentials`) as TokenServiceURL, or any other OAuth token service which accepts client credentials only as authorization header, you must set the `clientId` and `clientSecret` values also for the `tokenServiceUser` and `tokenServicePassword` properties.

Example: Neo OAuth Token Service

↳ Sample Code

```
URL=https://api.{landscape-domain}/desired-service-path
Name=sapOAuthCC
TrustAll=true
```

```
ProxyType=Internet
Type=HTTP
Authentication=OAuth2ClientCredentials
tokenServiceURL=(https://api.{landscape-domain}/oauth2/apitoken/v1?
grant_type=client_credentials
tokenServiceUser=clientIdValue
tokenServicePassword=secretValue
clientId=clientIdValue
clientSecret=secretValue
```

Example: OAuth Token Service Accepting Client Credentials as Body

↳ Sample Code

```
URL=https://demo.sapjam.com/OData/OData.svc
Name=sap_jam_odata
TrustAll=true
ProxyType=Internet
Type=HTTP
Authentication=OAuth2ClientCredentials
tokenServiceURL=http://demo.sapjam.com/api/v1/auth/token
tokenServiceUser=tokenserviceuser
tokenServicePassword=pass
clientId=clientId
clientSecret=secret
```

Example: AuthTokens Object Response

The response for "find destination" contains an `authTokens` object in the format given below. For more information on the fields in `authTokens`, see ["Find Destination" Response Structure \[page 199\]](#).

↳ Sample Code

```
"authTokens": [
    {
        "type": "Bearer",
        "value": "eyJhbGciOiJSUzI1NiIsInR5cC...",
        "http_header": {
            "key": "Authorization",
            "value": "Bearer eyJhbGciOiJSUzI1NiIsInR5cC..."
        }
    }
]
```

1.1.3.2.6 OAuth User Token Exchange Authentication

Learn about the `OAuth2UserTokenExchange` authentication type for HTTP destinations in the Cloud Foundry environment: use cases, supported properties and ways to retrieve an access token in an automated way.

Content

[Overview \[page 87\]](#)

[Properties \[page 87\]](#)

[Example: AuthTokens Object Response \[page 91\]](#)

Overview

When a user is logged into an application that needs to call another application and pass the user context, the caller application must perform a user token exchange.

The user token exchange is a sequence of steps during which the initial user token is handed over to the authorization server and, in exchange, another access token is returned.

The calling application first receives a refresh token out of which the actual user access token is created. The resulting user access token contains the user and tenant context as well as technical access metadata, like scopes, that are required for accessing the target application.

Using the `OAuth2UserTokenExchange` authentication type, the Destination service performs all these steps automatically, which lets you simplify your application development in the Cloud Foundry environment.

[Back to Content \[page 87\]](#)

Properties

To configure a destination of this type, you must specify all the required properties. You can create destinations of this type via the cloud cockpit ([Access the Destinations Editor \[page 41\]](#)) or the [Destination Service REST API \[page 59\]](#).

The following table shows the required properties along with their semantics.

Field/Parameter	JSON Key	Input/Description
Required		
URL	URL	URL of the target endpoint.
Token Service URL	tokenServiceURL	<p>The URL of the token service, against which the token exchange is performed. Depending on the Token Service URL Type, this property is interpreted in different ways during the automatic token retrieval:</p> <ul style="list-style-type: none"> For Dedicated, the token service URL is taken as is. For Common, the token service URL is searched for the tenant placeholder <code>{tenant}</code>. <code>{tenant}</code> is resolved as the subdomain of the subaccount on behalf of which the caller is performing the call. If the placeholder is not found, <code>{tenant}</code> is inserted as a subdomain of the token service URL. See Automated Access Token Retrieval [page 207] for information about how the tenant is determined. <p>The subaccount subdomain is mandated during creation of the subaccount, see Create a Subaccount.</p> <p>Examples of interpreting the token service URL for the token service URL type Common, if the call to the Destination service is on behalf of a subaccount subdomain with value <code>mytenant</code>:</p> <ul style="list-style-type: none"> <code>https://authentication.us10.hana.ondemand.com/oauth/token</code> → <code>https://mytenant.authentication.us10.hana.ondemand.com/oauth/token</code> <code>https://</code> <code>{tenant}.authentication.us10.hana.ondemand.com/oauth/token</code> → <code>https://mytenant.authentication.us10.hana.ondemand.com/oauth/token</code> <code>https://authentication.myauthserver.com/tenant/{tenant}/oauth/token</code> → <code>https://authentication.myauthserver.com/tenant/mytenant/oauth/token</code> <code>https://oauth.{tenant}.myauthserver.com/token</code> → <code>https://oauth.mytenant.myauthserver.com/token</code>
Name	Name	Name of the destination. Must be unique for the destination level.
Description	Description	A human-readable description of the destination.
Client Secret	clientSecret	OAuth 2.0 client secret to be used for the user access token exchange.
Client ID	clientId	OAuth 2.0 client ID to be used for the user access token exchange.
Authentication	Authentication	OAuth2UserTokenExchange in this case.
Authenticati on		

Field/Parameter	JSON Key	Input/Description
Proxy Type	ProxyType	Choose Internet.OnPremise is not supported for this authentication type.
Type	Type	Choose HTTP (for HTTP or HTTPS communication).
Token Service URL Type	tokenServiceURLType	<ul style="list-style-type: none"> Choose Dedicated if the token service URL serves only a single tenant. Choose Common if the token service URL serves multiple tenants.
Optional		
Description	Description	Description of the destination.
Additional		
	scope	The value of the OAuth 2.0 scope parameter expressed as a list of space-delimited, case-sensitive strings.
	tokenServiceURL.headers.<header-key>	A static key prefix used as a namespace grouping of the tokenServiceUrl's HTTP headers. Its values will be sent to the token service during token retrieval. For each HTTP header's key you must add a 'tokenServiceURL.headers' prefix separated by dot delimiter. For example:
<p>Sample Code</p> <pre>{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-value-1>", ... "tokenServiceURL.headers.<header-key-N>" : "<header-value-N>", }</pre>		
	tokenServiceURL.queries.<query-key>	A static key prefix used as a namespace grouping of tokenServiceUrl's query parameters. Its values will be sent to the token service during token retrieval. For each query paramester's key you must add a 'tokenServiceURL.queries' prefix separated by dot delimiter. For example:
<p>Sample Code</p> <pre>{ ... "tokenServiceURL.queries.<query-key-1>" : "<query-value-1>", ... "tokenServiceURL.queries.<query-key-N>" : "<query-value-N>", }</pre>		

Field/Parameter	JSON Key	Input/Description
URL.headers .⟨header-key⟩		A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:
↳ Sample Code	<pre>{ ... "URL.headers.<header-key-1>" : "<header-value-1>", ... "URL.headers.<header-key-N>" : "<header-value-N>", }</pre>	i Note
<p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>	<p>URL.queries .⟨query-key⟩</p>	A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:
↳ Sample Code	<pre>{ ... "URL.queries.<query-key-1>" : "<query-value-1>", ... "URL.queries.<query-key-N>" : "<query-value-N>", }</pre>	i Note
<p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>	<p>tokenService e.KeyStoreL ocation</p>	Contains the name of the certificate configuration to be used. This property is required when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .

Field/Parameter	JSON Key	Input/Description
	tokenService e.KeyStoreP password	Contains the password for the certificate configuration (if one is needed) when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .
x_user_toke n.jwks		Base64-encoded JSON web key set, containing the signing keys which are used to validate the JWT provided in the X-User-Token header. For more information, see JWK Set Format .
x_user_toke n.jwks_uri		URI of the JSON web key set, containing the signing keys which are used to validate the JWT provided in the X-User-Token header. For more information, see OpenID Connect Discovery .

Back to [Content \[page 87\]](#)

Example: AuthTokens Object Response

The response for "find destination" contains an authTokens object in the format given below. For more information on the fields in authTokens, see ["Find Destination" Response Structure \[page 199\]](#).

↳ Sample Code

```
"authTokens": [
  {
    "type": "Bearer",
    "value": "eyJhbGciOiJSUzI1NiIsInR5cC...",
    "http_header": {
      "key": "Authorization",
      "value": "Bearer eyJhbGciOiJSUzI1NiIsInR5cC..."
    }
  }
]
```

Back to [Content \[page 87\]](#)

Related Information

[Exchanging User JWTs via OAuth2UserTokenExchange Destinations \[page 207\]](#)

1.1.3.2.7 SAP Assertion SSO Authentication

Create and configure an SAP Assertion SSO destination for an application in the Cloud Foundry environment.

⚠ Caution

Authentication type SAP Assertion SSO is deprecated and will be removed soon. The recommended authentication types for establishing single sign-on (SSO) are:

- [Principal Propagation SSO Authentication for HTTP \[page 68\]](#) for on-premise connections.
- [OAuth SAML Bearer Assertion Authentication \[page 71\]](#) or [SAML Assertion Authentication \[page 106\]](#) for Internet connections.

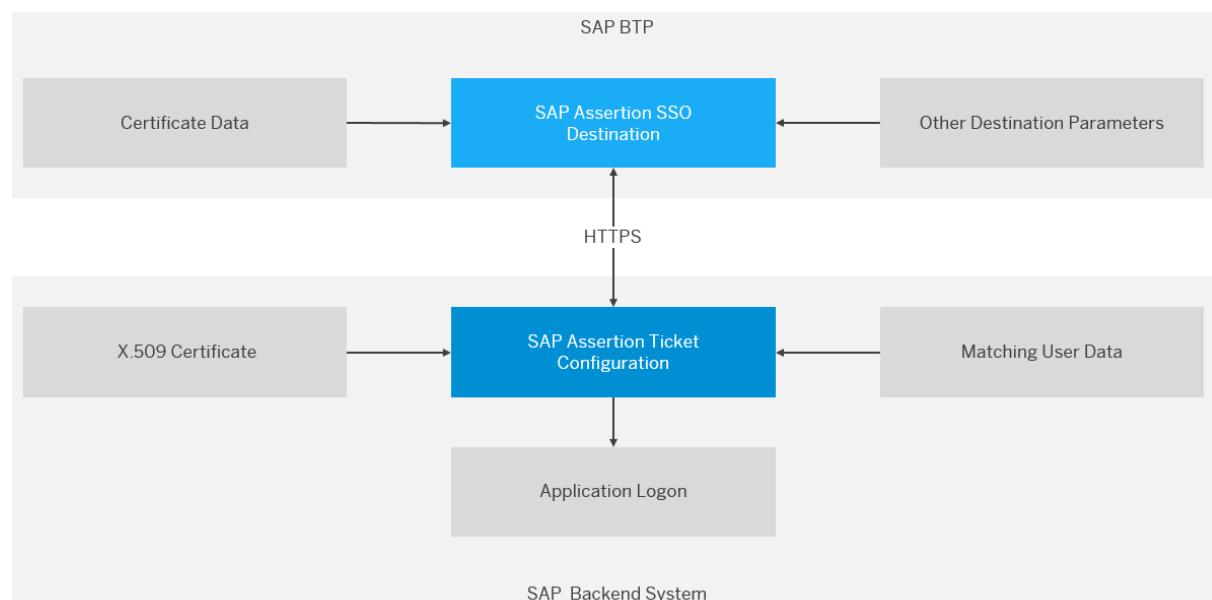
Context

By default, all SAP systems accept SAP assertion tickets for user propagation.

ℹ Note

For more information, see [Authentication Assertion Tickets](#).

The aim of the SAPAssertionSSO destination is to generate such an assertion ticket in order to propagate the currently logged-on SAP BTP user to an SAP backend system. You can only use this authentication type if the user IDs on both sides are the same. The following diagram shows the elements of the configuration process on the SAP BTP and in the corresponding backend system:



Configuration Steps

1. Configure the backend system to accept SAP assertion tickets signed by a trusted x.509 key pair. For more information, see [Configuring a Trust Relationship for SAP Assertion Tickets](#).
2. Create and configure an **SAPAssertionSSO** destination using the properties listed below in the SAP BTP Destination service. For more information, see:
 - o [Access the Destinations Editor \[page 41\]](#)
 - o [Create HTTP Destinations \[page 44\]](#)
 - o [Destination service REST API](#)

Properties

The following credentials must be specified:

Property	Description
Required	
Name	Destination name. It must be the same as the destination name you use in the configuration tools, that is, <i>Destinations</i> editor (cockpit).
Type	Destination type. Use HTTP for all HTTP(S) destination.
URL	URL of the protected resource on the called application.
Authentication	Authentication type. Use SAPAssertionSSO as a value.
IssuerSID	This system ID should be trusted by the backend system.
IssuerClient	This client ID should be trusted by the backend system.
RecipientSID	System ID (SID) of the backend system.
RecipientClient	Client ID of the backend system.
Certificate	A base64 encoded certificate that is trusted by the SAP system.
SigningKey	A base64 encoded signing/private key that is trusted by the SAP system.

Property	Description
(Deprecated) SystemUser	<p>i Note</p> <p>Deprecated. This property will be removed soon.</p> <p>Optional property.</p> <ul style="list-style-type: none"> • If specified, all SAP assertion tickets are generated with the specified user ID. • If not specified, all SAP assertion tickets are sent on behalf of the currently logged-on user. <p>Thus, if the current user needs to be propagated, do not use this property.</p>
ProxyType	You can use both proxy types Internet and OnPremise .
CloudConnectorLocationId	<p>Optional property.</p> <p>As of Cloud Connector version 2.9.0, you can connect multiple Cloud Connectors to an account as long as their location ID is different. The value defines the location ID identifying the Cloud Connector over which the connection is opened. The default value is an empty string identifying the Cloud Connector that is connected without any location ID, which is also the case for all Cloud Connector versions prior to 2.9.0.</p>
Additional	
userIdSource	When this property is set, you can choose which claim in a JWT (JSON Web token) to be considered as <code><user_ID></code> field in the generated assertion.
x_user_token.jwks_uri	<p>URI of the <i>JSON web key set</i>, containing the signing keys which are used to validate the JWT provided in the <i>X-User-Token</i> header.</p> <p>For more information, see OpenID Connect Discovery.</p>
x_user_token.jwks	<p>Base64-encoded <i>JSON web key set</i>, containing the signing keys which are used to validate the JWT provided in the <i>X-User-Token</i> header.</p> <p>For more information, see JWK Set Format.</p>

Property	Description
URL.headers.<header-key>	A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:

↳ Sample Code

```
{  
    ...  
    "URL.headers.<header-key-1>" :  
    "<header-value-1>,  
    ...  
    "URL.headers.<header-key-N>":  
    "<header-value-N>",  
}
```

ℹ Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

Property	Description
URL.queries.<query-key>	A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:

↳ Sample Code

```
{
    ...
    "URL.queries.<query-key-1>" : "<query-value-1>",
    ...
    "URL.queries.<query-key-N>" : "<query-value-N>",
}
```

ℹ Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

Example

```
{
  "Name": "weather",
  "Type": "HTTP",
  "Authentication": "SAPAssertionSSO",
  "IssuerSID": "JAV",
  "IssuerClient": "000",
  "RecipientSID": "SAP",
  "RecipientClient": "100",
  "Certificate": "MIICidCCAkegAwI...rvHTQ\=\=",
  "SigningKey": "MIIBSwIB...RuqNKGA\="
}
```

The response for "find destination" contains an authTokens object in the format given below. For more information on the fields in authTokens, see ["Find Destination" Response Structure \[page 199\]](#).

↳ Sample Code

```
"authTokens": [
  {
```

```
        "type": "MYSAPSSO2",
        "value": "AjExMDACAANKQVYDA...",
        "http_header": {
            "key": "MYSAPSSO2",
            "value": "AjExMDACAANKQVYDA..."
        }
    }
]
```

1.1.3.2.8 OAuth Password Authentication

Learn about the OAuth password authentication type for HTTP destinations in the Cloud Foundry environment: use cases, supported properties and examples.

Content

[Overview \[page 97\]](#)

[Properties \[page 97\]](#)

[Example: OAuth Token Service \[page 100\]](#)

Overview

SAP BTP provides support for applications to use the OAuth password grant flow for consuming OAuth-protected resources.

The client credentials as well as the user name and password are used to request an access token from an OAuth server, referred to as *token service* below. Access token retrieval is performed automatically by the Destination service when using the "find destination" REST endpoint.

[Back to Content \[page 97\]](#)

Properties

The table below lists the destination properties needed for the OAuth2Password authentication type.

⚠ Caution

Do not use your own *personal credentials* in the `<User>` and `<Password>` fields. Always use a *technical user* instead.

Property	Description
Required	
Name	Destination name. It must be the same as the destination name you use for the configuration tools, that is, the console client and <i>Destinations</i> editor (cockpit).
Type	Destination type. Choose HTTP (for HTTP or HTTPS communication).
URL	URL of the protected resource being accessed.
ProxyType	You can only use proxy type Internet .
Authentication	Authentication type. Use OAuth2Password as value.
clientId	Client ID used to retrieve the access token.
clientSecret	Client secret for the client ID.
User	User name of the technical user trying to get a token.
Password	Password of the technical user trying to get a token.
tokenServiceURL	Token retrieval URL of the OAuth server.
RL	
Additional	
scope	Value of the OAuth 2.0 scope parameter, expressed as a list of space-delimited, case-sensitive strings.
tokenServiceURL.headers.<header-key>	Static key prefix used as a namespace grouping of the <code>tokenServiceUrl</code> 's HTTP headers. Its values will be sent to the token service during token retrieval. For each HTTP header's key you must add a ' <code>tokenServiceURL.headers</code> ' prefix separated by dot delimiter. For example:
<div style="border: 1px solid #ccc; padding: 10px;"><p>↳ Sample Code</p><pre>{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-value-1>", ... "tokenServiceURL.headers.<header-key-N>": "<header-value-N>", }</pre></div>	

Property	Description
tokenServiceURL.queries.<query-key>	Static key prefix used as a namespace grouping of tokenServiceUrl's query parameters. Its values will be sent to the token service during token retrieval. For each query parameter's key you must add a 'tokenServiceURL.queries' prefix separated by dot delimiter. For example:
<p>↳ Sample Code</p> <pre>{ ... "tokenServiceURL.queries.<query-key-1>" : "<query-value-1>", ... "tokenServiceURL.queries.<query-key-N>": "<query-value-N>", }</pre>	
tokenService.KeyStoreLocation	Contains the name of the certificate configuration to be used. This property is required when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .
tokenService.KeyStorePassword	Contains the password for the certificate configuration (if one is needed) when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .
URL.headers.<header-key>	Static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:
<p>↳ Sample Code</p> <pre>{ ... "URL.headers.<header-key-1>" : "<header-value-1>", ... "URL.headers.<header-key-N>": "<header-value-N>", }</pre>	
<p>i Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>	

Property	Description
URL.queries.<query-key>	Static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:

↳ Sample Code

```
{
  ...
  "URL.queries.<query-key-1>" : "<query-value-1>",
  ...
  "URL.queries.<query-key-N>": "<query-value-N>",
}
```

i Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

i Note

When the OAuth server is called, the caller side trusts the server based on the trust settings of the destination. For more information, see [Server Certificate Authentication \[page 66\]](#).

Back to [Content \[page 97\]](#)

Example: OAuth Token Service

↳ Sample Code

```
{
  "Name": "SapOAuthPassGrant",
  "Type": "HTTP",
  "URL": "https://myapp.cfapps.sap.hana.ondemand.com/mypath",
  "ProxyType": "Internet",
  "Authentication": "OAuth2Password",
  "clientId": "my-client-id",
  "clientSecret": "my-client-pass",
  "User": "my-username",
  "Password": "my-password",
  "tokenServiceURL": "https://authentication.sap.hana.ondemand.com/oauth/
  token"
}
```

The response for "find destination" contains an `authTokens` object in the format given below. For more information on the fields in `authTokens`, see ["Find Destination" Response Structure \[page 199\]](#).

↳ Sample Code

```
"authTokens": [
  {
    "type": "Bearer",
    "value": "eyJhbGciOiJSUzI1NiIsInR5cC...",
    "http_header": {
      "key": "Authorization",
      "value": "Bearer eyJhbGciOiJSUzI1NiIsInR5cC..."
    }
  }
]
```

Back to [Content \[page 97\]](#)

1.1.3.2.9 OAuth JWT Bearer Authentication

Learn about the OAuth JWT bearer authentication type for HTTP destinations in the Cloud Foundry environment: use cases, supported properties and examples.

Content

[Overview \[page 101\]](#)

[Properties \[page 102\]](#)

[Example: AuthTokens Object Response \[page 106\]](#)

Overview

To allow an application to call another application, passing the user context, and fetch resources, the caller application must pass an access token. In this authorization flow, the initial user token is passed to the OAuth server as input data. This process is performed automatically by the Destination service, which helps simplifying the application development: You only have to construct the right request to the target URL, by using the outcome (another access token) of the service-side automation.

Back to [Content \[page 101\]](#)

Properties

To configure a destination of this authentication type, you must specify all the required properties. You can do this via SAP BTP cockpit (see [Create HTTP Destinations \[page 44\]](#)), or using the [Destination Service REST API \[page 59\]](#). The following table shows the properties along with their semantics.

Field/Parameter (Cockpit)	JSON Key	Description
Required		
Authentication	Authentication	OAuth2JWTBearer in this case.
Client ID	clientId	OAuth 2.0 client ID to be used for the user access token exchange.
Client Secret	clientSecret	OAuth 2.0 client secret to be used for the user access token exchange.
Name	Name	Name of the destination. Must be unique for the destination level.
Proxy Type	ProxyType	Choose Internet.OnPremise is not supported for this authentication type.

Field/Parameter (Cockpit)	JSON Key	Description
Token Service URL	tokenServiceURL	<p>The URL of the token service, against which the token exchange is performed. Depending on the Token Service URL Type, this property is interpreted in different ways during the automatic token retrieval:</p> <ul style="list-style-type: none"> For Dedicated, the token service URL is taken as is. For Common, the token service URL is searched for the tenant placeholder <code>{tenant}</code>. <code>{tenant}</code> is resolved as the subdomain of the subaccount on behalf of which the caller is performing the call. If the placeholder is not found, <code>{tenant}</code> is inserted as a subdomain of the token service URL. See Automated Access Token Retrieval [page 207] for information about how the tenant is determined. <p>The subaccount subdomain is mandated during creation of the subaccount, see Create a Subaccount.</p> <p>Examples of interpreting the token service URL for the token service URL type Common, if the call to the Destination service is on behalf of a subaccount subdomain with value <code>mytenant</code>:</p> <ul style="list-style-type: none"> <code>https://authentication.us10.hana.ondemand.com/oauth/token</code> → <code>https://mytenant.authentication.us10.hana.ondemand.com/oauth/token</code> <code>https://</code> <code>{tenant}.authentication.us10.hana.ondemand.com/oauth/token</code> → <code>https://mytenant.authentication.us10.hana.ondemand.com/oauth/token</code> <code>https://authentication.myauthserver.com/tenant/{tenant}/oauth/token</code> → <code>https://authentication.myauthserver.com/tenant/mytenant/oauth/token</code> <code>https://oauth.{tenant}.myauthserver.com/token</code> → <code>https://oauth.mytenant.myauthserver.com/token</code>
Token Service URL Type	tokenServiceURLType	<ul style="list-style-type: none"> Choose Dedicated if the token service URL serves only a single tenant. Choose Common if the token service URL serves multiple tenants.
Type	Type	Choose HTTP (for HTTP or HTTPS communication).
URL	URL	URL of the target endpoint.
Optional		
Description	Description	A human-readable description of the destination.
Additional		

Field/Parameter (Cockpit)	JSON Key	Description
	scope	The value of the OAuth 2.0 scope parameter, expressed as a list of space-delimited, case-sensitive strings.
	tokenServiceURL.headers.<header-key>	A static key prefix used as a namespace grouping of the tokenServiceUrl's HTTP headers. Its values will be sent to the token service during token retrieval. For each HTTP header's key you must add a 'tokenServiceURL.headers' prefix separated by dot delimiter. For example:
↳ Sample Code <pre>{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-value-1>", ... "tokenServiceURL.headers.<header-key-N>" : "<header-value-N>", }</pre>		
	tokenServiceURL.querie s.<query-key>	A static key prefix used as a namespace grouping of tokenServiceUrl's query parameters. Its values will be sent to the token service during token retrieval. For each query paramester's key you must add a 'tokenServiceURL.queries' prefix separated by dot delimiter. For example:
↳ Sample Code <pre>{ ... "tokenServiceURL.queries.<query-key-1>" : "<query-value-1>", ... "tokenServiceURL.queries.<query-key-N>" : "<query-value-N>", }</pre>		
x_user_token.jwks		Base64-encoded JSON web key set, containing the signing keys which are used to validate the JWT provided in the X-User-Token header.
		For more information, see JWK Set Format .
x_user_token.jwks_uri		URI of the JSON web key set, containing the signing keys which are used to validate the JWT provided in the X-User-Token header.
		For more information, see OpenID Connect Discovery .

Field/Parameter (Cockpit)	JSON Key	Description
URL.headers . <header-key>	URL.headers. . <header-key>	A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:
<p>↳ Sample Code</p> <pre>{ ... "URL.headers.<header-key-1>" : "<header-value-1>", ... "URL.headers.<header-key-N>" : "<header-value-N>", }</pre>	<p>i Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>	
URL.queries . <query-key>	URL.queries. . <query-key>	A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:
<p>↳ Sample Code</p> <pre>{ ... "URL.queries.<query-key-1>" : "<query-value-1>", ... "URL.queries.<query-key-N>" : "<query-value-N>", }</pre>	<p>i Note</p> <p>This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.</p>	
tokenService .KeyStoreLocation	tokenService .KeyStoreLocation	Contains the name of the certificate configuration to be used. This property is required when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .

Field/Parameter (Cockpit)	JSON Key	Description
	tokenService.e.KeyStoreP password	Contains the password for the certificate configuration (if one is needed) when using client certificates for authentication. See OAuth with X.509 Client Certificates [page 110] .

Back to [Content \[page 101\]](#)

Example: AuthTokens Object Response

The response for "find destination" contains an authTokens object in the format given below. For more information on the fields in authTokens, see ["Find Destination" Response Structure \[page 199\]](#).

↳ Sample Code

```
"authTokens": [
    {
        "type": "Bearer",
        "value": "eyJhbGciOiJSUzI1NiIsInR5cC...",
        "http_header": {
            "key": "Authorization",
            "value": "Bearer eyJhbGciOiJSUzI1NiIsInR5cC..."
        }
    }
]
```

Back to [Content \[page 101\]](#)

1.1.3.2.10 SAML Assertion Authentication

Create and configure an *SAML Assertion* destination for an application in the Cloud Foundry environment.

Context

The Destination service lets you generate SAML assertions as per SAML 2.0 specification. You can retrieve a generated SAML assertion from the Destination service by using the `SAMLAssertion` authentication type, whereas [OAuth SAML Bearer Assertion Authentication \[page 71\]](#) sends the generated SAML assertion to an OAuth server to get a token. The Destination service provides functionality for caching the generated SAML assertion for later use, and caching by the app whenever needed, which helps simplifying application development.

Properties

The table below lists the destination properties for the **SAMLAssertion** authentication type.

Property	Description
Required	
Name	Name of the destination. Must be unique for the destination level.
Type	Destination type. Choose HTTP for all HTTP(S) destinations.
URL	URL of the target endpoint.
ProxyType	Choose Internet or OnPremise .
CloudConnectorLocationId	(only if ProxyType=OnPremise) Starting with Cloud Connector 2.9.0, you can connect multiple Cloud Connectors to an account as long as their location ID is different. The value defines the location ID identifying the Cloud Connector over which the connection is opened. The default value is an empty string identifying the Cloud Connector that is connected without any location ID, which is also the case for all Cloud Connector versions prior to 2.9.0.
Authentication	Authentication type. Use SAMLAssertion as value.
audience	Value of the Audience tag, which is part of the generated SAML assertion. For more information, see SAML 2.0 specification .
authnContextClassRef	Value of the AuthnContextClassRef tag, which is part of generated SAML assertion. For more information, see SAML 2.0 specification .
Additional	
clientKey	Key that identifies the consumer to the authorization server.
nameQualifier	When this property is set, the NameQualifier under the NameId tag of the generated SAML assertion is determined in accordance to the value.
companyId	Company identifier.
assertionIssuer	Issuer of the SAML assertion.
nameIdFormat	Value of the NameIdFormat tag, which is part of generated SAML Assertion. For more information, see SAML 2.0 specification .
userIdSource	When this property is set, the user ID in the NameId tag of the generated SAML assertion is determined in accordance to the value of this attribute. For more information, see User Propagation via SAML 2.0 Bearer Assertion Flow [page 202] .

Property	Description
x_user_token.jwks	<p>Base64-encoded <i>JSON web key set</i>, containing the signing keys which are used to validate the JWT provided in the <i>X-User-Token</i> header.</p> <p>For more information, see JWK Set Format.</p>
x_user_token.jwks_uri	<p>URI of the <i>JSON web key set</i>, containing the signing keys which are used to validate the JWT provided in the <i>X-User-Token</i> header.</p> <p>For more information, see OpenID Connect Discovery.</p>
URL.headers.<header-key>	<p>A static key prefix used as a namespace grouping of the URL's HTTP headers whose values will be sent to the target endpoint. For each HTTP header's key, you must add a URL.headers prefix separated by dot-delimiter. For example:</p> <div style="background-color: #f0f0f0; padding: 10px; border-radius: 5px;"> Sample Code <pre>{ ... "URL.headers.<header-key-1>" : "<header-value-1>", ... "URL.headers.<header-key-N>" : "<header-value-N>", }</pre> </div>

Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

Property	Description
URL.queries.<query-key>	A static key prefix used as a namespace grouping of URL's query parameters whose values will be sent to the target endpoint. For each query parameter's key, you must add a URL.queries prefix separated by dot-delimiter. For example:

↳ Sample Code

```
{
    ...
    "URL.queries.<query-key-1>" : "<query-value-1>",
    ...
    "URL.queries.<query-key-N>" : "<query-value-N>",
}
```

ℹ Note

This is a naming convention. As the call to the target endpoint is performed on the client side, the service only provides the configured properties. The expectation for the client-side processing logic is to parse and use them. If you are using higher-level libraries and tools, please check if they support this convention.

Example

The connectivity destination below provides HTTP access to the OData API of the SuccessFactors Jam.

```
Name=destinationSamlAssertion
Type=HTTP
URL=https://myXXXXXX-api.s4hana.ondemand.com
Authentication=SAMLAssertion
ProxyType=Internet
audience=https://myXXXXXX.s4hana.ondemand.com
authnContextClassRef=urn:oasis:names:tc:SAML:2.0:ac:classes:X509
```

The response for "find destination" contains an authTokens object in the format given below. For more information on the fields in authTokens, see ["Find Destination" Response Structure \[page 199\]](#).

↳ Sample Code

```
"authTokens": [
{
    "type": "SAML2.0",
    "value": "PD94bWwgdmVyc2lvbj0iMS4wIiBhb...",
}
```

```
        "http_header": {
            "key": "Authorization",
            "value": "SAML2.0 PD94bWwgdmVyc2lvbj0iMS4wIiB1bmNvZ..."
        }
    }
```

Related Information

[Create HTTP Destinations \[page 44\]](#)

[Destination Examples \[page 48\]](#)

[Exchanging User JWTs via OAuth2UserTokenExchange Destinations \[page 207\]](#)

1.1.3.2.11 OAuth with X.509 Client Certificates

Use an X.509 certificate instead of a secret to authenticate against the authentication server.

To perform mutual TLS, you can use an X.509 client certificate instead of a client secret when connecting to the authorization server. To do so, you must create a certificate configuration containing a valid X.509 client certificate or a keystore, and link it to the destination configuration using these properties:

Property	Description
tokenService.KeyStoreLocation	Contains the name of the certificate configuration to be used. This property is required.
tokenService.KeyStorePassword	Contains the password for the certificate configuration (if one is needed).

⚠ Caution

Mutual TLS with an X.509 client certificate is performed only if the `tokenService.KeyStoreLocation` property is set in the destination configuration. Otherwise, the client secret is used.

Supported Certificate Configuration Formats

- Java Keystore (.jks): Requires the `tokenService.KeyStorePassword` property.
- PKCS12 (.pfx or .p12): Requires the `tokenService.KeyStorePassword` property.
- PEM-encoded X.509 client certificate and private key (.crt, .cer and .pem): The certificate configuration can contain several valid X.509 certificates and private keys.

Supported OAuth Flows

- OAuth Client Credentials Authentication [page 81]
- OAuth Password Authentication [page 97]
- OAuth User Token Exchange Authentication [page 87]
- OAuth JWT Bearer Authentication [page 101]

1.1.3.3 RFC Destinations

RFC destinations provide the configuration required for communication with an on-premise ABAP system via Remote Function Call. The RFC destination data is used by the Java Connector (JCo) version that is available within SAP BTP to establish and manage the connection.

RFC Destination Properties

The RFC destination specific configuration in SAP BTP consists of properties arranged in groups, as described below. The supported set of properties is a subset of the standard JCo properties in arbitrary environments. The configuration data is divided into the following groups:

- User Logon Properties [page 112]
- Pooling Configuration [page 114]
- Repository Configuration [page 116]
- Target System Configuration [page 117]
- Parameters Influencing Communication Behavior [page 121]

The minimal configuration contains user logon properties and information identifying the target host. This means you must provide at least a set of properties containing this information.

Example

```
Name=SalesSystem
Type=RFC
jco.client.client=000
jco.client.lang=EN
jco.client.user=consultant
jco.client.passwd=<password>
jco.client.ashost=sales-system.cloud
jco.client.sysnr=42
jco.destination.pool_capacity=5
jco.destination.peak_limit=10
```

Related Information

[Invoking ABAP Function Modules via RFC \[page 217\]](#)

1.1.3.3.1 User Logon Properties

JCo properties that cover different types of user credentials, as well as the ABAP system client and the logon language.

The currently supported logon mechanism uses user or password as credentials.

Property	Description
jco.client.client	Represents the client to be used in the ABAP system. Valid format is a three-digit number.
jco.client.lang	Optional property. Represents the logon language. If the property is not provided, the user's or system's default language is used. Valid values are two-character ISO language codes or one-character SAP language codes.
jco.client.user	Represents the user to be used for logging on to the ABAP system. Max. 12 characters long. <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p>i Note When working with the <i>Destinations</i> editor in the cockpit, enter the value in the <User> field. Do not enter it as additional property.</p></div>
jco.client.alias_user	Represents the user to be used for logging on to the ABAP system. Either jco.client.user or jco.client.alias_user must be specified. The alias user may be up to 40 characters long. <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"><p>i Note When working with the <i>Destinations</i> editor in the cockpit, enter the value in the <Alias User> field. Do not enter it as additional property.</p></div>

Property	Description
jco.client.passwd	<p>Represents the password of the user that is used.</p> <p>i Note</p> <p>Passwords in systems of SAP NetWeaver releases lower than 7.0 are case-insensitive and can be only eight characters long. For releases 7.0 and higher, passwords are case-sensitive with a maximum length of 40.</p>
jco.client.tls_client_certificate_logon	<p>When set to 1, the client certificate provided by the Key-Store, which must be configured in addition, is used for authentication instead of jco.client.user/jco.client.alias_user and jco.client.passwd. This property is only relevant for a connection using WebSocket RFC (<Proxy Type>=Internet).</p> <p>The default value is 0.</p> <p>i Note</p> <p>When working with the Destinations editor in the cockpit, the <User>, <Alias User> and <Password> fields are hidden when setting the property to 1.</p>

For more information on WebSocket RFC, see also:

[WebSocket RFC](#)

Property	Description
jco.destination.auth_type	<p>Optional property.</p> <ul style="list-style-type: none"> If the property is not provided, its default value CONFIGURED_USER is used, which means that user, password, or other credentials are specified directly. To enable single sign-on via principal propagation (which means that the identity logged on in the cloud application is forwarded to the on-premise system), set the value to PrincipalPropagation. In this case, you do not need to provide jco.client.user and jco.client.passwd in the configuration. <p>i Note</p> <p>For PrincipalPropagation, you should configure the properties jco.destination.repository.user and jco.destination.repository.passwd instead, since there are special permissions needed (for metadata lookup in the back end) that not all business application users might have.</p>

1.1.3.3.2 Pooling Configuration

Learn about the JCo properties you can use to configure pooling in an RFC destination.

Overview

This group of JCo properties covers different settings for the behavior of the destination's connection pool. All properties are optional.

Property	Description
jco.destination.pool_capacity	Represents the maximum number of idle connections kept open by the destination. A value of 0 has the effect of no connection pooling, that is, connections will be closed after each request. The default value is 1 .

Property	Description
jco.destination.peak_limit	Represents the maximum number of active connections you can create for a destination simultaneously. Value 0 allows an unlimited number of active connections. Otherwise, if the value is less than the value of jco.destination.pool_capacity, it will be automatically increased to this value. Default setting is the value of jco.destination.pool_capacity. If jco.destination.pool_capacity is not specified, the default is 0 (unlimited).
jco.destination.max_get_client_time	Represents the maximum time in milliseconds to wait for a free connection in case the maximum number of active connections is already allocated by applications.
jco.destination.expiration_time	Represents the time in milliseconds after which idle connections that are available in the pool can be closed.
jco.destination.expiration_check_period	Represents the interval in milliseconds for the timeout checker thread to check the idle connections in the pool for expiration.
jco.destination.pool_check_connection	When setting this value to 1 , a pooled connection will be checked for corruption before being used for the next function module execution. Thus, it is possible to recognize corrupted connections and avoid exceptions being passed to applications when connectivity is basically working (default value is 0).

i Note

Turning on this check has performance impact for stateless communication. This is due to an additional low-level ping to the server, which takes a certain amount of time for non-corrupted connections, depending on latency.

Pooling Details

- Each destination is associated with a connection factory and, if the pooling feature is used, with a connection pool.
- Initially, the destination's connection pool is empty, and the JCo runtime does not preallocate any connection. The first connection will be created when the first function module invocation is performed. The `peak_limit` property describes how many connections can be created simultaneously, if applications allocate connections in different sessions at the same time. A connection is allocated either when a stateless function call is executed, or when a connection for a stateful call sequence is reserved within a session.

- After the `<peak_limit>` number of connections has been allocated (in `<peak_limit>` number of sessions), the next session will wait for at most `<max_get_client_time>` milliseconds until a different session releases a connection (either finishes a stateless call or ends a stateful call sequence). In case the waiting session does not get any connection during the `<max_get_client_time>` period, the function request will be aborted with `JCoException` with the key `JCO_ERROR_RESOURCE`.
- Connections that are no longer used by applications are returned to the destination pool. There is at most a `<pool_capacity>` number of connections kept open by the pool. Further connections (`<peak_limit>` - `<pool_capacity>`) will be closed immediately after usage. The pooled connections (open connections in the pool) are marked as expired if they are not used again during `<expiration_time>` milliseconds. All expired connections will be closed by a timeout checker thread which executes the check every `<expiration_check_period>` milliseconds.

1.1.3.3.3 Repository Configuration

JCo properties that allow you to define the behavior of the repository that dynamically retrieves function module metadata.

All properties below are optional. Alternatively, you can create the metadata in the application code, using the metadata factory methods within the `JCo` class, to avoid additional round-trips to the on-premise system.

Property	Description
<code>jco.destination.repository_destination</code>	Specifies which destination should be used for repository queries. If the destination does not exist, an error occurs when trying to retrieve the repository. Defaults to itself.
<code>jco.destination.repository.user</code>	Optional property. If this property is set, and the repository destination is not set, it is used as the user for repository queries. This configuration option allows using a different user for repository lookups with a single destination configuration, and restricting this user's permissions accordingly. See also SAP Note 460089 .
<code>jco.destination.repository.passwd</code>	Represents the password for a repository user. If you use such a user, this property is mandatory.

i Note

When working with the `Destinations` editor in the cockpit, enter the value in the `<Repository User>` field. Do not enter it as additional property.

i Note

When working with the `Destinations` editor in the cockpit, enter this password in the `<Repository Password>` field. Do not enter it as additional property.

1.1.3.3.4 Target System Configuration

Learn about the JCo properties you can use to configure the target system information in an RFC destination (Cloud Foundry environment).

i Note

This documentation refers to SAP BTP, Cloud Foundry environment. If you are looking for information about the Neo environment, see [Target System Configuration \(Neo environment\)](#).

Content

[Overview \[page 117\]](#)

[Proxy Types \[page 118\]](#)

[Direct Connection \[page 118\]](#)

[Load Balancing Connection \[page 118\]](#)

[WebSocket Connection \[page 119\]](#)

Overview

You can use the following configuration types alternatively:

- Direct connection to an ABAP application server
- Load balancing connection to a group of ABAP application servers via a message server
- WebSocket connection to an ABAP application server (RFC over Internet)

i Note

When using a WebSocket connection, the target ABAP system must be exposed to the Internet.

Depending on the configuration you use, different properties are mandatory or optional.

To improve performance, consider using optional properties additionally, such as `jco.client.serialization_format`. For more information, see [JCo documentation](#).

Back to [Content \[page 117\]](#)

Proxy Types

The field `<Proxy Type>` lets you choose between `Internet` and `OnPremise`. When choosing `OnPremise`, the RFC communication is routed over a Cloud Connector that is connected to the subaccount. When choosing `Internet`, the RFC communication is done over a WebSocket connection.

Back to [Content \[page 117\]](#)

Direct Connection

To use a direct connection (connection without load balancing) to an application server over Cloud Connector, you must set the value for `<Proxy Type>` to `OnPremise`.

Property	Description
<code>jco.client.ashost</code>	Represents the application server host to be used. For configurations on SAP BTP, the property must match a virtual host entry in the Cloud Connector <code>Access Control</code> configuration. The property indicates that a direct connection is established.
<code>jco.client.sysnr</code>	Represents the so-called "system number" and has two digits. It identifies the logical port on which the application server is listening for incoming requests. For configurations on SAP BTP, the property must match a virtual port entry in the Cloud Connector <code>Access Control</code> configuration.
<code>jco.client.client</code>	Three-digit ABAP client number. Defines the client of the target ABAP system.

Back to [Content \[page 117\]](#)

i Note

The virtual port in the above access control entry must be named `sapgw<##>`, where `<##>` is the value of `sysnr`.

Load Balancing Connection

To use load balancing to a system over Cloud Connector, you must set the value for `<Proxy Type>` to `OnPremise`.

Property	Description
jco.client.mshost	Represents the message server host to be used. For configurations on SAP BTP, the property must match a virtual host entry in the Cloud Connector <i>Access Control</i> configuration. The property indicates that load balancing is used for establishing a connection.
jco.client.group	Optional property. Identifies the group of application servers that is used, the so-called "logon group". If the property is not specified, the group PUBLIC is used.
jco.client.r3name	Represents the three-character system ID of the ABAP system to be addressed. For configurations on SAP BTP, the property must match a virtual port entry in the Cloud Connector <i>Access Control</i> configuration.
<p>i Note</p> <p>The virtual port in the above access control entry must be named sapms<###>, where <###> is the value of r3name.</p>	
jco.client.msserv	Represents the port on which the message server is listening for incoming requests. You can use this property as an alternative to jco.client.r3name. One of these two must be present. For configurations on SAP BTP, the property must match a virtual port entry in the Cloud Connector <i>Access Control</i> configuration. You can therefore avoid look-ups in the /etc/services file (<Install_Drive>\Windows\System32\drivers\etc\services) on the Cloud Connector host.
jco.client.client	Three-digit ABAP client number. Defines the client of the target ABAP system.

Back to [Content \[page 117\]](#)

WebSocket Connection

To use a direct connection over WebSocket, you must set the value for <Proxy Type> to Internet.

Prerequisites

- Your target system is an ABAP server as of S/4HANA (on-premise) version 1909, or a cloud ABAP system.
- Your SAP Java buildpack version is at least 1.26.0.

Property	Description
jco.client.wshost	Represents the WebSocket RFC server host on which the target ABAP system is running. The system must be exposed to the Internet.
jco.client.wsport	Represents the WebSocket RFC server port on which the target ABAP system is listening.
jco.client.client	Three-digit ABAP client number. Defines the client of the target ABAP system.
jco.client.tls_trust_all	If set to 1, all server certificates are considered trusted during TLS handshake. If set to 0, either a dedicated trust store must be configured, or the JDK trust store is used as default. Default value is 0.
<p>i Note</p> <p>We recommend that you do not use value 1 ("trust all") in productive scenarios, but only for demo/test purposes.</p>	
<Trust Store Location>	<p>If you don't want to use the default JDK trust store (option <i>Use default JDK truststore</i> is unchecked), you must enter a <Trust Store Location>. This field indicates the path to the JKS file which contains trusted certificates (Certificate Authorities) for authentication against a remote client.</p> <ol style="list-style-type: none"> The relative path to the JKS file. The root path is the server's location on the file system. The name of the JKS file.
<p>i Note</p> <p>If the <Trust Store Location> is not specified, the JDK trust store is used as a default trust store for the destination.</p>	
<Trust Store Password>	Password for the JKS trust store file. This field is mandatory if <Trust Store Location> is used.

i Note

You can upload trust store JKS files using the same command as for uploading destination configuration property files. You only need to specify the JKS file instead of the destination configuration file.

i Note

Connections to remote services which require *Java Cryptography Extension (JCE) unlimited strength jurisdiction policy* are not supported.

See also [WebSocket RFC](#) (ABAP Platform documentation).

Back to [Content \[page 117\]](#)

1.1.3.3.5 Parameters Influencing Communication Behavior

JCo properties that allow you to control the connection to an ABAP system.

All properties are optional.

Property	Description
jco.client.trace	Defines whether protocol traces are created. Valid values are 1 (trace is on) and 0 (trace is off). The default value is 0 .
jco.client.codepage	Declares the 4-digit SAP codepage that is used when initiating the connection to the backend. The default value is 1100 (comparable to iso-8859-1). It is important to provide this property if the password that is used contains characters that cannot be represented in 1100 .
jco.client.delta	Enables or disables table parameter delta management. It is enabled if set to 1 , and respectively disabled if set to 0 . The default value is 1 .
jco.client.cloud_connector_version	Defines the Cloud Connector version used for establishing a connection to the on-premise system. The default value is 2 . Currently, no other values are supported.
jco.client.cloud_connector_location_id	As of Cloud Connector 2.9.0, you can connect multiple Cloud Connectors to a subaccount as long as their location ID is different. The location ID specifies the Cloud Connector over which the connection is opened. The default value is an empty string identifying the Cloud Connector that is connected without any location ID. This is also valid for all Cloud Connector versions prior to 2.9.0.

i Note

When working with the *Destinations* editor in the cockpit, enter the Cloud Connector location ID in the **<Location ID>** field. Do not enter it as additional property.

Property	Description
jco.client.serialization_format	Defines the serialization format that is used when transferring function module data to the partner system. The property impacts the serialization behavior of function module data. Valid values are columnBased and rowBased . If you choose columnBased , the <i>fast RFC</i> serialization is used, as long as the partner system supports it, see SAP Note 2372888 . When choosing the rowBased option, <i>classic</i> or <i>basXML</i> serialization are used. The default value is rowBased .
jco.client.network	Defines which network type is expected to be used for the destination. The property impacts the serialization behavior of function module data, see SAP Note 2372888 . Valid values are WAN and LAN . The default value is LAN .

1.1.3.4 Principal Propagation

Enable single sign-on (SSO) by forwarding the identity of cloud users to a remote system or service (Cloud Foundry environment).

The Connectivity and Destination services let you forward the identity of a cloud user to a remote system. This process is called principal propagation (also known as *user propagation* or *user principal propagation*). It uses a JSON Web token (JWT) as exchange format for the user information.

Two scenarios are supported: Cloud to on-premise (using the Connectivity service) and cloud to cloud (using the Destination service).

- [Scenario: Cloud to On-Premise \[page 123\]](#): The user is propagated from a cloud application to an on-premise system using a destination configuration with authentication type `PrincipalPropagation`.

i Note

This scenario requires the Cloud Connector to connect to your on-premise system.

- [Scenario: Cloud to Cloud \[page 124\]](#): The user is propagated from a cloud application to another remote (cloud) system using a destination configuration with authentication type `OAuth2SAMLBearerAssertion`.

For more information on setting up destinations, see:

- [Create HTTP Destinations \[page 44\]](#)
- [Create RFC Destinations \[page 45\]](#)

1.1.3.4.1 Scenario: Cloud to On-Premise

Forward the identity of cloud users from the Cloud Foundry environment to on-premise systems using principal propagation.

Concept

The Connectivity service lets you connect your cloud applications to on-premise systems through the Cloud Connector and forward the identity of a cloud user. This process is called principal propagation (also known as user propagation). The JSON Web token (JWT) representing the cloud user identity is forwarded to the Cloud Connector, which verifies it, and propagates the user identity via either an X.509 certificate or Kerberos.

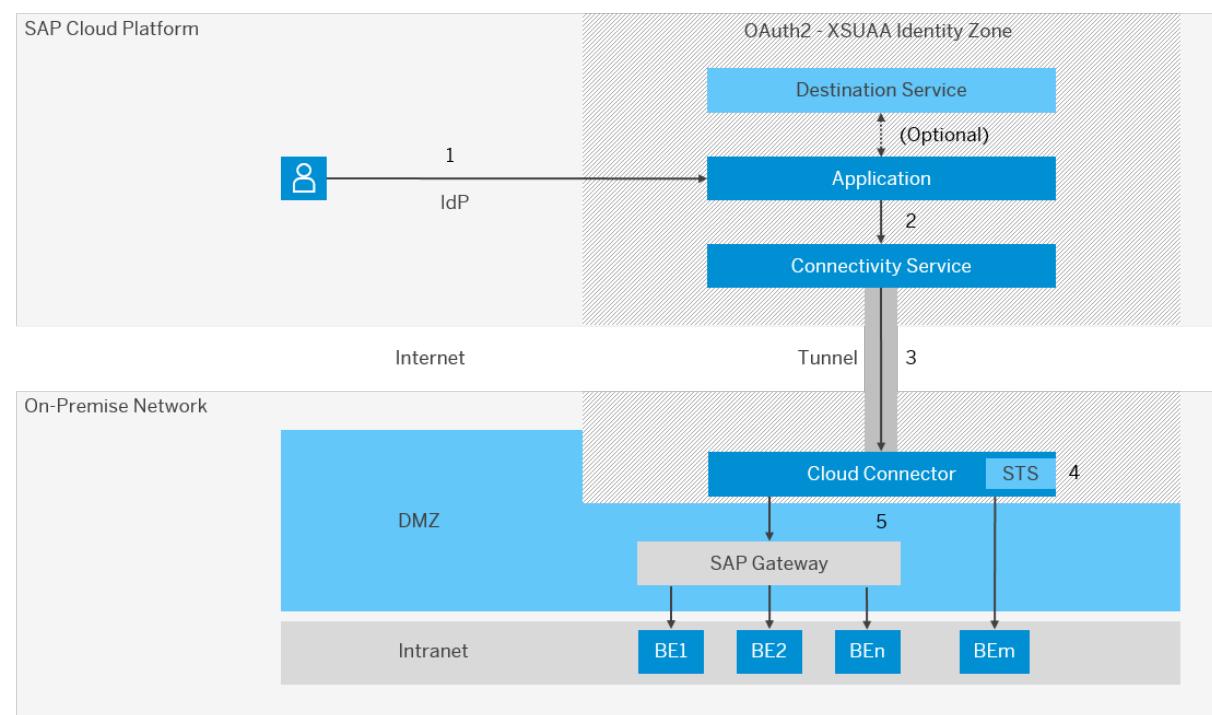
Optionally, you can configure and use a destination configuration by setting the authentication type as `PrincipalPropagation`. For more information, see [Managing Destinations \[page 39\]](#).

i Note

This scenario is only applicable if the on-premise system is exposed to the cloud via the Cloud Connector.

You can configure principal propagation for HTTP or RFC communication.

Scenario: Cloud to On-Premise



1. A user logs in to the cloud application. Its identity is established by an identity provider (this can be the default IdP for the subaccount or another trusted IdP).
2. The cloud application then uses a user exchange token (or a designated secondary header) to propagate the user to the Connectivity service. See also [Configure Principal Propagation via User Exchange Token \[page 176\]](#).
 - Optionally, the application may use the Destination service to externalize the connection configuration that points to the target on-premise system. See also [Consuming the Destination Service \[page 191\]](#).
 - If you are using RFC as communication protocol with the SAP Java Buildpack, this step is already done by the Java Connector (JCo).
3. The Connectivity service forwards the JWT (that represents the user) to the Cloud Connector.
4. The Cloud Connector receives the JWT, verifies it, extracts the attributes, and uses its STS (security token service) component to issue a new token (for example, an X.509 certificate) with the same or similar attributes to assert the identity to the backend (BE1-BEm). The Cloud Connector and the cloud application share the same trust settings, see [Set Up Trust for Principal Propagation \[page 351\]](#).
5. The Cloud Connector sends the new token (for example, an X.509 certificate) to the backend system.

Configuration: Cloud to On-Premise

Task Type	Task
	Configuring Principal Propagation [page 350] (Cloud Connector)
Operator	
	<p>Use cases:</p> <ul style="list-style-type: none"> • HTTP communication: Configure Principal Propagation via User Exchange Token [page 176] (Connectivity service) • RFC communication: Configure Principal Propagation for RFC [page 274]
Developer	

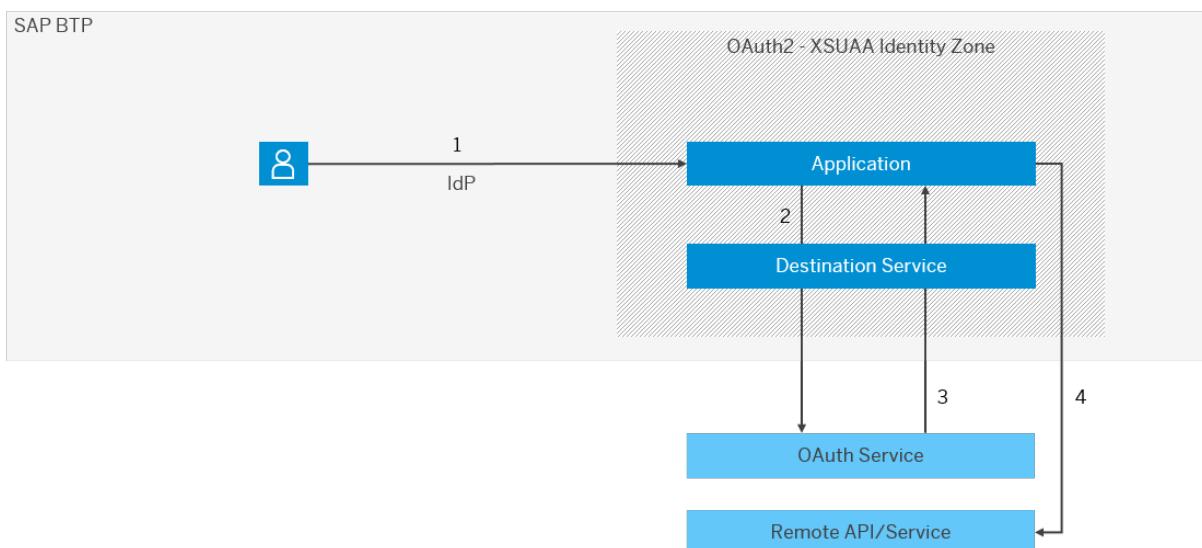
1.1.3.4.2 Scenario: Cloud to Cloud

Forward the identity of cloud users from the Cloud Foundry environment to remote systems on the Internet, enabling single sign-on (SSO).

Concept

The Destination service provides a secure way of forwarding the identity of a cloud user to another remote system or service using a destination configuration with authentication type `OAuth2SAMLBearerAssertion`. This enables the cloud application to consume OAuth-protected APIs exposed by the target remote system.

Scenario: Cloud to Cloud



1. A user logs in to the cloud application. Its identity is established by an identity provider (this can be the default IdP for the subaccount or another trusted IdP).
2. When the application retrieves an `OAuthSAMLBearer` destination, the user is made available to the Destination Service by means of a user exchange JWT. The service then wraps the user identity in a SAML assertion, signs it with the subaccount's private key and sends it to the specified OAuth token service.
3. The OAuth token service accepts the SAML assertion and returns an OAuth access token. In turn, the Destination service returns both the destination and the access token to the requesting application.
4. The application uses the destination properties and the access token to consume the remote API.

You can set up user propagation for connections to applications in different cloud systems or environments.

Configuration: Cloud to Cloud

Task Type	Task
	Set up Trust Between Systems [page 126]
Operator	
	User Propagation via SAML 2.0 Bearer Assertion Flow [page 202] (Destination service)
Operator and/or Developer	

Use Cases: Cloud to Cloud

- [User Propagation from the Cloud Foundry Environment to SAP S/4HANA Cloud \[page 128\]](#)
- [User Propagation from the Cloud Foundry Environment to SAP SuccessFactors \[page 137\]](#)
- [User Propagation between Cloud Foundry Applications \[page 142\]](#)
- [User Propagation from the Cloud Foundry Environment to the Neo Environment \[page 150\]](#)

1.1.3.4.2.1 Set up Trust Between Systems

Download and configure X.509 certificates as a prerequisite for user propagation from the Cloud Foundry environment.

Setting up a trust scenario for user propagation requires the exchange of public keys and certificates between the affected systems, as well as the respective trust configuration within these systems. This enables you to use an HTTP destination with authentication type `OAuth2SAMLBearerAssertion` for the communication.

A trust scenario can include user propagation from the Cloud Foundry environment to another SAP BTP environment, to another Cloud Foundry subaccount, or to a remote system outside SAP BTP, like S/4HANA Cloud, C4C, Success Factors, and others.

[Set Up a Certificate \[page 126\]](#)

[Renew a Certificate \[page 127\]](#)

Set Up a Certificate

Download and save locally the identifying X509 certificate of the subaccount in the Cloud Foundry environment.

1. In the cloud cockpit, log on with `Administrator` permission.
2. Navigate to your subaccount in the Cloud Foundry environment.
3. From the left-side menu, choose  `Connectivity`  `Destinations`.

- Choose the *Download Trust* button and save locally the X.509 certificate that identifies this subaccount.

The screenshot shows the SAP BTP Cockpit interface. On the left, there's a navigation menu with items like Overview, Services, Subscriptions, Cloud Foundry, Connectivity (with Destinations selected), Security, Entitlements, and Usage Analytics. The main area is titled 'Subaccount: trial - Destinations' and shows a table with one row. The table has columns for Type (HTTP), Name (test), Basic Properties (Authentication: NoAuthentication, ProxyType: Internet, URL: http://sap.com), and Actions (edit, download, delete). At the top of the table, there are tabs: New Destination, Import Destination, Certificates, Download Trust (which is highlighted with a red box), and Renew Trust.

- Configure the downloaded X.509 certificate in the target system to which you want to propagate the user.

Renew a Certificate

If the X.509 certificate validity is about to expire, you can renew the certificate and extend its validity by another 2 years.

- In the cloud cockpit, log on with **Administrator** permission.
- Navigate to your subaccount in the Cloud Foundry environment.
- From the left-side menu, choose **Connectivity > Destinations**.
- Choose the *Renew Trust* button to trigger a renewal of the existing X509 certificate.

This screenshot is similar to the previous one, showing the SAP BTP Cockpit Destinations list. The 'Renew Trust' button is highlighted with a red box. The rest of the interface, including the table and navigation menu, is identical to the first screenshot.

- Choose the *Download Trust* button and save locally the X.509 certificate that identifies this subaccount.
- Configure the renewed X.509 certificate in the target system to which you want to propagate the user.

Related Information

[Principal Propagation from the Cloud Foundry to the Neo Environment](#)

[User Propagation from the Cloud Foundry Environment to SAP S/4HANA Cloud \[page 128\]](#)

[User Propagation from the Cloud Foundry Environment to SAP SuccessFactors \[page 137\]](#)

1.1.3.4.2.2 User Propagation from the Cloud Foundry Environment to SAP S/4HANA Cloud

Configure user propagation (single sign-on), using OAuth communication from the SAP BTP Cloud Foundry environment to S/4HANA Cloud. As OAuth mechanism, you use the *OAuth 2.0 SAML Bearer Assertion Flow*.

Steps

[Scenario \[page 128\]](#)

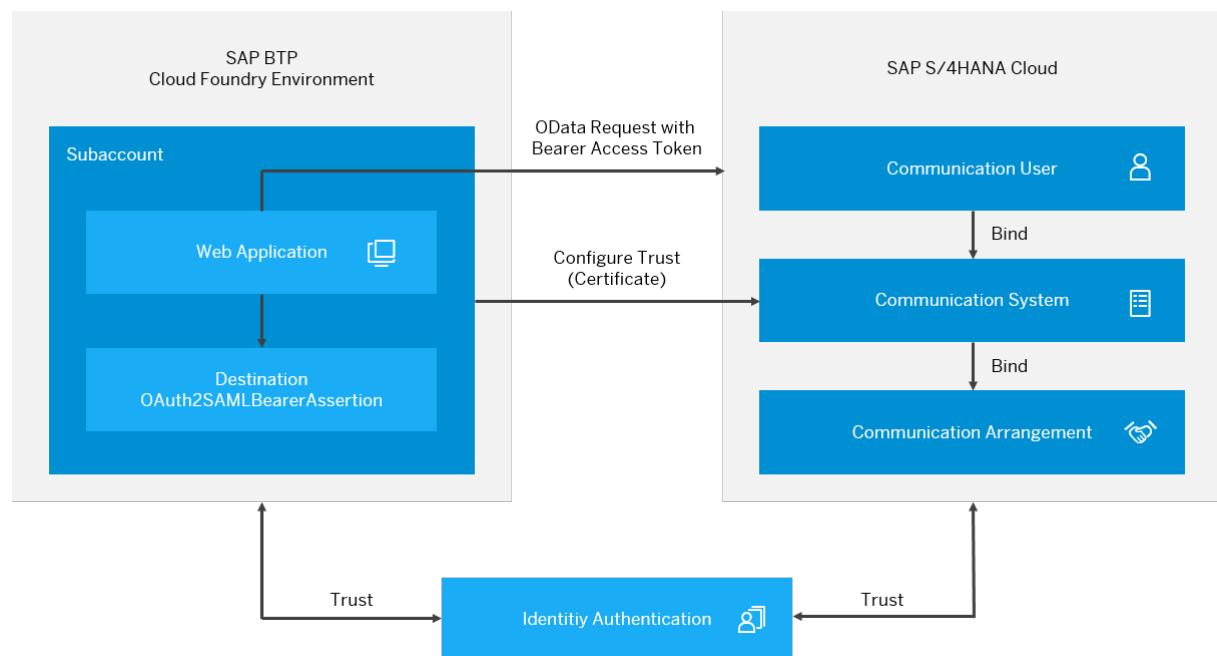
[Prerequisites \[page 129\]](#)

[Configuration Tasks \[page 129\]](#)

Scenario

As a customer, you own an SAP BTP global account and have created at least one subaccount therein. Within the subaccount, you have deployed a Web application. Authentication against the Web application is based on a trusted identity provider (IdP) that you need to configure for the subaccount.

On the S/4HANA Cloud side, you own an S/4HANA ABAP tenant. Authentication against the S/4HANA ABAP tenant is based on the trusted IdP which is always your Identity Authentication Service (IAS) tenant. Typically, you will configure this S/4HANA Cloud Identity tenant to forward authentication requests to your corporate IdP.



Prerequisites

- You have an S/4HANA Cloud tenant and a user with the following business catalogs assigned:

Business Role ID	Area
SAP_BCR_CORE_COM	Communication Management
SAP_BCR_CORE_IAM	Identity and Access Management
SAP_BCR_CORE_EXT	Extensibility

- You have administrator permission for the configured S/4HANA Cloud IAS tenant.
- You have a subaccount and PaaS tenant in the SAP BTP Cloud Foundry environment.

Next Step

- Configuration Tasks [page 129]

1.1.3.4.2.2.1 Configuration Tasks

Perform these steps to set up user propagation between S/4HANA Cloud and the SAP BTP Cloud Foundry environment.

Tasks

1. Configure Single Sign-On between S/4HANA Cloud and the Cloud Foundry Organization on SAP BTP [page 129]
2. Configure OAuth Communication [page 130]
3. Configure Communication Settings in S/4HANA Cloud [page 130]
4. Configure Communication Settings in SAP BTP [page 134]
5. Consume the Destination and Execute the Scenario [page 136]

Configure Single Sign-On between S/4HANA Cloud and the Cloud Foundry Organization on SAP BTP

To configure SSO with S/4HANA you must configure trust between the S/4HANA IAS tenant and the Cloud Foundry organization, see [Manually Establish Trust and Federation Between UAA and Identity Authentication](#).

Configure OAuth Communication

Download the certificate from your Cloud Foundry subaccount on SAP BTP.

1. From the SAP BTP cockpit, choose [Cloud Foundry environment](#) [your global account](#) .
2. Choose or create a subaccount, and from your left-side subaccount menu, go to [Connectivity](#) [Destinations](#) .
3. Press the *Download Trust* button.

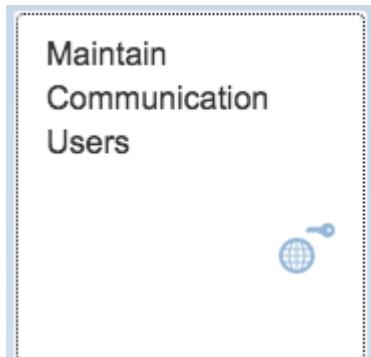
The screenshot shows the SAP BTP Cockpit interface. The left sidebar has sections for Overview, Services, Subscriptions, Cloud Foundry, Connectivity, Destinations (which is selected and highlighted with a blue border), Security, Entitlements, and Usage Analytics. The main content area shows a subaccount named "trial - Destinations" with one item listed: "All: 1". Below this is a table with columns for Type, Name, Basic Properties, and Actions. A single row is shown for an "HTTP" type destination named "test". In the "Basic Properties" column, it shows "Authentication: NoAuthentication", "ProxyType: Internet", and "URL: http://sap.com". At the top of the table, there are buttons for New Destination, Import Destination, Certificates, Download Trust (which is highlighted with a red box), and Renew Trust. A search bar is also present at the top right of the main content area.

Back to [Tasks \[page 129\]](#)

Configure Communication Settings in S/4HANA Cloud

1. Create a Communication User

1. In your S/4HANA Cloud launchpad, choose the application [Maintain Communication Users](#).



2. From the *User List* view, create a new user.
3. Set *<User Name>*, *<Password>* and *<Description>*.
4. Copy this password, you will need it in a later step.

5. Press the **Save** on the bottom of the screen.

VIKTOR

User ID: C0000000

User Data

*User Name:	VIKTOR	*Description:	VIKTOR
User ID:	C0000000	User Lock Status:	<input type="checkbox"/>

Password

Password:	*****
	Password Status: Productive

[Propose Password](#)

Certificate

Subject:	
Issuer:	

[Remove Certificate](#) [Upload Certificate](#)

Used by Communication Systems

System ID	System Name	Description	Host Name
HCPEXT_CF	HCPEXT_CF		int.sap.hana.ondemand.com

[Save](#) [Cancel](#)

6. Close the *Communication Users* application.
2. **Set up a Communication System for OAuth**
 1. From the launchpad, choose the application *Communication Systems*.



2. From the list view, select *New*.
3. A popup window appears. Enter the <System ID> and the <System Name>, then choose *Create*.

New Communication System

*System ID:	HCPEXT_CF
*System Name:	HCPEXT_CF

Create **Cancel**

4. Enter the host name. This is your Cloud Foundry region, for example: cf.eu10.hana.ondemand.com for Europe (Frankfurt).

i Note

For the complete list of standard regions, see [Regions](#).

General

*Host Name:	<...>.hana.ondemand.com
Logical System:	
HTTPS Port:	443

5. Enable the OAuth Identity Provider.

OAuth 2.0 Identity Provider

Enabled <input checked="" type="checkbox"/>
Provider Name <input type="text"/>
Signing Certificate ...
Signing Certificate I...

6. Upload the subaccount certificate that you have downloaded before from the SAP BTP cockpit.

OAuth 2.0 Identity Provider

Enabled:

*Provider Name: Upload Signing Certificate

Signing Certificate Subject: OU=CP Destination
Configuration,O=SAP,CN=cfapps.sap.hana.ondemand.com/
a352a17b- <...>

Signing Certificate Issuer: OU=CP Destination
Configuration,O=SAP,CN=cfapps.sap.hana.ondemand.com/
a352a17b- <...>

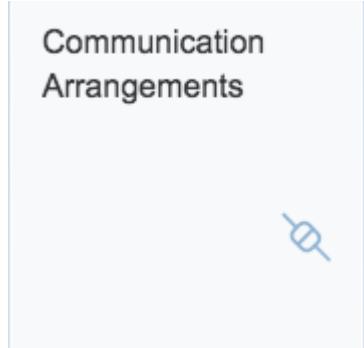
7. From <Signing Certificate Subject>, copy the CN value (for example: cfapps.sap.hana.ondemand.com/a352a17b-<...>) and paste it in the field <Provider Name>.
8. Add the *Communication User* you have created in the previous step.

User for Inbound Communication		+
Authentication Method	User Name	
User ID and Password	VIKTOR	×

9. Save your settings and go back to the launchpad.

3. Create a Communication Arrangement

1. Start the *Communication Arrangements* application.



2. From the list view, select *New*.
3. In the popup, choose a scenario. For our example, we use SAP_COM_0013. Set the arrangement name, for example **SAP_COM_0013_MY_TEST**.
4. In the *Common Data* section of the configuration screen, select the <Communication System> that you have created in the step before. The communication user is added automatically in the *Inbound Communication* section, and the <Authentication Method> is set to OAuth 2.0.

SAP_COM_0013_CV_CF

Scenario ID: SAP_COM_0013 Scenario Description: SAP Web IDE Integration Draft Last Changed By: Draft Last Changed On: 12/01/2017, 12:42:01

Editing Status: Draft

Common Data

Arrangement Name:	SAP_COM_0013_CV_CF	Own System:
*Communication System:	HCPEXT_CF	HCPEXT_CF

Inbound Communication

*User Name:	VIKTOR	Authentication Method	OAuth 2.0
-------------	--------	-----------------------	-----------

Inbound Services

Service	Application Protocol	Service URL / Service Interface	WSDL	Additional Properties
Catalog Service Version 2	OData V2	https://my300117-api.s4hana.ondemand.com/sap/opu/odata/IWFND/CATALOGSERVICE;v=2		
Gateway service for ADT	OData V2	https://my300117-api.s4hana.ondemand.com/sap/opu/odata/sap/ADT_SRV		
UI2 App Index Services	OData V2			

5. In the *Outbound Services* section, go to [Launch SAP Web IDE](#) and uncheck the Active checkbox of the field <Service Status>.

Outbound Services

[▼ Launch SAP Web IDE](#)

Service Status:	<input type="checkbox"/> Active	Path:	/
Application Protocol:	UI Link	Service URL:	
Port:	443		

6. Save your settings and go back to the launchpad.

Back to [Tasks \[page 129\]](#)

Configure Communication Settings in SAP BTP

1. From the SAP BTP cockpit, choose  .
2. Choose your subaccount, and from the left-side subaccount menu, go to  .
3. Press the [New Destination](#) button.
4. Enter the following parameters for your destination:

Parameter	Value
Name	Enter a meaningful name.
Type	HTTP

Parameter	Value
Description	(Optional) Enter a meaningful description.
URL	The OData URL, for example <code>https://my300117-api.s4hana.ondemand.com/sap/opu/odata/IWFND/CATALOGSERVICE;v=2;\$format=json</code>
Proxy Type	Internet
Authentication	OAuth2SAMLBearerAssertion
Audience	<p>The URL of your SAP S/4HANA Cloud account.</p> <p>To get it, log on to your SAP S/4HANA Cloud account. Select the profile picture. Then choose <i>Settings</i> and copy the value from the <code><Server></code> field. Add <code>https://</code> to the beginning of this string, for example, <code>https://my300117.s4hana.ondemand.com</code>.</p>
<p>i Note</p> <p>This URL does not contain <code>my300117-api</code>, but only <code>my300117</code>.</p>	
Client Key	The name of the communication user you have in the SAP S/4HANA ABAP tenant, e.g VIKTOR.
Token Service URL	<p>For this field, you need the part of the URL before <code>/sap/...</code> that you copied before from <i>Communications Arrangements</i> service URL/service interface:</p> <pre>https://my300117-api.s4hana.ondemand.com/sap/bc/sec/oauth2/token?</pre> <p><code>scope=ADT_0001%20%2fUI5%2fAPP_INDEX_001%20%2fIWFND%2fSG_MED_CATALOG_0002</code></p>
<p>i Note</p> <p>This URL is pointing to the scope of the <i>Inbound Services</i> of the communication scenario that we have defined when creating the communication arrangement. The scopes have a fixed naming and are separated by %20 for the space and %2f for the slash :</p> <ul style="list-style-type: none"> ○ ADT_001: scope of the Gateway service for ADT. ○ /UI5/APP_INDEX_0001: scope of the UI2 App Index. ○ /IWFND/SG_MED_CATALOG_0002: scope of the Catalog service version 2.0. 	
Token Service User	The same user as for the Client Key parameter.

Parameter	Value
Token Service Password	The password for the communication user.
System User	This parameter is not used, leave the field empty.
authnContextClassRef	urn:oasis:names:tc:SAML:2.0:ac:classes:X509

Destination Configuration

*Name:	my300117_	Additional Properties
Type:	HTTP	<input type="checkbox"/> authnContextClass <input type="checkbox"/> urn:oasis:names:tc:SAML:2.0:ac:classes:X509 <input type="checkbox"/>
Description:		
*URL:	https://my300117-api.s4hana.ondemand.com/sap/opu/odata/	
Proxy Type:	Internet	<input checked="" type="checkbox"/> Use default JDK truststore
Authentication:	OAuth2SAMLBearerAssertion	
*Audience:	https://my300117.s4hana.ondemand.com	
*Client Key:	*****	
*Token Service URL:	https://my300117-api.s4hana.ondemand.com/sap/bc/sec/oau	
Token Service User:	VIKTOR	
Token Service Password:	*****	
System User:		

Back to [Tasks \[page 129\]](#)

Consume the Destination and Execute the Scenario

To perform the scenario and execute the request from the source application towards the target application, proceed as follows:

1. Decide on where the user identity will be located when calling the Destination service. For details, see [User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#). This will determine how exactly you will perform step 2.
2. Execute a "find destination" request from the source application to the Destination service. For details, see [Consuming the Destination Service \[page 191\]](#) and the [REST API documentation](#).
3. From the Destination service response, extract the access token and URL, and construct your request to the target application. See ["Find Destination" Response Structure \[page 199\]](#) for details on the structure of the response from the Destination service.

Back to [Tasks \[page 129\]](#)

1.1.3.4.2.3 User Propagation from the Cloud Foundry Environment to SAP SuccessFactors

Configure user propagation from the SAP BTP Cloud Foundry environment to SAP SuccessFactors.

Steps

[Scenario \[page 137\]](#)

[Prerequisites \[page 137\]](#)

[Concept Overview \[page 137\]](#)

[Create an OAuth Client in SAP SuccessFactors \[page 138\]](#)

[Create and Consume a Destination for the Cloud Foundry Application \[page 140\]](#)

Scenario

- From an application in the SAP BTP Cloud Foundry environment, you want to consume OData APIs exposed by SuccessFactors modules.
- To enable single sign-on, you want to propagate the identity of the application's logged-in user to SuccessFactors.

Prerequisites

- In your Cloud Foundry space, you have a deployed application.
- You have an instance of the Destination Service that is bound to the application.
- An instance of the xsuaa service with `application` plan is bound to the application.

Concept Overview

A user logs in to the Cloud Foundry application. Its identity is established by an **Identity Provider** (IdP). This could be the default IdP for the Cloud Foundry subaccount or a trusted IdP, for example the SuccessFactors IdP.

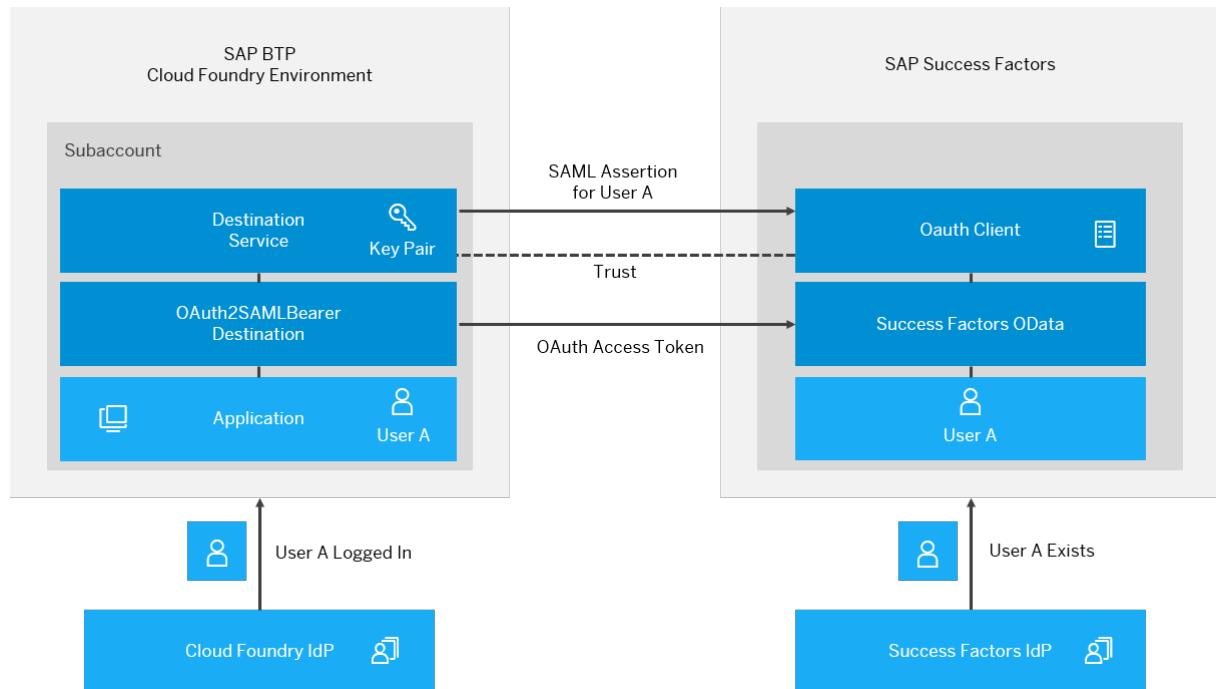
When the application retrieves an `OAuth2SAMLBearerToken` destination, the user is made available to the Cloud Foundry Destination service by means of a **user exchange token**, represented by a JSON Web Token (JWT).

The service then wraps the user identity in a SAML assertion, signs it with the Cloud Foundry subaccount private key and sends it to the token endpoint of the SuccessFactors OAuth server.

To accept the SAML assertion and return an **access token**, a **trust** relationship must be set up between SuccessFactors and the Cloud Foundry subaccount public key. You can achieve this by providing the Cloud Foundry subaccount **X.509 certificate** when creating the OAuth client in SuccessFactors.

Users that are propagated from the Cloud Foundry application, are verified by the SuccessFactors OAuth server before granting them access tokens. This means, users that do not exist in the SuccessFactors user store will be rejected.

For valid users, the OAuth server accepts the SAML assertion and returns an OAuth access token. In turn, the Destination service returns both the destination and the access token to the requesting application. The application then uses the destination properties and the access token to consume SuccessFactors APIs.



Next Steps

- Create an OAuth Client in SAP SuccessFactors [page 138]
- Create and Consume a Destination for the Cloud Foundry Application [page 140]

1.1.3.4.2.3.1 Create an OAuth Client in SAP SuccessFactors

Create an OAuth client in SuccessFactors for user propagation from the SAP BTP Cloud Foundry environment.

1. Download the X.509 certificate from your Cloud Foundry subaccount:

In the cloud cockpit, navigate to your Cloud Foundry subaccount and from the left-side subaccount menu, choose **Connectivity > Destinations**. Choose **Download Trust** to get the certificate for this subaccount.

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has a 'Destinations' section selected. The main area displays a table with one row for an 'HTTP' destination named 'test'. The 'Certificates' column contains a link labeled 'Download Trust', which is highlighted with a red box. Other buttons in the row include 'New Destination', 'Import Destination', 'Certificates', 'Download Trust' (highlighted), and 'Renew Trust'.

2. Create a SuccessFactors OAuth Client:

In SuccessFactors, go to the **Admin Center** and search for **OAuth**. Choose **Manage OAuth2 Client Applications**.

The top screenshot shows the SuccessFactors Admin Center. It features a sidebar with various management modules like Objective Management, Calibration, Recruiting, Payroll, etc. The main content area has sections for Home, SAP Jam, Objectives, Performance, and Continuous Performance. A 'Tool Search' bar at the top right has 'oauth' typed into it. The bottom navigation bar has a 'Admin Center' link highlighted with a red box. The bottom screenshot shows the 'Company Processes' page. It has a similar layout with a sidebar and main content area. A 'Tool Search' bar at the top right also has 'oauth' typed into it. A button labeled 'Manage OAuth2 Client Applications' is highlighted with a red box in the top right corner of the main content area.

3. Press the **Register Client Application** button on the right. In the **<Application Name>** field, provide some arbitrary descriptive name for the client. For **<Application URL>**, enter the Cloud Foundry host of the application, followed by the subaccount GUID, for example `cfapps.stagingaws.hanavlab.ondemand.com/17d146c3-bc6c-4424-8360-7d56ee73bd32`. This information is available in the cloud cockpit under subaccount details:

The screenshot shows two sections: 'Subaccount Details' and 'Cloud Foundry'. In 'Subaccount Details', there is a field for 'Subdomain: trial' and 'ID: 17d146c3-bc6c-4424-' which is highlighted with a red box. In 'Cloud Foundry', there is an 'Organization: trial_trial', 'Spaces: 1', 'Members: 2', and 'API Endpoint: https://api.cf.stagingaws.hanavlab.ondemand.com' which is also highlighted with a red box. A 'Disable Cloud Foundry' button is visible at the bottom.

4. In the field <[X.509 Certificate](#)>, paste the certificate that you downloaded in step 1.
5. Choose [Register](#) to save the OAuth client.
6. Now, locate your client in the list by its application name, choose [View](#) in the [Actions](#) column and take note of the <[API Key](#)> that has been generated for it. You will use this key later in the OAuth2SAMLBearer destination in the Cloud Foundry environment.

Manage OAuth2 Client Applications

CAUTION: External OAuth works as an enhanced internal OAuth, it is very powerful, so please use with caution.
[View an existing OAuth Client Application](#)

The screenshot shows a form for creating a new OAuth2 Client Application. The fields are as follows:

- Company:** 019820
- Application Name:** staging
- Description:** (empty text area)
- API Key:** ZGNmNDNhMWI5NjU2MD
- Shared Secret:** (empty text area)
- Application URL:** <http://cfapps.stagingaws.hanavlab.ondemand.com/17d146c3-bc6c-4424->
- X.509 Certificate:** A large text area containing a base64 encoded X.509 certificate, starting with MIIFnDCCA4SgAwIBAgIKG+kbRaI0aM6z8zANBkgkqkhiG9w0BAQ0FADCBjDFVMFMGA1UEAwxMY2ZhcHBzLnN0YWdpbmhd3MuAGFuYXZsYWlub25kZW1hbhmQuY29tLzE3ZDE0NmMzLWJjNmMtNDQyNC04MzYwLTdkNTZTczYmQzMjE MMAoGA1UECgwDU0FQMSUwlvYDVQQLDbxDUCBEZXN0aW5hdGlvbIBDb25maWd1cmF0aW9uMB4XDTE3MTEyNDExMTk1MFoXDTE4MDUyNDExMTk1MFowgYwxVTBTBgNVBAMMTGNmYXBwc5zdGFnaW5nYXdzLmhbbmF2bGFLm9uZGVVYw5kLmNvbS8xN2QxNDZjMy1YZZjLTQ0MjQtODM2MC03ZDU22WU3M2JkMzIxDDAKBgNVBAoMA1NBUDEM CMGA1UECwwcQ1AgRGVzdGluYXRpb24gQ29uZmndXJhdGlvbjCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgcggIBALIxkjxLP7TGhCJLdw/3mWDVHFdlMwQZ/y+FgqjLM0VfPj5j6HtxVK8jfNjCx99iaiKmPjpIZsEVJb7uer388d7Cq+1izJ 75SLJVU01UEEAuMUs6PqEjk7uJuwagmnfh8dtHZ+pvIMQ2lonXL8xyA1VNj0mO4yYxMKWaGUfFq+voy1ehkLR3V7H41ZNKyHwiAq8ZhiJf1mO1Qk8OeYHq6Zl9NPS2PTR4hW70cyjrFATBFo2A+w1xk/WO70lpJhmHdelb3LcdbrZ3fJaRt2m/UPC

Next Step

- [Create and Consume a Destination for the Cloud Foundry Application \[page 140\]](#)

1.1.3.4.2.3.2 Create and Consume a Destination for the Cloud Foundry Application

Create and consume an OAuth2SAMLBearerAssertion destination for your Cloud Foundry application.

Create the Destination

1. In the cloud cockpit, navigate to your Cloud Foundry subaccount and from the left-side subaccount menu, choose . Choose *New Destination* and enter a name Then provide the following settings:
 - <URL>: URL of the SuccessFactors OData API you want to consume.
 - <Authentication>: **OAuth2SAMLBearerAssertion**
 - <Audience>: **www.successfactors.com**
 - <Client Key>: API Key of the OAuth client you created in SuccessFactors.
 - <Token Service URL>: API endpoint URL for the SuccessFactors instance, followed by /oauth/ token and the URL parameter company_id with the company ID, for example `https://apisalesdemo2.successfactors.eu/oauth/token?company_id=SFPART019820`.
2. Enter three additional properties:
 - **apiKey**: the API Key of the OAuth client you created in SuccessFactors.
 - **authnContextClassRef**: `urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession`
 - **nameIdFormat**:
 - `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`, if the **user ID** will be propagated to a SuccessFactors application, or
 - `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`, if the **user e-mail** will be propagated to SuccessFactors.

The screenshot shows the 'Destination Configuration' dialog box. The main panel contains the following fields:

*Name:	sf_sf_serious_destination
Type:	HTTP
Description:	
*URL:	<code>https://salesdemo.successfactors.eu/oauth/token?company_id=SFPART019820</code>
Proxy Type:	Internet
Authentication:	OAuth2SAMLBearerAssertion
*Audience:	<code>www.successfactors.com</code>
*Client Key:	*****
*Token Service URL:	<code>https://salesdemo.successfactors.eu/oauth/token?company_id=SFPART019820</code>
Token Service User:	
Token Service Password:	
System User:	

On the right side, under 'Additional Properties', there are three entries:

apiKey	ZGNmNDNhMWl5Nj...
authnContextCla...	urn:oasis:names:tc:S...
nameIdFormat	urn:oasis:names:tc:S...

A checkbox labeled 'Use default JDK truststore' is checked.

At the bottom of the dialog are buttons for Edit, Clone, Export, Delete, and Check Connection.

Consume the Destination and Execute the Scenario

To perform the scenario and execute the request from the source application towards the target application, proceed as follows:

1. Decide on where the user identity will be located when calling the Destination service. For details, see [User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#). This will determine how exactly you will perform step 2.
2. Execute a "find destination" request from the source application to the Destination service. For details, see [Consuming the Destination Service \[page 191\]](#) and the [REST API documentation](#).

- From the Destination service response, extract the access token and URL, and construct your request to the target application. See "[Find Destination](#)" [Response Structure \[page 199\]](#) for details on the structure of the response from the Destination service.

1.1.3.4.2.4 User Propagation between Cloud Foundry Applications

Propagate the identity of a user between Cloud Foundry applications that are located in different subaccounts or regions.

Steps

[Scenario \[page 142\]](#)

[Prerequisites \[page 142\]](#)

[Concept \[page 143\]](#)

[Procedure \[page 145\]](#)

- Assemble IdP Metadata for Subaccount 1 [\[page 145\]](#)
- Establish Trust between Subaccount 1 and Subaccount 2 [\[page 146\]](#)
- Create an OAuthSAMLBearerAssertion Destination for Application 1 [\[page 147\]](#)
- Consume the Destination and Execute the Scenario [\[page 149\]](#)

Scenario

- You have deployed an application in a Cloud Foundry environment (**application 1**).
- You want to call another Cloud Foundry application (**application 2**) in a different subaccount, in the same or another region.
- You want to propagate the identity of the user that is logged in to application 1, to application 2.

[Back to Steps \[page 142\]](#)

Prerequisites

- You have two applications (application 1 and application 2) deployed in Cloud Foundry spaces in different subaccounts in the same region or even in different regions.

- You have an instance of the Destination service bound to application 1.
- You have a user JWT (JSON Web Token) in application 1 where the call to application 2 is performed.

Back to [Steps \[page 142\]](#)

Concept

The identity of a user logged in to application 1 is established by an identity provider (IdP) of the respective subaccount (**subaccount 1**).

i Note

You can use the default IdP for the Cloud Foundry subaccount or a custom-configured IdP.

When the application retrieves an OAuthSAMLBearer destination, the user is made available to the Cloud Foundry Destination service by means of a *user exchange* JWT. See also [User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#).

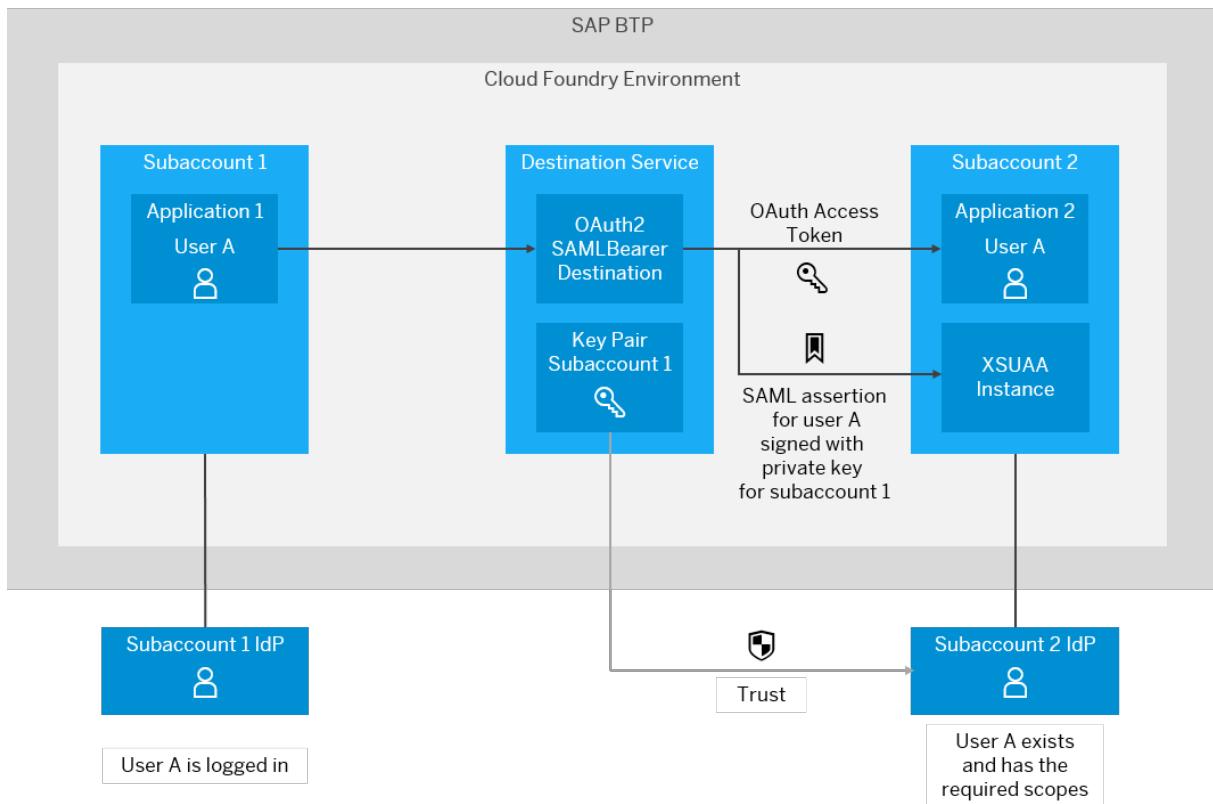
The service then wraps the user identity in a SAML assertion, signs it with subaccount 1's private key (which is part of the special key pair for the subaccount, maintained by the Destination service) and sends it to the authentication endpoint of **subaccount 2**, which hosts application 2.

To make the authentication endpoint accept the SAML assertion and return an access token, you must set up a trust relationship between the two subaccounts, by using subaccount 1's public key. You can achieve this by assembling the SAML IdP metadata, using subaccount 1's public key and setting up a new trust configuration for subaccount 2, which is based on that metadata.

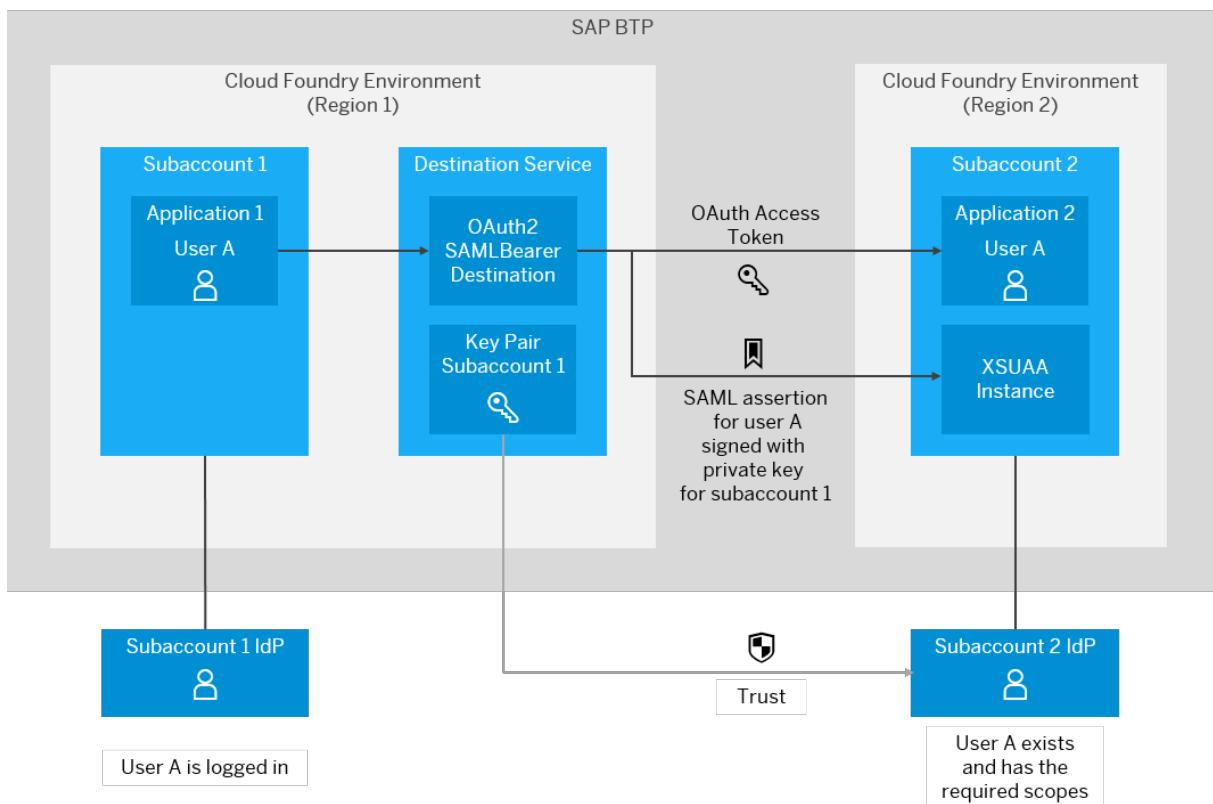
This way, users propagated from application 1 can be verified by subaccount 2's IdP before granting them access tokens with their respective scopes in the context of subaccount 2.

The authentication endpoint accepts the SAML assertion and returns an OAuth access token. In turn, the Destination service returns both the destination configuration and the access token to the requesting application (application 1). Application 1 then uses the destination properties and the access token to call application 2.

Option 1 - Setting up Trust between Subaccounts in the Same Region



Option 2 - Setting up Trust between Subaccounts in Different Regions



[Back to Steps \[page 142\]](#)

Procedure

Assemble IdP Metadata for Subaccount 1

1. Download the X.509 certificate of subaccount 1. For instructions, see [Set up Trust Between Systems \[page 126\]](#). The content of the file is shown as:

```
-----BEGIN CERTIFICATE-----<content>-----END CERTIFICATE-----
```

Below, we refer to the value of <content> as \${S1_CERTIFICATE}.

2. In the cockpit, navigate to the overview page of subaccount 1. For details, see [Navigate in the Cockpit](#). Here you can see the landscape domain, subaccount ID and subdomain. Below, we refer to the landscape domain as \${S1_LANDSCAPE_DOMAIN}, to the subaccount ID as \${S1_SUBACCOUNT_ID} and to the subdomain as \${S1_SUBDOMAIN}.

3. In your browser, call `https://${S1_SUBDOMAIN}.authentication.${S1_LANDSCAPE_DOMAIN}/saml/metadata` and download the XML file. Within the XML file you can find the following structure:

Sample Code

```
<?xml version="1.0" encoding="UTF-8"?>
...
<md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:URI" Location="https://${S1_SUBDOMAIN}.authentication.${S1_LANDSCAPE_DOMAIN}/oauth/token/alias/<alias>" index="1"/>
...
```

Below, we refer to the value of <alias> as \${S1_ALIAS}.

4. Assemble the new IdP metadata for subaccount 1 by replacing the \${...} placeholders in the following template with the values determined in the previous steps:

Sample Code

```
<ns3:EntityDescriptor
  ID="cfapps.${S1_LANDSCAPE_DOMAIN}/${S1_SUBACCOUNT_ID}"
  entityID="cfapps.${S1_LANDSCAPE_DOMAIN}/${S1_SUBACCOUNT_ID}"
  xmlns="http://www.w3.org/2000/09/xmldsig#"
```

```

xmlns:ns2="http://www.w3.org/2001/04/xmlenc#"
xmlns:ns4="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:ns3="urn:oasis:names:tc:SAML:2.0:metadata">
<ns3:SPSSODescriptor AuthnRequestsSigned="true"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <ns3:KeyDescriptor use="signing">
        <KeyInfo>
            <KeyName>${S1_ALIAS}</KeyName>
            <X509Data>
                <X509Certificate>
                    ${S1_CERTIFICATE}
                </X509Certificate>
            </X509Data>
        </KeyInfo>
    </ns3:KeyDescriptor>
</ns3:SPSSODescriptor>
<ns3:IDPSSODescriptor
    WantAuthnRequestsSigned="true"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <ns3:KeyDescriptor use="signing">
        <KeyInfo>
            <KeyName>${S1_ALIAS}</KeyName>
            <X509Data>
                <X509Certificate>
                    ${S1_CERTIFICATE}
                </X509Certificate>
            </X509Data>
        </KeyInfo>
    </ns3:KeyDescriptor>
</ns3:IDPSSODescriptor>
</ns3:EntityDescriptor>

```

[Back to Steps \[page 142\]](#)

Establish Trust between Subaccount 1 and Subaccount 2

1. In the cockpit, navigate to the overview page for subaccount 2.
2. From the left panel, select Choose *New Trust Configuration*. For details, see [Establish Trust and Federation with UAA Using Any SAML Identity Provider](#).
3. Paste the assembled IdP metadata for subaccount 1 in the `<Metadata>` text box and uncheck *Available for User Logon*.

New Trust Configuration

Metadata:^{*}

```
9exej/8vILgRH8HUFFinxGxh1mPy1O0kUHG9bDvgYkvNwxJ6hQL5cEh1WipOYtp  
tNUipfreWwPkpwAVY97SjuqH/u4Bgvq30+sVPDsS+B1v5XD1S6RMtKB332K0Z7I  
a9XHZK9jpwCIMcSSHTSdENN8nvQX59CVuwU+S38V4q2JJ7HUIPNmtTS51QYzPF  
kV  
8p0823u5+NgLJJA8fbWrhlwgDS++oZAqlTffw==  
    </X509Certificate>  
    </X509Data>  
    <KeyInfo>  
        </ns3:KeyDescriptor>  
        </ns3:IDPSSODescriptor>  
    </ns3:EntityDescriptor>
```

Name:^{*}

Description:

Origin Key:^{*}

Status:

Available for User Logon: ←

Link Text for User Logon:

Create Shadow Users During Logon:

4. Choose [Parse](#).
5. Enter a [`<Name>`](#) for the trust configuration and choose [Save](#).

i Note

Additionally, you must add users to this new trust configuration and assign appropriate scopes to them.

[Back to Steps \[page 142\]](#)

Create an OAuthSAMLBearerAssertion Destination for Application 1

1. In the cockpit, navigate to the overview page for subaccount 2.
2. Here you can see the landscape domain, subaccount ID and subdomain of subaccount 2. Below, we refer to the landscape domain as `${S2_LANDSCAPE_DOMAIN}`, to the subaccount ID as `${S2_SUBACCOUNT_ID}` and to the subdomain as `${S2_SUBDOMAIN}`.

The screenshot shows the SAP BTP Cockpit interface. In the left sidebar, under the 'Subscriptions' section, 'Cloud Foundry' is selected. On the main page, 'Subaccount: 2' is shown with a subdomain ID of 'eu10.hana.ondemand.com'. There is one active subscription. Below this, the 'Cloud Foundry Environment' tab is selected, showing an org name '6a3af0fbtrial', org ID '3a593519-9d5c-4b16-aee7-6d3b2ea60d5f', and members count '1'. The API endpoint listed is 'https://api.eu10.hana.ondemand.com'. A table below lists two spaces: 'dev' and 'trialfm', each with its respective applications and service instances.

- In your browser, call `https://${S2_SUBDOMAIN}.authentication.${S2_LANDSCAPE_DOMAIN}/saml/metadata` and download the XML file. Within the XML file, you can find the following structure. It contains the `<audience>` and the `<alias>` variables:

↳ Sample Code

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor entityID="<audience>" ...>
...
  <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:URI" Location="https://${S2_SUBDOMAIN}.authentication.${S2_LANDSCAPE_DOMAIN}/oauth/token/alias/<alias>" index="1"/>
...

```

Below, we refer to the value of `<alias>` as `${S2_ALIAS}` and `<audience>` as `${S2_AUDIENCE}`.

- In cockpit, navigate to subaccount 1.
- From the left panel, select **Connectivity** **Destinations**.
- Choose **New Destination** and configure the values as described below. Replace the `${...}` placeholders with the values you determined in the previous steps and sections.

Property	Value
Name	Choose any name for your destination. You will use this name to request the destination from the Destination service.
Type	HTTP
URL	The URL of application 2, identifying the resource you want to consume.
Proxy Type	Internet
Authentication	OAuth2SAMLBearerAssertion
Audience	<code> \${S2_AUDIENCE}</code>
Client Key	The <code>clientid</code> of the XSUAA instance in subaccount 2. Can be acquired via a binding or service key.
Token Service URL	<code>https://\${S2_SUBDOMAIN}.authentication.\${S2_LANDSCAPE_DOMAIN}/oauth/token/alias/\${S2_ALIAS}</code>

Property	Value
Token Service URL Type	Dedicated
Token Service User	The <i>clientid</i> of the XSUAA instance in subaccount 2. Can be acquired via a binding or service key.
Token Service Password	The <i>clientsecret</i> of the XSUAA instance in subaccount 2. Can be acquired via a binding or service key.
authnContextClassRef	urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession

Additional Properties

Property	Value
nameIdFormat	urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress

Example

Destination Configuration

7. Choose [Save](#).

Back to [Steps \[page 142\]](#)

Consume the Destination and Execute the Scenario

To perform the scenario and execute the request from application 1, targeting application 2, proceed as follows:

1. Decide on where the user identity will be located when calling the Destination service. For details, see [User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#). This will determine how exactly you will perform step 2.
2. Execute a "find destination" request from application 1 to the Destination service. For details, see [Consuming the Destination Service \[page 191\]](#) and the [REST API documentation](#).

- From the Destination service response, extract the access token and URL, and construct your request to application 2. See "[Find Destination](#)" Response Structure [page 199] for details on the structure of the response from the Destination service.

Back to [Steps](#) [page 142]

1.1.3.4.2.5 User Propagation from the Cloud Foundry Environment to the Neo Environment

Propagate the identity of a user from a Cloud Foundry application to a Neo application.

Steps

[Scenario](#) [page 150]

[Prerequisites](#) [page 151]

[Concept](#) [page 151]

[Procedure](#) [page 152]

- Configure a Local Service Provider for the Neo Subaccount [page 152]
- Establish Trust between Cloud Foundry and Neo Subaccounts [page 153]
- Create an OAuth Client for the Neo Application [page 155]
- Create an OAuth2SAMLBearerAssertion Destination for the Cloud Foundry Application [page 155]
- Consume the Destination and Execute the Scenario [page 157]

Scenario

- You have deployed an application in the Cloud Foundry environment.
- You want to consume OAuth protected APIs exposed by an application deployed in the Neo environment.
- You want to propagate the identity of the user logged in to the Cloud Foundry application, to the Neo application.

Back to [Steps](#) [page 150]

Prerequisites

- You have deployed an application in the Cloud Foundry environment.
- You have bound an instance of the Destination Service to the application.
- You have bound an instance of the XSUAA service with the application plan to the application.
- You have deployed an application in the Neo environment.

Back to [Steps \[page 150\]](#)

Concept

The identity of a user logged in to the Cloud Foundry application is established by an identity provider (IdP).

i Note

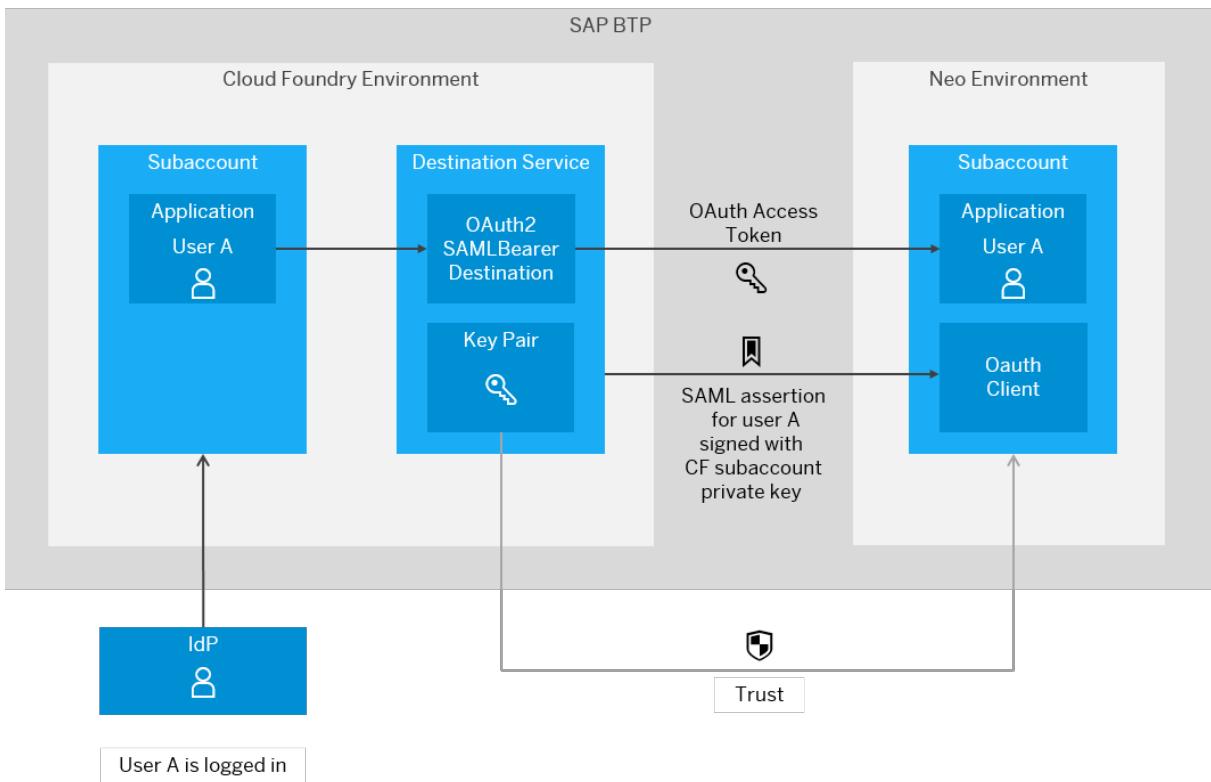
You can use the default IdP of the Cloud Foundry subaccount or any trusted IdP, for example, the Neo subaccount IdP.

When the application retrieves an OAuthSAMLBearer destination, the user is made available to Cloud Foundry Destination service by means of a *user exchange* JWT (JSON Web Token).

The service then wraps the user identity in a SAML assertion, signs it with the Cloud Foundry subaccount private key and sends it to the token endpoint of the OAuth service for the Neo application.

To make the Neo application accept the SAML assertion, you must set up a trust relationship between the Neo subaccount and the Cloud Foundry subaccount public key. You can achieve this by adding the Cloud Foundry subaccount X.509 certificate as trusted IdP in the Neo subaccount. Thus, the Cloud Foundry application starts acting as an IdP and any users propagated by it are accepted by the Neo application, even users that do not exist in the IdP.

The OAuth service accepts the SAML assertion and returns an OAuth access token. In turn, the Destination service returns both the destination and the access token to the requesting application. The application then uses the destination properties and the access token to consume the remote API.



Back to [Steps \[page 150\]](#)

Procedure

Configure a Local Service Provider for the Neo Subaccount

1. In the cockpit, navigate to your Neo subaccount, choose **Security > Trust** from the left menu, and go to tab **Local Service Provider** on the right. For **Configuration Type**, select **Custom** and choose **Generate Key Pair**.
2. **Save** the configuration.

i Note

IMPORTANT: When you choose **Custom** for the **Local Service Provider** type, the default IdP (SAP ID service) will no longer be available. If your scenario requires login to the SAP ID service as well, you can safely skip this step and leave the default settings for the Local Service Provider.

Back to [Steps \[page 150\]](#)

Establish Trust between Cloud Foundry and Neo Subaccounts

- Download the X.509 certificate of the Cloud Foundry subaccount:

In the cockpit, navigate to your Cloud Foundry subaccount and choose **Connectivity > Destinations**. Press [Download Trust](#) to get the certificate for this subaccount.

- Configure trust in the Neo subaccount:

In the cockpit, navigate to your Neo subaccount, choose **Trust** from the left menu, and select the **Application Identity Provider** tab on the right. Then choose [Add Trusted Identity Provider](#).

The screenshot shows the SAP BTP Cockpit interface. On the left, there's a navigation sidebar with options like Subscriptions, Services, Solutions, Virtual Machines, SAP HANA / SAP ASE, Connectivity, Destinations, Cloud Connectors, Security, and Trust. The Trust option is currently selected. In the main content area, the URL is Home / Testing / FM_NEO. The page title is "Subaccount: FM_NEO - Trust". Under "Trust Management", the "Application Identity Provider" tab is active. A note says: "Note: The default configuration for the local service provider is selected. For this configuration, the SAP ID Service is the default trusted identity provider. To configure a custom identity provider, change the local service provider configuration type to "Custom"." There's a table with one row: Default Name https://accounts.sap.com, Description SAP ID Service, and a Delete button. A red box highlights the "Add Trusted Identity Provider" button at the top right.

In the <Name> field, enter the cfapps host followed by the subaccount GUID, for example `cfapps.sap.hana.ondemand.com/bf7f2876-5080-40ad-a56b-fff3ee5cff9d`. This information is available in the cockpit, on the overview page of your Cloud Foundry subaccount:

The screenshot shows the SAP BTP Cockpit interface. The navigation sidebar includes Overview, Services, Subscriptions, Cloud Foundry, Connectivity, Destinations, Security, Users, Role Collections, Roles, and Trust Configuration. The Cloud Foundry section is selected. The URL is Trial Home / trial / trial. The page title is "Subaccount: trial". It shows a subdomain of trial, an ID (redacted), and 1 active subscription. Below that, it lists the Cloud Foundry Environment (selected), Kyma Environment, and Entitlements. The Cloud Foundry Environment section shows Org Name: trial, Org ID: (redacted), Members: 1, and API Endpoint: `https://api.cf.eu10.hana.ondemand.com`. A Create Space button is visible. The screenshot also shows a table for Spaces (2) with rows for dev (1 application, 3 service instances) and trialfm (0 applications, 0 service instances). A Disable Cloud Foundry button is present.

In the <Signing Certificate> field, paste the X.509 certificate you downloaded in step 1. Make sure you remove the *BEGIN CERTIFICATE* and *END CERTIFICATE* strings. Then check *Only for IDP-Initiated SSO* and save the configuration:

Trusted Identity Provider

The screenshot shows the Trusted Identity Provider configuration page. The navigation sidebar includes Overview, Single Sign-On URL: `https://localhost`, Default Attributes, Assertion Attributes, Default Groups, Assertion Groups, Need Help?, How to configure a SAML identity provider, and How to manage roles and groups. The General tab is selected. It shows a table with fields: * Name: `cfapps.sap.hana.ondemand.com`, Description: `cfapps.sap.hana.ondemand.com`, Assertion Consumer Service: Application Root (default), Single Sign-On URL: `http(s)://www.example.com/saml2/sso`, Single Sign-On Binding: HTTP-POST, Single Logout URL: `http(s)://www.example.com/saml2/slo`, Single Logout Binding: HTTP-POST, Signature Algorithm: SHA-1, Signing Certificate: (redacted), User ID Source: subject, Source Value: (redacted), User ID Prefix: (redacted), User ID Suffix: (redacted), Enabled: checked, and Only for IDP-Initiated SSO: checked. A red box highlights the 'Only for IDP-Initiated SSO' checkbox.

[Back to Steps \[page 150\]](#)

Create an OAuth Client for the Neo Application

1. In the cockpit, navigate to the Neo subaccount, choose **Security > OAuth** from the left menu, select tab **Client**, and choose **Register New Client**:

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has a 'Security' section with 'OAuth' selected. The main area is titled 'Subaccount: FM_NEKO - OAuth'. It shows a table with columns 'Client ID', 'Name', 'Subscription', and 'Actions'. A red box highlights the 'Clients' tab in the top navigation bar and the 'Register New Client' button.

2. Enter a <Name> for the client.
3. In the <Subscription> field, select your Neo application.
4. For <Authorization Grant> select **Authorization Code**.
5. Check the **Confidential** checkbox and provide a secret for the OAuth client.

i Note

Make sure you remember the secret, because it will not be visible later.

6. <Redirect URI> is irrelevant for the OAuth SAML Bearer Assertion flow, so you can provide any URL in the Cloud Foundry application.

The screenshot shows the 'Register New Client' form. It includes fields for 'Name', 'Subscription' (set to 'services/dispatcher'), 'ID' (auto-generated as '1a15f44d-3738-3815-9f10-252609c84cd9'), 'Authorization Grant' (set to 'Authorization Code'), 'Confidential' (checkbox checked), 'Skip Consent Screen' (checkbox unchecked), and 'Redirect URI' (set to 'https://myclient.com/callback'). A red box highlights the 'Redirect URI' field.

Back to [Steps \[page 150\]](#)

Create an OAuth2SAMLBearerAssertion Destination for the Cloud Foundry Application

1. In the cockpit, navigate to the Cloud Foundry subaccount, choose **Connectivity > Destinations** from the left menu, select the **Client** tab and press **New Destination**.
2. Enter a <Name> for the destination, then provide:
 - o <URL>: the URL of the Neo application/API you want to consume.

- <Authentication>: **OAuth2SAMLBearerAssertion**
- <Audience>: can be taken from the Neo subaccount, if you choose ► **Security** ▶ **Trust** ▶ form the left menu, go to the *Local Service Provider* tab, and copy the value of <Local Provider Name>:

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has 'Security' selected under 'Trust'. The main area is titled 'Subaccount: FM_NEO - Trust' and 'Trust Management'. It shows tabs for 'Local Service Provider', 'Application Identity Provider', and 'Platform Identity Provider'. Under 'Local Service Provider', there's a section for 'Manage Local Provider Settings' with a 'Configuration Type' dropdown set to 'Custom'. The 'Local Provider Name' field contains a URL and is highlighted with a red box. Below it is a 'Signing Key' field containing a long string of characters.

- <Client Key>: the ID of the OAuth client for the Neo application
- <Token Service URL>: can be taken from the *Branding* tab in the Neo subaccount (choose ► **Security** ▶ **OAuth** ▶ from the left menu):

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has 'Security' selected under 'OAuth'. The main area is titled 'Subaccount: FM_NEO - OAuth' and 'OAuth Settings'. It shows tabs for 'Branding', 'Clients', and 'Platform API'. The 'Branding' tab is selected and highlighted with a red box. It includes fields for 'Logo Image' (with a 'Browse...' button), 'Default Theme' (selected), 'Custom Theme' (radio button), 'Background Color' (set to #CCCCCC), 'Button Color' (set to #007CC0), and 'Button Text Color' (set to #FFFFFF). Below this is a 'Save' button. The 'OAuth URLs' section contains fields for 'Authorization Endpoint', 'Token Endpoint' (highlighted with a red box), and 'End User Uri', all of which contain placeholder URLs.

- <Token Service User>: again the ID of the OAuth client for the Neo application.
- <Token Service Password>: the OAuth client secret.

Enter two additional properties:

- **authnContextClassRef**: `urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession`
- **nameIdFormat**:
 - `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`, if the user ID is propagated to the Neo application, or
 - `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`, if the user email is propagated to the Neo application.

Destination Configuration

The screenshot shows the 'Destination Configuration' page with the following fields filled in:

*Name:	neo
Type:	HTTP
Description:	
*URL:	https://neouserpropagationtrial.hana.ondemand.com
Proxy Type:	Internet
Authentication:	OAuth2SAMLBearerAssertion
*Audience:	https://hana.ondemand.com/trial
*Client Key:
*Token Service URL:	https://oauthasservices-trial.hana.ondemand.com/oauth2/api/
Token Service User:	2af27bf-4297-3faa-b1a2-bffae3a58fe8
Token Service Password:
System User:	

Additional Properties section:

authnContextClassRef	urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession
nameIDFormat	urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress

Use default JDK truststore

New Prop...

Back to [Steps \[page 150\]](#)

Consume the Destination and Execute the Scenario

To perform the scenario and execute the request from the source application towards the target application, proceed as follows:

1. Decide on where the user identity will be located when calling the Destination service. For details, see [User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#). This will determine how exactly you will perform step 2.
2. Execute a "find destination" request from the source application to the Destination service. For details, see [Consuming the Destination Service \[page 191\]](#) and the [REST API documentation](#).
3. From the Destination service response, extract the access token and URL, and construct your request to the target application. See ["Find Destination" Response Structure \[page 199\]](#) for details on the structure of the response from the Destination service.

Back to [Steps \[page 150\]](#)

1.1.3.5 Multitenancy in the Connectivity Service

Using multitenancy for Cloud Foundry applications that require a connection to a remote service or on-premise application.

Endpoint Configuration

Applications that require a connection to a remote service can use the Connectivity service to configure HTTP or RFC endpoints. In a provider-managed application, such an endpoint can either be once defined by the application provider ([Provider-Specific Destination \[page 158\]](#)), or by each application subscriber ([Subscriber-Specific Destination \[page 159\]](#)).

If the application needs to use the same endpoint, independently from the current application subscriber, the destination that contains the endpoint configuration is uploaded by the application provider. If the endpoint should be different for each application subscriber, the destination can be uploaded by each particular application subscriber.

i Note

This connectivity type is fully applicable also for on-demand to on-premise connectivity.

Destination Levels

You can configure destinations simultaneously on two levels: *subaccount* and *service instance*. This means that it is possible to have one and the same destination on more than one configuration level. For more information, see [Managing Destinations \[page 39\]](#).

Destination lookup according to the level, when configured on:

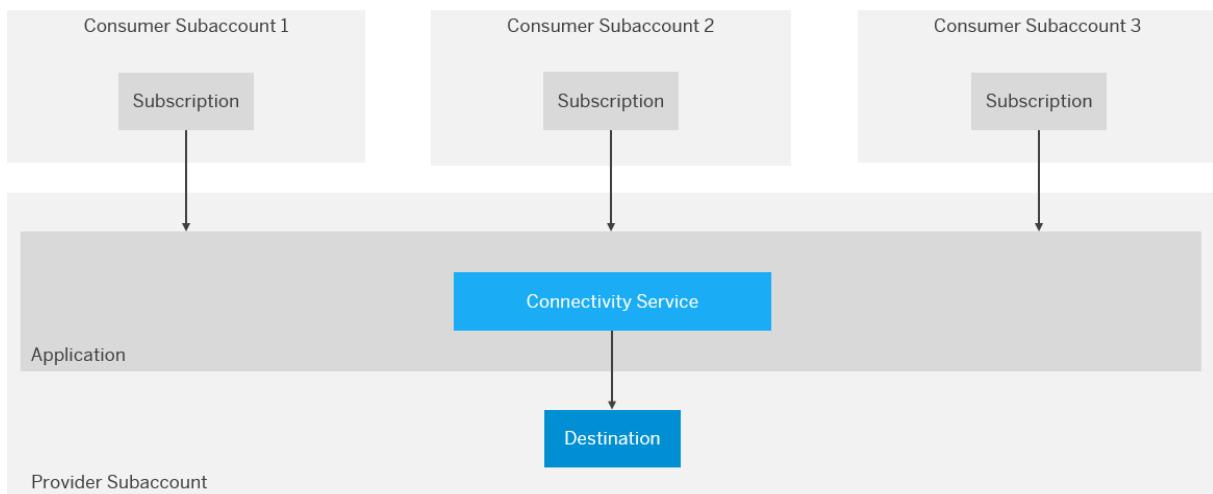
Level	Lookup
Subaccount level	Looked up on subaccount level, no matter which destination service instance is used.
Service instance level	Lookup via particular service instance (in provider or subscriber subaccount associated with this service instance).

When the application accesses the destination at runtime, the Connectivity service does the following:

- For a destination associated with a **provider** subaccount:
 1. Checks if the destination is available on the *service instance* level. If there is no destination found, it
 2. Searches the destination on *subaccount* level.
- For a destination associated with a **subscriber** subaccount:
 1. Checks if the destination is available on the *subscription* level. If there is no destination found, it
 2. Searches the destination on *subaccount* level.

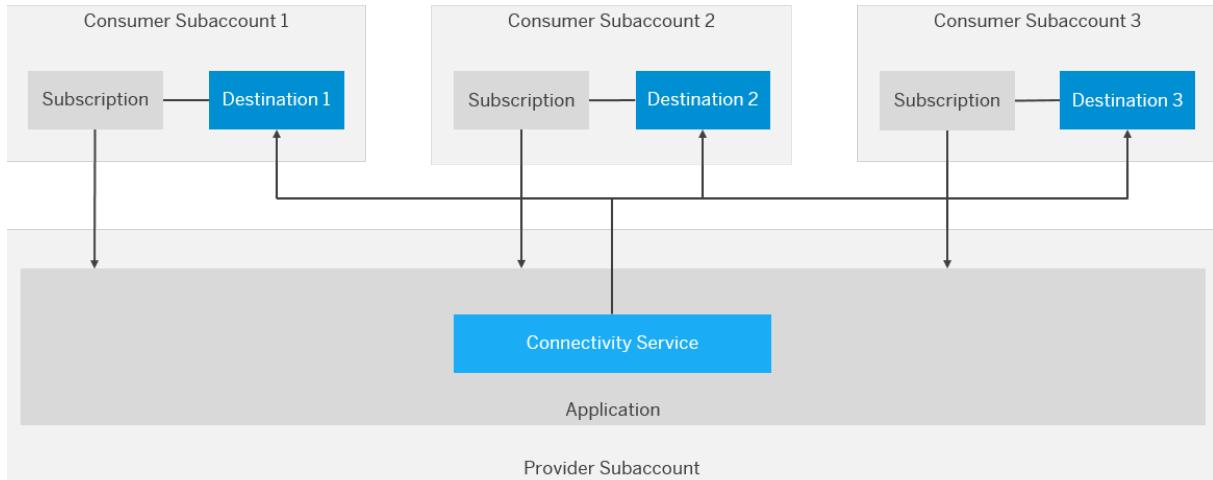
Back to [Top \[page 157\]](#)

Provider-Specific Destination



[Back to Top \[page 157\]](#)

Subscriber-Specific Destination



[Back to Top \[page 157\]](#)

Related Information

[Developing Multitenant Applications in the Cloud Foundry Environment](#)

1.1.3.6 Create and Bind a Connectivity Service Instance

To use the Connectivity service in your application, you need an instance of the service.

Prerequisites

When using service plan "lite", quota management is no longer required for this service. From any subaccount you can consume the service using service instances without restrictions on the instance count.

Previously, access to service plan "lite" has been granted via entitlement and quota management of the application runtime. It has now become an integral service offering of SAP BTP to simplify its usage. See also [Entitlements and Quotas](#).

Procedure

You have two options for creating a service instance – using the CLI or using the SAP BTP cockpit:

- [Create and Bind a Service Instance from the CLI \[page 160\]](#)
 - [Example \[page 160\]](#)
 - [Result \[page 162\]](#)
- [Create and Bind a Service Instance from the Cockpit \[page 161\]](#)
 - [Result \[page 162\]](#)

Create and Bind a Service Instance from the CLI

Use the following CLI commands to create a service instance and bind it to an application:

1. `cf marketplace`
2. `cf create-service connectivity <service-plan> <service-name>`
3. `cf bind-service <app-name> <service-name>`

Back to [Procedure \[page 160\]](#)

Example

To bind an instance of the Connectivity service "lite" plan to application "myapp", use following commands on the Cloud Foundry command line:

```
cf create-service connectivity lite myinstance
```

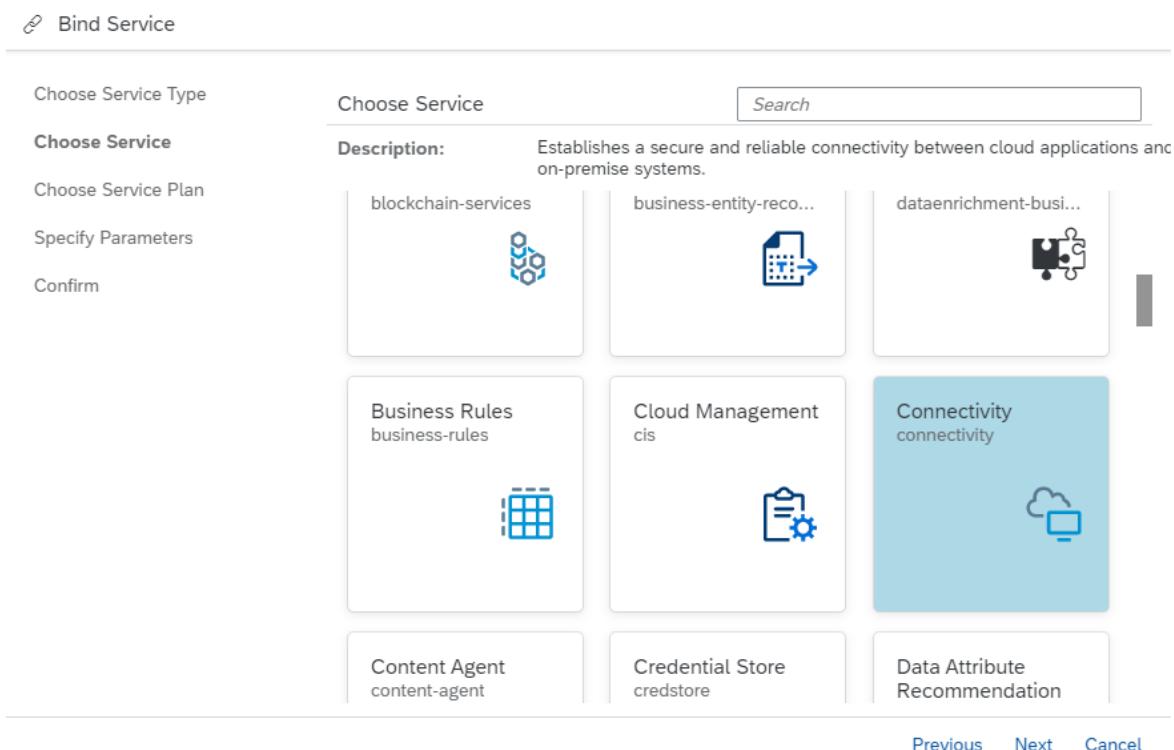
```
cf bind-service myapp myinstance
```

Back to [Procedure \[page 160\]](#)

Create and Bind a Service Instance from the Cockpit

Assuming that you have already deployed your application to the platform, follow these steps to create a service instance and bind it to an application:

1. In the SAP BTP Cockpit, navigate to your application.
2. From the left navigation menu, choose [Service Bindings](#).
3. Choose the [Bind Service](#) button.
4. The [Bind Service](#) wizard appears.
5. Select the [Service from the catalog](#) radio button, then choose [Next](#).
6. From the list of available services, select [Connectivity](#), then choose [Next](#).



7. On the next page of the wizard, select the [Create new instance](#) radio button.
8. Leave [`<Plan>`](#) `lite` selected and choose [Next](#).
9. The next page is used for specifying user-provided parameters in JSON format. If you do not want to do that, skip this step by choosing [Next](#).
10. In the [`<Instance Name>`](#) textbox, enter an unique name for your service instance.
11. Choose [Finish](#).

[Back to Procedure \[page 160\]](#)

Result

When the binding is created, the application gets the corresponding connectivity credentials in its environment variables:

↳ Sample Code

```
"VCAP_SERVICES": {
  "connectivity": [
    {
      "credentials": {
        "onpremise_proxy_host": "10.0.85.1",
        "onpremise_proxy_port": "20003",
        "onpremise_proxy_http_port": "20003",
        "clientid": "sb-connectivity-app",
        "clientsecret": "KXqObiN6d9gLA4cS2rOVAahPCX0=",
        "token_service_url": "<token_service_url>",
      },
      "label": "connectivity",
      "name": "conn-lite",
      "plan": "default",
      "provider": null,
      "syslog_drain_url": null,
      "tags": [
        "connectivity",
        "conn",
        "connsvc"
      ],
      "volume_mounts": []
    }
  ],
}
```

i Note

"onpremise_proxy_http_port" replaces the deprecated variable "onpremise_proxy_port", which will be removed soon. Same goes for "token_service_url", which replaces "url".

[Back to Procedure \[page 160\]](#)

1.1.3.7 Create and Bind a Destination Service Instance

To use the Destination service in your application, you need an instance of the service.

Concept

To consume the Destination service, you must provide the appropriate credentials through a service instance and a service binding/service key. The Destination service is publicly visible and cross-consumable from several environments and provides the service plan *lite* to all those environments. Provisioning a service instance and service key is done in the standard way for the respective environment, see:

- Cloud Foundry: [Using Services in the Cloud Foundry Environment](#)
- Kyma: [Using Services in the Kyma Environment](#)
- Kubernetes: [Consuming SAP BTP Services in Kubernetes with SAP Service Manager Broker Proxy \(Service Catalog\)](#)
- Other environments: [Consuming Services in Other Environments Using the SAP Service Manager Instances](#)

In all environments, the Destination service lets you provide a configuration JSON during instance creation or update.

Using a Configuration JSON

You can pass a configuration JSON during instance creation or update to modify some of the default settings of the instance and/or provide some content to be created during the operation.

The detailed structure of the configuration JSON is described in [Use a Config.JSON to Create or Update a Destination Service Instance \[page 164\]](#).

Troubleshooting

If you get this failure message:

```
Failed to create all provided configurations. Will delete all on instance level,  
for configurations on subaccount level you can set policies to handle  
duplicates: "existing_destinations_policy" with value "update|fail|ignore" and  
"existing_certificates_policy" with value "fail|ignore".
```

the root cause may be:

- A provided destination or certificate with the same name already exists in this subaccount. Solution: set policies on subaccount level to update or ignore in case of conflicts.

```
"existing_destinations_policy": "update|fail|ignore"  
"existing_certificates_policy": "fail|ignore"
```

- A client input error occurred. Solution: check the input, apply correction and try again.
- An internal server error occurred. In this case, please try again later or report a support incident.

1.1.3.7.1 Use a Config.JSON to Create or Update a Destination Service Instance

Configure specific parameters in a `config.json` file to create or update a Destination service instance.

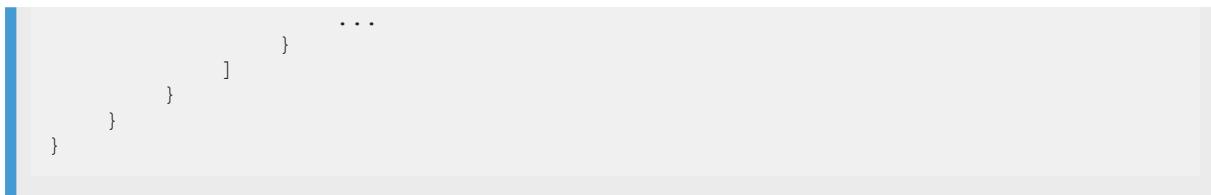
When creating or updating a Destination service instance, you can configure the following settings, which are part of the `config.json` input file (see [Open Service Broker API](#)), both via SAP BTP cockpit or Cloud Foundry command line interface (CLI):

Parameter	Value	Description
<code>init_data</code>	JSON	The data (destinations, certificates) to initialise or update the service instance with. The data can be stored on both <i>service instance</i> data and <i>subaccount</i> data.
<code>HTML5Runtime_enabled</code>	Boolean	Indicates whether the SAP BTP HTML5 runtime should be enabled to work with the service instance on behalf of the HTML5 applications associated with it during deployment.

Find the `config.json` structure below:

↳ Sample Code

```
{  
    "HTML5Runtime_enabled" : "true",  
    "init_data" : {  
        "subaccount" : {  
            "existing_destinations_policy": "update|fail|ignore",  
            "existing_certificates_policy": "fail|ignore",  
            "destinations" : [  
                {  
                    ...  
                }  
            ],  
            "certificates" : [  
                {  
                    ...  
                }  
            ]  
        },  
        "instance" : {  
            "existing_destinations_policy": "update|fail|ignore",  
            "existing_certificates_policy": "fail|ignore",  
            "destinations" : [  
                {  
                    ...  
                }  
            ],  
            "certificates" : [  
                {  
                    ...  
                }  
            ]  
        }  
    }  
}
```



1.1.4 Developing Applications

Consume the Connectivity service and the Destination service from an application in the Cloud Foundry environment.

Task	Description
Consuming the Connectivity Service [page 165]	Connect your Cloud Foundry application to an on-premise system.
Consuming the Destination Service [page 191]	Retrieve and store externalized technical information about the destination that is required to consume a target remote service from your application.
Invoking ABAP Function Modules via RFC [page 217]	Call a remote-enabled function module (RFM) in an on-premise or cloud ABAP server from your Cloud Foundry application, using the RFC protocol.

1.1.4.1 Consuming the Connectivity Service

Connect your Cloud Foundry application to an on-premise system via HTTP.

i Note

To use the Connectivity service with a protocol other than HTTP, see

- [Invoking ABAP Function Modules via RFC \[page 217\]](#)
- [Using the TCP Protocol for Cloud Applications \[page 182\]](#)

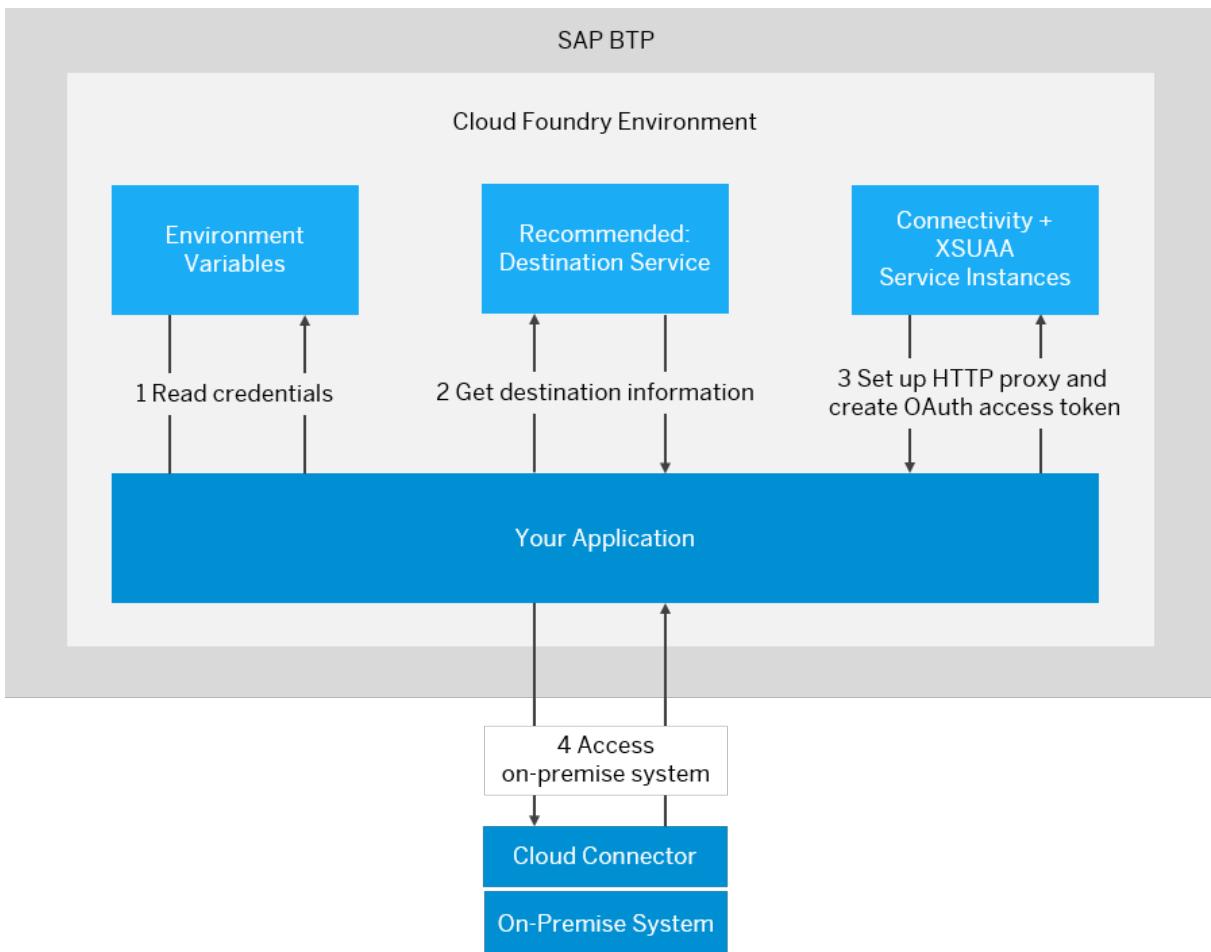
Tasks

Task Type	Task
 	Overview [page 166] Prerequisites [page 167]
Operator and/or Developer	
	Basic Steps [page 168] <ol style="list-style-type: none">1. Read Credentials from the Environment Variables [page 169]2. Provide the Destination Information [page 169]3. Set up the HTTP Proxy for On-Premise Connectivity [page 170]
Developer	
 	Additional Steps [page 172] <ul style="list-style-type: none">• Authentication against the On-Premise System [page 172]• Specify a Cloud Connector Location ID [page 172]• Multitenancy in the Connectivity Service [page 173]
Operator and/or Developer	

Overview



Using the Connectivity service, you can connect your Cloud Foundry application to an on-premise system through the Cloud Connector. To achieve this, you must provide the required information about the target system (destination), and set up an HTTP proxy that lets your application access the on-premise system.



[Back to Tasks \[page 165\]](#)

Prerequisites



- You must be a *Global Account* member to connect through the Connectivity service with the Cloud Connector. See [Add Members to Your Global Account](#).

Also *Security Administrators* (which must be either *Global Account* members or *Cloud Foundry Organization/Space* members) can do it. See [Managing Security Administrators in Your Subaccount \[Feature Set A\]](#).

i Note

To connect a Cloud Connector to your subaccount, you must currently be a **Security Administrator**.

- You have installed and configured a Cloud Connector in your on-premise landscape for the scenario you want to use. See [Installation \[page 282\]](#) and [Configuration \[page 321\]](#).

- You have deployed an application in a landscape of the Cloud Foundry environment that complies with the [Business Application Pattern](#).
- Your application is secured as described in .
- The Connectivity service is a regular service in the Cloud Foundry environment. Therefore, to consume the Connectivity service from an application, you must create a service instance and bind it to the application. See [Create and Bind a Connectivity Service Instance \[page 160\]](#).
- To get the required authorization for making on-premise calls through the connected Cloud Connector, the application must be bound to an instance of the xsuaa service using the service plan 'application'. The xsuaa service instance acts as an OAuth 2.0 client and grants user access to the bound application. Make sure you set the `xsappname` property to the name of the application when creating the instance. Find a detailed guide for this procedure in section 3. *Creation of the Authorization & Trust Management Instance (aka XSUAA)* of the SCN blog [How to use SAP BTP Connectivity and Cloud Connector in the Cloud Foundry environment](#).

i Note

Currently, the only supported protocol for connecting the Cloud Foundry environment to an on-premise system is HTTP. HTTPS is not needed, since the tunnel used by the Cloud Connector is TLS-encrypted.

⚠ Caution

There is a limit of **8192** bytes for the size of the HTTP lines (for example, request line or header) that you send via the Connectivity service. If this limit is exceeded, you receive an HTTP error of type 4xx. This issue is usually caused by the size of the `path + query` string of the request.

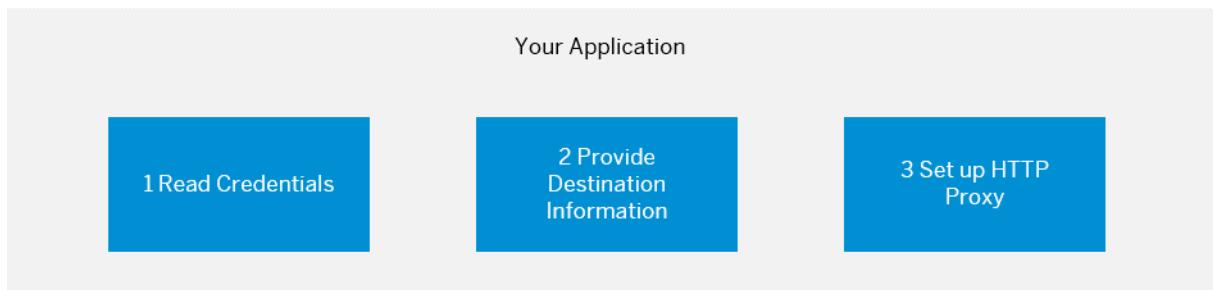
Back to [Tasks \[page 165\]](#)

Basic Steps



To consume the Connectivity service from your Cloud Foundry application, perform the following basic steps:

1. [Read Credentials from the Environment Variables \[page 169\]](#)
2. [Provide the Destination Information \[page 169\]](#)
3. [Set up the HTTP Proxy for On-Premise Connectivity \[page 170\]](#)



[Back to Tasks \[page 165\]](#)

Read Credentials from the Environment Variables



Consuming the Connectivity service requires credentials from the xsuaa and Connectivity service instances which are bound to the application. By binding the application to service instances of the xsuaa and Connectivity service as described in the prerequisites, these credentials become part of the environment variables of the application. You can access them as follows:

Sample Code

```
JSONObject jsonObj = new JSONObject(System.getenv("VCAP_SERVICES"));
JSONArray jsonArr = jsonObj.getJSONArray("<service name, not the instance
name>");
JSONObject credentials =
jsonArr.getJSONObject(0).getJSONObject("credentials");
```

i Note

If you have multiple instances of the same service bound to the application, you must perform additional filtering to extract the correct credential from `jsonArr`. You must go through the elements of `jsonArr` and find the one matching the correct instance name.

This code stores a JSON object in the `credentials` variable. Additional parsing is required to extract the value for a specific key.

i Note

We refer to the result of the above code block as `connectivityCredentials`, when called for `connectivity`, and `xsuaaCredentials` for `xsuaa`.

[Back to Tasks \[page 165\]](#)

Provide the Destination Information



To consume the Connectivity service, you must provide some information about your on-premise system and the system mappings for it in the Cloud Connector. You require the following:

- The endpoint in the Cloud Connector (virtual host and virtual port) and accessible URL paths on it (destinations). See [Configure Access Control \(HTTP\) \[page 380\]](#).
- The required authentication type for the on-premise system. See [HTTP Destinations \[page 64\]](#).
- Depending on the authentication type, you may need a username and password for accessing the on-premise system. For more details, see [Client Authentication Types for HTTP Destinations \[page 78\]](#).
- (Optional) You can use a *location Id*. For more details, see section [Specify a Cloud Connector Location ID \[page 172\]](#).

We recommend that you use the *Destination service* (see [Consuming the Destination Service \[page 191\]](#)) to procure this information. However, using the Destination service is optional. You can also provide (look up) this information in another appropriate way.

Back to [Tasks \[page 165\]](#)

Set up the HTTP Proxy for On-Premise Connectivity



Proxy Setup

The Connectivity service provides a standard HTTP proxy for on-premise connectivity that is accessible by any application. Proxy host and port are available as the environment variables `<onpremise_proxy_host>` and `<onpremise_proxy_http_port>`. You can set up the on-premise HTTP proxy like this:

« Sample Code

```
// get value of "onpremise_proxy_host" and "onpremise_proxy_http_port" from
// the environment variables
// and create on-premise HTTP proxy
String connProxyHost =
connectivityCredentials.getString("onpremise_proxy_host");
int connProxyPort =
Integer.parseInt(credentials.getString("onpremise_proxy_http_port"));
Proxy proxy = new Proxy(Proxy.Type.HTTP, new InetSocketAddress(connProxyHost,
connProxyPort));

// create URL to the remote endpoint you like to call:
// virtualhost:1234 is defined as an endpoint in the Cloud Connector, as
// described in the Required Information section
URL url = new URL("http://virtualhost:1234");

// create the connection object to the endpoint using the proxy
// this does not open a connection but only creates a connection object,
// which can be modified later, before actually connecting
urlConnection = (HttpURLConnection) url.openConnection(proxy);
```

i Note

"onpremise_proxy_http_port" replaces the deprecated variable "onpremise_proxy_port", which will be removed soon.

Authorization

To make calls to on-premise services configured in the Cloud Connector through the HTTP proxy, you must authorize at the HTTP proxy. For this, the *OAuth Client Credentials* flow is used: applications must create an OAuth access token using the parameters `clientid` and `clientsecret` that are provided by the Connectivity service in the environment, as shown in the example code below. When the application has retrieved the access token, it must pass the token to the connectivity proxy using the `Proxy-Authorization` header.

The sample code below uses the following Maven artifacts:

- `org.springframework.security:spring-security-oauth2-core`
- `org.springframework.security:spring-security-oauth2-client`

↳ Sample Code

```
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.oauth2.client.ClientCredentialsOAuth2AuthorizedClientProvider;
import org.springframework.security.oauth2.client.OAuth2AuthorizationContext;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProvider;
import org.springframework.security.oauth2.client.registration.ClientRegistration;
import org.springframework.security.oauth2.core.AuthorizationGrantType;
import org.springframework.security.oauth2.core.OAuth2AccessToken;

...
// get value of "clientid" and "clientsecret" from the environment variables
String clientid = connectivityCredentials.getString("clientid");
String clientsecret = connectivityCredentials.getString("clientsecret");

// get the URL to xsuaa from the environment variables
String xsuaaUrl = xsuaaCredentials.getString("token_service_url");

// make request to UAA to retrieve access token
ClientRegistration clientRegistration =
    ClientRegistration.withRegistrationId("some-id").
        authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS).
        clientId(clientid).
        clientSecret(clientsecret).
        authorizationUri(xsuaaUrl + "/oauth/authorize").
        tokenUri(xsuaaUrl + "/oauth/token").
        build();

OAuth2AuthorizationContext xsuaaContext =
    OAuth2AuthorizationContext.withClientRegistration(clientRegistration).
        principal(new AbstractAuthenticationToken(null) {
            @Override
            public Object getPrincipal() {
                return null;
            }
        });
// use xsuaaContext to make requests to on-premise services
```

```

        }

    @Override
    public Object getCredentials() {
        return null;
    }

    @Override
    public String getName() {
        return "dummyPrincipalName";      // There is no principal in the
        client credentials authorization grant but a non-empty name is still required.
    }
}).build();

OAuth2AuthorizedClientProvider clientCredentialsAccessTokenProvider = new
ClientCredentialsOAuth2AuthorizedClientProvider();
OAuth2AccessToken token =
clientCredentialsAccessTokenProvider.authorize(xsuaaContext).getAccessToken();

// set access token as Proxy-Authorization header in the URL connection
urlConnection.setRequestProperty("Proxy-Authorization",
token.getTokenType().getValue() + " " + token.getValue());

```

i Note

`xsuaaCredentials.getString("token_service_url")` replaces the deprecated property `xsuaaCredentials.getString("url")`, which will be removed soon.

Back to [Tasks \[page 165\]](#)

Authentication against the On-Premise System



Depending on the required authentication type for the desired on-premise resource, you may have to set an additional header in your request. This header provides the required information for the authentication process against the on-premise resource. See [Authentication to the On-Premise System \[page 174\]](#).

Back to [Tasks \[page 165\]](#)

Specify a Cloud Connector Location ID



Note

This is an advanced option when using more than one Cloud Connector for a subaccount. For more information how to set the `location_ID` in the Cloud Connector, see [Managing Subaccounts \[page 338\]](#), step 4 in section *Subaccount Dashboard*.

As of Cloud Connector 2.9.0, you can connect multiple Cloud Connectors to a subaccount if their `location_ID` is different. Using the header `SAP-Connectivity-SCC-Location_ID` you can specify the Cloud Connector over which the connection should be opened. If this header is not specified, the connection is opened to the Cloud Connector that is connected without any `location_ID`. This also applies for all Cloud Connector versions prior to 2.9.0. For example:

Sample Code

```
// Optionally, if configured, add the SCC location ID.  
urlConnection.setRequestProperty("SAP-Connectivity-SCC-Location_ID",  
"orlando");
```

Back to [Tasks \[page 165\]](#)

Multitenancy in the Connectivity Service



To consume the Connectivity service from an SaaS application in a multitenant way, the only requirement is that the SaaS application returns the Connectivity service as a dependent service in its dependencies list.

For more information about the subscription flow, see [Develop the Multitenant Business Application](#).

Back to [Tasks \[page 165\]](#)

Related Information

[Cloud Connector \[page 276\]](#)

[Set Up an Application as a Sample Backend System](#)

[Create and Bind a Connectivity Service Instance \[page 160\]](#)

[Authentication to the On-Premise System \[page 174\]](#)

[Consuming the Destination Service \[page 191\]](#)

[What Is the SAP Authorization and Trust Management Service?](#)

[Multitarget Applications in the Cloud Foundry Environment](#)

[Security](#)

[Invoking ABAP Function Modules via RFC \[page 217\]](#)

[Using the TCP Protocol for Cloud Applications \[page 182\]](#)

1.1.4.1.1 Authentication to the On-Premise System

Provide authentication information for the authentication type you use.

You can use the Connectivity service in different authentication scenarios:

- **No authentication** to the on-premise system
- **Principal propagation** (user propagation) to the on-premise system
- **Basic authentication** to the on-premise system

Procedure

For each authentication type, you must provide specific information in the request to the virtual host:

- [The SAP-Connectivity-Authentication Header \[page 174\]](#)
- [Authentication Types \[page 175\]](#)

The SAP-Connectivity-Authentication Header

i Note

For the principal propagation scenario, the SAP-Connectivity-Authentication header is only required if you do not use the user exchange token flow, see [Configure Principal Propagation via User Exchange Token \[page 176\]](#).

Applications must propagate the user JWT token (`userToken`) using the SAP-Connectivity-Authentication header. This is required for the Connectivity service to open a tunnel to the subaccount for which a configuration is made in the Cloud Connector. The following example shows you how to do this using the *Spring* framework:

Sample Code

```
Authentication auth = SecurityContextHolder.getContext().getAuthentication();
if (auth == null) {
    throw new UserInfoException("User not authenticated");
}
OAuth2AuthenticationDetails details = (OAuth2AuthenticationDetails)
auth.getDetails();
String userToken = details.getTokenValue();
urlConnection.setRequestProperty("SAP-Connectivity-Authentication", "Bearer "
+ userToken);
```

Back to [Procedure \[page 174\]](#)

Authentication Types

The required setup for each of the authentication type is as follows:

No Authentication

If the on-premise system does not need to identify the user, you should use this authentication type. It requires no additional information to be passed with the request.

Principal Propagation

When you open the application router to access your cloud application, you are prompted to log in. Doing so means that the cloud application now knows your identity. Principal propagation forwards this identity via the Cloud Connector to the on-premise system. This information is then used to grant access without additional input from the user. To achieve this, you do not need to send any additional information from your application, but you must set up the Cloud Connector for principal propagation. See [Configuring Principal Propagation \[page 350\]](#).

Basic Authentication

If the on-premise system requires username and password to grant access, the cloud application must provide these data using the *Authorization* header. The following example shows how to do this:

Sample Code

```
// Basic authentication to backend system
String credentials = MessageFormat.format("{0}:{1}", backendUser,
backendPassword);
byte[] encodedBytes = Base64.encodeBase64(credentials.getBytes());
urlConnection.setRequestProperty("Authorization", "Basic " + new
String(encodedBytes));
```

Back to [Procedure \[page 174\]](#)

Related Information

[Configure Principal Propagation via User Exchange Token \[page 176\]](#)

[Configure Principal Propagation via IAS Token \[page 180\]](#)

[Configure Principal Propagation via OIDC Token \[page 181\]](#)

1.1.4.1.1.1 Configure Principal Propagation via User Exchange Token

Configure a user exchange token for principal propagation (user propagation) from your Cloud Foundry application to an on-premise system.

Tasks

Task Type	Task
 	Scenario [page 176]
Operator and/or Developer	
	Solutions [page 176]
Developer	Generate the Authentication Token [page 177]

Scenario



For a Cloud Foundry application that uses the Connectivity service, you want the currently logged-in user to be propagated to an on-premise system. For more information, see [Principal Propagation \[page 122\]](#).

Back to [Tasks \[page 176\]](#)

Solutions



You have two options to implement user propagation:

1. **Recommended:** The application sends one header containing the user exchange token to the Connectivity proxy:

```
Header: "Proxy-Authorization" : "Bearer <userExchangeAccessToken>"
```

This option is described in detail in [Generate the Authentication Token \[page 177\]](#) below.

2. The application sends two headers to the Connectivity proxy:

```
Header: "SAP-Connectivity-Authentication" : "Bearer <userToken>"  
Header: "Proxy-Authorization" : "Bearer <accessToken>"
```

For more information about this solution, see also [Consuming the Connectivity Service \[page 165\]](#).

i Note

This solution is supported to guarantee backward compatibility.

Back to [Tasks \[page 176\]](#)

Generate the Authentication Token



To propagate a user to an on-premise system, you must call the Connectivity proxy using a special JWT (JSON Web token). This token is obtained by exchanging a valid user token following the [OAuth2 JWT Bearer grant type](#).

- [Example: Obtaining a User Token Following the JWT Bearer Grant Type \[page 177\]](#)
- [Example: Calling the Connectivity Proxy with the Exchanged User Token \[page 179\]](#)

Example: Obtaining a User Token Following the [JWT Bearer Grant Type](#)

Request:

Sample Code

```
POST https://<host: the value of 'url' from Connectivity service credentials  
in VCAP_SERVICES>/oauth/token  
Accept: application/json  
Content-Type: application/x-www-form-urlencoded  
client_id=<connectivity_service_client_id>  
client_secret=<connectivity_service_client_secret>  
grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer  
token_format=jwt  
response_type=token  
assertion=<logged-in-user-JWT>
```

Response:

↳ Sample Code

```
{  
    "access_token" : "<new-user-JWT>",  
    "token_type" : "bearer",  
    "expires_in" : 43199,  
    "scope" : "<token-scopes>",  
    "jti" : "7cc917b8bf6347a2aa18d7ac8f38a1c2"  
}
```

The JWT in `access_token`, also referred to as *user exchange token*, now contains the user details. It is used to consume the Connectivity service.

In case of a Java application, you can use a library that implements the user exchange OAuth flow. Here is an example of how the `userExchangeAccessToken` can be obtained using the [XSUAA Token Client and Token Flow API](#) :

⚠ Caution

Make sure you get the latest API version.

A sample Maven dependency declaration:

```
<dependency>  
    <groupId>com.sap.cloud.security.xsuaa</groupId>  
    <artifactId>token-client</artifactId>  
    <version><latest version (e.g.: 2.7.7)></version>  
</dependency>
```

→ Remember

The XSUAA Token Client library works with multiple HTTP client libraries. Make sure you have one as Maven dependency.

The following sample uses the Apache REST client:

↳ Sample Code

```
// service instance specific OAuth client credentials shall be used  
String connectivityServiceClientId = credentials.getString(CLIENT_ID);  
String connectivityServiceClientSecret = credentials.getString(CLIENT_SECRET);  
// get the URL to xsuaa from the environment variables  
URI xsuaaUri = new URI(xsuaaCredentials.getString("token_service_url"));  
  
// use the XSUAA client library to ease the implementation of the user token  
// exchange flow  
XsuaaTokenFlows tokenFlows = new XsuaaTokenFlows(new  
DefaultOAuth2TokenService(), new XsuaaDefaultEndpoints(xsuaaUri.toString()),  
new ClientCredentials(connectivityServiceClientId,  
connectivityServiceClientSecret));
```

```
String userExchangeAcessToken =  
tokenFlows.userTokenFlow().token(<jwtToken_to_exchange>).execute().getAccessToken();
```

For more information about caching, see also [XSUAA Token Client and Token Flow API - Cache](#).

i Note

`xsuaaCredentials.getString("token_service_url")` replaces the deprecated property `xsuaaCredentials.getString("url")`, which will be removed soon.

See also: [XSUAA Token Client and Token Flow API](#).

After obtaining the `userExchangeAcessToken`, you can use it to consume the Connectivity service.

Back to [Generate the Authentication Token \[page 177\]](#)

Example: Calling the Connectivity Proxy with the Exchanged User Token

As a prerequisite to this step, you must configure the Connectivity proxy to be used by your client, see [Set up the HTTP Proxy for On-Premise Connectivity \[page 170\]](#).

Once the application has retrieved the user exchange token, it must pass the token to the Connectivity proxy via the `Proxy-Authorization` header. In this example, we use `URLConnection` as a client.

↳ Sample Code

```
// set user exchange token as Proxy-Authorization header in the URL connection  
urlConnection.setRequestProperty("Proxy-Authorization", "Bearer " +  
userExchangeAcessToken);
```

i Note

Alternatively, you can use the *basic authentication* scheme:

↳ Sample Code

```
// Basic authentication to Connectivity Proxy  
String credentials = MessageFormat.format("{0}:{1}", userExchangeAcessToken,  
"");  
byte[] encodedBytes = Base64.encodeBase64(credentials.getBytes());  
urlConnection.setRequestProperty("Proxy-Authorization", "Basic " + new  
String(encodedBytes));
```

Back to [Generate the Authentication Token \[page 177\]](#)

Back to [Tasks \[page 176\]](#)

1.1.4.1.1.2 Configure Principal Propagation via IAS Token

Configure an Identity Authentication service (IAS) token for principal propagation (user propagation) from your Cloud Foundry application to an on-premise system.

Scenario

For a Cloud Foundry application that uses the Connectivity service, you want the currently logged-in user to be propagated via the Cloud Connector to an on-premise system. This user is represented in the cloud application by an IAS token.

For more information, see [Principal Propagation \[page 122\]](#) and [Getting Started with the Identity Service of SAP BTP](#).

Prerequisites

- Cloud Connector 2.13 (or newer) must be used.
- The Cloud Connector must be connected to a subaccount that is configured with the IAS tenant issuing the tokens.

For more information, see [Establish Trust and Federation Between UAA and Identity Authentication](#).

Solution

The application sends two headers to the Connectivity proxy:

```
Header: "SAP-Connectivity-Authentication" : "Bearer <IAS-Token>"  
Header: "Proxy-Authorization" : "Bearer <accessToken>"
```

More Information

[Identity Authentication](#)

1.1.4.1.1.3 Configure Principal Propagation via OIDC Token

Configure an OpenID-Connect (OIDC) token for principal propagation (user propagation) from your Cloud Foundry application to an on-premise system.

Tasks

Task Type	Task
 	Scenario [page 181]
Operator and/or Developer	
	Solution [page 181]
Developer	

Scenario



For a Cloud Foundry application that uses the Connectivity service, you want the currently logged-in user to be propagated via the Cloud Connector to an on-premise system. This user is represented in the cloud application by an OIDC token. For more information, see [Principal Propagation \[page 122\]](#) and the [OIDC specification](#).

Back to [Tasks \[page 181\]](#)

Solution



As of version 2.13.0, the Cloud Connector supports principal propagation for OIDC tokens. If on the cloud application side the user is represented by an OIDC token, the application can send the user principal to the Connectivity service (thus reaching the Cloud Connector), using the `SAP-Connectivity-Authentication` HTTP header.

The Cloud Connector validates the token, extracts the available user data, and enables further processing through a configured subject pattern for the resulting short-lived X.509 client certificate.

By default, the user principal is identified by one of the following JWT (JSON web token) attributes:

- user_name
- email
- mail
- user_uuid
- sub

This list specifies the priority (in descending order from top to bottom) for the default value of \${name} in the subject pattern of the X.509 client certificate. If a token has more than one of the above claims, the value of \${name} is extracted from the claim with the highest priority by default.

For the example token below, the default value of \${name} is test@user.com:

↳ Sample Code

```
{  
  "aud": "111111111111-2222-3333-444444444444",  
  "iss": "https://issuer.com",  
  "exp": 2091269073,  
  "iat": 1601901108,  
  "jti": "111222333444555666777888999000",  
  "sub": "test",  
  "email": "test@users.com"  
}
```

The Cloud Connector administrator can control the exact value to be used as user principal for the *subject CN* of the X.509 client certificate by configuring a subject pattern. For more information, see [Configure a Subject Pattern for Principal Propagation \[page 369\]](#).

Back to [Tasks \[page 181\]](#)

1.1.4.1.2 Using the TCP Protocol for Cloud Applications

Access on-premise systems from a Cloud Foundry application via TCP-based protocols, using a SOCKS5 Proxy.

Content

[Concept \[page 183\]](#)

[Restrictions \[page 184\]](#)

[Example \[page 184\]](#)

Concept

SAP BTP Connectivity provides a SOCKS5 proxy that you can use to access on-premise systems via TCP-based protocols. SOCKS5 is the industry standard for proxying TCP-based traffic (for more information, see IETF [RFC 1928](#)).

The SOCKS5 proxy host and port are accessible through the environment variables, which are generated after binding an application to a Connectivity service instance. For more information, see [Consuming the Connectivity Service \[page 165\]](#).

You can access the host under `onpremise_proxy_host`, and the port through `onpremise_socks5_proxy_port`, obtained from the Connectivity service instance.

Authentication to the SOCKS5 proxy is mandatory. It involves the usage of a JWT (JSON Web token) access token (for more information, see IETF [RFC 7519](#)). The JWT can be retrieved through the `client_id` and `client_secret`, obtained from the Connectivity service instance. For more information, see [Set up the HTTP Proxy for On-Premise Connectivity \[page 170\]](#), section **Authorization**.

The value of the SOCKS5 protocol authentication method is defined as **0x80** (defined as X'80' in IETF, refer to the official specification [SOCKS Protocol Version 5](#)). This value should be sent as part of the authentication method's negotiation request (known as *Initial Request* in SOCKS5). The server then confirms with a response containing its decimal representation (either **128** or **-128**, depending on the client implementation).

After a successful SOCKS5 *Initial Request*, the authentication procedure follows the standard SOCKS5 authentication sub-procedure, that is SOCKS5 *Authentication Request*. The request bytes, in sequence, should look like this:

Bytes	Description
1 byte	Authentication method version - currently 1
4 bytes	Length of the JWT
X bytes	X - The actual value of the JWT in its encoded form
1 byte	Length of the Cloud Connector location ID (0 if no Cloud Connector location ID is used)
Y bytes	Optional. Y - The value of the Cloud Connector location ID in base64-encoded form (if the the value of the location ID is not 0)

The Cloud Connector location ID identifies Cloud Connector instances that are deployed in various locations of a customer's premises and connected to the same subaccount. Since the location ID is an optional property, you should include it in the request only if it has already been configured in the Cloud Connector. For more information, see [Set up Connection Parameters and HTTPS Proxy \[page 325\]](#) (Step 4).

If not set in the Cloud Connector, the byte representing the length of the location ID in the *Authentication Request* should have the value **0**, without including any value for the Cloud Connector location ID (`sccLocationId`).

Back to [Content \[page 182\]](#)

Restrictions

- You cannot use the provided SOCKS5 proxy as general-purpose proxy.
- Proxying UDP traffic is not supported.

Back to [Content \[page 182\]](#)

Example

The following code snippet demonstrates an example based on the Apache Http Client library and Java code, which represents a way to replace the standard socket used in the Apache HTTP client with one that is responsible for authenticating with the Connectivity SOCKS5 proxy:

Sample Code

```
@Override  
public void connect(SocketAddress endpoint, int timeout) throws IOException {  
    super.connect(getProxyAddress(), timeout);  
  
    OutputStream outputStream = getOutputStream();  
  
    executeSOCKS5InitialRequest(outputStream); // 1. Negotiate authentication  
method, i.e. 0x80 (128)  
  
    executeSOCKS5AuthenticationRequest(outputStream); // 2. Negotiate  
authentication sub-version and send the JWT (and optionally the Cloud  
Connector Location ID)  
  
    executeSOCKS5ConnectRequest(outputStream, (InetSocketAddress)  
endpoint); // 3. Initiate connection to target on-premise backend system  
}
```

Sample Code

```
private byte[] createInitialSOCKS5Request() throws IOException {  
    ByteArrayOutputStream byteArraysStream = new ByteArrayOutputStream();  
    try {  
        byteArraysStream.write(SOCKS5_VERSION);  
        byteArraysStream.write(SOCKS5_AUTHENTICATION_METHODS_COUNT);  
        byteArraysStream.write(SOCKS5_JWT_AUTHENTICATION_METHOD);  
        return byteArraysStream.toByteArray();  
    } finally {  
        byteArraysStream.close();  
    }  
}
```

```

private void executeSOCKS5InitialRequest(OutputStream outputStream) throws
IOException {
    byte[] initialRequest = createInitialSOCKS5Request();
    outputStream.write(initialRequest);

    assertServerInitialResponse();
}

```

↳ Sample Code

```

private byte[] createJWTAuthenticationRequest() throws IOException {
    ByteArrayOutputStream byteArraysStream = new ByteArrayOutputStream();
    try {
        byteArraysStream.write(SOCKS5_JWT_AUTHENTICATION_METHOD_VERSION);

        byteArraysStream.write(ByteBuffer.allocate(4).putInt(jwtToken.getBytes().length).array());
        byteArraysStream.write(jwtToken.getBytes());
        byteArraysStream.write(ByteBuffer.allocate(1).put((byte)sccLocationId.length).array());
        byteArraysStream.write(sccLocationId.getBytes());
        return byteArraysStream.toByteArray();
    } finally {
        byteArraysStream.close();
    }
}

private void executeSOCKS5AuthenticationRequest(OutputStream outputStream)
throws IOException {
    byte[] authenticationRequest = createJWTAuthenticationRequest();
    outputStream.write(authenticationRequest);

    assertAuthenticationResponse();
}

```

In version 4.2.6 of the Apache HTTP client, the class responsible for connecting the socket is `DefaultClientConnectionOperator`. By extending the class and replacing the standard socket with the complete example code below, which implements a Java Socket, you can handle the SOCKS5 authentication with ID **0x80**. It is based on a JWT and supports the Cloud Connector location ID.

↳ Sample Code

```

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;
import java.net.SocketException;
import java.nio.ByteBuffer;

import java.util.Base64; // or any other library for base64 encoding
import org.json.JSONArray; // or any other library for JSON objects
import org.json.JSONObject; // or any other library for JSON objects
import org.json.JSONException; // or any other library for JSON objects

public class ConnectivitySocks5ProxySocket extends Socket {

```

```

private static final byte SOCKS5_VERSION = 0x05;
private static final byte SOCKS5_JWT_AUTHENTICATION_METHOD = (byte) 0x80;
private static final byte SOCKS5_JWT_AUTHENTICATION_METHOD_VERSION = 0x01;
private static final byte SOCKS5_COMMAND_CONNECT_BYTE = 0x01;
private static final byte SOCKS5_COMMAND_REQUEST_RESERVED_BYTE = 0x00;
private static final byte SOCKS5_COMMAND_ADDRESS_TYPE_IPv4_BYTE = 0x01;
private static final byte SOCKS5_COMMAND_ADDRESS_TYPE_DOMAIN_BYTE = 0x03;
private static final byte SOCKS5_AUTHENTICATION_METHODS_COUNT = 0x01;
private static final int SOCKS5_JWT_AUTHENTICATION_METHOD_UNSIGNED_VALUE
= 0x80 & 0xFF;
private static final byte SOCKS5_AUTHENTICATION_SUCCESS_BYTE = 0x00;

private static final String SOCKS5_PROXY_HOST_PROPERTY =
"onpremise_proxy_host";
private static final String SOCKS5_PROXY_PORT_PROPERTY =
"onpremise_socks5_proxy_port";

private final String jwtToken;
private final String sccLocationId;

public ConnectivitySocks5ProxySocket(String jwtToken, String
sccLocationId) {
    this.jwtToken = jwtToken;
    this.sccLocationId = sccLocationId != null ?
Base64.getEncoder().encodeToString(sccLocationId.getBytes()) : "";
}

protected InetSocketAddress getProxyAddress() {
    try {
        JSONObject credentials = extractEnvironmentCredentials();
        String proxyHost =
credentials.getString(SOCKS5_PROXY_HOST_PROPERTY);
        int proxyPort =
Integer.parseInt(credentials.getString(SOCKS5_PROXY_PORT_PROPERTY));
        return new InetSocketAddress(proxyHost, proxyPort);
    } catch (JSONException ex) {
        throw new IllegalStateException("Unable to extract the SOCKS5
proxy host and port", ex);
    }
}

private JSONObject extractEnvironmentCredentials() throws JSONException {
    JSONObject jsonObj = new JSONObject(System.getenv("VCAP_SERVICES"));
    JSONArray jsonArr = jsonObj.getJSONArray("connectivity");
    return jsonArr.getJSONObject(0).getJSONObject("credentials");
}

@Override
public void connect(SocketAddress endpoint, int timeout) throws
IOException {
    super.connect(getProxyAddress(), timeout);

    OutputStream outputStream = getOutputStream();
    executeSOCKS5InitialRequest(outputStream);

    executeSOCKS5AuthenticationRequest(outputStream);

    executeSOCKS5ConnectRequest(outputStream, (InetSocketAddress)
endpoint);
}

private void executeSOCKS5InitialRequest(OutputStream outputStream)
throws IOException {
    byte[] initialRequest = createInitialSOCKS5Request();
    outputStream.write(initialRequest);

    assertServerInitialResponse();
}

```

```

    }

    private byte[] createInitialSOCKS5Request() throws IOException {
        ByteArrayOutputStream byteArraysStream = new ByteArrayOutputStream();
        try {
            byteArraysStream.write(SOCKS5_VERSION);
            byteArraysStream.write(SOCKS5_AUTHENTICATION_METHODS_COUNT);
            byteArraysStream.write(SOCKS5_JWT_AUTHENTICATION_METHOD);
            return byteArraysStream.toByteArray();
        } finally {
            byteArraysStream.close();
        }
    }

    private void assertServerInitialResponse() throws IOException {
        InputStream inputStream = getInputStream();

        int versionByte = inputStream.read();
        if (SOCKS5_VERSION != versionByte) {
            throw new SocketException(String.format("Unsupported SOCKS
version - expected %s, but received %s", SOCKS5_VERSION, versionByte));
        }

        int authenticationMethodValue = inputStream.read();
        if (SOCKS5_JWT_AUTHENTICATION_METHOD_UNSIGNED_VALUE !=
authenticationMethodValue) {
            throw new SocketException(String.format("Unsupported
authentication method value - expected %s, but received %s",
                SOCKS5_JWT_AUTHENTICATION_METHOD_UNSIGNED_VALUE,
                authenticationMethodValue));
        }
    }

    private void executeSOCKS5AuthenticationRequest(OutputStream
outputStream) throws IOException {
        byte[] authenticationRequest = createJWTAuthenticationRequest();
        outputStream.write(authenticationRequest);

        assertAuthenticationResponse();
    }

    private byte[] createJWTAuthenticationRequest() throws IOException {
        ByteArrayOutputStream byteArraysStream = new ByteArrayOutputStream();
        try {
            byteArraysStream.write(SOCKS5_JWT_AUTHENTICATION_METHOD_VERSION);

byteArraysStream.write(ByteBuffer.allocate(4).putInt(jwtToken.getBytes().length).array());
            byteArraysStream.write(jwtToken.getBytes());
            byteArraysStream.write(ByteBuffer.allocate(1).put((byte)
sccLocationId.getBytes().length).array());
            byteArraysStream.write(sccLocationId.getBytes());
            return byteArraysStream.toByteArray();
        } finally {
            byteArraysStream.close();
        }
    }

    private void assertAuthenticationResponse() throws IOException {
        InputStream inputStream = getInputStream();

        int authenticationMethodVersion = inputStream.read();
        if (SOCKS5_JWT_AUTHENTICATION_METHOD_VERSION !=
authenticationMethodVersion) {
            throw new SocketException(String.format("Unsupported
authentication method version - expected %s, but received %s",
                SOCKS5_JWT_AUTHENTICATION_METHOD_VERSION,
                authenticationMethodVersion));
        }
    }
}

```

```

        }

        int authenticationStatus = inputStream.read();
        if (SOCKS5_AUTHENTICATION_SUCCESS_BYTE != authenticationStatus) {
            throw new SocketException("Authentication failed!");
        }
    }

    private void executeSOCKS5ConnectRequest(OutputStream outputStream,
    InetSocketAddress endpoint) throws IOException {
        byte[] commandRequest = createConnectCommandRequest(endpoint);
        outputStream.write(commandRequest);

        assertConnectCommandResponse();
    }

    private byte[] createConnectCommandRequest(InetSocketAddress endpoint)
throws IOException {
    String host = endpoint.getHostName();
    int port = endpoint.getPort();
    ByteArrayOutputStream byteArraysStream = new ByteArrayOutputStream();
    try {
        byteArraysStream.write(SOCKS5_VERSION);
        byteArraysStream.write(SOCKS5_COMMAND_CONNECT_BYTE);
        byteArraysStream.write(SOCKS5_COMMAND_REQUEST_RESERVED_BYTE);
        byte[] hostToIPv4 = parseHostToIPv4(host);
        if (hostToIPv4 != null) {
            byteArraysStream.write(SOCKS5_COMMAND_ADDRESS_TYPE_IPV4_BYTE);
            byteArraysStream.write(hostToIPv4);
        } else {

byteArraysStream.write(SOCKS5_COMMAND_ADDRESS_TYPE_DOMAIN_BYTE);
            byteArraysStream.write(ByteBuffer.allocate(1).put((byte)
host.getBytes().length).array());
            byteArraysStream.write(host.getBytes());
        }
        byteArraysStream.write(ByteBuffer.allocate(2).putShort((short)
port).array());
        return byteArraysStream.toByteArray();
    } finally {
        byteArraysStream.close();
    }
}

private void assertConnectCommandResponse() throws IOException {
    InputStream inputStream = getInputStream();

    int versionByte = inputStream.read();
    if (SOCKS5_VERSION != versionByte) {
        throw new SocketException(String.format("Unsupported SOCKS
version - expected %s, but received %s", SOCKS5_VERSION, versionByte));
    }

    int connectStatusByte = inputStream.read();
    assertConnectStatus(connectStatusByte);

    readRemainingCommandResponseBytes(inputStream);
}

private void assertConnectStatus(int commandConnectStatus) throws
IOException {
    if (commandConnectStatus == 0) {
        return;
    }

    String commandConnectStatusTranslation;
    switch (commandConnectStatus) {
        case 1:

```

```

        commandConnectStatusTranslation = "FAILURE";
        break;
    case 2:
        commandConnectStatusTranslation = "FORBIDDEN";
        break;
    case 3:
        commandConnectStatusTranslation = "NETWORK_UNREACHABLE";
        break;
    case 4:
        commandConnectStatusTranslation = "HOST_UNREACHABLE";
        break;
    case 5:
        commandConnectStatusTranslation = "CONNECTION_REFUSED";
        break;
    case 6:
        commandConnectStatusTranslation = "TTL_EXPIRED";
        break;
    case 7:
        commandConnectStatusTranslation = "COMMAND_UNSUPPORTED";
        break;
    case 8:
        commandConnectStatusTranslation = "ADDRESS_UNSUPPORTED";
        break;
    default:
        commandConnectStatusTranslation = "UNKNOWN";
        break;
    }
    throw new SocketException("SOCKS5 command failed with status: " +
commandConnectStatusTranslation);
}

private byte[] parseHostToIPv4(String hostName) {
    byte[] parsedHostName = null;
    String[] virtualHostOctets = hostName.split("\\.", -1);
    int octetsCount = virtualHostOctets.length;
    if (octetsCount == 4) {
        try {
            byte[] ipOctets = new byte[octetsCount];
            for (int i = 0; i < octetsCount; i++) {
                int currentOctet = Integer.parseInt(virtualHostOctets[i]);
                if ((currentOctet < 0) || (currentOctet > 255)) {
                    throw new
IllegalArgumentException(String.format("Provided octet %s is not in the range
of [0-255]", currentOctet));
                }
                ipOctets[i] = (byte) currentOctet;
            }
            parsedHostName = ipOctets;
        } catch (IllegalArgumentException ex) {
            return null;
        }
    }
    return parsedHostName;
}

private void readRemainingCommandResponseBytes(InputStream inputStream)
throws IOException {
    inputStream.read(); // skipping over SOCKS5 reserved byte
    int addressTypeByte = inputStream.read();
    if (SOCKS5_COMMAND_ADDRESS_TYPE_IPV4_BYTE == addressTypeByte) {
        for (int i = 0; i < 6; i++) {
            inputStream.read();
        }
    } else if (SOCKS5_COMMAND_ADDRESS_TYPE_DOMAIN_BYTE ==
addressTypeByte) {
        int domainNameLength = inputStream.read();
        int portBytes = 2;

```

```

        inputStream.read(new byte[domainNameLength + portBytes], 0,
domainNameLength + portBytes);
    }
}
}

```

[Back to Example \[page 184\]](#)

[Back to Content \[page 182\]](#)

Troubleshooting

If the handshake with the SOCKS5 proxy server fails, a SOCKS5 protocol error is returned, see IETF [RFC 1928](#). The table below shows the most common errors and their root cause in the scenario you use:

SOCKS5 Error Code	Technical Description	Client-Side Error Description	Scenario Error
0x00	SUCCESS		Success
0x01	FAILURE		Connection closed by backend or general scenario failure.
0x02	FORBIDDEN	Connection not allowed by ruleset	No matching host mapping found in Cloud Connector access control settings, see Configure Access Control (TCP) [page 396] .
0x03	NETWORK_UNREACHABLE		The Cloud Connector is not connected to the subaccount and the Cloud Connector Location ID that is used by the cloud application can't be identified. See Connect and Disconnect a Cloud Subaccount [page 524] and Managing Subaccounts [page 338] , section <i>Procedure</i> .
0x04	HOST_UNREACHABLE		Cannot open connection to the backend, that is, the host is unreachable.
0x05	CONNECTION_REFUSED		Authentication failure
0x06	TTL_EXPIRED		Not used
0x07	COMMAND_UNSUPPORTED		Only the SOCKS5 CONNECT command is supported.

SOCKS5 Error Code	Technical Description	Client-Side Error Description	Scenario Error
0x08	ADDRESS_UNSUPPORTED		Only the SOCKS5 DOMAIN and IPv4 commands are supported.

Back to [Content \[page 182\]](#)

1.1.4.2 Consuming the Destination Service

Retrieve and store externalized technical information about the destination to consume a target remote service from your Cloud Foundry application.

Tasks

Task Type	Task
	Overview [page 191] Prerequisites [page 192]
Operator and/or Developer	 Steps [page 193] <ol style="list-style-type: none"> 1. Read Credentials from the Environment Variables [page 194] 2. Generate a JSON Web Token (JWT) [page 194] 3. Call the Destination Service [page 195]
	Destination Configuration Attributes [page 198]
Operator and/or Developer	

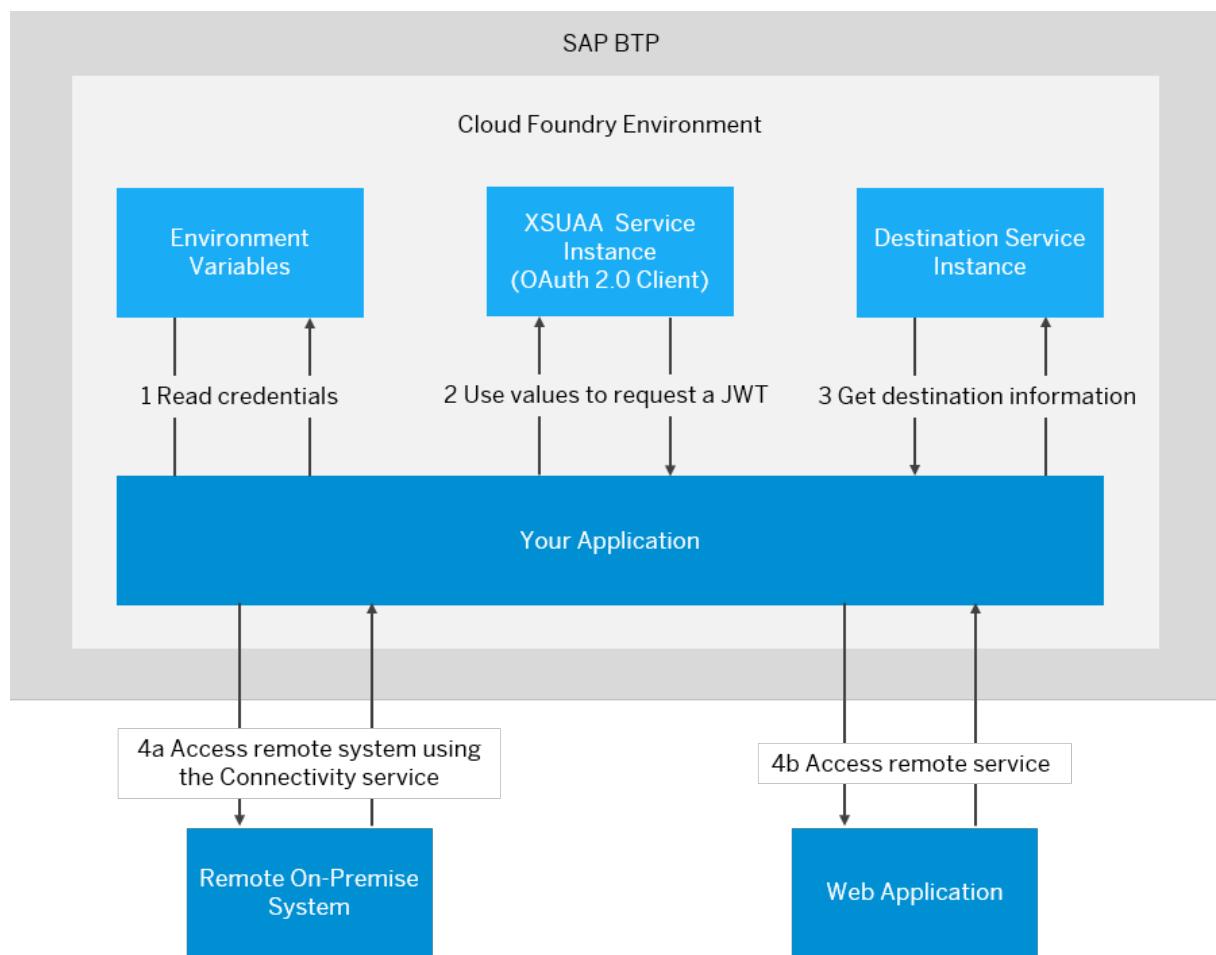
Overview



The Destination service lets you find the destination information that is required to access a remote service or system from your Cloud Foundry application.

- For the connection to an *on-premise system*, you can optionally use this service, together with (i.e. in addition to) the Connectivity service, see [Consuming the Connectivity Service \[page 165\]](#).
- For the connection to any other *Web application* (remote service), you can use the Destination service without the Connectivity service.

Consuming the Destination Service includes user authorization via a JSON Web Token (JWT) that is provided by the xsuaa service.



Back to [Tasks \[page 191\]](#)

Prerequisites



- To manage destinations and certificates on service instance level (all CRUD operations), you must be assigned to one of the following roles: `OrgManager`, `SpaceManager` or `SpaceDeveloper`.

i Note

The role `SpaceAuditor` has only `Read` permission for destinations and certificates.

- To consume the Destination service from an application, you must create a service instance and bind it to the application. See [Create and Bind a Destination Service Instance \[page 162\]](#).
- To generate the required JSON Web Token (JWT), you must bind the application to an instance of the `xsuaa` service using the service plan 'application'. The `xsuaa` service instance acts as an OAuth 2.0 client and grants user access to the bound application. Make sure that you set the `xsappname` property when creating the instance. Find a detailed guide for this procedure in section 3. *Creation of the Authorization & Trust Management Instance (aka XSUAA)* of the SCN blog [How to use SAP BTP Connectivity and Cloud Connector in the Cloud Foundry environment](#).
- You need at least one configured destination, otherwise there will be nothing to retrieve via the service. To access the `Destinations` editor in the cockpit, follow the steps in [Access the Destinations Editor \[page 41\]](#).

To manage destinations via REST API, see [Destination Service REST API \[page 59\]](#).

Back to [Tasks \[page 191\]](#)

Steps



To consume the Destination service from your application, perform the following basic steps:

1. [Read Credentials from the Environment Variables \[page 194\]](#)
2. [Generate a JSON Web Token \(JWT\) \[page 194\]](#)
3. [Call the Destination Service \[page 195\]](#)



Back to [Tasks \[page 191\]](#)

Read Credentials from the Environment Variables



The Destination service stores its credentials in the environment variables. To consume the service, you require the following information:

- The value of `clientid`, `clientsecret` and `uri` from the Destination service credentials.
- The values of `url` from the `xsuaa` credentials.

You can access this information as follows:

- From the CLI, the following command lists the environment variables of `<app-name>`:

```
cf env <app-name>
```

- From within the application, the service credential can be accessed as described in [Consuming the Connectivity Service \[page 165\]](#).

i Note

Below, we refer to the `JSONObject`s, containing the instance credentials as `destinationCredentials` (for the Destination service) and `xsuaaCredentials` (for `xsuaa`).

Back to [Tasks \[page 191\]](#)

Generate a JSON Web Token (JWT)



Your application must create an OAuth client using the attributes `clientid` and `clientsecret`, which are provided by the Destination service instance. Then, you must retrieve a new JWT from UAA and pass it in the [Authorization](#) HTTP header.

Two examples how to achieve this (Java and cURL):

Java:

For a Java application, you can use a library that implements the client credentials OAuth flow. Here is an example of how the `clientCredentialsTokenFlow` can be obtained using the [XSUAA Token Client and Token Flow API](#) :

⚠ Caution

Make sure you get the latest API version.

A sample Maven dependency declaration:

```
<dependency>
    <groupId>com.sap.cloud.security.xsuaa</groupId>
    <artifactId>token-client</artifactId>
    <version><latest version (e.g.: 2.7.7)></version>
</dependency>
```

→ Remember

The XSUAA Token Client library works with multiple HTTP client libraries. Make sure you have one as Maven dependency.

The following sample uses the Apache REST client:

↳ Sample Code

```
// get value of "clientid" and "clientsecret" from the environment variables
String clientid = destinationCredentials.getString("clientid");
String clientsecret = destinationCredentials.getString("clientsecret");

// get the URL to xsuaa from the environment variables
URI xsuaaUri = new URI(xsuaaCredentials.getString("url"));

// use the XSUAA client library to ease the implementation of the user token
// exchange flow
XsuaaTokenFlows tokenFlows = new XsuaaTokenFlows(new
DefaultOAuth2TokenService(), new XsuaaDefaultEndpoints(xsUaaUri.toString()),
new ClientCredentials(clientid, clientsecret));

String jwtToken =
tokenFlows.clientCredentialsTokenFlow().execute().getAccessToken();
```

For more information about caching, see also [XSUAA Token Client and Token Flow API - Cache](#).

cURL:

↳ Sample Code

```
curl -X POST \
<xsuaa-url>/oauth/token \
-H 'authorization: Basic <<clientid>:<clientsecret> encoded with Base64>' \
-H 'content-type: application/x-www-form-urlencoded' \
-d 'client_id=<clientid>&grant_type=client_credentials'
```

Back to [Tasks \[page 191\]](#)

Call the Destination Service



When calling the Destination service, use the `uri` attribute, provided in `VCAP_SERVICES`, to build the request URLs.

[Read a Destination by only Specifying its Name \("Find Destination"\) \[page 196\]](#)

[Read a Destination Associated with a Subaccount \[page 197\]](#)

[Get All Destinations Associated with a Subaccount \[page 197\]](#)

[Response Codes \[page 198\]](#)

Read a Destination by only Specifying its Name ("Find Destination")

This lets you provide simply a name of the destination while the service will search for it. First, the service searches the destinations that are associated with the service instance. If none of the destinations match the requested name, the service searches the destinations that are associated with the subaccount.

- Path: `/destination-configuration/v1/destinations/<destination-name>`
- Example of a call (cURL):

↳ Sample Code

```
curl "<uri>/destination-configuration/v1/destinations/<destination-name>" \
-X GET \
-H "Authorization: Bearer <jwtToken>"
```

- Example of a response (this is a destination found when going through the subaccount destinations):

↳ Sample Code

```
{
  "owner": {
    "SubaccountId": "<id>",
    "InstanceId": null
  },
  "destinationConfiguration": {
    "Name": "demo-internet-destination",
    "URL": "http://www.google.com",
    "ProxyType": "Internet",
    "Type": "HTTP",
    "Authentication": "NoAuthentication"
  }
}
```

i Note

The response from this type of call contains not only the configuration of the requested destination, but also some additional data. See ["Find Destination" Response Structure \[page 199\]](#).

Back to [Call the Destination Service \[page 195\]](#)

Read a Destination Associated with a Subaccount

This lets you retrieve the configurations of a destination that is defined within a subaccount, by providing the name of the destination.

- Path: /destination-configuration/v1/subaccountDestinations/<destination-name>
- Example of a call (cURL):

« Sample Code

```
curl "<uri>/destination-configuration/v1/subaccountDestinations/
<destination name>" \
-X GET \
-H "Authorization: Bearer <jwtToken>"
```

- Example of a response:

« Sample Code

```
{
  "Name": "demo-internet-destination",
  "URL": "http://www.google.com",
  "ProxyType": "Internet",
  "Type": "HTTP",
  "Authentication": "NoAuthentication"
}
```

Back to [Call the Destination Service \[page 195\]](#)

Get All Destinations Associated with a Subaccount

This lets you retrieve the configurations of all destinations that are defined within a subaccount.

- Path: /destination-configuration/v1/subaccountDestinations
- Example of a call (cURL):

« Sample Code

```
curl "<uri>/destination-configuration/v1/subaccountDestinations" \
-X GET \
-H "Authorization: Bearer <jwtToken>"
```

- Example of a response:

« Sample Code

```
[
{
  "Name": "demo-onpremise-destination1",
  "URL": "http://virtualhost:1234",
  "ProxyType": "OnPremise",
  "Type": "HTTP",
  "Authentication": "NoAuthentication"
}, {

```

```
"Name": "demo-onpremise-destination2",
"URL": "http://virtualhost:4321",
"ProxyType": "OnPremise",
>Type": "HTTP",
"Authentication": "BasicAuthentication",
>User": "mynname123",
>Password": "123456"
}
]
```

Back to [Call the Destination Service \[page 195\]](#)

Response Codes

When calling the Destination service, you may get the following response codes:

- **200**: OK (Json of Destination)
- **401**: Unauthorized (Authentication Failed)
- **403**: Forbidden (Authorization Failed)
- **404**: The requested destination could not be found (not applicable to 'get all destinations associated with a subaccount')
- **500**: Internal Server Error

Back to [Call the Destination Service \[page 195\]](#)

Back to [Tasks \[page 191\]](#)

Destination Configuration Attributes



The JSON object that serves as the response of a successful request (value of the `destinationConfiguration` property for "Find destination") can have different attributes, depending on the authentication type and proxy type of the corresponding destination. See [HTTP Destinations \[page 64\]](#).

Back to [Tasks \[page 191\]](#)

Related Information

[User Propagation via SAML 2.0 Bearer Assertion Flow \[page 202\]](#)

[Destination Service REST API \[page 59\]](#)

[Exchanging User JWTs via OAuth2UserTokenExchange Destinations \[page 207\]](#)

[Use Cases \[page 220\]](#)

[Multitenancy in the Destination Service \[page 209\]](#)

[Destination Java APIs \[page 211\]](#)

1.1.4.2.1 "Find Destination" Response Structure

Overview of data that are returned by the Destination service for the call type "find destination".

Response Structure

When you use the "find destination" call (read a destination by only specifying its name), the structure of the response includes four parts:

- The [owner of the destination \[page 199\]](#).
- The actual [destination configuration \[page 200\]](#).
- (Optional) [Authentication tokens \[page 200\]](#) that are relevant to the destination.
- (Optional) [Certificates \[page 201\]](#) that are relevant to the destination.

Each of these parts is represented in the JSON object as a key-value pair and their values are JSON objects, see [Example \[page 202\]](#).

See also [Call the Destination Service \[page 195\]](#).

Destination Owner

- **Key:** `owner`

The JSON object that represents the value of this property contains two properties itself: `SubaccountId` and `InstanceId`. Depending on where the destination was found (as a service instance destination or a subaccount destination) one of these properties has the value null, and the other one shows the ID of the subaccount/service instance, to which the destination belongs.

- **Example:**

« Sample Code

```
"owner": {  
    "SubaccountId": "9acf4877-5a3d-43d2-b67d-7516efe15b11",  
    "InstanceId": null  
}
```

Back to [Response Structure \[page 199\]](#)

Destination Configuration

- **Key:** destinationConfiguration
The JSON object that represents the value of this property contains the actual properties of the destination. To learn more about the available properties, see [HTTP Destinations \[page 64\]](#).
- **Example:**

« Sample Code

```
"destinationConfiguration": {  
    "Name": "TestBasic",  
    "Type": "HTTP",  
    "URL": "http://sap.com",  
    "Authentication": "BasicAuthentication",  
    "ProxyType": "OnPremise",  
    "User": "test",  
    "Password": "pass12345"  
}
```

Back to [Response Structure \[page 199\]](#)

Authentication Tokens

i Note

This property is only applicable to destinations that use the following authentication types:
BasicAuthentication, OAuth2SAMLBearerAssertion, OAuth2ClientCredentials, OAuthUserTokenExchange, OAuth2JWTBearer, OAuth2Password, and SAPAssertionSSO.

- **Key:** authTokens
The JSON array that represents the value of this property contains tokens that are required for authentication. These tokens are represented by JSON objects with these properties (expect more new properties to be added in the future):
 - **type:** the type of the token.
 - **value:** the actual token.
 - **http_header:** JSON object containing the prepared token in the correct format. The **<key>** field contains the key of the HTTP header. The **<value>** field contains the value of the header.
 - **expires_in** (only in OAuth2 destinations): The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.
 - **error** (optional): if the retrieval of the token fails, the value of both **type** and **value** is an empty string and this property shows an error message, explaining the problem.
 - **scope** (optional) (only in OAuth2 destinations): The scopes issued with the token. The value of the **scope** parameter is expressed as a list of space-delimited strings. For example, `read write execute`.

- **Example:**

 Sample Code

```
"authTokens": [
    {
        "type": "Basic",
        "value": "dGVzdDpwYXNzMTIzMjU=",
        "http_header": {
            "key": "Authorization",
            "value": "Basic dGVzdDpwYXNzMTIzMjU="
        }
    }
]
```

[Back to Response Structure \[page 199\]](#)

Certificates

 Note

This property is only applicable to destinations that use the following authentication types:
ClientCertificateAuthentication, *OAuth2SAMLBearerAssertion* (when default JDK trust store is not used).

- **Key:** certificates

The JSON array that represents the value of this property contains the certificates, specified in the destination configuration. These certificates are represented by JSON objects with these properties (expect more new properties to be added in the future):

- type
- content: the encoded content of the certificate.
- name: the name of the certificate, as specified in the destination configuration.

- **Example:**

 Sample Code

```
"certificates": [
    {
        "Name": "keystore.jks",
        "Content": "<value>",
        "Type": "CERTIFICATE"
    },
    {
        "Name": "truststore.jks",
        "Content": "<value>",
        "Type": "CERTIFICATE"
    }
]
```

[Back to Response Structure \[page 199\]](#)

Example

Example of a full response for a destination using basic authentication:

↳ Sample Code

```
{  
    "owner": {  
        "SubaccountId": "9acf4877-5a3d-43d2-b67d-7516efe15b11",  
        "InstanceId": null  
    },  
    "destinationConfiguration": {  
        "Name": "TestBasic",  
        "Type": "HTTP",  
        "URL": "http://sap.com",  
        "Authentication": "BasicAuthentication",  
        "ProxyType": "OnPremise",  
        "User": "test",  
        "Password": "pass12345"  
    },  
    "authTokens": [  
        {  
            "type": "Basic",  
            "value": "dGVzdDpwYXNzMTIzNDU=",  
            "http_header": {  
                "key": "Authorization",  
                "value": "Basic dGVzdDpwYXNzMTIzNDU="  
            }  
        }  
    ]  
}
```

Back to [Response Structure \[page 199\]](#)

1.1.4.2.2 User Propagation via SAML 2.0 Bearer Assertion Flow

Learn about the process for automatic token retrieval, using the OAuth2SAMLBearerAssertion authentication type for HTTP destinations.

Tasks

Task Type	Task
 	Prerequisites [page 203]
Operator and/or Developer	
	Automated Access Token Retrieval [page 203] <ul style="list-style-type: none"> • Determine the Propagated User ID [page 203] • Propagate User Attributes [page 205] • Scenarios [page 206]
Developer	

Prerequisites



- You have configured an `OAuth2SAMLBearerAssertion` destination. See [OAuth SAML Bearer Assertion Authentication \[page 71\]](#).
- Unless using the destination property `SystemUser`, the user's identity should be represented by a JSON Web token (JWT).

 Note

Though actually not being a strict requirement, it is likely that you need a user JWT to get the relevant information. See [SAP Authorization and Trust Management Service in the Cloud Foundry Environment](#).

- If you are using custom user attributes to determine the user, the JWT representing the user (that is passed to the Destination service) must have the `user_attributes` scope.

[Back to Tasks \[page 202\]](#)

Automated Access Token Retrieval



For an `OAuth2SAMLBearerAssertion` destination, you can use the automated token retrieval functionality that is available via the "find destination" endpoint. See [Destination Service REST API \[page 59\]](#).

Determine the Propagated User ID

There are currently three sources that can provide the propagated user ID. They are prioritized, meaning that the lookup always starts from the top-priority source and goes down the list. If the propagated user ID is not found at a given level, the next level is checked. If not found on any level, the operation would fail.

Find the available sources in the table below, in order of their priority.

Propagated User ID: Sources

Source	Procedure
<i>System User</i>	The system user is a special user ID that is hardcoded in your destination as value of the <code>SystemUser</code> property. If you set this property, its value is used as the propagated user ID.
<i>Field in the JWT</i>	<p>In this case, the Destination service looks for the user ID as a field in the provided JWT. When you make the HTTP call to the Destination service, you must provide the Authorization header. The value must be a JWT in its encoded form (see RFC 7519). The procedure is as follows:</p> <ul style="list-style-type: none">• If the <code>userIdSource</code> property is configured in the destination, its value is the key of the JWT field that will be the user ID (if there is no such key in the JWT, the flow proceeds to the next level). There are 2 options:<ul style="list-style-type: none">◦ plain string: the exact match is searched on the root-level element keys of the JWT.◦ <code>JsonPath</code> expression: lets you use non-root-level elements of the JWT.• If the <code>userIdSource</code> property is missing, the flow falls back to the destination property <code>nameIdFormat</code>. It must have one of the following two values (or not be set at all), otherwise an exception is thrown:<ul style="list-style-type: none">◦ <code>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</code>: the <code>email</code> element of the JWT is the user ID. If there is no such element in the JWT, an exception is thrown.◦ <code>urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</code> (or not set): the <code>user_name</code> element of the JWT is the user ID. If there is no such element in the JWT, an exception is thrown.

Source	Procedure
Custom User Attribute	<p>Like <i>Field in the JWT</i>, this source must use the Authorization header. In this case, its value is used to retrieve the custom user attributes from the Identity Provider (XSUAA). One of those attributes can be used as the propagated user ID. The access token from the header must be a user JWT with the <code>user_attributes</code> scope. Otherwise the custom attributes cannot be retrieved, and the operation results in an error.</p> <ul style="list-style-type: none"> • If the <code>userIdSource</code> property is configured in the destination, the same logic applies as for <i>Field in the JWT</i>, but this time on the JSON containing the custom user attributes. • If <code>userIdSource</code> is missing or the desired key is not found in the custom attributes, the operation fails (<code>user ID could not be determined</code>).

Back to [Tasks \[page 202\]](#)

Propagate User Attributes

You can read additional user attributes from the identity provider (XSUAA), and propagate them as SAML attributes in the generated SAML bearer assertion.

These attributes are similar to the ones returned by the Cloud Foundry UAA user info endpoint. However, they may differ depending on the XSUAA behavior, and are specific to the identity provider you use.

For more details about these attributes and possible values, see <https://docs.cloudfoundry.org/api/uaa/version/74.15.0/index.html#user-info>.

i Note

When adding the attributes, the following rules apply:

- All root elements, except for `user_attributes`, are added "as is", that is, the attribute name and value are displayed as seen in the source (user info response structure).
- Elements under the `user_attribute` key are parsed and added as attributes prefixed with '`user_attributes.`'. For example, having `{"user_attributes": { "my_param": "my_value" }}` will result in an attribute called '`user_attributes.my_param`' with value '`my_value`' in the SAML assertion.

In addition to identity provider (XSUAA) user info attributes, there are some more attributes which are read from the passed JWT. They are located via predefined JsonPath expressions and cannot be controlled by the end user:

- `$.['xs.system.attributes']['xs.saml.groups']`
- `$.['user_attributes']['xs.saml.groups']`

i Note

The '`xs.saml.groups`' attribute, read from the passed JWT, is renamed to '`Groups`' in the resulting SAML assertion. See also [Federation Attribute Settings of Any Identity Provider](#).

Back to [Tasks \[page 202\]](#)

Scenarios

Refer to the table below to find the JWT requirements for a specific scenario:

Scenario	Authorization Header
Propagate a technical user principal, using the <code>SystemUser</code> property of an <code>OAuth2SAMLBearerAssertion</code> destination maintained in the subscriber subaccount , and used by the provider application .	Access token, retrieved <ul style="list-style-type: none">via the client credentials of the Destination service instance (bound to the application).using the subscriber's tenant-specific Token Service URL.
Propagate a technical user principal, using the <code>SystemUser</code> property of an <code>OAuth2SAMLBearerAssertion</code> destination maintained in the same subaccount where the application is deployed.	Access token, retrieved <ul style="list-style-type: none">via the client credentials of the Destination service instance (bound to the application).using the provider's tenant-specific Token Service URL.
Propagate a business user principal, using an <code>OAuth2SAMLBearerAssertion</code> destination maintained in the subscriber subaccount where the application is deployed. The business user is represented by a JWT that was issued by the subscriber .	The JWT, previously retrieved from the application <ul style="list-style-type: none">by exchanging the JWT (that represents the user) to another user access token via the client credentials of the Destination service instance (bound to the application).using the subscriber's tenant-specific Token Service URL.
Propagate a business user principal, using an <code>OAuth2SAMLBearerAssertion</code> destination maintained in the same subaccount where the application is deployed. The business user is represented by a JWT that was issued by the provider .	The JWT, previously retrieved by the application <ul style="list-style-type: none">by exchanging the JWT (that represents the user) to another user access token via the client credentials of the Destination service instance (bound to the application).using the provider's tenant-specific Token Service URL.

Back to [Tasks \[page 202\]](#)

1.1.4.2.3 Destination Service REST API

Destination service REST API specification for the SAP Cloud Foundry environment.

The Destination service provides a REST API that you can use to read and manage destinations and certificates on all available levels. This API is documented in the [SAP API Business Hub](#).

It shows all available endpoints, the supported operations, parameters, possible error cases and related status codes, etc. You can also execute requests using the credentials (for example, the service key) of your Destination service instance, see [Create and Bind a Destination Service Instance \[page 162\]](#).

1.1.4.2.4 Exchanging User JWTs via OAuth2UserTokenExchange Destinations

Automatic token retrieval using the `OAuth2UserTokenExchange` authentication type for HTTP destinations.

Content

[Prerequisites \[page 207\]](#)

[Automated Access Token Retrieval \[page 207\]](#)

[Scenarios \[page 208\]](#)

Prerequisites

You have configured an `OAuth2UserTokenExchange` destination. See [OAuth User Token Exchange Authentication \[page 87\]](#).

The token to be exchanged must have the `uaa.user` scope to enable the token exchange. See [SAP Authorization and Trust Management Service in the Cloud Foundry Environment](#) for more details.

Back to [Content \[page 207\]](#)

Automated Access Token Retrieval

For destinations of authentication type `OAuth2UserTokenExchange`, you can use the automated token retrieval functionality via the "find destination" endpoint, see [Call the Destination Service \[page 195\]](#).

If you provide the user token exchange header with the request to the Destination service and its value is not empty, it is used instead of the `Authorization` header to specify the user and the tenant subdomain. It will be the token for which token exchange is performed.

- The header value must be a user JWT (JSON Web token) in encoded form, see [RFC 7519](#).
- If the user token exchange header is not provided with the request to the Destination Service or it is provided, but its value is empty, the token from the `Authorization` header is used instead. In this case, the JWT in the `Authorization` header must be a user JWT in encoded form, otherwise the token exchange does not work.

For information about the response structure of this request, see "[Find Destination](#)" Response Structure [page 199].

Back to [Content](#) [page 207]

Scenarios

To achieve specific token exchange goals, you can use the following headers and values when calling the Destination service:

Goal	User Token Exchange Header	Authorization Header
Exchange a user token: <ul style="list-style-type: none"> • Issued on behalf of the application provider tenant • Using a destination in the application provider tenant 	Not used	The user token to be exchanged Previously retrieved by the application via exchanging the initial user token, passed to the application (to another user access token) via the client credentials of the Destination service instance (bound to the application), using the provider tenant-specific token service URL.
Exchange a user token: <ul style="list-style-type: none"> • Issued on behalf of a tenant, subscribed to your application • Using a destination in the application provider tenant 	<User token to be exchanged>	Access token Retrieved via the client credentials of the Destination service instance (bound to the application), using the provider tenant-specific token service URL.
Exchange a user token: <ul style="list-style-type: none"> • Issued on behalf of a tenant, subscribed to your application • Using a destination in the subscriber tenant 	<User token to be exchanged>	Access token Retrieved via the client credentials of the Destination service instance (bound to the application), using the subscriber tenant-specific token service URL.

Back to [Content](#) [page 207]

1.1.4.2.5 Multitenancy in the Destination Service

Establish multitenancy in the Destination service using subscription-level destinations.

Concept

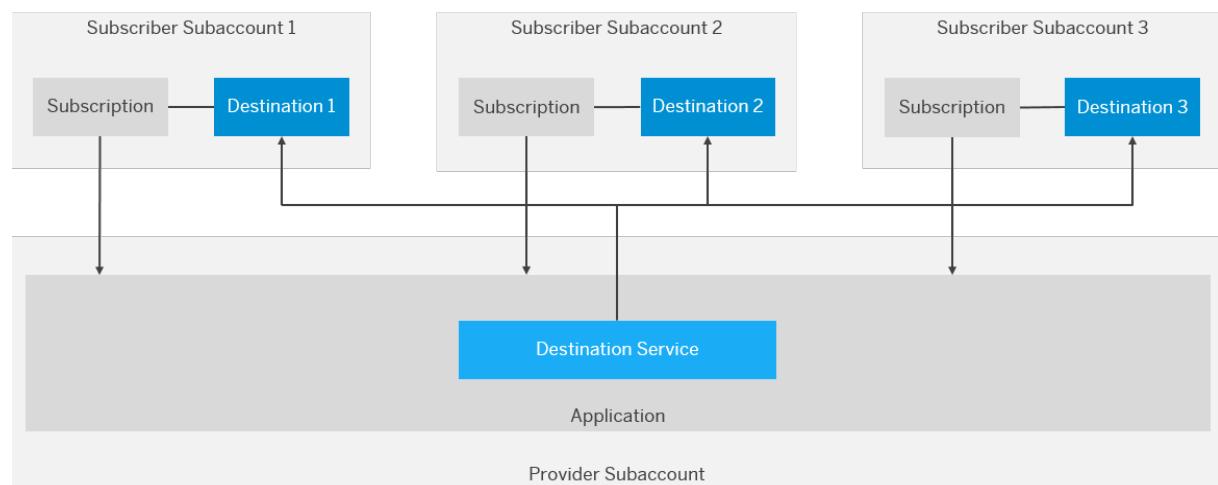
When developing a provider application (SaaS application) that consumes the Destination service, you can choose between the following destination levels:

Level	Who has Access and How?
Subaccount	Any application using <i>any</i> destination service instance in the subaccount context. This is the common level for all applications and service instances in the subaccount.
Service Instance	Any application using the concrete destination service instance in the context of the <i>provider</i> subaccount (the subaccount in which the instance is provisioned). Each service instance has its own level.
Subscription	Any application using the <i>concrete</i> destination service instance in the context of the <i>subscribed</i> subaccount. Each combination of service instance and subscriber subaccount is a unique level.

i Note

The term *level* is used here to represent an area or visibility scope. The higher the level, the broader is the visibility scope.

If you, as an application provider, want to create a destination that is used at runtime *only by the subscriber* and that should be *visible and accessible only to the subscriber*, you can create a subscription-level destination for each subscriber subaccount (tenant):



[Create a Subscription-Level Destination \[page 210\]](#)

[Consume a Subscription-Level Destination \[page 210\]](#)

Create a Subscription-Level Destination

1. Retrieve an OAuth token from the subscriber token service URL using the OAuth client credentials from the destination service instance, for example:

```
POST https://{{subscriber subdomain}}.authentication.{region host}/oauth/token
```

2. Use the retrieved token from step 1 to create (POST) a subscription-level destination in the Destination service, see *Destinations on service instance (subscription) level* in the [REST API specification](#).

[Back to Concept \[page 209\]](#)

Consume a Subscription-Level Destination

1. Retrieve an OAuth token from the subscriber token service URL using the OAuth client credentials from the destination service instance, for example:

```
POST https://{{subscriber subdomain}}.authentication.{region host}/oauth/token
```

2. Use the token from step 1 to retrieve (GET) the destination stored on subscription level from the Destination service via:
 - *Find a destination* in the [REST API specification](#)
 - *Destinations on service instance (subscription) level* in the [REST API specification](#)

[Back to Concept \[page 209\]](#)

Related Information

[Multitenancy in the Connectivity Service \[page 157\]](#)

1.1.4.2.6 Destination Java APIs

Use Destination service Java APIs to optimize application development in the Cloud Foundry environment.

When running your cloud application with SAP Java Buildpack, you can use the following Java APIs to optimize the application development:

- [ConnectivityConfiguration API \[page 211\]](#): Retrieve destination configurations.
- [AuthenticationHeaderProvider API \[page 213\]](#): Retrieve prepared authentication headers, ready to be used towards the remote target system.

For more information on SAP Java Buildpack, see [Developing Java in the Cloud Foundry Environment](#).

1.1.4.2.6.1 ConnectivityConfiguration API

Use the ConnectivityConfiguration API to retrieve destination configurations and certificate configurations in the Cloud Foundry environment.

Overview

The ConnectivityConfiguration API is visible by default from the web applications hosted on SAP Java Buildpack. You can access it via a JNDI lookup.

Besides managing destination configurations, you can also allow your applications to use their own managed HTTP clients. The ConnectivityConfiguration API provides you with direct access to the destination configurations of your applications.

To learn how to retrieve destination configurations, follow the procedure below.

Procedure

1. To consume a connectivity configuration using JNDI, you must define the ConnectivityConfiguration API as a resource in the `context.xml` file.

Example of a ConnectivityConfiguration resource named `connectivityConfiguration`, which is described in the `context.xml` file:

↳ Sample Code

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
    <Resource name="connectivityConfiguration" auth="Container"
```

```
type="com.sap.core.connectivity.api.configuration.ConnectivityConfiguration"
"
factory="com.sap.core.connectivity.api.jndi.ServiceObjectFactory"/>
</Context>
```

2. In your servlet code, you can look up the ConnectivityConfiguration API from the JNDI registry as follows:

↳ Sample Code

```
import javax.naming.Context;
import javax.naming.InitialContext;
import
com.sap.core.connectivity.api.configuration.ConnectivityConfiguration;
...
// look up the connectivity configuration API "connectivityConfiguration"
Context ctx = new InitialContext();
ConnectivityConfiguration configuration = (ConnectivityConfiguration)
ctx.lookup("java:comp/env/connectivityConfiguration");
```

3. With the retrieved ConnectivityConfiguration API, you can read all properties of any destination defined on subscription, application, or subaccount level.

↳ Sample Code

```
// get destination configuration for "myDestinationName"
DestinationConfiguration destConfiguration =
configuration.getConfiguration("myDestinationName");

// get a single destination property
String authenticationType =
destConfiguration.getProperty("Authentication");

// get all destination properties
Map<String, String> allDestinationProperties =
destConfiguration.getAllProperties();
```

i Note

If you have two destinations with the same name, for example, one configured on subaccount level and the other on service instance/subscription level, the `getConfiguration()` method will return the destination on instance/subscription level.

The preference order is:

1. Instance/subscription level
2. Subaccount level

4. If a trust store and a key store are defined in the corresponding destination, you can access them by using the methods `getKeyStore` and `getTrustStore`.

↳ Sample Code

```
// get destination configuration for "myDestinationName"
```

```

DestinationConfiguration destConfiguration =
configuration.getConfiguration("myDestinationName");

// get the configured keystore
KeyStore keyStore = destConfiguration.getKeyStore();

// get the configured truststore
KeyStore trustStore = destConfiguration.getTrustStore();

// create sslcontext
TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(trustStore);

KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());

// get key store password from destination
String keyStorePassword =
destConfiguration.getProperty("KeyStorePassword");
keyManagerFactory.init(keyStore, keyStorePassword.toCharArray());

SSLContext sslcontext = SSLContext.getInstance("TLSv1");
sslcontext.init(keyManagerFactory.getKeyManagers(),
tmf.getTrustManagers(), null);
SSLSocketFactory sslSocketFactory = sslcontext.getSocketFactory();

// get the destination URL
String value = destConfiguration.getProperty("URL");
URL url = new URL(value);

// use the sslcontext for url connection
URLConnection urlConnection = url.openConnection();
((HttpsURLConnection) urlConnection).setSSLSocketFactory(sslSocketFactory);
urlConnection.connect();
InputStream in = urlConnection.getInputStream();
...

```

1.1.4.2.6.2 AuthenticationHeaderProvider API

Use the `AuthenticationHeaderProvider` API for applications in the Cloud Foundry environment to retrieve prepared authentication headers that are ready to be used towards a remote target system.

Overview

The `AuthenticationHeaderProvider` API is visible by default from the web applications hosted on SAP Java Buildpack. You can access it via a JNDI lookup.

This API lets your applications use their own managed HTTP clients, as it provides them with automated authentication token retrieval, making it easy to implement *single sign-on* (SSO) towards the remote target.

Procedure

1. To consume the authentication header provider API using JNDI, you need to define the

`AuthenticationHeaderProvider` API as a resource in the `context.xml` file.

Example of an `AuthenticationHeaderProvider` resource named `myAuthHeaderProvider`, which is described in the `context.xml` file:

« Sample Code

```
<?xml version='1.0' encoding='utf-8'?>

<Context>
    <Resource name="myAuthHeaderProvider" auth="Container"
        type="com.sap.core.connectivity.api.authentication.AuthenticationHeaderProvider"
        factory="com.sap.core.connectivity.api.jndi.ServiceObjectFactory"/>
</Context>
```

2. In your servlet code, you can look up the `AuthenticationHeaderProvider` API from the JNDI registry as follows:

« Sample Code

```
import javax.naming.Context;
import javax.naming.InitialContext;
import
com.sap.core.connectivity.api.authentication.AuthenticationHeaderProvider;
...

// look up the connectivity authentication header provider resource called
//"myAuthHeaderProvider"
Context ctx = new InitialContext();
AuthenticationHeaderProvider authHeaderProvider =
(AuthenticationHeaderProvider) ctx.lookup("java:comp/env/
myAuthHeaderProvider");
```

Single Sign-On to On-Premise Systems

The Connectivity service supports a mechanism to enable SSO using the so-called *principal propagation* authentication type of a destination configuration. To propagate the logged-in user, the application can use the `AuthenticationHeaderProvider` API to retrieve a prepared HTTP header, which then embeds in the HTTP request to the on-premise system.

Prerequisites

To connect to on-premise systems and perform single sign-on, you must bind a Connectivity service instance to the cloud application.

References

For more information, see also:

- [Principal Propagation SSO Authentication for HTTP \[page 68\]](#)
- [Create and Bind a Connectivity Service Instance \[page 160\]](#)
- [Consuming the Connectivity Service \(Java\) \(Neo environment\)](#)

Example

Sample Code

```
String proxyHost = <connectivity_service_credentials_onPremiseProxyHost>;
int proxyPort =
Integer.parseInt(<connectivity_service_credentials_onPremiseProxyPort>);
String account = SecurityContext.getAccessToken().getZoneId();

// setup the on-premise HTTP proxy
HttpClient httpClient = new DefaultHttpClient();
httpClient.getParams().setParameter(ConnRoutePNames.DEFAULT_PROXY, new
HttpHost(proxyHost, proxyPort));

// look up the connectivity authentication header provider resource called
//"authHeaderProvider" (must be defined in web.xml)
Context ctx = new InitialContext();
AuthenticationHeaderProvider authHeaderProvider =
(AuthenticationHeaderProvider) ctx.lookup("java:comp/env/authHeaderProvider");

// get header for principal propagation
AuthenticationHeader principalPropagationHeader =
authHeaderProvider.getPrincipalPropagationHeader();

//insert the necessary headers in the request
HttpGet request = new HttpGet("http://virtualhost:1234");
request.addHeader(principalPropagationHeader.getName(),
principalPropagationHeader.getValue());
request.addHeader("SAP-Connectivity-ConsumerAccount", account);

// execute the request
HttpResponse response = httpClient.execute(request);
```

OAuth2 SAML Bearer Assertion

The Destination service provides support for applications to use the SAML Bearer assertion flow for consuming OAuth-protected resources. In this way, applications do not need to deal with some of the complexities of OAuth and can reuse user data from existing identity providers.

Users are authenticated by using a SAML assertion against the configured and trusted OAuth token service. The SAML assertion is then used to request an access token from an OAuth token service. This access token is returned by the API and can be injected in the HTTP request to access the remote OAuth-protected resources via SSO.

→ Tip

The access tokens are cached by `AuthenticationHeaderProvider` and are auto-updated: When a token is about to expire, a new token is created shortly before the expiration of the old one.

The `AuthenticationHeaderProvider` API provides the following method to retrieve such headers:

```
List<AuthenticationHeader>
getOAuth2SAMLBearerAssertionHeaders(DestinationConfiguration
destinationConfiguration);
```

For more information, see also [Principal Propagation SSO Authentication for HTTP \[page 68\]](#).

Client Credentials

The Destination service provides support for applications to use the OAuth client credentials flow for consuming OAuth-protected resources.

The client credentials are used to request an access token from an OAuth token service.

→ Tip

The access tokens are cached by `AuthenticationHeaderProvider` and are auto-updated: When a token is about to expire, a new token is created shortly before the expiration of the old one.

The `AuthenticationHeaderProvider` API provides the following method to retrieve such headers:

```
AuthenticationHeader getOAuth2ClientCredentialsHeader (DestinationConfiguration
destinationConfiguration);
```

For more information, see:

- [Principal Propagation SSO Authentication for HTTP \[page 68\]](#)
- [OAuth SAML Bearer Assertion Authentication \[page 71\]](#)
- [OAuth Client Credentials Authentication \[page 81\]](#)

Related Information

[Principal Propagation \[page 122\]](#)

1.1.4.3 Invoking ABAP Function Modules via RFC

Call a remote-enabled function module in an on-premise or cloud ABAP server from your Cloud Foundry application, using the RFC protocol.

Find the tasks and prerequisites that are required to consume an ABAP function module via RFC, using the Java Connector (JCo) API as a built-in feature of SAP BTP.

Tasks

Task Type	Task
	Prerequisites [page 217]
Operator	
	About JCo [page 218] Installation Prerequisites for JCo Applications [page 218]
Operator and/or Developer	Consume Connectivity via RFC [page 218] Restrictions [page 219]

Prerequisites



Before you can use RFC communication for an SAP BTP application, you must configure:

- A destination on SAP BTP to use RFC.
For more information, see [RFC Destinations \[page 111\]](#).
- (Only for on-premise backend systems) RFC connectivity between a backend system and the application.
To do this, you must install the [Cloud Connector \[page 276\]](#) in your internal network and configure it to expose a remote-enabled function module in an ABAP system.
For more information, see [Initial Configuration \(RFC\) \[page 331\]](#) and [Configure Access Control \(RFC\) \[page 387\]](#).

Back to [Tasks \[page 217\]](#)

About JCo



To learn in detail about the SAP JCo API, see the JCo 3.0 documentation on [SAP Support Portal](#).

i Note

Some sections of this documentation are not applicable to SAP BTP:

- **Architecture:** CPIC is only used in the last mile from your Cloud Connector to an *on-premise* ABAP backend. From SAP BTP to the Cloud Connector, TLS-protected communication is used.
- **Installation:** SAP BTP runtimes already include all required artifacts.
- **Customizing and Integration:** On SAP BTP, the integration is already done by the runtime. You can concentrate on your business application logic.
- **Server Programming:** The programming model of JCo on SAP BTP does not include server-side RFC communication.
- **IDoc Support for External Java Applications:** Currently, there is no IDocLibrary for JCo available on SAP BTP

Back to [Tasks \[page 217\]](#)

Installation Prerequisites for JCo Applications



- For connections to an *on-premise* ABAP backend, you have downloaded the Cloud Connector installation archive from [SAP Development Tools for Eclipse](#) and connected the Cloud Connector to your subaccount.
- You have downloaded and set up your Eclipse IDE and the [Eclipse Tools for Cloud Foundry](#).
- You have downloaded the Cloud Foundry CLI, see [Tools](#).

Back to [Tasks \[page 217\]](#)

Consuming Connectivity via RFC



You can call a service from a fenced customer network using an application that consumes a remote-enabled function module.

Invoking function modules via RFC is enabled by a JCo API that is comparable to the one available in SAP NetWeaver Application Server Java (version 7.10+), and in JCo standalone 3.0. If you are an experienced JCo developer, you can easily develop a Web application using JCo: you simply consume the APIs like you do in other Java environments. Restrictions that apply in the cloud environment are mentioned in the **Restrictions** section below.

Find a sample Web application in [Invoke ABAP Function Modules in On-Premise ABAP Systems \[page 220\]](#).

Back to [Tasks \[page 217\]](#)

Restrictions



- The supported **runtime environment** is SAP Java Buildpack as of version 1.8.0.

i Note

You must use the Tomcat or TomEE runtime offered by the build pack to make JCo work correctly. You cannot bring a container of your own.

- JCoServer** functionality cannot be used within SAP BTP.
- Environment embedding**, such as offered by JCo standalone 3.0, is not possible. This is, however, similar to SAP NetWeaver AS Java.
- Logon authentication** only supports user/password credentials (basic authentication) and principal propagation. See [Authentication to the On-Premise System \[page 174\]](#).
- The supported set of **configuration properties** is restricted. For details, see [RFC Destinations \[page 111\]](#).

Back to [Tasks \[page 217\]](#)

Related Information

[Use Cases \[page 220\]](#)

1.1.4.3.1 Use Cases

Find instructions for typical RFC end-to-end scenarios that use the Connectivity service and/or the Destination service (Cloud Foundry environment).

Invoke ABAP Function Modules in On-Premise ABAP Systems [page 220]	Call a function module in an on-premise ABAP system via RFC, using a sample Web application (Cloud Foundry environment).
Invoke ABAP Function Modules in Cloud ABAP Systems [page 245]	Call a function module in a cloud ABAP system via RFC, using a sample Web application (Cloud Foundry environment).
Multitenancy for JCo Applications (Advanced) [page 263]	Learn about the required steps to make your Cloud Foundry JCo application tenant-aware.
Configure Principal Propagation for RFC [page 274]	Enable single sign-on (SSO) via RFC by forwarding the identity of cloud users from the Cloud Foundry environment to an on-premise system.

1.1.4.3.1.1 Invoke ABAP Function Modules in On-Premise ABAP Systems

Call a function module in an on-premise ABAP system via RFC, using a sample Web application (Cloud Foundry environment).

This scenario performs a remote function call to invoke an ABAP function module, by using the Connectivity service and the Destination service in the Cloud Foundry environment, as well as a Cloud Foundry application router.

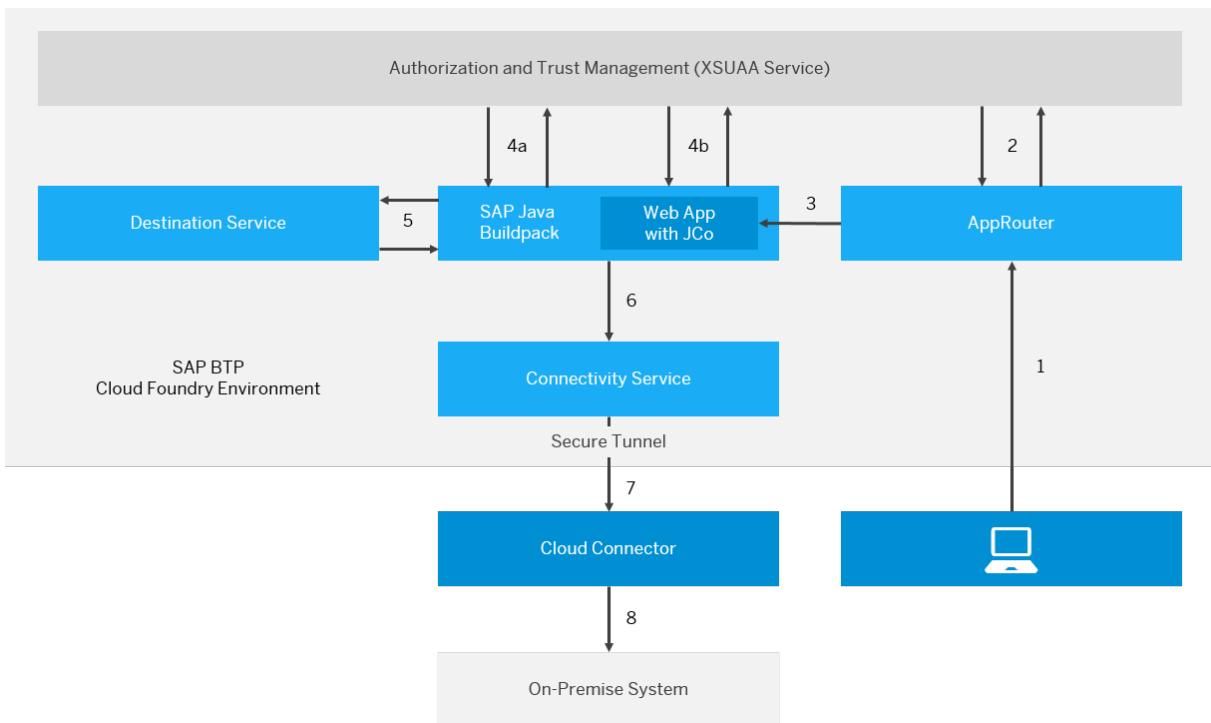
Tasks

Task Type	Task
 Operator and/or Developer	Scenario Overview [page 221] Connectivity User Roles [page 223] Installation Prerequisites [page 224] Used Values [page 223]
 Developer	Develop a Sample Web Application [page 224]

Task Type	Task
	Create and Bind Service Instances [page 229]
Operator and/or Developer	
	Deploy the Application [page 232]
Developer	
	Configure Roles and Trust [page 234]
Operator	
	Set Up an Application Router [page 235]
Operator and/or Developer	
	Configure the RFC Destination [page 239]
Operator	
	Configure the Cloud Connector [page 240]
Operator and/or Developer	
	Monitoring Your Web Application [page 244] (Optional)

Scenario Overview

Control Flow for Using the Java Connector (JCo) with Basic Authentication



Process Steps:

1. Call through AppRouter (entry point for business applications).

i Note

AppRouter is only required if you want to use multitenancy or perform user-specific service calls. In all other cases, JCo uses [cloud-security-xsuua-integration](#) with ClientCredentialFlow.

2. Redirect to XSUAA for login. JSON Web Token (JWT1) is sent to AppRouter and cached there.
3. AppRouter calls Web app and sends JWT1 with credentials.
4. Buildpack/XSUAA interaction:
 - 4a. Buildpack requests JWT2 to access the Destination service instance (JCo call).
 - 4b. Buildpack requests JWT3 to access the Connectivity service instance.
5. Buildpack requests destination configuration (JWT2).
6. Buildpack sends request to the Connectivity service instance (with JWT3 and Authorization Header).
7. Connectivity service forwards request to the Cloud Connector.
8. Cloud Connector sends request to on-premise system.

Since token exchanges are handled by the buildpack, you must only create and bind the service instances, see [Create and Bind Service Instances \[page 229\]](#).

Back to [Tasks \[page 220\]](#)

Used Values

This scenario uses:

- A subaccount in region **Europe (Frankfurt)**, for which the API endpoint is `api.cf.eu10.hana.ondemand.com`.
- The application name **jco-demo-<subaccount name>**, where `<subaccount name>` is the subdomain name of the subaccount. For this example, we use **p1234**.

Back to [Tasks \[page 220\]](#)

Connectivity User Roles

Different user roles are involved in the cloud to on-premise connectivity end-to-end scenario. The particular steps for the relevant roles are described below:

IT Administrator

Sets up and configures the Cloud Connector. Scenario steps:

1. Downloads the Cloud Connector from <https://tools.hana.ondemand.com/#cloud>
2. Installs the Cloud Connector.
3. Establishes an SSL tunnel from the connector to an SAP BTP subaccount.
4. Configures the exposed back-end systems and resources.

Application Developer

Develops Web applications using destinations. Scenario steps:

1. Installs Eclipse IDE, the Cloud Foundry Plugin for Eclipse and the Cloud Foundry CLI.
2. Develops a Java EE application using the JCo APIs.
3. Configures connectivity destinations via the SAP BTP cockpit.
4. Deploys and tests the Java EE application on SAP BTP.

Account Operator

Deploys Web applications, creates application routers, creates and binds service instances, conducts tests. Scenario steps:

1. Obtains a ready Java EE application WAR file.
2. Deploys an application router with respective routes to the application.
3. Creates an XSUAA service instance and binds it to the router.
4. Deploys the Java EE application to an SAP BTP subaccount.
5. Creates a Connectivity service and Destination service instance, and binds them to the application.
6. Creates and manages roles and role collections.

Back to [Tasks \[page 220\]](#)

Installation Prerequisites

- You have downloaded the Cloud Connector installation archive from [SAP Development Tools for Eclipse](#) and connected the Cloud Connector to your subaccount.
- You have downloaded and set up your Eclipse IDE and the [Eclipse Tools for Cloud Foundry](#) .
- You have downloaded the Cloud Foundry CLI, see [Tools](#).

Back to [Tasks \[page 220\]](#)

Next Steps

- [Develop a Sample Web Application \[page 224\]](#)
- [Create and Bind Service Instances \[page 229\]](#)
- [Deploy the Application \[page 232\]](#)
- [Configure Roles and Trust \[page 234\]](#)
- [Set Up an Application Router \[page 235\]](#)
- [Configure the RFC Destination \[page 239\]](#)
- [Configure the Cloud Connector \[page 240\]](#)
- [Monitoring Your Web Application \[page 244\]](#) (Optional)

Related Information

[Multitenancy for JCo Applications \(Advanced\) \[page 263\]](#)

1.1.4.3.1.1.1 Develop a Sample Web Application

Create a Web application to call an ABAP function module via RFC.

Steps

1. [Create a Dynamic Web Project \[page 225\]](#)
2. [Include JCo Dependencies \[page 226\]](#)
3. [Create a Sample Servlet \[page 227\]](#)

Create a Dynamic Web Project

1. Open the *Java EE* perspective of the Eclipse IDE.
2. On the *Project Explorer* view, choose *New* > *Dynamic Web Project* in the context menu.
3. Enter **jco_demo** as the project name.
4. In the *Target Runtime* pane, select **Cloud Foundry**. If it is not yet in the list of available runtimes, choose *New Runtime* and select it from there.
5. In the *Configuration* pane, leave the default configuration.
6. Choose *Finish*.

 New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.



Project name:

Project location

Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Cloud Foundry runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

Add project to an EAR

EAR project name:

Working sets

Add project to working sets

Working sets:

Back to [Steps \[page 224\]](#)

Include JCo Dependencies

To use JCo functionality seamlessly at compile time in Eclipse, you must include the JCo dependencies into your web project. Therefore, you must convert it into a maven project.

1. In the *Project Explorer* view, right-click on the project `jco-demo` and choose ► *Configure* ► *Convert to Maven Project*.
2. In the dialog window, leave the default settings unchanged and choose *Finish*.
3. Open the `pom.xml` file and include the following dependency:

```
<dependencies>
  <dependency>
    <groupId>com.sap.cloud</groupId>
    <artifactId>neo-java-web-api</artifactId>
    <version>[3.71.8,4.0.0)</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Back to [Steps \[page 224\]](#)

Create a Sample Servlet

1. From the `jco_demo` project node, choose ► *New* ► *Servlet* in the context menu.
2. Enter `com.sap.demo.jco` as the `<>` and `ConnectivityRFCExampleJava` as the `<Class name>`. Choose *Next*.
3. Choose *Finish* to create the servlet and open it in the Java editor.
4. Replace the entire servlet class to make use of the JCo API. The JCo API is visible by default for cloud applications. You do not need to add it explicitly to the application class path.

« Sample Code

```
package com.sap.demo.jco;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sap.conn.jco.AbapException;
import com.sap.conn.jco.JCoDestination;
import com.sap.conn.jco.JCoDestinationManager;
import com.sap.conn.jco.JCoException;
import com.sap.conn.jco.JCoFunction;
import com.sap.conn.jco.JCoParameterList;
import com.sap.conn.jco.JCoRepository;

/**
 * Sample application that uses the Connectivity
 * service. In particular, it is
 * making use of the capability to invoke a function module in an ABAP
 * system
```

```

* via RFC
*
* Note: The JCo APIs are available under <code>com.sap.conn.jco</code>.
*/
@WebServlet("/ConnectivityRFCExample/*")
public class ConnectivityRFCExample extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        PrintWriter responseWriter = response.getWriter();
        try {
            // access the RFC Destination "JCoDemoSystem"
            JCoDestination destination =
JCoDestinationManager.getDestination("JCoDemoSystem");

            // make an invocation of STFC_CONNECTION in the backend;
            JCoRepository repo = destination.getRepository();
            JCoFunction stfcConnection =
repo.getFunction("STFC_CONNECTION");

            JCoParameterList imports =
stfcConnection.getImportParameterList();
            imports.setValue("REQUTEXT", "SAP BTP Connectivity runs with
JCo");
            stfcConnection.execute(destination);
            JCoParameterList exports =
stfcConnection.getExportParameterList();
            String echotext = exports.getString("ECHOTEXT");
            String resptext = exports.getString("RESPTEXT");
            response.addHeader("Content-type", "text/html");
            responseWriter.println("<html><body>");
            responseWriter.println("<h1>Executed STFC_CONNECTION in system
JCoDemoSystem</h1>");
            responseWriter.println("<p>Export parameter ECHOTEXT of
STFC_CONNECTION:<br>");
            responseWriter.println(echotext);
            responseWriter.println("<p>Export parameter RESPTEXT of
STFC_CONNECTION:<br>");
            responseWriter.println(resptext);
            responseWriter.println("</body></html>");
        } catch (AbapException ae) {
            // just for completeness: As this function module does not
have an exception
            // in its signature, this exception cannot occur. But you
should always
            // take care of AbapExceptions
        } catch (JCoException e) {
            response.addHeader("Content-type", "text/html");
            responseWriter.println("<html><body>");
            responseWriter
                .println("<h1>Exception occurred while executing
STFC_CONNECTION in system JCoDemoSystem</h1>");
            responseWriter.println("<pre>");
            e.printStackTrace(responseWriter);
            responseWriter.println("</pre>");
            responseWriter.println("</body></html>");
        }
    }
}

```

- Save the Java editor and make sure that the project compiles without errors.

Back to [Steps \[page 224\]](#)

Next Steps

- [Create and Bind Service Instances \[page 229\]](#)
- [Deploy the Application \[page 232\]](#)
- [Configure Roles and Trust \[page 234\]](#)
- [Set Up an Application Router \[page 235\]](#)
- [Configure the RFC Destination \[page 239\]](#)
- [Configure the Cloud Connector \[page 240\]](#)
- [Monitoring Your Web Application \[page 244\] \(Optional\)](#)

1.1.4.3.1.1.2 Create and Bind Service Instances

You must create and bind several service instances, before you can use your application.

Procedure

1. Logon to the cloud cockpit and choose your subaccount.
2. Choose the space where you want to deploy your demo application.
3. Choose [Service Marketplace](#) and find these 3 services:
 - [Connectivity](#)
 - [Destination](#)
 - [Authorization & Trust Management \(XSUAA\)](#)

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has a navigation menu with options like Applications, Services (selected), Service Marketplace (highlighted in blue), Service Instances, SAP HANA Cloud, Routes, Security Groups, Events, and Members. The main area is titled "Space: dev - Service Marketplace" and shows a list of services. Three specific services are highlighted with red boxes:

- Connectivity**: Establishes a secure and reliable connectivity between cloud applications and on-premise... (Cloud Foundry)
- Destination**: Provides a secure and reliable access to destination and certificate configurations (Cloud Foundry | Kyma | Kubernetes | Other)
- Authorization & Trust Management**: Manage application authorizations and trust to identity providers. (Cloud Foundry | Kyma | Kubernetes | Other)

4. Create and bind a service instance for each of these services.
 - [Connectivity service \[page 230\]](#)
 - [Destination service \[page 230\]](#)
 - [Authorization & Trust Management \(XSUAA service\) \[page 231\]](#)

Connectivity Service

1. Choose **Connectivity > Create Instance**.
2. Insert an instance name (for example, `connectivity_jco`) and choose *Create Instance*.

New Instance

1 Basic Info 2 Parameters 3 Review

Enter basic info for your service instance

Service: *

Service Plan: *

Instance Name: *

Next > **Create Instance** **Cancel**

Back to [Procedure \[page 229\]](#)

Destination Service

1. Choose **Destination > Create Instance**.

2. Insert an instance name (for example, **destination_jco**) and choose *Create Instance*.

Back to [Procedure \[page 229\]](#)

Authorization & Trust Management (XSUAA Service)

1. Choose *Authorization & Trust Management* *Create Instance*
2. Select **<Service Plan>** **application**.
3. Enter an **<Instance Name>** and choose *Next*.

i Note

The instance name must match the one defined in the manifest file.

4. In the next tab *Parameters*, insert the following as a JSON file:

«, Sample Code

```
{  
    "xsappname" : "jco-demo-p1234",  
    "tenant-mode": "dedicated",  
    "scopes": [  
        {  
            "name": "$XSAPPNAME.all",  
            "description": "all"  
        }  
    ],  
    "role-templates": [  
        {  
            "name": "all",  
            "description": "all",  
            "scope-references": [  
                "$XSAPPNAME.all"  
            ]  
        }  
    ]  
}
```

5. Go to tab *Review* and choose *Create Instance*.

Back to [Procedure \[page 229\]](#)

Next Steps

- [Deploy the Application \[page 232\]](#)
- [Configure Roles and Trust \[page 234\]](#)
- [Set Up an Application Router \[page 235\]](#)
- [Configure the RFC Destination \[page 239\]](#)

- Configure the Cloud Connector [page 240]
- Monitoring Your Web Application [page 244] (Optional)

1.1.4.3.1.1.3 Deploy the Application

Deploy your Cloud Foundry application to call an ABAP function module via RFC.

Prerequisites

You have created and bound the required service instances, see [Create and Bind Service Instances \[page 229\]](#).

Procedure

1. To deploy your Web application, you can use the following two alternative procedures:
 - [Deploying from the Eclipse IDE](#)
 - Deploying from the CLI, see [Developing Java in the Cloud Foundry Environment](#)
2. In the following, we publish it with the CLI.
3. To do this, create a `manifest.yml` file. The key parameter is `USE_JCO: true`, which must be set to include JCo into the buildpack during deployment.

`manifest.yml`

```
---  
applications:  
  - name: jco-demo-p1234  
    buildpacks:  
      - sap_java_buildpack  
    env:  
      USE_JCO: true  
      # This is necessary only if more than one instance is bound  
      xsuaa_connectivity_instance_name: "xsuaa_jco"  
      connectivity_instance_name: "connectivity_jco"  
      destination_instance_name: "destination_jco"  
    services:  
      - xsuaa_jco  
      - connectivity_jco  
      - destination_jco
```

Caution

The client libraries (java-security, spring-xsuaa, and container security api for node.js as of version 3.0.6) have been updated. When using these libraries, setting the parameter `SAP_JWT_TRUST_ACL` has become obsolete. This update comes with a change regarding scopes:

- For a business application A calling an application B, it is now mandatory that application B grants at least one scope to the calling business application A.
- Business application A must accept these granted scopes or authorities as part of the application security descriptor.

You can grant scopes using the `xs-security.json` file.

For more information, see [Application Security Descriptor Configuration Syntax](#), specifically the sections *Referencing the Application* and *Authorities*.

Note

If you have more than one instance of those three services bound to your application, you must specify which one JCo should use with the respective env parameters:

- `xsuaa_connectivity_instance_name`
- `connectivity_instance_name`
- `destination_instance_name`.

4. In Eclipse, right-click on the project and navigate to  .
5. Choose a destination by pressing the [Browse...](#) button next to the `manifest.yml` you created before, for example as `jcodemo.war`.
6. Leave the other default settings unchanged and choose [Finish](#) to export the WAR file.
7. Perform a CLI login via `cf login -a api.cf.eu10.hana.ondemand.com -u <your_email_address>` (password, org and space are prompted after a successful login).
8. Push the application with `cf push -f manifest.yml -p jcodemo.war`.
9. Now, the application should be deployed successfully.

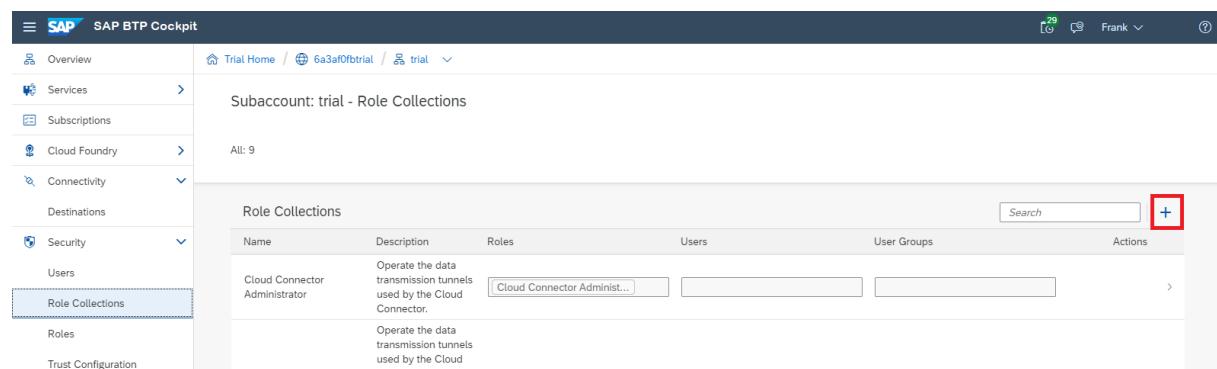
Next Steps

- [Configure Roles and Trust \[page 234\]](#)
- [Set Up an Application Router \[page 235\]](#)
- [Configure the RFC Destination \[page 239\]](#)
- [Configure the Cloud Connector \[page 240\]](#)
- [Monitoring Your Web Application \[page 244\]](#) (Optional)

1.1.4.3.1.1.4 Configure Roles and Trust

Configure a role that enables your user to access your Web application.

To add and assign roles, navigate to the subaccount view of the cloud cockpit and choose ► **Security** ► **Role Collections** ▶.



Name	Description	Roles	Users	User Groups	Actions
Cloud Connector Administrator	Operate the data transmission tunnels used by the Cloud Connector.	[Cloud Connector Administ...]			>
	Operate the data transmission tunnels used by the Cloud				

1. Create a new role collection with the name **all**.
2. From the subaccount menu, choose **Trust Configuration**.
3. If you don't have a trust configuration, follow the steps in [Manually Establish Trust and Federation Between UAA and Identity Authentication](#).
4. Click on the IdP name of your choice.
5. Type in your e-mail address and choose **Show Assignments**.
6. If your user has not yet been added to the SAP ID service, you see following popup. In this case, add your user now.

Confirmation

To see and assign role collections, you must first add <user>@sap.com as a user of identity provider SAP ID Service.

[Add User](#) [Cancel](#)

7. You should now be able to click **Assign Role Collection**. Choose role collection **all** and assign it.

Next Steps

- [Set Up an Application Router \[page 235\]](#)
- [Configure the RFC Destination \[page 239\]](#)
- [Configure the Cloud Connector \[page 240\]](#)
- [Monitoring Your Web Application \[page 244\] \(Optional\)](#)

Related Information

[Working with Role Collections](#)

1.1.4.3.1.1.5 Set Up an Application Router

For authentication purposes, configure and deploy an application router for your test application.

i Note

AppRouter is only required if you want to use multitenancy or perform user-specific service calls. In all other cases, JCo uses [cloud-security-xsuaa-integration](#) with ClientCredentialFlow.

1. To set up an application router, follow the steps in [Application Router](#) or use the demo file [approuter.zip \(download\)](#).
2. For deployment, you need a manifest file, similar to this one:

« Sample Code

```
---
applications:
- name: approuter-jco-demo-p1234
  path: ./.
  buildpacks:
  - nodejs_buildpack
  memory: 120M
  routes:
  - route: approuter-jco-demo-p1234.cfapps.eu10.hana.ondemand.com
  env:
    NODE_TLS_REJECT_UNAUTHORIZED: 0
    destinations: >
      [
        {"name": "dest-to-example", "url": "https://jco-demo-p1234.cfapps.eu10.hana.ondemand.com/ConnectivityRFCExample",
         "forwardAuthToken": true }
      ]
  services:
  - xsuaa_jco
```

i Note

- The routes and destination URLs need to fit your test application.
- In this example, we already bound our XSUAA instance to the application router. Alternatively, you could also do this via the cloud cockpit.

3. Push the approuter with `cf push -f manifest.yml -p approuter.zip`.
4. To navigate to the approuter application in the cloud cockpit, choose [`<your_space>`](#) [`Applications`](#) [`<your_application>`](#) [`Overview`](#).

5. When choosing the application route, you are requested to login. Provide the credentials known by the IdP you configured in [Roles & Trust](#).
6. After successful login, you are routed to the test application which is then executed.
7. If the application issues an exception, saying that the JCoDemoSystem destination has not yet been specified, you must configure the JCoDemoSystem destination first.

```
Exception occurred while executing STFC_CONNECTION in system JCoDemoSystem
com.sap.conn.jco.JCoException: (106) JCÖ_ERROR_RESOURCE: Destination
JCoDemoSystem does not exist
    at
com.sap.conn.jco.rt.DefaultDestinationManager.update(DefaultDestinationManager
.java:223)
    at
com.sap.conn.jco.rt.DefaultDestinationManager.searchDestination(DefaultDestina
tionManager.java:377)
    at
com.sap.conn.jco.rt.DefaultDestinationManager.getDestinationInstance(DefaultDe
stinationManager.java:96)
    at
com.sap.conn.jco.JCoDestinationManager.getDestination(JCoDestinationManager.ja
va:52)
    at
com.sap.demo.jco.ConnectivityRFCExample.doGet(ConnectivityRFCExample.java:47)
..... (cut rest of the call stack)
```

i Note

Make sure you **don't** include this dependency

```
<dependency>
    <groupId>com.sap.cloud.security</groupId>
    <artifactId>java-security</artifactId>
</dependency>
```

or any of its dependencies such as `java-api` with scope `compile` directly or transitively with any other jar.

Calling JCo APIs from Newly Created Threads

If you are using an Application Router and it is mandatory for you to call JCo APIs from a different thread than the one which is executing your servlet function, make sure the thread local information of the [cloud-security-xsuaa-integration API](#), used by JCo internally, is set again within your newly created thread.

To do this, add the following dependency to your project:

```
<dependency>
    <groupId>com.sap.cloud.security</groupId>
    <artifactId>java-api</artifactId>
    <version>2.7.7</version>
    <scope>provided</scope>
</dependency>
```

Adjust your code from the step [Develop a Sample Web Application \[page 224\]](#) in the following way:

Sample Code

```
package com.sap.demo.jco;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sap.cloud.security.token.SecurityContext;
import com.sap.cloud.security.token.Token;
import com.sap.conn.jco.AbapException;
import com.sap.conn.jco.JCoDestination;
import com.sap.conn.jco.JCoDestinationManager;
import com.sap.conn.jco.JCoException;
import com.sap.conn.jco.JCoFunction;
import com.sap.conn.jco.JCoParameterList;
import com.sap.conn.jco.JCoRepository;
/**
 *
 * Sample application that uses the connectivity service. In particular, it is
 * making use of the capability to invoke a function module in an ABAP system
 * via RFC
 *
 * Note: The JCo APIs are available under <code>com.sap.conn.jco</code>.
 */
@WebServlet("/ConnectivityRFCExample/*")
public class ConnectivityRFCExample extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        PrintWriter responseWriter = response.getWriter();
        // access the token from the thread which is executing the servlet
        Token token = SecurityContext.getToken();
        Thread runThread = new Thread(() -> {
            // set the information in the newly created thread
            SecurityContext.setToken(token);
            try {
                // access the RFC Destination "JCoDemoSystem"
                JCoDestination destination =
JCoDestinationManager.getDestination("JCoDemoSystem_Normal");
                // make an invocation of STFC_CONNECTION in the backend
            }
        });
    }
}
```

```

        JCoRepository repo = destination.getRepository();
        JCoFunction stfcConnection =
repo.getFunction("STFC_CONNECTION");
        JCoParameterList imports =
stfcConnection.getImportParameterList();
        imports.setValue("REQUTEXT", "SAP BTP Connectivity runs with
JCo");
        stfcConnection.execute(destination);
        JCoParameterList exports =
stfcConnection.getExportParameterList();
        String echotext = exports.getString("ECHOTEXT");
        String resptext = exports.getString("RESPTEXT");
        response.addHeader("Content-type", "text/html");
        responseWriter.println("<html><body>");
        responseWriter.println("<h1>Executed STFC_CONNECTION in
system JCoDemoSystem</h1>");
        responseWriter.println("<p>Export parameter ECHOTEXT of
STFC_CONNECTION:<br>");
        responseWriter.println(echotext);
        responseWriter.println("<p>Export parameter RESPTEXT of
STFC_CONNECTION:<br>");
        responseWriter.println(resptext);
        responseWriter.println("</body></html>");
    } catch (AbapException ae) {
        // just for completeness: As this function module does not
have an exception
        // in its signature, this exception cannot occur. But you
should always
        // take care of AbapExceptions
    } catch (JCoException e) {
        response.addHeader("Content-type", "text/html");
        responseWriter.println("<html><body>");
        responseWriter
            .println("<h1>Exception occurred while executing
STFC_CONNECTION in system JCoDemoSystem</h1>");
        responseWriter.println("<pre>");
        e.printStackTrace(responseWriter);
        responseWriter.println("</pre>");
        responseWriter.println("</body></html>");
    } finally {
        // after execution clear the token again
        SecurityContext.clearToken();
    }
});
runThread.start();
// wait to be finished
try {
    runThread.join();
} catch (InterruptedException e) {
    e.printStackTrace(responseWriter);
}
}
}

```

i Note

If you want to use [thread pools](#), make sure that in your thread pool implementation this information is set correctly in the thread which is about to be (re)used, and removed as soon as the thread is put back into the pool.

Next Steps

- Configure the RFC Destination [page 239]
- Configure the Cloud Connector [page 240]
- Monitoring Your Web Application [page 244] (Optional)

1.1.4.3.1.1.6 Configure the RFC Destination

Configure an RFC destination on SAP BTP that you can use in your Web application to call the on-premise ABAP system.

To configure the destination, you must use a virtual application server host name (**abapserver.hana.cloud**) and a virtual system number (**42**) that you will expose later in the Cloud Connector. Alternatively, you could use a load balancing configuration with a message server host and a system ID.

1. Create a `.properties` file with the following settings:

```
Name=JCoDemoSystem
Type=RFC
jco.client.ashost=abapserver.hana.cloud
jco.client.sysnr=42
jco.destination.proxy_type=OnPremise
jco.client.user=<DEMOUSER>
jco.client.passwd=<Password>
jco.client.client=000
jco.client.lang=EN
jco.destination.pool_capacity=5
```

2. Go to your subaccount in the cloud cockpit.

1. From the subaccount menu, choose and upload this file.
2. Alternatively, you can create a destination for the service instance **destination_jco** to make it visible only for this instance. To do this, go to and choose *Destinations*.
3. Call again the URL that references the cloud application in the Web browser. The Web application should now return a different exception:

```
Exception occurred while executing STFC_CONNECTION in system JCoDemoSystem
com.sap.conn.jco.JCoException: (102) JCO_ERROR_COMMUNICATION: Opening
connection to backend failed: Opening connection to
abapserver.hana.cloud:sapgw42 denied. Expose the system in your Cloud
Connector in case it was a valid request.
    at
com.sap.conn.jco.rt.MiddlewareJavaRfc.generateJCoException(MiddlewareJavaRfc.j
ava:487)
    at com.sap.conn.jco.rt.MiddlewareJavaRfc
$JavaRfcClient.connect(MiddlewareJavaRfc.java:1216)
    at com.sap.conn.jco.rt.ClientConnection.connect(ClientConnection.java:700)
    at
com.sap.conn.jco.rt.RepositoryConnection.connect(RepositoryConnection.java:72)
    at com.sap.conn.jco.rt.PoolingFactory.init(PoolingFactory.java:113)
    at
com.sap.conn.jco.rt.ConnectionManager.createFactory(ConnectionManager.java:
426)
```

```
at  
com.sap.conn.jco.rt.DefaultConnectionManager.createFactory(DefaultConnectionMa  
nager.java:46)  
at  
com.sap.conn.jco.rt.ConnectionManager.getFactory(ConnectionManager.java:376)  
at com.sap.conn.jco.rt.RfcDestination.getSystemID(RfcDestination.java:  
1101)  
..... (cut rest of the call stack)
```

4. This means that the Cloud Connector denied opening a connection to this system. As a next step, you must configure the system in your installed Cloud Connector.

Next Steps

- [Configure the Cloud Connector \[page 240\]](#)
- [Monitoring Your Web Application \[page 244\]](#) (Optional)

Related Information

[Target System Configuration \[page 117\]](#)

1.1.4.3.1.1.7 Configure the Cloud Connector

Configure the system mapping and the function module in the Cloud Connector.

Steps

1. [Configure Host Mapping \[page 240\]](#)
2. [Configure the Function Module \[page 243\]](#)

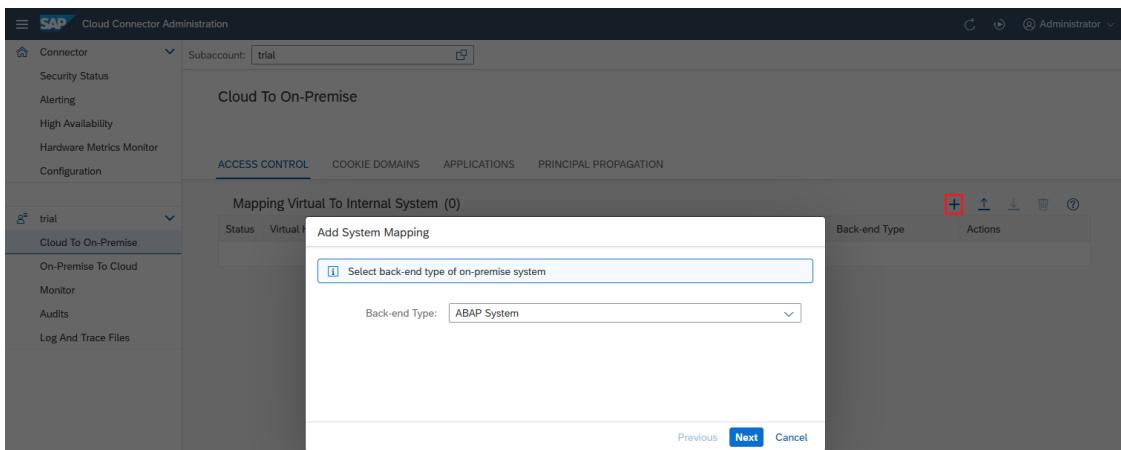
Configure Host Mapping

The Cloud Connector only allows access to trusted backend systems. To configure this, follow the steps below:

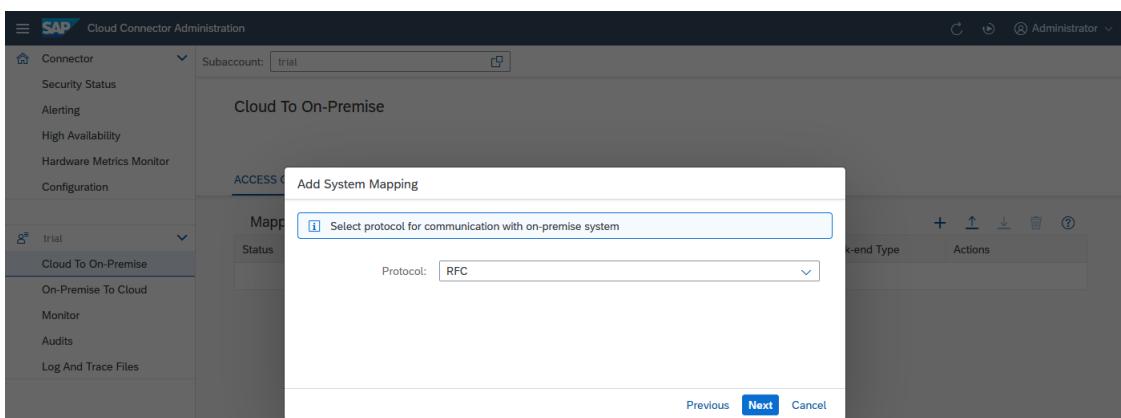
1. Optional: In the Cloud Connector administration UI, you can check under [Audits](#) whether access has been denied:

Denying access for user DEMouser to system abapserver.hana.cloud:sapgw42
[connectionId=-1547299395]

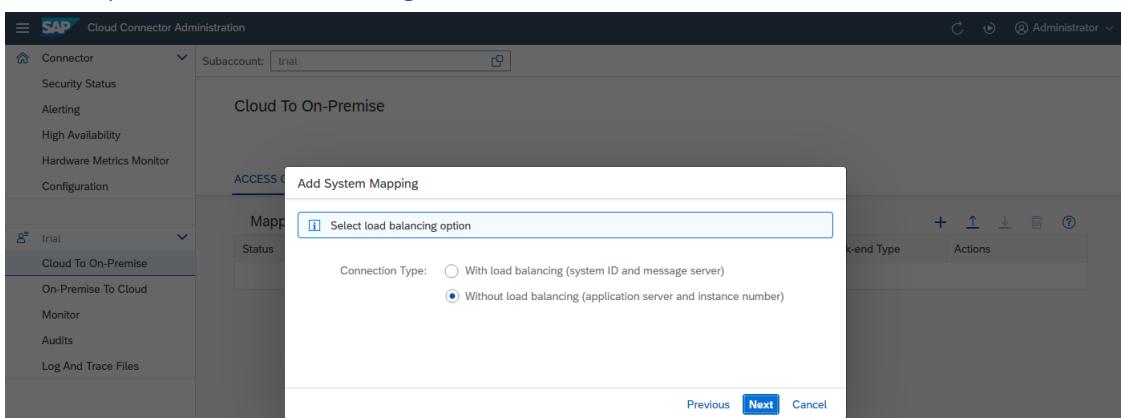
2. In the Cloud Connector administration UI, choose *Cloud To On-Premise* from your *Subaccount* menu, tab *Access Control*.
3. In section *Mapping Virtual To Internal System* choose *Add* to define a new system.
 1. For *Backend Type*, select **ABAP System** and choose *Next*.



2. For *Protocol*, select **RFC** and choose *Next*.

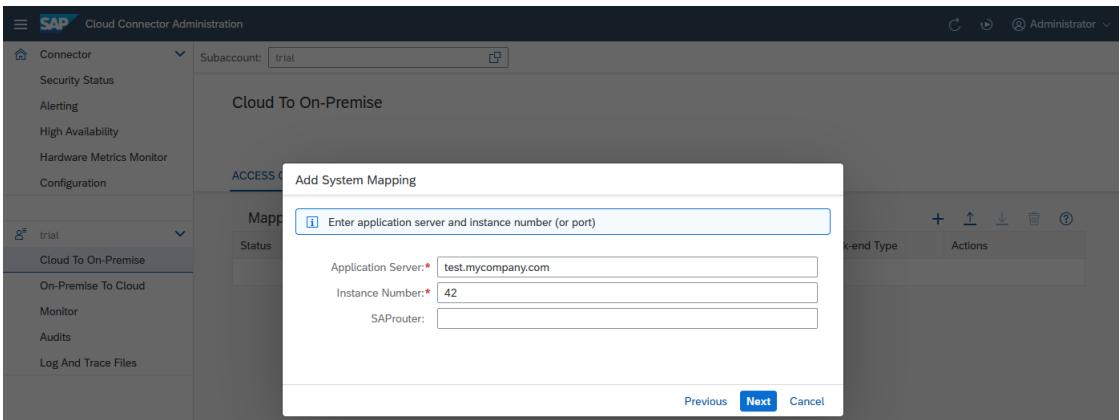


3. Choose option *Without load balancing*.



4. Enter application server and instance number. The *Application Server* entry must be the physical host name of the machine on which the ABAP application server is running. Choose *Next*.

Example:

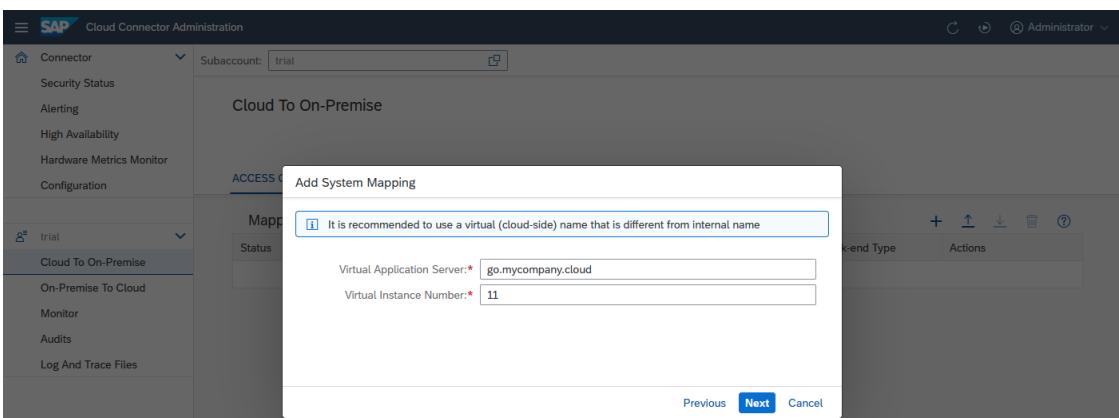


- Enter server and instance number for virtual mapping.

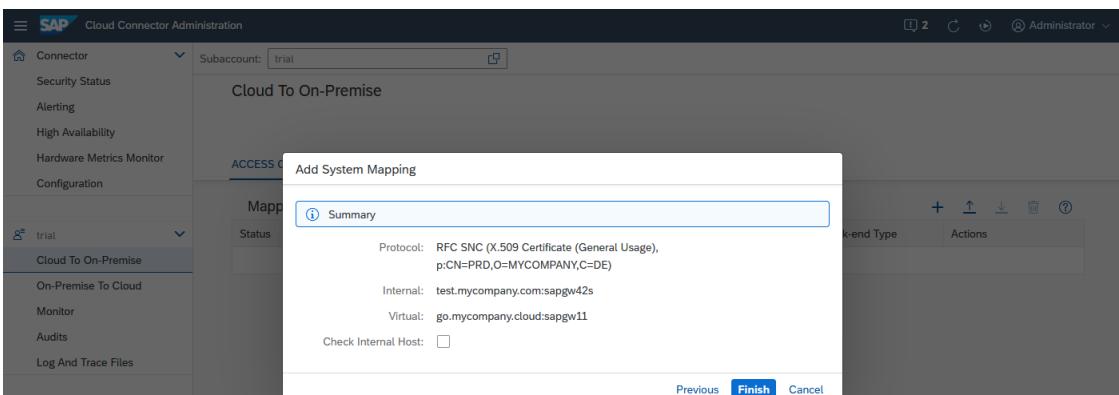
i Note

The values must match with the ones of the destination configuration in the cloud cockpit.

Example:



- Summary (example):



- Call again the URL that references the cloud application in the Web browser. The application should now throw a different exception:

```
om.sap.conn.jco.JCoException: (102) JCO_ERROR_COMMUNICATION: Partner
signaled an error: Access denied for STFC_CONNECTION on
abapserver.hana.cloud:sapgw42. Expose the function module in your Cloud
Connector in case it was a valid request.
```

```

at
com.sap.conn.jco.rt.MiddlewareJavaRfc.generateJCoException(MiddlewareJavaRfc.j
ava:632)
    at com.sap.conn.jco.rt.MiddlewareJavaRfc
$JavaRfcClient.execute(MiddlewareJavaRfc.java:1764)
    at com.sap.conn.jco.rt.ClientConnection.execute(ClientConnection.java:
1110)
    at com.sap.conn.jco.rt.ClientConnection.execute(ClientConnection.java:943)
    at com.sap.conn.jco.rt.RfcDestination.execute(RfcDestination.java:1307)
    at com.sap.conn.jco.rt.RfcDestination.execute(RfcDestination.java:1278)
    at com.sap.conn.jco.rt.AbapFunction.execute(AbapFunction.java:295)
    at
com.sap.demo.jco.ConnectivityRFCExample.doGet(ConnectivityRFCExample.java:55)
    ..... (cut rest of the call stack)

```

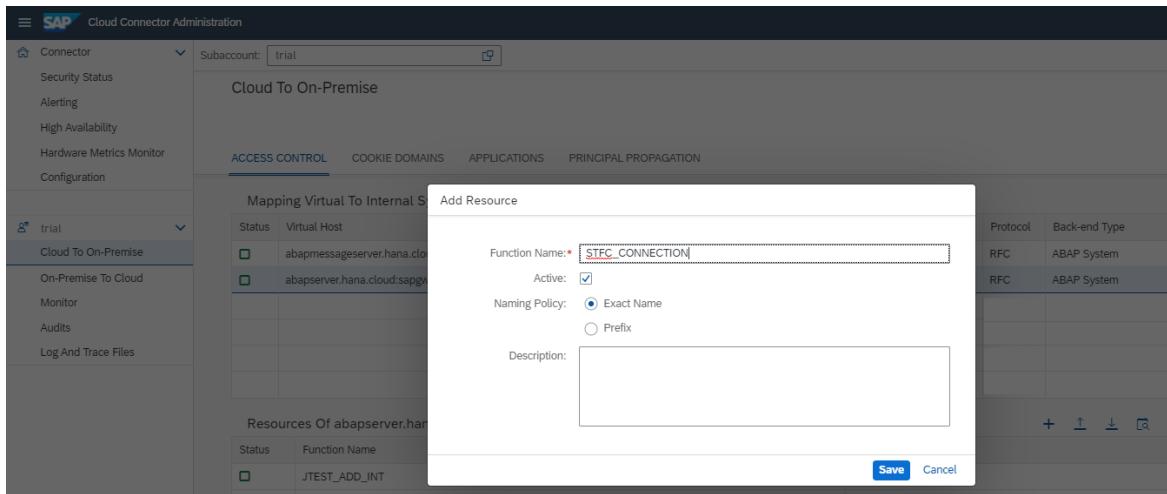
5. This means that the Cloud Connector denied invoking STFC_CONNECTION in this system. As a final step, you must provide access to this function module.

Back to [Steps \[page 240\]](#)

Configure the Function Module

The Cloud Connector only allows access to explicitly allowed resources (which, in an RFC scenario, are defined on the basis of function module names). To configure the function module, follow the steps below:

1. Optional: In the Cloud Connector administration UI, you can check under whether access has been denied:
Denying access for user DEMouser to resource STFC_CONNECTION on system abapserver.hana.cloud:sapgw42 [connectionId=609399452]
2. In the Cloud Connector administration UI, choose again *Cloud To On-Premise* from your *Subaccount* menu, and go to tab *Access Control*.
3. For the specified internal system referring to *abapserver.hana.cloud*, add a new resource. To do this, select the system in the table.
4. Add a new function name under the list of exposed resources. In section *Resources Accessible On abapserver.hana.cloud:sapgw42*, choose the *Add* button and specify STFC_CONNECTION as accessible resource, as shown in the screenshot below. Make sure that you have selected the *Exact Name* option to only expose this specific function module.



- Call again the URL that references the cloud application in the Web browser. The application should now return a message showing the export parameters of the function module.

See also [Configure Access Control \(RFC\) \[page 387\]](#).

[Back to Steps \[page 240\]](#)

Next Step (Optional)

- [Monitoring Your Web Application \[page 244\]](#)

1.1.4.3.1.1.8 Monitoring Your Web Application

Monitor the state and logs of your Web application deployed on SAP BTP, using the Application Logging service.

For this purpose, create an instance of the Application Logging service (as you did for the Destination and Connectivity service) and bind it to your application, see [Create and Bind Service Instances \[page 229\]](#).

To activate JCo logging, set the following property in the `env` section of your manifest file:

```
SET_LOGGING_LEVEL: '{com.sap.core.connectivity.jco: INFO}'
```

Now you can see and open the logs in the cloud cockpit or in the Kibana Dashboard in the tab [Logs](#), if you are within your application.

For detailed information, you can activate the internal JCo logs:

```
SET_LOGGING_LEVEL: '{com.sap.core.connectivity.jco: DEBUG, com.sap.conn.jco:  
DEBUG}'
```

Including other relevant components for logging:

```
SET_LOGGING_LEVEL: '{com.sap.core.connectivity.jco: DEBUG, com.sap.conn.jco:  
DEBUG, com.sap.xs.security: DEBUG, com.sap.cloud.security: DEBUG,  
com.sap.xs.env: DEBUG}'
```

1.1.4.3.1.2 Invoke ABAP Function Modules in Cloud ABAP Systems

Call a function module in a cloud ABAP system via RFC, using a sample Web application (Cloud Foundry environment).

This scenario performs a remote function call to invoke an ABAP function module, by using the Destination service in the Cloud Foundry environment, as well as a Cloud Foundry application router.

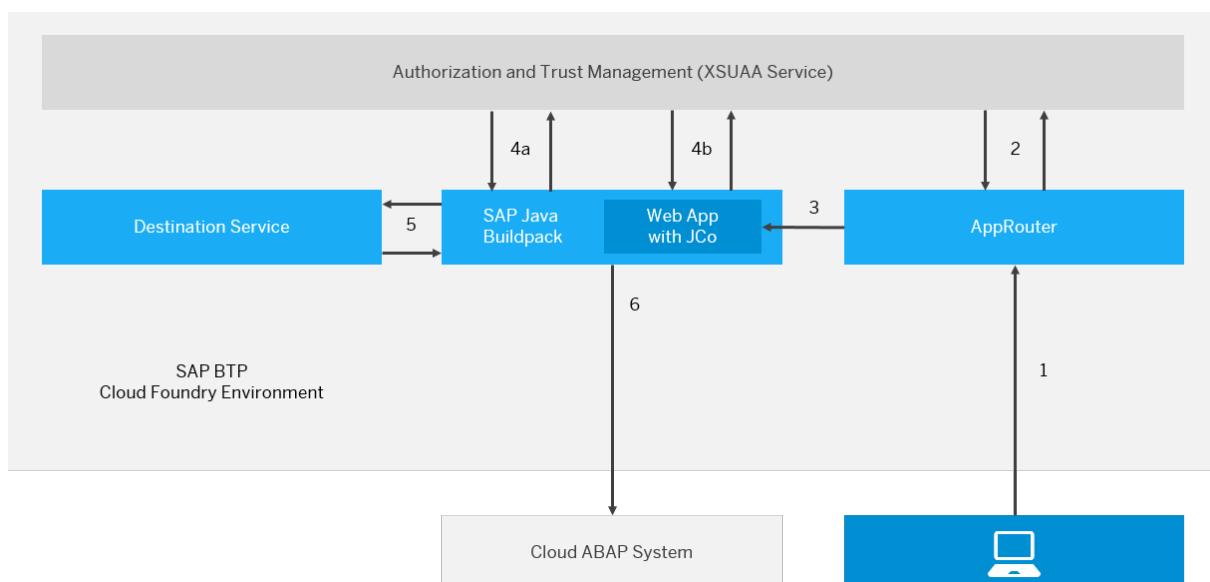
Tasks

Task Type	Task
	Scenario Overview [page 246]
Operator and/or Developer	Used Values [page 247]
	Connectivity User Roles [page 247]
	Installation Prerequisites [page 248]
	Develop a Sample Web Application [page 248]
Developer	
	Create and Bind Service Instances [page 253]
Operator and/or Developer	

Task Type	Task
	Deploy the Application [page 255]
Developer	
	Configure Roles and Trust [page 257]
Operator	
	Set Up an Application Router [page 258]
Operator and/or Developer	
	Configure the RFC Destination [page 262]
Operator	
	Monitoring Your Web Application [page 263] (Optional)
Operator and/or Developer	

Scenario Overview

Control Flow for Using the Java Connector (JCo) with Basic Authentication



Process Steps:

1. Call through AppRouter (entry point for business applications).

i Note

AppRouter is only required if you want to use multitenancy or perform user-specific service calls. In all other cases, JCo uses [cloud-security-xsuaa-integration](#) with ClientCredentialFlow.

2. Redirect to XSUAA for login. JSON Web Token (JWT1) is sent to AppRouter and cached there.
3. AppRouter calls Web app and sends JWT1 with credentials.
4. Buildpack/XSUAA interaction: Buildpack requests JWT2 to access the Destination service instance (JCo call).
5. Buildpack requests destination configuration (JWT2).
6. Buildpack sends request to the cloud ABAP system (with JWT2 and Authorization Header).

Since token exchanges are handled by the buildpack, you must only create and bind the service instances, see [Create and Bind Service Instances \[page 229\]](#).

Back to [Tasks \[page 245\]](#)

Used Values

This scenario uses:

- A subaccount in region **Europe (Frankfurt)**, for which the API endpoint is `api.cf.eu10.hana.ondemand.com`.
- The application name **jco-demo-<subaccount name>**, where `<subaccount name>` is the subdomain name of the subaccount. For this example, we use **p1234**.

Back to [Tasks \[page 245\]](#)

Connectivity User Roles

Different user roles are involved in the cloud-to-cloud connectivity scenario. The particular steps for the relevant roles are described below:

Application Developer

Develops Web applications using destinations. Scenario steps:

1. Installs Eclipse IDE, the Cloud Foundry Plugin for Eclipse and the Cloud Foundry CLI.
2. Develops a Java EE application using the JCo APIs.
3. Configures connectivity destinations via the SAP BTP cockpit.
4. Deploys and tests the Java EE application on SAP BTP.

Account Operator

Deploys Web applications, creates application routers, creates and binds service instances, conducts tests.
Scenario steps:

1. Obtains a ready Java EE application WAR file.
2. Deploys an application router with respective routes to the application.
3. Creates an XSUAA service instance and binds it to the router.
4. Deploys the Java EE application to an SAP BTP subaccount.
5. Creates a Destination service instance, and binds it to the application.
6. Creates and manages roles and role collections.

Back to [Tasks \[page 245\]](#)

Installation Prerequisites

- You have downloaded and set up your Eclipse IDE and the [Eclipse Tools for Cloud Foundry](#).
- You have downloaded the Cloud Foundry CLI, see [Tools](#).

Back to [Tasks \[page 245\]](#)

Next Steps

- [Develop a Sample Web Application \[page 248\]](#)
- [Create and Bind Service Instances \[page 253\]](#)
- [Deploy the Application \[page 255\]](#)
- [Configure Roles and Trust \[page 257\]](#)
- [Set Up an Application Router \[page 258\]](#)
- [Configure the RFC Destination \[page 262\]](#)
- [Monitoring Your Web Application \[page 263\]](#) (Optional)

Related Information

[Multitenancy for JCo Applications \(Advanced\) \[page 263\]](#)

1.1.4.3.1.2.1 Develop a Sample Web Application

Create a Web application to call an ABAP function module via RFC.

Steps

1. Create a Dynamic Web Project [page 225]
2. Include JCo Dependencies [page 226]
3. Create a Sample Servlet [page 227]

Create a Dynamic Web Project

1. Open the *Java EE* perspective of the Eclipse IDE.
2. On the *Project Explorer* view, choose *New* > *Dynamic Web Project* in the context menu.
3. Enter `jco_demo` as the project name.
4. In the *Target Runtime* pane, select `Cloud Foundry`. If it is not yet in the list of available runtimes, choose *New Runtime* and select it from there.
5. In the *Configuration* pane, leave the default configuration.
6. Choose *Finish*.

 New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.



Project name:

Project location

Use default location

Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Cloud Foundry runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

Add project to an EAR

EAR project name:

Working sets

Add project to working sets

Working sets:

[Back to Steps \[page 224\]](#)

Include JCo Dependencies

To use JCo functionality seamlessly at compile time in Eclipse, you must include the JCo dependencies into your web project. Therefore, you must convert it into a maven project.

1. In the *Project Explorer* view, right-click on the project `jco-demo` and choose ► *Configure* ► *Convert to Maven Project*.
2. In the dialog window, leave the default settings unchanged and choose *Finish*.
3. Open the `pom.xml` file and include the following dependency:

```
<dependencies>
  <dependency>
    <groupId>com.sap.cloud</groupId>
    <artifactId>neo-java-web-api</artifactId>
    <version>[3.71.8,4.0.0)</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Back to [Steps \[page 224\]](#)

Create a Sample Servlet

1. From the `jco_demo` project node, choose ► *New* ► *Servlet* in the context menu.
2. Enter `com.sap.demo.jco` as the `<package>` and `ConnectivityRFCExampleJava` as the `<Class name>`. Choose *Next*.
3. Choose *Finish* to create the servlet and open it in the Java editor.
4. Replace the entire servlet class to make use of the JCo API. The JCo API is visible by default for cloud applications. You do not need to add it explicitly to the application class path.

↳ Sample Code

```
package com.sap.demo.jco;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sap.conn.jco.AbapException;
import com.sap.conn.jco.JCoDestination;
import com.sap.conn.jco.JCoDestinationManager;
import com.sap.conn.jco.JCoException;
import com.sap.conn.jco.JCoFunction;
import com.sap.conn.jco.JCoParameterList;
import com.sap.conn.jco.JCoRepository;

/**
 * Sample application that makes use of the capability to invoke a
function module in an ABAP system
 * via RFC
 *
```

```

 * Note: The JCo APIs are available under <code>com.sap.conn.jco</code>.
 */
@WebServlet("/ConnectivityRFCExample/*")
public class ConnectivityRFCExample extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        PrintWriter responseWriter = response.getWriter();
        try {
            // access the RFC Destination "JCoDemoSystem"
            JCoDestination destination =
JCoDestinationManager.getDestination("JCoDemoSystem");

            // make an invocation of STFC_CONNECTION in the backend;
            JCoRepository repo = destination.getRepository();
            JCoFunction stfcConnection =
repo.getFunction("STFC_CONNECTION");

            JCoParameterList imports =
stfcConnection.getImportParameterList();
            imports.setValue("REQUTEXT", "SAP BTP Connectivity runs with
JCo");
            stfcConnection.execute(destination);
            JCoParameterList exports =
stfcConnection.getExportParameterList();
            String echotext = exports.getString("ECHOTEXT");
            String resptext = exports.getString("RESPTEXT");
            response.addHeader("Content-type", "text/html");
            responseWriter.println("<html><body>");
            responseWriter.println("<h1>Executed STFC_CONNECTION in system
JCoDemoSystem</h1>");
            responseWriter.println("<p>Export parameter ECHOTEXT of
STFC_CONNECTION:<br>");
            responseWriter.println(echotext);
            responseWriter.println("<p>Export parameter RESPTEXT of
STFC_CONNECTION:<br>");
            responseWriter.println(resptext);
            responseWriter.println("</body></html>");
        } catch (AbapException ae) {
            // just for completeness: As this function module does not
have an exception
            // in its signature, this exception cannot occur. But you
should always
            // take care of AbapExceptions
        } catch (JCoException e) {
            response.addHeader("Content-type", "text/html");
            responseWriter.println("<html><body>");
            responseWriter
                .println("<h1>Exception occurred while executing
STFC_CONNECTION in system JCoDemoSystem</h1>");
            responseWriter.println("<pre>");
            e.printStackTrace(responseWriter);
            responseWriter.println("</pre>");
            responseWriter.println("</body></html>");
        }
    }
}

```

- Save the Java editor and make sure that the project compiles without errors.

Back to [Steps \[page 224\]](#)

Next Steps

- [Create and Bind Service Instances \[page 253\]](#)
- [Deploy the Application \[page 255\]](#)
- [Configure Roles and Trust \[page 257\]](#)
- [Set Up an Application Router \[page 258\]](#)
- [Configure the RFC Destination \[page 262\]](#)
- [Monitoring Your Web Application \[page 263\] \(Optional\)](#)

1.1.4.3.1.2.2 Create and Bind Service Instances

You must create and bind several service instances, before you can use your application.

Procedure

1. Logon to the cloud cockpit and choose your subaccount.
2. Choose the space where you want to deploy your demo application.
3. Choose [Service Marketplace](#) and find these 2 services:
 - [Destination](#)
 - [Authorization & Trust Management \(XSUAA\)](#)

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has a navigation menu with options like Applications, Services (which is selected), Service Instances, SAP HANA Cloud, Routes, Security Groups, Events, and Members. The main area is titled 'Space: dev - Service Marketplace' and shows a list of services. The services listed are: Application Logging, auditlog-api, Authorization & Trust Management, Connectivity, Destination, and Cloud Management. The 'Authorization & Trust Management' and 'Destination' services are both highlighted with a red box.

4. Create and bind a service instance for each of these services.
 - [Destination service \[page 230\]](#)
 - [Authorization & Trust Management \(XSUAA service\) \[page 231\]](#)

Destination Service

1. Choose *Destination* *Create Instance*
2. Insert an instance name (for example, `destination_jco`) and choose *Create Instance*.

Back to [Procedure \[page 229\]](#)

Authorization & Trust Management (XSUAA Service)

1. Choose *Authorization & Trust Management* *Create Instance*
2. Select `<Service Plan>` `application`.
3. Enter an `<Instance Name>` and choose *Next*.

Note

The instance name must match the one defined in the manifest file.

4. In the next tab *Parameters*, insert the following as a JSON file:

Sample Code

```
{  
    "xsappname" : "jco-demo-p1234",  
    "tenant-mode": "dedicated",  
    "scopes": [  
        {  
            "name": "$XSAPPNAME.all",  
            "description": "all"  
        }  
    ],  
    "role-templates": [  
        {  
            "name": "all",  
            "description": "all",  
            "scope-references": [  
                "$XSAPPNAME.all"  
            ]  
        }  
    ]  
}
```

5. Go to tab *Review* and choose *Create Instance*.

Back to [Procedure \[page 229\]](#)

Next Steps

- [Deploy the Application \[page 255\]](#)
- [Configure Roles and Trust \[page 257\]](#)
- [Set Up an Application Router \[page 258\]](#)
- [Configure the RFC Destination \[page 262\]](#)
- [Monitoring Your Web Application \[page 263\] \(Optional\)](#)

1.1.4.3.1.2.3 Deploy the Application

Deploy your Cloud Foundry application to call an ABAP function module via RFC.

Prerequisites

You have created and bound the required service instances, see [Create and Bind Service Instances \[page 229\]](#).

Procedure

1. To deploy your Web application, you can use the following two alternative procedures:
 - [Deploying from the Eclipse IDE](#)
 - Deploying from the CLI, see [Developing Java in the Cloud Foundry Environment](#)
2. In the following, we publish it with the CLI.
3. To do this, create a `manifest.yml` file. The key parameter is `USE_JCO: true`, which must be set to include JCo into the buildpack during deployment.

manifest.yml

```
↳ Sample Code  
---  
applications:  
- name: jco-demo-p1234  
  buildpacks:  
    - sap_java_buildpack  
  env:  
    USE_JCO: true  
    # This is necessary only if more than one instance is bound  
    xsuaa_connectivity_instance_name: "xsuaa_jco"  
    connectivity_instance_name: "connectivity_jco"  
    destination_instance_name: "destination_jco"  
  services:  
    - xsuaa_jco  
    - connectivity_jco
```

```
- destination_jco
```

⚠ Caution

The client libraries (java-security, spring-xsuaa, and container security api for node.js as of version 3.0.6) have been updated. When using these libraries, setting the parameter `SAP_JWT_TRUST_ACL` has become obsolete. This update comes with a change regarding scopes:

- For a business application A calling an application B, it is now mandatory that application B grants at least one scope to the calling business application A.
- Business application A must accept these granted scopes or authorities as part of the application security descriptor.

You can grant scopes using the `xs-security.json` file.

For more information, see [Application Security Descriptor Configuration Syntax](#), specifically the sections *Referencing the Application* and *Authorities*.

ℹ Note

If you have more than one instance of those three services bound to your application, you must specify which one JCo should use with the respective env parameters:

- `xsuaa_connectivity_instance_name`
- `connectivity_instance_name`
- `destination_instance_name`.

4. In Eclipse, right-click on the project and navigate to [Export](#) [WAR file](#).
5. Choose a destination by pressing the [Browse...](#) button next to the `manifest.yml` you created before, for example as `jcodemo.war`.
6. Leave the other default settings unchanged and choose [Finish](#) to export the WAR file.
7. Perform a CLI login via `cf login -a api.cf.eu10.hana.ondemand.com -u <your_email_address>` (password, org and space are prompted after a successful login).
8. Push the application with `cf push -f manifest.yml -p jcodemo.war`.
9. Now, the application should be deployed successfully.

Next Steps

- [Configure Roles and Trust \[page 257\]](#)
- [Set Up an Application Router \[page 258\]](#)
- [Configure the RFC Destination \[page 262\]](#)
- [Monitoring Your Web Application \[page 263\]](#) (Optional)

1.1.4.3.1.2.4 Configure Roles and Trust

Configure a role that enables your user to access your Web application.

To add and assign roles, navigate to the subaccount view of the cloud cockpit and choose **Security** **Role Collections**.

Name	Description	Roles	Users	User Groups	Actions
Cloud Connector Administrator	Operate the data transmission tunnels used by the Cloud Connector	Cloud Connector Administ...			
	Operate the data transmission tunnels used by the Cloud				

1. Create a new role collection with the name **all**.
2. From the subaccount menu, choose *Trust Configuration*.
3. If you don't have a trust configuration, follow the steps in [Manually Establish Trust and Federation Between UAA and Identity Authentication](#).
4. Click on the IdP name of your choice.
5. Type in your e-mail address and choose *Show Assignments*.
6. If your user has not yet been added to the SAP ID service, you see following popup. In this case, add your user now.

Confirmation

To see and assign role collections, you must first add <user>@sap.com as a user of identity provider SAP ID Service.

[Add User](#) [Cancel](#)

7. You should now be able to click *Assign Role Collection*. Choose role collection **all** and assign it.

Next Steps

- [Set Up an Application Router \[page 258\]](#)
- [Configure the RFC Destination \[page 262\]](#)
- [Monitoring Your Web Application \[page 263\] \(Optional\)](#)

Related Information

[Working with Role Collections](#)

1.1.4.3.1.2.5 Set Up an Application Router

For authentication purposes, configure and deploy an application router for your test application.

i Note

AppRouter is only required if you want to use multitenancy or perform user-specific service calls. In all other cases, JCo uses [cloud-security-xsuaa-integration](#) with ClientCredentialFlow.

1. To set up an application router, follow the steps in [Application Router](#) or use the demo file [approuter.zip \(download\)](#).
2. For deployment, you need a manifest file, similar to this one:

« Sample Code

```
---
applications:
- name: approuter-jco-demo-p1234
  path: ./.
  buildpacks:
    - nodejs_buildpack
  memory: 120M
  routes:
    - route: approuter-jco-demo-p1234.cfapps.eu10.hana.ondemand.com
  env:
    NODE_TLS_REJECT_UNAUTHORIZED: 0
    destinations: >
      [
        {"name": "dest-to-example", "url": "https://jco-demo-p1234.cfapps.eu10.hana.ondemand.com/ConnectivityRFCExample",
         "forwardAuthToken": true }
      ]
  services:
    - xsuaa_jco
```

i Note

- The routes and destination URLs need to fit your test application.
- In this example, we already bound our XSUAA instance to the application router. Alternatively, you could also do this via the cloud cockpit.

3. Push the approuter with `cf push -f manifest.yml -p approuter.zip`.
4. To navigate to the `approuter` application in the cloud cockpit, choose [`<your_space>`](#) [`Applications`](#) [`<your_application>`](#) [`Overview`](#).

The screenshot shows the SAP BTP Cockpit interface. The left sidebar has a navigation menu with items like Overview, Service Bindings, Security, User-Provided Variables, Environment Variables, Events, and Logs. The main content area is titled 'Application: approouter-demo - Overview'. It shows a green 'Started' button and other action buttons: Restart, Start, Stop, + Instance, - Instance, and Delete. Below these buttons, it displays the 'Application Routes' section with the URL 'approouter-demo.cfapps.eu10.hana.ondemand.com'.

5. When choosing the application route, you are requested to login. Provide the credentials known by the IdP you configured in [Roles & Trust](#).
6. After successful login, you are routed to the test application which is then executed.
7. If the application issues an exception, saying that the `JCoDemoSystem` destination has not yet been specified, you must configure the `JCoDemoSystem` destination first.

```
Exception occurred while executing STFC_CONNECTION in system JCoDemoSystem
com.sap.conn.jco.JCoException: (106) JCÖ_ERROR_RESOURCE: Destination
JCoDemoSystem does not exist
    at
com.sap.conn.jco.rt.DefaultDestinationManager.update(DefaultDestinationManager
.java:223)
    at
com.sap.conn.jco.rt.DefaultDestinationManager.searchDestination(DefaultDestina
tionManager.java:377)
    at
com.sap.conn.jco.rt.DefaultDestinationManager.getDestinationInstance(DefaultDe
stinationManager.java:96)
    at
com.sap.conn.jco.JCoDestinationManager.getDestination(JCoDestinationManager.ja
va:52)
    at
com.sap.demo.jco.ConnectivityRFCExample.doGet(ConnectivityRFCExample.java:47)
..... (cut rest of the call stack)
```

i Note

Make sure you **don't** include this dependency

```
<dependency>
  <groupId>com.sap.cloud.security</groupId>
  <artifactId>java-security</artifactId>
</dependency>
```

or any of its dependencies such as `java-api` with scope `compile` directly or transitively with any other jar.

Calling JCo APIs from Newly Created Threads

If you are using an Application Router and it is mandatory for you to call JCo APIs from a different thread than the one which is executing your servlet function, make sure the thread local information of the [cloud-security-xsuaa-integration API](#), used by JCo internally, is set again within your newly created thread.

To do this, add the following dependency to your project:

```
<dependency>
    <groupId>com.sap.cloud.security</groupId>
    <artifactId>java-api</artifactId>
    <version>2.7.7</version>
    <scope>provided</scope>
</dependency>
```

Adjust your code from the step [Develop a Sample Web Application \[page 248\]](#) in the following way:

Sample Code

```
package com.sap.demo.jco;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sap.cloud.security.token.SecurityContext;
import com.sap.cloud.security.token.Token;
import com.sap.conn.jco.AbapException;
import com.sap.conn.jco.JCoDestination;
import com.sap.conn.jco.JCoDestinationManager;
import com.sap.conn.jco.JCoException;
import com.sap.conn.jco.JCoFunction;
import com.sap.conn.jco.JCoParameterList;
import com.sap.conn.jco.JCoRepository;
/**
 *
 * Sample application that uses the connectivity service. In particular, it is
 * making use of the capability to invoke a function module in an ABAP system
 * via RFC
 *
 * Note: The JCo APIs are available under <code>com.sap.conn.jco</code>.
 */
@WebServlet("/ConnectivityRFCExample/*")
public class ConnectivityRFCExample extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        PrintWriter responseWriter = response.getWriter();
        // access the token from the thread which is executing the servlet
        Token token = SecurityContext.getToken();
        Thread runThread = new Thread(() -> {
            // set the information in the newly created thread
            SecurityContext.setToken(token);
            try {
                // access the RFC Destination "JCoDemoSystem"
                JCoDestination destination =
JCoDestinationManager.getDestination("JCoDemoSystem_Normal");
                // make an invocation of STFC_CONNECTION in the backend
            }
        });
    }
}
```

```

        JCoRepository repo = destination.getRepository();
        JCoFunction stfcConnection =
repo.getFunction("STFC_CONNECTION");
        JCoParameterList imports =
stfcConnection.getImportParameterList();
        imports.setValue("REQUTEXT", "SAP BTP Connectivity runs with
JCo");
        stfcConnection.execute(destination);
        JCoParameterList exports =
stfcConnection.getExportParameterList();
        String echotext = exports.getString("ECHOTEXT");
        String resptext = exports.getString("RESPTEXT");
        response.addHeader("Content-type", "text/html");
        responseWriter.println("<html><body>");
        responseWriter.println("<h1>Executed STFC_CONNECTION in
system JCoDemoSystem</h1>");
        responseWriter.println("<p>Export parameter ECHOTEXT of
STFC_CONNECTION:<br>");
        responseWriter.println(echotext);
        responseWriter.println("<p>Export parameter RESPTEXT of
STFC_CONNECTION:<br>");
        responseWriter.println(resptext);
        responseWriter.println("</body></html>");
    } catch (AbapException ae) {
        // just for completeness: As this function module does not
have an exception
        // in its signature, this exception cannot occur. But you
should always
        // take care of AbapExceptions
    } catch (JCoException e) {
        response.addHeader("Content-type", "text/html");
        responseWriter.println("<html><body>");
        responseWriter
            .println("<h1>Exception occurred while executing
STFC_CONNECTION in system JCoDemoSystem</h1>");
        responseWriter.println("<pre>");
        e.printStackTrace(responseWriter);
        responseWriter.println("</pre>");
        responseWriter.println("</body></html>");
    } finally {
        // after execution clear the token again
        SecurityContext.clearToken();
    }
});
runThread.start();
// wait to be finished
try {
    runThread.join();
} catch (InterruptedException e) {
    e.printStackTrace(responseWriter);
}
}
}

```

i Note

If you want to use [thread pools](#), make sure that in your thread pool implementation this information is set correctly in the thread which is about to be (re)used, and removed as soon as the thread is put back into the pool.

Next Steps

- Configure the RFC Destination [page 262]
- Monitoring Your Web Application [page 263] (Optional)

1.1.4.3.1.2.6 Configure the RFC Destination

Configure an RFC destination on SAP BTP that you can use in your Web application to call the cloud ABAP system.

To configure the destination, you must use a WebSocket application server host name (`<id>.abap.eu10.hana.ondemand.com`) and a WebSocket port (443).

1. Create a `.properties` file with the following settings:

```
Name=JCoDemoSystem
Type=RFC
jco.client.wshost= <id>.abap.eu10.hana.ondemand.com
jco.client.wsport=443
jco.client.alias_user=<DEMOUSER>
jco.client.passwd=<Password>
jco.client.client=100
jco.client.lang=EN
jco.destination.pool_capacity=5
jco.destination.proxy_type=Internet
```

2. Go to your subaccount in the cloud cockpit.

1. From the subaccount menu, choose and upload this file.
2. Alternatively, you can create a destination for the service instance `destination_jco` to make it visible only for this instance. To do this, go to and choose `Destinations`.
3. Specify a trust/key store or security information, see [WebSocket Connection \[page 119\]](#).

Next Steps

- Monitoring Your Web Application [page 263] (Optional)

Related Information

[Target System Configuration \[page 117\]](#)

1.1.4.3.1.2.7 Monitoring Your Web Application

Monitor the state and logs of your Web application deployed on SAP BTP, using the Application Logging service.

For this purpose, create an instance of the Application Logging service (as you did for the Destination service) and bind it to your application, see [Create and Bind Service Instances \[page 229\]](#).

To activate JCo logging, set the following property in the `env` section of your manifest file:

```
SET_LOGGING_LEVEL: '{com.sap.core.connectivity.jco: INFO}'
```

Now you can see and open the logs in the cloud cockpit or in the Kibana Dashboard in the tab [Logs](#), if you are within your application.

For detailed information, you can activate the internal JCo logs:

```
SET_LOGGING_LEVEL: '{com.sap.core.connectivity.jco: DEBUG, com.sap.conn.jco: DEBUG}'
```

Including other relevant components for logging:

```
SET_LOGGING_LEVEL: '{com.sap.core.connectivity.jco: DEBUG, com.sap.conn.jco: DEBUG, com.sap.xs.security: DEBUG, com.sap.cloud.security: DEBUG, com.sap.xs.env: DEBUG}'
```

1.1.4.3.1.3 Multitenancy for JCo Applications (Advanced)

Learn about the required steps to make your Cloud Foundry JCo application tenant-aware.

Using this procedure, you can enable the sample JCo application created in [Invoke ABAP Function Modules in On-Premise ABAP Systems \[page 220\]](#) or [Invoke ABAP Function Modules in Cloud ABAP Systems \[page 245\]](#), for multitenancy.

Steps

1. [Prerequisites \[page 264\]](#)
2. [Adjust the Application Router \[page 264\]](#)
3. [Adjust the XSUAA Service Instance and Roles \[page 265\]](#)
4. [Make the Application Subscribable \[page 266\]](#)
5. [Create the SAAS Provisioning Service \[page 271\]](#)

6. Subscribe to the Application [page 272]
7. Create a New Route [page 274]

Prerequisites

- Your runtime environment uses SAP Java Buildpack version 1.9.0 or higher.
- You have successfully completed one of these procedures:
[Invoke ABAP Function Modules in On-Premise ABAP Systems \[page 220\]](#)
[Invoke ABAP Function Modules in Cloud ABAP Systems \[page 245\]](#)
- You have created a second subaccount (in the same global account), that is used to subscribe to your application.

Back to [Steps \[page 263\]](#)

Adjust the Application Router

The application router needs to be tenant-aware with a TENANT_HOST_PATTERN to recognize different tenants from the URL, see [Multitenancy](#). TENANT_HOST_PATTERN should have the following format:
"^(.*) .<application domain>". The application router extracts the token captured by "(.*)" to use it as the subscriber tenant. The manifest file might look like this:

↳ Sample Code

```
manifest.yml
---
applications:
- name: approuter-jco-demo-p42424242
  path: /
  buildpacks:
    - nodejs_buildpack
  memory: 120M
  routes:
    - route: approuter-jco-demo-p42424242.cfapps.eu10.hana.ondemand.com
  env:
    TENANT_HOST_PATTERN: "^(.).*approuter-jco-demo-
p42424242.cfapps.eu10.hana.ondemand.com"
    NODE_TLS_REJECT_UNAUTHORIZED: 0
  destinations:
    [
      {"name": "dest-to-example", "url": "https://jco-demo-
p42424242.cfapps.eu10.hana.ondemand.com/ConnectivityRFCExample",
       "forwardAuthToken": true }
    ]
  services:
    - xsuaa_jco
```

Back to [Steps \[page 263\]](#)

Adjust the XSUAA Service Instance and Roles

To call the XSUAA in a tenant-aware way, you must adjust the configuration JSON file. The tenant mode must now have the value "shared". Also, you must allow calling the previously defined REST APIs (callbacks).

↳ Sample Code

```
{
  "xsappname" : "jco-demo-p42424242",
  "tenant-mode": "shared",
  "scopes": [
    {
      "name": "$XSAPPNAME.Callback",
      "description": "With this scope set, the callbacks for tenant onboarding, offboarding and getDependencies can be called.",
      "grant-as-authority-to-apps": [
        "$XSAPPNAME(application,sap-provisioning,tenant-onboarding)"
      ]
    },
    {
      "name": "$XSAPPNAME.access",
      "description": "app access"
    },
    {
      "name": "uaa.user",
      "description": "uaa.user"
    }
  ],
  "role-templates": [
    {
      "name": "MultitenancyCallbackRoleTemplate",
      "description": "Call callback-services of applications",
      "scope-references": [
        "$XSAPPNAME.Callback"
      ]
    },
    {
      "name": "UAAaccess",
      "description": "UAA user access",
      "scope-references": [
        "uaa.user",
        "$XSAPPNAME.access"
      ]
    }
  ]
}
```

Add Roles

1. In the cloud cockpit, navigate to the subaccount view and go the tab [Role Collections](#) under [Security](#) (see step [Configure Roles and Trust \[page 234\]](#) from the previous procedure).
2. Click on the role collection name.
3. Choose [Add Role](#).
4. In the popup window, select the demo application as <Application Identifier>.

5. For <Role Template> and <Role>, use **MultitenancyCallbackRoleTemplate** and choose **Save**.
6. Choose **Add Role** again.
7. Select the demo application as <Application Identifier>.
8. For **Role Template** and **Role** use **UAAaccess** and choose **Save**.

Back to [Steps \[page 263\]](#)

Make the Application Subscribable

Firstly, in order to make the application subscribable, it must provide at least the following REST APIs:

- [GET dependent services of an application \[page 266\]](#)
- [PUT tenant subscription to an application \[page 269\]](#)

In our sample application, we implement new servlets for each of these APIs.

The following servlets need additional maven dependencies:

Sample Code

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.5</version>
</dependency>
<dependency>
    <groupId>com.unboundid.components</groupId>
    <artifactId>json</artifactId>
    <version>1.0.0</version>
</dependency>
<dependency>
    <groupId>javax.ws.rs</groupId>
    <artifactId>javax.ws.rs-api</artifactId>
    <version>2.1.1</version>
</dependency>
```

GET Dependencies

The current JCo dependencies are the Connectivity and Destination service. Thus, the GET API must return information about these two services:

Sample Code

```
import java.io.IOException;
import java.util.Arrays;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```

import javax.ws.rs.core.MediaType;

import org.json.JSONException;
import org.json.JSONObject;

import com.google.gson.Gson;

@WebServlet("/callback/v1.0/dependencies")
public class GetDependencyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private static final String DESTINATION_SERVICE_NAME = "destination";
    private static final String CONNECTIVITY_SERVICE_NAME = "connectivity";
    private static final String XSAPPNAME_PROPERTY = "xsappname";

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        try {
            DependantServiceDto destinationService =
createLPSDependency(DESTINATION_SERVICE_NAME);
            DependantServiceDto connectivityService =
createLPSDependency(CONNECTIVITY_SERVICE_NAME);
            List<DependantServiceDto> dependenciesList =
Arrays.asList(destinationService, connectivityService);

            response.setStatus(200);
            response.setContentType(MediaType.APPLICATION_JSON);

            String json = new Gson().toJson(dependenciesList);
            response.getWriter().println(json);
        } catch (JSONException e) {
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
e.getMessage());
        }
    }

    private static DependantServiceDto createLPSDependency(String
serviceName) throws JSONException {
        JSONObject credentials =
EnvironmentVariableAccessor.getServiceCredentials(serviceName);
        String xsappname = credentials.getString(XSAPPNAME_PROPERTY);
        return new DependantServiceDto(serviceName, xsappname);
    }
}

```

Find the code of the two helper classes below:

DependantServiceDto.java

```

public class DependantServiceDto {
    private String appName;
    private String appId;
    public DependantServiceDto() {}
    public DependantServiceDto(String appName, String appId) {
        this.appName = appName;
        this.appId = appId;
    }
    public String getAppName() {
        return appName;
    }
    public void setAppName(String appName) {
        this.appName = appName;
    }
    public String getAppId() {

```

```

        return appId;
    }
    public void setAppId(String appId) {
        this.appId = appId;
    }
    @Override public boolean equals(Object o) {
        if (this == o)
            return true;
        if (!(o instanceof DependantServiceDto))
            return false;
        DependantServiceDto that = (DependantServiceDto) o;
        if (!appName.equals(that.appName))
            return false;
        return appId.equals(that.appId);
    }
    @Override public int hashCode() {
        int result = appName.hashCode();
        result = 31 * result + appId.hashCode();
        return result;
    }
}

```

EnvironmentVariableAccessor.java

```

import java.text.MessageFormat;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
/**
 * Methods for extracting configurations from the environment variables
 */
public final class EnvironmentVariableAccessor
{
    public static final String BEARER_WITH_TRAILING_SPACE="Bearer ";
    private static final String VCAP_SERVICES=System.getenv("VCAP_SERVICES");
    private static final String VCAP_SERVICES_CREDENTIALS="credentials";
    private static final String VCAP_SERVICES_NAME="name";
    private static final String
PROP_XSUAA_CONNECTIVITY_INSTANCE_NAME="xsuaa_connectivity_instance_name";
    private static final String DEFAULT_XSUAA_CONNECTIVITY_INSTANCE_NAME="conn-
xsuaa";
    private EnvironmentVariableAccessor()
    {
    }
    /**
     * Returns service credentials for a given service from VCAP_SERVICES
     *
     * @see <a href= "https://docs.run.pivotal.io/devguide/deploy-apps/
environment-variable.html#VCAP-SERVICES">VCAP_SERVICES</a>
     */
    public static JSONObject getServiceCredentials(String serviceName) throws
JSONException
    {
        return new
JSONObject(VCAP_SERVICES).getJSONArray(serviceName).getJSONObject(0).getJSONObject(
VCAP_SERVICES_CREDENTIALS);
    }
    /**
     * Returns service credentials for a given service instance from
VCAP_SERVICES
     *
     * @see <a href= "https://docs.run.pivotal.io/devguide/deploy-apps/
environment-variable.html#VCAP-SERVICES">VCAP_SERVICES</a>
     */
    public static JSONObject getServiceCredentials(String serviceName, String
serviceInstanceName) throws JSONException
}

```

```

    {
        JSONArray jsonarr=new
        JSONObject(VCAP_SERVICES).getJSONArray(serviceName);
        for (int i=0; i<jsonarr.length(); i++)
        {
            JSONObject serviceInstanceObject=jsonarr.getJSONObject(i);
            String
instanceName=serviceInstanceObject.getString(VCAP_SERVICES_NAME);
            if (instanceName.equals(serviceInstanceName))
            {
                return
serviceInstanceObject.getJSONObject(VCAP_SERVICES_CREDENTIALS);
            }
        }
        throw new RuntimeException(MessageFormat.format("Service instance {0} of
service {1} not bound to application", serviceInstanceName, serviceName));
    }
    /**
     * Returns service credentials attribute for a given service from
VCAP_SERVICES
    *
     * @see <a href= "https://docs.run.pivotal.io/devguide/deploy-apps/
environment-variable.html#VCAP-SERVICES">VCAP_SERVICES</a>
    */
    public static String getServiceCredentialsAttribute(String serviceName,
String attributeName) throws JSONException
    {
        return getServiceCredentials(serviceName).getString(attributeName);
    }
    /**
     * Returns the name of the xsuaa service for connectivity service.
    */
    public static String getXsuaaConnectivityInstanceName()
    {
        String
xsuaaConnectivityInstanceName=System.getenv(PROP_XSUAA_CONNECTIVITY_INSTANCE_NAME
);
        return xsuaaConnectivityInstanceName!=null?
xsuaaConnectivityInstanceName:DEFAULT_XSUAA_CONNECTIVITY_INSTANCE_NAME;
    }
}

```

Back to [Make the Application Subscribable \[page 266\]](#)

PUT Tenant Subscription

This API is called whenever a tenant is subscribing. In our example, we just read the received JSON, and return the tenant-aware URL of the application router which points to our application. Also, if a tenant wants to unsubscribe, DELETE does currently nothing.

Sample Code

```

SubscribeServlet
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.gson.Gson;

```

```

@WebServlet("/callback/v1.0/tenants/*")
public class SubscribeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        PayloadDataDto payload = new Gson().fromJson(request.getReader(), PayloadDataDto.class);
        response.getWriter().println("https://" + payload.getSubscribedSubdomain() + ".approuter-jco-
demo.cfapps.eu10.hana.ondemand.com");
    }

    @Override
    protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        super.doDelete(req, resp);
    }
}

```

Here is the helper class:

PayloadDataDto.java

```

import java.util.Map;
public class PayloadDataDto {
    private String subscriptionAppName;
    private String subscriptionAppId;
    private String subscribedTenantId;
    private String subscribedSubdomain;
    private String subscriptionAppPlan;
    private long subscriptionAppAmount;
    private String[] dependantServiceInstanceIdAppIds = null;
    private Map<String, String> additionalInformation;
    public PayloadDataDto() {}
    public PayloadDataDto(String subscriptionAppName, String subscriptionAppId,
                         String subscribedTenantId, String subscribedSubdomain, String
subscriptionAppPlan,
                         Map<String, String> additionalInformation) {
        this.subscriptionAppName = subscriptionAppName;
        this.subscriptionAppId = subscriptionAppId;
        this.subscribedTenantId = subscribedTenantId;
        this.subscribedSubdomain = subscribedSubdomain;
        this.subscriptionAppPlan = subscriptionAppPlan;
        this.additionalInformation = additionalInformation;
    }
    public String getSubscriptionAppName() {
        return subscriptionAppName;
    }
    public void setSubscriptionAppName(String subscriptionAppName) {
        this.subscriptionAppName = subscriptionAppName;
    }
    public String getSubscriptionAppId() {
        return subscriptionAppId;
    }
    public void setSubscriptionAppId(String subscriptionAppId) {
        this.subscriptionAppId = subscriptionAppId;
    }
    public String getSubscribedTenantId() {
        return subscribedTenantId;
    }
    public void setSubscribedTenantId(String subscribedTenantId) {
        this.subscribedTenantId = subscribedTenantId;
    }
}

```

```

    }
    public String getSubscribedSubdomain() {
        return subscribedSubdomain;
    }
    public void setSubscribedSubdomain(String subscribedSubdomain) {
        this.subscribedSubdomain = subscribedSubdomain;
    }
    public String getSubscriptionAppPlan() {
        return subscriptionAppPlan;
    }
    public void setSubscriptionAppPlan(String subscriptionAppPlan) {
        this.subscriptionAppPlan = subscriptionAppPlan;
    }
    public Map<String, String> getAdditionalInformation() {
        return additionalInformation;
    }
    public void setAdditionalInformation(Map<String, String>
additionalInformation) {
        this.additionalInformation = additionalInformation;
    }
    public long getSubscriptionAppAmount() {
        return subscriptionAppAmount;
    }
    public void setSubscriptionAppAmount(long subscriptionAppAmount) {
        this.subscriptionAppAmount = subscriptionAppAmount;
    }
    public String[] getDependantServiceInstanceIdAppIds() {
        return dependantServiceInstanceIdAppIds;
    }
    public void setDependantServiceInstanceIdAppIds(
        String[] dependantServiceInstanceIdAppIds) {
        this.dependantServiceInstanceIdAppIds = dependantServiceInstanceIdAppIds;
    }
    public String toString() {
        return String.format("Payload data: subscriptionAppName=%s,
subscriptionAppId=%s, subscribedTenantId=%s,"
            + "subscribedSubdomain=%s subscriptionAppPlan=%s
subscriptionAppAmount=%s dependantServiceInstanceIdAppIds=%s",
            this.subscriptionAppName, this.subscriptionAppId,
            this.subscribedTenantId, this.subscribedSubdomain,
            this.subscriptionAppPlan, this.getSubscriptionAppAmount(),
            dependantServiceInstanceIdAppIds);
    }
}

```

[Back to Make the Application Subscribable \[page 266\]](#)

[Back to Steps \[page 263\]](#)

Create the SAAS Provisioning Service

For the subscription of other tenants, your application must have a bound SAAS provisioning service instance. You can do this using the cockpit:

1. Go to the [Service Marketplace](#) in the cloud cockpit:

2. Choose **SaaS Provisioning > Instances > New Instance**.
3. Select **application** as **<Service Plan>** and choose **Next**.
4. In the step **Specify Parameters (Optional)**, insert the following as a JSON file:

↳ Sample Code

```
{
  "xsappname" : "jco-demo-p42424242",
  "appName" : "JCo-Demo",
  "appUrls": {
    "getDependencies": "https://jco-demo-
p42424242.cfapps.eu10.hana.ondemand.com/callback/v1.0/dependencies",
    "onSubscription" : "https://jco-demo-
p42424242.cfapps.eu10.hana.ondemand.com/callback/v1.0/tenants/{tenantId}"
  }
}
```

5. Choose **Next**, and select the sample application **jco-demo-p42424242** in the drop-down menu to assign the SAAS service to it.
6. Choose **Next**, insert an instance name, for example, **saas_jco**, and confirm the creation by pressing **Finish**.

Back to [Steps \[page 263\]](#)

Subscribe to the Application

1. To subscribe the new application from a different subaccount, go to **Subscriptions** in the cockpit:

SAP BTP Cockpit

Subaccount: trial2 - Subscriptions

Other

JCo-Demo
Not Subscribed

Go to Application

2. Click on **JCo-Demo**.
3. In the next window, choose **Subscribe**:

SAP BTP Cockpit

Subscription: JCo-Demo - Overview

Not Subscribed

Subscribe

Application Description

JCo-Demo

Go to Application

Availability

4. If the subscription was successful, your window should look like that:

SAP BTP Cockpit

Subscription: JCo-Demo - Overview

Subscribed

Unsubscribe

Application Description

JCo-Demo

Go to Application

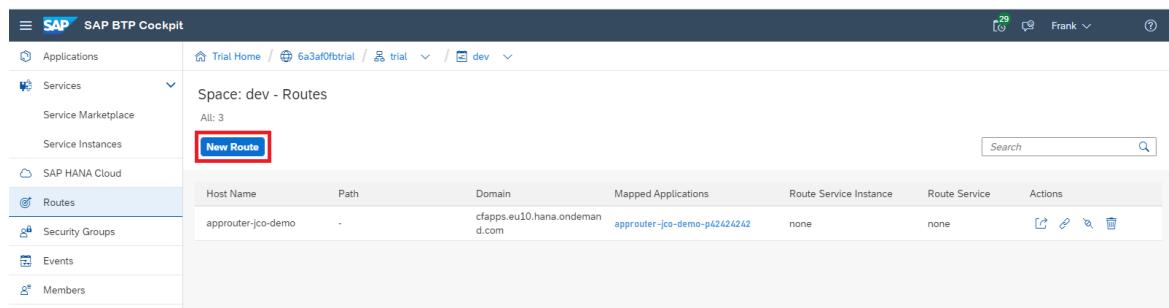
Availability

Back to [Steps \[page 263\]](#)

Create a New Route

1. To call the application with a new tenant, you must create a new route (URL). In the cockpit, choose

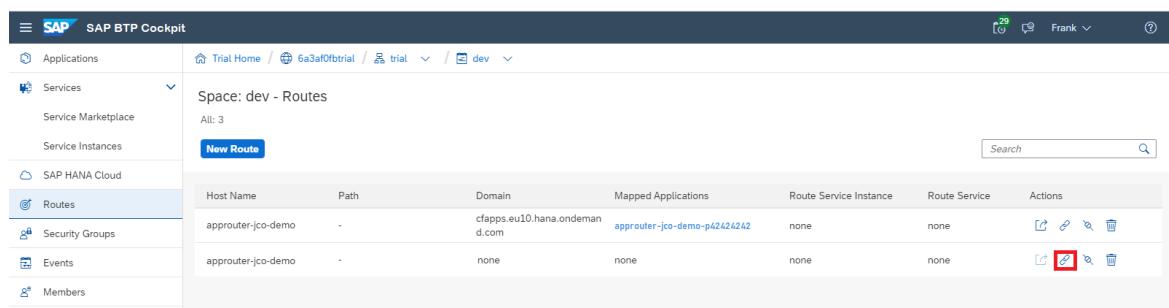
► [Routes](#) ► [New Route](#) ▶:



The screenshot shows the SAP BTP Cockpit interface. The left sidebar has a 'Routes' section highlighted. The main area shows a table of routes for the 'dev' space. A red box highlights the 'New Route' button at the top of the table.

Host Name	Path	Domain	Mapped Applications	Route Service Instance	Route Service	Actions
approuter-jco-demo	-	cfapps.eu10.hana.ondemand.com	approuter-jco-demo-p42424242	none	none	Edit Delete Copy

2. For <Domain>, select the landscape your application is deployed in (e.g. `cfapps.eu10.hana.ondemand.com`).
3. For <Host Name>, choose the tenant-specific link. In our example it would be <tenant-subdomain-name>.approuter-jco-demo-p42424242.
4. Choose [Save](#).
5. Choose [Map Route](#) of the newly created route where the field <Mapped Applications> is empty (value none):



The screenshot shows the SAP BTP Cockpit interface after saving a new route. The 'Routes' menu item is highlighted. A red box highlights the 'Edit' icon for the first route row in the table.

Host Name	Path	Domain	Mapped Applications	Route Service Instance	Route Service	Actions
approuter-jco-demo	-	cfapps.eu10.hana.ondemand.com	approuter-jco-demo-p42424242	none	none	Edit Delete Copy
approuter-jco-demo	-	none	none	none	none	Edit Delete Copy

6. Select the approuter application and choose [Save](#).
7. Congratulations, you are done, now you are able to call the sample application with this newly created route from the other subaccount!

Back to [Steps \[page 263\]](#)

1.1.4.3.1.4 Configure Principal Propagation for RFC

Enable single sign-on (SSO) via RFC by forwarding the identity of cloud users from the Cloud Foundry environment to an on-premise system.

Prerequisites

You have set up the Cloud Connector and the relevant backend system for principal propagation. For more information, see [Configuring Principal Propagation \[page 350\]](#).

Procedure

- Make sure your RFC destination is configured with `jco.destination.auth_type=PrincipalPropagation`. For more information, see [User Logon Properties \[page 112\]](#).
- For your Java application, use an application router to forward a user token which is used by the Java Connector (JCo) for principal propagation. For more information, see [Set Up an Application Router \[page 235\]](#).

1.1.5 Security

Find an overview of recommended security measures for SAP BTP Connectivity.

Topic	More Information
Enable single sign-on by forwarding the identity of cloud users to a remote system or service.	Principal Propagation [page 122]
Set up and run the Cloud Connector according to the highest security standards.	Security Guidelines [page 574]

1.1.6 Monitoring and Troubleshooting

Find information on monitoring and troubleshooting for SAP BTP Connectivity.

Getting Support

If you encounter an issue with this service, we recommend to follow the procedure below:

Check Platform Status

Check the availability of the platform at [SAP BTP Status Page](#).

For more information about selected platform incidents, see [Root Cause Analyses](#).

Check Guided Answers

In the SAP Support Portal, check the [Guided Answers](#) section for SAP BTP. You can find solutions for general SAP BTP issues as well as for specific services there.

Contact SAP Support

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

To find the relevant component for your SAP BTP Connectivity incident, see [Connectivity Support \[page 644\]](#) (section *SAP Support Information*).

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

- Region information (for example: 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

More Information

Topic	More Information
Monitor the Cloud Connector from the SAP BTP cockpit and from the Cloud Connector administration UI.	Monitoring [page 526]
Troubleshoot connection problems and view different types of logs and traces in the Cloud Connector.	Troubleshooting [page 559]
Detailed support information for SAP Connectivity service and the Cloud Connector.	Connectivity Support [page 644]

1.2 Cloud Connector

Learn more about the Cloud Connector: features, scenarios and setup.

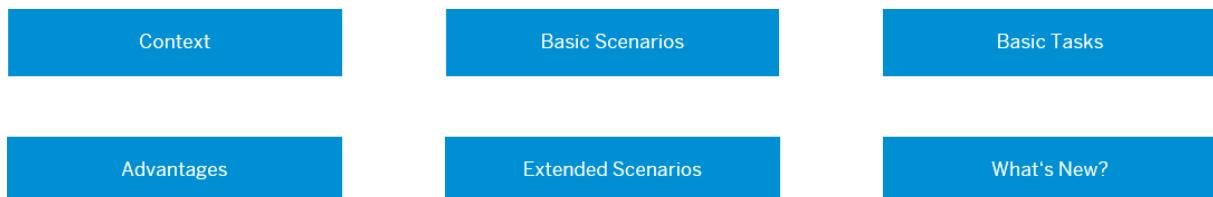
i Note

This documentation refers to SAP BTP, Cloud Foundry environment. If you are looking for information about the Neo environment, see [Connectivity for the Neo Environment](#).

Content

In this Topic

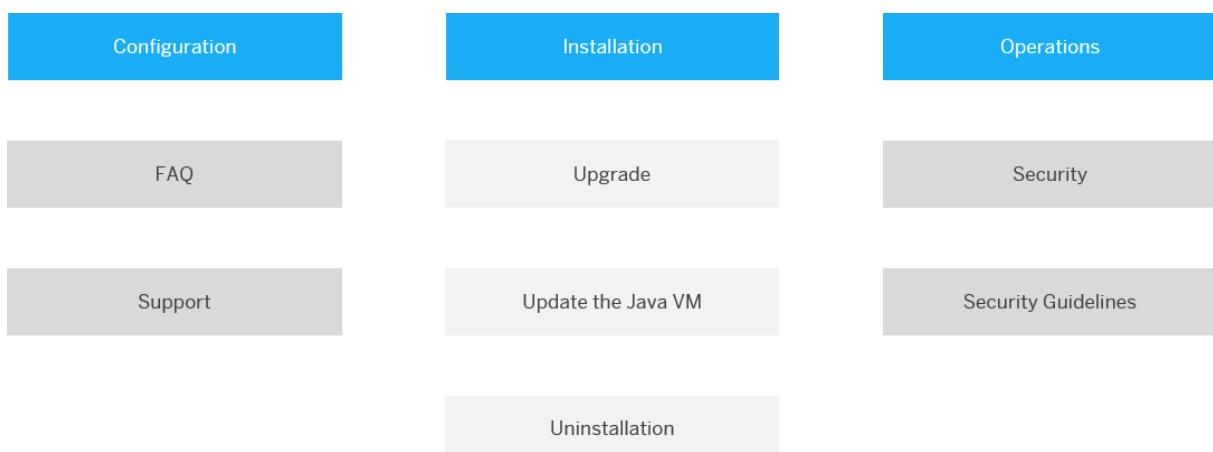
Hover over the elements for a description. Click an element for more information.



- [Context \[page 278\]](#)
- [Basic Scenarios \[page 279\]](#)
- [Basic Tasks \[page 281\]](#)
- [Advantages \[page 278\]](#)
- [Extended Scenarios \[page 281\]](#)
- [What's New? \[page 282\]](#)

In this Guide

Hover over the elements for a description. Click an element for more information.



- [Configuration \[page 321\]](#)
- [Installation \[page 282\]](#)
- [Operations \[page 512\]](#)

- [Frequently Asked Questions \[page 582\]](#)
- [Upgrade \[page 578\]](#)
- [Security \[page 567\]](#)
- [Update the Java VM \[page 580\]](#)
- [Uninstallation \[page 581\]](#)
- [Connectivity Support \[page 644\]](#)
- [Security Guidelines \[page 574\]](#)

Context

The Cloud Connector:

- Serves as a link between SAP BTP applications and on-premise systems.
 - Combines an easy setup with a clear configuration of the systems that are exposed to the SAP BTP.
 - Lets you use existing on-premise assets without exposing the entire internal landscape.
- Runs as on-premise agent in a secured network.
 - Acts as a reverse invoke proxy between the on-premise network and SAP BTP.
- Provides fine-grained control over:
 - On-premise systems and resources that can be accessed by cloud applications.
 - Cloud applications using the Cloud Connector.
- Lets you use the features that are required for business-critical enterprise scenarios.
 - Recovers broken connections automatically.
 - Provides audit logging of inbound traffic and configuration changes.
 - Can be run in a high-availability setup.

⚠ Caution

The Cloud Connector must not be used to connect to products other than SAP BTP or S/4HANA Cloud.

Back to [Content \[page 277\]](#)

Advantages

Compared to the approach of opening ports in the firewall and using reverse proxies in the DMZ to establish access to on-premise systems, the Cloud Connector offers the following benefits:

- You don't need to configure the on-premise firewall to allow external access from SAP BTP to internal systems. For allowed outbound connections, no modifications are required.
- The Cloud Connector supports HTTP as well as additional protocols. For example, the RFC protocol supports native access to ABAP systems by invoking function modules.
- You can use the Cloud Connector to connect on-premise databases or BI tools to SAP HANA databases in the cloud.

- The Cloud Connector lets you propagate the identity of cloud users to on-premise systems in a secure way.
- Easy installation and configuration, which means that the Cloud Connector comes with a low TCO and is tailored to fit your cloud scenarios.
- SAP provides standard support for the Cloud Connector.

[Back to Content \[page 277\]](#)

Basic Scenarios

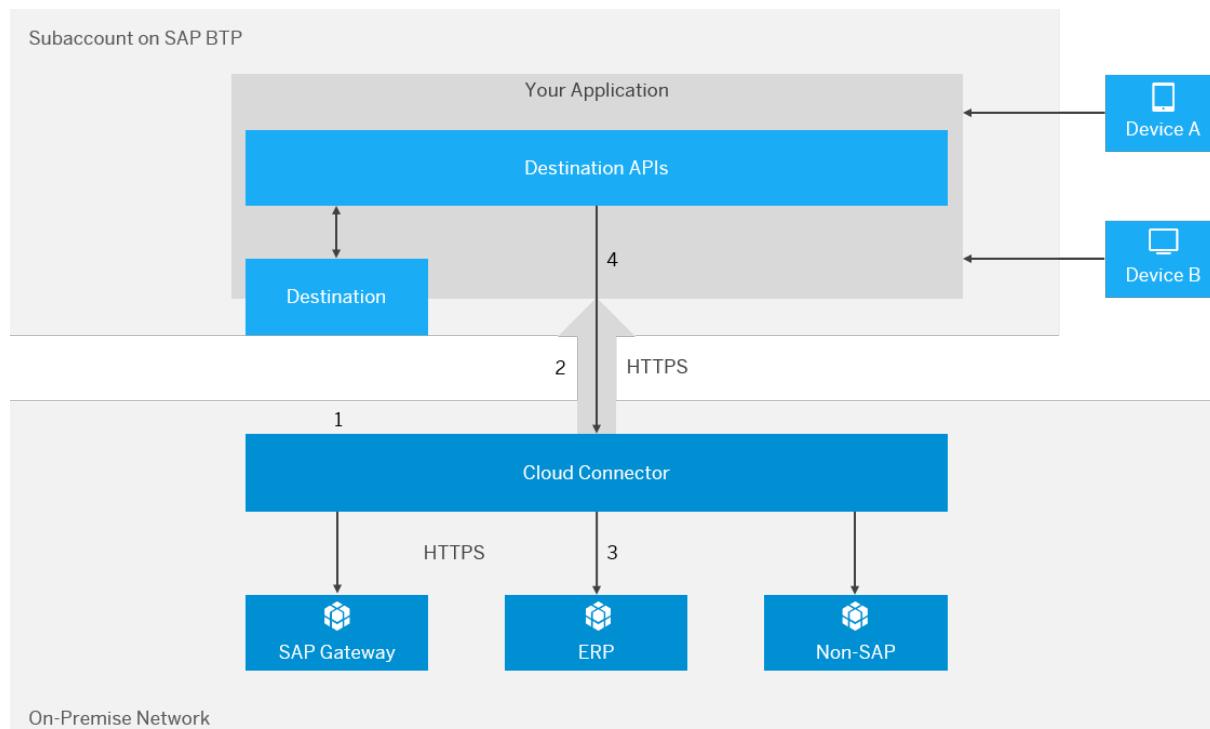
[Connecting Cloud Applications to On-Premise Systems \[page 279\]](#)

[Connecting On-Premise Database Tools to SAP HANA Databases \[page 280\]](#)

i Note

This section refers to the Cloud Connector installation in a standard on-premise network. Find setup options for other system environments in [Extended Scenarios \[page 281\]](#).

Connecting Cloud Applications to On-Premise Systems



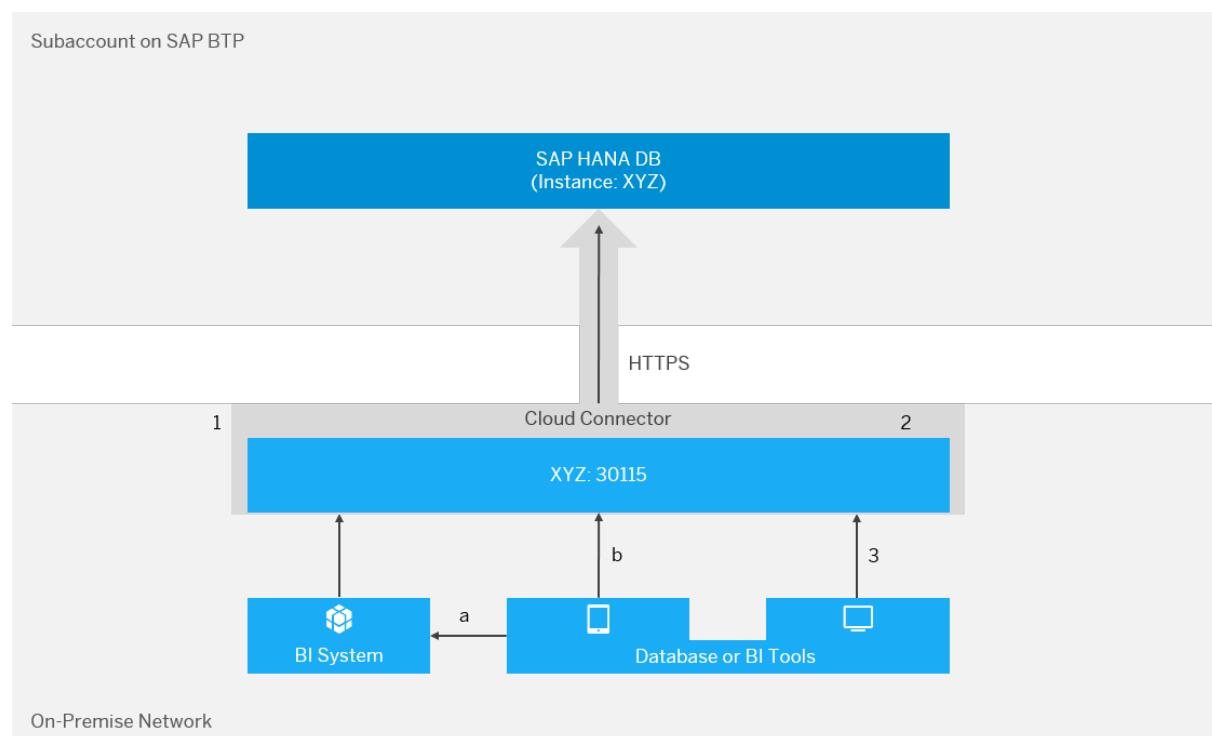
1. Install the Cloud Connector: [Installation \[page 282\]](#)
2. Set up the connection between Cloud Connector, back-end system and your SAP BTP subaccount : [Initial Configuration \[page 322\]](#), [Managing Subaccounts \[page 338\]](#)

3. Allow your cloud application to access a back-end system on the intranet: [Configure Access Control \[page 379\]](#)
4. Connect your cloud application to an on-premise system:
[Consuming the Connectivity Service \[page 165\]](#) (Cloud Foundry environment)

[Back to Basic Scenarios \[page 279\]](#)

[Back to Content \[page 277\]](#)

Connecting On-Premise Database Tools to SAP HANA Databases



1. Install and configure the Cloud Connector: [Installation \[page 282\]](#), [Initial Configuration \[page 322\]](#), [Managing Subaccounts \[page 338\]](#)
2. Access HANA databases on SAP BTP: [Configure a Service Channel for an SAP HANA Database \[page 493\]](#)
3. Connect on-premise database or BI tools to a HANA database on SAP BTP: [Connect DB Tools to SAP HANA via Service Channels \[page 496\]](#)

i Note

You can use service channels also for other purposes:

- Connect to a virtual machine on SAP BTP.
- Configure an RFC connection from your on-premise system to S/4HANA Cloud.

See [Using Service Channels \[page 492\]](#).

[Back to Basic Scenarios \[page 279\]](#)

Back to [Content \[page 277\]](#)

Extended Scenarios

Besides the standard setup: SAP BTP - Cloud Connector - on-premise system/network, you can also use the Cloud Connector to connect SAP BTP applications to other cloud-based environments, as long as they are operated in a way that is comparable to an on-premise network from a functional perspective. This is particularly true for infrastructure (IaaS) hosting solutions.

Here's an overview of all environments in which you can or cannot set up the Cloud Connector:

Cloud Connector	Environment	Examples
Can be set up in:	Customer on-premise network (see Basic Scenarios [page 279])	SAP ERP, SAP S/4HANA
i Note Within extended scenarios that allow a Cloud Connector setup, special procedures may apply for configuration. If so, they are mentioned in the corresponding configuration steps.	SAP Hosting	SAP HANA Enterprise Cloud (HEC)
	Third-party IaaS providers (hosting)	Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP)
Cannot be set up in:	SAP SaaS solutions	SAP SuccessFactors, SAP Concur, SAP Ariba
	SAP cloud-based enterprise solutions	SAP S/4HANA Cloud, SAP C/4HANA
	Third-party PaaS providers	AWS, Azure, GCP
	Third-party SaaS providers	

Back to [Content \[page 277\]](#)

Basic Tasks

The following steps are required to connect the Cloud Connector to your SAP BTP subaccount:

- Install the Cloud Connector: [Installation \[page 282\]](#)
- Perform the initial configuration for the Cloud Connector: [Initial Configuration \[page 322\]](#)
- Register the Cloud Connector for your SAP BTP subaccount: [Managing Subaccounts \[page 338\]](#)

Back to [Content \[page 277\]](#)

What's New?

Follow the SAP BTP [Release Notes](#) to stay informed about Cloud Connector and Connectivity updates.

Back to [Content \[page 277\]](#)

Related Information

[Installation \[page 282\]](#)

[Configuration \[page 321\]](#)

[Operations \[page 512\]](#)

[Security \[page 567\]](#)

[Upgrade \[page 578\]](#)

[Update the Java VM \[page 580\]](#)

[Uninstallation \[page 581\]](#)

[Frequently Asked Questions \[page 582\]](#)

1.2.1 Installation

Choose a procedure to install the Cloud Connector on your operating system.

Portable Version vs. Installer Version

On **Microsoft Windows** and **Linux**, two installation modes are available: a portable version and an installer version. On **Mac OS X**, only the portable version is available.

- Portable version: can be installed easily, by extracting a compressed archive into an empty directory. It does not require administrator or root privileges for the installation, and you can run multiple instances on the same host.

Restrictions:

- You cannot run it in the background as a Windows Service or Linux daemon (with automatic start capabilities at boot time).
- The portable version does not support an automatic upgrade procedure. To update a portable installation, you must delete the current one, extract the new version, and then re-do the configuration.
- Portable versions are meant for non-productive scenarios only.
- The environment variable `JAVA_HOME` is relevant when starting the instance, and therefore must be set properly.

- **Installer** version: requires administrator or root permissions for the installation and can be set up to run as a Windows service or Linux daemon in the background. You can upgrade it easily, retaining all the configuration and customizing.

i Note

We strongly recommend that you use this variant for a productive setup.

Prerequisites

- There are some general prerequisites you must fulfill to successfully install the Cloud Connector, see [Prerequisites \[page 283\]](#).
- For OS-specific requirements and procedures, see section **Tasks** below.

Tasks

- [Installation on Microsoft Windows OS \[page 302\]](#)
- [Installation on Linux OS \[page 305\]](#)
- [Installation on Mac OS X \[page 308\]](#)

Related Information

[Sizing Recommendations \[page 296\]](#)

[Recommendations for Secure Setup \[page 309\]](#)

[\[Deprecated\] Replace the Default SSL Certificate \[page 316\]](#)

[Uninstallation \[page 581\]](#)

1.2.1.1 Prerequisites

Prerequisites for successful installation of the Cloud Connector.

Content

Section	Description
Connectivity Restrictions [page 284]	General information about SAP BTP and connectivity restrictions.
Hardware [page 284]	Hardware prerequisites for a physical or virtual machine.
Software [page 285]	Required software download and installation.
JDKs [page 285]	Java Development Kit (JDK) versions that you can use.
Product Availability Matrix [page 285]	Availability of operating systems/versions for specific Cloud Connector versions.
Network [page 286]	Required Internet connection to SAP BTP hosts per region.

i Note

For additional system requirements, see also [System Requirements \[page 294\]](#).

Connectivity Restrictions

For general information about SAP BTP restrictions, see [Prerequisites and Restrictions](#).

For specific information about all Connectivity restrictions, see [Connectivity: Restrictions \[page 5\]](#).

Back to [Content \[page 283\]](#)

Hardware

Hardware prerequisites, physical or virtual machine:

	Minimum	Recommended
CPU	Single core 3 GHz, x86-64 architecture compatible	Dual core 2 GHz, x86-64 architecture compatible
Memory (RAM)	2 GB	4 GB
Free disk space	3 GB	20 GB

Back to [Content \[page 283\]](#)

Software

- You have downloaded the Cloud Connector installation archive from [SAP Development Tools for Eclipse](#).
- A JDK 8 must be installed. You can download an up-to-date SAP JVM from [SAP Development Tools for Eclipse](#) as well.

⚠ Caution

Do not use [Apache Portable Runtime \(APR\)](#) on the system on which you use the Cloud Connector. If you cannot avoid this restriction and want to use APR at your own risk, you must manually adopt the `server.xml` configuration file in directory `<scc_installation_folder>/conf`. To do so, follow the steps in [HTTPS port configuration](#) for APR.

Back to [Content \[page 283\]](#)

JDKs

JDK	Version	Cloud Connector Version
SAP JVM 64-bit (recommended)	7	2.x up to 2.12.2
	8	2.7.2 and higher
Oracle JDK 64-bit	7	2.x up to 2.12.2
	8	2.7.2 and higher

ℹ Note

The support for using Cloud Connector with Java runtime version 7 ended on December 31, 2019. Any Cloud Connector version released after that date may contain Java byte code requiring at least a JVM 8.

We therefore strongly recommend that you perform **fresh** installations only with **Java 8**, and **update** existing installations running with Java 7, to Java 8.

See [SAP Cloud Connector – Java 7 support will phase out](#) and [Update the Java VM \[page 580\]](#).

Back to [Content \[page 283\]](#)

Product Availability Matrix

Operating System Version	Architecture	Cloud Connector Version
Windows 7, Windows Server 2008 R2	x86_64	2.x

Operating System Version	Architecture	Cloud Connector Version
SUSE Linux Enterprise Server 11, Redhat Enterprise Linux 6	x86_64	2.x
Mac OS X 10.7 (Lion), Mac OS X 10.8 (Mountain Lion)	x86_64	2.x
Windows 8.1, Windows Server 2012, Windows Server 2012 R2	x86_64	2.5.1 and higher
SUSE Linux Enterprise Server 12, Redhat Enterprise Linux 7	x86_64	2.5.1 and higher
Mac OS X 10.9 (Mavericks), Mac OS X 10.10 (Yosemite)	x86_64	2.5.1 and higher
Windows 10	x86_64	2.7.2 and higher
Mac OS X 10.11 (El Capitan)	x86_64	2.8.1 and higher
Windows Server 2016	x86_64	2.9.1 and higher
Windows Server 2019, Mac OS X 10.12 (Sierra), Mac OS X 10.13 (High Sierra), Mac OS X 10.14 (Mojave)	x86_64	2.11.3 and higher
SUSE Linux Enterprise Server 15	x86_64	2.12.0 and higher
Redhat Enterprise Linux 8	x86_64	2.12.2 and higher
SUSE Linux Enterprise Server 12, SUSE Linux Enterprise Server 15, Redhat Enterprise Linux 7, Redhat Enterprise Linux 8	ppc64le	2.13.0 and higher

Back to [Content \[page 283\]](#)

Network

You must have Internet connection at least to the following Connectivity service hosts (depending on the region), to which you can connect your Cloud Connector. All connections to the hosts are TLS-based and connect to port 443.

→ Remember

For some solutions of the BTP portfolio, you must include additional hosts to set up an on-premise connectivity scenario with the Cloud Connector. This applies, for example, to: SAP Data Intelligence, SAP HANA Cloud, and Business Application Studio. Check the respective solution documentation for details.

i Note

For general information on IP ranges per region, see [Regions](#) (Cloud Foundry and ABAP environment) or [Regions and Hosts Available for the Neo Environment](#). Find detailed information about the region status and planned network updates on <http://sapcp.statuspage.io/>.

[Cloud Foundry Environment \[page 287\]](#)

[ABAP Environment \[page 290\]](#)

[Neo Environment \[page 290\]](#)

[Trial \[page 292\]](#)

Region (Region Host)	Hosts	IP Address
Cloud Foundry Environment		
i Note		
	In the Cloud Foundry environment, IPs are controlled by the respective IaaS provider - Amazon Web Services (AWS), Microsoft Azure (Azure), or Google Cloud Platform (GCP). IPs may change due to network updates on the provider side. Any planned changes will be announced at least 4 weeks before they take effect. See also Regions .	
Europe (Frankfurt) - AWS (cf.eu10.hana.ondemand.com)	connectivitynotification.cf.eu10.hana.ondemand.com Enterprise & Trial	3.124.222.77, 3.122.209.241, 3.124.208.223 connectivitycertsigning.cf.eu10.hana.ondemand.com connectivitytunnel.cf.eu10.hana.ondemand.com
		3.124.222.77, 3.122.209.241, 3.124.208.223 3.124.222.77, 3.122.209.241, 3.124.208.223
Europe (Frankfurt) - AWS (cf.eu11.hana.ondemand.com)	connectivitynotification.cf.eu11.hana.ondemand.com Enterprise & Trial	3.124.207.41, 18.157.105.117, 18.156.209.198 connectivitycertsigning.cf.eu11.hana.ondemand.com connectivitytunnel.cf.eu11.hana.ondemand.com
		3.124.207.41, 18.157.105.117, 18.156.209.198 3.124.207.41, 18.157.105.117, 18.156.209.198
Europe (Netherlands) - Azure (cf.eu20.hana.ondemand.com)	connectivitynotification.cf.eu20.hana.ondemand.com Enterprise & Trial	40.119.153.88 connectivitycertsigning.cf.eu20.hana.ondemand.com connectivitytunnel.cf.eu20.hana.ondemand.com
US East (VA) - AWS (cf.us10.hana.ondemand.com)	connectivitynotification.cf.us10.hana.ondemand.com Enterprise & Trial	52.23.189.23, 52.4.101.240, 52.23.1.211 52.23.189.23, 52.4.101.240, 52.23.1.211

Region (Region Host)	Hosts	IP Address
	connectivitytunnel.cf.us10.hana.ondemand.com	52.23.189.23, 52.4.101.240, 52.23.1.211
US West (WA) - Azure	connectivitynotification.cf.us20.hana.ondemand.com	40.91.120.100
(cf.us20.hana.ondema nd.com)	connectivitycertsigning.cf.us20.hana.ondemand.com	40.91.120.100
	connectivitytunnel.cf.us20.hana.ondemand.com	40.91.120.100
US East (VA) - Azure	connectivitynotification.cf.us21.hana.ondemand.com	40.88.52.17
(cf.us21.hana.ondema nd.com)	connectivitycertsigning.cf.us21.hana.ondemand.com	40.88.52.17
	connectivitytunnel.cf.us21.hana.ondemand.com	40.88.52.17
US Central (IA) - GCP	connectivitynotification.cf.us30.hana.ondemand.com	35.184.169.79
(cf.us30.hana.ondema nd.com)	connectivitycertsigning.cf.us30.hana.ondemand.com	35.184.169.79
	connectivitytunnel.cf.us30.hana.ondemand.com	35.184.169.79
Brazil (São Paulo) - AWS	connectivitynotification.cf.br10.hana.ondemand.com	18.229.91.150, 52.67.135.4
(cf.br10.hana.ondema nd.com)	connectivitycertsigning.cf.br10.hana.ondemand.com	18.229.91.150, 52.67.135.4
	connectivitytunnel.cf.br10.hana.ondemand.com	18.229.91.150, 52.67.135.4
Japan (Tokyo) - AWS	connectivitynotification.cf.jp10.hana.ondemand.com	13.114.117.83, 3.114.248.68, 3.113.252.15
(cf.jp10.hana.ondema nd.com)	connectivitycertsigning.cf.jp10.hana.ondemand.com	13.114.117.83, 3.114.248.68, 3.113.252.15
	connectivitytunnel.cf.jp10.hana.ondemand.com	13.114.117.83, 3.114.248.68, 3.113.252.15
Japan (Tokyo) - Azure	connectivitynotification.cf.jp20.hana.ondemand.com	20.43.89.91
(cf.jp20.hana.ondema nd.com)	connectivitycertsigning.cf.jp20.hana.ondemand.com	20.43.89.91
	connectivitytunnel.cf.jp20.hana.ondemand.com	20.43.89.91
Australia (Sydney) - AWS	connectivitynotification.cf.ap10.hana.ondemand.com	13.236.220.84, 13.211.73.244, 3.105.95.184
(cf.ap10.hana.ondema nd.com)	connectivitycertsigning.cf.ap10.hana.ondemand.com	13.236.220.84, 13.211.73.244, 3.105.95.184

Region (Region Host)	Hosts	IP Address
	connectivitytunnel.cf.ap10.hana.ondemand.com	13.236.220.84, 13.211.73.244, 3.105.95.184
Asia Pacific (Singapore) - AWS (cf.ap11.hana.ondema nd.com)	connectivitynotification.cf.ap11.hana.ondemand.com connectivitycertsigning.cf.ap11.hana.ondemand.com	3.0.9.102, 18.140.39.70, 18.139.147.53 3.0.9.102, 18.140.39.70, 18.139.147.53
	connectivitytunnel.cf.ap11.hana.ondemand.com	3.0.9.102, 18.140.39.70, 18.139.147.53
Asia Pacific (Seoul) - AWS (cf.ap12.hana.ondema nd.com)	connectivitynotification.cf.ap12.hana.ondemand.com connectivitycertsigning.cf.ap12.hana.ondemand.com connectivitytunnel.cf.ap12.hana.ondemand.com	3.35.255.45, 3.35.106.215, 3.35.215.12 3.35.255.45, 3.35.106.215, 3.35.215.12 3.35.255.45, 3.35.106.215, 3.35.215.12
Australia (Sydney) - Azure (cf.ap20.hana.ondema nd.com)	connectivitynotification.cf.ap20.hana.ondemand.com connectivitycertsigning.cf.ap20.hana.ondemand.com connectivitytunnel.cf.ap20.hana.ondemand.com	20.53.99.41 20.53.99.41 20.53.99.41
Singapore - Azure (cf.ap21.hana.ondema nd.com)	connectivitynotification.cf.ap21.hana.ondemand.com connectivitycertsigning.cf.ap21.hana.ondemand.com connectivitytunnel.cf.ap21.hana.ondemand.com	20.184.61.122 20.184.61.122 20.184.61.122
<i>Enterprise & Trial</i>		
Canada (Montreal) - AWS (cf.ca10.hana.ondema nd.com)	connectivitynotification.cf.ca10.hana.ondemand.com connectivitycertsigning.cf.ca10.hana.ondemand.com connectivitytunnel.cf.ca10.hana.ondemand.com	35.182.75.101, 35.183.74.34 35.182.75.101, 35.183.74.34 35.182.75.101, 35.183.74.34
China (Shanghai) - Alibaba Cloud (cf.cn40.platform.sa pccloud.cn)	connectivitynotification.cf.cn40.platform.sapcloud.cn connectivitycertsigning.cf.cn40.platform.sapcloud.cn connectivitytunnel.cf.cn40.platform.sapcloud.cn	139.224.7.71 139.224.7.71 139.224.7.71

Region (Region Host)	Hosts	IP Address
Back to Network [page 286]		
Back to Content [page 283]		
Region (Region Host)	Hosts	IP Address
ABAP Environment		
<p>i Note</p> <p>For scenarios using the ABAP environment, include the IPs of the corresponding region below in your firewall rules, in addition to the IPs listed for the same region above (Cloud Foundry environment), if you use IP-based firewall rules.</p>		
Europe (Frankfurt) - AWS	*.abap.eu10.hana.ondemand.com	18.185.129.45, 52.29.97.247
US East (VA) - AWS	*.abap.us10.hana.ondemand.com	52.4.22.116, 52.1.251.254
Japan (Tokyo) - AWS	*.abap.jp10.hana.ondemand.com	18.177.215.21, 3.115.146.41
Back to Network [page 286]		
Back to Content [page 283]		
Region (Region Host)	Hosts	IP Address
Neo Environment		
Europe (Rot)	connectivitynotification.hana.ondemand.com	155.56.210.83
(hana.ondemand.com)	connectivitycertsigning.hana.ondemand.com	155.56.210.43
(eu1.hana.ondemand.com)	connectivitytunnel.hana.ondemand.com	155.56.210.84
Europe (Frankfurt)	connectivitynotification.eu2.hana.ondemand.com	157.133.206.143
(eu2.hana.ondemand.com)	connectivitycertsigning.eu2.hana.ondemand.com	157.133.205.174
	connectivitytunnel.eu2.hana.ondemand.com	157.133.205.233
Europe (Amsterdam)	connectivitynotification.eu3.hana.ondemand.com	157.133.141.140
(eu3.hana.ondemand.com)	connectivitycertsigning.eu3.hana.ondemand.com	157.133.141.132
	connectivitytunnel.eu3.hana.ondemand.com	157.133.141.141
United States East (Ashburn)	connectivitynotification.us1.hana.ondemand.com	Old: 65.221.12.40
(us1.hana.ondemand.com)	connectivitycertsigning.us1.hana.ondemand.com	New: 157.133.18.120
		Old: 65.221.12.241
		New: 157.133.18.101

Region (Region Host)	Hosts	IP Address
i Note Due to a network update, IP addresses for this region change as of May 24, 2020. Please make sure to include also the new IP addresses in your firewall rules if you use IP-based firewall rules.	connectivitytunnel.us1.hana.ondemand.com	Old: 65.221.12.41 New: 157.133.18.121
United States West (Chandler) (us2.hana.ondemand.com)	connectivitynotification.us2.hana.ondemand.com connectivitycertsigning.us2.hana.ondemand.com	Old: 64.95.110.215 New: 157.133.26.25 Old: 64.95.110.211 New: 157.133.26.20
i Note Due to a network update, IP addresses for this region change as of June 14, 2020. Please make sure to include also the new IP addresses in your firewall rules if you use IP-based firewall rules.	connectivitytunnel.us2.hana.ondemand.com	Old: 64.95.110.214 New: 157.133.26.24
United States East (Sterling) (us3.hana.ondemand.com)	connectivitynotification.us3.hana.ondemand.com connectivitycertsigning.us3.hana.ondemand.com connectivitytunnel.us3.hana.ondemand.com	169.145.118.140 169.145.118.132 169.145.118.141
US States West (Colorado Springs) (us4.hana.ondemand.com)	connectivitynotification.us4.hana.ondemand.com connectivitycertsigning.us4.hana.ondemand.com connectivitytunnel.us4.hana.ondemand.com	157.133.45.140 157.133.45.132 157.133.45.141
Australia (Sydney) (ap1.hana.ondemand.com)	connectivitynotification.ap1.hana.ondemand.com connectivitycertsigning.ap1.hana.ondemand.com connectivitytunnel.ap1.hana.ondemand.com	157.133.97.47 157.133.97.27 157.133.97.46
China (Shanghai) (cn1.platform.sapcloud.cn)	connectivitynotification.cn1.platform.sapcloud.cn connectivitycertsigning.cn1.platform.sapcloud.cn connectivitytunnel.cn1.platform.sapcloud.cn	157.133.194.81 157.133.194.77 157.133.194.82
Japan (Tokyo)	connectivitynotification.jp1.hana.ondemand.com	157.133.150.140

Region (Region Host)	Hosts	IP Address
(jp1.hana.ondemand.com)	connectivitycertsigning.jp1.hana.ondemand.com	157.133.150.132
	connectivitytunnel.jp1.hana.ondemand.com	157.133.150.141
Canada (Toronto)	connectivitynotification.ca1.hana.ondemand.com	157.133.54.140
(ca1.hana.ondemand.com)	connectivitycertsigning.ca1.hana.ondemand.com	157.133.54.132
	connectivitytunnel.ca1.hana.ondemand.com	157.133.54.141
Russia (Moscow)	connectivitynotification.ru1.hana.ondemand.com	157.133.2.140
(ru1.hana.ondemand.com)	connectivitycertsigning.ru1.hana.ondemand.com	157.133.2.132
	connectivitytunnel.ru1.hana.ondemand.com	157.133.2.141
Brazil (São Paulo)	connectivitynotification.br1.hana.ondemand.com	157.133.246.140
(br1.hana.ondemand.com)	connectivitycertsigning.br1.hana.ondemand.com	157.133.246.132
	connectivitytunnel.br1.hana.ondemand.com	157.133.246.141
UAE (Dubai)	connectivitynotification.ae1.hana.ondemand.com	157.133.85.140
(ae1.hana.ondemand.com)	connectivitycertsigning.ae1.hana.ondemand.com	157.133.85.132
	connectivitytunnel.ae1.hana.ondemand.com	157.133.85.141
KSA (Riyadh)	connectivitynotification.sa1.hana.ondemand.com	157.133.93.140
(sa1.hana.ondemand.com)	connectivitycertsigning.sa1.hana.ondemand.com	157.133.93.132
	connectivitytunnel.sa1.hana.ondemand.com	157.133.93.141

Back to [Network \[page 286\]](#)

Back to [Content \[page 283\]](#)

Region (Region Host)	Hosts	IP Address
Trial (Cloud Foundry Environment)		
i Note <p>In the Cloud Foundry environment, IPs are controlled by the respective IaaS provider (AWS, Azure, or GCP). IPs may change due to network updates on the provider side. Any planned changes will be announced several weeks before they take effect. See also Regions.</p>		
Europe (Frankfurt) - AWS (cf.eu10.hana.ondemand.com)	connectivitynotification.cf.eu10.hana.ondemand.com	3.124.222.77, 3.122.209.241, 3.124.208.223
	connectivitycertsigning.cf.eu10.hana.ondemand.com	3.124.222.77, 3.122.209.241, 3.124.208.223
	connectivitytunnel.cf.eu10.hana.ondemand.com	3.124.222.77, 3.122.209.241, 3.124.208.223

Region (Region Host)	Hosts	IP Address
United States East (VA) - AWS	connectivitynotification.cf.us10.hana.ondemand.com	52.23.189.23, 52.4.101.240, 52.23.1.211
(cf.us10.hana.ondemand.com)	connectivitycertsigning.cf.us10.hana.ondemand.com	52.23.189.23, 52.4.101.240, 52.23.1.211
	connectivitytunnel.cf.us10.hana.ondemand.com	52.23.189.23, 52.4.101.240, 52.23.1.211
Singapore - Azure	connectivitynotification.cf.ap21.hana.ondemand.com	20.184.61.122
(cf.ap21.hana.ondemand.com)	connectivitycertsigning.cf.ap21.hana.ondemand.com	20.184.61.122
	connectivitytunnel.cf.ap21.hana.ondemand.com	20.184.61.122
Trial (Neo Environment)		
Trial (Europe only)	connectivitynotification.hanatrial.ondemand.com	155.56.219.26
(hanatrial.ondemand.com)	connectivitycertsigning.hanatrial.ondemand.com	155.56.219.22
	connectivitytunnel.hanatrial.ondemand.com	155.56.219.27

Back to [Network \[page 286\]](#)

Back to [Content \[page 283\]](#)

i Note

If you install the Cloud Connector in a network segment that is isolated from the backend systems, make sure the exposed hosts and ports are still reachable and open them in the firewall that protects them:

- for HTTP, the ports you chose for the HTTP/S server.
- for LDAP, the port of the LDAP server.
- for RFC, it depends on whether you use an SAProuter or not and whether load balancing is used:
 - if you use an SAProuter, it is typically configured as visible in the network of the Cloud Connector and the corresponding routtab is exposing all the systems that should be used.
 - without SAProuter, you must open the application server hosts and the corresponding gateway ports (33##, 48##). When using load balancing for the connection, you must also open the message server host and port.

See also [Network Zones \[page 295\]](#).

For more information about the used ABAP server ports, see: [Ports of SAP NetWeaver Application Server ABAP](#).

Back to [Network \[page 286\]](#)

Back to [Content \[page 283\]](#)

Related Information

[Installation on Microsoft Windows OS \[page 302\]](#)

[Installation on Linux OS \[page 305\]](#)

[Installation on Mac OS X \[page 308\]](#)

[Recommendations for Secure Setup \[page 309\]](#)

[Sizing Recommendations \[page 296\]](#)

1.2.1.1 System Requirements

Additional system requirements for installing and running the Cloud Connector.

Supported Browsers

The browsers you can use for the Cloud Connector Administration UI are the same as those currently supported by SAPUI5. See: [Browser and Platform Support](#).

Minimum Disk Space for Download and Installation

The minimum free disk space required to download and install a new Cloud Connector server is as follows:

- Size of downloaded Cloud Connector installation file (ZIP, TAR, MSI files): 50 MB
- Newly installed Cloud Connector server: 70 MB
- Total: 120 MB as a minimum

Additional Disk Space for Log and Configuration Files

The Cloud Connector writes configuration files, audit log files and trace files at runtime. We recommend that you reserve between 1 and 20 GB of disk space for those files.

Trace and log files are written to `<scct_dir>/log/` within the Cloud Connector `root` directory. The `ljs_trace.log` file contains traces in general, communication payload traces are stored in `traffic_trace_*.trc`. These files may be used by SAP Support to analyze potential issues. The default trace level is `Information`, where the amount of written data is generally only a few KB per day. You can turn off these traces to save disk space. However, we recommend that you don't turn off this trace completely, but that you leave it at the default settings, to allow root cause analysis if an issue occurs. If you set the trace level to `All`, the amount of data can easily reach the range of several GB per day. Use trace level `All` only to analyze a specific issue. Payload trace, however, should normally be turned off, and used only for analysis by SAP Support.

i Note

Regularly back up or delete written trace files to clean up the used disk space.

Audit log files are written to `/log/audit/<subaccount-name>/audit-log_<subaccount-name>_<date>.csv` within the Cloud Connector root directory. By default, only security-related events are written in the audit log. You can change the audit log level using the administration UI, as described in [Manage Audit Logs \[page 556\]](#).

To be compliant with the regulatory requirements of your organization and the regional laws, the audit log files must be persisted for a certain period of time for traceability purposes. Therefore, we recommend that you back up the audit log files regularly from the Cloud Connector file system and keep the backup for the length of time required.

Related Information

[Prerequisites \[page 283\]](#)

1.2.1.1.2 Network Zones

Choose a network zone for your Cloud Connector installation.

A customer network is usually divided into multiple network zones or subnetworks according to the security level of the contained components. For example, the DMZ that contains and exposes the external-facing services of an organization to an untrusted network, usually the Internet, and there are one or more other network zones which contain the components and services provided in the company's intranet.

You can generally choose the network zone in which to set up the Cloud Connector:

- Internet access to the SAP BTP region host, either directly or via HTTPS proxy.
- Direct access to the internal systems it provides access to, which means that there is transparent connectivity between the Cloud Connector and the internal system.

The Cloud Connector can be set up either in the DMZ and operated centrally by the IT department, or set up in the intranet and operated by the appropriate line of business.

i Note

The internal network must allow access to the required ports; the specific configuration depends on the firewall software used.

The default ports are **80** for HTTP and **443** for HTTPS. For RFC communication, you need to open a gateway port (default: **33+<instance number>**) and an arbitrary message server port. For a connection to a HANA Database (on SAP BTP) via JDBC, you need to open an arbitrary *outbound* port in your network. Mail (SMTP) communication is not supported.

1.2.1.2 Sizing Recommendations

When installing a Cloud Connector, the first thing you need to decide is the sizing of the installation.

This section gives some basic guidance what to consider for this decision. The provided information includes the shadow instance, which should always be added in productive setups. See also [Install a Failover Instance for High Availability \[page 518\]](#).

i Note

The following recommendations are based on current experiences. However, they are only a rule of thumb since the actual performance strongly depends on the specific environment. The overall performance of a Cloud Connector is impacted by many factors (number of hosted subaccounts, bandwidth, latency to the attached regions, network routers in the corporate network, used JVM, and others).

Restrictions

The sizing data refer to a single Cloud Connector installation.

i Note

Up until now, you cannot perform horizontal scaling directly. However, you can distribute the load statically by operating multiple Cloud Connector installations with different location IDs for all involved subaccounts. In this scenario, you can use multiple destinations with virtually the same configuration, except for the location ID. See also [Managing Subaccounts \[page 338\]](#), step 4. Alternatively, each of the Cloud Connector instances can host its own list of subaccounts without any overlap in the respective lists. Thus, you can handle more load, if a single installation risks to be overloaded.

Related Information

[Hardware Setup \[page 296\]](#)

[Configuration Setup \[page 300\]](#)

1.2.1.2.1 Hardware Setup

How to choose the right sizing for your Cloud Connector installation.

Regarding the hardware, we recommend that you use different setups for master and shadow. One dedicated machine should be used for the master, another one for the shadow. Usually, a shadow instance takes over the master role only temporarily. During most of its lifetime, in the shadow state, it needs less resources compared to the master.

If the master instance is available again after a downtime, we recommend that you switch back to the actual master.

i Note

The sizing recommendations refer to the overall load across all subaccounts that are connected via the Cloud Connector. This means that you need to accumulate the expected load of all subaccounts and should not only calculate separately per subaccount (taking the one with the highest load as basis).

Related Information

[Sizing for the Master Instance \[page 297\]](#)

[Sizing for the Shadow Instance \[page 299\]](#)

1.2.1.2.1.1 Sizing for the Master Instance

Learn more about the basic criteria for the sizing of your Cloud Connector master instance.

For the master setup, keep in mind the expected load for communication between the SAP BTP and on-premise systems. The setups listed below differ in a mostly qualitative manner, without hard limits for each of them.

i Note

The mentioned sizes are considered as minimal configuration, larger ones are always ok. In general, the more applications, application instances, and subaccounts are connected, the more competition will exist for the limited resources on the machine.

Installation Size (S, M, L)	CPU (x86_64)	Machine Memory / Heap /	
		Direct Memory	Disk Space
S:	2 cores 2.6 GHz	4GB RAM / 1GB / 2GB	10GB
	The expected load is small (up to 1 million requests per day, request concurrency and size is in average low) and only a few subaccounts with some applications are connected.		
	In addition, only a few service channels are used, only small data amount is replicated to cloud systems.		
	Setup can be done in a virtual or physical machine.		
M :	4 cores 3.0 Ghz	16GB RAM / 4GB / 8GB	20GB
	The expected load is medium (up to 10 million requests per day, request concurrency and size is in average medium) and multiple subaccounts with multiple applications are connected.		
	In addition, many service channels are used, and a medium data amount is replicated to cloud systems.		
	We recommend that you do the setup in a virtual or physical machine. A virtual machine should be on a host without overcommitted resources.		

Installation Size (S, M, L)	CPU (x86_64)	Machine Memory / Heap / Direct Memory	Disk Space
L: The expected load is large (more than 10 million requests per day, request concurrency and size is in average medium or high) and multiple subaccounts with multiple applications are connected. In addition, many service channels are used, and a large data amount is replicated to cloud systems concurrently. We recommend that you do the setup in a virtual or physical machine. A virtual machine should be on a host without overcommitted resources.	8 cores 3.0 Ghz	32GB RAM / 8GB / 16GB	40GB

Particularly the **heap size** is critical. If you size it too low for the load passing the Cloud Connector, at some point the Java Virtual Machine will execute full GCs (garbage collections) more frequently, blocking the processing of the Cloud Connector completely for multiple seconds, which massively slows down overall performance. If you experience such situations regularly, you should increase the heap size in the Cloud Connector UI (choose   ). See also [Configure the Java VM \[page 507\]](#).

i Note

You should use the same value for both `<initial heap size>` and `<maximum heap size>`.

1.2.1.2.1.2 Sizing for the Shadow Instance

Learn more about the basic criteria for the sizing of your Cloud Connector shadow instance (high availability mode).

The shadow installation is typically not used in standard situations and hence does not need the same sizing, assuming that the time span in which it takes over the master role is limited.

i Note

The shadow only acts as master, for example, during an upgrade or when an abnormal situation occurs on the master machine, and either the Cloud Connector or the full machine on OS level needs to be restarted.

While being in the shadow state, the resource consumption is very low, especially in productive environments, where typically only few configuration changes are required. Therefore, the machine sizing can usually be smaller than the one for the master. However, if you want to mitigate the risk of a longer outage of the master machine, you should increase the sizing of the shadow up to the master size:

Master	Shadow
S	S installation as virtual machine
M	<p>S installation (with double memory) as virtual or physical machine</p> <p>M installation as virtual or physical machine for risk mitigation</p>
L	<p>M installation as virtual or physical machine</p> <p>L installation as virtual or physical machine for risk mitigation</p>

1.2.1.2.2 Configuration Setup

Choose the right connection configuration options to improve the performance of the Cloud Connector.

This section provides detailed information how you can adjust the configuration to improve overall performance. This is typically relevant for an M or L installation (see [Hardware Setup \[page 296\]](#)). For S installations, the default configuration will probably be sufficient to handle the traffic.

To change the connection parameters, proceed as follows:

- As of Cloud Connector 2.11, you can configure the number of physical connections through the Cloud Connector UI. See also [Configure Tunnel Connections \[page 506\]](#).
- In versions prior to 2.11, you have to modify the configuration files with an editor and restart the Cloud Connector to activate the changes.

In general, the Cloud Connector tunnel is multiplexing multiple virtual connections over a single physical connection. Thus, a single connection can handle a considerable amount of traffic. However, increasing the maximum number of physical connections allows you to make use of the full available bandwidth and to minimize latency effects.

If the bandwidth limit of your network is reached, adding additional connections doesn't increase the throughput, but will only consume more resources.

i Note

Different network access parameters may impact and limit your configuration options: if the access to an external network is a 1 MB line with an added latency of 50 ms, you will not be able to achieve the same data volumes like with a 10 GB line with an added latency of < 1 ms. However, even if the line is good, for example 10 GB, but with an added latency of 100 ms, the performance might still be bad.

Optimal configuration strongly depends on your actual scenarios. A good approach is to try out different settings, if the current performance does not meet your expectations.

Related Information

[On-Demand To On-Premise Connections \[page 301\]](#)

[On-Premise To On-Demand Connections \(Service Channels\) \[page 302\]](#)

1.2.1.2.2.1 On-Demand To On-Premise Connections

Configure the physical connections for on-demand to on-premise calls in the Cloud Connector.

Adjusting the number of physical connections for this direction is possible both globally in the Cloud Connector UI (▶ [Configuration](#) ▶ [Advanced](#) ▶), and for individual communication partners on cloud side (▶ [On-Demand To On-Premise](#) ▶ [Applications](#) ▶).

Connections are established for each defined and connected subaccount. The current number of opened connections is visible in the Cloud Connector UI via ▶ [<Subaccount>](#) ▶ [Cloud Connections](#) ▶.

The global default is 1 physical connection per connected subaccount. This value is used across all subaccounts hosted by the Cloud Connector instance and applies for all communication partners.

In general, the default should be sufficient for applications with low traffic. If you expect medium traffic for most applications, it may be useful to set the default value to 2.

i Note

An exact traffic forecast is difficult to achieve. It requires a deep understanding of the use case and of the possible future load generated by different applications. For this reason, we recommend that you focus on subsequent configuration adjustments, using the Cloud Connector monitoring tools to recognize bottlenecks in time, and adjust Cloud Connector configuration accordingly.

Tunnel Worker Threads

In addition to the number of connections, you can configure the number of [`<Tunnel Worker Threads>`](#). This value should be at least equal to the maximum of all individual application tunnel connections in all subaccounts, to have at least 1 thread available for each connection that can process incoming requests and outgoing responses.

Protocol Processor Worker Threads

The value for `<Protocol Processor Worker Threads>` is mainly relevant if RFC is used as protocol. Since its communication model towards the ABAP system is a blocking one, each thread can handle only one call at a time and cannot be shared. Hence, you should provide 1 thread per 5 concurrent RFC requests.

i Note

The longer the RFC execution time in the backend, the more threads you should provide. Threads can be reused only after the response of a call was returned to SAP BTP.

1.2.1.2.2 On-Premise To On-Demand Connections (Service Channels)

Configure the number of physical connections for a Cloud Connector service channel.

Service channels let you configure the number of physical connections to the communication partner on cloud side, see [Using Service Channels \[page 492\]](#). The default is 1. This value is used as well in versions prior to Cloud Connector 2.11, which did not offer a configuration option for each service channel. You should define the number of connections depending on the expected number of clients and, with lower priority, depending on the size of the exchanged messages.

If there is only a single RFC client for an S/4HANA Cloud channel or only a single HANA client for a HANA DB on SAP BTP side, increasing the number doesn't help, as each virtual connection is assigned to one physical connection. The following simple rule lets you to define the required number of connections per service channel:

- Per 10 concurrent clients, use one physical connection.
- If the transferred net data size is larger than 500k per request, make sure to add an additional connection per 2 of such clients.

Example

For a HANA system in the SAP BTP, data is replicated using 18 concurrent clients in the on-premise network. In average, about 5 of those clients are regularly sending 600k. For the number of clients, you should use 2 physical connections, for the 5 clients sending larger amounts add an additional 3, which sums up to 5 connections.

1.2.1.3 Installation on Microsoft Windows OS

Installing the Cloud Connector on a Microsoft Windows operating system.

Context

You can choose between a simple portable variant of the Cloud Connector and the MSI-based installer. The `installer` is the generally recommended version that you can use for both developer and productive scenarios. It lets you, for example, register the Cloud Connector as a Windows service and this way automatically start it after machine reboot.

→ Tip

If you are a developer, you might want to use the `portable` variant as you can run the Cloud Connector after a simple unzip (archive extraction). You might want to use it also if you cannot perform a full installation due to lack of permissions, or if you want to use multiple versions of the Cloud Connector simultaneously on the same machine.

Prerequisites

- You have either of the following 64-bit operating systems: Windows 7, Windows 8.1, Windows 10, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows Server 2016, or Windows Server 2019.
- You have downloaded either the `portable` variant as ZIP archive for Windows, or the MSI `installer` from the [SAP Development Tools for Eclipse](#) page.
- You must install Microsoft Visual Studio C++ 2013 runtime libraries (`vcredist_x64.exe`). For more information, see [Visual C++ Redistributable Packages for Visual Studio 2013](#).

i Note

Even if you have a more recent version of the Microsoft Visual C++ runtime libraries, you still must install the Microsoft Visual Studio C++ 2013 libraries.

- Java 8 must be installed. In case you want to use SAP JVM, you can download it from the [SAP Development Tools for Eclipse](#) page.
- When using the `portable` variant, the environment variable `<JAVA_HOME>` must be set to the Java installation directory, so that the `bin` subdirectory can be found. Alternatively, you can add the relevant `bin` subdirectory to the `<PATH>` variable.

Portable Scenario

1. Extract the `<sapcc-<version>-windows-x64.zip>` ZIP file to an arbitrary directory on your local file system.
2. Set the environment variable `JAVA_HOME` to the installation directory of the JDK that you want to use to run the Cloud Connector. Alternatively, you can add the `bin` subdirectory of the JDK installation directory to the `PATH` environment variable.

3. Go to the Cloud Connector installation directory and start it using the `go.bat` batch file.
4. Continue with the **Next Steps** section.

i Note

The Cloud Connector is not started as a service when using the portable variant, and hence will not automatically start after a reboot of your system. Also, the portable version does not support the automatic upgrade procedure.

Installer Scenario

1. Start the `<sapcc-<version>-windows-x64.msi` installer by double-clicking it.
2. The installer informs you that you are now guided through the installation process. Choose **Next>**.
3. Navigate to the desired installation directory for your Cloud Connector and choose **Next>**. When doing the installation in the context of an upgrade, make sure you choose the previous installation directory again.
4. You can choose the port on which the administration UI is reachable. Either leave the default **8443** or choose a different port if needed. Then, choose **Next>**.
5. Select the JDK to be used for running the Cloud Connector. The installer displays a list of all usable JDKs that are installed on your machine. If the needed JDK is not listed in the drop-down box (for example, if it's an SAP JVM that is not registered in the Windows registry upon installation), you can browse to its installation directory and select it. We recommend that you use an up-to-date Java **8** installation to run the Cloud Connector.
6. Decide whether the Cloud Connector should be started immediately after finishing the setup. Then, choose **Next>**.
7. To start the installation, press the **Next>** button again.
8. After successful installation, choose **Close**.
9. Continue with the **Next Steps** section.

i Note

The Cloud Connector is started as a Windows service in the productive use case. Therefore, installation requires administration permissions. After installation, manage this service under  **Control Panel**  **Administrative Tools**  **Services**. The service name is `Cloud Connector` (formerly named `Cloud Connector 2.0`). Make sure the service is executed with a user that has limited privileges. Typically, privileges allowed for service users are defined by your company policy. Adjust the folder and file permissions to be manageable by only this user and system administrators.

On Windows, the file `scc_service.log` is created and used by the Microsoft MSI installer (during Cloud Connector installation), and by the `scchost.exe` executable, which registers and runs the Windows service if you install the Cloud Connector as a Windows background job.

This log file is only needed if a problem occurs during Cloud Connector installation, or during creation and start of the Windows service, in which the Cloud Connector is running. You can find the file in the `log` folder of your Cloud Connector installation directory.

Starting the Cloud Connector

After installation, the Cloud Connector is registered as a Windows service that is configured to be started automatically after a system reboot. You can start and stop the service via shortcuts on the desktop ("Start Cloud Connector" and "Stop Cloud Connector"), or by using the Windows Services manager and look for the service **SAP Cloud Connector**.

Access the Cloud Connector administration UI at <https://localhost:<port>>, where the default port is **8443** (but this port might have been modified during the installation).

Next Steps

1. Open a browser and enter: <https://<hostname>:8443>. <hostname> is the host name of the machine on which you have installed the Cloud Connector. If you access the Cloud Connector locally from the same machine, you can simply enter **localhost**.
2. Continue with the initial configuration of the Cloud Connector, see [Initial Configuration \[page 322\]](#).

Related Information

(Optional) Install SAP JVM

[Recommendations for Secure Setup \[page 309\]](#)

[\[Deprecated\] Replace the Default SSL Certificate \[page 316\]](#)

1.2.1.4 Installation on Linux OS

Installing the Cloud Connector on a Linux operating system.

Context

You can choose between a simple portable variant of the Cloud Connector and the RPM-based `installer`. The `installer` is the generally recommended version that you can use for both the developer and the productive scenario. It registers, for example, the Cloud Connector as a daemon service and this way automatically starts it after machine reboot.

→ Tip

If you are a developer, you might want to use the portable variant as you can run the Cloud Connector after a simple "`tar -xzof`" execution. You also might want to use it if you cannot perform a full installation

due to missing permissions for the operating system, or if you want to use multiple versions of the Cloud Connector simultaneously on the same machine.

Prerequisites

- You have either of the following 64-bit operating systems: SUSE Linux Enterprise Server 11, 12, or 15, or Redhat Enterprise Linux 6, 7, or 8.
- The supported platforms are `x64` and `ppc64le`, represented below by the variable `<platform>`. Variable `<arch>` is `x86_64` or `ppc64le` respectively.
- You have downloaded either the portable variant as `.tar.gz` archive for Linux or the RPM installer contained in the ZIP for Linux, from [SAP Development Tools for Eclipse](#).
- Java 8 must be installed. If you want to use SAP JVM, you can download an up-to-date version from [SAP Development Tools for Eclipse](#) as well. Use the following command to install it:

```
rpm -i sapjvm-<version>-linux-<platform>.rpm
```

If you want to check the JVM version installed on your system, use the following command:

```
rpm -qa | grep jvm
```

When installing it using the RPM package, the Cloud Connector will detect it and use it for its runtime.

- When using the `.tar.gz` archive, the environment variable `<JAVA_HOME>` must be set to the Java installation directory, so that the `bin` subdirectory can be found. Alternatively, you can add the Java installation's `bin` subdirectory to the `<PATH>` variable.

Portable Scenario

1. Extract the `.tar.gz` file to an arbitrary directory on your local file system using the following command:

```
tar -xzof sapcc-<version>-linux-<platform>.tar.gz
```

i Note

If you use the parameter "o", the extracted files are assigned to the user ID and the group ID of the user who has unpacked the archive. This is the default behavior for users other than the `root` user.

2. Go to this directory and start the Cloud Connector using the `go.sh` script.
3. Continue with the [Next Steps](#) section.

i Note

In this case, the Cloud Connector is not started as a daemon, and therefore will not automatically start after a reboot of your system. Also, the portable version does not support the automatic upgrade procedure.

Installer Scenario

1. Extract the `sapcc-<version>-linux-<platform>.zip` archive to an arbitrary directory by using the following command:

```
unzip sapcc-<version>-linux-<platform>.zip
```

2. Go to this directory and install the extracted RPM using the following command. You can perform this step only as a root user.

```
rpm -i com.sap.scc-ui-<version>.<arch>.rpm
```

3. Continue with the **Next Steps** section.

In the productive case, the Cloud Connector is started as a daemon. If you need to manage the daemon process, execute:

```
System V init distributions: service scc_daemon stop|restart|start|status  
systemd distributions: systemctl stop|restart|start|status scc_daemon
```

⚠ Caution

When adjusting the Cloud Connector installation (for example, restoring a backup), make sure the RPM package management is synchronized with such changes. If you simply replace files that do not fit to the information stored in the package management, lifecycle operations (such as upgrade or uninstallation) might fail with errors. Also, the Cloud Connector might get into unrecoverable state.

Example: After a file system restore, the system files represent Cloud Connector 2.3.0 but the RPM package management "believes" that version 2.4.3 is installed. In this case, commands like `rpm -U` and `rpm -E` do not work as expected. Furthermore, avoid using the `--force` parameter as it may lead to an unpredictable state with two versions being installed concurrently, which is not supported.

Extending the Daemon (as of Cloud Connector version 2.12.3)

When using SNC for encrypting RFC communication, it might be required to provide some settings, for example, environment variables that must be visible for the Cloud Connector process. To achieve this, you must store a file named `scc_daemon_extension.sh` in the installation directory of the Cloud Connector (`/opt/sap/scc`), containing all commands needed for initialization without a shebang.

Example (SAP CommonCryptoLibrary requires SECUDIR to be set):

↳ Sample Code

```
export SECUDIR=/path/to/psefile
```

To activate it, you must reinstall the daemon. Make sure `JAVA_HOME` is set to the JVM used. Then execute the following command to reinstall the daemon:

```
System V init distributions: /opt/sap/scc/daemon.sh reinstall  
systemd distributions: /opt/sap/scc/daemon.sh reinstallSystemd
```

The daemon extension will survive Cloud Connector version updates.

Starting the Cloud Connector

After installation via RPM manager, the Cloud Connector process is started automatically and registered as a daemon process, which ensures the automatic restart of the Cloud Connector after a system reboot.

To start, stop, or restart the process explicitly, open a command shell and use the following commands, which require `root` permissions:

```
System V init distributions: service scc_daemon start|stop|restart  
systemd distributions: systemctl start|stop|restart scc_daemon
```

Next Steps

1. Open a browser and enter: <https://<hostname>:8443>. <hostname> is the host name of the machine on which you installed the Cloud Connector.
If you access the Cloud Connector locally from the same machine, you can simply enter `localhost`.
2. Continue with the initial configuration of the Cloud Connector, see [Initial Configuration \[page 322\]](#).

Related Information

[Recommendations for Secure Setup \[page 309\]](#)

[\[Deprecated\] Replace the Default SSL Certificate \[page 316\]](#)

1.2.1.5 Installation on Mac OS X

Installing the Cloud Connector on a Mac OS X operating system.

Prerequisites

i Note

Mac OS X is not supported for productive scenarios. The developer version described below must not be used as productive version.

- You have either of the following 64-bit operating systems: Mac OS X 10.7 (Lion), Mac OS X 10.8 (Mountain Lion), Mac OS X 10.9 (Mavericks), Mac OS X 10.10 (Yosemite), or Mac OS X 10.11 (El Capitan), Mac OS X 10.12 (Sierra), Mac OS X 10.13 (High Sierra), or Mac OS X 10.14 (Mojave).
- You have downloaded the `.tar.gz` archive for the developer use case on Mac OS X from [SAP Development Tools for Eclipse](#).

- Java 8 must be installed. If you want to use SAP JVM, you can download it from [SAP Development Tools for Eclipse](#) as well.
- Environment variable `<JAVA_HOME>` must be set to the Java installation directory so that the `bin` subdirectory can be found. Alternatively, you can add the Java installation's `bin` subdirectory to the `<PATH>` variable.

Procedure

1. Extract the `.tar.gz` file to an arbitrary directory on your local file system using the following command:

```
tar -xzof sapcc-<version>-macosx-x64.tar.gz
```

2. Go to this directory and start Cloud Connector using the `go.sh` script.
3. Continue with the [Next Steps](#) section.

i Note

The Cloud connector is not started as a daemon, and therefore will not automatically start after a reboot of your system. Also, the Mac OS X version of Cloud Connector does not support the automatic upgrade procedure.

Next Steps

1. Open a browser and enter: `https://<hostname>:8443`. `<hostname>` is the host name of the machine on which you installed the Cloud Connector.
If you access the Cloud Connector locally from the same machine, you can simply enter `localhost`.
2. Continue with the initial configuration of the Cloud Connector, see [Initial Configuration \[page 322\]](#).

Related Information

[Recommendations for Secure Setup \[page 309\]](#)
[\[Deprecated\] Replace the Default SSL Certificate \[page 316\]](#)

1.2.1.6 Recommendations for Secure Setup

For the Connectivity service and the Cloud Connector, you should apply the following guidelines to guarantee the highest level of security for these components.

Security Status

From the *Connector* menu, choose *Security Status* to access an overview showing potential security risks and the recommended actions.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a 'Connector' dropdown set to 'Subaccount: conn2'. Under 'Subaccount: conn2', there are options: Cloud To On-Premise, On-Premise To Cloud, Monitor, Audits, and Log And Trace Files. The main content area is titled 'Security Status' and includes a 'Risk' section with a red exclamation mark icon. Below it is a 'General Security Status' table:

Status	Area	Description	Actions
⚠	UI Certificate	Replace the default UI certificate with a certificate that uses the host name as its common name (CN)	>
⚠	Cipher Suites	Only TLS ciphers with SHA256 (or greater bit lengths) are deemed secure	>
⚠	Trust Store	Trust store is empty — no access restrictions	>
⚠	Authentication	Configure local LDAP for authentication of Cloud Connector users	>
✓	CPIC Trace	Trace is off	>
!	Service User	Set up service user specifically for this Cloud Connector	edit

Below the general status is a 'Subaccount-Specific Security Status' table:

Display Name	Application White-List	Payload Trace
conn2	⚠ White-list is empty — all applications will be trusted	✓ Trace is off
conn3	⚠ White-list is empty — all applications will be trusted	✓ Trace is off

The *General Security Status* addresses security topics that are subaccount-independent.

- Choose any of the *Actions* icons in the corresponding line to navigate to the UI area that deals with that particular topic and view or edit details.

i Note

Navigation is not possible for the last item in the list (*Service User*).

- The service user is specific to the Windows operating system (see [Installation on Microsoft Windows OS \[page 302\]](#) for details) and is only visible when running the Cloud Connector on Windows. It cannot be accessed or edited through the UI. If the service user was set up properly, choose *Edit* and check the corresponding checkbox.

The *Subaccount-Specific Security Status* lists security-related information for each and every subaccount.

i Note

The security status only serves as a reminder to address security issues and shows if your installation complies with all recommended security settings.

Password Policy

Upon installation, the Cloud Connector provides an initial user name and password for the administration UI, and forces the user (**Administrator**) to change the password. You must change the password immediately after installation.

The connector itself does not check the strength of the password. You should select a strong password that cannot be guessed easily.

i Note

To enforce your company's password policy, we recommend that you configure the Administration UI to use an LDAP server for authorizing access to the UI.

UI Access

The Cloud Connector administration UI can be accessed remotely via HTTPS. The connector uses a standard X.509 self-signed certificate as SSL server certificate. You can exchange this certificate with a specific certificate that is trusted by your company. See [\[Deprecated\] Replace the Default SSL Certificate \[page 316\]](#).

i Note

Since browsers usually do not resolve *localhost* to the host name whereas the certificate usually is created under the host name, you might get a certificate warning. In this case, simply skip the warning message.

OS-Level Access

The Cloud Connector is a security-critical component that handles the external access to systems of an isolated network, comparable to a reverse proxy. We therefore recommend that you restrict the access to the operating system on which the Cloud Connector is installed to the minimal set of users who would administrate the Cloud Connector. This minimizes the risk of unauthorized users getting access to credentials, such as certificates stored in the secure storage of the Cloud Connector.

We also recommend that you use the machine to operate only the Cloud Connector and no other systems.

Administrator Privileges

To log on to the Cloud Connector administration UI, the **Administrator** user of the connector must not have an operating system (OS) user for the machine on which the connector is running. This allows the OS administrator to be distinguished from the Cloud Connector administrator. To make an initial connection between the connector and a particular SAP BTP subaccount, you need an SAP BTP user with the required permissions for the related subaccount. We recommend that you separate these roles/duties (that means, you have separate users for Cloud Connector administrator and SAP BTP).

i Note

We recommend that only a small number of users are granted access to the machine as *root* users.

Hard Drive Encryption

Hard drive encryption for machines with a Cloud Connector installation ensures that the Cloud Connector configuration data cannot be read by unauthorized users, even if they obtain access to the hard drive.

Supported Protocols

Currently, the protocols HTTP and RFC are supported for connections between the SAP BTP and on-premise systems when the Cloud Connector and the Connectivity service are used. The whole route from the application virtual machine in the cloud to the Cloud Connector is always SSL-encrypted.

The route from the connector to the back-end system can be SSL-encrypted or SNC-encrypted. See [Configure Access Control \(HTTP\) \[page 380\]](#) and [Configure Access Control \(RFC\) \[page 387\]](#).

Audit Log on OS Level

We recommend that you turn on the audit log on operating system level to monitor the file operations.

Audit Log on Cloud Connector Level

The Cloud Connector audit log must remain switched on during the time it is used with productive systems. The default audit level is **SECURITY**. Set it to **ALL** if required by your company policy. The administrators who are responsible for a running Cloud Connector must ensure that the audit log files are properly archived, to conform to the local regulations. You should switch on audit logging also in the connected back-end systems.

Encryption Ciphers

By default, all available encryption ciphers are supported for HTTPS connections to the administration UI. However, some of them may not conform to your security standards and therefore should be excluded:

1. From the main menu, choose *Configuration* and select the tab *User Interface*, section *Cipher Suites*. By default, all available ciphers are marked as selected.
2. Choose the *Remove* icon to unselect the ciphers that do not meet your security requirements.

i Note

We recommend that you revert the selection to the default (all ciphers selected) whenever you plan to switch to another JVM. As the set of supported ciphers may differ, the selected ciphers may not be supported by the new JVM. In that case the Cloud Connector does not start anymore. You need to fix the issue manually by adapting the file `default-server.xml` (cp. attribute `ciphers`, see section *Accessing the Cloud Connector Administrator UI* above). After switching the JVM, you can adjust the list of eligible ciphers.

3. Choose [Save](#).

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar shows the navigation tree with 'Connector' selected. The main area is titled 'Configuration' and has tabs for 'USER INTERFACE', 'CLOUD', 'ON PREMISE', 'REPORTING', and 'ADVANCED'. The 'USER INTERFACE' tab is active. On the left, under 'trial', there are links for 'Cloud To On-Premise', 'On-Premise To Cloud', 'Monitor', and 'Audits'. The 'Cipher Suites (6)' section is highlighted with a red box. It contains the following table:

Enabled Cipher Suites
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Related Information

[Connectivity via Reverse Proxy \[page 643\]](#)

[Security \[page 567\]](#)

1.2.1.7 Recommended: Exchange UI Certificates in the Administration UI

By default, the Cloud Connector includes a self-signed UI certificate. It is used to encrypt the communication between the browser-based user interface and the Cloud Connector itself. For security reasons, however, you

should replace this certificate with your own one to let the browser accept the certificate without security warnings.

Procedure

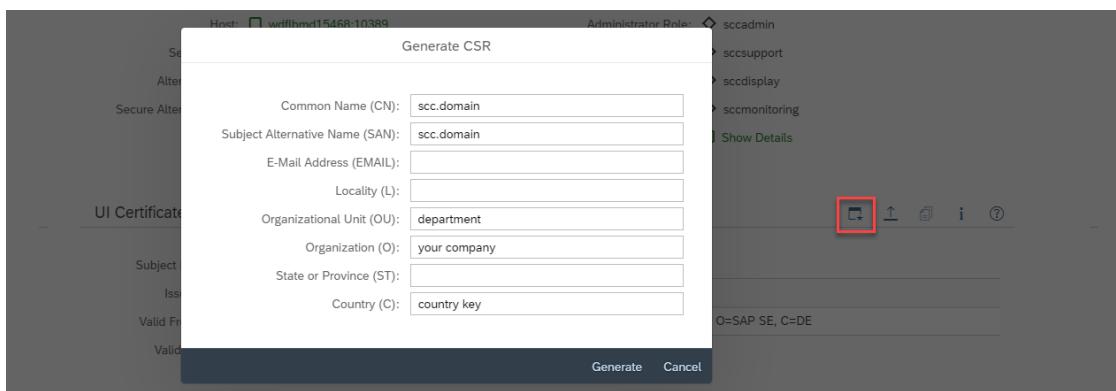
Master Instance

- From the main menu, choose *Configuration* and go to the *User Interface* tab.
- In the *UI Certificate* section, start a certificate signing request procedure by choosing the icon *Generate a Certificate Signing Request*.
- In the pop-up *Generate CSR*, specify a subject fitting to your host name.

For host matching, you should use the available names within the `subjectAlternativeName` (SAN) extension, see [RFC 2818](#). A check verifies whether the host matches one of the entries in the SAN extension.

Choose either of the procedures below, according to your Cloud Connector version:

- Procedure up to Cloud Connector version 2.12.0**



The field <SAN> allows a simple value as well as formatted complex values:

- A simple value is treated as DNS name, for example, `xyz.sap.com` means that the allowed host is `xyz.sap.com`.
- <SAN> also allows a list of DNS names, IPs (4 byte or IPv6), URLs, and [RFC 822](#) names (for example, e-mail addresses).

i Note

In this case, the field <SAN> contains key:value pairs separated by ';' . ';' must not be used in a value.

Example for a complex SAN value : `DNS:localhost;DNS:*.sap.com;IP:10.20.30.40;IP:fe80::78cc:a107:dcf2:73fe%13`.

- Procedure as of Cloud Connector version 2.12.1**

Generate CSR

Subject DN

*Common Name (CN):	scc.internal.corp
E-Mail Address (EMAIL):	
Locality (L):	
Organizational Unit (OU):	department
Organization (O):	company name
State or Province (ST):	
Country (C):	ISO country key

Subject Alternative Names + Delete

Type	Value	Actions
DNS	scc.internal.corp	Delete
DNS	*.internal.corp	Delete
RFC822	ownerscc@company.com	Delete

Generate Cancel

This new version simplifies the SAN management. The CSR generation dialog now separates SAN values from the Subject DN of the certificate by introducing two sections. In the new section **Subject Alternative Names**, you can add additional values easily by pressing the *Add* button. Choose one or more of the following SAN types and provide the matching values:

- DNS: a specific host name (for example, www.sap.com) or a wildcard hostname (for example, *.sap.com).
- IP: an IPv4 or IPv6 address.
- **RFC822**  : an example for this type of value is a simple email address: for example, donotreply@sap.com.
- URI: a URI for which the certificate should be valid.

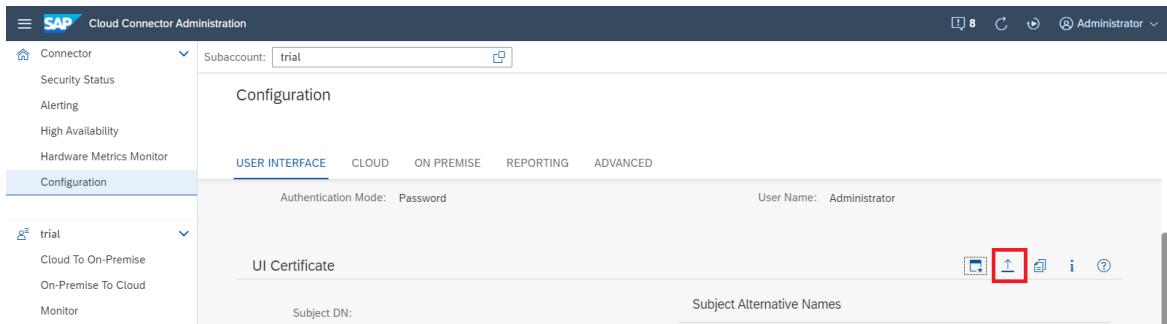
4. Press *Generate*.
5. You are prompted to save the signing request in a file. The content of the file is the signing request in PEM format.

The signing request must be provided to a Certificate Authority (CA) - either one within your company or another one you trust. The CA signs the request and the returned response should be stored in a file.

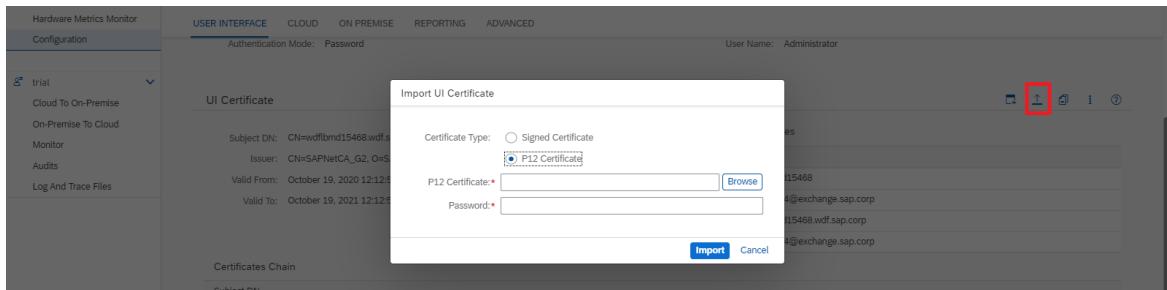
i Note

The response should be either an X.509 certificate or a PKCS#7 in PEM format.

6. To import the signing response, choose the *Upload* icon.



As of Cloud Connector version 2.13, you can also upload an existing PKCS#12 certificate directly (instead of generating a CSR).



7. Select *Browse* to locate the file and then choose the *Import* button.
8. Review the certificate details that are displayed.
9. Restart the Cloud Connector to activate the new certificate.

Shadow Instance

In a High Availability setup, perform the same operation on the shadow instance.

1.2.1.7.1 [Deprecated] Replace the Default SSL Certificate

i Note

This procedure only applies for Cloud Connector versions prior to 2.13. You can now use the UI-based procedure instead, see [Recommended: Exchange UI Certificates in the Administration UI \[page 313\]](#).

Overview

By default, the Cloud Connector includes a self-signed UI certificate. It is used to encrypt the communication between the browser-based user interface and the Cloud Connector itself. For security reasons, however, you should replace this certificate with your own certificate so that the browser accepts the certificate without security warnings.

Up to version **2.5.2**, for this purpose, you need to know the password of the Cloud Connector's Java keystore. This password is generated during installation and then kept in an encrypted secure storage area. To obtain the password, follow the steps described below.

i Note

As of version **2.6.0**, you can easily replace the default certificate within the Cloud Connector administration UI. See [Recommended: Exchange UI Certificates in the Administration UI \[page 313\]](#).

⚠ Caution

The Cloud Connector's keystore may contain a certificate used in the High Availability setup. This certificate has the alias "**ha**". Any changes on it or removal would cause a disruption of communication between the shadow and the master instance, and therefore to a failed procedure. We recommend that you replace the keystore on both the master and shadow server before establishing the connection between the two instances.

Procedure

You can read the password by executing the following command:

- on Microsoft Windows OS:

```
java -cp <scc_install_dir>\plugins\com.sap.scc.rt*.jar -  
Djava.library.path=<scc_install_dir>\auditor com.sap.scc.jni.SecStoreAccess -  
path <scc_install_dir>\scc_config -p
```

- on Linux OS:

```
java -cp /opt/sap/scc/plugins/com.sap.scc.rt*.jar -  
Djava.library.path=/opt/sap/scc/auditor com.sap.scc.jni.SecStoreAccess -  
path /opt/sap/scc/scc_config -p
```

i Note

Memorize the keystore password, as you will need it for later operations. See related links.

Make sure you go to directory `/opt/sap/scc/config` before executing the commands described in the following procedures.

i Note

For a detailed description of the `keytool` tool, see <http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html>.

Related Information

Recommended: Exchange UI Certificates in the Administration UI [page 313]

[Deprecated] Use a Self-Signed Certificate [page 318]

[Deprecated] Use Certificates Signed by a Trusted Certificate Authority [page 319]

1.2.1.7.2 [Deprecated] Use a Self-Signed Certificate

Generate a self-signed certificate for special purposes like, for example, a demo setup.

Context

i Note

As of Cloud Connector 2.10 you can generate self-signed certificates also from the administration UI. See [Configure a CA Certificate for Principal Propagation \[page 354\]](#) and [Initial Configuration \(HTTP\) \[page 329\]](#). In this case, the steps below are not required.

If you want to use a simple, self-signed certificate, follow the procedure below.

i Note

The parameter values in the following section are examples.

The server configuration delivered by SAP uses the same password for key store (option \-storepass) and key (option \-keypass) under alias `tomcat`.

Procedure

1. Remove the current default certificate:

```
keytool -delete -alias tomcat -keystore ks.store -storepass <password>
```

2. Generate a certificate:

```
keytool -genkey -v -keyalg RSA -alias tomcat -keypass <password> -keystore ks.store -storepass <password> -dname "CN=SCC, OU=<YourCompany>, O=<YourCompany>"
```

3. Self-sign it - you will be prompted for the keypass password defined in step 2:

```
keytool -selfcert -v -alias tomcat -storepass <password> -keystore ks.store
```

1.2.1.7.3 [Deprecated] Use Certificates Signed by a Trusted Certificate Authority

Use a signed certificate by a trusted certificate authority (CA) to increase the security level when running the Cloud Connector.

i Note

This procedure only applies for Cloud Connector versions prior to 2.13. You can now use the UI-based procedure instead, see [Recommended: Exchange UI Certificates in the Administration UI \[page 313\]](#).

Before starting the procedure, note the following:

- The parameter values of the following steps are only examples.
- We recommend that you use a signed certificate by a trusted CA, because it is more secure than a self-signed certificate.
- For your convenience, you can set the generated password as environment variable, like in the command below, and then use **\$PASS** as a password:
export PASS='<the release-dependent command as given in the parent page>'
- The keytool supports **delete** and **changealias** commands. If the Cloud Connector SSL certificate is changed on a running instance, we recommend that you prepare a new certificate under a temporary alias. Once everything is ready, you change the alias.

Procedure

i Note

If you already have a signed certificate produced by a trusted certificate authority (CA), skip steps 1,2, and 4, and only follow the instructions provided in step 3.

1. Generate your key pair if you start fresh:

```
keytool -genkey -v -keyalg RSA -alias tomcat -keystore ks.store -storepass <password> -keypass <password> -dname "CN=SCC, OU=<YourCompany>, O=<YourCompany>"
```

Alternatively, you may reuse an existing key store.

2. Create a local certificate signing request (CSR):

```
keytool -certreq -keyalg RSA -alias tomcat -keystore ks.store -storepass <password> -keypass <password> -file <csr-file-name>
```

You now have a file called <csr-file-name> that you can submit to the certificate authority. In return, you get a certificate.

3. Import the certificate chain that you obtained from your trusted CA:

```
keytool -import -alias root -keystore ks.store -storepass <password> -trustcacerts -file <filename_of_the_certificate_chain>
```

i Note

If you already have a signed certificate produced by a trusted certificate authority (CA), continue with the following steps (skipping 1,2, and 4):

- Create a backup copy of the keystore file `ks.store`.
- Import your keystore with option `-importkeystore` (documented [here](#)) into the Cloud Connector keystore `ks.store`.

```
keytool -importkeystore -srckeystore mypfxfile.pfx -srcstoretype pkcs12  
-destkeystore ks.store -deststoretype JKS -srcstorepass <your_pw> -  
deststorepass <pw_from_sec_store> -srcalias <your_alias> -destalias  
tomcat -srckeypass <your_pw> -destkeypass <pw_from_sec_store>
```

4. Import your new certificate:

```
keytool -import -alias tomcat -keystore ks.store -storepass <password> -file  
<your_certificate_filename>
```

The password is created at installation time and stored in the secure storage. Thus, only applications with access can read the password. You can read password using Java:

```
jar -xf /opt/sap/scc/dropins/scc/plugins/com.sap.scc.tomcat.utils*.jar lib/  
libsapsecstore4j.so  
java -cp /opt/sap/scc/dropins/scc/plugins/com.sap.scc.tomcat.utils*.jar -  
Djava.library.path=./lib/ com.sap.mw.scc.util.SecStoreAccess -show
```

You might need to adapt the configuration if you want to use another key storage file or change the current configuration (HTTPS port, authentication type, SSL protocol, and so on). You can find the SSL configuration in the `Connector` section of the file:

- Microsoft Windows OS: <install_dir>\config_master\org.eclipse.gemini.web.tomcat\default-server.xml
- Linux OS: /opt/sap/scc/config_master/org.eclipse.gemini.web.tomcat/default-server.xml

i Note

We recommend that you do not modify the configuration unless you have expertise in this area.

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"  
maxThreads="150" scheme="https" secure="true"  
keystoreFile="config/ks.store" keystorePass="${jks.password}" keyPass="${  
jks.password}" keyAlias="tomcat"  
truststoreFile="config/ks.store" truststorePass="${jks.password}"  
clientAuth="want" sslProtocol="TLS"  
compression="on" compressionMinSize="1024"  
noCompressionUserAgents="ozilla,traviata,*MSIE 6.*"  
compressableMimeType="text/html,text/xml,text/plain,text/javascript,text/  
css,text/json,application/x-javascript,application/javascript,application/json"/>
```

Related Information

For more information about configuring SSL, see http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html#SSL_and_Tomcat.

1.2.2 Configuration

Configure the Cloud Connector to make it operational for connections between your SAP BTP applications and on-premise systems.

Topic	Description
Initial Configuration [page 322]	After installing the Cloud Connector and starting the Cloud Connector daemon, you can log on and perform the required configuration to make your Cloud Connector operational.
Managing Subaccounts [page 338]	How to connect SAP BTP subaccounts to your Cloud Connector.
Authenticating Users against On-Premise Systems [page 349]	Basic authentication and principal propagation (user propagation) are the authentication types currently supported by the Cloud Connector.
Configure Access Control [page 379]	Configure access control or copy the complete access control settings from another subaccount on the same Cloud Connector.
Configuration REST APIs [page 404]	Configure a newly installed Cloud Connector (initial configuration, subaccounts, access control) using the configuration REST API.
Configure an On-Premise User Store [page 490]	Configure applications running on SAP BTP to use your corporate LDAP server as a user store.
Using Service Channels [page 492]	Service channels provide access from an external network to certain services on SAP BTP, which are not exposed to direct access from the Internet.
Configure Trust [page 500]	Set up an allowlist for trusted cloud applications and a trust store for on-premise systems in the Cloud Connector.
Connect DB Tools to SAP HANA via Service Channels [page 496]	How to connect database, BI, or replication tools running in the on-premise network to a HANA database on SAP BTP using the service channels of the Cloud Connector.
Configure Domain Mappings for Cookies [page 503]	Map virtual and internal domains to ensure correct handling of cookies in client/server communication.
Configure Solution Management Integration [page 504]	Activate Solution Management reporting in the Cloud Connector.
Configure Tunnel Connections [page 506]	Adapt connectivity settings that control the throughput by choosing the appropriate limits (maximal values).
Configure the Java VM [page 507]	Adapt the JVM settings that control memory management.

Topic	Description
Configuration Backup [page 508]	Backup and restore your Cloud Connector configuration.

1.2.2.1 Initial Configuration

After installing and starting the Cloud Connector, log on to the administration UI and perform the required configuration to make your Cloud Connector operational.

Tasks

[Prerequisites \[page 322\]](#)

[Log on to the Cloud Connector \[page 323\]](#)

[Change your Password and Choose Installation Type \[page 324\]](#)

[Set up Connection Parameters and HTTPS Proxy \[page 325\]](#)

[Establish Connections to SAP BTP \[page 328\]](#)

Prerequisites

- You have assigned one of these roles/role collections to the subaccount user that you use for initial Cloud Connector setup, depending on the SAP BTP environment in which your subaccount is running:

i Note

For the **Cloud Foundry** environment, you must know on which cloud management tools feature set (A or B) your account is running. For more information on feature sets, see [Cloud Management Tools – Feature Set Overview](#).

Environment	Required Roles/Role Collections	More Information
Cloud Foundry [feature set A]	The user must be a member of the <i>global account</i> that the subaccount belongs to. Alternatively, you can assign the user as <i>Security Administrator</i> .	Add Members to Your Global Account Managing Security Administrators in Your Subaccount [Feature Set A]

Environment	Required Roles/Role Collections	More Information
Cloud Foundry [feature set B]	<p>Assign at least one of these <i>default role collections</i> (all of them including the role Cloud Connector Administrator):</p> <ul style="list-style-type: none"> ○ Subaccount Administrator ○ Cloud Connector Administrator ○ Connectivity and Destination Administrator <p>Alternatively, you can assign a <i>custom role collection</i> to the user that includes the role Cloud Connector Administrator.</p>	Default Role Collections [Feature Set B] [page 13] Role Collections and Roles in Global Accounts, Directories, and Subaccounts [Feature Set B]
Neo	<p>Assign at least one of these <i>default roles</i>:</p> <ul style="list-style-type: none"> ○ Cloud Connector Admin ○ Administrator <p>Alternatively, you can assign a <i>custom role</i> to the user that includes the permission <code>manageSCCTunnels</code>.</p>	Managing Member Authorizations in the Neo Environment

After establishing the Cloud Connector connection, this user is not needed any more, since it serves only for initial connection setup. You may revoke the corresponding role assignment then and remove the user from the [Members](#) list.

i Note

If the Cloud Connector is installed in an environment that is operated by SAP, SAP provides a user that you can add as member in your SAP BTP subaccount and assign the required role.

- We strongly recommend that you read and follow the steps described in [Recommendations for Secure Setup \[page 309\]](#). For operating the Cloud Connector securely, see also [Security Guidelines \[page 574\]](#).

Back to [Tasks \[page 322\]](#)

Log on to the Cloud Connector

To administer the Cloud Connector, you need a Web browser. To check the list of supported browsers, see [Prerequisites and Restrictions](#) → section *Browser Support*.

1. In a Web browser, enter: `https://<hostname>:<port>`

- <hostname> refers to the machine on which the Cloud Connector is installed. If installed on your machine, you can simply enter **localhost**.
 - <port> is the Cloud Connector port specified during installation (the default port is **8443**).
2. On the logon screen, enter **Administrator / manage** (case sensitive) for <User Name> / <Password>.

Back to [Tasks \[page 322\]](#)

Change your Password and Choose Installation Type

1. When you first log in, you must change the password before you continue, regardless of the installation type you have chosen.
2. Choose between master and shadow installation. Use **Master** if you are installing a single Cloud Connector instance or a main instance from a pair of Cloud Connector instances. See [Install a Failover Instance for High Availability \[page 518\]](#).

Initial Setup

You are required to change your password before being permitted to continue

Mandatory Password Change	Choose Installation Type
Current Password: <input type="text"/>	<input checked="" type="radio"/> Master (Primary Installation)
New Password: <input type="text"/>	<input type="radio"/> Shadow (Backup Installation)
Repeat New Password: <input type="text"/>	Description: <input type="text"/>

3. You can edit the password for the **Administrator** user from *Configuration* in the main menu, tab *User Interface*, section *Authentication*:

No data

i Note

User name and password cannot be changed at the same time. If you want to change the user name, you must enter only the current password in a first step. Do not enter values for <New Password> or <Repeat New Password> when changing the user name. To change the password in second step, enter the old password, the new one, and the repeated (new) password, but leave the user name unchanged.

Back to [Tasks \[page 322\]](#)

Set up Connection Parameters and HTTPS Proxy

When logging in for the first time, the following screen is displayed every time you choose an option from the main menu that requires a configured subaccount:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a tree view with 'Connector' expanded, showing 'Security Status', 'Alerting', 'High Availability', 'Hardware Metrics Monitor', and 'Configuration'. Under 'Define Subaccount', it lists 'Cloud To On-Premise', 'On-Premise To Cloud', 'Monitor', 'Audits', and 'Log And Trace Files'. The main content area is titled 'Define Subaccount' with a note: 'Cloud Connector is not configured and remains inoperative unless you define at least one subaccount'. It contains two sections: 'First Subaccount' and 'HTTPS Proxy'. The 'First Subaccount' section has fields for Region*, Subaccount*, Display Name, Subaccount User*, Password*, Location ID (with placeholder 'Enter location ID to overwrite default'), and Description. The 'HTTPS Proxy' section has fields for Host, Port, User, and Password.

If your internal landscape is protected by a firewall that blocks any outgoing TCP traffic, you must specify an HTTPS proxy that the Cloud Connector can use to connect to SAP BTP. Normally, you must use the same proxy settings as those being used by your standard Web browser. The Cloud Connector needs this proxy for two operations:

- Download the correct connection configuration corresponding to your subaccount ID in SAP BTP.
- Establish the SSL tunnel connection from the Cloud Connector user to your SAP BTP subaccount.

i Note

If you want to skip the initial configuration, you can click the icon in the upper right corner. You might need this in case of connectivity issues shown in your logs. You can add subaccounts later as described in [Managing Subaccounts \[page 338\]](#).

The Cloud Connector collects the following required information for your subaccount connection:

1. For <Region>, specify the SAP BTP region that should be used. You can choose it from the drop-down list, see [Regions](#).

i Note

You can also configure a region yourself, if it is not part of the standard list. Either insert the region host manually, or create a custom region, as described in [Configure Custom Regions \[page 349\]](#).

- For <Subaccount>, <Subaccount User> and <Password>, enter the values you obtained when you registered your subaccount on SAP BTP.

i Note

For a subaccount in the **Cloud Foundry** environment, you must enter the subaccount **ID** as <Subaccount>, rather than its actual (technical) name. For information on getting the subaccount ID, see [Find Your Subaccount ID \(Cloud Foundry Environment\) \[page 348\]](#). As <Subaccount User> you must provide your **Login E-mail** instead of a user ID. The user must be a member of the global account the subaccount belongs to.

You can also add a new subaccount user with the role `Cloud Connector Admin` in the SAP BTP cockpit and use the new user and password.

For the **Neo** environment, see [Add Members to Your Neo Subaccount](#).

For the **Cloud Foundry** environment, see [Add Org Members Using the Cockpit](#).

→ Tip

When using SAP Cloud Identity Services - Identity Authentication (IAS) as platform identity provider with two-factor authentication for your subaccount, you can simply append the required token to the regular password.

→ Tip

For a subaccount in the **Cloud Foundry** environment, the Cloud Connector supports the use of a custom identity provider (IDP) via single sign-on (SSO) passcode. For more information, see [Use a Custom IDP for Subaccount Configuration \[page 334\]](#).

- (Optional) You can define a <Display Name> that lets you easily recognize a specific subaccount in the UI compared to the technical subaccount name.
- (Optional) You can define a <Location ID> identifying the location of this Cloud Connector for a specific subaccount. As of Cloud Connector release **2.9.0**, the location ID is used as routing information and therefore you can connect multiple Cloud Connectors to a single subaccount. If you don't specify any value for <Location ID>, the default is used, which represents the behavior of previous Cloud Connector versions. The location ID must be unique per subaccount and should be an identifier that can be used in a URI. To route requests to a Cloud Connector with a location ID, the location ID must be configured in the respective destinations.

i Note

Location IDs provided in older versions of the Cloud Connector are discarded during upgrade to ensure compatibility for existing scenarios.

- Enter a suitable proxy host from your network and the port that is specified for this proxy. If your network requires an authentication for the proxy, enter a corresponding proxy user and password. You must specify a proxy server that supports SSL communication (a standard HTTP proxy does not suffice).

i Note

These settings strongly depend on your specific network setup. If you need more detailed information, please contact your local system administrator.

6. (Optional) You can provide a <Description> (free-text) of the subaccount that is shown when choosing the *Details* icon in the *Actions* column of the *Subaccount Dashboard*. It lets you identify the particular Cloud Connector you use.
7. Choose *Save*.

The Cloud Connector now starts a handshake with SAP BTP and attempts to establish a secure SSL tunnel to the server that hosts the subaccount in which your on-demand applications are running. However, no requests are yet allowed to pass from the cloud side to any of your internal back-end systems. To allow your on-demand applications to access specific internal back-end systems, proceed with the access configuration described in the next section.

i Note

The internal network must allow access to the port. Specific configuration for opening the respective port(s) depends on the firewall software used. The default ports are **80** for HTTP and **443** for HTTPS. For RFC communication, you must open a gateway port (default: **33+<instance number>**) and an arbitrary message server port. For a connection to a HANA Database (on SAP BTP) via JDBC, you must open an arbitrary *outbound* port in your network. Mail (SMTP) communication is not supported.

- If you later want to change your proxy settings (for example, because the company firewall rules have changed), choose *Configuration* from the main menu and go to the *Cloud* tab, section *HTTPS Proxy*. Some proxy servers require credentials for authentication. In this case, you must provide the relevant user/password information.

The screenshot shows the SAP Cloud Connector Administration interface. The top navigation bar includes the SAP logo, the title 'Cloud Connector Administration', and a user 'Administrator'. The left sidebar has a 'Connector' section with sub-options: Security Status, Alerting, High Availability, Hardware Metrics Monitor, and Configuration (which is selected). Below this is a 'conn2' section with options: Cloud To On-Premise, On-Premise To Cloud, Monitor, Audits, and Log And Trace Files. The main content area has a 'Subaccount: conn2' dropdown. The 'CLOUD' tab is active, showing the 'Configuration' section. Under 'Connector Info', there is a 'Description:' field and a 'HTTPS Proxy' section. The 'HTTPS Proxy' section is highlighted with a red box and contains fields for 'Host:', 'Port:', and 'User:'. There are edit, info, and help icons at the bottom right of the 'Connector Info' section.

- If you want to change the description for your Cloud Connector, choose *Configuration* from the main menu, go to the *Cloud* tab, section *Connector Info* and edit the description:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a 'Connector' dropdown set to 'Subaccount: conn2'. The main area has tabs: USER INTERFACE, CLOUD (which is selected), ON PREMISE, REPORTING, and ADVANCED. Under 'CLOUD', there's a 'Connector Info' section with a description: 'Description: Cloud Connector on VM in DMZ'.

[Back to Tasks \[page 322\]](#)

Establish Connections to SAP BTP

As soon as the initial setup is complete, the tunnel to the cloud endpoint is open, but no requests are allowed to pass until you have performed the *Access Control* setup, see [Configure Access Control \[page 379\]](#).

To manually close (and reopen) the connection to SAP BTP, choose your subaccount from the main menu and select the *Disconnect* button (or the *Connect* button to reconnect to SAP BTP).

The screenshot shows the SAP Cloud Connector Administration interface with 'Trial' selected in the subaccount dropdown. The main area shows 'Trial' connected since 12 August 2020. Below is a 'Subaccount Overview' table:

Region: Europe (Rot) - Trial	Subaccount: trial
Region Host: hanatrial.ondemand.com	Initiated By: Anonymous
HTTPS Proxy: ◇	Location ID:
Subaccount Certificate: Certificate valid until 12 August 2021 16:25:36 CEST	Description:
System Certificate: ◇	

A red box highlights the Region Host, HTTPS Proxy, and Subaccount Certificate rows.

- The green icon next to *Region Host* indicates that it is valid and can be reached.
- If an *HTTPS Proxy* is configured, its availability is shown the same way. In the screenshot, the grey diamond icon next to *HTTPS Proxy* indicates that connectivity is possible without proxy configuration.

In case of a timeout or a connectivity issue, these icons are yellow (warning) or red (error), and a tooltip shows the cause of the problem. *Initiated By* refers to the user that has originally established the tunnel. During normal operations, this user is no longer needed. Instead, a certificate is used to open the connection to a subaccount.

- The status of the certificate is shown next to *Subaccount Certificate*. It is shown as valid (green icon), if the expiration date is still far in the future, and turns to yellow if expiration approaches according to your alert settings. It turns red as soon as it has expired. This is the latest point in time, when you should [Update the Certificate for Your Subaccount \[page 345\]](#).

i Note

When connected, you can monitor the Cloud Connector also in the *Connectivity* section of the SAP BTP cockpit. There, you can track attributes like version, description and high availability set up. Every Cloud Connector configured for your subaccount automatically appears in the *Connectivity* section of the cockpit.

Back to [Tasks \[page 322\]](#)

Related Information

[Managing Subaccounts \[page 338\]](#)

[Initial Configuration \(HTTP\) \[page 329\]](#)

[Initial Configuration \(RFC\) \[page 331\]](#)

[Configuring the Cloud Connector for LDAP \[page 333\]](#)

[Managing Member Authorizations in the Neo Environment](#)

[Use a Custom IDP for Subaccount Configuration \[page 334\]](#)

1.2.2.1.1 Initial Configuration (HTTP)

Configure the Cloud Connector for HTTP communication.

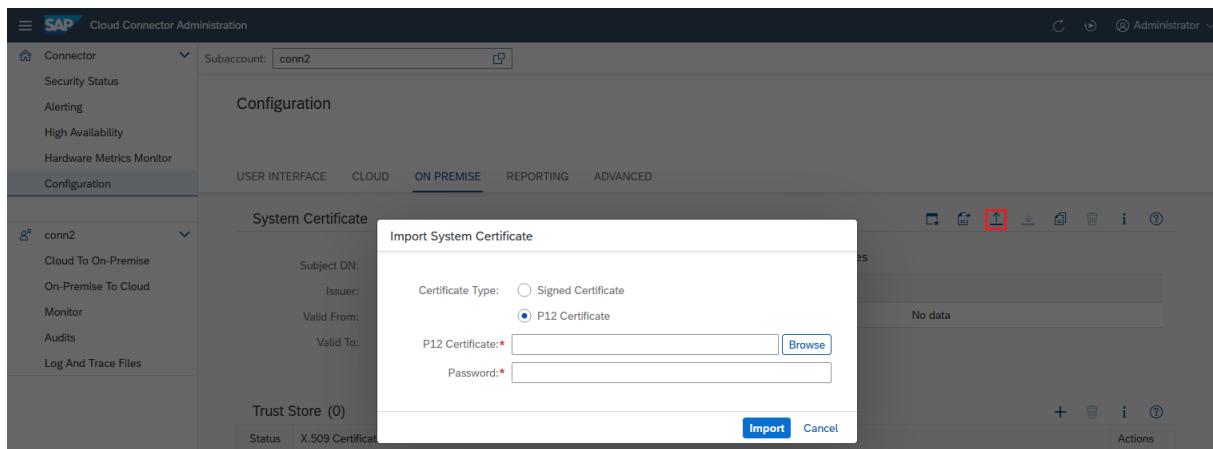
Installation of a System Certificate for Mutual Authentication

To set up a mutual authentication between the Cloud Connector and any backend system it connects to, you can import an X.509 client certificate into the Cloud Connector. The Cloud Connector then uses the so-called *system certificate* for all HTTPS requests to backends that request or require a client certificate. The CA that signed the Cloud Connector's client certificate must be trusted by all backend systems to which the Cloud Connector is supposed to connect.

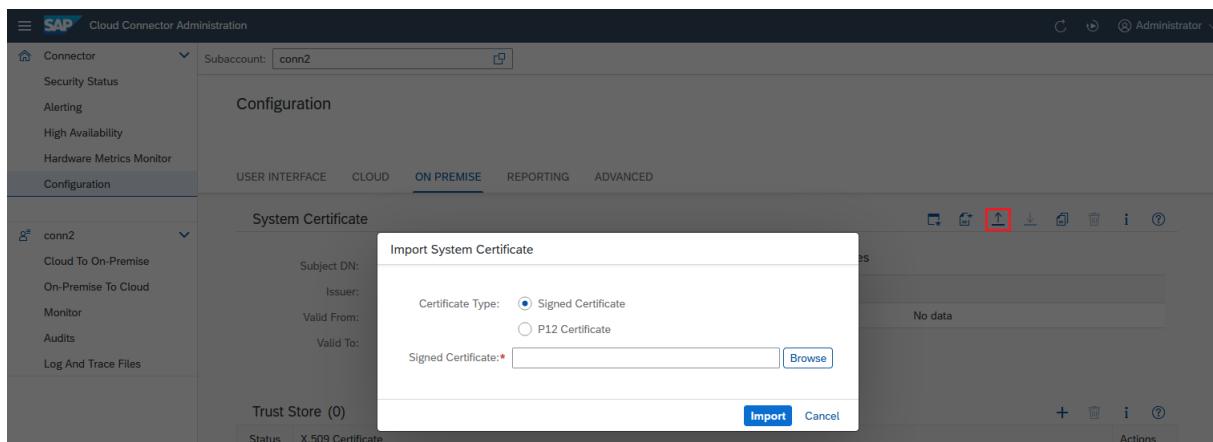
You must provide the system certificate as PKCS#12 file containing the client certificate, the corresponding private key and the CA root certificate that signed the client certificate (plus potentially the certificates of any intermediate CAs, if the certificate chain is longer than 2).

Procedure

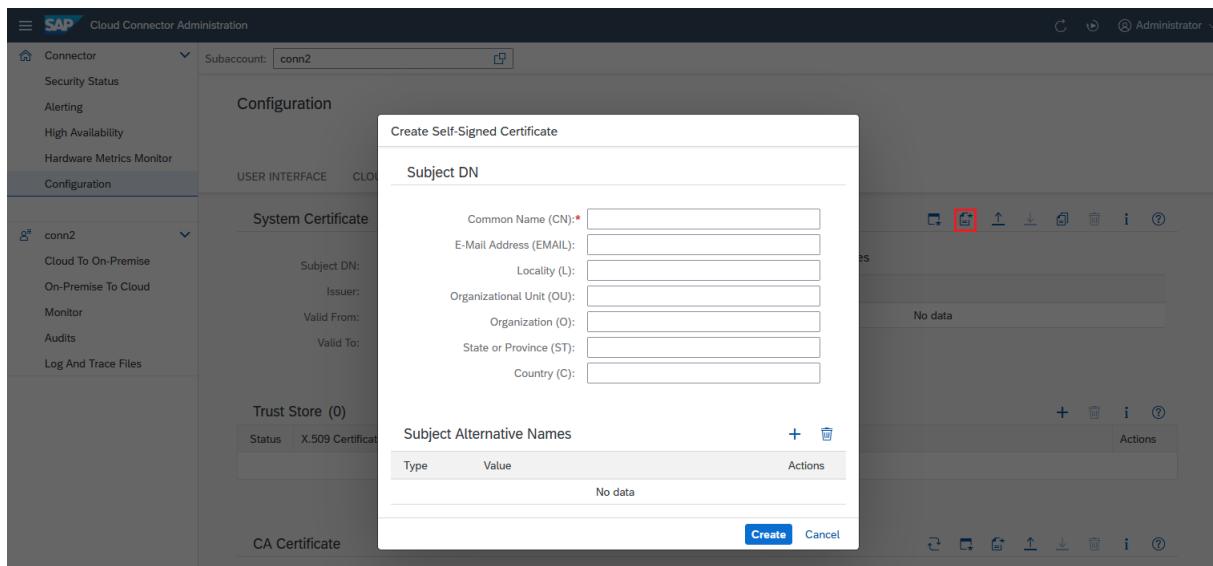
From the left panel, choose *Configuration*. On the tab *On Premise*, choose *System Certificate* *Import a certificate* to upload a certificate and provide its password:



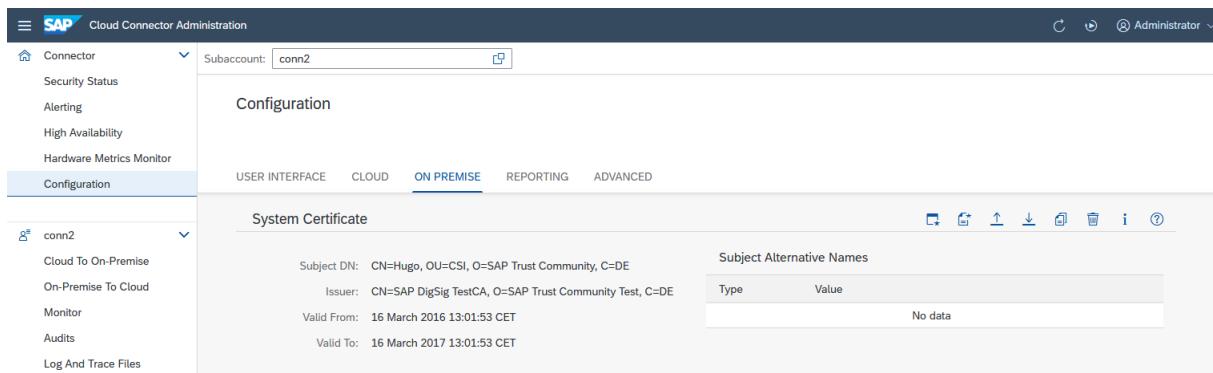
A second option is to start a *certificate signing request* procedure as described for the UI certificate in [Recommended: Exchange UI Certificates in the Administration UI \[page 313\]](#) and upload the resulting signed certificate.



As of version 2.10, there is a third option - generating a self-signed certificate. It might be useful if no CA is needed, for example, in a demo setup or if you want to use a dedicated CA. For this option, choose [*Create and import a self-signed certificate*](#):



If a system certificate has been imported successfully, its distinguished name, the name of the issuer, and the validity dates are displayed:



If a system certificate is no longer required, you can delete it. To do this, use the respective button and confirm deletion. If you need the public key for establishing trust with a server, you can simply export the full chain via the *Export* button.

Related Information

[Configure Access Control \(HTTP\) \[page 380\]](#)

1.2.2.1.2 Initial Configuration (RFC)

Configure a Secure Network Connection (SNC) to set up the Cloud Connector for RFC communication to an ABAP backend system.

SNC Configuration for Mutual Authentication

To set up a mutual authentication between Cloud Connector and an ABAP backend system (connected via RFC), you can configure SNC for the Cloud Connector. It will then use the associated PSE for all RFC SNC requests. This means that the SNC identity, represented by this PSE, must:

- Be trusted by all backend systems to which the Cloud Connector is supposed to connect
- Play the role of a trusted external system by adding the SNC name of the Cloud Connector to the SNCSYSACL table. You can find more details in the SNC configuration documentation for the release of your ABAP system.

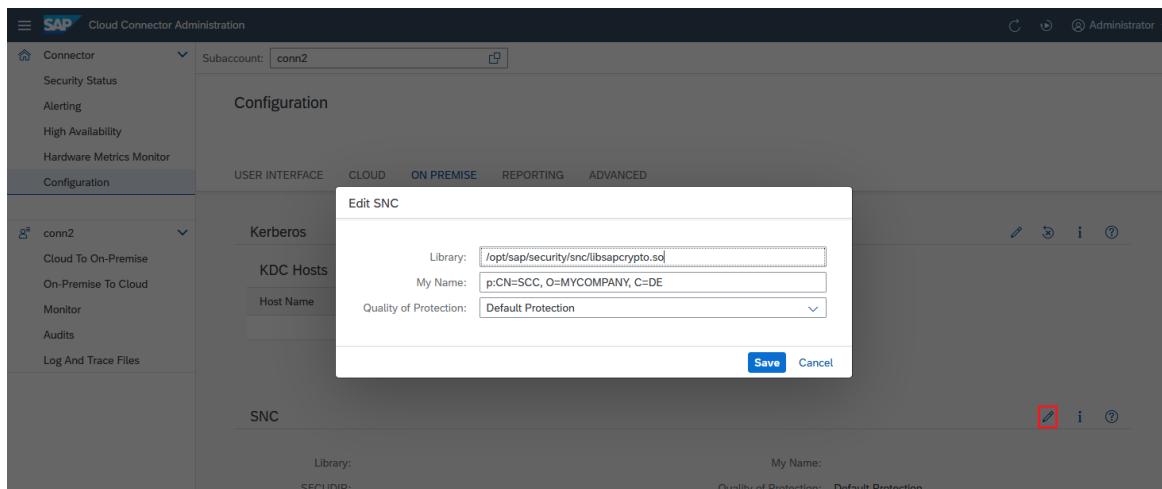
Prerequisites

You have configured your ABAP system(s) for SNC. For detailed information on configuring SNC for an ABAP system, see also [Configuring SNC on AS ABAP](#). In order to establish trust for Principal Propagation, follow the steps described in [Configure Principal Propagation for RFC \[page 364\]](#).

Configuration Steps

1. Logon to the Cloud Connector
2. Choose *Configuration* from the main menu and go to tab *On Premise*, section *SNC*.
3. Enter the corresponding values in the fields *Library Name*, *My Name*, and *Quality of Protection*
4. Press *Save*.

Example:



- *Library Name*: Provides the location of the SNC library you are using for the Cloud Connector.

i Note

Bear in mind that you must use one and the same security product on both sides of the communication.

- **My Name:** The SNC name that identifies the Cloud Connector. It represents a valid scheme for the SNC implementation that is used.
- **Quality of Protection:** Determines the level of protection that you require for the connectivity to the ABAP systems.

i Note

When using **CommonCryptoLibrary** as SNC implementation, note [1525059](#) will help you to configure the PSE to be associated with the user running the Cloud Connector process.

Related Information

[Configure Principal Propagation for RFC \[page 364\]](#)

1.2.2.1.3 Configuring the Cloud Connector for LDAP

Configure the Cloud Connector to support LDAP in different scenarios (cloud applications using LDAP or Cloud Connector authentication).

Prerequisites

You have installed the Cloud Connector and done the basic configuration:

[Installation \[page 282\]](#)

[Initial Configuration \[page 322\]](#)

Steps

When using LDAP-based user management, you have to configure the Cloud Connector to support this feature. Depending on the scenario, you need to perform the following steps:

Scenario 1: Cloud applications using LDAP for authentication. Configure the destination of the LDAP server in the Cloud Connector: [Configure Access Control \(LDAP\) \[page 393\]](#).

Scenario 2: Internal Cloud Connector user management. Activate LDAP user management in the Cloud Connector: [Use LDAP for Authentication \[page 513\]](#).

1.2.2.1.4 Use a Custom IDP for Subaccount Configuration

Enable custom identity provider (IDP) authentication to configure a Cloud Foundry subaccount in the Cloud Connector by using a one-time passcode.

Content

[Context \[page 334\]](#)

[Get the URL \[Feature Set A\] \[page 337\]](#)

[Get the URL \[Feature Set B\] \[page 338\]](#)

[Get the One-Time Passcode \[page 338\]](#)

Context

For a subaccount in the **Cloud Foundry** environment that uses a custom IDP, you can choose this IDP for authentication instead of the (default) SAP ID service when configuring the subaccount in the Cloud Connector.

Using custom IDP authentication, you can perform the following operations in the Cloud Connector:

Operation	Description
Set up Connection Parameters and HTTPS Proxy [page 325]	Add an <i>initial</i> subaccount to a fresh Cloud Connector installation.
Managing Subaccounts [page 338]	Add <i>additional</i> subaccounts to an existing Cloud Connector installation.
Update the Certificate for a Subaccount [page 345]	Refresh a subaccount certificate's validity period.

To enable custom IDP authentication, for each of these operations you must enter the marker value **\$SAP-CP-SSO-PASSCODE\$** in the **<Subaccount User>** or **<User Name>** field, and a one-time generated passcode (known as *temporary authentication code*) in the **<Password>** field:

- When adding the **initial subaccount** to a fresh Cloud Connector installation, enter the user name **\$SAP-CP-SSO-PASSCODE\$** and the passcode on the Cloud Connector's *Define Subaccount* screen (see also [Set up Connection Parameters and HTTPS Proxy \[page 325\]](#)):

SAP Cloud Connector Administration

Administrator

Connector

Define Subaccount

Cloud Connector is not configured and remains inoperative unless you define at least one subaccount

First Subaccount

Region: * [Input Field]
Subaccount: * [Input Field]
Display Name: [Input Field]
Subaccount User: * \$SAP-CP-SSO-PASSCODE\$ [Input Field] (highlighted)
Password: * <Passcode> [Input Field] (highlighted)
Location ID: Enter location ID to overwrite default
Description: [Text Area]

HTTPS Proxy

Host: [Input Field]
Port: [Input Field]
User: [Input Field]
Password: [Input Field]

Save ?

- When adding one or more **additional subaccount(s)** to an existing Cloud Connector installation, provide the user name **\$SAP-CP-SSO-PASSCODE\$** and the passcode via the [Connector](#) screen (see also [Managing Subaccounts \[page 338\]](#)):

SAP Cloud Connector Administration

Administrator

Connector

Subaccount: trial

Connector

+ Add Subaccount Backup ...

Connector Overview

Connector ID: [REDACTED] Security Status: ⚠ Low risk
Local Name: [REDACTED] High Availability: ⚠ Disconnected
Local IP: [REDACTED] Alerts: ⚠ 19

Connector

Cloud To On-Premise

On-Premise To Cloud

Monitor

Add Subaccount

Region:*

Subaccount:*

Display Name:

Subaccount User:*

Subaccount User: \$SAP-CP-SSO-PASSCODE\$

Password:*

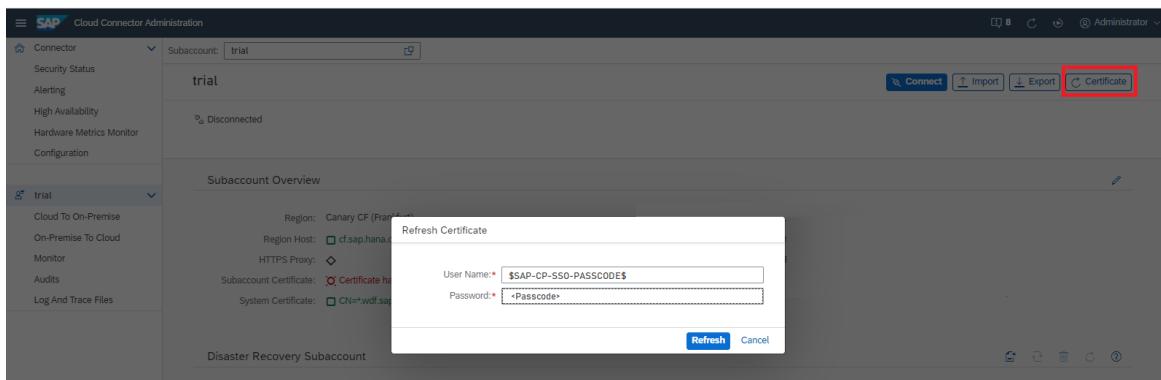
Password: <Passcode>

Location ID: Enter location ID to overwrite default

Description:

Save **Cancel**

- To refresh a subaccount certificate, enter the user name **\$SAP-CP-SSO-PASSCODE\$** and the passcode via the corresponding **<Subaccount>** screen (see also [Update the Certificate for a Subaccount \[page 345\]](#)):



To retrieve the one-time generated passcode, you must use the correct login URL for single sign-on (SSO) to access your custom IDP. The procedure to get this URL depends on the SAP BTP feature set you are using.

i Note

To choose the right procedure, you must know on which cloud management tools feature set (A or B) your SAP BTP account is running. For more information on feature sets, see [Cloud Management Tools – Feature Set Overview](#).

Next Step

[Get the URL \[Feature Set A\] \[page 337\]](#)

[Get the URL \[Feature Set B\] \[page 338\]](#)

⚠ Caution

Mind the respective user rights described in the prerequisites for [Initial Configuration \[page 322\]](#) and [Managing Subaccounts \[page 338\]](#). For feature set A, it is mandatory to be a subaccount *Security Administrator*, not only a global account member.

Back to [Content \[page 334\]](#)

Get the URL [Feature Set A]

Choose one of the following options:

Option 1: Assemble the URL

The URL pattern is `https://login.cf.<btp-region-host>/passcode`.

1. Get the SAP BTP region host, for example, `eu10.hana.ondemand.com`.
2. Assemble the final URL to be used, in this case: `https://login.cf.eu10.hana.ondemand.com/passcode`.

Option 2: Get the URL Using the Cloud Foundry CLI

Use the Cloud Foundry [CLI](#) to perform the following steps:

1. Execute the command `cf api` to navigate to the SAP BTP region.
2. Execute `cf login --sso` to get the URL.

↳ Sample Code

```
$ cf api api.cf.eu10.hana.ondemand.com
Setting api endpoint to api.cf.eu10.hana.ondemand.com...
OK
api endpoint: https://api.cf.eu10.hana.ondemand.com
api version: 2.156.0
$ cf login --sso
API endpoint: https://api.cf.eu10.hana.ondemand.com
Temporary Authentication Code ( Get one at https://
login.cf.eu10.hana.ondemand.com/passcode ):
```

Next Step

[Get the One-Time Passcode \[page 338\]](#)

Back to [Content \[page 334\]](#)

Get the URL [Feature Set B]

Choose one of the following options:

Option 1: Assemble the URL

The URL pattern is `https://<subdomain>.authentication.<btp-XSUAH-host>/passcode`.

1. Get the SAP BTP region host, for example, `eu10.hana.ondemand.com`.
2. Assemble the final URL to be used, in this case: `https://mysubdomain.authentication.eu10.hana.ondemand.com/passcode`.

Option 2: Get the URL Using the Connectivity Service Instance Credentials

1. Choose one of these steps to obtain the URL:
 - [Create and Bind a Connectivity Service Instance \[page 160\]](#)
 - [Create a service key ↗](#)
2. Get the value of the `token_service_url` attribute.
3. Append `/passcode` at the end of the obtained URL.

Next Step

[Get the One-Time Passcode \[page 338\]](#)

Back to [Content \[page 334\]](#)

Get the One-Time Passcode

1. Open the resulting URL in your browser to get the one-time passcode via SSO:
 - If *there is* an active user session, the passcode is generated automatically and returned right away.
 - If there is *no* active user session, you are asked to log on to the IDP manually. If several IDPs are configured, you can choose one from the available options.
2. Use the passcode to proceed with the subaccount configuration in the Cloud Connector UI.

Back to [Content \[page 334\]](#)

1.2.2.2 Managing Subaccounts

Add and connect your SAP BTP subaccounts to the Cloud Connector.

i Note

This topic refers to subaccount management in the Cloud Connector. If you are looking for information about managing subaccounts on SAP BTP (Cloud Foundry or Neo environment), see

- [Account Administration \(Cloud Foundry environment\)](#)
- [Administration and Operations, Neo Environment](#)

Context

As of version **2.2**, you can connect to several subaccounts within a single Cloud Connector installation. Those subaccounts can use the Cloud Connector concurrently with different configurations. By selecting a subaccount from the drop-down box, all tab entries show the configuration, audit, and state, specific to this subaccount. In case of audit and traces, cross-subaccount info is merged with the subaccount-specific parts of the UI.

i Note

We recommend that you group only subaccounts with the same qualities in a single installation:

- Productive subaccounts should reside on a Cloud Connector that is used for productive subaccounts only.
- Test and development subaccounts can be merged, depending on the group of users that are supposed to deal with those subaccounts. However, the preferred logical setup is to have separate development and test installations.

Prerequisites

You have assigned one of these roles/role collections to the subaccount user that you use for initial Cloud Connector setup, depending on the SAP BTP environment in which your subaccount is running:

i Note

For the **Cloud Foundry** environment, you must know on which cloud management tools feature set (A or B) your account is running. For more information on feature sets, see [Cloud Management Tools — Feature Set Overview](#).

Environment	Required Roles/Role Collections	More Information
Cloud Foundry [feature set A]	<p>The user must be a <i>member of the global account</i> that the subaccount belongs to.</p> <p>Alternatively, you can assign the user as <i>Security Administrator</i>.</p>	Add Members to Your Global Account Managing Security Administrators in Your Subaccount [Feature Set A]
Cloud Foundry [feature set B]	<p>Assign at least one of these <i>default role collections</i> (all of them including the role Cloud Connector Administrator):</p> <ul style="list-style-type: none"> • Subaccount Administrator • Cloud Connector Administrator • Connectivity and Destination Administrator <p>Alternatively, you can assign a <i>custom role collection</i> to the user that includes the role Cloud Connector Administrator.</p>	Default Role Collections [Feature Set B] [page 13] Role Collections and Roles in Global Accounts, Directories, and Subaccounts [Feature Set B]
Neo	<p>Assign at least one of these <i>default roles</i>:</p> <ul style="list-style-type: none"> • Cloud Connector Admin • Administrator <p>Alternatively, you can assign a <i>custom role</i> to the user that includes the permission <code>manageSCCTunnels</code>.</p>	Managing Member Authorizations in the Neo Environment

After establishing the Cloud Connector connection, this user is not needed any more, since it serves only for initial connection setup. You may revoke the corresponding role assignment then and remove the user from the [Members](#) list.

i Note

If the Cloud Connector is installed in an environment that is operated by SAP, SAP provides a user that you can add as member in your SAP BTP subaccount and assign the required role.

Subaccount Dashboard

In the subaccount dashboard (choose your [Subaccount](#) from the main menu), you can check the state of all subaccount connections managed by this Cloud Connector at a glance.

The screenshot shows the SAP Cloud Connector Administration interface. On the left, there's a sidebar with navigation links like Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, and Log And Trace Files. The main area is titled 'Subaccount: conn2'. It displays 'Connector Overview' with details: Connector ID: ABCDEFG1234567890, Local Name: myname.company.org, Local IP: 11.22.33.44. It also shows security status (Risk), high availability (Disabled), and alerts (1). Below this is a 'Subaccount Dashboard' table:

Status	Subaccount	Display Name	Location ID	Region	Actions
Green	conn2	conn2	nirvana	Staging	Disconnect Edit View More
Green	conn3	conn3		Staging	Disconnect Edit View More
Orange	Anonymous	Trial		Europe (Rot) - Trial	Disconnect Edit View More

In the screenshot above, the test1 subaccount is already connected, but has no active resources exposed. The test2 subaccount is currently disconnected.

The dashboard also lets you disconnect or connect the subaccounts by choosing the respective button in the *Actions* column.

If you want to connect an additional subaccount to your on-premise landscape, choose the *Add Subaccount* button. A dialog appears, which is similar to the *Initial Configuration* operation when establishing the first connection.

Add Subaccount

The dialog box for adding a subaccount has the following fields:

- Region:**
- Subaccount:**
- Display Name:**
- Subaccount User:**
- Password:**
- Location ID:** *(Enter location ID to overwrite default)*
- Description:**

At the bottom right are two buttons: **Save** and **Cancel**.

Procedure

1. The `<Region>` field specifies the SAP BTP region that should be used, for example, **Europe (Rot)**. Choose the one you need from the drop-down list.

i Note

You can also configure a region yourself, if it is not part of the standard list. Either insert the region host manually, or create a custom region, as described in [Configure Custom Regions \[page 349\]](#).

2. For `<Subaccount>` and `<Subaccount User>` (user/password), enter the values you obtained when you registered your account on SAP BTP.

i Note

If your subaccount is on **Cloud Foundry**, you must enter the subaccount **ID** as `<Subaccount>`, rather than its actual (technical) name. For information on getting the subaccount ID, see [Find Your Subaccount ID \(Cloud Foundry Environment\) \[page 348\]](#). As `<Subaccount User>` you must provide your **Login E-mail** instead of a user ID.

For the **Neo** environment, enter the subaccount's **technical name** in the field `<Subaccount>`, not the subaccount ID.

Alternatively, you can add a new subaccount user in the SAP BTP cockpit, assign the required authorization (see section **Prerequisites** above), and use the new user and password.

→ Tip

When using SAP Cloud Identity Services - Identity Authentication (IAS) as platform identity provider with two-factor authentication for your subaccount, you can simply append the required token to the regular password.

→ Tip

For a subaccount in the **Cloud Foundry** environment, the Cloud Connector supports the use of a custom identity provider (IDP) via single sign-on (SSO) passcode. For more information, see [Use a Custom IDP for Subaccount Configuration \[page 334\]](#).

3. (Optional) You can define a `<Display Name>` that allows you to easily recognize a specific subaccount in the UI compared to the technical subaccount name.
4. (Optional) You can define a `<Location ID>` that identifies the location of this Cloud Connector for a specific subaccount. As of Cloud Connector release 2.9.0, the location ID is used as routing information and therefore you can connect multiple Cloud Connectors to a single subaccount. If you don't specify any value for `<Location ID>`, the default is used, which represents the behavior of previous Cloud Connector versions. The location ID must be unique per subaccount and should be an identifier that can be used in a URI. To route requests to a Cloud Connector with a location ID, the location ID must be configured in the respective destinations.
5. (Optional) You can provide a `<Description>` of the subaccount that is shown when clicking on the **Details** icon in the **Actions** column.
6. Choose **Save**.

Next Steps

- To modify an existing subaccount, choose the *Edit* icon and change the <Display Name>, <Location ID> and/or <Description>.

Edit Subaccount

The screenshot shows a dialog box titled "Edit Subaccount". It contains three input fields: "Display Name" with the value "test", "Location ID" with the placeholder "Enter location ID to overwrite default", and "Description" which is empty. At the bottom right are two buttons: "Save" (in a blue box) and "Cancel".

- You can also delete a subaccount from the list of connections. The subaccount will be disconnected and all configurations will be removed from the installation.

Related Information

[Managing Member Authorizations in the Neo Environment](#)

[Copy a Subaccount Configuration \[page 343\]](#)

[Update the Certificate for a Subaccount \[page 345\]](#)

[Configure a Disaster Recovery Subaccount \[page 346\]](#)

[Find Your Subaccount ID \(Cloud Foundry Environment\) \[page 348\]](#)

[Configure Custom Regions \[page 349\]](#)

1.2.2.2.1 Copy a Subaccount Configuration

Copy an existing subaccount configuration in the Cloud Connector to another subaccount.

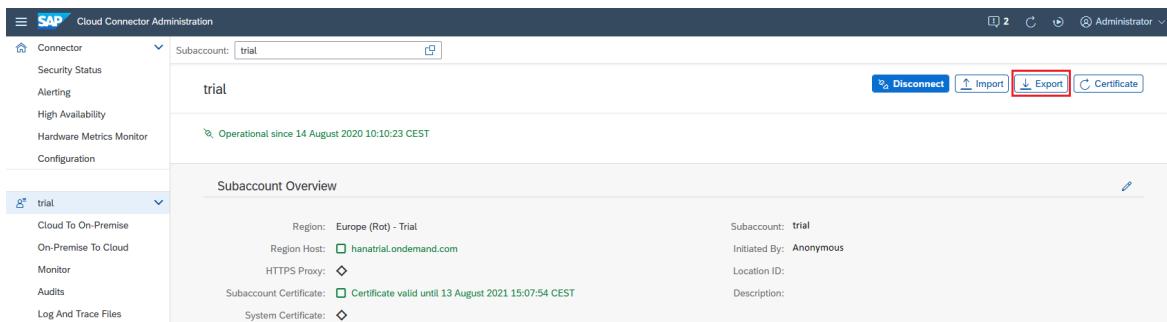
You can copy the configuration of a subaccount's *Cloud To On-Premise* and *On-Premise To Coud* sections to a new subaccount, by using the export and import functions in the Cloud Connector administration UI.

i Note

Principal propagation configuration (section *Cloud To On-Premise*) is not exported or imported, since it contains subaccount-specific data.

Procedure: Export an Existing Configuration

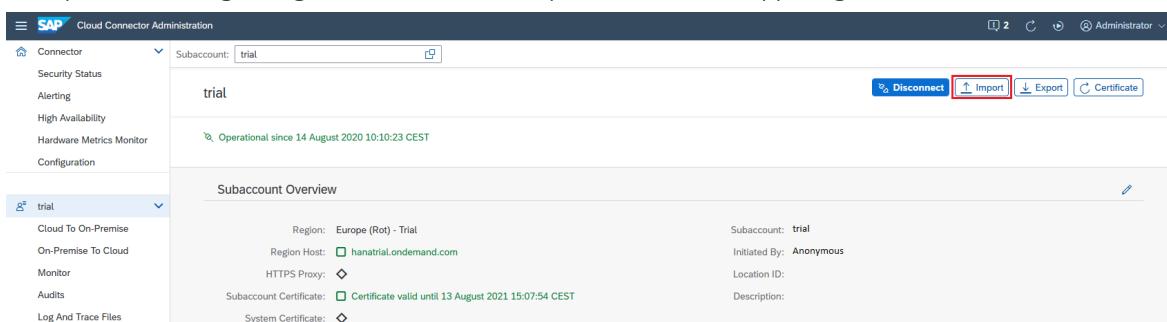
1. In the Cloud Connector administration UI, choose your subaccount from the navigation menu.
2. To export the existing configuration, choose the *Export* button in the upper right corner. The configuration is downloaded as a zip file to your local file system.



The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar shows a navigation menu with options like Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, and Configuration. Under Configuration, 'trial' is selected. The main content area displays 'Subaccount Overview' for 'trial'. It shows details such as Region: Europe (Rot) - Trial, Region Host: hanatrial.ondemand.com, and Subaccount Certificate: Certificate valid until 13 August 2021 15:07:54 CEST. At the top right, there are buttons for Disconnect, Import (highlighted with a red box), Export (also highlighted with a red box), and Certificate.

Procedure: Import an Existing Configuration

1. From the navigation menu, choose the subaccount to which you want to copy an existing configuration.
2. To import an existing configuration, choose the *Import* button in the upper right corner.



This screenshot is identical to the one above, showing the SAP Cloud Connector Administration interface with the 'Subaccount Overview' page for subaccount 'trial'. The 'Import' button in the top right corner is highlighted with a red box.

3. Select one of the following sources:
 1. *File*, if you want to copy the configuration from a previously downloaded zip file.
 2. *Subaccount*, if you want to copy the configuration directly from another existing subaccount.

Import Account Configuration

Source: File Subaccount

File:

1.2.2.2.2 Update the Certificate for a Subaccount

Certificates used by the Cloud Connector are issued with a limited validity period. To prevent a downtime while refreshing the certificate, you can update it for your subaccount directly from the administration UI.

Prerequisites

You must have the required subaccount authorizations on SAP BTP to update certificates for your subaccount.

See:

- [Connectivity: Technical Roles \(Cloud Foundry environment\)](#)
- [Connectivity: User Roles \(Neo environment\)](#)

Procedure

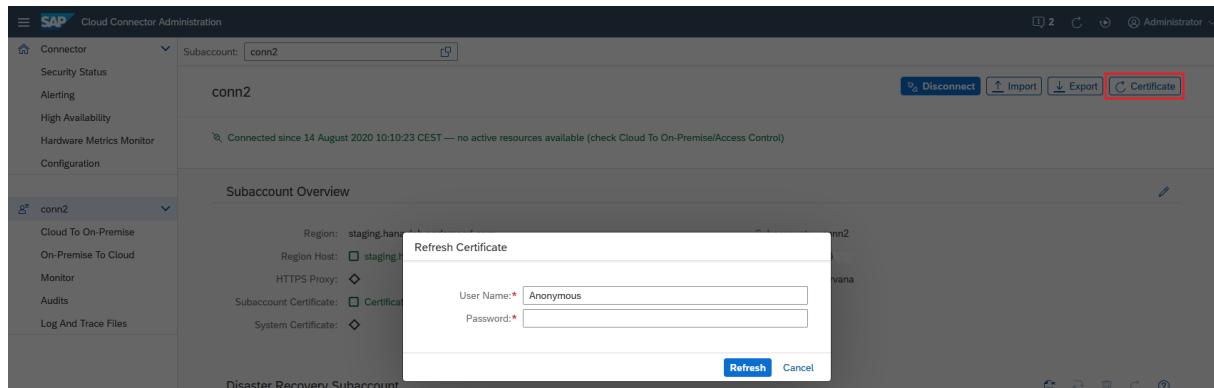
Proceed as follows to update your subaccount certificate:

1. From the main menu, choose your subaccount.
2. Choose the *Certificate* button. A dialog opens, requesting a user name and password.
3. Enter `<User Name>` and `<Password>` and choose *OK*. The certificate assigned to your subaccount is refreshed.

i Note

In the **Cloud Foundry** environment, you must provide your **Login E-mail** instead of a user ID as `<User Name>`.

4. If you have configured a disaster recovery subaccount, go to section *Disaster Recovery Subaccount* below and choose *Refresh Disaster Recovery Certificate*.
5. Enter <User Name> and <Password> as in step 3 and choose *OK*.



1.2.2.2.3 Configure a Disaster Recovery Subaccount

Configure a subaccount as backup for disaster recovery.

Each subaccount (except trial accounts) can optionally have a disaster recovery subaccount.

Prerequisite is that you are using the enhanced disaster recovery, see [What is Enhanced Disaster Recovery](#).

The disaster recovery subaccount is intended to take over if the region host of its associated original subaccount faces severe issues.

A disaster recovery account inherits the configuration from its original subaccount except for the region host. The user can, but does not have to be the same.

Procedure

1. From the main menu, choose your subaccount.
2. In section *Disaster Recovery Subaccount*, choose *Configure disaster recovery subaccount*.
3. In the configuration dialog, select an appropriate <Region Host> from the drop-down list.

i Note

The selected region host must be different from the region host of the original subaccount.

4. (Optional) You can adjust the <Subaccount User>.
5. Enter the <Password> for the subaccount user.
6. If configured, enter a <Location ID>.
7. Choose *Save*.

i Note

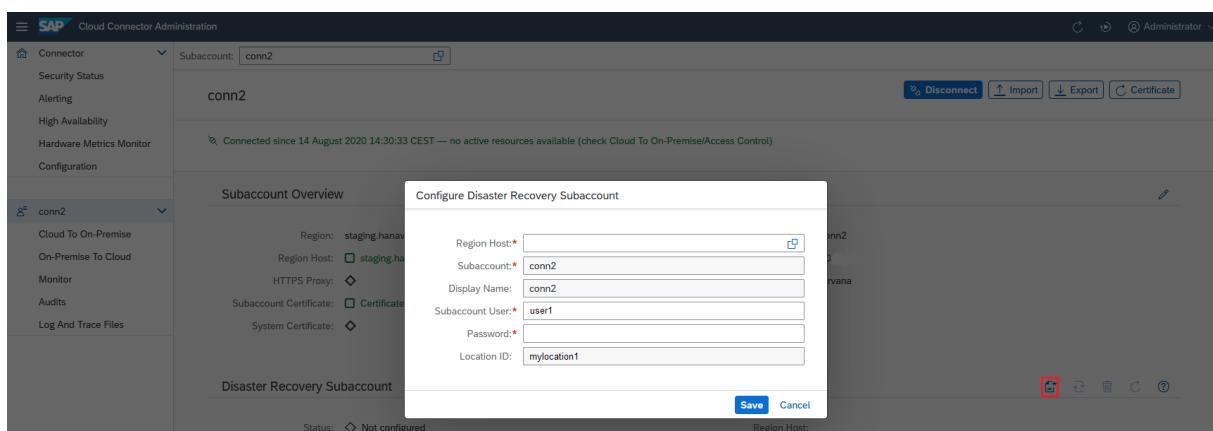
The technical subaccount name, the display name, and the location ID must remain the same. They are set automatically and cannot be changed.

i Note

You cannot choose another original subaccount nor a trial subaccount to become a disaster recovery subaccount.

i Note

If you want to change a disaster recovery subaccount, you must delete it first and then configure it again.



To switch from the original subaccount to the disaster recovery subaccount, choose [Employ disaster recovery subaccount](#).

The disaster recovery subaccount then becomes active, and the original subaccount is deactivated.

You can switch back to the original subaccount as soon as it is available again.

i Note

As of Cloud Connector 2.11, the cloud side informs about a disaster by issuing an event. In this case, the switch is performed automatically.

Related Information

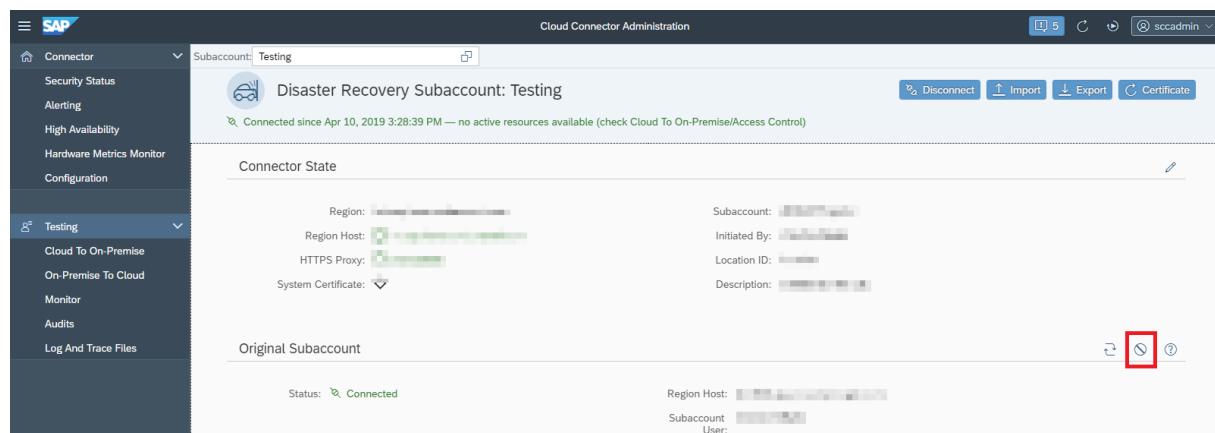
[Convert a Disaster Recovery Subaccount into a Standard Subaccount \[page 348\]](#)

1.2.2.2.3.1 Convert a Disaster Recovery Subaccount into a Standard Subaccount

Convert a disaster recovery subaccount into a standard subaccount if the former primary subaccount's region cannot be recovered.

Disaster recovery subaccounts that were switched to disaster recovery mode can be elevated to standard subaccounts if a disaster recovery region replaces an original region that is not expected to recover.

If a disaster recovery subaccount should be used as primary subaccount, you can convert it by choosing the button *Discard original subaccount and replace it with disaster recovery subaccount*.



1.2.2.2.4 Find Your Subaccount ID (Cloud Foundry Environment)

Get your subaccount ID to configure the Cloud Connector in the Cloud Foundry environment.

i Note

For the Beta version, the cloud cockpit is not yet available.

In order to set up your subaccount in the Cloud Connector, you must know the subaccount ID. Follow these steps to acquire it:

1. Open the SAP BTP cockpit.
2. Navigate to the subaccount list of the global account containing your subaccount: choose **Home > <Your Global Account> > Account Explorer**.
3. Find your subaccount in the list.
4. Choose the *Info* icon in the subaccount tile to display the subaccount ID:

The screenshot shows the SAP BTP Cockpit interface with three entries in a grid:

- TEST_CF_AWS_US**: Provider: Amazon Web Services (AWS), Region: US East (VA) - Staging, Description: test, Environment: Multi-Environment.
- TEST_CF_AWS_US_2**: Provider: Amazon Web Services (AWS), Region: US East (VA) - Staging, Description: test, Environment: Multi-Environment.
- TEST_CF**: Provider: Amazon Web Services (AWS), Region: Europe (Frankfurt), Description: -none-, Environment: Multi-Environment.

The edit icon for the TEST_CF entry is highlighted with a red box.

1.2.2.2.5 Configure Custom Regions

Configure regions that are not available in the standard selection.

If you want to use a custom region for your subaccount, you can configure regions in the Cloud Connector, which are not listed in the selection of standard regions.

To add a custom region, do the following:

- From the Cloud Connector main menu, choose **Configuration** and go to the *Custom Regions* section.
- To add a region to the list, choose the *Add* icon.

The screenshot shows the Cloud Connector Administration interface with the following details:

- Subaccount:** trial
- Configuration** section selected.
- CLOUD** tab selected.
- Custom Regions (2)** table:

Region	Region Host	Actions
<Custom Region 1>	<Region Host 1>	
<Custom Region 2>	<Region Host 2>	

- In the *Add Region* dialog, enter the *<Region>* and *<Region Host>* you want to use.
- Choose *Save*.
- To edit a region from the list, select the corresponding line and choose the *Edit* icon.

1.2.2.3 Authenticating Users against On-Premise Systems

Authentication types supported by the Cloud Connector.

Currently, the Cloud Connector supports **basic authentication** and **principal propagation** (user propagation) as user authentication types towards internal systems. The destination configuration of the used cloud application defines which of these types is used for the actual communication to an on-premise system through the Cloud Connector, see [Managing Destinations \[page 39\]](#).

- To use **basic authentication**, configure an on-premise system to accept basic authentication and to provide one or multiple service users. No additional steps are necessary in the Cloud Connector for this authentication type.

- To use **principal propagation**, you must explicitly configure trust to those cloud entities from which user tokens are accepted as valid. You can do this in the *Trust* view of the Cloud Connector, see [Set Up Trust for Principal Propagation \[page 351\]](#).

Related Information

[Configuring Principal Propagation \[page 350\]](#)

1.2.2.3.1 Configuring Principal Propagation

Use principal propagation to simplify the access of SAP BTP users to on-premise systems.

Tasks in this section:

Task	Description
Set Up Trust for Principal Propagation [page 351]	Configure a trusted relationship in the Cloud Connector to support principal propagation. Principal propagation lets you forward the logged-on identity in the cloud to the internal system without requesting a password.
Configure a CA Certificate for Principal Propagation [page 354]	Install and configure an X.509 certificate to enable support for principal propagation.
Configuring Principal Propagation to an ABAP System [page 357]	Learn more about the different types of configuring and supporting principal propagation for a particular AS ABAP.
Configure a Subject Pattern for Principal Propagation [page 369]	Define a pattern identifying the user for the subject of the generated short-lived X.509 certificate, as well as its validity period.
Configure a Secure Login Server [page 371]	Configuration steps for Java Secure Login Server (SLS) support.
Configure Kerberos [page 375]	The Cloud Connector lets you propagate users authenticated in SAP BTP via Kerberos against back-end systems. It uses the Service For User and Constrained Delegation protocol extension of Kerberos.
Configuring Principal Propagation to SAP NetWeaver AS for Java [page 377]	Find step-by-step instructions on how to configure principal propagation to an application server Java (AS Java).

Related Information

[Principal Propagation \[page 122\]](#) (Cloud Foundry environment)

[Principal Propagation](#) (Neo environment)

1.2.2.3.1.1 Set Up Trust for Principal Propagation

Establish trust to an identity provider to support principal propagation.

Tasks

[Configure Trusted Entities in the Cloud Connector \[page 351\]](#)

[Configure an On-Premise System for Principal Propagation \[page 352\]](#)

[Trust Cloud Applications in the Cloud Connector \[page 353\]](#)

[Set up a Trust Store \[page 353\]](#)

Configure Trusted Entities in the Cloud Connector

You perform trust configuration to support principal propagation. By default, your Cloud Connector does not trust any entity that issues tokens for principal propagation. Therefore, the list of trusted identity providers is empty by default. If you decide to use the principal propagation feature, you must establish trust to at least one identity provider. Currently, SAML2 identity providers are supported. You can configure trust to one or more SAML2 IdPs per subaccount. After you've configured trust in the cockpit for your subaccount, for example, to your own company's identity provider(s), you can synchronize this list with your Cloud Connector.

As of Cloud Connector **2 . 4**, you can also trust HANA instances and Java applications to act as identity providers.

Name	Description	Type	Trusted	Actions
	Synchronize to obtain a list of IdPs or applications			

From your subaccount menu, choose [Cloud to On-Premise](#) and go to the [Principal Propagation](#) tab. Choose the [Synchronize](#) button to store the list of existing identity providers locally in your Cloud Connector.

Select an entry to see its details:

- **Name:** the name associated with the identity provider.
- **Description:** descriptive information about this entry.
- **Type:** type of the trusted entity.
- **Trusted:** indicates whether the entry is trusted for principal propagation.
- **Actions:** Choose the *Show Certificate Information* icon to display detail information for the corresponding entry. The Cloud Connector runtime will use the certificate associated with the entry to verify that the assertion used for principal propagation was issued by a trusted entity.

You can decide for each entry, whether to trust it for the principal propagation use case by choosing *Edit* and (de)selecting the *Trusted* checkbox.

i Note

Whenever you update the SAML IdP configuration for a subaccount on cloud side, you must synchronize the trusted entities in theCloud Connector. Otherwise the validation of the forwarded SAML assertion will fail with an exception containing an exception message similar to this: *Caused by:*
`com.sap.engine.lib.xml.signature.SignatureException: Unable to validate signature ->`
`java.security.SignatureException: Signature decryption error: javax.crypto.BadPaddingException: Invalid`
`PKCS#1 padding: encrypted message and modulus lengths do not match!.`

Back to [Tasks \[page 351\]](#)

Configure an On-Premise System for Principal Propagation

Set up principal propagation from SAP BTP to your internal system that is used in a hybrid scenario.

i Note

As a prerequisite for principal propagation for RFC, the following cloud application runtime versions are required:

- for Java Web: **1.51.8** or higher
- for Java EE 6 Web Profile: **2.31.11** or higher
- other runtimes support it with any version

1. Set up trust to an entity that is issuing an assertion for the logged-on user (see section above).
 2. Set up the system identity for the Cloud Connector.
 - For HTTPS, you must import a system certificate into your Cloud Connector.
 - For RFC, you must import an SNC PSE into your Cloud Connector.
 3. Configure the target system to trust the Cloud Connector.
- There are two levels of trust:
1. First, you must allow the Cloud Connector to identify itself with its system certificate (for HTTPS), or with the SNC PSE (for RFC).
 2. Then, you must allow this identity to propagate the user accordingly:
 - For HTTPS, the Cloud Connector forwards the true identity in a short-lived x.509 certificate in an HTTP header named **SSL_CLIENT_CERT**. The system must use this certificate for logging on the

real user. The SSL handshake, however, is performed through the system certificate. For more information on identity forwarding, see [Configure Access Control \(HTTP\) \[page 380\]](#).

- For RFC, the Cloud Connector forwards the true identity as part of the RFC protocol.

For more information, see [Configuring Principal Propagation to an ABAP System \[page 357\]](#).

4. Configure the user mapping in the target system. The x.509 certificate contains information about the cloud user in its subject. Use this information to map the identity to the appropriate user in this system. This step applies for both HTTPS and RFC.

i Note

If you have the following scenario: *Application1->AppToAppSSO->Application2->Principal Propagation->On-premise Backend System* you must mark *Application2* as **trusted** by the Cloud Connector in tab *Principal Propagation*, section *Trust Configuration*.

If you use an identity provider that issues unsigned assertions, you must mark all relevant applications as **trusted** by the Cloud Connector in tab *Principal Propagation*, section *Trust Configuration*.

Back to [Tasks \[page 351\]](#)

Trust Cloud Applications in the Cloud Connector

Configure an allowlist for trusted cloud applications, see [Configure Trust \[page 500\]](#).

Back to [Tasks \[page 351\]](#)

Set up a Trust Store

Configure a trust store that acts as an allowlist for trusted on-premise systems. See [Configure Trust \[page 500\]](#).

Back to [Tasks \[page 351\]](#)

Related Information

[Principal Propagation \[page 122\]](#) (Cloud Foundry)

[Principal Propagation \(Neo\)](#)

1.2.2.3.1.2 Configure a CA Certificate for Principal Propagation

Install and configure an X.509 certificate to enable support for principal propagation in the Cloud Connector.

Supported CA Mechanisms

You can enable support for principal propagation with X.509 certificates by performing either of the following procedures:

- Using a *Local CA* in the Cloud Connector.

i Note

Prior to version 2.7.0, this was the only option and the system certificate was acting both as client certificate and CA certificate in the context of principal propagation.

- Using a *Secure Login Server* and delegate the CA functionality to it.

The Cloud Connector uses the configured CA approach to issue short-lived certificates for logging on the same identity in the back end that is logged on in the cloud. For establishing trust with the back end, the respective configuration steps are independent of the approach that you choose for the CA.

Install a local CA Certificate

To issue short-lived certificates that are used for principal propagation to a back-end system, you can import an X.509 client certificate into the Cloud Connector. This CA certificate must be provided as *PKCS#12* file containing the (intermediate) certificate, the corresponding private key, and the CA root certificate that signed the intermediate certificate (plus the certificates of any other intermediate CAs, if the certificate chain includes more than those two certificates).

Use either of the following options to install a local CA certificate:

- Option 1: Choose the PKCS#12 file from the file system, using the file upload dialog. For the import process, you must also provide the file password.
- Option 2: Start a *Certificate Signing Request* (CSR) procedure like for the UI certificate, see [Recommended: Exchange UI Certificates in the Administration UI \[page 313\]](#).
- Option 3: (As of version 2.10) Generate a self-signed certificate, which might be useful in a demo setup or if you need a dedicated CA. In particular for this option, it is useful to export the public key of the CA via the button [Download certificate in DER format](#).

i Note

The CA certificate should have the `KeyUsage` attribute `keyCertSign`. Many systems verify that the issuer of a certificate includes this attribute and deny a client certificate without this attribute. When using the CSR procedure, the attribute is requested for the CA certificate. Also, when generating a self-signed certificate, this attribute is added automatically.

Choose [Import a certificate](#) for option 1:

The screenshot shows the SAP Cloud Connector Administration interface. In the left sidebar, under 'Configuration', there is a section for 'Cloud To On-Premise' with options like 'On-Premise To Cloud', 'Monitor', 'Audits', and 'Log And Trace Files'. The main area is titled 'Configuration' and has tabs for 'USER INTERFACE', 'CLOUD', 'ON PREMISE' (which is selected), 'REPORTING', and 'ADVANCED'. A subaccount 'conn2' is selected. A modal dialog titled 'Import CA Certificate' is open. It asks for 'Certificate Type: Local CA' and 'Subject DN:'. Under 'Issuer:', there are fields for 'Valid From:' and 'Valid To:'. The 'P12 Certificate:' field contains a file path, and the 'Password:' field is filled with 'password'. At the bottom are 'Import' and 'Cancel' buttons.

Choose [Generate a certificate signing request](#) for option 2:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar includes 'Configuration' and 'Cloud To On-Premise' sections. The main area has tabs for 'USER INTERFACE', 'CLOUD', 'ON PREMISE' (selected), 'REPORTING', and 'ADVANCED'. A subaccount 'conn2' is selected. A modal dialog titled 'Generate CSR' is open. It has a 'Subject DN' section with fields for 'Common Name (CN)*', 'E-Mail Address (EMAIL)*', 'Locality (L)*', 'Organizational Unit (OU)*', 'Organization (O)*', 'State or Province (ST)*', and 'Country (C)*'. Below it is a 'Subject Alternative Names' table with columns 'Type', 'Value', and 'Actions'. At the bottom are 'Generate' and 'Cancel' buttons.

Choose [Create and import a self-signed certificate](#) if you want to use option 3:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar includes 'Configuration' and 'Cloud To On-Premise' sections. The main area has tabs for 'USER INTERFACE', 'CLOUD', 'ON PREMISE' (selected), 'REPORTING', and 'ADVANCED'. A subaccount 'conn2' is selected. A modal dialog titled 'Create Self-Signed Certificate' is open. It has a 'Subject DN' section with fields for 'Common Name (CN)*', 'E-Mail Address (EMAIL)*', 'Locality (L)*', 'Organizational Unit (OU)*', 'Organization (O)*', 'State or Province (ST)*', and 'Country (C)*'. Below it is a 'Subject Alternative Names' table with columns 'Type', 'Value', and 'Actions'. At the bottom are 'Create' and 'Cancel' buttons.

After successful import of the CA certificate, its distinguished name, the name of the issuer, and the validity dates are shown:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has 'Subaccount: conn2' selected. The main area is titled 'Configuration' with tabs: USER INTERFACE, CLOUD, ON PREMISE (selected), REPORTING, ADVANCED. Under 'ON PREMISE', there's a 'CA Certificate' section. It shows a certificate entry: 'Certificate Type: Local CA', 'Subject DN: CN=test,C=DE', 'Issuer: CN=test,C=DE', 'Valid From: 14 August 2020 17:57:49 CEST', and 'Valid To: 14 August 2021 18:07:49 CEST'. To the right is a table for 'Subject Alternative Names' with one row: 'Type Value' and 'No data'. Below the table are several icons for file operations.

If a CA certificate is no longer required, you can delete it. Use the respective **Delete** button and confirm the deletion.

Configure a CA Hosted by a Secure Login Server

If you want to delegate the CA functionality to a Secure Login Server, choose the CA using [Secure Login Server](#) option and configure the Secure Login Server as follows, after having configured the Secure Login server as described in [Configure a Secure Login Server \[page 371\]](#).

The screenshot shows the SAP Cloud Connector Administration interface with 'Subaccount: conn2' selected. The 'ON PREMISE' tab is selected. A modal dialog box is open titled 'Configure CA Via Secure Login Server'. It contains fields for 'Host Name': 'sis.mycompany.corp', 'Profiles Port': '50143', and 'Authentication Port': '50101'. At the bottom of the dialog are 'Previous', 'Next', and 'Cancel' buttons. The background shows the 'CA Certificate' configuration from the previous screenshot.

Enter the following:

- <Host Name>: The host, on which your Secure Login Server (SLS) is installed.
- <Profiles Port>: The profiles port must be provided only when your Secure Login Server is configured to not allow to fetch profiles via the privileged authentication port. In this case, you can provide here the port that is configured for that functionality.
- <Authentication Port>: The port, over which the Cloud Connector is requesting the short-lived certificates from SLS. Choose [Next](#).

i Note

For this privileged port, a client certificate authentication is required, for which the Cloud Connector system certificate is used.

- <Profile>: The Secure Login Server profile that allows to issue certificates as needed for principal propagation with the Cloud Connector.

Choose [Finish](#) to store the configuration.

Related Information

[Configure a Secure Login Server \[page 371\]](#)

[Initial Configuration \(HTTP\) \[page 329\]](#)

[Initial Configuration \(RFC\) \[page 331\]](#)

1.2.2.3.1.3 Configuring Principal Propagation to an ABAP System

Learn more about the different types of configuring and supporting principal propagation for a particular AS ABAP.

Task	Description
Configure Principal Propagation for HTTPS [page 357]	Step-by-step instructions to configure principal propagation to an ABAP server for HTTPS.
Configure Principal Propagation via SAP Web Dispatcher [page 361]	Set up a trust chain to use principal propagation to an ABAP server for HTTPS via SAP Web Dispatcher.
Configure Principal Propagation for RFC [page 364]	Step-by-step instructions to configure principal propagation to an ABAP server for RFC.
Rule-Based Mapping of Certificates [page 368]	Map short-lived certificates to users in the ABAP server.

1.2.2.3.1.3.1 Configure Principal Propagation for HTTPS

Find step-by-step instructions to configure principal propagation to an ABAP server for HTTPS.

Example Data

The following data are used in this example:

- System certificate was issued by: `CN=MyCompany CA, O=Trust Community, C=DE`.
- It has the subject: `CN=SCC, OU=HCP Scenarios, O=Trust Community, C=DE`.
- The short-lived certificate has the subject `CN=P1234567890`, where P1234567890 is the platform user.

Tasks

[Prerequisites \[page 358\]](#)

[Configure an ABAP System to Trust the Cloud Connector's System Certificate \[page 358\]](#)

[Map Short-Lived Certificates to Users \[page 360\]](#)

[Access ICF Services \[page 360\]](#)

Prerequisites

To perform the following steps, you must have the corresponding authorizations in the ABAP system for the transactions mentioned below (administrator role according to your specific authorization management) as well as an administrator user for the Cloud Connector.

[Back to Tasks \[page 357\]](#)

[Back to Example Data \[page 357\]](#)

1. Configure an ABAP System to Trust the Cloud Connector's System Certificate

This step includes two sub-steps:

[Configure the ABAP system to trust the Cloud Connector's system certificate \[page 358\]](#)

[Configure the Internet Communication Manager \(ICM\) to trust the system certificate for principal propagation \[page 359\]](#)

Configure the ABAP system to trust the Cloud Connector's system certificate:

1. Open the *Trust Manager* (transaction STRUST).
2. Double-click the *SSL-Server Standard* folder in the menu tree on the left.
3. In the displayed screen, click the *Import certificate* button.
4. In the dialog window, choose the certificate file representing the public key of the issuer of the system certificate, for example, in DER format. Typically, this is a CA certificate. If you decide to use a self-signed system certificate, it is the system certificate itself.
5. The details of this certificate are shown in the section above. Using the example certificate data, you would see **CN=MyCompany CA, O=Trust Community, C=DE** as subject.
6. If you are sure that you are importing the correct certificate, you can integrate the certificate into the certificate list by choosing the *Add to Certificate List* button.
7. Now, the CA certificate (**CN=MyCompany CA, O=Trust Community, C=DE**) is part of the certificate list.

[Back to Step \[page 358\]](#)

Configure the Internet Communication Manager (ICM) to trust the system certificate for principal propagation:

1. Open the *Profile Editor* (transaction RZ10).
2. Select the profile you want to edit, for example, the DEFAULT profile.
3. Select the *Extended maintenance* radio button and choose the *Change* button.
4. Create the following parameter: `icm/trusted_reverse_proxy_<x> = SUBJECT=<subject>, ISSUER=<issuer>`.
 - Select a free index for `<x>`.
 - `<subject>` is the subject of the system certificate (example data: `CN=SCC, OU=BTP Scenarios, O=Trust Community, C=DE`).
 - `<issuer>` is the issuer of the system certificate (example data: `CN=MyCompany CA, O=Trust Community, C=DE`).
 - Example: `icm/trusted_reverse_proxy_2 = SUBJECT=" CN=SCC, OU=BTP Scenarios, O=Trust Community, C=DE ", ISSUER=" CN=MyCompany CA, O=Trust Community, C=DE "`.

i Note

If your ABAP system uses kernel 7.42 or lower, see SAP note [2052899](#) or set the following two parameters:

- `icm/HTTPS/trust_client_with_issuer`: this is the issuer of the system certificate (example data: `CN=MyCompany CA, O=Trust Community, C=DE`).
- `icm/HTTPS/trust_client_with_subject`: this is the subject of the system certificate (example data: `CN=SCC, OU=HCP Scenarios, O=Trust Community, C=DE`).

⚠ Caution

Make sure that `icm/HTTPS/verify_client` is set to either **1** (request certificate) or **2** (require certificate). If the parameter is set to **0**, trust cannot be established.

5. Save the profile.
6. Open the *ICM Monitor* (transaction SMICM) and restart the ICM by choosing *Administration* *ICM* *Exit Hard* *Global*.
7. Verify that the two profile parameters have been taken over by ICM by choosing *Goto* *Parameters* *Display*.

i Note

If you have an SAP Web Dispatcher installed in front of the ABAP system, trust must be added in its configuration files with the same parameters as for the ICM. Also, you must add the system certificate of the Cloud Connector to the trust list of the Web dispatcher Server PSE. For more information, see [Configure Principal Propagation via SAP Web Dispatcher \[page 361\]](#).

⚠ Caution

When using principal propagation with X.509 certificates, you cannot use the strict mode in certificate block management (transaction code: CRCONFIG) for the CRL checks within profile `SSL_SERVER`.

[Back to Step \[page 358\]](#)

[Back to Tasks \[page 357\]](#)

[Back to Example Data \[page 357\]](#)

2. Map Short-Lived Certificates to Users

For systems later than SAP NetWeaver 7.3 EHP1 (7.31), you can use rule-based certificate mapping, which is the recommended way to create the required user mappings. For more information, see [Rule-Based Mapping of Certificates \[page 368\]](#).

In older releases (for which this feature does not exist yet), you can do this manually in the system as described below, or use an identity management solution generating the mapping table for a more comfortable approach.

1. Open *Assignment of External ID to Users* (transaction EXTID_DN).
2. Switch to the edit mode.
3. Create a new entry. Specify the subject of the certificate as `External_ID`. When using the example data, this is `CN=P1234567890`. In the *User* field, provide the appropriate ABAP user, for example `JOHNSMITH`.
4. Choose *Activate*.
5. Save the mapping.
6. Repeat the previous steps for all users that should be supported for the scenario.

[Back to Tasks \[page 357\]](#)

[Back to Example Data \[page 357\]](#)

3. Access ICF Services

To access the required ICF services for your scenario in the ABAP system, choose one of the following procedures:

- To access ICF services via certificate logon, choose the principal type `x.509 Certificate (general usage)` in the corresponding system mapping. This setting lets you use the system certificate for trust as well as for user authentication. For details, see [Configure Access Control \(HTTP\) \[page 380\]](#), step 7. Additionally, make sure that all required ICF services allow Logon Through SSL Certificate as logon method.
- To access ICF services via the logon method Basic Authentication (logon with user/password) and principal propagation, choose the principal type `x.509 Certificate (strict usage)` in the corresponding system mapping. This setting lets you use the system certificate for trust, but prevents its usage for user authentication. For details, see [Configure Access Control \(HTTP\) \[page 380\]](#), step 7. Additionally, make sure that all required ICF services allow Basic Authentication and Logon Through SSL Certificate as logon methods.

- If some of the ICF services require Basic Authentication, while others should be accessed via system certificate logon, proceed as follows:
 1. In the Cloud Connector system mapping, choose the principal type **X.509 Certificate (general usage)** as described above.
 2. In the ABAP system, choose transaction code SICF and go to [Maintain Services](#).
 3. Select the service that requires Basic Authentication as logon method.
 4. Double-click the service and go to tab *Logon Data*.
 5. Switch to [Alternative Logon Procedure](#) and ensure that the Basic Authentication logon procedure is listed before Logon Through SSL Certificate.

i Note

If you are using SAP Web Dispatcher for communication, you must configure it to forward the SSL certificate to the ABAP backend system, see [Forward SSL Certificates for X.509 Authentication](#) (SAP Web Dispatcher documentation).

Back to [Tasks \[page 357\]](#)

Back to [Example Data \[page 357\]](#)

Related Information

[Configure Principal Propagation via SAP Web Dispatcher \[page 361\]](#)

[Rule-Based Mapping of Certificates \[page 368\]](#)

[Configure a Subject Pattern for Principal Propagation \[page 369\]](#)

[Set Up Trust for Principal Propagation \[page 351\]](#)

[Principal Propagation \[page 122\]](#) (Cloud Foundry environment)

[Principal Propagation](#) (Neo environment)

1.2.2.3.1.3.1.1 Configure Principal Propagation via SAP Web Dispatcher

Set up a trust chain to use principal propagation to an ABAP System for HTTPS via SAP Web Dispatcher.

Concept

If you are using an intermediate SAP Web Dispatcher to connect to your ABAP backend system, you must set up a trust chain between the involved components Cloud Connector, SAP Web Dispatcher, and ABAP backend system.

Before configuring the ABAP system (see [Configure Principal Propagation for HTTPS \[page 357\]](#)), in a first step you must configure SAP Web Dispatcher to accept and forward user principals propagated from a cloud account to an ABAP backend.

To do this, follow the step-by-step instructions below.

Example Data

The following data and setup is used for this scenario:

- System certificate was issued by CN=MyCompany CA, O=Trust Community, C=DE
- It has the subject CN=SCC, OU=BTP Scenarios, O=Trust Community, C=DE

Tasks

- [Prerequisites \[page 362\]](#)
- [Configure the SAP Web Dispatcher to Trust the Cloud Connector's System Certificate \[page 363\]](#)
- [Next Steps \[page 364\]](#)

Prerequisites

- Your SAP Web Dispatcher version is 7.53 or higher. See SAP note [908097](#) for information on recommended SAP Web Dispatcher versions.
- Make sure your SAP Web Dispatcher supports SSL. See [Configure SAP Web Dispatcher to Support SSL](#).
- Ensure that SSL client certificates can be used for authentication in the backend system. See [How to Configure SAP Web Dispatcher to Forward SSL Certificates for X.509 Authentication](#) for step-by-step instructions.

Back to [Tasks \[page 362\]](#)

Back to [Concept \[page 361\]](#)

Configure SAP Web Dispatcher to Trust the Cloud Connector's System Certificate

To allow Cloud Connector client certificates for authentication in the backend system, perform the following two steps:

1. Configure SAP Web Dispatcher to trust the Cloud Connector's system certificate:

1. To import the system certificate to SAP Web Dispatcher, open the SAP Web Dispatcher administration interface in your browser.

i Note

The interface is usually configured on `/sap/wdisp/admin`.

2. In the menu, navigate to **SSL and Trust Configuration** and select **PSE Management**.
3. In the **Manage PSE** section, select **SAPSSLS.pse** from the drop-down list. By default, SAPSSLS.pse contains the server certificate and the list of trusted clients that SAP Web Dispatcher trusts as a server.
4. In the **Trusted Certificates** section, choose **Import Certificate**.
5. Enter the certificate as base64-encoded into the text box. The procedure to export your certificate in such a format is described in [Forward SSL Certificates for X.509 Authentication](#), step 1.

i Note

Typically, this is a CA certificate. If you are using a self-signed system certificate, it's the system certificate itself.

6. Choose **Import**.
 7. The certificate details are now shown in section **Trusted Certificates**.
2. Configure SAP Web Dispatcher to trust the Cloud Connector's system certificate for principal propagation:

- Create or edit the following parameter in SAP Web Dispatcher:

```
icm/trusted_reverse_proxy_<x> = SUBJECT="<subject>", ISSUER="<issuer>"  
○ Select a free index for <x>.   
○ <subject>: Subject of the system certificate (example data: CN=SCC, OU=BTP Scenarios,  
O=Trust Community, C=DE)  
○ <issuer>: Issuer of the system certificate (example data: CN=MyCompany CA, O=Trust  
Community, C=DE)
```

Example: `icm/trusted_reverse_proxy_0 = SUBJECT="CN=SCC, OU=BTP Scenarios,
O=Trust Community, C=DE", ISSUER="CN=MyCompany CA, O=Trust Community, C=DE"`

- **[Deprecated]** Create or edit the following two parameters in SAP Web Dispatcher:

i Note

Use the parameters below (instead of `icm/trusted_reverse_proxy_<x>`) only if your kernel release does not yet support parameter `icm/trusted_reverse_proxy_<x>`.

- `icm/HTTPS/trust_client_with_issuer`: Issuer of the system certificate (example data:
CN=MyCompany CA, O=Trust Community, C=DE)

- `icm/HTTPS/trust_client_with_subject`: Subject of the system certificate (example data:
CN=SCC, OU=BTP Scenarios, O=Trust Community, C=DE)

i Note

Make sure `icm/HTTPS/verify_client` is set to **1** (request certificate) or **2** (require certificate). If set to **0**, trust cannot be established. The default value is **1**, so it is OK if the parameter is not set at all.

Back to [Tasks \[page 362\]](#)

Back to [Concept \[page 361\]](#)

Next Steps

Now you can proceed with:

- Step 1 of the basic principal propagation setup for HTTPS, see [Configure an ABAP System to Trust the Cloud Connector's System Certificate \[page 358\]](#). However, when using SAP Web Dispatcher, the ABAP backend must trust the *SAP Web Dispatcher* instead of the *Cloud Connector*, see [Forward SSL Certificates for X.509 Authentication](#), step 2 for details.

Then perform the remaining steps of the basic principal propagation setup for HTTPS as described here:

- [Map Short-Lived Certificates to Users \[page 360\]](#)
- [Access ICF Services \[page 360\]](#)

Back to [Tasks \[page 362\]](#)

Back to [Concept \[page 361\]](#)

1.2.2.3.1.3.2 Configure Principal Propagation for RFC

Find step-by-step instructions to configure principal propagation to an ABAP server for RFC.

Configuring principal propagation for RFC requires an SNC (Secure Network Communications) connection. To enable SNC, you must configure the ABAP system and the Cloud Connector accordingly.

The following example provides step-by-step instructions for the SNC setup.

i Note

It is important that you use the same SNC implementation on both communication sides. Contact the vendor of your SNC solution to check the compatibility rules.

Example Data

The following data and setup is used:

i Note

The parameters provided in this example are based on an SNC implementation that uses the **SAP Cryptographic Library**. Other vendors' libraries may require different values.

- An SNC identity has been generated and installed on the Cloud Connector host. Generating this identity for the SAP Cryptographic Library is typically done using the tool `SAPGENPSE`. For more information, see [Configuring SNC for SAPCRYPTOLIB Using SAPGENPSE](#).
- The ABAP system is configured properly for SNC.

i Note

For the latest system releases, you can use the SSO wizard to configure SNC (transaction code: `SNCWIZARD`). System prerequisites are described in SAP note [2015966](#).

- The Cloud Connector system identity's SNC name is `p:CN=SCC, OU=SAP CP Scenarios, O=Trust Community, C=DE`.
- The ABAP system's SNC identity name is `p:CN=SID, O=Trust Community, C=DE`. This value can typically be found in the ABAP system instance profile parameter `snc/identity/as` and hence is provided per application server.
- When using the SAP Cryptographic Library, the ABAP system's SNC identity and the Cloud Connector's system identity should be signed by the same CA for mutual authentication.
- The example short-lived certificate has the subject `CN=P1234567`, where `P1234567` is the SAP BTP application user.

Tasks

1. Configure the ABAP System [page 365]
2. Map Short-Lived Certificates to Users [page 366]
3. Configure the Cloud Connector [page 366]

1. Configure the ABAP System to Trust the Cloud Connector's System SNC identity

1. Open the *SNC Access Control List for Systems* (transaction `SNC0`).
2. As the Cloud Connector does not have a system ID, use an arbitrary value for `<System ID>` and enter it together with its SNC name: `p:CN=SCC, OU=SAP CP Scenarios, O=Trust Community, C=DE`.
3. Save the entry and choose the *Details* button.

4. In the next screen, activate the checkboxes for *Entry for RFC activated* and *Entry for certificate activated*.
5. Save your settings.

Back to [Tasks \[page 365\]](#)

Back to [Example Data \[page 365\]](#)

2. Map Short-Lived Certificates to Users

You can do this manually in the system as described below or use an identity management solution for a more comfortable approach. For example, for large numbers of users the rule-based certificate mapping is a good way to save time and effort. See [Rule-Based Certificate Mapping](#).

1. Open [Assignment of External ID to Users](#) (transaction `EXTID_DN`).
2. Switch to the edit mode.
3. Create a new entry. Specify the subject of the certificate as `External_ID`. Using the example data, this is `CN=P1234567`. In the `<User>` field, provide an appropriate ABAP user, for example `JOHNDOE`.
4. Save the mapping.
5. Repeat the previous steps for all users that should be supported for the scenario.

Back to [Tasks \[page 365\]](#)

Back to [Example Data \[page 365\]](#)

3. Configure the Cloud Connector

[Prerequisites \[page 366\]](#)

[Set up the Cloud Connector to Use the SNC Implementation \[page 367\]](#)

[Create an RFC Hostname Mapping \[page 367\]](#)

Prerequisites

- The required security product for the SNC flavor that is used by your ABAP back-end systems, is installed on the Cloud Connector host.
- The Cloud Connector's system SNC identity is associated with the operating system user under which the Cloud Connector process is running.

i Note

SAP note [2642538](#) provides a description how you can associate an SNC identity of the SAP Cryptographic Library with a user running an external program that uses JCo. If you use the SAP Cryptographic Library as SNC implementation, perform the corresponding steps for the Cloud Connector. When using a different product, contact the SNC library vendor for details.

Back to [Step \[page 366\]](#)

Set up the Cloud Connector to Use the SNC Implementation

1. In the Cloud Connector UI, choose *Configuration* from the main menu, select the *On Premise* tab, and go to the *SNC* section.
2. Provide the fully qualified name of the SNC library (the security product's shared library implementing the GSS API), the SNC name of the above system identity, and the desired quality of protection by choosing the *Edit* icon.

For more information, see [Initial Configuration \(RFC\) \[page 331\]](#).

i Note

The example in [Initial Configuration \(RFC\) \[page 331\]](#) shows the library location if you use the SAP Secure Login Client as your SNC security product. In this case (as well as for some other security products), **SNC My Name** is optional, because the security product automatically uses the identity associated with the current operating system user under which the process is running, so you can leave that field empty. (Otherwise, in this example it should be filled with **p:CN=SCC, OU=SAP CP Scenarios, O=Trust Community, C=DE.**)

We recommend that you enter **Maximum Protection** for **<Quality of Protection>**, if your security solution supports it, as it provides the best protection.

3. Choose *Save* and *Close*.

Back to [Step \[page 366\]](#)

Create an RFC Hostname Mapping

1. In the *Access Control* section of the Cloud Connector, create a hostname mapping corresponding to the cloud-side RFC destination. See [Configure Access Control \(RFC\) \[page 387\]](#).
2. Make sure you choose **RFC SNC** as **<Protocol>** and **ABAP System** as **<Back-end Type>**. In the **<SNC Partner Name>** field, enter the ABAP system's SNC identitity name, for example, **p:CN=SID, O=Trust Community, C=DE.**
3. Save your mapping.

Back to [Step \[page 366\]](#)

Back to [Tasks \[page 365\]](#)

Back to [Example Data \[page 365\]](#)

Related Information

[Secure Network Communications \(SNC\)](#)

[Using the SAP Cryptographic Library for SNC](#)

[Rule-Based Mapping of Certificates \[page 368\]](#)

[Principal Propagation \[page 122\]](#) (Cloud Foundry environment)

[Principal Propagation](#) (Neo environment)

1.2.2.3.1.3.3 Rule-Based Mapping of Certificates

Learn how to efficiently map short-lived certificates to users in the ABAP server.

To perform rule-based mapping of certificates in the ABAP server, proceed as follows:

1. Enter a dynamic parameter using transaction RZ11.
 1. Enter the `login/certificate_mapping_rulebased` parameter.
 2. Choose the *Change Value* button.
 3. Enter the value **1**.
 4. Save the value.

i Note

If dynamic parameters are disabled, enter the value using transaction RZ10 and restart the whole ABAP system.

2. Configure rule-based mapping
 1. To create a sample certificate with the Cloud Connector, logon to the Cloud Connector UI and from the main menu, choose *Configuration*. In the *System Certificate* section, enter a sample CN name and download the sample certificate to the *Downloads* folder of your machine.
 2. To import the sample certificate into the ABAP server, choose transaction CERTRULE and select *Import certificate*.

i Note

To access transaction CERTRULE, you need the corresponding authorizations (see: [Assign Authorization Objects for Rule-based Mapping \[page 369\]](#)).

3. To create explicit rule mappings, choose the *Rule* button.
4. Choose *Save*.

i Note

When you save the changes and return to transaction CERTRULE, the sample certificate which you imported in Step 2b will not be saved. This is just a sample editor view to see the sample certificates and mappings.

Related Information

[Rule-Based Certificate Mapping](#)

1.2.2.3.1.3.3.1 Assign Authorization Objects for Rule-based Mapping

Assign authorizations to access transaction CERTRULE.

To access transaction CERTRULE, you need the following authorizations:

- CC control center: System administration (S_RZL_ADM)
 - Activity 03 grants display authorizations.
 - Activity 01 grants change authorizations.
- User Master Maintenance: User Groups (S_USER_GRP)
 - Activity 03 grants display authorizations.
 - Activity 02 grants change authorizations.
 - Class: enter the names of user groups for which the administrator can maintain explicit mappings.

To assign these authorization objects, proceed as follows:

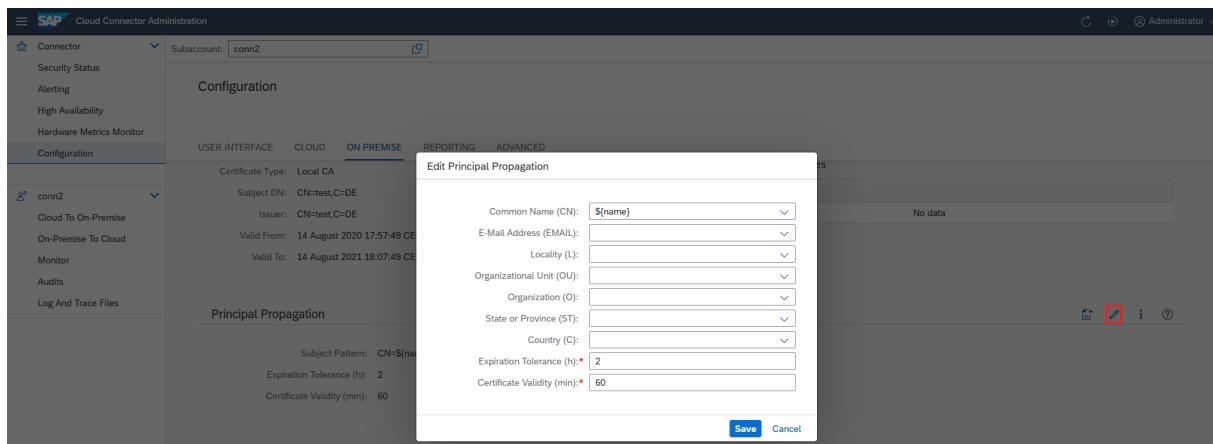
1. Create a *Single Role* using transaction PFCG.
2. To add the authorization objects **S_RZL_ADM** and **S_USER_GRP**, go to the *Authorizations* tab, choose *Change Authorization data* and select the *Manually* button .
3. To generate the profile, choose *Generate* and save the changes.
4. In the *User* tab, enter the user who should execute the transaction CERTRULE.
5. To match the generated profile to the users, choose *User comparison* .

1.2.2.3.1.4 Configure a Subject Pattern for Principal Propagation

Define a pattern identifying the user for the subject of the generated short-lived x.509 certificate, as well as its validity period.

Configure a Subject Pattern

To configure such a pattern, choose  *Configuration*  *On Premise*  and press the *Edit* icon in section *Principal Propagation*:



Subject Pattern Details

Use either of the following procedures to define the subject's distinguished name (DN), for which the certificate will be issued:

- Enter the values in the subject pattern fields manually.
- Use the selection menu of the corresponding field to enter a predefined parameter (constant).

Using the selection menu, you can assign values for the following parameters:

- \${name}
- \${mail}
- \${display_name}
- \${login_name} (as of Cloud Connector version 2.8.1.1)

i Note

If the token provided by the Identity Provider contains additional values that are stored in attributes with different names, but you still want to use it for the subject pattern, you can edit the variable name to place the corresponding attribute value in the subject accordingly. For example, provide \${email}, if a SAML assertion uses email instead of providing mail.

When using a subaccount in the **Cloud Foundry** environment: As of version 2.12.5, the Cloud Connector also offers direct access to custom variables injected in the JWT (JSON Web token) by SAP BTP *Authorization & Trust Management* that were taken over from a SAML assertion.

The values for these variables are provided by the trusted Identity Provider in the token which is passed to the Cloud Connector and specifies the user that has logged on to the cloud application.

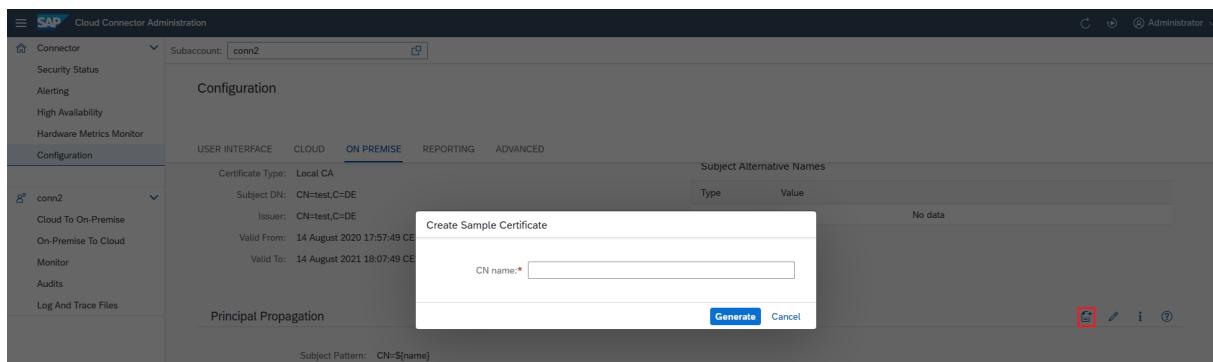
By default, the following attributes are provided:

- <CN>: (common name) – the name of the certificate owner
- <EMAIL>: (e-mail address) - the e-mail address of the certificate owner
- <L>: (locality) – the certificate owner's location
- <O>: (organization) – the certificate owner's organization or company

- <OU>: (name of organizational unit) – the organizational unit to which the certificate owner belongs
- <ST>: (state of residence) – the state in which the certificate issuer resides
- <C>: (country of residence) – the country in which the certificate owner resides
- <Expiration Tolerance (h)>: The length of time in hours, that an application can use a principal issued for a user after the token from cloud side has expired.
- <Certificate Validity (min)>: The length of time in minutes, that a certificate generated for principal propagation can authenticate against the back end. You can reuse a previously generated certificate to improve performance.

Sample Certificate

By choosing *Generate Sample Certificate* you can create a sample certificate that looks like one of the short-lived certificates created at runtime. You can use this certificate to, for example, generate user mapping rules in the target system, via transaction CERTRULE in an ABAP system. If your subject pattern contains variable fields, a wizard lets you provide meaningful values for each of them and eventually you can save the sample certificate in DER format.



Related Information

[Server Certificate Authentication \[page 66\]](#)

1.2.2.3.1.5 Configure a Secure Login Server

Configuration steps for Java SLS support.

Content

[Overview \[page 372\]](#)

[Requirements \[page 373\]](#)

[Implementation \[page 373\]](#)

Overview

The Cloud Connector can use on-the-fly generated X.509 user certificates to log in to on-premise systems if the external user session is authenticated (for example by means of SAML). If you do not want to use the built-in certification authority (CA) functionality of the Cloud Connector (for example because of security considerations), you can connect SAP SSO 2.0 Secure Login Server (SLS).

SLS is a Java application running on AS JAVA 7.20 or higher, which provides interfaces for certificate enrollment.

SLS supports the following formats:

- HTTPS
- REST
- JSON
- PKCS#10/PKCS#7

i Note

Any enrollment requires a successful user or client authentication, which can be a single, multiple or even a multi factor authentication.

The following schemes are supported:

- LDAP/ADS
- RADIUS
- SAP SSO OTP
- ABAP RFC
- Kerberos/SPNego
- X.509 TLS Client Authentication

SLS lets you define arbitrary enrollment profiles, each with a unique profile UID in its URL, and with a configurable authentication and certificate generation.

[Back to Content \[page 372\]](#)

Requirements

For user certification, SLS must provide a profile that adheres to the following:

- Cloud Connector client authentication by its X.509 service certificate
- Cloud Connector service certificate and SLS may live in different PKIs
- Cloud Connector hands over the full user's certificate subject name

With SAP SSO 2.0 SP06, SLS provides the following required features:

- TLS Client Authentication-based enrollment with `SecureLoginModuleUserDelegationWithSSL` (available since SP04)
- multi-PKI support is implemented by all standard components of Application Server (AS) JAVA, AS ABAP, HANA, by importing trusted root CA certificates
- SLS allows `PKCS10:SUBJECT` in a profile's certificate configuration (SP06)

Back to [Content \[page 372\]](#)

Implementation

INSTALLATION

Follow the standard installation procedures for SLS. This includes the initial setup of a PKI (public key infrastructure).

i Note

SLS allows you to set up one or more own PKIs with Root CA, User CA, and so on. You can also import CAs as PKCS#12 file or use a hardware security module (HSM) as "External User CA".

i Note

You should only use HTTPS connections for any communication with SLS. AS JAVA / ICM supports TLS, and the default configuration comes with a self-signed sever certificate. You may use SLS to replace this certificate by a PKI certificate.

CONFIGURATION

SSL Ports

1. Open the NetWeaver Administrator, choose  `Configuration`  and define a new port with **Client Authentication Mode = REQUIRED**.

i Note

You may also define another port with **Client Authentication Mode = Do not request** if you did not do so yet.

2. Import the root CA of the PKI that issued your Cloud Connector service certificate.

- Save the configuration and restart the Internet Communication Manager (ICM).

Authentication Policy

- Open the NetWeaver Administrator (NWA, <https://<host:port>/nwa>).
- From the top-level menu, choose **Configuration** **Authentication and Single Sign-On**.
- In the **Policy Configuration** table, switch to **Type = Custom**.
- Choose **Add** to create a new policy and assign it a name, for example, **SecureLoginCloudConnector**.
- Open **Edit** mode.
- In the **Details** section of the authentication configuration, choose **Authentication Stack** **Login Modules** and add **SecureLoginModuleUserDelegationWithSSL**.
- In **<Rule1.subjectName>** and **<Rule1.issuerName>**, enter the respective certificate names of your Cloud Connector service certificate.
- In the **Details** section of the authentication configuration, choose **Properties** and add the property **UserNameMapping** with value **VirtualUser**.
- Save the policy.

Client Authentication Profile

- Open the SLS Administration Console (SLAC, <https://host:port/slac>).
- From the top-level menu, choose **Profile Management** **Authentication Profiles**.
- Create a new profile with **Client Type = Secure Login Client** and assign it a name, for example, **Cloud Connector User Certificates**.
- Choose **User Authentication** **Use Policy Configuration** and select **Policy Configuration Name = SecureLoginCloudConnector**.
- Edit all required fields in the wizard according to your requirements.
- Save your entries.
- Select the new profile and open **Edit** mode.
- Choose **Certificate Configuration** **Certificate Name and Alternative Names** and set **Appendix Subject Name = (PKCS10:SUBJECT)**.
- Leave all other fields in **Certificate Name and Alternative Names** empty.
- On the **Enrollment Configuration** page, make sure that the **<Enrollment URL>** has the correct value, otherwise edit and fix it:
 - full DNS name
 - port with TLS Client Authentication (see port number in NWA SSL Configuration).
- Save your entries.

User Profile Group

- Open the SLS Administration Console (SLAC, <https://host:port/slac>).
- Go to the top level menu and choose **Profile Management** **User Profile Groups**.
- Create a new profile group, make sure the **<Policy URL>** has the correct value:
 - Full DNS name
 - Port without TLS Client Authentication (see port number in NWA SSL Configuration).
- In tab **Profiles**, add the profile **Cloud Connector User Certificates**.
- Save your entries.

Root CA Certificate

1. Open SLS Administration Console (SLAC, <https://host:port/slac>).
2. Go to the top level menu and choose *Certificate Management*.
3. Select the Root CA certificate you are using in your profile.
4. Choose *Export entry* and download the certificate file.

Cloud Connector

Follow the standard installation procedure of the Cloud Connector and configure SLS support:

1. Enter the policy URL that points to the SLS user profile group.
2. Select the profile, for example, **Cloud Connector User Certificates**.
3. Import the Root CA certificate of SLS into the Cloud Connector's [Trust Store](#).

On-Premise Target Systems

Follow the standard configuration procedure for Cloud Connector support in the corresponding target system and configure SLS support.

To do so, import the Root CA certificate of SLS into the system's truststore:

- **AS ABAP:** choose transaction STRUST and follow the steps in [Maintaining the SSL Server PSE's Certificate List](#).
- **AS Java:** open the Netweaver Administrator and follow the steps described in [Configuring the SSL Key Pair and Trusted X.509 Certificates](#).

Back to [Content \[page 372\]](#)

1.2.2.3.1.6 Configure Kerberos

Context

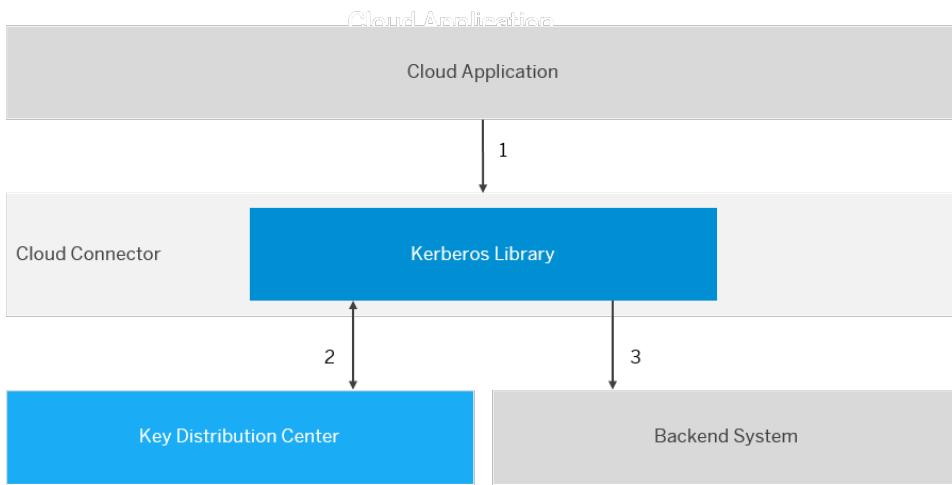
The Cloud Connector allows you to propagate users authenticated in SAP BTP via Kerberos against backend systems. It uses the **Service For User and Constrained Delegation** protocol extension of Kerberos.

i Note

This feature is not supported for ABAP backend systems. In this case, you can use the certificate-based principal propagation, see [Configure a CA Certificate for Principal Propagation \[page 354\]](#).

The Key Distribution Center (KDC) is used for exchanging messages in order to retrieve Kerberos tokens for a certain user and backend system.

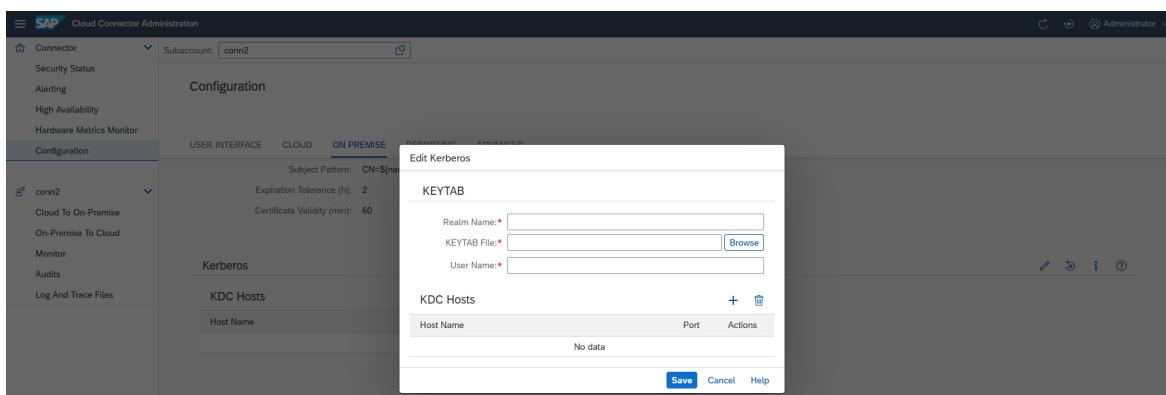
For more information, see [Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol](#).



1. An SAP BTP application calls a backend system via the Cloud Connector.
2. The Cloud Connector calls the KDC to obtain a Kerberos token for the user propagated from the Cloud Connector.
3. The obtained Kerberos token is sent as a credential to the backend system.

Procedure

1. Choose *Configuration* from the main menu.
2. From the *Kerberos* section of the *On Premise* tab, choose *Edit*.



3. Enter the name of your Kerberos realm.
4. Upload a KEYTAB file that contains the secret keys of your service user. The KEYTAB file should contain the **rc4-hmac** key for your user.
5. Enter the name of the service user to be used for communication with the KDC. This user should be allowed to request Kerberos tokens for other users for the backend systems that you are going to access.
6. In the **<KDC Hosts>** field (press **Add** to display the field), enter the host name of your KDC using the format **<host>:<port>**. The port is optional; if you leave it empty, the default, 88, is used.
7. Choose *Save*.

Example

You have a backend system protected with SPNego authentication in your corporate network. You want to call it from a cloud application while preserving the identity of a cloud-authenticated user.

Define the following:

- A connectivity destination in SAP BTP, with `ProxyType = OnPremise`.
- A system mapping made in the Cloud Connector. (Choose *Cloud to On Premise* from your subaccount menu, Go to tab **Access Control** **Add**, and for *Principal Type*, select **Kerberos**.)
- Kerberos configuration in the Cloud Connector, where the service user is allowed to delegate calls for your backend host service.

Result:

When you now call a backend system, the Cloud Connector obtains an SPNego token from your KDC for the cloud-authenticated user. This token is sent along with the request to the back end, so that it can authenticate the user and the identity to be preserved.

Related Information

[Set Up Trust for Principal Propagation \[page 351\]](#)

1.2.2.3.1.7 Configuring Principal Propagation to SAP NetWeaver AS for Java

Find step-by-step instructions on how to set up an application server for Java (AS Java) to enable principal propagation for HTTPS.

Prerequisites

To perform the following steps, you must have the corresponding administrator authorizations in AS Java (SAP NetWeaver Administrator) as well as an administrator user for the Cloud Connector.

Configure AS Java to Trust the Cloud Connector's System Certificate

Procedure

1. Go to  [SAP NetWeaver Administrator](#)  [Certificates and Keys](#)  and import the Cloud Connector's system certificate into the *Trusted CAs* keystore view. See [Importing Certificate and Key From the File System](#).
2. Configure the Internet Communication Manager (ICM) to trust the system certificate for principal propagation.
 - a. Add a new SSL access point. See [Adding New SSL Access Points](#).
 - b. Generate a certificate signing request and send it to the CA of your choice. See [Configuration of the AS Java Keystore Views for SSL](#).
 - c. Import the certificates and save the configuration.

Import the certificate signing response, the root X.509 certificate of the trusted CA, and the Cloud Connector's system certificate into the new SSL access point from step 2a. Save the configuration and restart the ICM. See [Configuring the SSL Key Pair and Trusted X.509 Certificates](#).
 - d. Test the SSL connection. See [Testing the SSL Connection](#).

Define Rules for User Mapping

Procedure

1. Add the *ClientCertLoginModule* to the policy configuration that the Cloud Connector connects to. See [Configuring the Login Module on the AS Java](#).
2. Define the rules to map users authenticated with their certificate to users that exist in the User Management Engine. See [Using Rules for User Mapping in Client Certificate Login Module](#).
 - To map the user ID of the certificate's subject name field to users, see [Using Rules Based on Client Certificate Subject Names](#).
 - To map the user ID based on rules for the certificate V3 extension *SubjectAlternativeName*, see [Using Rules Based on Client Certificate V3 Extensions](#).
 - To use client certificate filters, see [Defining Rules for Filtering Client Certificates](#).

Related Information

[Configuring Transport Layer Security on SAP NetWeaver AS for Java](#)

[Using X.509 Client Certificates](#)

[Configure Principal Propagation for HTTPS \[page 357\]](#)

1.2.2.4 Configure Access Control

Specify the backend systems that can be accessed by your cloud applications.

To allow your cloud applications to access a certain backend system on the intranet, you must specify this system in the Cloud Connector. The procedure is specific to the protocol that you are using for communication.

Find the detailed configuration steps for each communication protocol here:

[Configure Access Control \(HTTP\) \[page 380\]](#)

[Configure Access Control \(RFC\) \[page 387\]](#)

[Configure Access Control \(LDAP\) \[page 393\]](#)

[Configure Access Control \(TCP\) \[page 396\]](#)

Copy Access Control Settings

When you add new subaccounts, you can copy the complete access control settings from another subaccount on the same Cloud Connector. You can also do it any time later by using the import/export mechanism provided by the Cloud Connector.

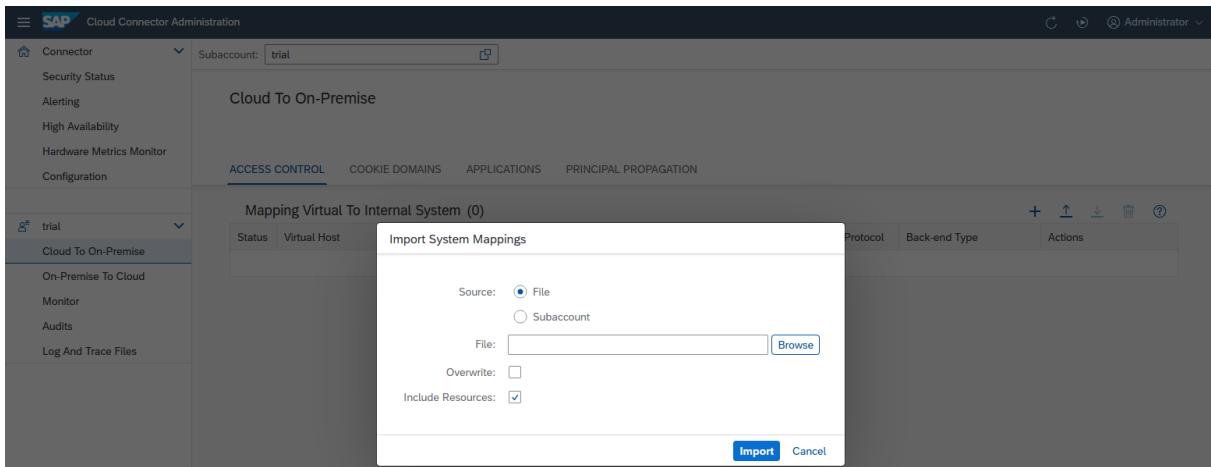
Export Access Control Settings

1. From your subaccount menu, choose [Cloud To On-Premise](#) and select the tab [Access Control](#).
2. To store the current settings in a ZIP file, choose [Download](#) icon in the upper-right corner.
3. You can later import this file into a different Cloud Connector.

Import Access Control Settings

There are two locations from which you can import access control settings:

- A file that was previously exported from a Cloud Connector
- A different subaccount on the same Cloud Connector



Two additional options define the behavior of the import:

- **Overwrite:** Select this checkbox if you want to replace existing system mappings with imported ones. Do not select this checkbox if you want to keep existing mappings and only import the ones that are not yet available (default).

i Note

A system mapping is uniquely identified by the combination of virtual host and port.

- **Include Resources:** When this checkbox is selected (default), the resources that belong to an imported system are also imported. Otherwise no resources are imported, that is, imported system mappings do not expose any resources.

Related Information

- [Configure Access Control \(HTTP\) \[page 380\]](#)
- [Configure Access Control \(RFC\) \[page 387\]](#)
- [Configure Access Control \(LDAP\) \[page 393\]](#)
- [Configure Access Control \(TCP\) \[page 396\]](#)
- [Configure Accessible Resources \[page 399\]](#)
- [Configure Domain Mappings for Cookies \[page 503\]](#)

1.2.2.4.1 Configure Access Control (HTTP)

Specify the backend systems that can be accessed by your cloud applications using HTTP.

To allow your cloud applications to access a certain backend system on the intranet via HTTP, you must specify this system in the Cloud Connector.

i Note

Make sure that also redirect locations are configured as internal hosts.

If the target server responds with a redirect HTTP status code (30x), the cloud-side HTTP client usually sends the redirect over the Cloud Connector as well. The Cloud Connector runtime then performs a reverse lookup to rewrite the location header that indicates where to route the redirected request.

If the redirect location is ambiguous (that is, several mappings point to the same internal host and port), the first one found is used. If none is found, the location header stays untouched.

Tasks

[Expose Intranet Systems \[page 381\]](#)

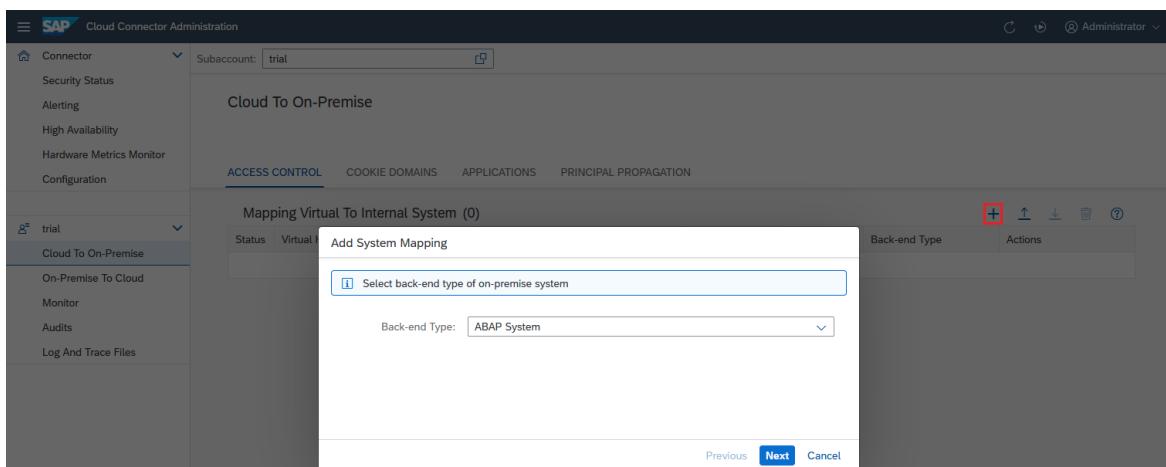
[Limit the Accessible Services for HTTP\(S\) \[page 385\]](#)

[Activate or Suspend Resources \[page 386\]](#)

Expose Intranet Systems

Insert a new entry in the Cloud Connector access control management:

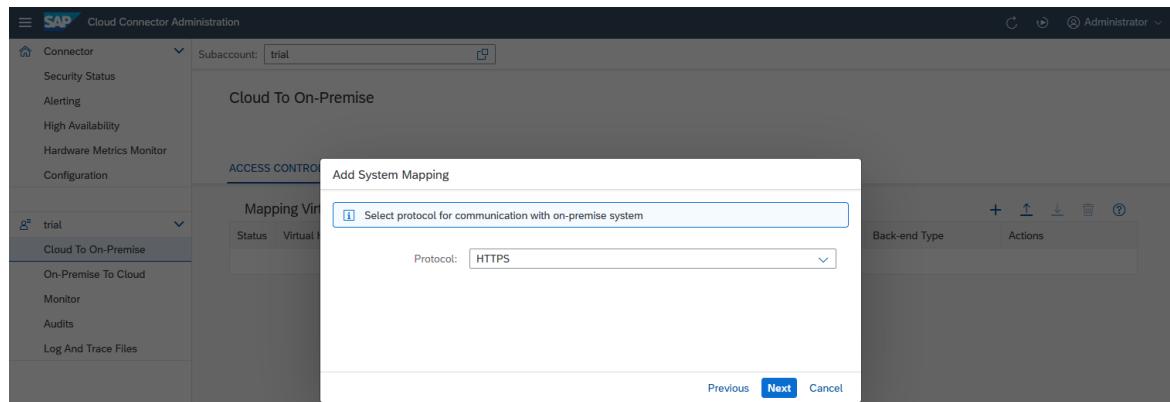
1. Choose *Cloud To On Premise* from your *Subaccount* menu.
2. Choose *Add*. A wizard will open and ask for the required values.
3. *Backend Type*: Select the description that matches best the addressed backend system.
When you are done, choose *Next*.



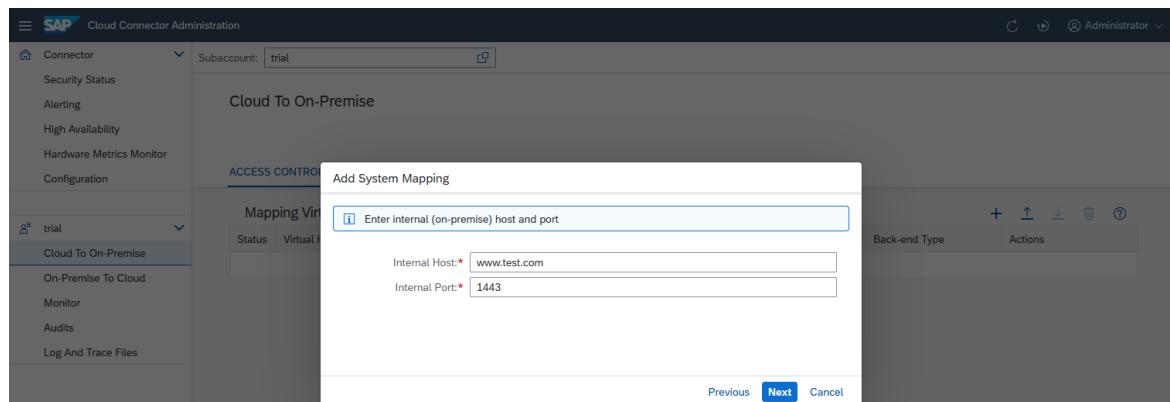
4. *Protocol*: This field allows you to decide whether the Cloud Connector should use **HTTP** or **HTTPS** for the connection to the backend system. Note that this is completely independent from the setting on cloud side. Thus, even if the HTTP destination on cloud side specifies "http://" in its URL, you can select **HTTPS**. This way, you are ensured that the entire connection from the cloud application to the actual backend system (provided through the SSL tunnel) is SSL-encrypted. The only prerequisite is that the

backend system supports HTTPS on that port. For more information, see [Initial Configuration \(HTTP\) \[page 329\]](#).

- If you specify HTTPS and there is a "system certificate" imported in the Cloud Connector, the latter attempts to use that certificate for performing a client-certificate-based logon to the backend system.
- If there is no system certificate imported, the Cloud Connector opens an HTTPS connection without client certificate.



5. *Internal Host* and *Internal Port* specify the actual host and port under which the target system can be reached within the intranet. It needs to be an existing network address that can be resolved on the intranet and has network visibility for the Cloud Connector without any proxy. Cloud Connector will try to forward the request to the network address specified by the internal host and port, so this address needs to be real.

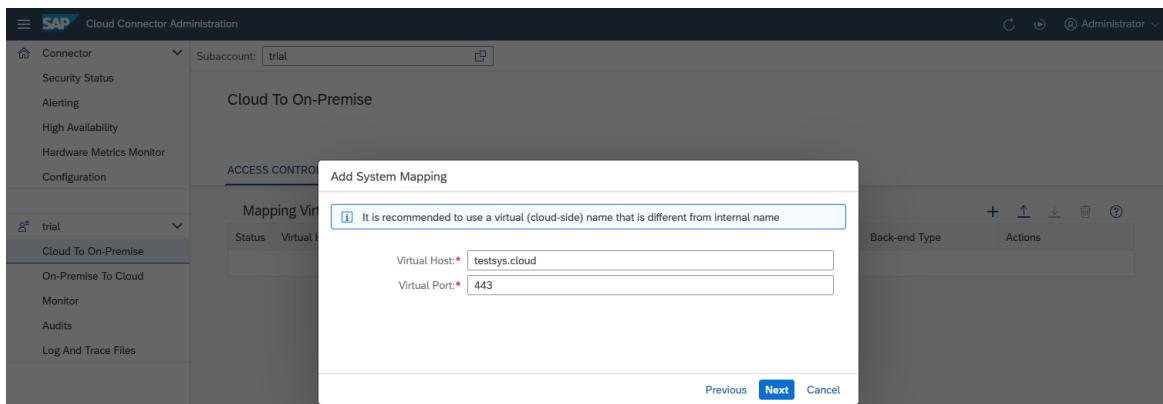


6. *Virtual Host* specifies the host name exactly as it is specified as the URL property in the HTTP destination configuration in SAP BTP

See:

[Create HTTP Destinations \[page 44\]](#) (Cloud Foundry environment)

The virtual host can be a fake name and does not need to exist. The *Virtual Port* allows you to distinguish between different entry points of your backend system, for example, **HTTP/80** and **HTTPS/443**, and have different sets of access control settings for them. For example, some noncritical resources may be accessed by HTTP, while some other critical resources are to be called using HTTPS only. The fields will be prepopulated with the values of the *Internal Host* and *Internal Port*. In case you don't modify them, you must provide your internal host and port also in the cloud-side destination configuration or in the URL used for your favorite HTTP client.



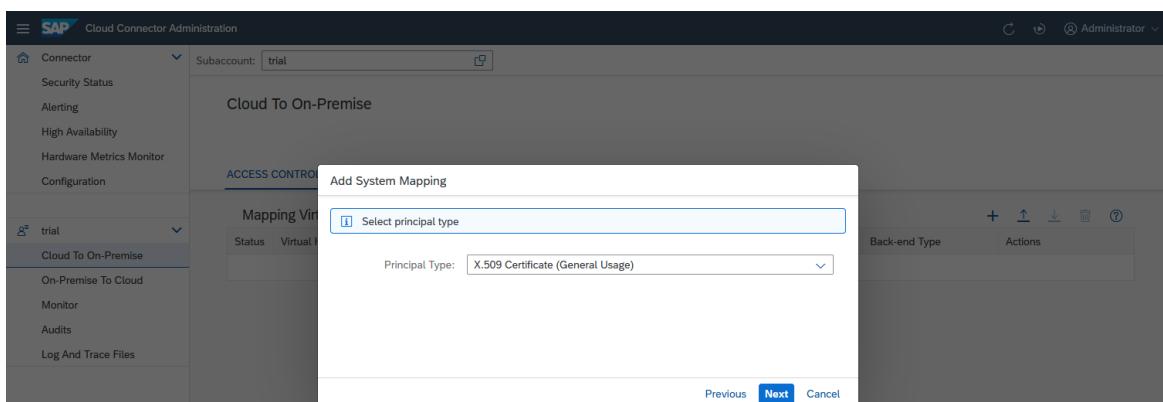
7. **Principal Type:** You can use two different variants of an X.509 certificate to define the principal type that is sent to the backend within the HTTP request: [X.509 Certificate \(General Usage\)](#) or [X.509 Certificate \(Strict Usage\)](#). The latter was introduced with Cloud Connector 2.11. If a principal is sent from the cloud side, it is injected in both cases as an HTTP header (`SSL_CLIENT_CERT`) and forwarded to the backend. If the backend is configured correctly for principal propagation, this certificate can be used for authentication. However, if the cloud side does not send a principal, the variants behave differently:
- *General Usage* (as well as `<Principal Type> = None`) allows to alternatively use the system certificate for the *TLS handshake* (actually used for trust) also for authentication.
 - *Strict Usage* does not allow this. In this case, another authentication type (specified in an additional header) is used instead, for example, basic authentication.
- This setting also applies to HTTP authentication types other than principal propagation.

i Note

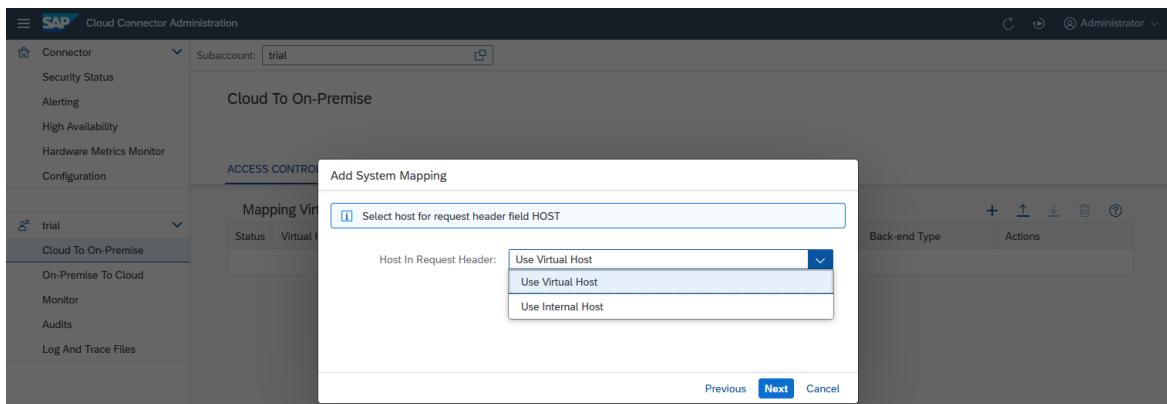
The recommended variant is [X.509 Certificate \(Strict Usage\)](#) as this lets you use principal propagation and, for example, basic authentication over the same access control entry, regardless of the logon order settings in the target system.

To prevent the use of principal propagation to the target system, choose `None` as `<Principal Type>`. In this case, no principal is injected.

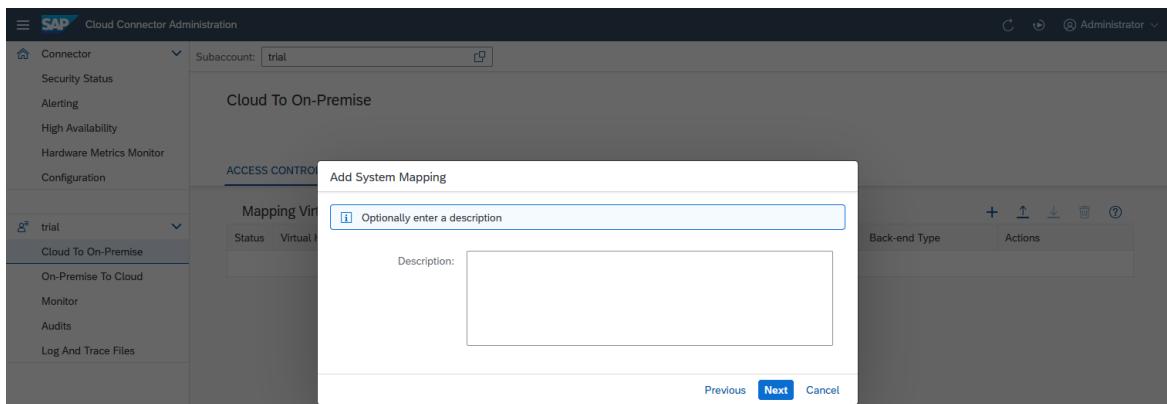
For more information on principal propagation, see [Configuring Principal Propagation \[page 350\]](#).



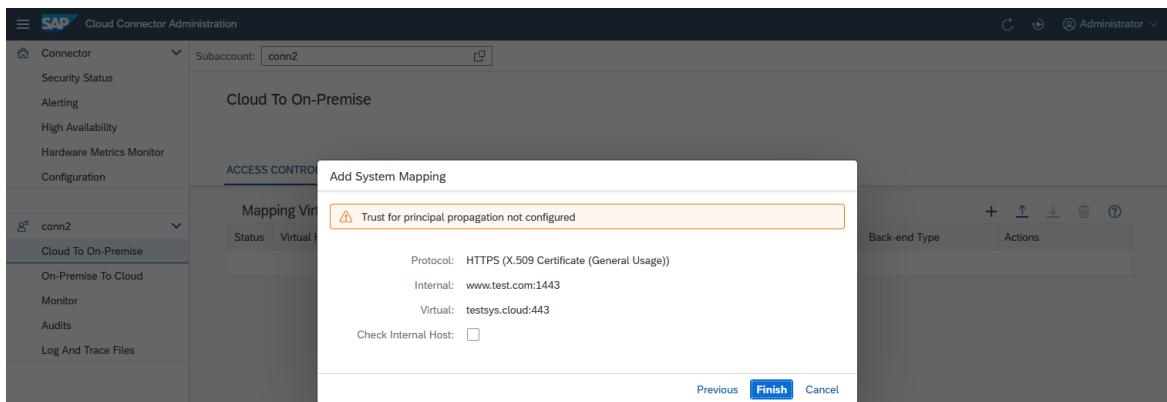
8. **Host In Request Header** lets you define, which host is used in the host header that is sent to the target server. By choosing [Use Internal Host](#), the actual host name is used. When choosing [Use Virtual Host](#), the virtual host is used. In the first case, the virtual host is still sent via the `X-Forwarded-Host` header.



9. You can enter an optional description at this stage. The respective description will be shown when pressing the [Info](#) button of the access control entry (table [Mapping Virtual to Internal System](#)).

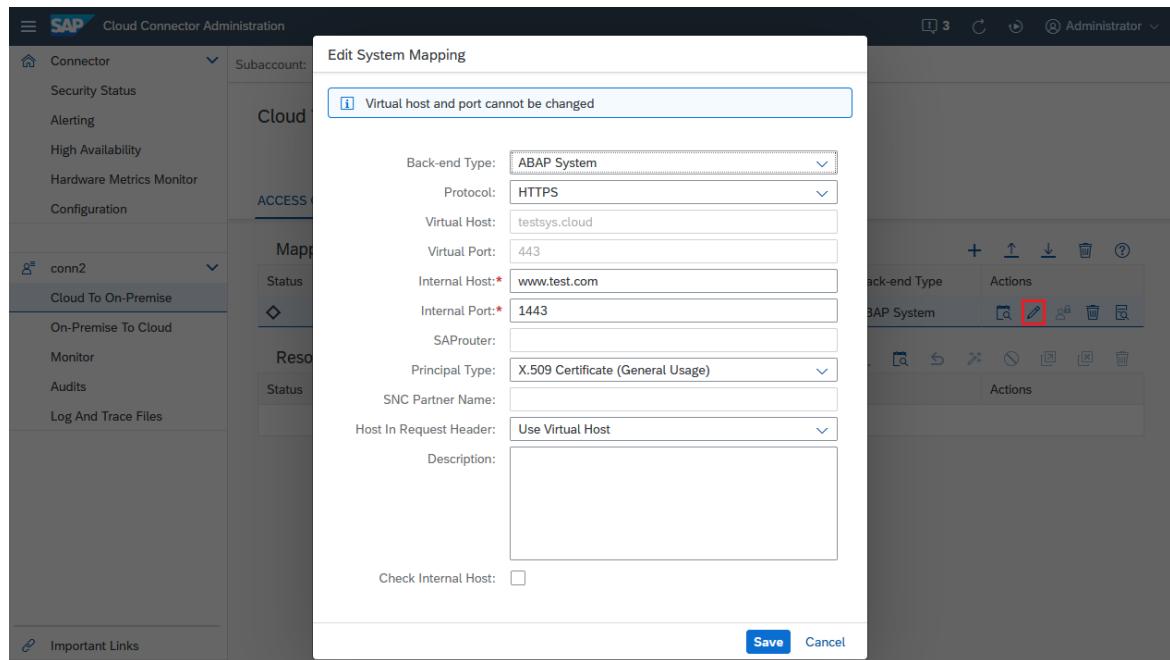


10. The summary shows information about the system to be stored and when saving the host mapping, you can trigger a ping from the Cloud Connector to the internal host, using the [Check availability of internal host](#) checkbox. This allows you to make sure the Cloud Connector can indeed access the internal system, and allows you to catch basic things, such as spelling mistakes or firewall problems between the Cloud Connector and the internal host. If the ping to the internal host is successful, the Cloud Connector saves the mapping without any remark. If it fails, a warning will pop up, that the host is not reachable. Details for the reason are available in the log files. You can execute such a check for all selected systems in the [Access Control](#) overview by pressing the button ([Check availability...](#)) in column [Actions](#).



11. Optional: You can later edit such a system mapping (via [Edit](#)) to make the Cloud Connector route the requests for **sales-system.cloud:443** to a different backend system. This can be useful if the system is currently down and there is a back-up system that can serve these requests in the meantime. However, you

cannot edit the virtual name of this system mapping. If you want to use a different fictional host name in your cloud application, you must delete the mapping and create a new one.

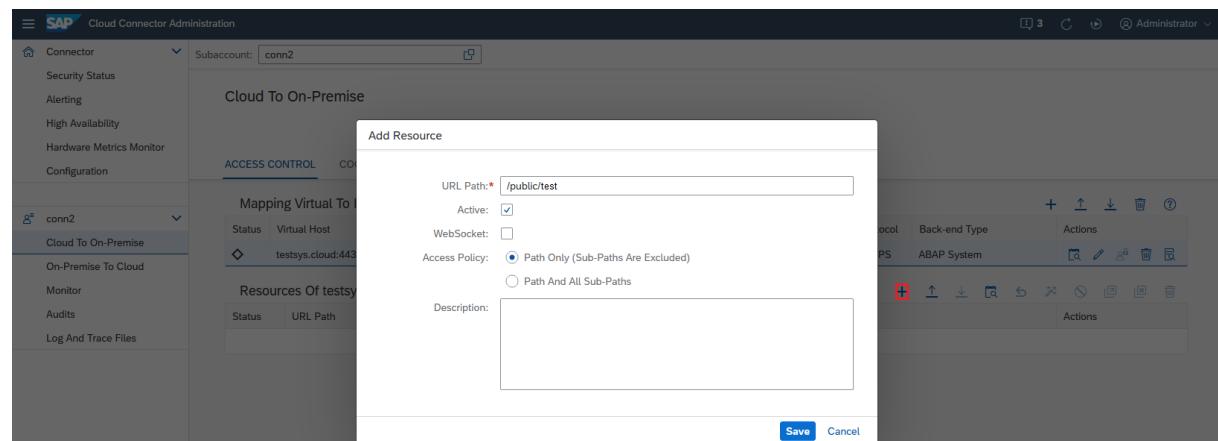


[Back to Tasks \[page 381\]](#)

Limit the Accessible Services for HTTP(S)

In addition to allowing access to a particular host and port, you also must specify which URL paths ([Resources](#)) are allowed to be invoked on that host. The Cloud Connector uses very strict allowlists for its access control. Only those URLs for which you explicitly granted access are allowed. All other HTTP(S) requests are denied by the Cloud Connector.

To define the permitted URLs for a particular backend system, choose the line corresponding to that backend system and choose [Add](#) in section [Resources Accessible On...](#) below. A dialog appears prompting you to enter the specific URL path that you want to allow to be invoked.



The Cloud Connector checks that the path part of the URL (up to but not including a possible question mark (?) that may denote the start of optional CGI-style query parameters) is exactly as specified in the configuration. If it is not, the request is denied. If you select option *Path and all sub-paths*, the Cloud Connector allows all requests for which the URL path (not considering any query parameters) starts with the specified string.

The *Active* checkbox lets you specify, if that resource is initially enabled or disabled. See the section below for more information on enabled and disabled resources.

The *WebSocket Upgrade* checkbox lets you specify, whether that resource allows a protocol upgrade.

Back to [Tasks \[page 381\]](#)

Activate or Suspend Resources

In some cases, it is useful for testing purposes to temporarily disable certain resources without having to delete them from the configuration. This allows you to easily reprovide access to these resources at a later point of time without having to type in everything once again.

- To suspend a resource, select it and choose the *Suspend* button:
The status icon turns red, and from now on, the Cloud Connector will deny all requests coming in for this resource.

Resources Of testsys.cloud:443 (1)			
Status	URL Path	Access Policy	Actions
	/public/test	Path Only (Sub-Paths Are Excluded)	

- To activate the resource again, select it and choose the *Activate* button.
- By choosing *Allow WebSocket upgrade/Disallow WebSocket upgrade* this is possible for the protocol upgrade setting as well.
- It is also possible to mark multiple lines and then suspend or activate all of them in one go by clicking the *Activate/Suspend* icons in the top row. The same is true for the corresponding *Allow WebSocket upgrade/Disallow WebSocket* icons.

Examples:

- /production/accounting* and *Path only (sub-paths are excluded)* are selected. Only requests of the form GET /production/accounting or GET /production/accounting?name1=value1&name2=value2... are allowed. (GET can also be replaced by POST, PUT, DELETE, and so on.)
- /production/accounting* and *Path and all sub-paths* are selected. All requests of the form GET /production/accounting-plus-some-more-stuff-here?name1=value1... are allowed.
- /* and *Path and all sub-paths* are selected. All requests to this server are allowed.

Back to [Tasks \[page 381\]](#)

Related Information

[Configure Domain Mappings for Cookies \[page 503\]](#)

[Consume Backend Systems \(Java Web or Java EE 6 Web Profile\)](#)

1.2.2.4.2 Configure Access Control (RFC)

Specify the backend systems that can be accessed by your cloud applications using RFC.

Tasks

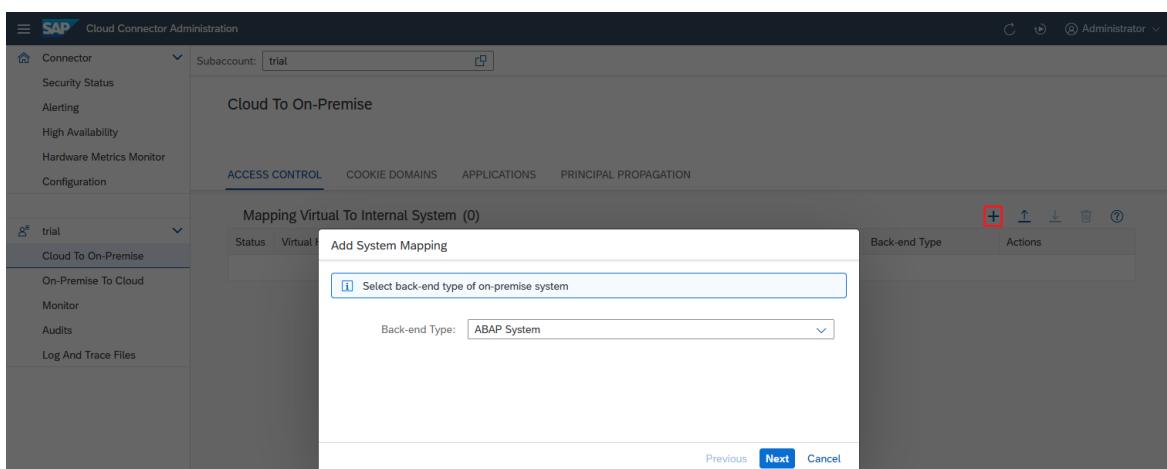
[Expose Intranet Systems \[page 387\]](#)

[Limit the Accessible Resources for RFC \[page 392\]](#)

Expose Intranet Systems

To allow your cloud applications to access a certain backend system on the intranet, insert a new entry in the Cloud Connector *Access Control* management.

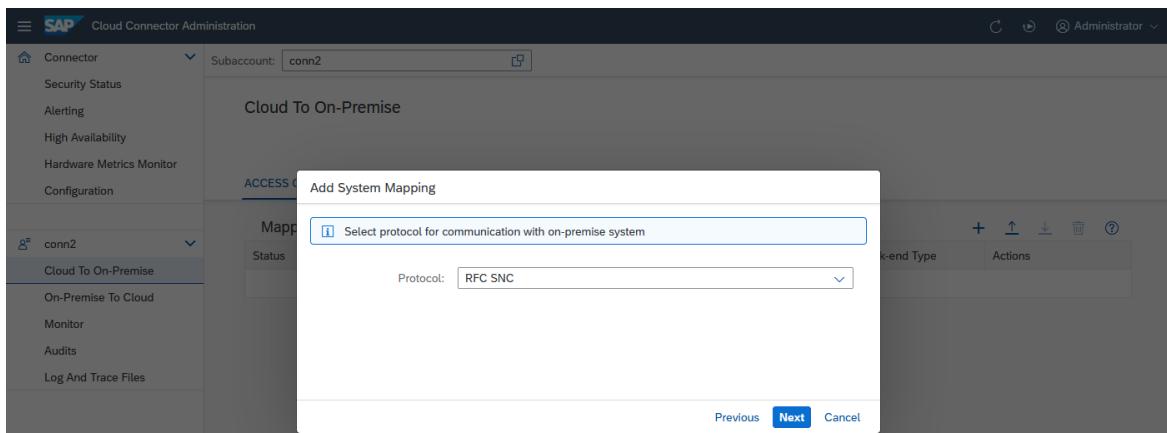
1. Choose [Cloud To On-Premise](#) from your *Subaccount* menu and go to tab *Access Control*.
2. Choose *Add*.
3. *Backend Type*: Select the backend system type ([ABAP System](#) or [SAP Gateway](#) for RFC).



4. Choose [Next](#).
5. *Protocol*: Choose [RFC](#) or [RFC SNC](#) for connecting to the backend system.

i Note

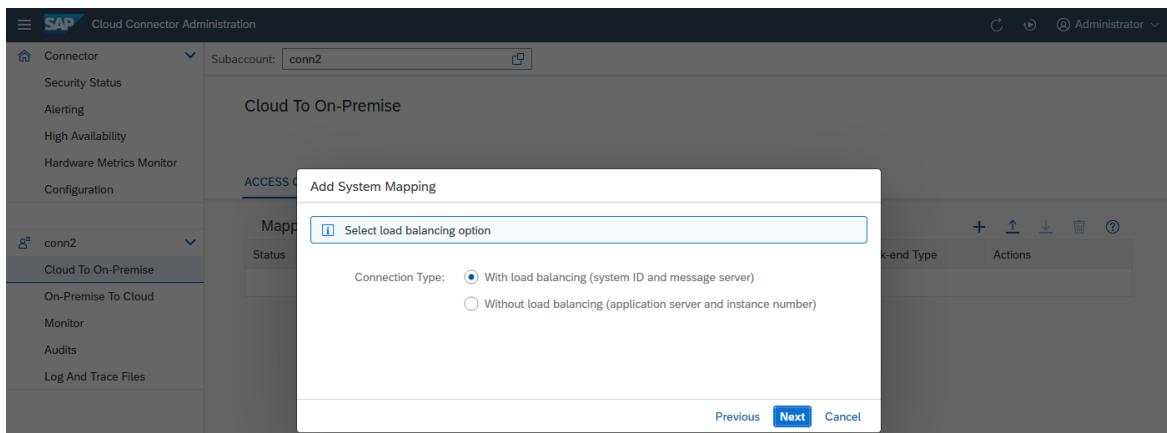
The value **RFC SNC** is independent from your settings on the cloud side, since it only specifies the communication between Cloud Connector and backend system. Using **RFC SNC**, you can ensure that the entire connection from the cloud application to the actual backend system (provided by the SSL tunnel) is secured, partly with SSL and partly with SNC. For more information, see [Initial Configuration \(RFC\) \[page 331\]](#).



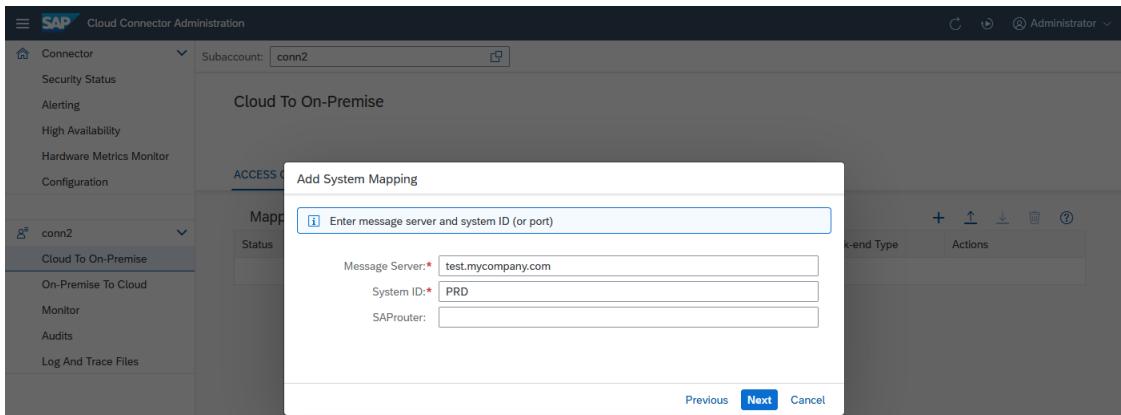
i Note

- The back end must be properly configured to support SNC connections.
- SNC configuration must be provided in the Cloud Connector.

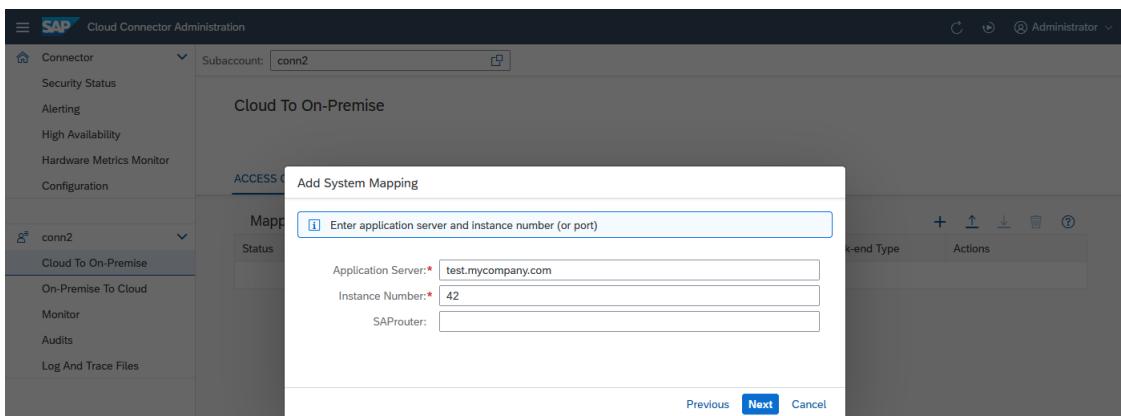
6. Choose [Next](#).
7. Choose whether you want to configure a load balancing logon or connect to a specific application server.



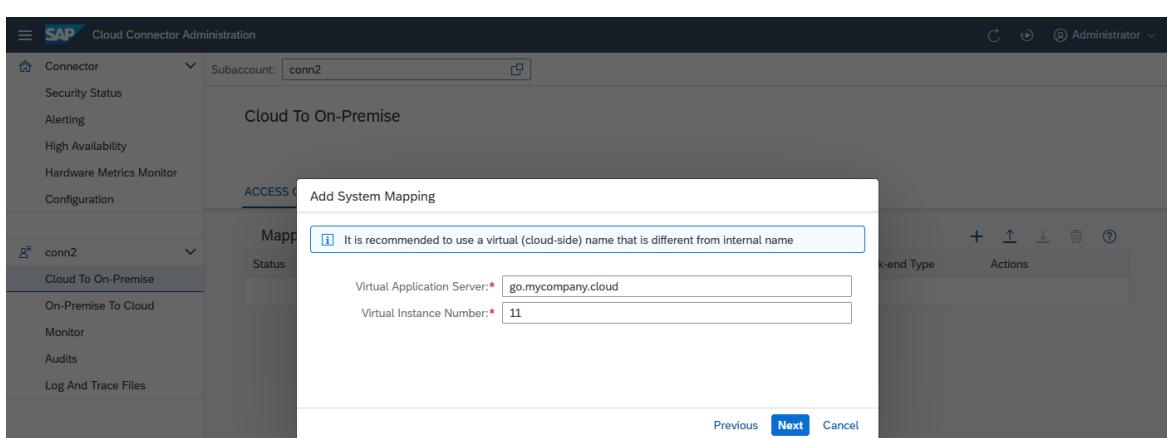
8. Specify the parameters of the backend system. It needs to be an existing network address that can be resolved on the intranet and has network visibility for the Cloud Connector. If this is only possible using a valid **SAProuter**, specify the router in the respective field. The Cloud Connector will try to establish a connection to this system, so the address has to be real.
 - When using a load-balancing configuration, the **Message Server** specifies the message server of the ABAP system. The system ID is a three-char identifier that is also found in the SAP Logon configuration. Alternatively, it's possible to directly specify the message server port in the **System ID** field.



- When using direct logon, the *Application Server* specifies one application server of the ABAP system. The instance number is a two-digit number that is also found in the SAP Logon configuration. Alternatively, it's possible to directly specify the gateway port in the *Instance Number* field.



- Optional: You can virtualize the system information in case you like to hide your internal host names from the cloud. The virtual information can be a fake name which does not need to exist. The fields will be pre-populated with the values of the configuration provided in *Message Server* and *System ID*, or *Application Server* and *Instance Number*.



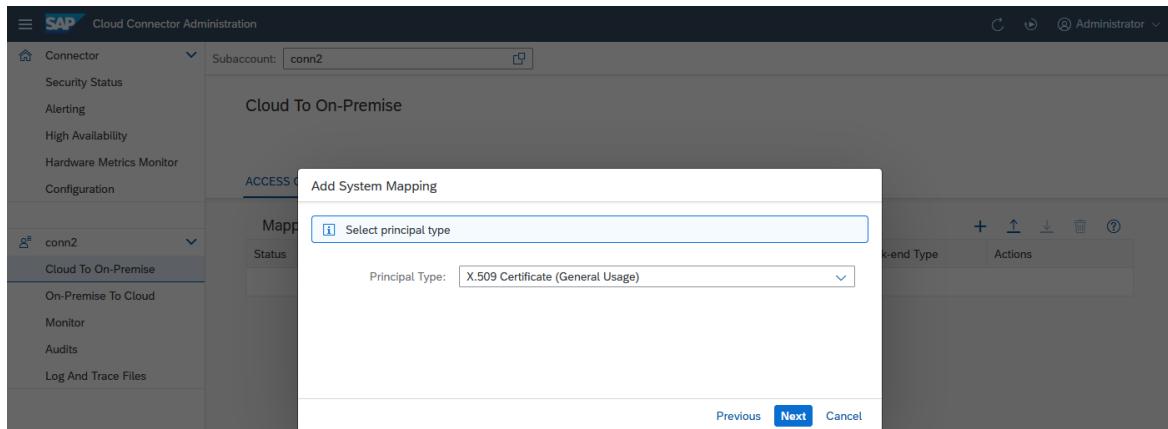
- Virtual Message Server* - specifies the host name exactly as specified as the `jco.client.mshost` property in the RFC destination configuration in the cloud. The *Virtual System ID* allows you to distinguish between different entry points of your backend system that have different sets of access control settings. The value needs to be the same like for the `jco.client.r3name` property in the RFC destination configuration in the cloud.

- **Virtual Application Server** - it specifies the host name exactly as specified as the `jco.client.ashost` property in the RFC destination configuration in the cloud. The **Virtual Instance Number** allows you to distinguish between different entry points of your backend system that have different sets of access control settings. The value needs to be the same like for the `jco.client.sysnr` property in the RFC destination configuration in the cloud.
10. This step is only relevant if you have chosen **RFC SNC**. The `<Principal Type>` field defines what kind of principal is used when configuring a destination on the cloud side using this system mapping with authentication type **Principal Propagation**. No matter what you choose, make sure that the general configuration for the `<Principal Type>` has been done to make it work correctly. For destinations using different authentication types, this setting is ignored. In case you choose **None** as `<Principal Type>`, it is not possible to apply principal propagation to this system.

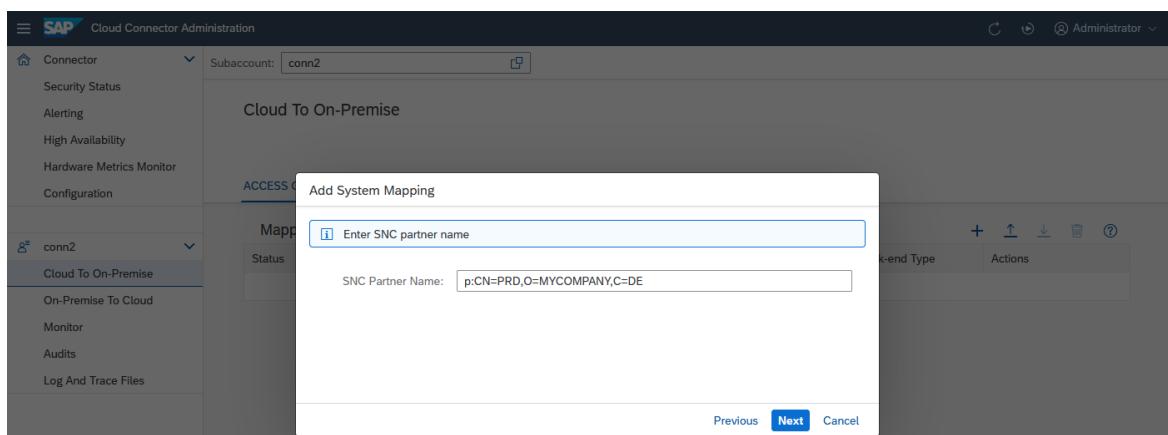
i Note

If you use an RFC connection, you cannot choose between different principal types. Only the **X.509** certificate is supported. You need an SNC-enabled backend connection to use it. For RFC, the two X.509 certificate variants **X.509 certificate (general usage)** and **X.509 certificate (strict usage)** do not differ in behavior.

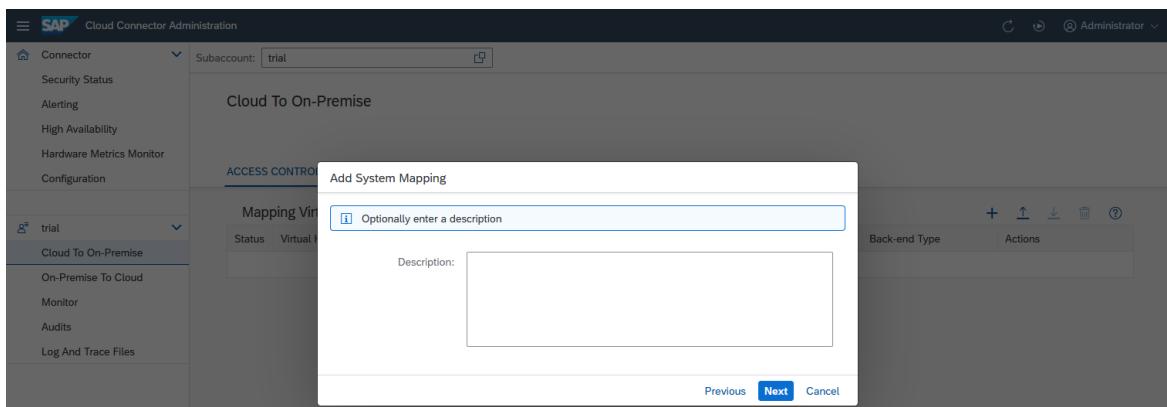
For more information on principal propagation, see [Configuring Principal Propagation \[page 350\]](#).



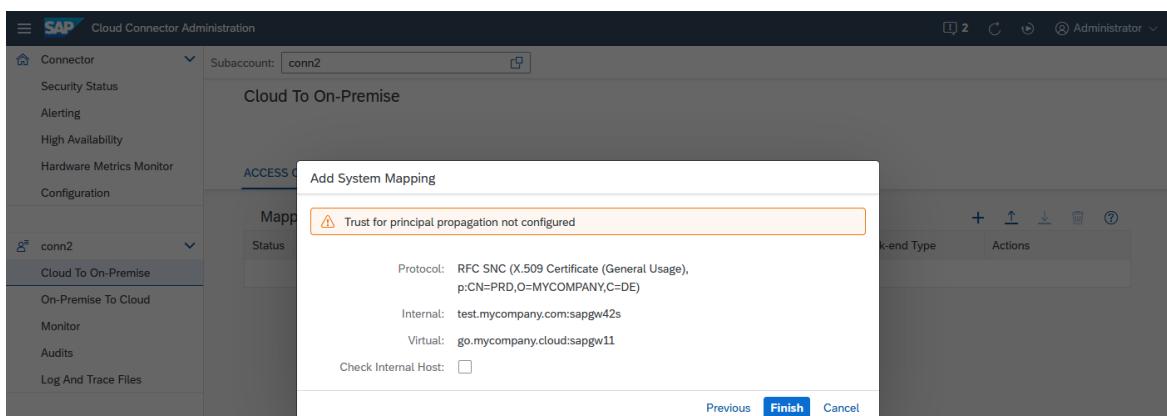
11. **SNC Partner Name:** This step will only come up if you have chosen **RFC SNC**. The SNC partner name needs to contain the correct SNC identification of the target system.



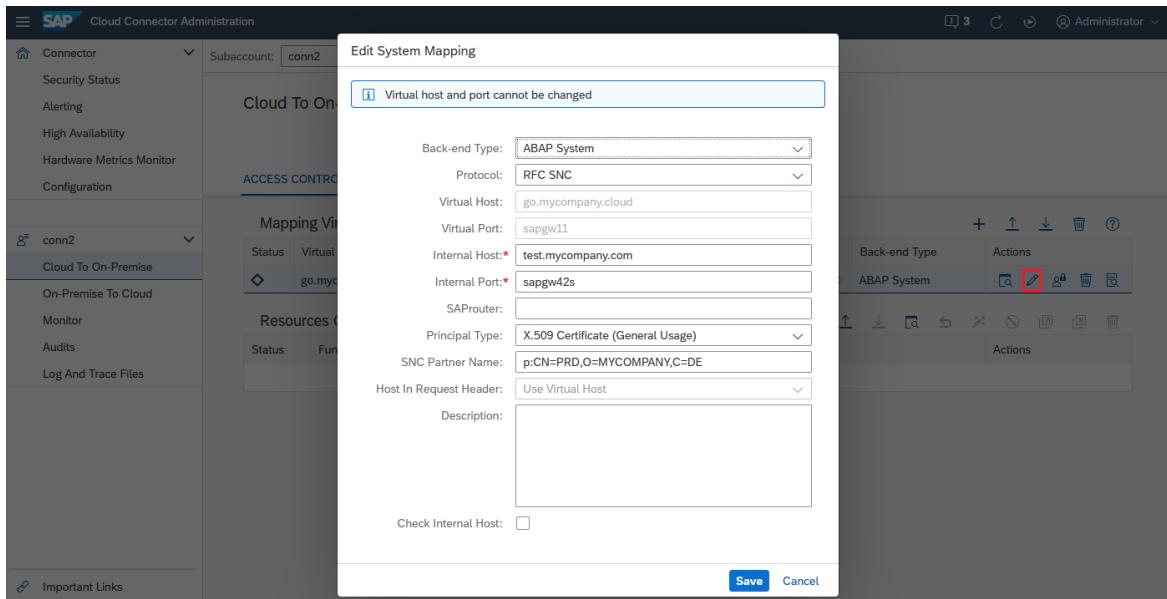
12. **Mapping Virtual to Internal System:** You can enter an optional description at this stage. The respective description will be shown as a rich tooltip when the mouse hovers over the entries of the virtual host column (table).



13. The summary shows information about the system to be stored. When saving the system mapping, you can trigger a ping from the Cloud Connector to the internal host, using the *Check availability of internal host* checkbox. This allows you to make sure the Cloud Connector can indeed access the internal system, and allows you to catch basic things, such as spelling mistakes or firewall problems between the Cloud Connector and the internal host. If the ping to the internal host is successful, the Cloud Connector saves the mapping without any remark. If it fails, a warning will pop up, that the host is not reachable. Details for the reason are available in the log files. You can execute such a check at any time later for all selected systems in the *Access Control* overview.



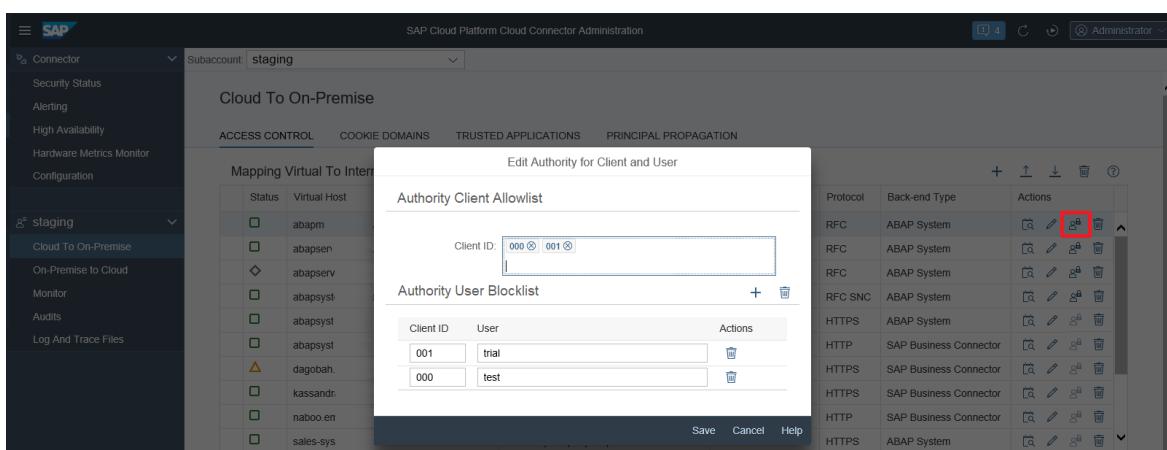
14. Optional: You can later edit a system mapping (choose *Edit*) to make the Cloud Connector route the requests for ***sales-system.cloud:sapgw42*** to a different backend system. This can be useful if the system is currently down and there is a back-up system that can serve these requests in the meantime. However, you cannot edit the virtual name of this system mapping. If you want to use a different fictional host name in your cloud application, you must delete the mapping and create a new one. Here, you can also change the *Principal Type* to **None** in case you don't want to allow principal propagation to a certain system.



15. *Optional.* You can later edit a system mapping to add more protection to your system when using RFC via the Cloud Connector, by restricting the mapping to specified clients and users: in column *Actions*, choose the button *Maintain Authority Lists (only RFC)* to open an allowlist/blocklist dialog. In section *Authority Client Allowlist*, enter all clients of the corresponding system in the field <Client ID> that you want to allow to use the Cloud Connector connection. In section *Authority User Blocklist*, press the button *Add a user authority* (+) to enter all users you want to exclude from this connection. Each user must be assigned to a specified client. When you are done, press *Save*.

i Note

This function applies for RFC/RFC SNC only.



[Back to Tasks \[page 387\]](#)

Limit the Accessible Resources for RFC

In addition to allowing access to a particular host and port, you also must specify which function modules (*Resources*) are allowed to be invoked on that host. You can enter an optional description at this stage. The

Cloud Connector uses very strict allowlists for its access control. Besides internally used infrastructure function modules, only function modules for which you explicitly granted access are allowed.

- To define the permitted function modules for a particular backend system, choose the row corresponding to that backend system and press **Add** in section *Resources Accessible On...* below. A dialog appears, prompting you to enter the specific function module name whose invoking you want to allow.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a tree view with 'conn2' selected, under which 'Cloud To On-Premise' is expanded. The main area shows a table of resources, with one row for 'abapmessageserver' highlighted. A modal dialog titled 'Add Resource' is open over this table. The dialog has fields for 'Function Name:' (empty), 'Active:' (checked), 'Naming Policy:' (radio buttons for 'Exact Name' (selected) and 'Prefix'), and a 'Description:' text area (empty). At the bottom of the dialog are 'Save' and 'Cancel' buttons. The background table lists various ABAP systems and a Business Connector, with the 'abapmessageserver' row being the current focus.

- The Cloud Connector checks that the function module name of an incoming request is exactly as specified in the configuration. If it is not, the request is denied.
- If you select the *Prefix* option, the Cloud Connector allows all incoming requests, for which the function module name begins with the specified string.
- The *Active* checkbox allows you to specify whether that resource should be initially enabled or disabled.

[Back to Tasks \[page 387\]](#)

1.2.2.4.3 Configure Access Control (LDAP)

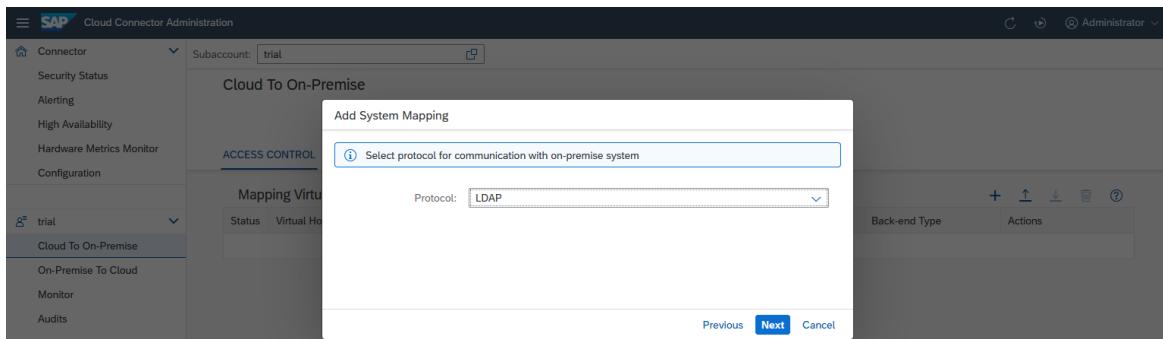
Add a specified system mapping to the Cloud Connector if you want to use an on-premise LDAP server or user authentication in your cloud application.

To allow your cloud applications to access an on-premise LDAP server, insert a new entry in the Cloud Connector access control management.

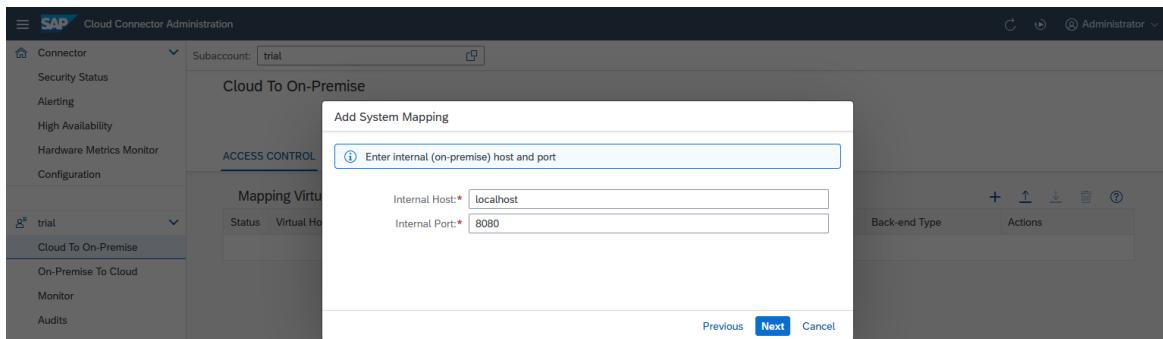
- Choose *Cloud To On-Premise* from your *Subaccount* menu.
- Choose **Add** (+). A wizard opens and asks for the required values.
- Backend Type:** Select **Non-SAP System** from the drop down list. When you are done, choose **Next**.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a tree view with 'trial' selected, under which 'Cloud To On-Premise' is expanded. The main area shows a table of system mappings, with the 'Back-end Type' column showing 'Non-SAP System'. A modal dialog titled 'Add System Mapping' is open over this table. It has a field 'Select back-end type of on-premise system' with a help icon, and a dropdown 'Back-end Type' set to 'Non-SAP System'. At the bottom of the dialog are 'Previous', 'Next', and 'Cancel' buttons. The background table lists various back-end types, with the 'Non-SAP System' row being the current focus.

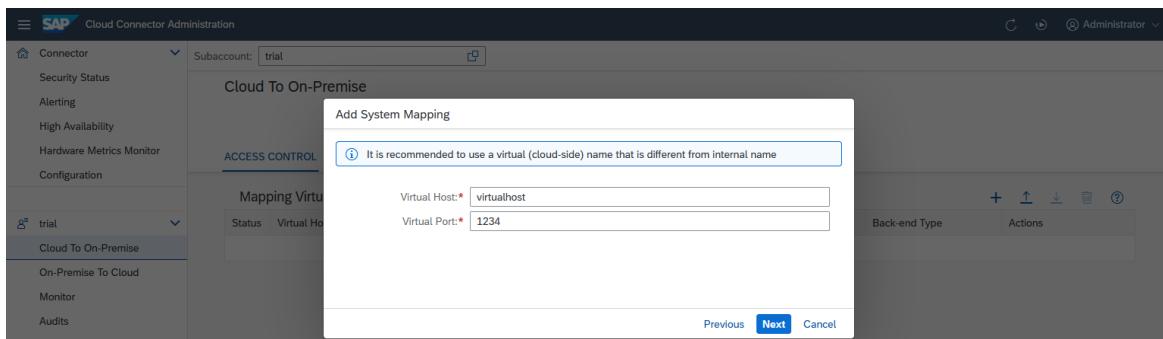
4. **Protocol:** Select **LDAP** or **LDAPS** for the connection to the backend system. When you are done, choose **Next**.



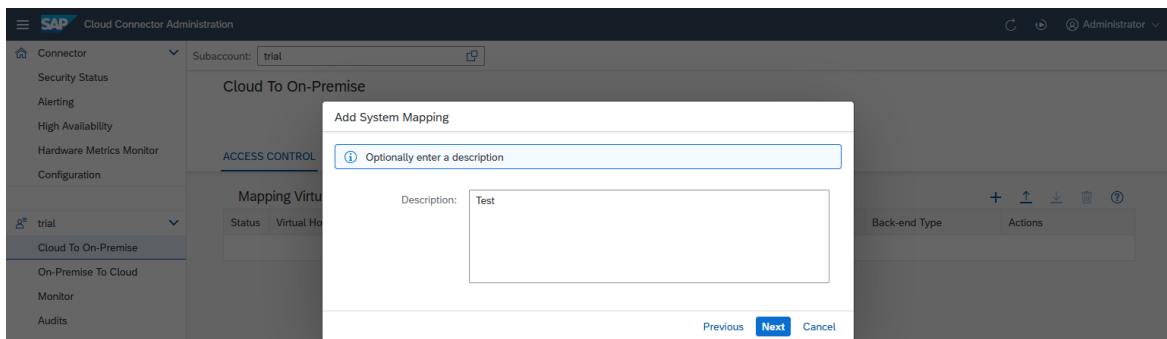
5. **Internal Host** and **Internal Port**: specify the host and port under which the target system can be reached within the intranet. It needs to be an existing network address that can be resolved on the intranet and has network visibility for the Cloud Connector. The Cloud Connector will try to forward the request to the network address specified by the internal host and port, so this address needs to be real.



6. Enter a **Virtual Host** and **Virtual Port**. The virtual host can be a fake name and does not need to exist. The fields are pre-populated with the values of the **Internal Host** and **Internal Port**.

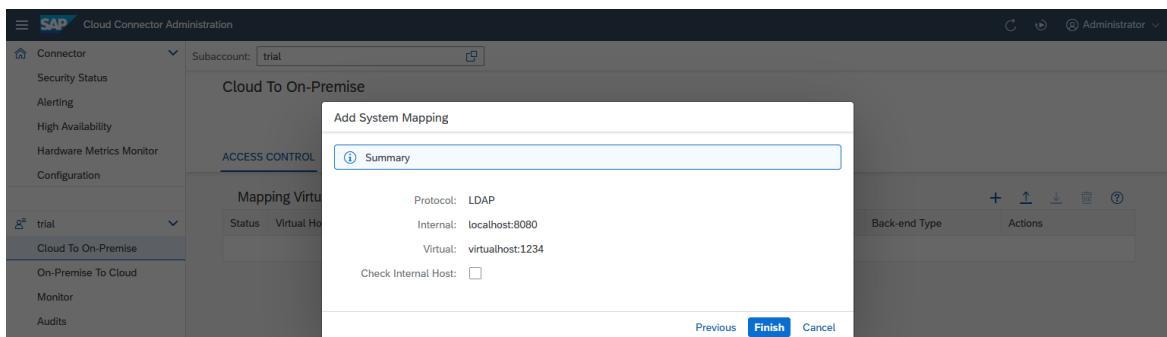


7. You can enter an optional description at this stage. The respective description will be shown as a tooltip when you press the button **Show Details** in column **Actions** of the **Mapping Virtual To Internal System** overview.

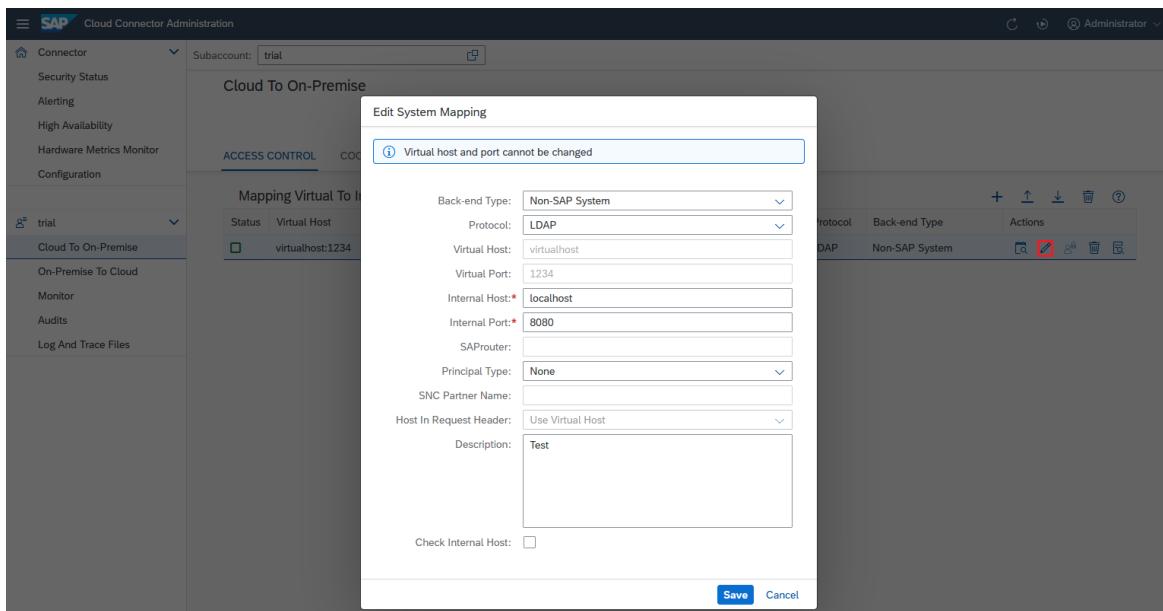


- The summary shows information about the system to be stored. When saving the host mapping, you can trigger a ping from the Cloud Connector to the internal host, using the [Check Internal Host](#) check box. This allows you to make sure the Cloud Connector can indeed access the internal system. Also, you can catch basic things, such as spelling mistakes or firewall problems between the Cloud Connector and the internal host.

If the ping to the internal host is successful, the Cloud Connector saves the mapping without any remark. If it fails, a warning is displayed in column [Check Result](#), that the host is not reachable. Details for the reason are available in the log files. You can execute such a check at any time later for all selected systems in the [Mapping Virtual To Internal System](#) overview by pressing [Check Availability of Internal Host](#) in column [Actions](#).



- Optional: You can later edit the system mapping (by choosing [Edit](#)) to make the Cloud Connector route the requests to a different LDAP server. This can be useful if the system is currently down and there is a back-up LDAP server that can serve these requests in the meantime. However, you cannot edit the virtual name of this system mapping. If you want to use a different fictional host name in your cloud application, you have to delete the mapping and create a new one.

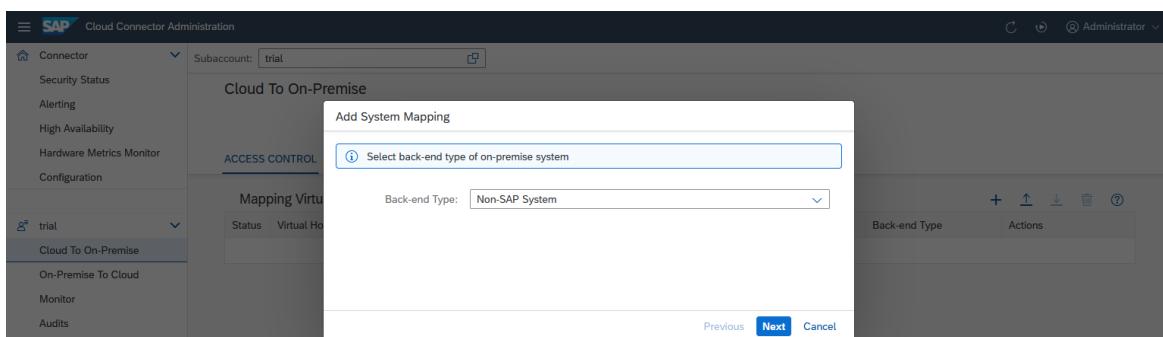


1.2.2.4.4 Configure Access Control (TCP)

Add a specified system mapping to the Cloud Connector if you want to use the TCP protocol for communication with a backend system.

To allow your cloud applications to access a certain backend system on the intranet via TCP, insert a new entry in the Cloud Connector access control management.

1. Choose *Cloud To On-Premise* from your *Subaccount* menu.
2. Choose *Add* (+). A wizard opens and asks for the required values.
3. *Backend Type*: Select **Non-SAP System** from the drop-down list. When you are done, choose *Next*.



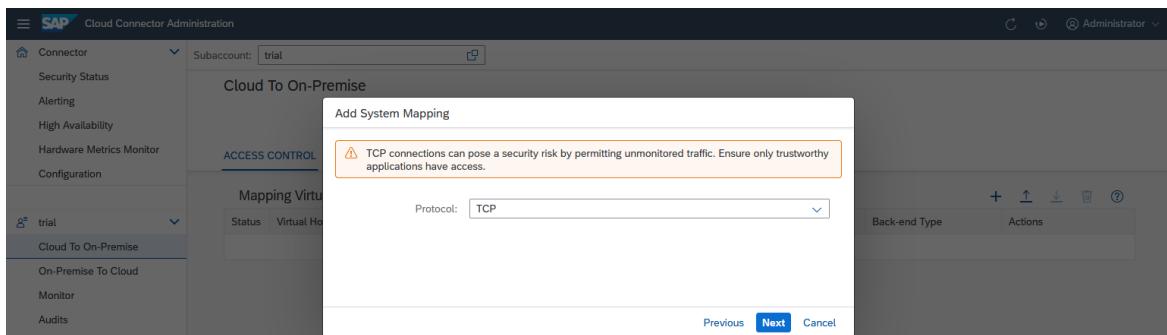
4. *Protocol*: Select **TCP** or **TCP SSL** for the connection to the backend system. When choosing TCP, you can perform an end-to-end TLS handshake from the cloud client to the backend. If the cloud-side client is using plain communication, but you still need to encrypt the hop between Cloud Connector and the backend, choose **TCP SSL**. When you are done, choose *Next*.

i Note

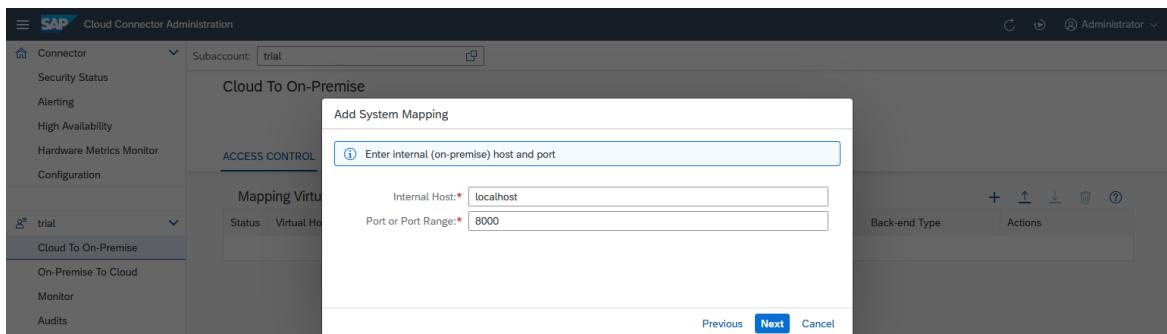
When selecting TCP as protocol, the following warning message is displayed: TCP connections can pose a security risk by permitting unmonitored traffic. Ensure only

trustworthy applications have access. The reason is that using plain TCP, the Cloud Connector cannot see or log any detail information about the calls. Therefore, in contrast to HTTP or RFC (both running on top of TCP), the Cloud Connector cannot check the validity of a request. To minimize this risk, make sure you

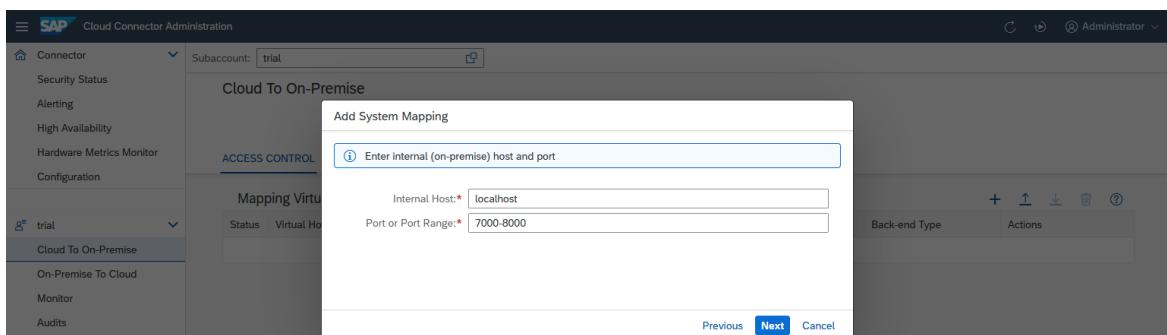
- deploy only trusted applications on SAP BTP.
- configure an application allowlist in the Cloud Connector, see [Set Up Trust for Principal Propagation \[page 351\]](#).
- take the recommended security measures for your SAP BTP (sub)account. See section [Security](#) in the SAP BTP documentation.



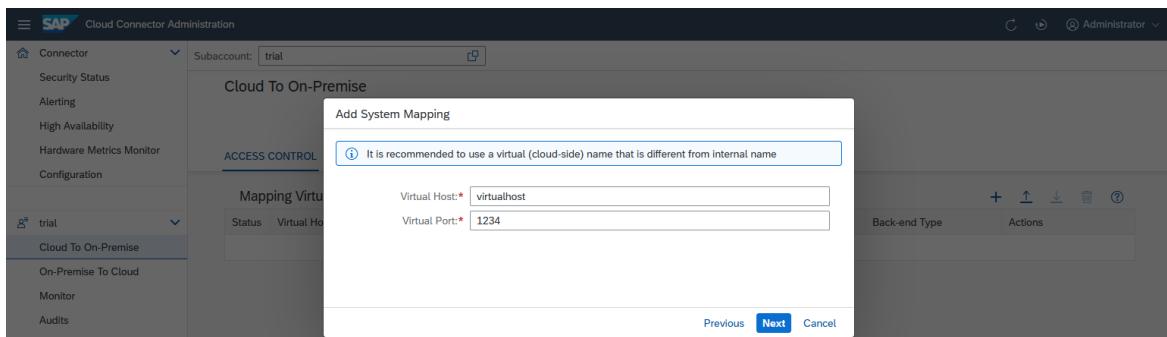
5. *Internal Host and Port or Port Range*: specify the host and port under which the target system can be reached within the intranet. It needs to be an existing network address that can be resolved on the intranet and has network visibility for the Cloud Connector. The Cloud Connector will try to forward the request to the network address specified by the internal host and port. That is why this address needs to be real.



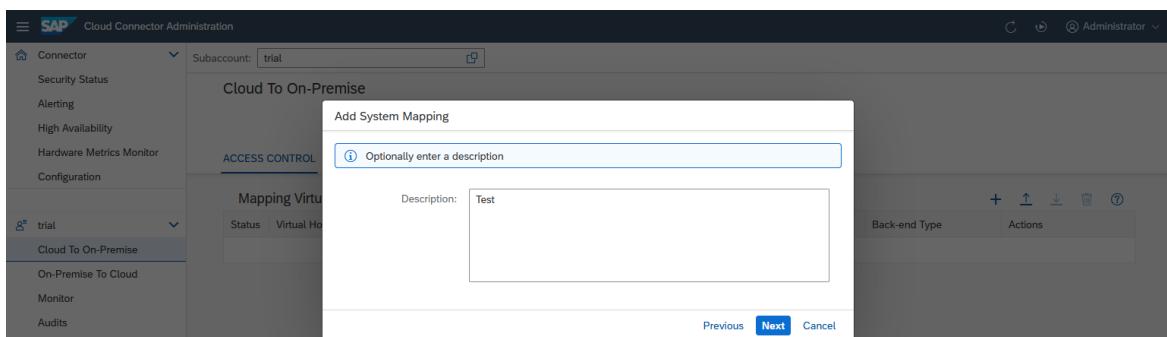
For TCP and TCP SSL, you can also specify a port range through its lower and upper limit, separated by a hyphen.



6. Enter a *Virtual Host* and *Virtual Port*. The virtual host can be a fake name and does not need to exist. The fields are prepopulated with the values of the *Internal Host* and *Port or Port Range*.

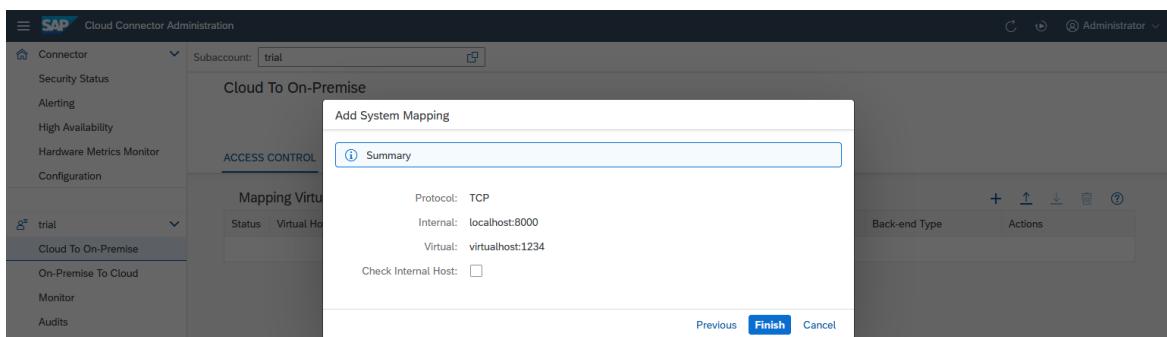


7. You can enter an optional description at this stage. The respective description will be shown as a tooltip when you press the button *Show Details* in column *Actions* of the *Mapping Virtual To Internal System* overview.

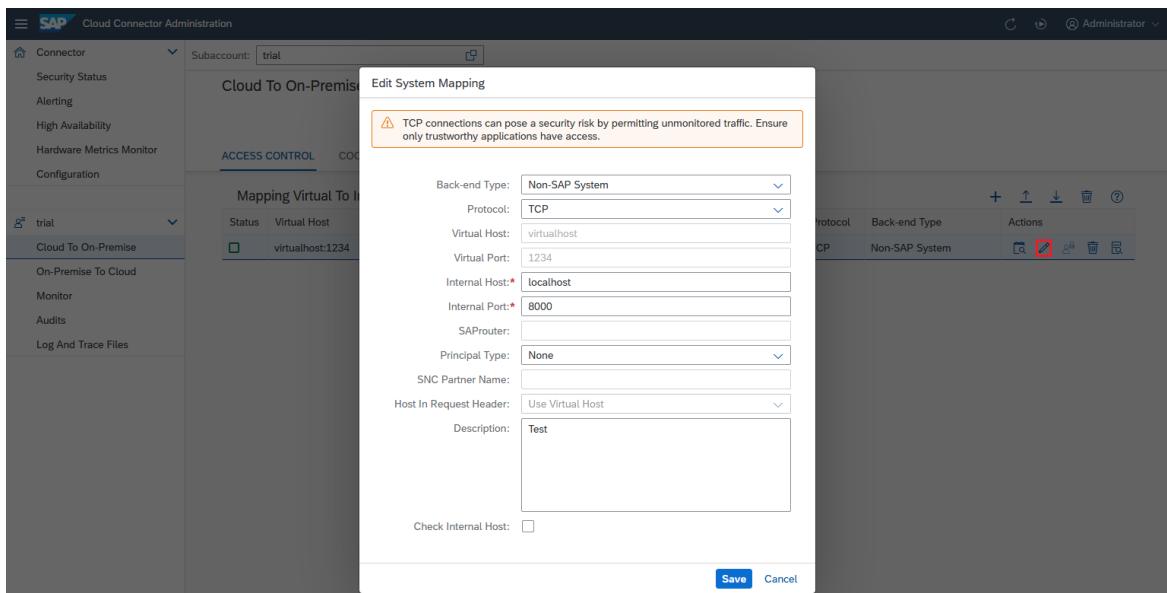


8. The summary shows information about the system to be stored. When saving the host mapping, you can trigger a ping from the Cloud Connector to the internal host, using the *Check Internal Host* checkbox. This allows you to make sure the Cloud Connector can indeed access the internal system. Also, you can catch basic things, such as spelling mistakes or firewall problems between the Cloud Connector and the internal host.

If the ping to the internal host is successful, the Cloud Connector saves the mapping without any remark. If it fails, a warning is displayed in column *Check Result*, that the host is not reachable. Details for the reason are available in the log files. You can execute such a check at any time later for all selected systems in the *Mapping Virtual To Internal System* overview by pressing *Check Availability of Internal Host* in column *Actions*.



9. Optional: You can later edit the system mapping (by choosing *Edit*) to make the Cloud Connector route the requests to a different backend system. This can be useful if the system is currently down and there is a backup system that can serve these requests in the meantime. However, you cannot edit the virtual name nor port of this system mapping. If you want to use a different fictional host name in your cloud application, you must delete the mapping and create a new one. The same goes for port ranges. If a port range needs to be changed, you must delete the mapping and create it again with the desired port range.



1.2.2.4.5 Configure Accessible Resources

Configure backend systems and resources in the Cloud Connector, to make them available for a cloud application.

Tasks

[Map Systems and Limit Access \[page 399\]](#)

[Use Scenarios for Resources \[page 401\]](#)

Map Systems and Limit Access

Initially, after installing a new Cloud Connector, no network systems or resources are exposed to the cloud. You must configure each system and resource used by applications of the connected cloud subaccount. To do this, choose [Cloud To On Premise](#) from your subaccount menu and go to tab [Access Control](#):

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a tree view with 'Connector' selected, and a subaccount dropdown set to 'trial'. The main area is titled 'Cloud To On-Premise' and contains four tabs: 'ACCESS CONTROL', 'COOKIE DOMAINS', 'APPLICATIONS', and 'PRINCIPAL PROPAGATION'. Below these tabs are two tables:

- Mapping Virtual To Internal System (7)**: A table with columns: Status, Virtual Host, Internal Host, Check Result, Protocol, Back-end Type, and Actions. It lists mappings like 'abapmessageserver.hana.cloud:saprmsFCB' to 'xxxx.wdf.sap.corp:sapmsV9U'.
- Resources Of abapmessageserver.hana.cloud:saprmsFCB (4)**: A table with columns: Status, Function Name, Naming Policy, and Actions. It lists resources like 'JTEST_ADD_INT' with 'Exact Name' policy.

The Cloud Connector supports any type of system (SAP or non-SAP system) that can be called via one of the supported protocols. For example, a convenient way to access an ABAP system from a cloud application is via [SAP NetWeaver Gateway](#), as it allows the consumption of ABAP content via HTTP and open standards.

- For systems using HTTP communication, see: [Configure Access Control \(HTTP\) \[page 380\]](#).
- For information on configuring RFC resources, see: [Configure Access Control \(RFC\) \[page 387\]](#).

We recommend that you limit the access to backend services and resources. Instead of configuring a system and granting access to all its resources, grant access only to the resources needed by the cloud application. For example, define access to an HTTP service by specifying the service URL root path and allowing access to all its subpaths.

When configuring an on-premise system, you can define a virtual host and port for the specified system. The virtual host name and port represent the fully qualified domain name of the related system in the cloud. We recommend that you use the virtual host name/port mapping to prevent leaking information about a system's physical machine name and port to the cloud.

Edit System Mapping

(i) Virtual host and port cannot be changed

Back-end Type:	ABAP System
Protocol:	RFC
Virtual Host:	abapmessageserver.hana.cloud
Virtual Port:	sapmsFCB
Internal Host: [*]	xxxxx.wdf.sap.corp
Internal Port: [*]	sapmsV9U
SAProuter:	
Principal Type:	None
SNC Partner Name:	
Host In Request Header:	Use Virtual Host
Description:	
Check Internal Host:	<input type="checkbox"/>

Save **Cancel**

Back to [Tasks \[page 399\]](#)

Use Scenarios for Resources

As of version 2.12, the Cloud Connector lets you define a set of resources as a scenario that you can export, and import into another Cloud Connector.

Scenarios are useful if you provide an application to many consumers, which invokes a large number of resources in an on-premise system. In this case, you must expose a system on your Cloud Connector that covers all required resources, which increases the risk of incorrect configuration.

[Define and Export a Scenario \[page 402\]](#)

[Import a Scenario \[page 402\]](#)

[Remove a Scenario \[page 403\]](#)

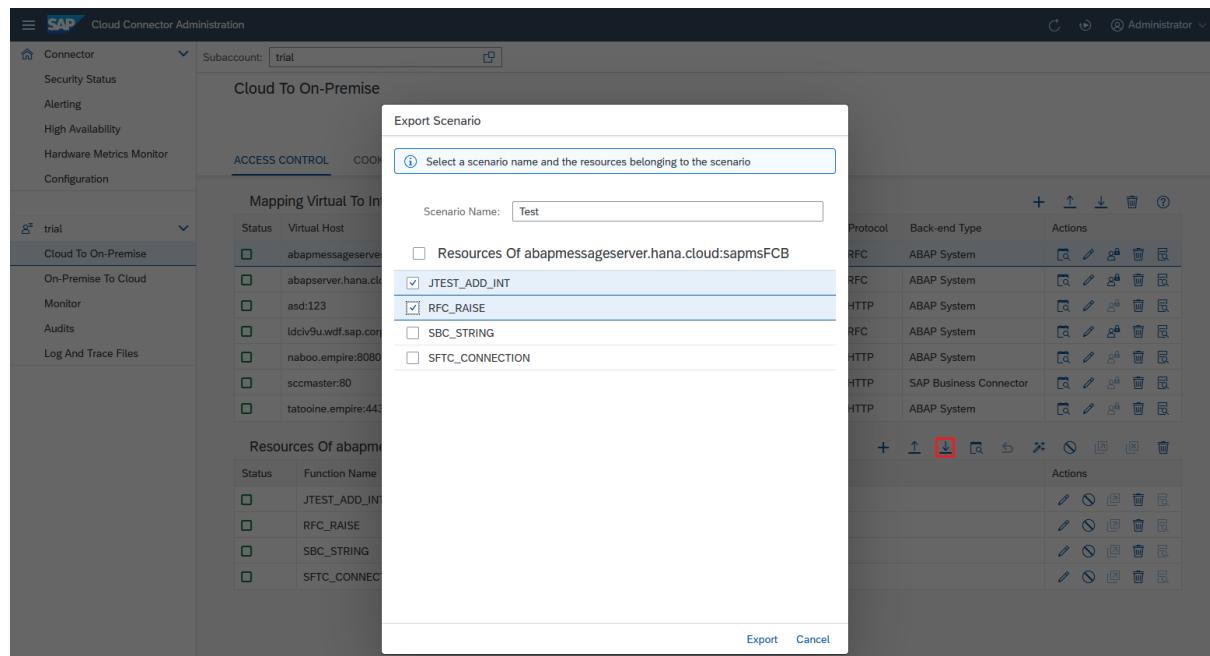
Define and Export a Scenario

If you, as application owner, have implemented and tested a scenario, and configured a Cloud Connector accordingly, you can define the scenario as follows:

1. Choose the *Export Scenario* button.
2. In the dialog, select the resources that belong to the scenario.
3. In the field <Scenario Name>, choose a descriptive name, under which the set of resources can be identified in the consuming Cloud Connector.
4. Choose *Export* to store the scenario.

i Note

For applications provided by SAP, default scenario definitions may be available. To verify this, check the corresponding application documentation.



[Back to Use Scenarios for Resources \[page 401\]](#)

[Back to Tasks \[page 399\]](#)

Import a Scenario

As an administrator taking care for a scenario configuration in some other Cloud Connector, perform the following steps:

1. Choose the *Import Scenario* button to add all required resources to the desired access control entry.
2. In the dialog, navigate to the folder of the archive that contains the scenario definition.
3. Choose *Import*. The resources of the scenario are merged with the existing set of resources which are already available in the access control entry.

The screenshot shows the SAP Cloud Connector Administration interface. On the left, there's a sidebar with options like Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, and Configuration. The main area is titled 'Cloud To On-Premise' and has tabs for ACCESS CONTROL, COOKIE DOMAINS, APPLICATIONS, and PRINCIPAL PROPAGATION. Under ACCESS CONTROL, there are two tables: 'Mapping Virtual To Internal System (7)' and 'Resources Of abapmessageserver.hana.cloud:saprmsFCB (4)'. The 'Import Scenario' dialog is open over the first table. It has fields for 'Scenario File:' with a 'Browse' button, an 'Import' button, and a 'Cancel' button. The 'Mapping Virtual To Internal System' table lists various resources with columns for Status, Virtual Host, Internal Host, Check Result, Protocol, Back-end Type, and Actions. The 'Resources Of abapmessageserver.hana.cloud:saprmsFCB' table lists function names with columns for Status, Function Name, Naming Policy, and Actions.

All resources belonging to a scenario get an additional scenario icon in their status. When hovering over it, the assigned scenario(s) of this resource are listed.

[Back to Use Scenarios for Resources \[page 401\]](#)

[Back to Tasks \[page 399\]](#)

Remove a Scenario

To remove a scenario:

1. Choose the *Remove Scenario* button.
2. In the dialog, choose the scenario(s) you want to remove.
3. Choose *Remove* to remove all resources associated with the chosen entry from the access control entry. Resources that existed before, or are assigned to another scenario as well, are not removed.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar is titled 'Connector' and lists subaccounts: trial, Cloud To On-Premise, On-Premise To Cloud, Monitor, Audits, and Log And Trace Files. The main area is titled 'Cloud To On-Premise' and has tabs for ACCESS CONTROL, COOKIE DOMAINS, APPLICATIONS, and PRINCIPAL PROPAGATION. Under ACCESS CONTROL, there's a section titled 'Mapping Virtual To Internal System (6)' with a table:

Status	Virtual Host	Internal Host	Check Result	Protocol	Back-end Type	Actions
✓	abapmessageserver.hana.cloud:sapms...	xxxx.wdf.sap.corp:sapmsv9U	Unchecked	RFC	ABAP System	
✓	abapserver.hana.cloud:sapgw42	xxxx.wdf.sap.corp:sapgw51	Unchecked	RFC	ABAP System	
✓	asd:123	asd:123	Unchecked	HTTP	ABAP System	
✓	naboo.empire:8080	xxxx.dhcp.wdf.sap.corp:5555	Unchecked	HTTP	ABAP System	
✓	sccmaster:80	xxxx.wdf.sap.corp:443	Unchecked	HTTP	SAP Business Connector	
✓	tatooine.empire:443	xxxx.dhcp.wdf.sap.corp:4443	Unchecked	HTTP	ABAP System	

Below this is a section titled 'Resources Of asd:123 (3)' with a table:

Status	URL Path	Access Policy	Actions
✗	/everybody	Path And All Sub-Paths	
✗	/public	Path Only (Sub-Paths Are Excluded)	
✓	/qwe	Path And All Sub-Paths	

Back to [Use Scenarios for Resources \[page 401\]](#)

Back to [Tasks \[page 399\]](#)

1.2.2.5 Configuration REST APIs

You can use a set of APIs to perform the basic setup of the Cloud Connector.

Context

As of version 2.11, the Cloud Connector provides several REST APIs that let you configure a newly installed Cloud Connector. The configuration options correspond to the following steps:

- [Initial Configuration \[page 322\]](#)
- [Managing Subaccounts \[page 338\]](#)
- [Configure Access Control \[page 379\]](#)
- [Master and Shadow Administration \[page 521\]](#)

All configuration APIs start with the following string: /api/v1/configuration.

i Note

Use the same host and port for the REST APIs that you use to access the Cloud Connector.

Prerequisites

- After installing the Cloud Connector, you have changed the initial password.
- You have specified the high availability role of the Cloud Connector (master or shadow).
- You have configured the proxy on the master instance if required for your network.

Request and Response Format

Requests and responses are coded in JSON format. The following example shows the request payload {description:<value>} coded in JSON:

Sample Code

```
{"description": "very helpful description"}
```

Values that represent a date are given as a UTC long number, which is the number of milliseconds since 1 January 1970 00:00:00 UT (GMT+00).

In case of errors, the HTTP status code is 4xx. Error details are supplied in the response body in JSON format:

```
{type: <type>, message: <message>}
```

type can have one of these values:

INVALID_REQUEST, ILLEGAL_STATE, NOT_FOUND, INVALID_CONFIGURATION, RUNTIME_FAILURE,
SHADOW_UPDATE, FORBIDDEN_REQUEST

Security and Sessions

The Cloud Connector supports basic authentication and form-based authentication. Once authenticated, the client can keep the session and execute subsequent requests in this session. A session avoids the overhead caused by authentication.

You can get the session ID from the response header field `<Set-cookie>` (as `JSESSIONID=<session ID>`), and send it in the request header `Cookie: JSESSIONID=<session Id>`.

The Cloud Connector uses CSRF tokens to prevent CSRF (cross-site request forgery) attacks. Upon first request, a CSRF token is generated and sent back in the response header in field `<X-CSRF-Token>`. The client application must keep this token and send it in all subsequent requests as header field `<X-CSRF-Token>`, together with the session cookie as described above.

i Note

If the request header field `<Connection>` has the value `close` (as opposed to `keep-alive`), no CSRF token is generated. If you want to make stateful, session-based REST calls, use `Connection: keep-`

alive, extract the CSRF token from the first response header, and include it in all request headers. Otherwise, use `Connection: close` and always submit user and password through basic authentication.

An inactive session causes a timeout at some point, and will consequently be removed. A request using an expired session receives a login page (`Content-type: text/html`). The status code of the response is 200 in this case. The only way to detect an expired session is to check the content type and status code. Content type `text/html` in a connection with status code 200 indicates an expired session.

For security reasons, a session should be closed or invalidated once it is not needed anymore. You can achieve this by including `Connection: close` in the header of the final call of the relevant session. As a result, the Cloud Connector invalidates the session. Subsequent attempts to send a request in the context of that session will respond with a login page as described above.

User Roles

As of Cloud Connector 2.12, the REST API supports different user roles. Depending on the role, an API grants or denies access. In default configuration, the Cloud Connector uses local user storage and supports the single user `Administrator` (administrator role). Using LDAP user storage, you can use various users (see also [Configure Named Cloud Connector Users \[page 512\]](#)):

Role	Technical Role Name	Authorization
Administrator	admin or sccadmin	All CRUD operations.
Support	sccsupport	Edit log and trace configuration.
Display	sccdisplay	Read configuration.
Monitoring	sccmonitoring	Read monitoring information.

Return Codes

Successful requests return the code 200, or, if there is no content, 204. `POST` actions that create new entities return 201, with the location link in the header.

The following general errors can be returned by each API:

400 – invalid request. For example, if parameters are invalid or the API is not supported anymore, an unexpected state occurs and in case of other non-critical errors.

401 – authorization required.

403 – the current Cloud Connector instance does not allow changes. For example, the instance has been assigned to the shadow role and therefore does not allow configuration changes, or the user role does not have required permission.

405 – an entity does not support the requested operation.

409 – current state of the Cloud Connector does not allow changes. For example, the password has to be changed first.

415 – unexpected mime type.

500 – operation failed.

The status line provides details explaining the reason.

Most APIs may also return specific error details, depending on the situation. Such errors are mentioned in the corresponding API description.

i Note

The error texts depend on the request locale.

Hypertext Application Language

Entities returned by the APIs contain links as suggested by the current draft *JSON Hypertext Application Language* (see <https://tools.ietf.org/html/draft-kelly-json-hal-08>).

Available APIs

API Name	Use
Common Properties [page 409]	<ul style="list-style-type: none">• Read common properties• Edit common properties
High Availability Settings [page 410]	<ul style="list-style-type: none">• Get high availability settings• Set high availability settings
Proxy Settings [page 421]	<ul style="list-style-type: none">• Get proxy settings• Set proxy settings• Remove proxy settings
Authentication and UI Settings [page 424]	<ul style="list-style-type: none">• Read authentication settings• Edit authentication settings• Edit LDAP authentication settings

API Name	Use
Certificate Management for Backend Communication [page 432]	<ul style="list-style-type: none"> Get description for CA certificate for principal propagation Get binary content of CA certificate for principal propagation Create a self-signed CA certificate for principal propagation Create a certificate signing request for CA certificate for principal propagation Upload a signed certificate chain as CA certificate for principal propagation Upload a PKCS#12 certificate as CA certificate for principal propagation Delete CA certificate for principal propagation Get description for system certificate Get binary content of system certificate Create a self-signed system certificate Create a certificate signing request for a system certificate Upload a signed certificate chain as system certificate Upload a PKCS#12 certificate as system certificate Delete system certificate
Solution Management Configuration [page 445]	<ul style="list-style-type: none"> Get solution management configuration Set solution management configuration and turn on reporting Turn off reporting Download current LMDB XML report
Backup [page 447]	<ul style="list-style-type: none"> Create backup configuration Restore backup configuration
Subaccount [page 449]	<ul style="list-style-type: none"> Get list of subaccounts Create subaccount Delete subaccount Edit subaccount Connect or disconnect subaccount Extend validity of subaccount Create or change recovery subaccount Delete recovery subaccount Extend validity of recovery subaccount Get subaccount configuration
System Mappings [page 458]	<ul style="list-style-type: none"> Get system mappings Create system mapping Delete system mapping Delete all system mappings Edit system mapping

API Name	Use
System Mapping Resources [page 463]	<ul style="list-style-type: none"> Get system mapping resources Get system mapping resource Create system mapping resource Edit system mapping resource Delete system mapping resource Delete all system mapping resources
Domain Mappings [page 467]	<ul style="list-style-type: none"> List domain mappings Create domain mappings Delete domain mappings Edit domain mappings
Subaccount Service Channels [page 470]	<ul style="list-style-type: none"> Get service channels Create service channel Delete service channel Edit service channel Enable or disable service channel

Related Information

[Examples \[page 473\]](#)

1.2.2.5.1 Common Properties

Read and edit the Cloud Connector's common properties via API.

Get Common Properties

URI	/api/v1/configuration/connector
Method	GET
Request	
Response	{ha, description}
Errors	

Roles	Administrator, Display, Support
-------	---------------------------------

- **Response Properties:**
 - ha: Cloud Connector instance assigned to a high-availability role. Its value is either the string *master* or *shadow*.
 - description: Description of the Cloud Connector.

Set Description (Master Only)

URI	/api/v1/configuration/connector
Method	PUT
Request	{description}
Response	{ha, description}
Errors	INVALID_REQUEST
Roles	Administrator

- **Request Properties:**

description: A string; use an empty string to remove the description.
- **Response Properties:**
 - ha: Cloud Connector instance assigned to a high-availability role. Its value is either the string *master* or *shadow*.
 - description: Description of the Cloud Connector.
- **Errors:**

INVALID_REQUEST ((400)): The value of `description` is not a JSON string.

i Note

`null` is not a JSON string.

1.2.2.5.2 High Availability Settings

Read and edit the high availability settings of a Cloud Connector instance via API.

When installing a Cloud Connector instance, you usually define its high availability role (master or shadow instance) during initial configuration, see [Change your Password and Choose Installation Type \[page 324\]](#).

If the high availability role was not defined before, you can set the master or shadow role via this API.

If a shadow instance is connected to the master, this API also lets you switch the roles: the master instance requests the shadow instance to take over the master role, and then takes the shadow role itself.

i Note

Editing the high availability settings is only allowed on the master instance, and supports only **shadow** as input.

Get High Availability Settings

URI	/api/v1/configuration/connector/haRole
Method	GET
Request	
Response	"<HA role>"
Errors	
Roles	Administrator, Display, Support

Example

```
curl -i -k -u Administrator:<password> https://localhost:8443/api/v1/configuration/connector/haRole
```

Set High Availability Settings

Use this API if you want to set the role of a fresh installation (no role assigned yet).

As of version 2.12.0, this API also allows to switch the roles if a shadow instance is connected to the master. In this case, the API is only allowed on the master instance and supports only the value **shadow** as input. The master instance requests the shadow instance to take over the master role and then assumes the shadow role itself.

URI	/api/v1/configuration/connector/haRole
Method	POST
Request	"master" or "shadow"
Response	
Errors	ILLEGAL_STATE, INVALID_REQUEST
Roles	Administrator

Errors:

- INVALID_REQUEST (400): If a high-availability role other than "master" or "shadow" is supplied.
- ILLEGAL_STATE (409): If changing the high-availability role is not possible; changing the role is only possible if no role has been assigned, or if the current role is master and a shadow system is connected.

Example

```
curl -i -k -H 'Content-Type: application/json' -d '"shadow"' -u
Administrator:<password> -X POST https://localhost:8443/api/v1/configuration/
connector/haRole
```

Related Information

[Master Instance Configuration \[page 412\]](#)
[Shadow Instance Configuration \[page 416\]](#)

1.2.2.5.2.1 Master Instance Configuration

Read and edit the high availability settings for a Cloud Connector master instance via API.

i Note

The APIs below are available as of Cloud Connector version 2.13.0.

! Restriction

These APIs are only permitted on a Cloud Connector master instance. The shadow instance rejects the requests with error code 400 – *Invalid Request*.

Get Configuration

URI `/api/v1/configuration/connector/ha/master/config`

Method `GET`

Request

Response `{haEnabled, allowedShadowHost}`

Errors

Roles Administrator, Display, Support

Response Properties:

- `haEnabled`: a Boolean value that indicates whether or not a shadow system is allowed to connect
- `allowedShadowHost`: the name of the shadow host (a string) that is allowed to connect; an empty string signifies that any host is allowed to connect as shadow.

Example

```
curl -i -k -u <user>:<password> -X GET https://<host>:<port>/api/v1/configuration/connector/ha/master/config
```

Set Configuration

URI `/api/v1/configuration/connector/ha/master/config`

Method `PUT`

Request `{haEnabled, allowedShadowHost}`

Response 204 on success

Errors	INVALID_REQUEST
Roles	Administrator

Response Properties:

- haEnabled: Boolean value that indicates whether or not a shadow system is allowed to connect.
- allowedShadowHost: Name of the shadow host (a string) that is allowed to connect. An empty string means that any host is allowed to connect as shadow.

Errors:

- INVALID_REQUEST (400): if the name of the shadow host is not a valid host name

Example

```
curl -i -k -u <user>:<password> -X PUT -H 'Content-Type: application/json' -d
'{"haEnabled":true}' https://<host>:<port>/api/v1/configuration/connector/ha/
master/config
```

Get State

URI	/api/v1/configuration/connector/ha/master/state
Method	<i>GET</i>
Request	
Response	{state, shadowHost}
Errors	
Roles	Administrator, Display, Support

Response Properties:

- state: One of the following strings: ALONE, BINDING, CONNECTED or BROKEN.
- shadowHost: Connected shadow host (a string)

Example

```
curl -i -k -u <user>:<password> -X GET https://<host>:<port>/api/v1/
configuration/connector/ha/master/state
```

Set State

URI	/api/v1/configuration/connector/ha/master/state
Method	<i>POST</i>
Request	
Response	{op}
Errors	ILLEGAL_STATE, INVALID_REQUEST
Roles	Administrator

Request Properties:

The value of property `op` is one of the following strings:

- `SWITCH`: Switch roles with shadow
- `FORCE_SWITCH`: Take over the shadow role, even if shadow instance does not respond.

Errors:

- `INVALID_REQUEST` (400): Value of property `op` is neither `SWITCH` nor `FORCE_SWITCH`.
- `ILLEGAL_STATE` (409): Master system is in a state that does not permit the requested operation.

Example

```
curl -i -k -u <user>:<password> -X POST -H 'Content-Type: application/json' -d
'{"op":"SWITCH"}' https://<host>:<port>/api/v1/configuration/connector/ha/master/
state
```

Reset

A successful call to this API restores default values for all settings related to high availability on the master side.

⚠ Caution

Do not perform this call if the shadow is connected to a master.

URI	/api/v1/configuration/connector/ha/master/state
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	ILLEGAL_STATE
Roles	Administrator

Errors:

- ILLEGAL_STATE (409): A shadow instance is connected to the master.

Example

```
curl -i -k -u <user>:<password> -X DELETE https://<host>:<port>/api/v1/configuration/connector/ha/master/state
```

1.2.2.5.2.2 Shadow Instance Configuration

Read and edit the configuration settings for a Cloud Connector shadow instance via API (available as of Cloud Connector version 2.12.0, or, where mentioned, as of version 2.13.0).

i Note

The APIs below are only permitted on a Cloud Connector shadow instance. The master instance will reject the requests with error code 403 – FORBIDDEN_REQUEST.

Get Configuration

URI	/api/v1/configuration/connector/ha/shadow/config
Method	<i>GET</i>
Request	
Response	{masterHost, masterPort, ownHost, checkIntervalInSeconds, takeoverDelayInSeconds, connectTimeoutInMillis, requestTimeoutInMillis}
Errors	
Roles	Administrator, Display, Support

Response Properties:

- masterHost: Host name of the master instance (string)
 - masterPort: Port of the master instance (string or number)
 - ownHost: Host name of the shadow instance (string)
 - checkIntervalInSeconds: Time between two health checks against the master instance (number)
 - takeoverDelayInSeconds: The time a master instance may stay unreachable before the shadow instance takes over (number)
 - connectTimeoutInMillis: Timeout for connection attempts between shadow and master instance (number)
 - requestTimeoutInMillis: Timeout for requests between shadow and master instance (number)

i Note

This API may take some time to fetch the own hosts from the environment.

Example

```
curl -i -k -u <user>:<password> -X GET https://<host>:<port>/api/v1/configuration/connector/ha/shadow/config
```

Set Configuration

URI	/api/v1/configuration/connector/ha/shadow/config
Method	<i>PUT</i>
Request	{masterPort, masterHost, ownHost, checkIntervalInSeconds, takeoverDelayInSeconds, connectTimeoutInMillis, requestTimeoutInMillis}
Response	{masterPort, masterHost, ownHost, checkIntervalInSeconds, takeoverDelayInSeconds, connectTimeoutInMillis, requestTimeoutInMillis}
Errors	
Roles	Administrator

Request Properties:

- `masterHost`: Host name of the master instance (string)
- `masterPort`: Port of the master instance (string or number)
- `ownHost`: Host name of the shadow instance (string)
- `checkIntervalInSeconds`: Time between two health checks against the master instance (number)
- `takeoverDelayInSeconds`: The time a master instance may stay unreachable before the shadow instance takes over (number)
- `connectTimeoutInMillis`: Timeout for connection attempts between shadow and master instance (number)
- `requestTimeoutInMillis`: Timeout for requests between shadow and master instance (number)

Response Properties:

See *Request Properties*.

Example

```
curl -i -k -u <user>:<password> -X PUT https://<host>:<port>/api/v1/configuration/connector/ha/shadow/config
-H 'Content-Type: application/json' -d '{"masterHost":"localhost", "masterPort":8443, "ownHost":"localhost", "checkIntervalInSeconds":30, "takeoverDelayInSeconds":10, "connectTimeoutInMillis":1000, "requestTimeoutInMillis":12000}'
```

Get State

i Note

Available as of version 2.13.0.

URI	/api/v1/configuration/connector/ha/shadow/state
Method	<i>GET</i>
Request	
Response	{state, ownHosts, stateMessage, masterVersions}
Errors	
Roles	Administrator, Display, Support

Response Properties:

- **state**: Possible string values are: INITIAL, DISCONNECTED, DISCONNECTING, HANDSHAKE, INITSYNC, READY, or LOST.
- **ownHosts**: List of alternative host names for the shadow instance.
- **stateMessage**: Message providing details on the current state. This property may not always be present. Typically, this property is available if an error occurred (for example, a failed attempt to connect to the master instance).
- **masterVersions**: Overview of relevant component versions of the master system, including a flag (*property ok*) that indicates whether or not there are incompatibility issues because of differing master and shadow versions.

i Note

This property is only available if the shadow instance is connected to the master instance, or if there has been a successful connection to the master system at some point in the past.

Example

```
curl -i -k -u <user>:<password> -X GET https://<host>:<port>/api/v1/configuration/connector/ha/shadow/state
```

Change State

URI	/api/v1/configuration/connector/ha/shadow/state
Method	POST
Request	
Response	{op, user, password}
Errors	INVALID_REQUEST, ILLEGAL_STATE
Roles	Administrator

Request Properties:

- op: String value representing the state change operation. Possible values are CONNECT or DISCONNECT.
- user: User for logon to the master instance
- password: Password for logon to the master instance

Errors:

- INVALID_REQUEST (400): Invalid or missing property values were supplied; this includes wrong user or password
- ILLEGAL_STATE (409): The requested operation cannot be executed given the current state of master and shadow instance. This typically means the master instance does not allow high availability.

i Note

The logon credentials are used for initial logon to master instance only. If a shadow instance is disconnected from its master instance, it will reconnect to the (same) master instance using a certificate. Hence, user and password can be omitted when reconnecting.

Example

```
curl -i -k -u <user>:<password> -X POST https://<host>:<port>/api/v1/configuration/connector/ha/shadow/state
-H 'Content-Type: application/json' -d '{"op":"CONNECT",
"user":"<user on master>", "password":"<password>"}'
curl -i -k -u <user>:<password> -X POST https://<host>:<port>/api/v1/configuration/connector/ha/shadow/state
-H 'Content-Type: application/json' -d '{"op":"DISCONNECT"}'
```

Reset

i Note

Available as of version 2.13.0.

A successful call to this API deletes master host and port, and restores default values for all other settings related to a connection to the master.

⚠ Caution

Do not perform this call if the shadow is connected to a master.

URI	/api/v1/configuration/connector/ha/shadow/state
Method	<i>DELETE</i>
Request	
Response	
Errors	ILLEGAL_STATE
Roles	Administrator

Errors:

- ILLEGAL_STATE (409): the shadow is currently connected to a master.

Example

```
curl -i -k -u <user>:<password> -X DELETE https://<host>:<port>/api/v1/configuration/connector/ha/shadow/state
```

1.2.2.5.3 Proxy Settings

Read and edit the Cloud Connector's proxy settings via API.

Get Proxy Settings

URI	/api/v1/configuration/connector/proxy
Method	GET
Request	
Response	{host, port, user}
Errors	
Roles	Administrator, Display, Support

Response Properties:

- host: the name of the proxy host (a string)
- port: the port of the proxy host (a string)
- user: the user name (a string)

↳ Sample Code

```
curl -ik -u Administrator:<password> https://localhost:8443/api/v1/  
configuration/connector/proxy
```

Set Proxy Settings (Master Only)

URI	/api/v1/configuration/connector/proxy
Method	PUT
Request	{host, port, user, password}
Response	
Errors	INVALID_REQUEST, FORBIDDEN_REQUEST
Roles	Administrator

Request Properties:

- host: the name of the proxy host (a string)
- port: the port of the proxy host (a string)

- `user`: the user name (a string)
- `password`: the password (a string - optional)

Errors:

- `INVALID_REQUEST` (400): invalid values were supplied, or mandatory values are missing.
- `FORBIDDEN_REQUEST` (403): the target of the call is a shadow instance.

The following example sets empty `user` and `password`.

 **Sample Code**

```
curl -ik -u Administrator:<password> https://localhost:8443/api/v1/
configuration/connector/proxy -X PUT -H 'Content-Type: application/json' -d
'{"host":"proxy", "port":"8080"}'
```

This request removes the proxy configuration.

 **Sample Code**

```
curl -ik -u Administrator:<password> https://localhost:8443/api/v1/
configuration/connector/proxy -X PUT -H 'Content-Type: application/json' -d
'{}'
```

Remove Proxy Settings (Master Only)

URI	/api/v1/configuration/connector/proxy
Method	DELETE
Request	
Response	204 on success
Errors	<code>FORBIDDEN_REQUEST</code>
Roles	Administrator

Errors:

- `FORBIDDEN_REQUEST` (403): the target of the call is a shadow instance.

 **Sample Code**

```
curl -ik -u Administrator:<password> https://localhost:8443/api/v1/
configuration/connector/proxy -X DELETE
```

1.2.2.5.4 Authentication and UI Settings

Read and edit the Cloud Connector's authentication and UI settings via API.

Get Authentication Settings

URI	/api/v1/configuration/connector/authentication
Method	<i>GET</i>
Request	
Response	{type, configuration}
Errors	
Roles	Administrator, Display, Support

Response Properties:

- **type**: The authentication type, which is one of the following strings: *basic* or *ldap*.
- **configuration**: The configuration of the active LDAP authentication. This property is only available if **type** is *ldap*. Its value is an object with properties that provide details on LDAP configuration.

Example

```
curl -i -k -H 'Accept:application/json'  
-u Administrator:<password> -X GET https://<scchost>:8443/api/v1/configuration/  
connector/authentication
```

Change Basic Authentication User

URI	/api/v1/configuration/connector/authentication/basic
Method	<i>PUT</i>
Request	{password, user}
Response	204 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Properties:

- `password`: The current password (a string)
- `user`: The new user name (a string), overwriting the current user name.

Errors:

- `INVALID_REQUEST` (400): Current password is wrong.

Change Basic Authentication Password

URI	/api/v1/configuration/connector/authentication/basic
Method	<i>PUT</i>
Request	{oldPassword, newPassword}
Response	204 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Properties:

- `oldPassword`: The current password, about to be changed (a string)
- `newPassword`: The new password (a string)

Errors:

- `INVALID_REQUEST` (400): Passwords are the same or current password is wrong.

Example

```
curl -i -k -H 'Content-Type:application/json' -d '{"oldPassword":"manage", "newPassword":"test"}' -u Administrator:manage -X PUT https://localhost:8443/api/v1/configuration/connector/authentication/basic
```

Change LDAP Authentication

⚠ Caution

The Cloud Connector will restart if the request was successful. There is no test that confirms login will work afterwards. If you run into problems you can revert to basic authentication by executing the script `useFileUserStore` located in the root directory of your Cloud Connector installation.

URI	/api/v1/configuration/connector/authentication/ldap
Method	<i>PUT</i>
Request	{enable, configuration}
Response	204 on success
Errors	INVALID_REQUEST, INVALID_CONFIGURATION
Roles	Administrator

Request Properties:

- `enable`: Boolean flag that indicates whether or not to employ LDAP authentication.
- `configuration`: The LDAP configuration, a JSON object with the properties `{config, hosts, user, password, customAdminRole, customDisplayRole, customMonitoringRole, customSupportRole}`.
 - Property `hosts` is an array. Each element of the array defines a host, again specified through a JSON object, with the properties `{host, port, isSecure}`, accepting string, string (or number), and Boolean values, respectively.
 - All properties of the top-level object except `hosts` accept string values.
 - Properties `config` and `hosts` are mandatory. The array of hosts needs to have at least one element.
 - All other properties are optional.

Errors:

- `INVALID_REQUEST` (400): Configuration is invalid.

- INVALID_CONFIGURATION (409): LDAP server is not accessible (does not respond).

i Note

In both error cases nothing is stored, and LDAP authentication is disabled.

Example

```
curl -i -k -H 'Content-Type: application/json' -u Administrator:<password> -X
PUT https://localhost:8443/api/v1/configuration/connector/authentication/ldap -d
'{"enable":true, "configuration":{"hosts":[{"host":"ldaphost", "port":10389,
"isSecure":false}], "config":{"roleBase="ou=groups,dc=scc\"
" roleName="cn\"
" roleSearch="(uniqueMember={0})\"
" userBase="ou=users,dc=scc\"
" userSearch=
"(uid={0})\"
" , "user": "ldapadmin", "password":<ldapadminpassword>"} }'
```

Get Description for UI Certificate

This API returns a textual description for the system certificate.

i Note

Available as of version 2.13.0.

URI	/api/v1/configuration/connector/ui/uiCertificate
-----	--

Method	GET
--------	-----

Response	{ subjectDN, issuer, notBeforeTimeStamp, notAfterTimeStamp, subjectAltNames }
----------	---

Errors

Roles	Administrator, Display, Support
-------	---------------------------------

Response Properties:

- subjectDN: The subject distinguished name (a string)
- issuer: The issuer (a string)
- notBeforeTimeStamp: Timestamp of the beginning of the validity period (a UTC long number)
- notAfterTimeStamp: Timestamp of the end of the validity period (a UTC long number)

- `subjectAltNames`: Subject alternative names, as an array of objects with properties `type` and `value`. The value of property `type` is one of the following strings: IP, DNS, URI, or RFC822. The value of property `value` is the associated value (a string).

i Note

`subjectAltNames` is not present if there are no subject alternative names.

Example

```
curl -i -k -H "Accept: application/json" -u <user>:<password> -X GET https://<host>:<port>/api/v1/configuration/connector/ui/uiCertificate
```

Create a Self-Signed UI Certificate

i Note

Available as of version 2.13.0.

URI	/api/v1/configuration/connector/ui/uiCertificate
Method	<i>POST</i>
Request	{ <code>type</code> , <code>subjectDN</code> , <code>subjectAltNames</code> }
Response	201 on success
Errors	
Roles	Administrator

Request Properties:

- `type`: The string `selfsigned`
- `subjectDN`: The subject distinguished name (a string)
- `subjectAltNames`: Subject alternative names, as an array of objects with properties `type` and `value`. The value of property `type` is one of the following strings: IP, DNS, URI, or RFC822. The value of property `value` is the associated value (a string). This property is optional.

The UI certificate created this way has a validity of 1 year.

Example

```
curl -k -u Administrator:<password> -X POST -H "Content-Type: application/json"  
--data '{"type":"selfsigned", "subjectDN":"CN=me"}'  
https://<host>:<port>/api/v1/configuration/connector/ui/uiCertificate
```

Create a Certificate Signing Request for a UI Certificate

i Note

Available as of version 2.13.0.

URI	/api/v1/configuration/connector/ui/uiCertificate
Method	POST
Request	{type, subjectDN, subjectAltNames}
Response	PEM-encoded certificate request
Errors	
Roles	Administrator

Request Properties:

- type: The string `csr`
- subjectDN: The subject distinguished name (a string)
- subjectAltNames: Subject alternative names, as an array of objects with properties `type` and `value`. The value of property `type` is one of the following strings: IP, DNS, URI, or RFC822. The value of property `value` is the associated value (a string). This property is optional.

Example

```
curl -k -u Administrator:<password> -X POST -H "Content-Type: application/json"  
--data '{"type":"csr", "subjectDN":"CN=me"}'  
https://<host>:<port>/api/v1/configuration/connector/ui/uiCertificate -o  
csr.pem
```

Upload a Signed Certificate Chain as UI Certificate

i Note

Available as of version 2.13.0.

URI	/api/v1/configuration/connector/ui/uiCertificate
Method	<i>PATCH</i>
Request	multipart/form-data with the following form parameter: signedCertificate
Response	201 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Parameters:

- signedCertificate: The signed certificate and CA certificate chain (PEM-encoded)

Errors:

- INVALID_REQUEST (400): The certificate chain provided does not match the most recent certificate request, or it is not a certificate chain in the proper format (PEM-encoded).

Example

```
curl -i -k -u <user>:<password> -X PATCH -F signedCertificate=@<signedchain.pem>
https://<host>:<port>/api/v1/configuration/connector/ui/uiCertificate
```

Example

For test purposes, you can sign the certificate signing request with keytool.

```
keytool -genkeypair -keyalg RSA -keysize 1024 -alias mykey -dname "cn=very
trusted, c=test" -validity 365 -keystore ca.ks -keypass testit -storepass testit
keytool -gencert -rfc -infile csr.pem -outfile signedcsr.pem -alias mykey -
keystore ca.ks -keypass testit -storepass testit
keytool -exportcert -rfc -file ca.pem -alias mykey -keystore ca.ks -keypass
testit -storepass testit
```

```
cat signedcsr.pem ca.pem > signedchain.pem
```

Upload a PKCS#12 Certificate as UI Certificate

i Note

Available as of version 2.13.0.

URI	/api/v1/configuration/connector/ui/uiCertificate
Method	<i>PUT</i>
Request	multipart/form-data with the following form parameters: pkcs12 password keyPassword
Response	204 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Parameters:

- `pkcs12`: Contents of PKCS#12 file
- `password`: Password for decrypting the PKCS#12 file
- `keyPassword`: Optional password for the private key

Errors:

- `INVALID_REQUEST` (400): Contents of PKCS#12 or password are invalid

i Note

`keyPassword` is optional. If missing, `password` is used to decrypt the pkcs#12 file and the private key.

Example

```
curl -i -k -u <user>:<password> https://<host>:<port>/api/v1/configuration/connector/ui/uiCertificate -X PUT -F 'password=<p12Password>' -F pkcs12=@<p12file>
```

Example

For test purposes, you can create an own self-signed pkcs#12 certificate with keytool.

```
keytool -genkeypair -alias key -keyalg RSA -keysize 2048 -validity 365 -keystore test.p12 -storepass test20 -storetype PKCS12 -dname 'CN=test'
```

1.2.2.5.5 Certificate Management for Backend Communication

Manage a CA certificate for principal propagation or a system certificate via API.

i Note

The APIs below are available as of Cloud Connector version 2.13.

There are two similar sets of APIs for system certificate and CA certificate for principal propagation.

i Note

Some of the APIs list a parameter `subjectAltNames` (*subject alternative names* or SAN) for the request or response object. This parameter is an array of objects with the following properties:

- `type`: one of the strings *DNS*, *URI*, *IP*, or *RFC822*.
- `value`: the value associated with the chosen type.

- [CA Certificate for Principal Propagation: APIs \[page 432\]](#)
- [System Certificate: APIs \[page 438\]](#)

1.2.2.5.5.1 CA Certificate for Principal Propagation: APIs

Manage a CA certificate for principal propagation via API.

i Note

The APIs below are available as of Cloud Connector version 2.13.0.

Get Description for a CA Certificate for Principal Propagation

URI	/api/v1/configuration/connector/onPremise/ppCaCertificate
Method	<i>GET</i>
Header	Accept: application/json
Response	{ subjectDN, issuer, notBeforeTimeStamp, notAfterTimeStamp, subjectAltNames }
Errors	NOT_FOUND
Roles	Administrator, Display, Support

Response Properties:

- **subjectDN**: the subject distinguished name (a string)
- **issuer**: the issuer (a string)
- **notBeforeTimeStamp**: timestamp of the beginning of the validity period (a UTC long number)
- **notAfterTimeStamp**: timestamp of the end of the validity period (a UTC long number)
- **subjectAltNames**: subject alternative names (see [Certificate Management for Backend Communication \[page 432\]](#) for details).

Errors:

- NOT_FOUND (404): there is no CA certificate for principal propagation.

i Note

subjectAltNames is not present if there are no subject alternative names.

Example

```
curl -i -k -H "Accept: application/json" -u <user>:<password> -X GET https://<host>:<port>/api/v1/configuration/connector/onPremise/ppCaCertificate
```

Get Binary Content of a CA Certificate for Principal Propagation

URI	/api/v1/configuration/connector/onPremise/ppCaCertificate
Method	<i>GET</i>
Header	Accept: application/pkix-cert
Response	Binary data of the certificate.
Errors	NOT_FOUND
Roles	Administrator, Display, Support

Response:

- Success: the binary data of the certificate; you can verify the downloaded certificate by storing it in file `ppca.crt`, for instance, and then running

```
keytool -printcert -file ppca.crt
```
- Failure: an error in the usual JSON format; the content type of the response is *application/json* in this case.

Errors:

- NOT_FOUND (404): there is no CA certificate for principal propagation.

Example

```
curl -k -H "Accept: application/pkix-cert" -u <user>:<password> -X GET --output sys.crt https://<host>:<port>/api/v1/configuration/connector/onPremise/ppCaCertificate
```

Create a Self-Signed CA Certificate for Principal Propagation (Master Only)

URI	/api/v1/configuration/connector/onPremise/ppCaCertificate
-----	---

Method	<i>POST</i>
Request	{type, subjectDN, subjectAltNames}
Response	201 on success
Errors	
Roles	Administrator

Request Properties:

- type: the string *selfsigned*
- subjectDN: the subject distinguished name (a string)
- subjectAltNames: subject alternative names (see [Certificate Management for Backend Communication \[page 432\]](#) for details). This property is optional.

The certificate created this way has a validity of 1 year.

Example

```
curl -i -k -H "Accept: application/json" -H "Content-Type: application/json" -u <user>:<password> -X POST --data '{"type":"selfsigned", "subjectDN":"CN=me"}' https://<host>:<port>/api/v1/configuration/connector/onPremise/ppCaCertificate
```

Create a Certificate Signing Request for a CA Certificate for Principal Propagation (Master Only)

URI	/api/v1/configuration/connector/onPremise/ppCaCertificate
Method	<i>POST</i>
Request	{type, subjectDN, subjectAltNames}
Response	PEM-encoded certificate request
Errors	
Roles	Administrator

Request Properties:

- type: the string `selfsigned`
- subjectDN: the subject distinguished name (a string)
- subjectAltNames: subject alternative names (see [Certificate Management for Backend Communication \[page 432\]](#) for details). This property is optional.

Example

```
curl -k -u <user>:<password> -X POST -H "Content-Type: application/json" --data
'{"type":"csr", "subjectDN":"CN=me"}'
https://<host>:<port>/api/v1/configuration/connector/onPremise/ppCaCertificate -
o csr.pem
```

Upload a Signed Certificate Chain as CA Certificate for Principal Propagation (Master Only)

URI	/api/v1/configuration/connector/onPremise/ppCaCertificate
Method	PATCH
Request	multipart/form-data with the following parameter: signedCertificate.
Response	201 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Properties:

- signedCertificate: the signed certificate and CA certificate chain (PEM-encoded)

Errors:

- INVALID_REQUEST (400): the certificate chain provided does not match the most recent certificate request, or it is not a certificate chain in the correct format (PEM-encoded).

Example

```
curl -i -k -u <user>:<password> -X PATCH -F signedCertificate=@<signedchain.pem>
https://<host>:<port>/api/v1/configuration/connector/onPremise/ppCaCertificate
```

Example: Sign The Certificate Signing Request

```
keytool -genkeypair -keyalg RSA -keysize 1024 -alias mykey -dname "cn=very
trusted, c=test" -validity 365 -keystore ca.ks -keypass testit -storepass testit
keytool -gencert -rfc -infile csr.pem -outfile signedcsr.pem -alias mykey -
keystore ca.ks -keypass testit -storepass testit
keytool -exportcert -rfc -file ca.pem -alias mykey -keystore ca.ks -keypass
testit -storepass testit
cat signedcsr.pem ca.pem > signedchain.pem
```

Upload a PKCS#12 Certificate as CA Certificate for Principal Propagation (Master Only)

URI	/api/v1/configuration/connector/ onPremise/ppCaCertificate
Method	<i>PUT</i>
Request	Multipart/form-data with the following form parameters: pkcs12, password, keyPassword
Response	204 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Parameters:

- `pkcs12`: contents of PKCS#12 file
- `password`: password for decrypting PKCS#12 file
- `keyPassword`: optional password for the private key

Errors:

- `INVALID_REQUEST` (400): contents of PKCS#12 or password are invalid

i Note

`keyPassword` is optional. If it is missing, `password` is used to decrypt the pkcs#12 file and the private key.

Example

```
curl -i -k -u <user>:<password> https://<host>:<port>/api/v1/configuration/connector/onPremise/ppCaCertificate -X PUT -F 'password=<p21Password>' -F pkcs12=@<p12file>
```

Create an own self-signed pkcs#12 certificate for tests with:

```
keytool -genkeypair -alias key -keyalg RSA -keysize 2048 -validity 365 -keystore test20 -storepass test20 -storetype PKCS12 -dname 'CN=test'
```

Delete a CA Certificate for Principal Propagation (Master Only)

URI	/api/v1/configuration/connector/onPremise/ppCaCertificate
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	NOT_FOUND
Roles	Administrator

Errors:

- NOT_FOUND (404): there is no CA certificate for principal propagation.

Example

```
curl -k -H "Accept: application/json" -u <user>:<password> --request DELETE https://localhost:8443/api/v1/configuration/connector/onPremise/ppCaCertificate
```

1.2.2.5.5.2 System Certificate: APIs

Manage a system certificate via API.

i Note

The APIs below are available as of Cloud Connector version 2.13.0.

Get Description for a System Certificate

URI	/api/v1/configuration/connector/onPremise/systemCertificate
Method	<i>GET</i>
Header	Accept: application/json
Response	{ subjectDN, issuer, notBeforeTimeStamp, notAfterTimeStamp, subjectAltNames }
Errors	NOT_FOUND
Roles	Administrator, Display, Support

Response Properties:

- `subjectDN`: the subject distinguished name (a string)
- `issuer`: the issuer (a string)
- `notBeforeTimeStamp`: timestamp of the beginning of the validity period (a UTC long number)
- `notAfterTimeStamp`: timestamp of the end of the validity period (a UTC long number)
- `subjectAltNames`: subject alternative names (see [Certificate Management for Backend Communication \[page 432\]](#) for details).

Errors:

- NOT_FOUND (404): there is no system certificate.

i Note

`subjectAltNames` is not present if there are no subject alternative names.

Example

```
curl -i -k -H "Accept: application/json" -u <user>:<password> -X GET https://<host>:<port>/api/v1/configuration/connector/onPremise/systemCertificate
```

Get Binary Content of a System Certificate

URI	/api/v1/configuration/connector/onPremise/systemCertificate
Method	<i>GET</i>
Header	Accept: application/pkix-cert
Response	Binary data of the certificate.
Errors	NOT_FOUND
Roles	Administrator, Display, Support

Response:

- Success: the binary data of the certificate; you can verify the downloaded certificate by storing it in file sys.crt, for instance, and then running

```
keytool -printcert -file sys.crt
```
- Failure: an error in the usual JSON format; the content type of the response is *application/json* in this case.

Errors:

- NOT_FOUND (404): there is no system certificate.

Example

```
curl -i -k -H "Accept: application/pkix-cert" -u <user>:<password> -X GET --output sys.crt https://<host>:<port>/api/v1/configuration/connector/onPremise/systemCertificate
```

Create a Self-Signed System Certificate (Master Only)

URI	/api/v1/configuration/connector/onPremise/systemCertificate
Method	<i>POST</i>
Header	CONTENT_TYPE: application/json

Request	{type, subjectDN, subjectAltNames}
Response	201 on success
Errors	
Roles	Administrator

Request Properties:

- type: the string *selfsigned*
- subjectDN: the subject distinguished name (a string)
- subjectAltNames: subject alternative names (see [Certificate Management for Backend Communication \[page 432\]](#) for details). This property is optional.

The certificate created this way has a validity of 1 year.

Example

```
curl -i -k -H "Accept: application/json" -H "Content-Type: application/json" -u <user>:<password> -X POST --data '{"type":"selfsigned", "subjectDN":"CN=me"}' https://<host>:<port>/api/v1/configuration/connector/onPremise/systemCertificate
```

Create a Certificate Signing Request for a System Certificate (Master Only)

URI	/api/v1/configuration/connector/onPremise/systemCertificate
Method	POST
Header	CONTENT_TYPE: application/json
Request	{type, subjectDN, subjectAltNames}
Response	PEM-encoded certificate request
Errors	
Roles	Administrator

Request Properties:

- type: the string *selfsigned*

- `subjectDN`: the subject distinguished name (a string)
- `subjectAltNames`: subject alternative names (see [Certificate Management for Backend Communication \[page 432\]](#) for details). This property is optional.

Example

```
curl -k -u <user>:<password> -X POST -H "Content-Type: application/json" --data
'{"type":"csr", "subjectDN":"CN=me"}'
https://<host>:<port>/api/v1/configuration/connector/onPremise/
systemCertificate -o csr.pem
```

Upload a Signed Certificate Chain as System Certificate (Master Only)

URI	/api/v1/configuration/connector/ onPremise/systemCertificate
Method	<i>PATCH</i>
Request	multipart/form-data with the following parameter: <code>signedCertificate</code> .
Response	201 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Properties:

- `signedCertificate`: the signed certificate and CA certificate chain (PEM-encoded)

Errors:

- `INVALID_REQUEST` (400): the certificate chain provided does not match the most recent certificate request, or it is not a certificate chain in the correct format (PEM-encoded).

Example

```
curl -i -k -u <user>:<password> -X PATCH -F signedCertificate=@<signedchain.pem>
https://<host>:<port>/api/v1/configuration/connector/onPremise/systemCertificate
```

For test purposes, you can sign the certificate signing request with keytool:

```
keytool -genkeypair -keyalg RSA -keysize 1024 -alias mykey -dname "cn=very
trusted, c=test" -validity 365 -keystore ca.ks -keypass testit -storepass testit
keytool -gencert -rfc -infile csr.pem -outfile signedcsr.pem -alias mykey -
keystore ca.ks -keypass testit -storepass testit
keytool -exportcert -rfc -file ca.pem -alias mykey -keystore ca.ks -keypass
testit -storepass testit
cat signedcsr.pem ca.pem > signedchain.pem
```

Upload a PKCS#12 Certificate as System Certificate (Master Only)

URI	/api/v1/configuration/connector/onPremise/systemCertificate
Method	<i>PUT</i>
Request	Multipart/form-data with the following form parameters: pkcs12, password, keyPassword
Response	204 on success
Errors	INVALID_REQUEST
Roles	Administrator

Request Parameters:

- `pkcs12`: contents of PKCS#12 file
- `password`: password for decrypting PKCS#12 file
- `keyPassword`: optional password for the private key

Errors:

- `INVALID_REQUEST` (400): contents of PKCS#12 or password are invalid

i Note

`keyPassword` is optional. If it is missing, `password` is used to decrypt the pkcs#12 file and the private key.

Example

```
curl -i -k -u <user>:<password> https://<host>:<port>/api/v1/configuration/connector/onPremise/systemCertificate -X PUT -F password=<p21Password> -F pkcs12=@<p12file>
```

Create an own self-signed pkcs#12 certificate for tests with:

```
keytool -genkeypair -alias key -keyalg RSA -keysize 2048 -validity 365 -keystore test20 -storepass test20 -storetype PKCS12 -dname 'CN=test'
```

Delete a System Certificate (Master Only)

URI	/api/v1/configuration/connector/onPremise/systemCertificate
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	NOT_FOUND
Roles	Administrator

Errors:

- NOT_FOUND (404): there is no system certificate.

Example

```
curl -k -H "Accept: application/json" -u <user>:<password> --request DELETE https://localhost:8443/api/v1/configuration/connector/onPremise/systemCertificate
```

1.2.2.5.6 Solution Management Configuration

Manage the Cloud Connector's solution management configuration via API.

Get Solution Management Configuration

URI	/api/v1/configuration/connector/ solutionManagement
Method	GET
Request	
Response	{hostAgentPath, isEnabled, dsrEnabled}
Errors	
Roles	Administrator, Display, Support

Response Properties:

- isEnabled: flag indicating if reporting to solution management is active.
- hostAgentPath: path for host agent executable.
- dsrEnabled: indicating if DSR reporting is active.

Example

```
curl -ik -u <user>:<password> https://<host>:<port>/api/v1/configuration/  
connector/solutionManagement
```

Set Solution Management Configuration and Turn On Reporting

This API turns on the integration with the Solution Manager. The prerequisite is an available Host Agent. You can specify a path to the Host Agent executable, if you don't use the default path.

URI	/api/v1/configuration/connector/ solutionManagement
-----	--

Method	<i>POST</i>
Request	{hostAgentPath, dsrEnabled}
Response	
Errors	
Roles	Administrator, Support

Response Properties:

- hostAgentPath: path for host agent executable (string, optional).
- dsrEnabled: flag indicating if DSR reporting is active (boolean, optional)-

Example

```
curl -ik -u <user>:<password> https://<host>:<port>/api/v1/configuration/
connector/solutionManagement -X POST
```

or, if configuration has to be changed:

```
curl -ik -u <user>:<password> https://<host>:<port>/api/v1/configuration/
connector/solutionManagement -X POST -d "{\"hostAgentPath\":\"new/path/to/
hostAgent\"}" -H "Content-Type:application/json"
```

Turn Off Solution Management Reporting

URI	/api/v1/configuration/connector/ solutionManagement
Method	<i>DELETE</i>
Request	
Response	
Errors	
Roles	Administrator, Support

Example

```
curl -ik -u <user>:<password> https://<host>:<port>/api/v1/configuration/connector/solutionManagement -X DELETE
```

Download Current LMDB XML Report

Generates a zip file containing the registration file for the solution management LMDB (Landscape Management Database).

i Note

Available as of Cloud Connector version 2.12.0.

URI	/api/v1/configuration/connector/solutionManagement/registrationFile
Method	<i>GET</i>
Request	
Response	
Errors	
Roles	Administrator, Support

1.2.2.5.7 Backup

Manage the Cloud Connector's configuration backup via API.

Create Backup Configuration

URI	/api/v1/configuration/backup
Method	<i>POST</i>

Request	{password}
Response	ZIP archive (content type application/zip)
Errors	
Roles	Administrator

Request Properties:

- password: the password used to encrypt sensitive data.

i Note

Only sensitive data in the backup are encrypted with an arbitrary password of your choice. The password is required for the restore operation. The returned ZIP archive itself is not password-protected.

Restore Backup Configuration

URI	/api/v1/configuration/backup
Method	PUT
Request	Multipart/form-data backup file and password as form parameter. Successful request forces restart of the Cloud Connector.
Response	204 - in case of success.
Errors	400 <div style="background-color: #f0f0f0; padding: 5px;"> {type: INVALID_REQUEST, message: <msg>} </div>
Roles	Administrator

i Note

Since this API uses a multipart request, it requires a multipart request header.

1.2.2.5.8 Subaccount

Manage the Cloud Connector's subaccount settings via API.

Operations

<i>Subaccount</i>	Get Subaccounts [page 449] Create Subaccount (Master Only) [page 450] Delete Subaccount (Master Only) [page 451] Edit Subaccount (Master Only) [page 451] Connect/Disconnect Subaccount (Master Only) [page 452] Refresh Subaccount Certificate (Master Only) [page 453] Get Subaccount Configuration [page 454]
<i>Recovery Subaccount</i>	Create Recovery Subaccount (Master Only) [page 454] Delete Recovery Subaccount (Master Only) [page 455] Refresh Certificate of Recovery Subaccount (Master Only) [page 456] Activate/Deactivate Recovery Subaccount (Master Only) [page 457] Takeover (Master Only) [page 457]

Get Subaccounts

URI	/api/v1/configuration/subaccounts
Method	GET
Request	
Response	<pre>[{regionHost, subaccount, locationID}]</pre>
Errors	
Roles	Administrator, Display, Support

Response:

An array of objects with the following properties:

- `regionHost`: region hosts (a string).
- `subaccount`: subaccount name (a string).
- `locationID`: location identifier for the Cloud Connector instance (a string); this property is not available if the default location ID is in use.

Back to [Operations \[page 449\]](#)

Create Subaccount (Master Only)

Creates and connects a subaccount.

URI	/api/v1/configuration/subaccounts
Method	POST
Request	<pre>{regionHost, subaccount, cloudUser, cloudPassword, locationID, displayName, description}</pre>
Response	201, created subaccount entity: <pre>{regionHost, subaccount, locationID, displayName, description, tunnel}</pre>
Errors	INVALID_REQUEST, INVALID_CONFIGURATION
Roles	Administrator

Request Properties:

- `regionHost`: region host name (a string).
- `subaccount`: subaccount technical name (a string).
- `cloudUser`: user for the specified subaccount and region host.
- `cloudPassword`: password for the cloud user.
- `locationID`: location identifier for the Cloud Connector instance (a string; optional).
- `displayName`: display name of the subaccount (a string; optional).
- `description`: subaccount description (a string; optional).

Response Properties:

- `regionHost`: region host name (a string).
- `subaccount`: subaccount technical name (a string).
- `locationID`: location ID (a string); this property is not available if the default location ID is in use.

- `displayName`: display name of the subaccount (a string); this property is not available if there is no specified display name.
- `description`: subaccount description (a string); this property is not available if there is no description.
- `tunnel`: object outlining the current state of the tunnel.

Errors:

- `INVALID_REQUEST` (400): one or more mandatory parameters are missing or invalid.
- `INVALID_CONFIGURATION` (409): the subaccount already exists as a regular subaccount or recovery subaccount.

Back to [Operations \[page 449\]](#)

Delete Subaccount (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	NOT_FOUND, ILLEGAL_STATE
Roles	Administrator

Errors:

- `NOT_FOUND` (404): subaccount does not exist (in the specified region).
- `ILLEGAL_STATE` (409): there is at least one session that has access to the subaccount.

Back to [Operations \[page 449\]](#)

Edit Subaccount (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>
Method	<i>PUT</i>

Request	{ locationID, displayName, description }
Response	{ regionHost, subaccount, locationID, displayName, description, tunnel, recoveryAccountState }
Errors	NOT_FOUND
Roles	Administrator

Request Properties:

- `locationID`: location identifier for the Cloud Connector instance (a string; optional); if this parameter is not supplied the location ID will not change. Revert to the default location ID by supplying the empty string.
- `displayName`: subaccount display name (a string; optional); if this parameter is not supplied the display name will not change. Clear the display name by using an empty string.
- `description`: subaccount description (a string; optional); if this parameter is not supplied the description will not change. Clear the description by using an empty string.

Response Properties:

- `regionHost`: region host name (a string).
- `subaccount`: subaccount technical name (a string).
- `locationID`: location identifier for the Cloud Connector instance (a string); this property is not available if the default location ID is in use.
- `displayName`: display name of the subaccount (a string); this property is not available if there is no specified display name.
- `description`: subaccount description (a string); this property is not available if there is no description.
- `tunnel`: object outlining the current state of the tunnel.
- `recoveryAccountState`: object detailing the current state of the recovery subaccount. This property is supplied only if a recovery subaccount is configured.

Errors

- NOT_FOUND (404): subaccount does not exist (in the specified region).

Back to [Operations \[page 449\]](#)

Connect/Disconnect Subaccount (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/state
-----	---

Method	<i>PUT</i>
Request	{connected}
Response	204 on success
Errors	
Roles	Administrator

Request Properties:

- `connected`: a Boolean value indicating whether the subaccount should be connected (true) or disconnected (false).

Back to [Operations \[page 449\]](#)

Refresh Subaccount Certificate (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/validity
Method	<i>POST</i>
Request	{user, password}
Response	{regionHost, subaccount, locationID, displayName, description, tunnel, recoveryAccountState}
Errors	
Roles	Administrator

Request Properties:

- `user`: user for the specified region host and subaccount.
- `password`: password for the (cloud) user.

Response Properties:

- `regionHost`: region host name (a string).
- `subaccount`: subaccount technical name (a string).
- `locationID`: location identifier for the Cloud Connector instance (a string); this property is not available if the default location ID is in use.

- `displayName`: display name of the subaccount (a string); this property is not available if there is no specified display name.
- `description`: subaccount description (a string); this property is not available if there is no description.
- `tunnel`: object outlining the current state of the tunnel.
- `recoveryAccountState`: object detailing the current state of the recovery subaccount. This property is supplied only if a recovery subaccount is configured.

Back to [Operations \[page 449\]](#)

Get Subaccount Configuration

URI	<code>/api/v1/configuration/subaccounts/<regionHost>/<subaccount></code>
Method	<code>GET</code>
Request	
Response	<pre>{regionHost, subaccount, locationID, displayName, description, tunnel: {state, connections, applicationConnections:[], serviceChannels:[]}}</pre>
Errors	
Roles	Administrator, Display, Support

Response Properties:

- `regionHost`: region host name (a string).
- `subaccount`: subaccount technical name (a string).
- `locationID`: location ID (a string); this property is not available if the default location ID is in use.
- `displayName`: display name of the subaccount (a string); this property is not available if there is no specified display name.
- `description`: description (a string); this property is not available if there is no description.
- `tunnel`: object outlining the current state of the tunnel.

Back to [Operations \[page 449\]](#)

Create Recovery Subaccount (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/recovery
Method	<i>POST</i>
Request	{regionHost, cloudUser, cloudPassword}
Response	201 on success
Errors	INVALID_REQUEST, ILLEGAL_STATE
Roles	Administrator

Request Properties:

- `regionHost`: region host of the recovery subaccount.
- `cloudUser`: user for the specified region host and recovery subaccount.
- `cloudPassword`: password for the cloud user.

Errors:

- `INVALID_REQUEST` (400): invalid or missing request parameters.
- `ILLEGAL_STATE` (409): a recovery subaccount already exists.

i Note

A recovery subaccount cannot be changed. Delete it and create a new one instead.

Back to [Operations \[page 449\]](#)

Delete Recovery Subaccount (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/recovery
Method	<i>DELETE</i>
Request	
Response	
Errors	NOT_FOUND
Roles	Administrator

Errors:

- NOT_FOUND (404): there is no recovery subaccount.

Back to [Operations \[page 449\]](#)

Refresh Certificate of Recovery Subaccount (Master Only)

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/recovery/validity
Method	<i>POST</i>
Request	{user, password}
Response	{regionHost, subaccount, locationID, displayName, description, tunnel, recoveryAccountState}
Errors	INVALID_REQUEST, NOT_FOUND
Roles	Administrator

Request Properties:

- user: user for the specified region host and subaccount.
- password: password for the (cloud) user.

Response Properties:

- regionHost: region host name (a string).
- subaccount: subaccount technical name (a string).
- locationID: location identifier for the Cloud Connector instance (a string); this property is not available if the default location ID is in use.
- displayName: display name of the subaccount (a string); this property is not available if there is no specified display name.
- description: subaccount description (a string); this property is not available if there is no description.
- tunnel: object outlining the current state of the tunnel.
- recoveryAccountState: object detailing the current state of the recovery subaccount. This property is supplied only if a recovery subaccount is configured.

Errors

- INVALID_REQUEST (400): user or password are missing.

- NOT_FOUND (404): there is no recovery subaccount.

Back to [Operations \[page 449\]](#)

Activate/Deactivate Recovery Subaccount (Master Only)

i Note

Available as of Cloud Connector 2.13.1.

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/recovery/state
Method	<i>PUT</i>
Request	{active}
Response	204 on success
Errors	NOT_FOUND
Roles	Administrator

Request Properties:

- active: Boolean value indicating whether the recovery subaccount should be active (true) or inactive (false).

Errors:

- NOT_FOUND (404): there is no recovery subaccount.

Back to [Operations \[page 449\]](#)

Takeover (Master Only)

⚠ Caution

When performing this operation, the recovery subaccount *permanently takes over* from the original subaccount, and the *original subaccount is deleted*. The recovery subaccount must be active for takeover to succeed, see [Activate/Deactivate Recovery Subaccount \(Master Only\) \[page 457\]](#).

i Note

Available as of Cloud Connector 2.13.1.

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/recovery/ takeover
-----	---

Method	<i>POST</i>
--------	-------------

Request	
---------	--

Response	204 on success
----------	----------------

Errors	NOT_FOUND, ILLEGAL_STATE
--------	--------------------------

Roles	Administrator
-------	---------------

Errors:

- NOT_FOUND (404): there is no recovery subaccount.
- ILLEGAL_STATE (409): recovery subaccount is not active.

Back to [Operations \[page 449\]](#)

1.2.2.5.9 System Mappings

Manage the Cloud Connector's system mappings via API.

Get All System Mappings

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/systemMappings
-----	--

Method	<i>GET</i>
--------	------------

Request	
---------	--

Response	<pre>[{virtualHost, virtualPort, localhost, localPort, protocol, backendType, authenticationMode, hostInHeader, description, ...}]</pre>
Errors	
Roles	Administrator, Display, Support

Response:

An array of objects with the following properties:

- **virtualHost**: Virtual host used on the cloud side (a string)
- **virtualPort**: Virtual port used on the cloud side (a string)
- **localhost**: Host on the on-premise side (a string)
- **localPort**: Port on the on-premise side (a string)
- **protocol**: Protocol used when sending requests and receiving responses (a string)
- **backendType**: Type of the backend (a string)
- **authenticationMode**: Authentication mode used on the backend side (a string).
- **hostInHeader**: Policy for setting the host in the response header. Available for HTTP(S) protocols only.
- **totalResourcesCount**: the total number of resources
- **enabledResourcesCount**: the number of enabled resources
- **description**: Description for the system mapping (a string).
- **sncPartnerName**: SNC name of an ABAP Server, required for RFCS communication only.
- **sapRouter**: SAP router route, required only if an SAP router is used.
- **allowedClients**: Array of strings, describing the SAP clients allowing to execute the calls in this system. Valid clients are 3 letters long. If no clients are defined here, there is no restriction – every client is allowed. Only applicable for RFC-based communication.
- **blacklistedClientUsers**: Array of {client, user}, describing users that are not allowed to execute the call, even if the client is listed under allowed clients. Only applicable for RFC-based communication.

Get System Mapping

URI	<code>/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/ systemMappings/ <virtualHost>:<virtualPort></code>
Method	<code>GET</code>
Request	

Response	<pre>{virtualHost, virtualPort, localHost, localPort, protocol, backendType, authenticationMode, hostInHeader, description, ...}</pre>
Errors	
Roles	Administrator, Display, Support

Response:

An array of objects with the following properties:

- **virtualHost**: Virtual host used on the cloud side (a string)
- **virtualPort**: Virtual port used on the cloud side (a string)
- **localHost**: Host on the on-premise side (a string)
- **localPort**: Port on the on-premise side (a string)
- **protocol**: Protocol used when sending requests and receiving responses (a string)
- **backendType**: Type of the backend (a string)
- **authenticationMode**: Authentication mode used on the backend side (a string).
- **hostInHeader**: Policy for setting the host in the response header (a string). Available for HTTP(S) protocols only.
- **totalResourcesCount**: the total number of resources
- **enabledResourcesCount**: the number of enabled resources
- **description**: Description for the system mapping (a string).
- **sncPartnerName**: SNC name of an ABAP Server, required for RFCS communication only.
- **sapRouter**: SAP router route, required only if an SAP router is used.
- **allowedClients**: Array of strings, describing the SAP clients allowing to execute the calls in this system. Valid clients are 3 letters long. If no clients are defined here, there is no restriction – every client is allowed. Only applicable for RFC-based communication.
- **blacklistedClientUsers**: Array of {client, user}, describing users that are not allowed to execute the call, even if the client is listed under allowed clients. Only applicable for RFC-based communication.

Create System Mapping (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/systemMappings
Method	<i>POST</i>

Request	<pre>{virtualHost, virtualPort, localHost, localPort, protocol, backendType, authenticationMode, hostInHeader, description, ...}</pre>
Response	201 on success
Errors	
Roles	Administrator

Request Properties:

- **virtualHost**: Virtual host used on the cloud side (a string)
- **virtualPort**: Virtual port used on the cloud side (a string)
- **localHost**: Host on the on-premise side (a string)
- **localPort**: Port on the on-premise side (a string)
- **protocol**: Protocol used when sending requests and receiving responses, which must be one of the following strings: **HTTP**, **HTTPS**, **RFC**, **RFCS**, **LDAP**, **LDAPS**, **TCP**, **TCPS**.
- **backendType**: Type of the backend system. Valid values are **abapSys**, **netweaverCE**, **netweaverGW**, **applServerJava**, **BC**, **PI**, **hana**, **otherSAPsys**, **nonSAPsys**.
- **authenticationMode**: Authentication mode to be used on the backend side, which must be one of the following strings: **NONE**, **X509_GENERAL**, **X509_RESTRICTED**, **KERBEROS**.
- **hostInHeader**: Policy for setting the host in the response header. This property is applicable to **HTTP(S)** protocols only, and it is **optional**. If set, it must be one of the following strings: **internal**, **virtual**. The default is **virtual**. You may also use all capital letters, i.e. **INTERNAL** and **VIRTUAL**.
- **description**: Description for the system mapping (string, optional). The default is no description, i.e. the empty string.
- **sncPartnerName**: SNC name of an ABAP Server, required for **RFCS** communication only.
- **sapRouter**: SAP router route, required only if an SAP router is used.
- **allowedClients**: Array of strings, describing the SAP clients allowing to execute the calls in this system. Valid clients are 3 letters long. If no clients are defined here, there is no restriction – every client is allowed. Only applicable for RFC-based communication.
- **blacklistedClientUsers**: Array of {client, user}, describing users that are not allowed to execute the call, even if the client is listed under allowed clients. Only applicable for RFC-based communication.

Delete System Mapping (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/systemMappings
Method	DELETE

Request	
Response	204 on success
Errors	
Roles	Administrator

Delete All System Mappings (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/systemMappings
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	
Roles	Administrator

Edit System Mapping

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/ systemMappings/virtualHost:virtualPort
Method	<i>PUT</i>
Request	{virtualHost, virtualPort, localhost, localPort, protocol, backendType, authenticationMode, hostInHeader, description, ...}
Response	
Errors	
Roles	Administrator

Request Properties:

- `virtualHost`: Virtual host used on the cloud side (a string)
- `virtualPort`: Virtual port used on the cloud side (a string)

These two properties are technically mandatory as they identify the system mapping to be edited or changed. They cannot be changed. However, they are also part of the URI path. If omitted from the request, they will effectively be copied from the URI path, and no error is thrown. This policy of leniency deviates from strict REST conventions, but was adopted to increase user-friendliness by avoiding unnecessary error situations.

All other properties are optional. Add those properties that you want to change.

- `localHost`: Host on the on-premise side (a string)
- `localPort`: Port on the on-premise side (a string)
- `protocol`: Protocol used when sending requests and receiving responses, which must be one of the following strings: `HTTP`, `HTTPS`, `RFC`, `RFCS`, `LDAP`, `LDAPS`, `TCP`, `TCPS`.
- `backendType`: Type of the backend system. Valid values are `abapSys`, `netweaverCE`, `netweaverGW`, `applServerJava`, `BC`, `PI`, `hana`, `otherSAPsys`, `nonSAPsys`.
- `authenticationMode`: Authentication mode to be used on the backend side, which must be one of the following strings: `NONE`, `X509_GENERAL`, `X509_RESTRICTED`, `KERBEROS`.
- `hostInHeader`: Policy for setting the host in the response header; this property is **optional**. If set, it must be one of the following strings: `internal`, `virtual`. The default is `virtual`. You may also use all capital letters, i.e. `INTERNAL` and `VIRTUAL`.
- `description`: Description for the system mapping (string, optional). The default is no description, i.e. the empty string.
- `sncPartnerName`: SNC name of an ABAP Server, only set for RFCS communication.
- `sapRouter`: SAP router route, only set if an SAP router is used.
- `allowedClients`: Array of strings, describing the SAP clients allowing to execute the calls in this system. Valid clients are 3 letters long. If no clients are defined here, there is no restriction – every client is allowed. Only applicable for RFC-based communication.
- `blacklistedClientUsers`: Array of `{client, user}`, describing users, that are not allowed to execute the call, even if the client is listed under allowed clients. Only applicable for RFC-based communication.

1.2.2.5.10 System Mapping Resources

Manage the Cloud Connector's system mapping resources via API.

Get System Mapping Resources

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/systemMappings/virtualHost:virtualPort/resources
Method	<i>GET</i>
Request	
Response	[{id, enabled, exactMatchOnly, websocketUpgradeAllowed, description}]
Errors	
Roles	Administrator, Display, Support

Response:

An array of objects, each representing a resource through the following properties:

- **id**: The resource itself, which, depending on the owning system mapping, is either a URL path (or the leading section of it), or a RFC function name (prefix)
- **enabled**: Boolean flag indicating whether the resource is enabled.
- **exactMatchOnly**: Boolean flag determining whether access is granted only if the requested resource is an exact match.
- **websocketUpgradeAllowed**: Boolean flag indicating whether websocket upgrade is allowed; this property is of relevance only if the owning system mapping employs protocol HTTP or HTTPS.
- **description**: Description (a string); this property is not available unless explicitly set.

Get System Mapping Resource

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/systemMappings/virtualHost:virtualPort/resources/<encodedResourceId>
Method	<i>GET</i>
Request	
Response	{id, enabled, exactMatchOnly, websocketUpgradeAllowed, description}
Errors	

Roles	Administrator, Display, Support
-------	---------------------------------

Response Properties:

- `id`: The resource itself, which, depending on the owning system mapping, is either a URL path (or the leading section of it), or a RFC function name (prefix)
- `enabled`: Boolean flag indicating whether the resource is enabled.
- `exactMatchOnly`: Boolean flag determining whether access is granted only if the requested resource is an exact match.
- `websocketUpgradeAllowed`: Boolean flag indicating whether websocket upgrade is allowed; this property is of relevance only if the owning system mapping employs protocol HTTP or HTTPS.
- `description`: Description (a string); this property is not available unless explicitly set.

Create System Mapping Resource

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/ systemMappings/virtualHost:virtualPort/ resources
Method	<i>POST</i>
Request	{ <code>id</code> , <code>enabled</code> , <code>exactMatchOnly</code> , <code>websocketUpgradeAllowed</code> , <code>description</code> }
Response	201 on success
Errors	
Roles	Administrator

Request Properties:

- `id`: The resource itself, which, depending on the owning system mapping, is either a URL path (or the leading section of it), or a RFC function name (prefix).
- `enabled`: Boolean flag indicating whether the resource is enabled (**optional**). The default value is `false`.
- `exactMatchOnly`: Boolean flag determining whether access is granted only if the requested resource is an exact match (**optional**). The default value is `false`.
- `websocketUpgradeAllowed`: Boolean flag indicating whether websocket upgrade is allowed (**optional**). The default value is `false`. This property is recognized only if the owning system mapping employs protocol HTTP or HTTPS.
- `description`: Description (a string, **optional**)

→ Tip

Encoded Resource ID

URI paths may contain the resource ID in order to identify the resource to be edited or deleted. A resource ID, however, may contain characters such as the forward slash that collide with the path separator of the URI and hence require an escape mechanism. We adopted the following simple escape or encoding method for a resource ID:

1. Replace all occurrences of character '+' with '+2B'.
2. Replace all occurrences of character '-' with '+2D'.
3. Replace all occurrences of character '/' with '-'.

Edit System Mapping Resource

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/systemMappings/virtualHost:virtualPort/resources/<encodedResourceId>
Method	<i>PUT</i>
Request	{enabled, exactMatchOnly, websocketUpgradeAllowed, description}
Response	204 on success
Errors	
Roles	Administrator

Request Properties:

- **enabled**: Boolean flag indicating whether the resource is enabled.
- **exactMatchOnly**: Boolean flag determining whether access is granted only if the requested resource is an exact match.
- **websocketUpgradeAllowed**: Boolean flag indicating whether websocket upgrade is allowed; this property is of relevance only if the owning system mapping employs protocol HTTP or HTTPS.
- **description**: Description (a string); this property is not available unless explicitly set.

Delete System Mapping Resource

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/ systemMappings/virtualHost:virtualPort/ resources/<encodedResourceId>
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	NOT_FOUND
Roles	Administrator

Errors:

NOT_FOUND (404): Resource was not found

Delete All System Mapping Resources

URI	/api/v1/configuration/subaccounts/ <region>/<subaccount>/systemMappings/ <virtualHost>:<virtualPort>/resources
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	
Roles	Administrator

1.2.2.5.11 Domain Mappings

Manage the Cloud Connector's configuration for domain mappings via API.

Get Domain Mappings

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/domainMappings
Method	<i>GET</i>
Request	
Response	[{virtualDomain, internalDomain}]
Errors	
Roles	Administrator, Display, Support

Response:

An array of objects, each representing a domain mapping through the following properties:

- `virtualDomain`: Domain used on the cloud side
- `internalDomain`: Domain used on the on-premise side

Create Domain Mappings (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/domainMappings
Method	<i>POST</i>
Request	{virtualDomain, internalDomain}
Response	201
Errors	
Roles	Administrator

Edit Domain Mappings (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/ domainMappings/<internalDomain>
Method	<i>PUT</i>
Request	{virtualDomain, internalDomain}
Response	204 on success
Errors	NOT_FOUND
Roles	Administrator

Request:

- virtualDomain: New virtual domain
- internalDomain: New internal domain

Errors:

- NOT_FOUND (404): Domain mapping does not exist.

i Note

The internal domain in the URI path (i.e., <internalDomain>) is the current internal domain of the domain mapping that is to be edited. It may differ from the new internal domain set in the request.

Delete Domain Mappings (Master Only)

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/ domainMappings/<internalDomain>
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	NOT_FOUND
Roles	Administrator

Errors:

- NOT_FOUND (404): Domain mapping does not exist.

Delete All Domain Mappings (Master Only)

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/domainMappings
Method	<i>DELETE</i>
Request	
Response	204 on success
Errors	
Roles	Administrator

1.2.2.5.12 Subaccount Service Channels

Manage Cloud Connector service channels via API.

Get Service Channels

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/channels
Method	<i>GET</i>
Request	
Response	<pre>[{typeDesc, details, port, enabled, connected, connectionsCount, availableConnectionsCount, connectedSinceTimeStamp}]</pre>
Errors	
Roles	Administrator, Display, Support

Response:

An array of objects, each of which represents a service channel through the following properties:

- `typeDesc`: an object specifying the service channel type through the properties `typeKey` and `typeName`.
- `details`

- port
- enabled
- connected
- connectionsCount
- availableConnectionsCount
- connectedSinceTimeStamp

Create Service Channel

URI	/api/v1/configuration/subaccounts/<regionHost>/<subaccount>/channels
Method	<i>POST</i>
Request	{typeKey, details, serviceNumber, connectionCount}
Response	201, Location header for new entity.
Errors	400 - if a parameter is invalid or account or service channel are invalid: {type: INVALID_REQUEST, message: <msg>}
Roles	Administrator

Request Properties:

- typeKey: type of service channel. Valid values are HANA_DB, HCPVM, RFC.
- details:
 - HANA instance name for HANA_DB
 - VM name for HCPVM
 - S/4HANA Cloud tenant host for RFC
- serviceNumber: service number, which is mapped to a port according to the type of service channel.
- connectionCount: number of connections for the channel.

Delete Service Channel

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/channels/<id>
Method	<i>DELETE</i>
Request	
Response	{ }
Errors	
Roles	Administrator

Edit Service Channel

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/channels/<id>
Method	<i>PUT</i>
Request	{typeKey, details, serviceNumber, connect ionCount}
Response	204
Errors	400 - if account or service channel Id are invalid: {type: INVALID_REQUEST, message: <msg>}
Roles	Administrator

Enable/Disable Service Channel

URI	/api/v1/configuration/subaccounts/ <regionHost>/<subaccount>/channels/<id>/ state
Method	<i>PUT</i>

Request	{enabled:boolean}
Response	
Errors	400 - if account or service channel Id are invalid: {type: "INVALID_REQUEST", message: <msg>}
	500 {type:"RUNTIME_FAILURE","message":"Service channel could not be opened"}
Roles	Administrator

1.2.2.5.13 Examples

Find examples on how to use the Cloud Connector's configuration REST APIs.

Concept

The sample code in this section ([download zip file](#)) demonstrates how to use the REST APIs provided by Cloud Connector to perform various configuration tasks.

Starting with a freshly installed Cloud Connector, the samples include initial configuration of the Cloud Connector instance, connectivity setup, high availability configuration, and common tasks like backup/restore operations, as well as integration with solution management.

The examples are implemented in *Kotlin*, a simple Java VM-based language. However, even if you are using a different language, they still show the basic use of the APIs and their parameters for specific configuration purposes.

If you are not familiar with Kotlin, find a brief introduction and some typical statements below.

[REST API Parameters \[page 474\]](#)

[REST API Invocation \[page 474\]](#)

[How To Use the Examples \[page 475\]](#)

REST API Parameters

In almost all requests and responses, structures are encoded in JSON format. To describe the parameter details, we use Kotlin data classes.

This class represents a structure that you can use as value in a request or response:

```
data class OnlyPropertiesNamesAreRelevant(  
    val user: String,  
    val password: String  
)
```

The JSON representation for that class is

```
{"user":<userValue>, "password":<passwordValue>}
```

Encoded in Kotlin this string looks like

```
""" {"user": "$userValue", "password": "$passwordValue"} """
```

or as an object:

```
val credentials = OnlyPropertiesNamesAreRelevant(userValue, passwordValue)
```

Back to [Concept \[page 473\]](#)

REST API Invocation

```
(a)   Fuel.put(url)  
(b)     .header("Connection", "close")  
(c)       .authentication().basic(user, password)  
(d)       .jsonBody(credentials)  
(e)       .responseObject<OnlyPropertiesNamesAreRelevant> { _, _, result ->  
        when (result) {  
            is Result.Failure -> processRequestError(result.error)  
            is Result.Success -> println("returned: ${result.get()}")  
        }  
    }  
.join()
```

- (a) - *Fuel* is an HTTP framework used in the examples. Important is the verb after *Fuel* - it is the REST API method.
- (b) - Adds a request header `Connection: close`, which forces a connection close after request. In the examples, this header is defined on *FuelManager* for all calls.
- (c) - Basic authentication is used for the call with user and password.

- (d) - HTTP requests with parameters require mostly JSON. The `jsonBody`-method adds the header `Content-Type: application/json` and serializes the provided object credentials to JSON. Requests without body like `DELETE` or `GET` omit body methods.
- (e) - The `responseObject<ClassType>`-method adds the header `Accept: application/json` and de-serializes the response to the specified class type. Some APIs do not have any response or non-JSON response. In such cases, the method `response` is used instead of `responseObject<ClassType>`.
- (f) - The way, how Kotlin decides between success (2xx) and failed responses. For details on the possible response status, see [Configuration REST APIs \[page 404\]](#).

Back to [Concept \[page 473\]](#)

How To Use the Examples

Some common details, used by different examples, were extracted to the `scenario.json` configuration file. This lets you use meaningful names like `config.master!.user` in the examples. The file is loaded by

```
val config = loadScenarioConfiguration()
```

Each example also invokes

```
disableTrustChecks()
```

This method disables all SSL-related checks.

Caution

`disableTrustChecks()` is only used for test purposes. *Do not use it in a productive environment.*

Both methods, as well as some common REST API parameter structures (data classes) are defined in the file [Scenario Configuration \[page 477\]](#). This is the only help class under `sources/`.

When using the examples, start with [Initial Configuration \[page 478\]](#).

Prerequisite is a freshly installed Cloud Connector.

After a mandatory password change and defining the high availability role of the Cloud Connector instance (master or shadow), the example demonstrates how to provide a description for the instance and how to set up the UI and system certificates.

Once the initial configuration is done, you can optionally proceed with these steps:

- Connect to your subaccount on BTP ([Subaccount Configuration \[page 480\]](#))
- Create or restore a configuration backup ([Backup And Restore Configuration \[page 487\]](#))
- Configure and connect high availability instances ([High Availability Settings \[page 485\]](#))
- Integrate the Cloud Connector with the solution management infrastructure ([Solution Management Integration \[page 488\]](#))

[Back to Concept \[page 473\]](#)

Related Information

[scenario.json \[page 476\]](#)

[Source Files \[page 476\]](#)

1.2.2.5.13.1 scenario.json

« Sample Code

```
{  
    "subaccount": {  
        "regionHost": "cf.eu10.hana.ondemand.com",  
        "subaccount": "11aabbcc-7821-448b-9ecf-a7d986effa7c",  
        "user": "xxx",  
        "password": "xxx"  
    },  
    "master": {  
        "url": "https://localhost:8443",  
        "user": "Administrator",  
        "password": "test"  
    },  
    "shadow": {  
        "url": "https://localhost:8444",  
        "user": "Administrator",  
        "password": "test"  
    }  
}
```

1.2.2.5.13.2 Source Files

[Scenario Configuration \[page 477\]](#)

[Initial Configuration \[page 478\]](#)

[Subaccount Configuration \[page 480\]](#)

[High Availability Settings \[page 485\]](#)

[Backup And Restore Configuration \[page 487\]](#)

[Solution Management Integration \[page 488\]](#)

1.2.2.5.13.2.1 Scenario Configuration

↳ Sample Code

```
package com.sap.scc.examples
import com.github.kittinunf.fuel.core.FuelError
import com.google.gson.Gson
import java.io.File
import java.security.SecureRandom
import java.security.cert.X509Certificate
import javax.net.ssl.*
data class CloudConnector(
    val url: String,
    var user: String,
    var password: String
)
fun disableTrustChecks() {
    try {
        HttpsURLConnection.setDefaultHostnameVerifier { hostname: String,
session: SSLSocketSession -> true }
        val context: SSLContext = SSLContext.getInstance("TLS")
        val trustAll: X509TrustManager = object : X509TrustManager {
            override fun checkClientTrusted(chain: Array<X509Certificate>,
authType: String) {}
            override fun checkServerTrusted(chain: Array<X509Certificate>,
authType: String) {}
            override fun getAcceptedIssuers(): Array<X509Certificate> {
                return arrayOf()
            }
        }
        context.init(null, arrayOf(trustAll), SecureRandom())
        HttpsURLConnection.setDefaultSSLSocketFactory(context.socketFactory)
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
class ScenarioConfiguration {
    var subaccount: SubaccountParameters? = null
    var master: CloudConnector? = null
    var shadow: CloudConnector? = null
}
data class SubaccountParameters(
    val regionHost: String,
    val subaccount: String,
    val user: String,
    val password: String,
    var locationId: String? = null
)
data class SccCertificate(
    var subjectDN: String? = null,
    var issuer: String? = null,
    var notAfter: String? = null,
    var notBefore: String? = null,
    var subjectAltNames: List<SubjectAltName>? = null
)
data class SubjectAltName(
    var type: String? = null,
    var value: String? = null
)
internal fun loadScenarioConfiguration(): ScenarioConfiguration {
    println("scenario.json will be loaded from ${File("scenario.json").absolutePath}")
    return Gson().fromJson(File("scenario.json").readText(),
    ScenarioConfiguration::class.java)
}
```

```

internal fun processRequestError(error: FuelError) {
    println("failed with ${error.message} ${String(error.errorData)}")
    throw RuntimeException("Stop here. ")
}

```

1.2.2.5.13.2.2 Initial Configuration

↳ Sample Code

```

package com.sap.scc.examples
//import com.sap.scc.examples.SccCertificate
import com.github.kittinunf.fuel
import com.github.kittinunf.fuel.core.BlobDataPart
import com.github.kittinunf.fuel.core.FuelManager
import com.github.kittinunf.fuel.core.Method
import com.github.kittinunf.fuel.core.extensions.authentication
import com.github.kittinunf.fuel.gson.responseObject
import com.github.kittinunf.result.Result
import java.io.ByteArrayInputStream
import java.io.File
/*
    This example shows how to use REST APIs to perform the initial configuration
    of a master instance
    after installing and starting the Cloud Connector.
    As a prerequisite you need to install and start the Cloud Connector.
    The example begins with changing the initial password, setting the instance
    to the master role, edit the description,
    and upload UI and system certificates.
    For the certificates used by Cloud Connector in order to access the UI and
    for the system certificate used to access backend systems,
    we simply upload the already available PKCS#12 certificates uiCert.p12 and
    systemCert.p12 encrypted with the password "test1234".
    Cloud Connector also provides other options for certificate management,
    please take a look at the documentation.
    The configuration details for master and shadow instances can be found in
    scenario.json.
*/
fun main() {
    //Cloud Connector distribution generates only an untrusted self-signed
    certificate.
    //So for this demonstration use case we need to deactivate all trust
    checks.
    disableTrustChecks()
    //Use 'Connection: close' header, to make stateless communication more
    efficient
    FuelManager.instance.baseHeaders = mapOf("Connection" to "close")
    //Add output of cURL commands for revision
    //FuelManager.instance.addRequestInterceptor(LogRequestAsCurlInterceptor)
    //Load configuration from property file
    val config = loadScenarioConfiguration()
    //Change the initial password
    Fuel.put("${config.master!!.url}/api/v1/configuration/connector/
authentication/basic")
        .authentication().basic(config.master!!.user, "manage")
        .body("""{"oldPassword":"manage", "newPassword":${
config.master!!.password}}""")
        .response { _, _, result ->
            when (result) {
                is Result.Failure -> processRequestError(result.error)
            }
        }
}

```

```

        is Result.Success -> println("Password successfully set to ${config.master!!.password}")
    }
}
.join()
//Set the High-Availability Role to master, use "shadow" if you want to
set it to shadow
Fuel.put("${config.master!!.url}/api/v1/configuration/connector/haRole")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .body("master")
    .response { _, _ , result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> println("high-availability role
successfully set to 'master'")
        }
    }
    .join()
//Edit Common Description
Fuel.put("${config.master!!.url}/api/v1/configuration/connector")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .body("""{"description":"<description of Cloud Connector
instance>}""")
    .responseObject<SccDescriptionResponse> { _, _ , result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> println(result.get())
        }
    }
    .join()
//Upload a PKCS#12 Certificate as UI Certificate
val uiCertificateFormData = listOf("password" to "test1234",
"keyPassword" to "test1234")
Fuel.upload(
    "${config.master!!.url}/api/v1/configuration/connector/ui/
uiCertificate",
    Method.PUT,
    uiCertificateFormData
)
    .add(BlobDataPart(ByteArrayInputStream(File("uiCert.p12").readBytes()))
, name = "pkcs12")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .response { _, _ , result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> println("PKCS#12 Certificate
'uiCert.p12' successfully uploaded")
        }
    }
    .join()
//Upload a PKCS#12 Certificate as System Certificate
val systemCertificateFormData = listOf("password" to "test1234",
"keyPassword" to "test1234")
Fuel.upload(
    "${config.master!!.url}/api/v1/configuration/connector/onPremise/
systemCertificate",
    Method.PUT,
    systemCertificateFormData
)
    .add(BlobDataPart(ByteArrayInputStream(File("systemCert.p12").readByte
s())), name = "pkcs12")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .response { _, _ , result ->
        when (result) {

```

```

        is Result.Failure -> processRequestError(result.error)
        is Result.Success -> println("PKCS#12 Certificate
'systemCert.p12' successfully uploaded")
    }
    .join()
}
//Data structures used by REST calls in this scenario
data class SccDescriptionResponse(
    var role: String,
    var description: String
)

```

1.2.2.5.13.2.3 Subaccount Configuration

↳ Sample Code

```

package com.sap.scc.examples
import com.github.kittinunf.fuel.Fuel
import com.github.kittinunf.fuel.core.FuelManager
import com.github.kittinunf.fuel.core.extensions.authentication
import com.github.kittinunf.fuel.gson.jsonBody
import com.github.kittinunf.fuel.gson.responseObject
import com.github.kittinunf.result.Result
/*
This example shows how to use REST APIs to configure and connect a
subaccount in cloud connector.
The example begins with (1) connecting of the subaccount, then we create a
system (2) for an HTTP service
and (3) for an RFC service.

The configuration details for master and shadow instances can be found in
scenario.json.
*/
fun main() {
    //Cloud Connector distribution generates only an untrusted self-signed
    certificate.
    //So for this demonstration use case we need to deactivate all trust
    checks.
    disableTrustChecks()
    //Use 'Connection: close' header, to make stateless communication more
    efficient
    FuelManager.instance.baseHeaders = mapOf("Connection" to "close")
    //Add output of CURL commands for revision
    //FuelManager.instance.addRequestInterceptor(LogRequestAsCurlInterceptor)
    //1.1. Create and connect subaccount
    //Load configuration from property file
    val config = loadScenarioConfiguration()
    //Some cloud regions require 2-Factor-Authentication
    println("Enter MFA (aka 2FA) token, if required: ")
    var token = readLine() ?: ""
    //Parameters required to establish the connection to the subaccount (aka
    the secure tunnel)
    var subaccountCreateData = SubaccountConfiguration(
        config.subaccount!!!.regionHost, config.subaccount!!!.subaccount,
        config.subaccount!!!.user, "" + config.subaccount!!!.password + token,
        locationId = config.subaccount!!!.locationId
    )
    //Optional: Initialize the map to work with generated _links
}

```

```

//If you don't like to use _links, you can easily compute the entity links
var subaccountLinks: Map<String, HalLink> = mapOf()
Fuel.post("${config.master.url}/api/v1/configuration/subaccounts")
    .authentication().basic(config.master!.user,
config.master!.password)
    .jsonBody(subaccountCreateData)
    .responseObject<SubaccountInfo> { _, response, result ->
        when (result) {
            is Result.Success -> {
                val subaccountLocation =
response.header("Location").first()
                println("1.1. subaccount was created under:
$subaccountLocation")
                println("subaccount: ${result.get()}")
                //GET details as SubaccountInfo every time possible by
invoking
                //https://<SCC>/api/v1/configuration/subaccounts/
<regionHost>/<subaccountId>
                subaccountLinks = result.get()._links
            }
            is Result.Failure -> processRequestError(result.error)
        }
    }
    .join()
}

//1.2. From time to time it is necessary to extend the validity of the
subaccount - refresh subaccount
//Again some cloud landscapes require 2-Factor-Authentication
println("Enter MFA (aka 2FA) token for refresh, if required: ")
token = readLine() ?: ""
Fuel.post(subaccountLinks["validity"]!!.href)
    .authentication().basic(config.master!.user,
config.master!.password)
    .jsonBody(SubaccountRefreshCred(config.subaccount!.user, "" +
config.subaccount!.password + token))
    .responseObject<SubaccountInfo> { _, _, result ->
        when (result) {
            is Result.Success -> println("1.2. validity of the subaccount
was extended")
            is Result.Failure -> processRequestError(result.error)
        }
    }
    .join()

//2.1. Create a system mapping (aka access control) for an HTTP service
val httpSystem = SystemMapping(
    "virtual.host", "vport", "local", "lport",
CommunicationProtocol.HTTPS,
    BackendType.abapSys, hostInHeader = HostInHeader.INTERNAL
)
var httpSystemLinks: Map<String, HalLink> = mapOf()
Fuel.post(subaccountLinks["systemMappings"]!!.href)
    .authentication().basic(config.master!.user,
config.master!.password)
    .jsonBody(httpSystem)
    .responseString { _, response, result ->
        when (result) {
            is Result.Success -> {
                val httpSystemMappingLocation =
response.header("Location").first()
                println("2.1. system mapping was created under:
$httpSystemMappingLocation")
                //GET the details about the system mapping by invoking
                //https://<SCC>/api/v1/configuration/subaccounts/
<regionHost>/<subaccountId>/systemMappings/<vhost>:<vport>
                Fuel.get(httpSystemMappingLocation)
                    .authentication().basic(config.master!.user,
config.master!.password)
                    .responseObject<SystemMapping> { _, _, result ->
                        when (result) {

```

```

        is Result.Success -> {
            println("system mapping: ${result.get()}")
            httpSystemLinks = result.get()._links
        }
        is Result.Failure ->
    processRequestError(result.error)
}
}
.join()
}
is Result.Failure -> processRequestError(result.error)
}
}
.join()
//2.2 Add an allowed resource to the system mapping. Since this system
mapping points to an HTTP service, use HTTP resource parameters
Fuel.post(httpSystemLinks["resources"]!!!.href)
.authentication().basic(config.master!!!.user,
config.master!!!.password)
.jsonBody(HttpResource("/", exactMatchOnly = false))
.responseString { _, response, result ->
when (result) {
    is Result.Success -> {
        val httpResourceLocation =
response.header("Location").first()
        println("2.2. http resource was created under:
$httpResourceLocation")
        //GET the details about the resource by invoking
        //https://<SCC>/api/v1/configuration/subaccounts/
<regionHost>/<subaccountId>/systemMappings/<vhost>:<vport>/<encoded-id>
        //<encoded-id> -> replace '/' with '-'
        Fuel.get(httpResourceLocation)
        .authentication().basic(config.master!!!.user,
config.master!!!.password)
        .responseObject<HttpResource> { _, _, result ->
            when (result) {
                is Result.Success -> println("http resource: $"
{result.get()})
                is Result.Failure ->
            processRequestError(result.error)
}
}
.join()
}
is Result.Failure -> processRequestError(result.error)
}
}
.join()
//3.1 Create a system mapping for an RFC service
var rfcSystemLinks: Map<String, HalLink> = mapOf()
Fuel.post(subaccountLinks["systemMappings"]!!!.href)
.authentication().basic(config.master!!!.user,
config.master!!!.password)
.jsonBody(
    SystemMapping(
        "virtual.host",
        "rfcport",
        "local",
        "rfcport",
        CommunicationProtocol.RFCS,
        BackendType.abapSys
    )
)
.responseString { _, response, result ->
when (result) {
    is Result.Success -> {
        val rfcSystemMappingLocation =
response.header("Location").first()

```

```

        println("3.1. system mapping was created under:
$rfcSystemMappingLocation")
            //GET the details about the system mapping by invoking
            //https://<SCC>/api/v1/configuration/subaccounts/
<regionHost>/<subaccountId>/systemMappings/<vhost>:<vport>
Fuel.get(rfcSystemMappingLocation)
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .responseObject<SystemMapping> { _, _, result ->
    when (result) {
        is Result.Success -> {
            println("system mapping: ${result.get()}")
            rfcSystemLinks = result.get()._links
        }
        is Result.Failure ->
processRequestError(result.error)
    }
}
    .join()
}
is Result.Failure -> processRequestError(result.error)
}
}
.join()
//3.2 Now add the function module RFC_SYSTEM_INFO as an allowed resource
to the system mapping
val rfcResource = RfcResource("RFC_SYSTEM_INFO")
Fuel.post(rfcSystemLinks["resources"]!!._href)
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .jsonBody(rfcResource)
    .responseString { _, response, result ->
    when (result) {
        is Result.Success -> {
            val resourceLocation = response.header("Location").first()
            println("3.2. rfc resource was created under:
$resourceLocation")
                //GET the details about the resource by invoking
                //https://<SCC>/api/v1/configuration/subaccounts/
<regionHost>/<subaccountId>/systemMappings/<vhost>:<vport>/<encoded-id>
                //<encoded-id> -> replace '/' with '-'
                Fuel.get(resourceLocation)
                    .authentication().basic(config.master!!.user,
config.master!!.password)
                    .responseObject<RfcResource> { _, _, result ->
                    when (result) {
                        is Result.Success -> println("rfc resource: ${result.get()}")
                        is Result.Failure ->
processRequestError(result.error)
                    }
}
                .join()
}
        is Result.Failure -> processRequestError(result.error)
    }
}
    .join()
}
//Data structures used by REST calls in this scenario
//Nullable properties are optional
data class SubaccountConfiguration(
    var regionHost: String,
    var subaccount: String,
    var cloudUser: String,
    var cloudPassword: String,
    var displayName: String? = null,
    var locationId: String? = null
}

```

```

)
data class SubaccountInfo(
    val displayName: String,
    val regionHost: String,
    val subaccount: String,
    val tunnel: SubaccountTunnelInfo,
    val user: String,
    val _links: Map<String, HalLink>
)
data class SubaccountTunnelInfo(
    val state: String, //Connected
    val connectedSince: String, //timestamp formatted with client locale
    val connections: Int,
    val applicationConnections: List<String>,
    val serviceChannels: List<String>,
    val subaccountCertificate: X509CertificateInfo,
)
data class X509CertificateInfo(
    val notAfter: String,
    val notBefore: String,
    val subjectDN: String,
    val issuer: String
)
data class HalLink(val href: String)
data class SubaccountRefreshCred(
    var cloudUser: String,
    var cloudPassword: String
)
data class SystemMapping(
    val virtualHost: String,
    val virtualPort: String,
    val localHost: String,
    val localPort: String,
    val protocol: CommunicationProtocol,
    val backendType: BackendType,
    val sncPartnerName: String? = null,
    val hostInHeader: HostInHeader? = null,
    val sapRouter: String? = null,
    val authenticationMode: AuthenticationMode = AuthenticationMode.NONE,
    val description: String = ""
) {
    val _links: Map<String, HalLink> = mapOf()
}
enum class CommunicationProtocol {
    HTTP, HTTPS, RFC, RFCS, LDAP, LDAPS, TCP, TCPS
}
enum class BackendType {
    abapSys, netweaverCE, applServerJava, BC, PI, hana, netweaverGW,
    otherSAPsys, nonSAPsys
}
enum class HostInHeader {
    INTERNAL, VIRTUAL
}
enum class AuthenticationMode {
    NONE, X509_CERTIFICATE, X509_CERTIFICATE_LOCAL, KERBEROS
}
data class HttpResource(
    val id: String, //URI path
    val enabled: Boolean = true,
    val exactMatchOnly: Boolean = true,
    val webSocketUpgradeAllowed: Boolean = true,
    val description: String = ""
)
data class RfcResource(
    val id: String, //Function module or prefix for function module
    val enabled: Boolean = true,
    val exactMatchOnly: Boolean = true,
    val description: String = ""
)

```

1.2.2.5.13.2.4 High Availability Settings

↳ Sample Code

```
package com.sap.scc.examples
import com.github.kittinunf.fuel.Fuel
import com.github.kittinunf.fuel.core.FuelManager
import com.github.kittinunf.fuel.core.Headers
import com.github.kittinunf.fuel.core.extensions.authentication
import com.github.kittinunf.fuel.gson.jsonBody
import com.github.kittinunf.fuel.gson.responseObject
import com.github.kittinunf.result.Result
import java.net.URL
/*
    This example shows how to use REST APIs to change high availability settings
    of the Cloud Connector instances.
    As a prerequisite you need to install and start a shadow and a master
    instance.
    Afterwards perform the initial configuration of the master instance (see
    InitialConfiguration.kt).
    The configuration details for master and shadow instances can be found in
    scenario.json.
*/
fun main() {
    //Cloud Connector distribution generates only an untrusted self-signed
    certificate.
    //So for this demonstration use case we need to deactivate all trust
    checks.
    disableTrustChecks()
    //Use 'Connection: close' header, to make stateless communication more
    efficient
    FuelManager.instance.baseHeaders = mapOf("Connection" to "close")
    //Add output of cURL commands for revision
    //FuelManager.instance.addRequestInterceptor(LogRequestAsCurlInterceptor)
    //Load configuration from property file
    val config = loadScenarioConfiguration()
    //The high-availability role 'master' in Cloud Connector instance
    'master' is already set in the initial configuration.
    //Set the high-availability role to 'shadow' in Cloud Connector instance
    'shadow'
    Fuel.put("${config.shadow!!.url}/api/v1/configuration/connector/haRole")
        .authentication().basic(config.shadow!!.user,
    config.shadow!!.password)
        .body("shadow")
        .response { _, _, result ->
            when (result) {
                is Result.Failure -> processRequestError(result.error)
                is Result.Success -> println("high-availability role
    successfully set to 'shadow'")
            }
        }
        .join()
    //Enable HA in the master instance and define the allowed shadow hosts
    Fuel.put("${config.master!!.url}/api/v1/configuration/connector/ha/master/
    config")
        .header(Headers.CONTENT_TYPE, "application/json")
```

```

        .authentication().basic(config.master!!.user,
config.master!!.password)
        .body("""{"haEnabled":"true", "allowedShadowHost":"${
URL(config.shadow!!.url).host}"}""")
        .responseObject<HAMasterConfiguration> { _, _, result ->
            when (result) {
                is Result.Failure -> processRequestError(result.error)
                is Result.Success -> {
                    val masterConfig = result.get()
                    println("Master configuration: $masterConfig ")
                }
            }
        }
    }
    .join()
//The configuration of the shadow instance
var shadowConfig: HADeviceConfiguration? = null
//Get the current configuration
Fuel.get("${config.shadow!!.url}/api/v1/configuration/connector/ha/shadow/
config")
    .authentication().basic(config.shadow!!.user,
config.shadow!!.password)
    .responseObject<HADeviceConfiguration> { _, _, result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> shadowConfig = result.get()
        }
    }
    .join()
//Edit the retrieved configuration and set it
shadowConfig!!.masterPort = "${URL(config.master!!.url).port}"
shadowConfig!!.masterHost = URL(config.master!!.url).host
shadowConfig!!.ownHost = URL(config.master!!.url).host
Fuel.put("${config.shadow!!.url}/api/v1/configuration/connector/ha/shadow/
config")
    .authentication().basic(config.shadow!!.user,
config.shadow!!.password)
    .jsonBody(shadowConfig!!)
    .responseObject<HADeviceConfiguration> { _, _, result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> shadowConfig = result.get()
        }
    }
    .join()
//Set the HA link of both cloud connector instances to 'CONNECT'
Fuel.post("${config.shadow!!.url}/api/v1/configuration/connector/ha/
shadow/state")
    .header(Headers.CONTENT_TYPE, "application/json")
    .authentication().basic(config.shadow!!.user,
config.shadow!!.password)
    .body("""{"op":"CONNECT", "user":"${config.master!!.user}",
"password":"${config.master!!.password}"}""")
    .response { _, _, result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> println("Master (${config.master!!.url}) and shadow (${config.shadow!!.url}) successfully connected")
        }
    }
    .join()
}
//Data structures used by REST calls in this scenario
//Structure used to read the high availability settings for a Cloud
Connector master instance
data class HAMasterConfiguration(
    var haEnabled: Boolean? = null,
    var allowedShadowHost: String? = null
)

```

```

//Structure used to read and edit the high availability settings for a Cloud
Connector shadow instance
data class HAShadowConfiguration(
    var masterPort: String,
    var masterHost: String,
    var ownHost: String? = null,
    var checkIntervalInSeconds: Int?,
    var takeoverDelayInSeconds: Int?,
    var connectTimeoutInMillis: Int?,
    var requestTimeoutInMillis: Int?
)

```

1.2.2.5.13.2.5 Backup And Restore Configuration

↳ Sample Code

```

package com.sap.scc.examples
import com.github.kittinunf.fuel.Fuel
import com.github.kittinunf.fuel.core.BlobDataPart
import com.github.kittinunf.fuel.core.FuelManager
import com.github.kittinunf.fuel.core.Headers
import com.github.kittinunf.fuel.core.Method
import com.github.kittinunf.fuel.core.extensions.authentication
import com.github.kittinunf.result.Result
import java.io.ByteArrayInputStream
import java.io.File
/*
This example shows how to use REST APIs to perform the Backup and Restore of
the Cloud Connector configuration.
As a prerequisite you have a stable Cloud Connector configuration and you
want to save the backup for later use.

The configuration details can be found in scenario.json.
*/
fun main() {
    //Cloud Connector distribution generates only an untrusted self-signed
    certificate.
    //So for this demonstration use case we need to deactivate all trust
    checks.
    disableTrustChecks()
    //Use 'Connection: close' header, to make stateless communication more
    efficient
    FuelManager.instance.baseHeaders = mapOf("Connection" to "close")
    //Add output of CURL commands for revision
    //FuelManager.instance.addRequestInterceptor(LogRequestAsCurlInterceptor)
    //Load configuration from property file
    val config = loadScenarioConfiguration()
    //Create the backup configuration of the cloud connector "master".
    Fuel.post("${config.master!!.url}/api/v1/configuration/backup")
        .header(Headers.CONTENT_TYPE, "application/json")
        .authentication().basic(config.master!!.user,
    config.master!!.password)
        .body("""{"password":"my-very-secret-password"}""")
        .response { _, _, result ->
            when (result) {
                is Result.Failure -> processRequestError(result.error)
                is Result.Success ->
                    File("backup.zip").writeBytes(result.get())
            }
}

```

```

        }
        .join()
    //Restore the backup configuration if some fatal accidental changes
    should be reverted
    val formData = listOf("password" to "my-very-secret-password")
    Fuel.upload("${config.master!!.url}/api/v1/configuration/backup",
Method.PUT, formData)
        .add(BlobDataPart(ByteArrayInputStream(File("backup.zip").readBytes()))
, name = "backup"))
        .authentication().basic(config.master!!.user,
config.master!!.password)
        .response { _, _, result ->
        when (result) {
            is Result.Failure -> processRequestError(result.error)
            is Result.Success -> println("Backup configuration
successfully restored in (${config.master!!.url}) ")
        }
        .join()
}

```

1.2.2.5.13.2.6 Solution Management Integration

Sample Code

```

package com.sap.scc.examples
import com.github.kittinunf.fuel.Fuel
import com.github.kittinunf.fuel.core.FuelManager
import com.github.kittinunf.fuel.core.extensions.authentication
import com.github.kittinunf.fuel.gson.jsonBody
import com.github.kittinunf.fuel.gson.responseObject
import com.github.kittinunf.result.Result
import java.io.File
/*
 This example shows how to use REST APIs to configure the integration with
SAP solution management.
 In most cases it should be only necessary to pass a boolean flag in order to
enable or
 disable solution management integration. It is expected that SAP host agent
is already
 installed on the host as prerequisite for solution management integration.

This example executes requests against master and shadow instances.
The shadow instance is optional. Ignore the requests to shadow instance if
there is only
a master instance in your environment.

The configuration details for master and shadow instances can be found in
scenario.json.
*/
fun main() {
    //Cloud Connector distribution generates only an untrusted self-signed
certificate.
    //So for this demonstration use case we need to deactivate all trust
checks.
    disableTrustChecks()
    //Use 'Connection: close' header, to make stateless communication more
efficient
    FuelManager.instance.baseHeaders = mapOf("Connection" to "close")
}

```

```

//Add output of cURL commands for revision
//FuelManager.instance.addRequestInterceptor(LogRequestAsCurlInterceptor)
//Load configuration from property file
val config = loadScenarioConfiguration()
//First we check the current configuration of solution management
Fuel.get("${config.master!!.url}/api/v1/configuration/connector/
solutionManagement")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .responseObject<SolutionManagementConfiguration> { _, _, result ->
        when (result) {
            is Result.Success -> println("response: ${result.get()}")
            is Result.Failure -> processRequestError(result.error)
        }
    }
    .join()
//Configure the hostAgent path on the shadow instance
//Invoking this API on a shadow instance changes only the configuration.
//Only when invoking it on a master instance it also activates solution
management integration.
//Note: This step is not required if host agent is installed at the
default location
Fuel.post("${config.shadow!!.url}/api/v1/configuration/connector/
solutionManagement")
    .authentication().basic(config.shadow!!.user,
config.shadow!!.password)
    .jsonBody(SolutionManagementConfiguration(hostAgentPath = "/path/to/
hostAgent"))
    .response { _, _, result ->
        when (result) {
            is Result.Success -> println("new path to host agent was set")
            is Result.Failure -> processRequestError(result.error)
        }
    }
    .join()
//Turns on the solution management integration.
//Note: this operation is possible only on the master instance.
//In case of an HA setup, the flag enabled is propagated to the shadow
automatically.
//Note: optionally you can set here the new path for host agent and
enable DSR.
Fuel.post("${config.master!!.url}/api/v1/configuration/connector/
solutionManagement")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .jsonBody(SolutionManagementConfiguration())
    .response { _, _, result ->
        when (result) {
            is Result.Success -> println("solution management is
activated")
            is Result.Failure -> processRequestError(result.error)
        }
    }
    .join()
//Turns off the solution management integration.
//Note: this operation is possible only on the master instance.
//In case of an HA setup, the flag not enabled is propagated to the
shadow automatically
Fuel.delete("${config.master!!.url}/api/v1/configuration/connector/
solutionManagement")
    .authentication().basic(config.master!!.user,
config.master!!.password)
    .response { _, _, result ->
        when (result) {
            is Result.Success -> println("solution management is
deactivated")
            is Result.Failure -> processRequestError(result.error)
        }
    }

```

```

        }
        .join()
    //Registration file with LMDB model can be downloaded by following
    request.
    //This file can be used for a manual upload to solution management.
    //If solution management integration is active, the updates are triggered
    by configuration changes automatically.
    Fuel.get("${config.master!.url}/api/v1/configuration/connector/
    solutionManagement/registrationFile")
        .authentication().basic(config.master!.user,
    config.master!!.password)
        .response { _, _ , result ->
            when (result) {
                is Result.Success ->
                    File("lmdbModel.zip").writeBytes(result.get())
                is Result.Failure -> processRequestError(result.error)
            }
        }
        .join()
    }
    //Data structures used by REST calls in this scenario
    //Nullable properties are optional
    data class SolutionManagementConfiguration(
        var hostAgentPath: String? = null,
        var enabled: Boolean? = null,
        var dsrEnabled: Boolean? = null
    )

```

1.2.2.6 Configure an On-Premise User Store

Configure SAP BTP Java applications to use your corporate LDAP server or on-premise SAP system as a user store.

Prerequisites

- You have configured your Java cloud application to use an on-premise user provider and to consume its users via the Cloud Connector. To do this, execute the following command:

```

neo deploy --host <host> --account <subaccount name> --application
<application name> --source <path to WAR file> --user <e-mail or user name> --
vm-arguments "-Dcom.sap.cloud.security.um.user_provider_name=onpremise -
Dcom.sap.cloud.security.um.destination_name=onpremiseumconnector"

```

- You have created a connectivity destination to configure the on-premise user provider, using the following parameters:

```

Name=onpremiseumconnector
Type=HTTP
URL= http://scc.scim:80/scim/v1
Authentication=NoAuthentication
CloudConnectorVersion=2
ProxyType=OnPremise

```

- You are using only one domain for user authentication. Authentication to multiple domains including sub-domains is not supported.

For more information, see also [On-Premise User Store](#).

Context

If you configure your SAP BTP applications to use the corporate LDAP server or on-premise SAP system as a user store, the platform doesn't need to keep the entire user database but requests the necessary information from the on-premise user store. Java applications running on SAP BTP can use the on-premise system to check credentials, search for users, and retrieve details. In addition to the user information, the cloud application may request information about the groups a user belongs to.

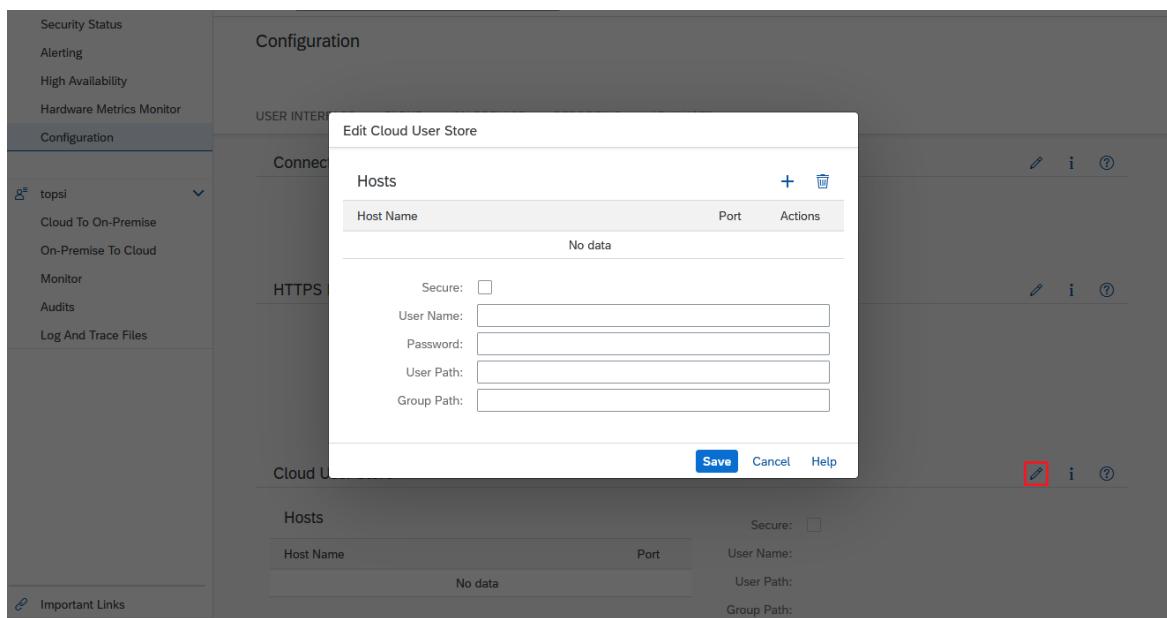
One way a Java cloud application can define user authorizations is by checking a user's membership to specific groups in the on-premise user store. The application uses the roles for the groups defined in SAP BTP. For more information, see [Managing Roles](#).

i Note

The configuration steps below are applicable only for Microsoft Active Directory (AD).

Procedure

1. From the main menu, choose *Configuration*.
2. From the *Cloud User Store* section on the *Cloud* tab, choose *Edit*.



3. To connect to LDAP using SSL, select *Secure*.
4. Manage the hosts and ports for your LDAP servers:

- Choose the *Add* icon to add a host.

i Note

Multiple hosts are currently not supported.

- Choose *Edit* to edit the selected host.
- Choose *Delete* to delete the selected hosts.

5. Enter the user name and password for the service user that is used to contact the LDAP system.

i Note

The user name must be fully qualified, including the AD domain suffix, for example,

john.smith@mycompany.com.

6. In *User Path*, specify the LDAP subtree that contains the users.
7. In *Group Path*, specify the LDAP subtree that contains the groups.
8. Choose **Save**.

Related Information

[Using an SAP System as an On-Premise User Store](#)

[Use LDAP for Authentication \[page 513\]](#)

[Managing Destinations \[page 39\]](#)

1.2.2.7 Using Service Channels

Configure Cloud Connector service channels to connect your on-premise network to specific services on SAP BTP.

Context

Cloud Connector service channels provide access from an external network to certain services on SAP BTP. The called services are not exposed to direct access from the Internet. The Cloud Connector ensures that the connection is always available and communication is secured.

Service Channel Type	Description
SAP HANA Database on SAP BTP	The service channel for the SAP HANA Database lets you access SAP HANA databases that run in the Cloud from database clients (for example, clients using ODBC/JDBC drivers). You can use the service channel to connect database, analytical, BI, or replication tools to your SAP HANA database in your SAP BTP subaccount.
RFC Connection to SAP BTP ABAP environment	The service channel for RFC supports calls from on-premise systems to the SAP BTP ABAP environment using RFC.

Next Steps

[Configure a Service Channel for an SAP HANA Database \[page 493\]](#)

[Connect DB Tools to SAP HANA via Service Channels \[page 496\]](#)

[Configure a Service Channel for RFC \[page 497\]](#)

Related Information

[Service Channels: Port Overview \[page 499\]](#)

1.2.2.7.1 Configure a Service Channel for an SAP HANA Database

Using Cloud Connector service channels, you can establish a connection to an SAP HANA database in SAP BTP that is not directly exposed to external access.

Context

The service channel for SAP HANA Database lets you access SAP HANA databases running in the cloud via ODBC/JDBC. You can use the service channel to connect database, analytical, BI, or replication tools to an SAP HANA database in your SAP BTP subaccount.

Restrictions

This feature is only available in regions (data centers) that provide the SAP HANA service (that is, SAP data centers (Neo environment), Azure and AWS).

For more information, see [SAP HANA Service for SAP BTP Getting Started Guide](#).

- In AWS regions, you can currently use the service channel only for the SAP HANA service provisioned *before June 4, 2018* (old version).
- If you are using the SAP HANA service provisioned *after June 4, 2018* (new version, available on AWS and GCP) or the *SAP HANA Cloud service*, you can currently access SAP HANA databases only directly, without going through the Cloud Connector.

For more information, see [Connecting to an SAP HANA Service Instance Directly from SAP HANA Clients](#).

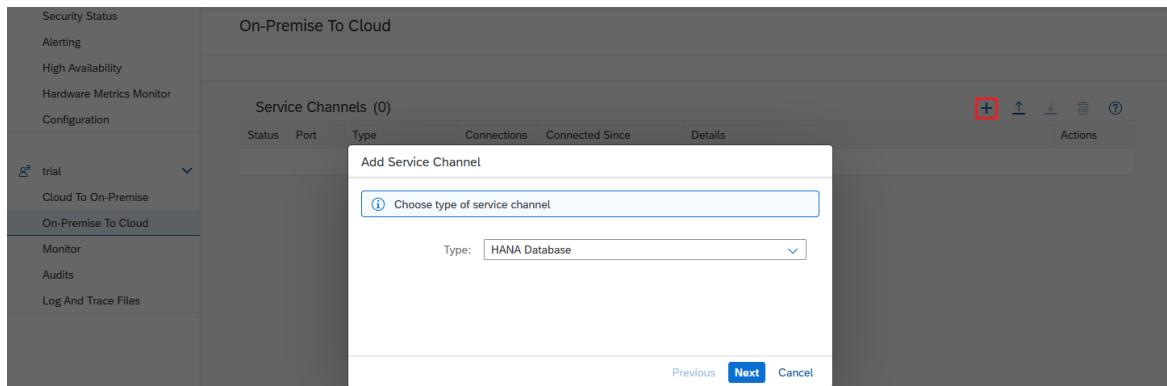
To find detailed information for a specific Cloud Foundry region and SAP HANA service version, see [Find the Right Guide](#).

i Note

The following procedure requires a productive SAP HANA instance that is available in the *same* subaccount. You cannot access an SAP HANA instance that is owned by a different subaccount within the same or another global account (shared SAP HANA database). See also [Sharing Databases with Other Subaccounts](#).

Procedure

1. From your subaccount menu, choose *On-Premise To Cloud*.
2. Choose *Add* (+).



3. In the *Add Service Channel* dialog, leave the default value **HANA Database** in the <Type> field.
4. Choose *Next*.
5. Choose the SAP HANA instance name. If you cannot select it from the drop-down list, enter the instance name manually. In the **Neo** environment, it must match one of the names (IDs) shown in the cockpit under **SAP HANA/SAP ASE** **Databases & Schemas**, in the <DB/Schema ID> column.

i Note

The SAP HANA instance name is case-sensitive.

In the **Cloud Foundry** environment, the format of the SAP HANA instance name includes the Cloud Foundry space name, the database name and the database ID.

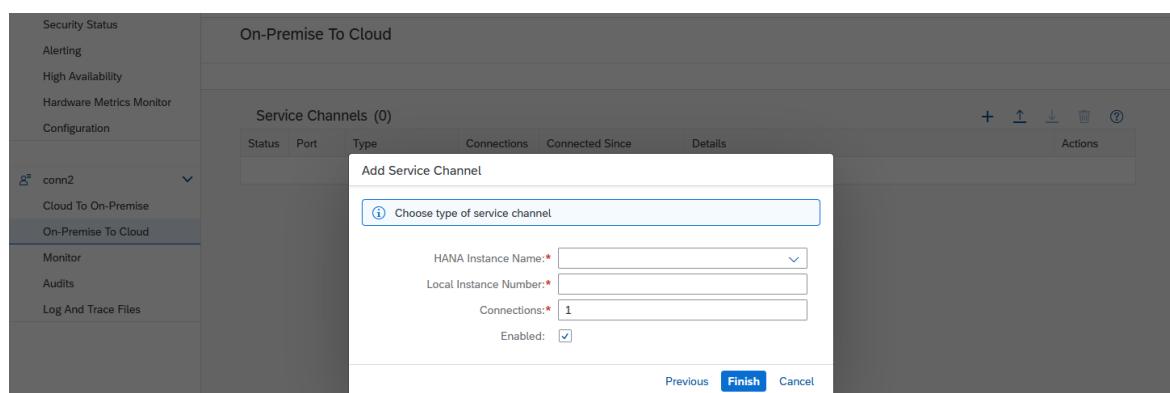
Example:

`test:testInstance:3fcc976d-457a-474e-975b-e572600f474e:de19c262-a1fc-4096-bfce-1c41388e4b49`

where

- `test`: Cloud Foundry space name
- `testInstance`: tenant database instance name
- `3fcc976d-457a-474e-975b-e572600f474e:de19c262-a1fc-4096-bfce-1c41388e4b49`: database ID

6. Specify the local instance number. This is a double-digit number which computes the local port used to access the SAP HANA instance in the cloud. The local port is derived from the local instance number as **3<instance number>15**. For example, if the instance number is **22**, then the local port is **32215**. The local instance number must not be **00**. This instance number might cause problems with some SAP HANA clients.
7. Specify the number of physical connections to SAP BTP. The default value is **1**. One physical connection can open various virtual connections through multiplexing.



8. Leave *Enabled* selected to establish the channel immediately after clicking *Finish*, or unselect it if you don't want to establish the channel immediately.
9. Choose *Finish*.

Next Steps

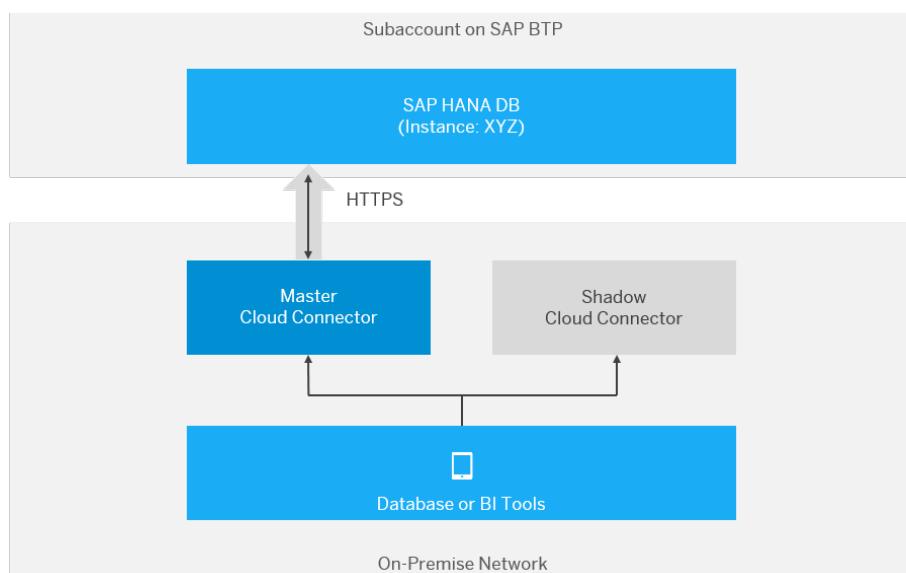
Once you have established an SAP HANA Database service channel, you can connect on-premise database or BI tools to the selected SAP HANA database in the cloud. This can be done by using **<cloud_connector_host>:<local_HANA_port>** in the JDBC/ODBC connect strings.

See [Connect DB Tools to SAP HANA via Service Channels \[page 496\]](#).

1.2.2.7.2 Connect DB Tools to SAP HANA via Service Channels

Context

You can connect database, BI, or replication tools running in on-premise network to an SAP HANA database on SAP BTP using service channels of the Cloud Connector. You can also use the high availability support of the Cloud Connector on a database connection. The picture below shows the landscape in such a scenario.



Follow the steps below to set up failover support, configure a service channel, and connect on-premise DB tools via JDBC or ODBC to the SAP HANA database.

- For more information on using SAP HANA instances, see [Using an SAP HANA XS Database System](#)
- For the connection string via ODBC you need a corresponding database user and password (see step 4 below). See also: [Creating Database Users](#).
- Find detailed information on failover support in the *SAP HANA Administration Guide*: [Configuring Clients for Failover](#).

i Note

This link points to the latest release of *SAP HANA Administration Guide*. Refer to the [SAP BTP Release Notes](#) to find out which SAP HANA SPS is supported by SAP BTP. Find the list of guides for earlier releases in the *Related Links* section below.

Procedure

1. To establish a highly available connection to one or multiple SAP HANA instances in the cloud, we recommend that you make use of the failover support of the Cloud Connector. Set up a master and a shadow instance. See [Install a Failover Instance for High Availability \[page 518\]](#).
2. In the master instance, configure a service channel to the SAP HANA database of the SAP BTP subaccount to which you want to connect. If, for example, the chosen HANA instance is 01, the port of the service channel is 30115. See also [Configure a Service Channel for an SAP HANA Database \[page 493\]](#).
3. Connect on-premise DB tools via JDBC to the SAP HANA database by using the following connection string:

Example:

```
jdbc:sap://<cloud-connector-master-host>:30115;<cloud-connector-shadow-host>:  
30115[/?<options>]
```

The SAP HANA JDBC driver supports failover out of the box. All you need is to configure the shadow instance of the Cloud Connector as a failover server in the JDBC connection string. The different options supported in the JDBC connection string are described in: [Connect to SAP HANA via JDBC](#)

4. You can also connect on-premise DB tools via ODBC to the SAP HANA database. Use the following connection string:

```
"DRIVER=HDBODBC32;UID=<user>;PWD=<password>;SERVENODE=<cloud-connector-  
master-host>:30115;<cloud-connector-shadow-host>:30115;"
```

Related Information

[Guides for earlier releases of SAP HANA](#)

1.2.2.7.3 Configure a Service Channel for RFC

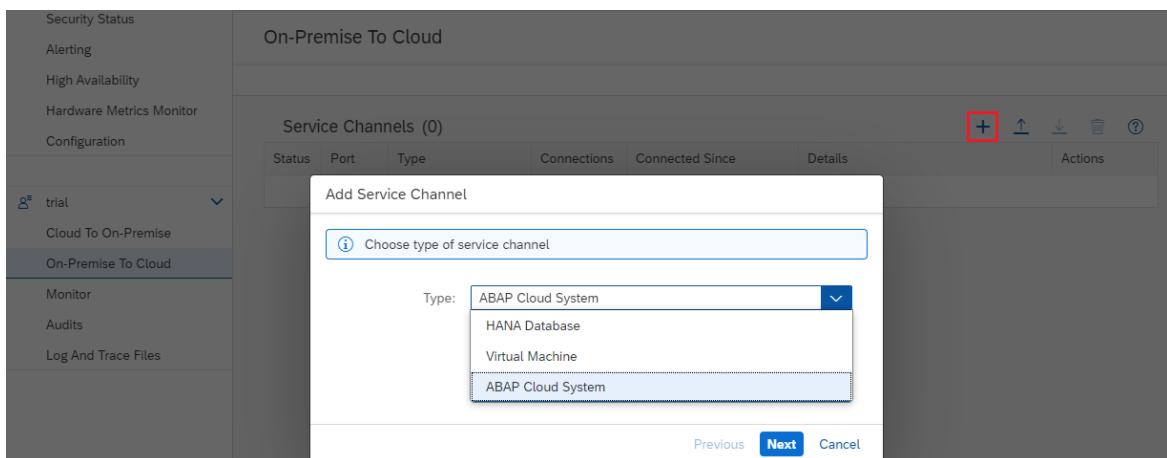
For scenarios that need to call from on-premise systems to SAP BTP ABAP environment using RFC, you can establish a connection to an ABAP Cloud tenant host. To do this, select  [On-Premise to Cloud](#)  [Service Channels](#) in the Cloud Connector.

Prerequisites

- When using the default connectivity setup with the **Cloud Foundry** subaccount in which the system has been provisioned, you can use a service channel without additional configuration, as long as the system is a single-tenant system.
- When using connectivity via a **Neo** subaccount, you must create a communication arrangement for the scenario SAP_COM_0200. For more information, see [Create a Communication Arrangement for Cloud Connector Integration](#) (documentation for ABAP environment on SAP BTP).

Procedure

- From your subaccount menu, choose *On Premise To Cloud*.
- Choose the *Add* (+) icon.

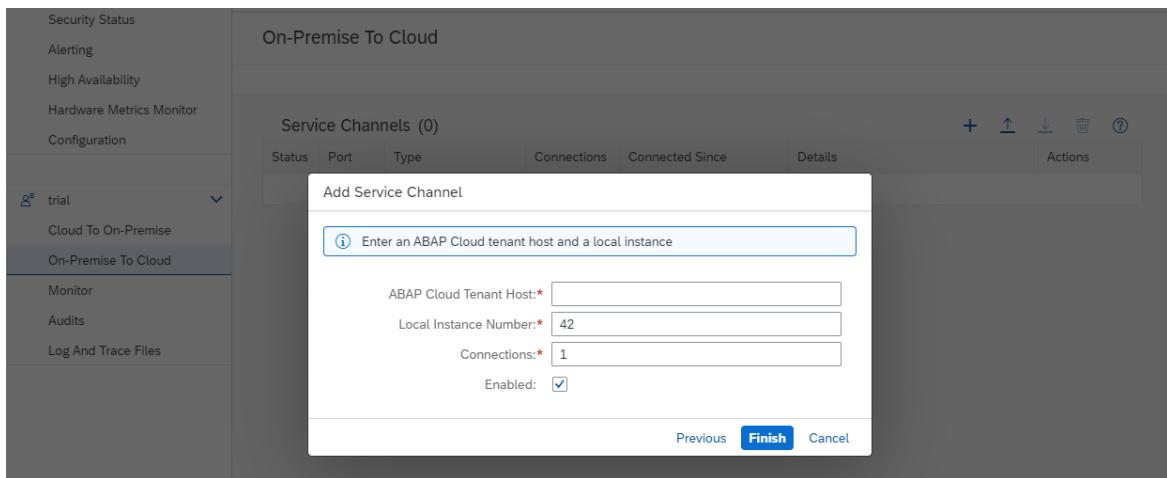


- In the *Add Service Channel* dialog, select **ABAP Cloud System** from the drop-down list of supported channel types.
- Choose *Next*. The *ABAP Cloud System* dialog opens.
- Enter the <ABAP Cloud Tenant Host> that you want to connect to.

i Note

For SAP BTP ABAP environment, the tenant host is
 <serviceinstanceguid>.abap.<region>.hana.ondemand.com (note that this is not the frontend address with the abap-web subdomain). The region is, for example, **eu10** or **us10**.

- In the same dialog window, define the <Local Instance Number> under which the ABAP Cloud system is reachable for the client systems. You can enter any instance number for which the port is not used yet on the Cloud Connector host. The port numbers result from the following pattern:
33<LocalInstanceNumber>.
- In the same dialog window, leave *Enabled* selected to establish the channel immediately after choosing *Finish*. Unselect it if you don't want to establish the channel immediately.



- Choose *Finish*.

i Note

When addressing an ABAP Cloud system in a **destination configuration**, you must enter the *Cloud Connector host* as application server host. As instance number, specify the `<Local Instance Number>` that you configured for the service channel. As user, you must provide the *business user* name but not the technical user name associated with the same.

1.2.2.7.4 Service Channels: Port Overview

A service channel overview lets you see the details of all service channels that are used by a Cloud Connector installation.

The service channel port overview lists all service channels that are configured in the Cloud Connector. It lets you see at a glance, which server ports are used by a Cloud Connector installation.

In addition, you can find the following information about each service channel:

- Status (enabled, disabled, disconnected)
- Service channel type (SAP HANA Database, Virtual Machine, ABAP Cloud System)
- Assigned subaccount
- Details (for example, the assigned SAP HANA instance name or virtual machine name)

From the *Actions* column, you can switch directly to the *On-Premise To Cloud* section of the corresponding subaccount and edit the selected service channel.

To find the overview list, choose *Connector* from the navigation menu and go to section *Service Channels Overview*:

The screenshot shows the SAP Cloud Connector Administration interface. On the left, there's a navigation sidebar with sections like 'Connector', 'Security Status', 'Alerting', 'High Availability', 'Hardware Metrics Monitor', 'Configuration', and a expanded section for 'conn2' with 'Cloud To On-Premise', 'On-Premise To Cloud', 'Monitor', 'Audits', and 'Log And Trace Files'. The main area has a header 'Subaccount: conn2' with a search icon. Below it is a 'Connector Overview' section with fields: 'Connector ID:', 'Local Name: global.corp.sap', 'Local IP:', 'Security Status: Risk' (with a red crossed-out checkmark), 'High Availability: Disabled' (with a red diamond icon), and 'Alerts: 1'. There are buttons for '+ Add Subaccount', 'Backup', and '...'. Below this is a 'Subaccount Dashboard (1)' table:

Status	Subaccount	Display Name	Location ID	Region	Actions
conn2	conn2			>

Below that is a 'Service Channels Overview (2)' table:

Status	Port	Type	Subaccount	Details	Actions
3308	ABAP Cloud System	conn2	SomeHost	>	
34215	HANA Database	conn2	Test	>	

1.2.2.8 Configure Trust

Set up an allowlist for cloud applications and a trust store for on-premise systems in the Cloud Connector.

Tasks

[Trust Cloud Applications in the Cloud Connector \[page 500\]](#)

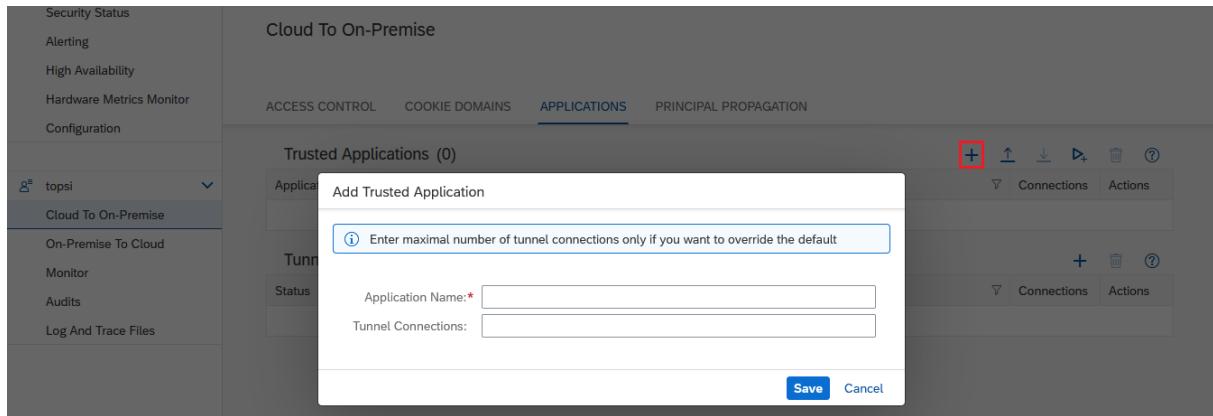
[Configure the Trust Store \[page 502\]](#)

Trust Cloud Applications in the Cloud Connector

! Restriction

Currently, the complete implementation of this feature is available only for interaction with the Neo environment.

By default, all applications within a subaccount are allowed to use the Cloud Connector associated with the subaccount they run in. However, this behavior might not be desired in any scenario. For example, this may be acceptable for some applications, as they must interact with on-premise resources, while other applications, for which it is not transparent whether they try to receive on-premise data, might turn out to be malicious. For such cases, you can use an application allowlist.



As long as there is no entry in this list, all applications are allowed to use the Cloud Connector. If one or more entries appear in the allowlist, then only these applications are allowed to connect to the exposed systems in the Cloud Connector.

You can add, edit, or delete entries as follows:

1. From your subaccount menu, choose *Cloud to On-Premise* and go to the *Applications* tab.
2. To add an application, choose the *Add* icon in section *Trusted Applications*.
3. Enter the <Application Name> in the *Add Tunnel Application* dialog.

i Note

To add all applications that are listed in section *Tunnel Connection Limits* on the same screen, you can also use the *Upload* button next to the *Add* button. The list *Tunnel Connection Limits* shows all applications for which a specific maximal number of tunnel connections was specified. See also: [Configure Tunnel Connections \[page 506\]](#).

4. (Optional) Enter the maximal number of <Tunnel Connections> only if you want to override the default value.
5. Choose *Save*.

i Note

The application name is visible in the SAP BTP cockpit under *Applications* *Java Applications*. To allow a subscribed application, you must add it to the allowlist in the format **<providerSubaccount>:<applicationName>**. In particular, when using HTML5 applications, an implicit subscription to services:dispatcher is required.

To edit an existing entry:

1. Choose the *Edit* button.
2. When you are done, select *Save*.

To remove an application from the list:

1. Select the entry.
2. Choose *Delete*.

To delete all entries, choose *Delete All*.

To add all applications from section *Tunnel Connection Limits* to the allowlist, choose the button *Add all applications...* from section **Trusted Applications**.

The screenshot shows the SAP Cloud Connector configuration interface. On the left, a sidebar lists various monitoring and configuration options like Security Status, Alerting, High Availability, and Configuration. A dropdown menu shows 'Cloud To On-Premise' selected. The main panel has tabs for ACCESS CONTROL, COOKIE DOMAINS, APPLICATIONS (which is selected), and PRINCIPAL PROPAGATION. Under APPLICATIONS, there are two sections: 'Trusted Applications (0)' and 'Tunnel Connections Limits (1)'. The 'Trusted Applications' section has a table with columns for Application Name, Connections, and Actions. The 'Tunnel Connections Limits' section has a table with columns for Status, Application Name, Connections, and Actions. A red box highlights the 'Add all applications...' button in the 'Trusted Applications' header.

[Back to Tasks \[page 500\]](#)

Configure the Trust Store

By default, the Cloud Connector trusts every on-premise system when connecting to it via TLS. As this may be an undesirable behavior from a security perspective, you can configure a trust store that acts as an allowlist of trusted certificate authorities. Any TLS server certificate issued by one of those CAs will be considered trusted. If the CA that has issued a concrete server certificate is not included in the trust store, the server is considered untrusted and the connection will fail.

You can configure the trust store as follows:

- From the main menu, choose *Configuration*.
- Select the *On Premise* tab, section *Trust Store*.

The screenshot shows the SAP Cloud Connector configuration interface. The sidebar includes Security Status, Alerting, High Availability, Hardware Metrics Monitor, and Configuration. A dropdown shows 'topsi' selected. The main area has tabs for USER INTERFACE, CLOUD, ON PREMISE (selected), REPORTING, and ADVANCED. Under ON PREMISE, there's a 'System Certificate' section. A modal dialog titled 'Add X.509 Certificate' is open, prompting to 'Select a file with extension .der or .cer'. The background shows a table for 'X.509 Certificate' with columns for Status and Actions. A red box highlights the 'Add' button in the 'Actions' column of the table.

An empty trust store does not impose any restrictions on the trusted on-premise systems. It becomes an allowlist as soon as you add the first public key.

i Note

You must provide the CA's X.509 certificates in `.der` or `.cer` format.

[Back to Tasks \[page 500\]](#)

1.2.2.9 Configure Domain Mappings for Cookies

Context

Some HTTP servers return cookies that contain a `domain` attribute. For subsequent requests, HTTP clients should send these cookies to machines that have host names in the specified domain.

For example, if the client receives a cookie like the following:

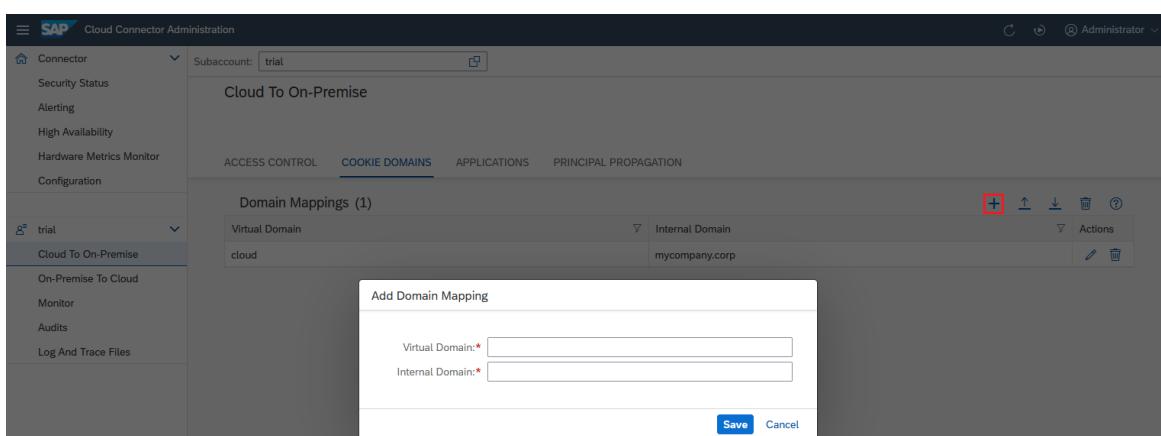
```
Set-Cookie: cookie-field=some-value; domain=mycompany.corp; path=...; ...
```

It returns the cookie in follow-up requests to all hosts like `ecc60.mycompany.corp`, `crm40.mycompany.corp`, and so on, if the other attributes like `path` and `attribute` require it.

However, in a Cloud Connector setup between a client and a Web server, this may lead to problems. For example, assume that you have defined a virtual host `sales-system.cloud` and mapped it to the internal host name `ecc60.mycompany.corp`. The client "thinks" it is sending an HTTP request to the host name `sales-system.cloud`, while the Web server, unaware of the above host name mapping, sets a cookie for the domain `mycompany.corp`. The client does not know this domain name and thus, for the next request to that Web server, doesn't attach the cookie, which it should do. The procedure below prevents this problem.

Procedure

1. From your subaccount menu, choose [Cloud To On-Premise](#), and go to the [Cookie Domains](#) tab.
2. Choose [Add](#).
3. Enter `cloud` as the virtual domain, and your company name as the internal domain.
4. Choose [Save](#).



The Cloud Connector checks the Web server's response for **Set-Cookie** headers. If it finds one with an attribute domain=intrantet.corp, it replaces it with domain=sales.cloud before returning the HTTP response to the client. Then, the client recognizes the domain name, and for the next request against [www1.sales.cloud](#) it attaches the cookie, which then successfully arrives at the server on **machine1.intranet.corp**.

i Note

Some Web servers use a syntax such as domain=.intranet.corp (RFC 2109), even though the newer RFC 6265 recommends using the notation without a dot.

i Note

The value of the domain attribute may be a simple host name, in which case no extra domain mapping is necessary on the Cloud Connector. If the server sets a cookie with domain=machine1.intranet.corp, the Cloud Connector automatically reverses the **mapping machine1.intranet.corp** to [www1.sales.cloud](#) and replaces the cookie domain accordingly.

Related Information

[Configure Access Control \[page 379\]](#)

1.2.2.10 Configure Solution Management Integration

Activate Solution Management reporting in the Cloud Connector.

If you want to monitor the Cloud Connector with the SAP Solution Manager, you can install a host agent on the machine of the Cloud Connector and register the Cloud Connector on your system.

Prerequisites

- You have installed the SAP Diagnostics Agent and SAP Host Agent on the Cloud Connector host and connected them to the SAP Solution Manager. As of Cloud Connector version 2.11.2, the RPM on Linux ensures that the host agent configuration is adjusted and that user groups are setup correctly. For more details about the host agent and diagnostics agent, see [SAP Host Agent](#) and the SCN Wiki [SAP Solution Manager Setup/Managed System Checklist](#).
See also SAP notes [2607632](#) (SAP Solution Manager 7.2 - Managed System Configuration for SAP Cloud Connector) and [1018839](#) (Registering in the System Landscape Directory using sldreg). For consulting, contact your local SAP partner.

i Note

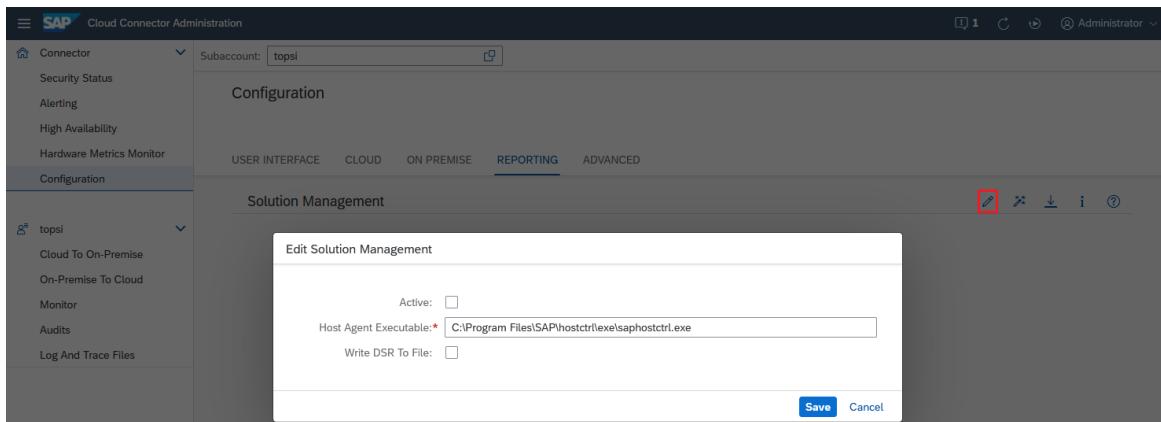
Linux OS: if you installed the host agent after installing the Cloud Connector, you can execute `enableSolMan.sh` in the installation directory (available as of Cloud Connector version 2.11.2) to adjust the host agent configuration and user group setup. This action requires root permission.

- The SAP Solution Manager must be of release 7.2 SP06 or higher.

Procedure

To enable the integration, do the following:

1. From the Cloud Connector main menu, choose In section *Solution Management* of the *Reporting* tab, select *Edit*.



2. Select the *Active* checkbox.
3. In the field *<Host Agent>*, specify the location of the host agent as filepath.
4. If you want to store the reporting results (*Dynamic Statistical Records*), select *Write DSR To File*.
5. Choose *Save*.

i Note

To download the registration file `lmdbModel.xml`, choose the icon *Download registration file* from the *Reporting* tab.

Related Information

[Monitoring \[page 526\]](#)

1.2.2.11 Configure Tunnel Connections

Adapt connectivity settings that control the throughput by choosing the appropriate limits (maximal values).

If required, you can adjust the following parameters for the communication tunnel by changing their default values:

- Application Tunnel Connections (default: 1)

i Note

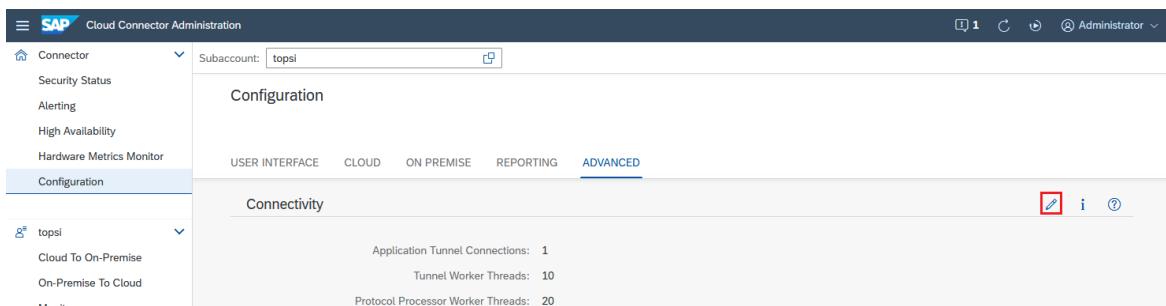
This parameter specifies the default value for the maximal number of tunnel connections *per application*. The value must be higher than 0.

- Tunnel Worker Threads (default: 10)
- Protocol Processor Worker Threads (default: 20)

For detailed information on connection configuration requirements, see [Configuration Setup \[page 300\]](#).

To change the parameter values, do the following:

1. From the Cloud Connector main menu, choose ► *Configuration* ► *Advanced* ▾. In section *Connectivity*, select *Edit*.



2. In the *Edit Connectivity Settings* dialog, change the parameter values as required.
3. Choose *Save*.

Additionally, you can specify the number of allowed tunnel connections for each application that you have specified as a [trusted application \[page 353\]](#).

i Note

If you don't change the value for a trusted application, it keeps the default setting specified above. If you change the value, it may be higher or lower than the default and must be higher than 0.

To add a specific connection limit to a trusted application, do the following:

1. From your subaccount menu, choose ► *Cloud To On-Premise* ► *Applications* ▾. In section *Tunnel Connection Limits*, choose *Add*.

Status	Application Name	Connections	Actions
<input checked="" type="checkbox"/>	abc	5	Edit Delete

- In the *Edit Tunnel Connections Limit* dialog, enter the <Application Name> and change the number of <Tunnel Connections> as required.

i Note

The application name is visible in the SAP BTP cockpit under **Applications** **Java Applications**. To allow a subscribed application, you must add it to the allowlist in the format **<providerSubaccount>:<applicationName>**. In particular, when using HTML5 applications, an implicit subscription to services:dispatcher is required.

- Choose *Save*.

To edit this setting, select the application from the *Limits* list and choose *Edit*.

1.2.2.12 Configure the Java VM

Adapt the JVM settings that control memory management.

If required, you can adjust the following parameters for the Java VM by changing their default values:

- Initial Heap Size (default: 1024 MB)
- Maximal Heap Size (default: 1024 MB)
- Maximal Direct Memory (default: 2 GB)

i Note

A restart is required when changing JVM settings.

We recommend that you set the initial heap size equal to the maximal heap size, to avoid memory fragmentation.

To change the parameter values, do the following:

- From the Cloud Connector main menu, choose **Configuration** **Advanced**. In section **JVM**, select *Edit*.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a 'Connector' dropdown set to 'topsi'. Under 'topsi', there are links for 'Cloud To On-Premise', 'On-Premise To Cloud', 'Monitor', 'Audits', and 'Log And Trace Files'. The main content area has a 'Subaccount: topsi' input field. Below it, the 'Configuration' tab is selected, showing tabs for 'USER INTERFACE', 'CLOUD', 'ON PREMISE', 'REPORTING', and 'ADVANCED'. The 'ADVANCED' tab is active, displaying the 'Connectivity' and 'JVM' sections. The 'JVM' section contains parameters like 'Initial Heap Size: 1024m', 'Maximal Heap Size: 1024m', and 'Maximal Direct Memory: 2G'. A red box highlights the 'Edit' icon next to the 'Initial Heap Size' parameter.

2. In the [Edit JVM Settings](#) dialog, change the parameter values as required.
3. Choose [Save](#).

1.2.2.13 Configuration Backup

You can backup and restore your Cloud Connector configuration.

To backup or restore your Cloud Connector configuration, do the following:

1. From the Cloud Connector main menu, choose [Connector](#).

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a 'Connector' dropdown set to 'topsi'. Under 'topsi', there are links for 'Security Status', 'Alerting', 'High Availability', 'Hardware Metrics Monitor', and 'Configuration'. The main content area shows a 'Connector' section with a '+ Add Subaccount' button, a 'Backup' button (highlighted with a red box), and a 'Restore' button.

2. To backup or restore your configuration, choose the respective icon in the upper right corner of the screen.
 1. To backup your configuration, enter and repeat a password in the [Backup](#) dialog and choose [Backup](#).

i Note

An archive containing a snapshot of the current Cloud Connector configuration is created and downloaded by your browser. You can use this archive to restore the current state on this or a new Cloud Connector installation, if the original installation can no longer be used. *Do not* restore the backup on a second Cloud Connector, while the instance from which the backup was taken is *still active*. Such a setup might cause issues and is therefore not supported.

For security reasons, some files are encrypted, using the password provided for the backup procedure.

2. To restore your configuration, enter the required *Archive Password* and the *Login Password* of the currently logged-in administrator user in the [Restore from Archive](#) dialog and choose [Restore](#).

i Note

The restore action overwrites the current configuration of the Cloud Connector. It will be permanently lost unless you have created another backup before restoring. Upon successfully restoring the configuration, the Cloud Connector restarts automatically. All sessions are then

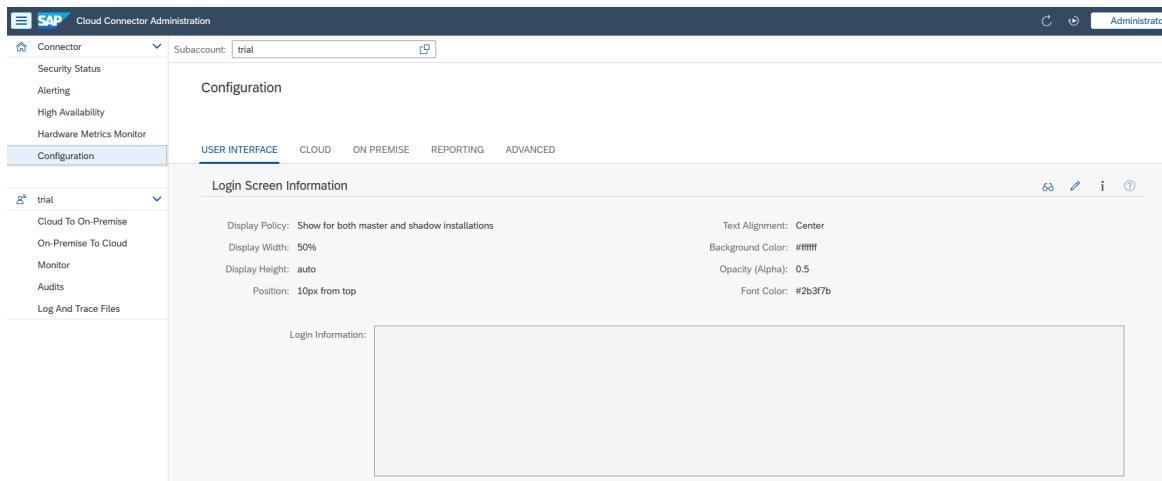
terminated. The `props.ini` file however is treated in a special way. If the file in the backup differs from the one that is used in the current installation, it will be placed next to the original one as `props.ini.restored`. If you want to use the `props.ini.restored` file, replace the existing one on OS level and restart the Cloud Connector.

1.2.2.14 Configure Login Screen Information

Add additional information to the login screen and configure its appearance.

To configure the login screen information, proceed as follows:

1. Go to   and press the *Edit* button in section *Login Screen Information*.



As a result, the following dialog is opened:

Edit Login Information

Display Policy

- Show for both master and shadow installations Show only if this is a master installation
 Show only if this is a shadow installation Do not show at all

Login Display Properties



Display Width:	<input type="text" value="50%"/>	Background Color:	<input type="text" value="#ffffff"/> <input type="button" value="□"/>
Display Height:	<input type="text" value="auto"/>	Opacity (Alpha):	<input type="text" value="0.5"/>
Text Alignment:	<input type="text" value="Center"/> <input type="button" value="▼"/>	Font Color:	<input type="text" value="#2b3f7b"/> <input type="button" value="□"/>
Position:	<input type="text" value="10px"/>	<input checked="" type="radio"/> from top <input type="radio"/> from bottom	

Login Information



Enter an HTML fragment to be displayed as login information. Restrictions apply. Compliance with these restrictions will be checked when saving. Consult the documentation for details.

2. In section *Display Policy*, select a display policy for the login screen information.
3. In section *Display Properties*, specify your preferred display properties (appearance and position):
 - The login information is displayed in a box with rounded corners. You can specify its width and height in pixels (unit px) or as a percentage (unit %).

i Note

Omitting any value triggers the default or auto behavior.

- For the width, the default behavior is equivalent to 100%.
- For the height, the default behavior sets the height to a value that accommodates the login information (that is, the given HTML fragment). For extensive information we therefore recommend that you limit the height to a suitable pixel or percentage value to induce scrolling, and to prevent the box from growing beyond a reasonable height.

- When customizing the **background color** of the box, the **opacity** (of the background color), **font color**, or **text alignment**, then the section *Login Information* automatically switches to preview mode. That way you can follow live the changes made to the appearance of the box and of the information displayed inside it.

i Note

You can hide the box and to show only the text of the login information by choosing an opacity value of **0** (opacity is the opposite of transparency. No opacity means *complete transparency*).

- You can **position the box** containing the login information at the top or bottom of the login page. To do this, set the field **<Position>** to the corresponding pixel or percentage value.
4. Enter the information to be displayed in section *Login Information*. The information must be supplied as an HTML fragment. There is a limited number of tags that can be used. Attributes available for these tags are subject to restrictions.

Available Tag	Attribute Restriction
p	Only attribute style is permitted, with property <code>text-align</code> set to a valid value (left , right , center , or justified)
ul	No attributes allowed
ol	No attributes allowed
li	No attributes allowed
br	No attributes allowed
h1	No attributes allowed
h2	No attributes allowed
h3	No attributes allowed
i	No attributes allowed
b	No attributes allowed
a	Must have exactly two attributes: <ul style="list-style-type: none"> <code>href</code> (its value must be a <URL> with protocol http or https) <code>target</code> (its value must be "_blank")

HTML syntax checking is strict. Attribute values must be enclosed by double quotes. Missing or unmatched opening or closing tags are not permitted.

i Note

Tag `br` does not require a closing tag as there cannot be any inner HTML.

1.2.3 Operations

Learn more about operating the Cloud Connector, using its administration tools and optimizing its functions.

Topic	Description
Configure Named Cloud Connector Users [page 512]	If you operate an LDAP server in your system landscape, you can configure the Cloud Connector to use the named users who are available on the LDAP server instead of the default Cloud Connector users.
High Availability Setup [page 517]	The Cloud Connector lets you install a redundant (shadow) instance, which monitors the main (master) instance.
Change the UI Port [page 523]	Use the changeport tool (Cloud Connector version 2.6.0+) to change the port for the Cloud Connector administration UI. .
Connect and Disconnect a Cloud Subaccount [page 524]	As a Cloud Connector administrator, you can connect the Cloud Connector to (and disconnect it from) the configured cloud subaccount.
Secure the Activation of Traffic Traces [page 525]	Tracing of network traffic data may contain business critical information or security sensitive data. You can implement a "four-eyes" (double check) principle to protect your traces (Cloud Connector version 1.3.2+).
Monitoring [page 526]	Use various views to monitor the activities and state of the Cloud Connector.
Alerting [page 553]	Configure the Cloud Connector to send email alerts whenever critical situations occur that may prevent it from operating.
Audit Logging [page 556]	Use the auditor tool to view and manage audit log information (Cloud Connector version 2.2+).
Troubleshooting [page 559]	Information about monitoring the state of open tunnel connections in the Cloud Connector. Display different types of logs and traces that can help you troubleshoot connection problems.
Process Guidelines for Hybrid Scenarios [page 564]	How to manage a hybrid scenario, in which applications running on SAP BTP require access to on-premise systems using the Cloud Connector.

1.2.3.1 Configure Named Cloud Connector Users

Set up LDAP-based user management for the Cloud Connector.

LDAP-Based User Management

We recommend that you configure LDAP-based user management for the Cloud Connector to allow only named administrator users to log on to the administration UI.

This guarantees traceability of the Cloud Connector configuration changes via the Cloud Connector audit log. If you use the default and built-in `Administrator` user, you cannot identify the actual person or persons who perform configuration changes. Also, you will not be able to use different types of user groups.

Configuration

If you have an LDAP server in your landscape, you can configure the Cloud Connector to authenticate Cloud Connector users against the LDAP server.

Valid users or user groups must be assigned to one of the following roles:

- Administrator users: `admin` or `sccadmin`
- Display users: `sccdisplay` or `sccmonitoring`

i Note

The role `sccmonitoring` provides access to the monitoring APIs, and is particularly used by the SAP Solution Manager infrastructure, see [Monitoring APIs \[page 534\]](#). It cannot be used to access the Cloud Connector administration UI.

- Support users: `sccsupport`

Alternatively, you can define custom role names for each of these user groups, see: [Use LDAP for Authentication \[page 513\]](#).

Once configured, the default Cloud Connector `Administrator` user becomes inactive and can no longer be used to log on to the Cloud Connector.

1.2.3.1.1 Use LDAP for Authentication

You can use LDAP (Lightweight Directory Access Protocol) to configure Cloud Connector authentication.

After installation, the Cloud Connector uses file-based user management by default. Alternatively, the Cloud Connector also supports LDAP-based user management. If you operate an LDAP server in your landscape, you can configure the Cloud Connector to use the LDAP user base.

If LDAP authentication is active, you can assign users or user groups to the following default roles:

Default Role	Authorization
<code>sccadmin</code> or <code>admin</code>	Administrate the Cloud Connector (all CRUD operations).

Default Role	Authorization
sccdisplay	Access the Cloud Connector administration UI in read-only mode.
sccsupport	<ul style="list-style-type: none"> Access the Cloud Connector administration UI in read-only mode. Perform support-related tasks like setting trace levels or creating a thread dump.
sccmonitoring	Provides access to the monitoring APIs, and is particularly used by the SAP Solution Manager infrastructure, see Monitoring APIs [page 534] .

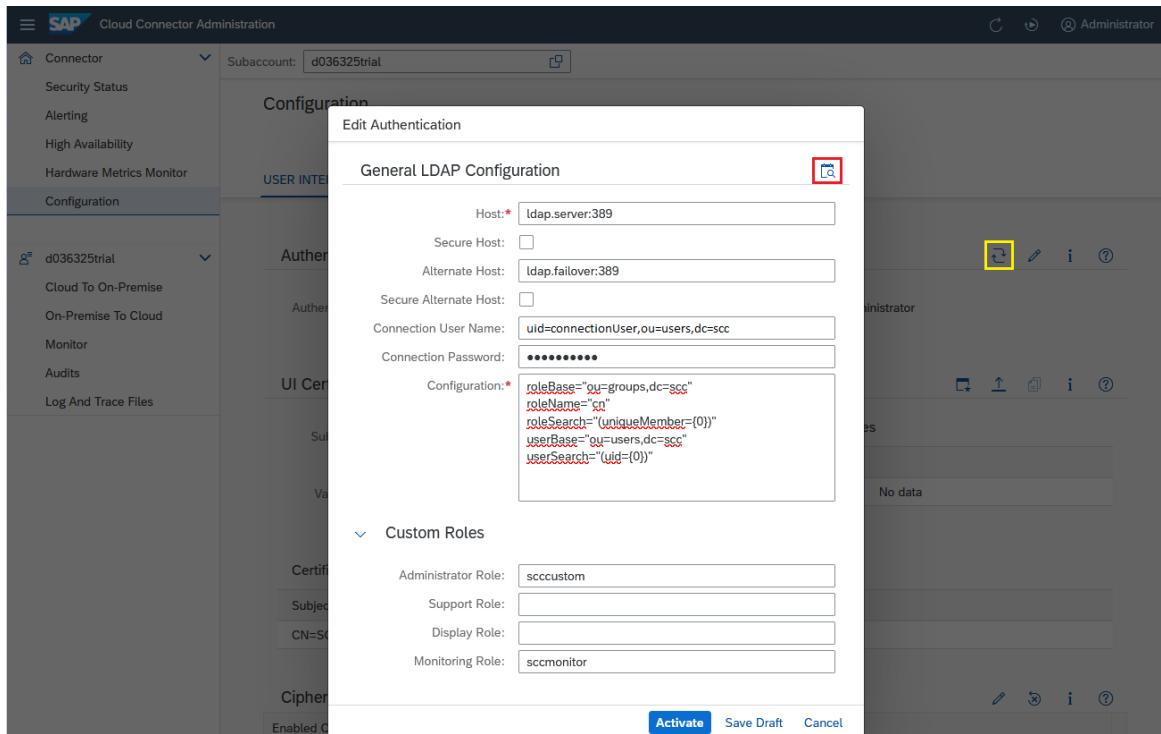
i Note

This role cannot be used to access the Cloud Connector administration UI.

Group membership is checked by the Cloud Connector.

Setting LDAP Authentication

- From the main menu, choose *Configuration* and go to the *User Interface* tab.
- From the *Authentication* section, choose *Switch to LDAP*.



- (Optional) To save intermediate adoptions of the LDAP configuration, choose *Save Draft*. This lets you store the changes in the Cloud Connector without activation.

- Usually, the LDAP server lists users in an LDAP node and user groups in another node. In this case, you can use the following template for LDAP configuration. Copy the template into the configuration text area:

```
roleBase="ou=groups,dc=scc"
roleName="cn"
roleSearch="(uniqueMember={0})"
userBase="ou=users,dc=scc"
userSearch="(uid={0})"
```

Change the `<ou>` and `<dc>` fields in `userBase` and `roleBase`, according to the configuration on your LDAP server, or use some other LDAP query.

i Note

The configuration depends on your specific LDAP server. For details, contact your LDAP administrator.

- Provide the LDAP server's host and port (port **389** is used by default) in the `<Host>` field. To use the secure protocol variant LDAPS based on TLS, select `Secure`.
- Provide a failover LDAP server's host and port (port **389** is used by default) in the `<Alternate Host>` field. To use the secure protocol variant LDAPS based on TLS, select `<Secure Alternate Host>`.
- (Optional) Depending on your LDAP server configuration you may need to specify the `<Connection User Name>` and its `<Connection Password>`. LDAP Servers supporting anonymous binding ignore these parameters.
- (Optional) To use your own role names, you can customize the default role names in the `Custom Roles` section. If no custom role is provided, the Cloud Connector checks permissions for the corresponding default role name:
 - `<Administrator Role>` (default: `sccadmin`)
 - `<Support Role>` (default: `sccsupport`)
 - `<Display Role>` (default: `sccdisplay`)
 - `<Monitoring Role>` (default: `sccmonitoring`)
- (Optional) Before activating the LDAP authentication, you can execute an authentication test by choosing the `Test LDAP Configuration` button. In the pop-up dialog, you must specify user name and password of a user who is allowed to logon after activating the configuration. The check verifies if authentication would be successful or not.

i Note

We strongly recommend that you perform an authentication test. If authentication should fail, login is not possible anymore. The test dialog also provides a test protocol, which could be helpful for troubleshooting.

For more information about how to set up LDAP authentication, see tomcat.apache.org/tomcat-8.5-doc/realm-howto.html.

i Note

To find a list of all supported attributes, see https://tomcat.apache.org/tomcat-8.5-doc/config/realm.html#JNDI_Directory_Realm_-_org.apache.catalina.realm.JNDIRealm.

You can also configure LDAP authentication on the shadow instance in a high availability setup (master and shadow). From the main menu of the shadow instance, select `Shadow Configuration`, go to tab `User Interface`, and check the `Authentication` section.

Note

If you are using LDAP together with a high availability setup, you cannot use the configuration option `userPattern`. Instead, use a combination of `userSearch`, `userSubtree` and `userBase`.

Caution

An LDAP connection over SSL/TLS can cause SSL errors if the LDAP server uses a certificate that is not signed by a trusted CA. If you cannot use a certificate signed by a trusted CA, you must set up the trust relationship manually, that is, import the public part of the issuer certificate to the JDK's trust storage.

Usually, the `cacerts` file inside the `java` directory (`jre/lib/security/cacerts`) is used for trust storage. To import the certificate, you can use `keytool`:

```
keytool -import -storepass changeit -file <certificate used by LDAP server> -keystore cacerts -alias <e.g. LDAP_xyz>
```

For more information, see also <https://docs.oracle.com/cd/E19830-01/819-4712/ablqw/index.html>



10. After finishing the configuration, choose *Activate*. Immediately after activating the LDAP configuration you must restart the Cloud Connector server, which invalidates the current browser session. Refresh the browser and logon to the Cloud Connector again, using the credentials configured at the LDAP server.
11. To switch back to file-based user management, choose the *Switch* icon again.

Note

If you have set up an LDAP configuration incorrectly, you may not be able to logon to the Cloud Connector again. In this case, adjust the Cloud Connector configuration to use the file-based user store again *without the administration UI*. For more information, see the next section.

Switching Back to File-Based User Store without the Administration UI

If your LDAP settings do not work as expected, you can use the `useFileUserStore` tool, provided with Cloud Connector version 2.8.0 and higher, to revert back to the file-based user store:

1. Change to the installation directory of the Cloud Connector and enter the following command:
 - **Microsoft Windows:** `useFileUserStore`
 - **Linux, Mac OS:** `./useFileUserStore.sh`
2. Restart the Cloud Connector to activate the file-based user store.

For versions older than 2.8.0, you must manually edit the configuration files.

Depending on your operating system, the configuration file is located at:

- **Microsoft Windows OS:** `<install_dir>\config_master\org.eclipse.gemini.web.tomcat\default-server.xml`
- **Linux OS:** `/opt/sap/scc/config_master/org.eclipse.gemini.web.tomcat/default-server.xml`

- **Mac OS X:** /opt/sap/scc/config_master/org.eclipse.gemini.web.tomcat/default-server.xml

1. Replace the Realm section with the following:

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
    <Realm className="org.apache.catalina.realm.CombinedRealm">
        <Realm
X509UsernameRetrieverClassName="com.sap.scc.tomcat.utils.SccX509SubjectDnRetriever"
        className="org.apache.catalina.realm.UserDatabaseRealm"
        digest="SHA-256" resourceName="UserDatabase"/>
        <Realm
X509UsernameRetrieverClassName="com.sap.scc.tomcat.utils.SccX509SubjectDnRetriever"
        className="org.apache.catalina.realm.UserDatabaseRealm" digest="SHA-1"
        resourceName="UserDatabase"/>
    </Realm>
</Realm>
```

2. Restart the Cloud Connector service:

- **Microsoft Windows OS:** Open the Windows [Services](#) console and restart the cloud connector service.
- **Linux OS:** Execute
 - System V init distributions: `service scc_daemon restart`
 - Systemd distributions: `systemctl restart scc_daemon`
- **Mac OS X:** Not applicable because no daemon exists (for Mac OS X, only a portable variant is available).

1.2.3.2 High Availability Setup

You can operate the Cloud Connector in a high availability mode, in which a master and a shadow instance are installed.

Task	Description
Install a Failover Instance for High Availability [page 518]	Install a redundant Cloud Connector instance (shadow) that monitors the main instance (master).
Master and Shadow Administration [page 521]	Learn how to operate master and shadow instances.

Related Information

[Connect DB Tools to SAP HANA via Service Channels \[page 496\]](#)

1.2.3.2.1 Install a Failover Instance for High Availability

The Cloud Connector lets you install a redundant instance that monitors the main instance.

Context

In a failover setup, when the main instance should go down for some reason, a redundant one can take over its role. The main instance of the Cloud Connector is called **master** and the redundant instance is called the **shadow**. The shadow has to be installed and connected to its master. During the setup of high availability, the master pushes the entire configuration to the shadow. Later on, during normal operation, the master also pushes configuration updates to the shadow. Thus, the shadow instance is kept synchronized with the master instance. The shadow pings the master regularly. If the master is not reachable for a while, the shadow tries to take over the master role and to establish the tunnel to SAP BTP.

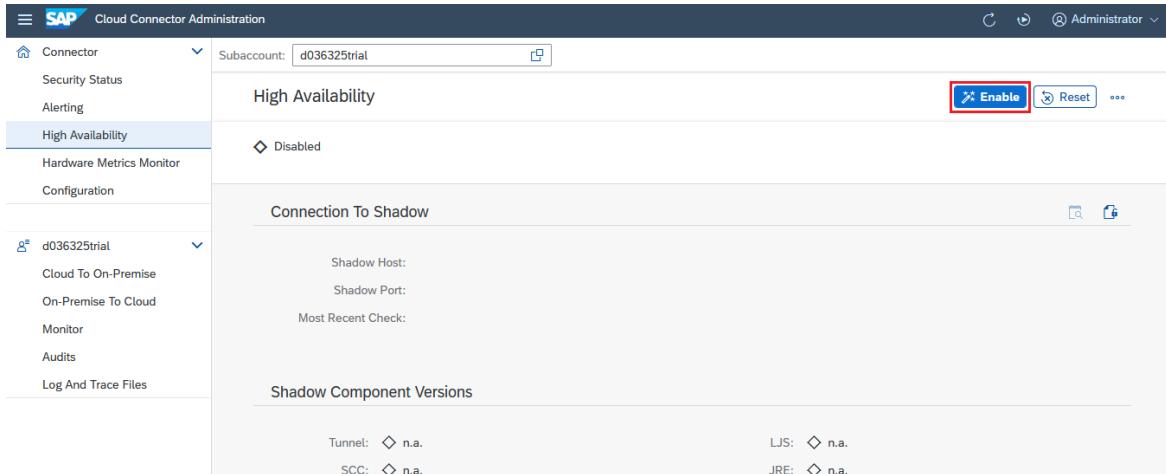
i Note

For detailed information about sizing of the master and the shadow instance, see also [Sizing Recommendations \[page 296\]](#).

Procedure

Preparing the Master Instance for High Availability

1. Open the Cloud Connector UI and go to the master instance.
2. From the main menu, choose *High Availability*.
3. Choose *Enable*.



The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has a tree view with 'Connector' expanded, showing 'Security Status', 'Alerting', 'High Availability' (which is selected), 'Hardware Metrics Monitor', and 'Configuration'. Under 'High Availability', there are sub-options: 'd036325trial' (selected), 'Cloud To On-Premise', 'On-Premise To Cloud', 'Monitor', 'Audits', and 'Log And Trace Files'. The main content area has a subaccount dropdown set to 'd036325trial'. The 'High Availability' section contains a 'Disabled' status indicator and an 'Enable' button, which is highlighted with a red box. Below it is the 'Connection To Shadow' section, which includes fields for 'Shadow Host' (empty), 'Shadow Port' (empty), and 'Most Recent Check' (empty). At the bottom is the 'Shadow Component Versions' section, showing 'Tunnel' and 'SCC' both set to 'n.a.'.

If this flag is not activated, no shadow instance can connect to this Cloud Connector. Additionally, when providing a concrete *Shadow Host*, you can ensure that only from this host a shadow instance can be connected.

⚠ Caution

Pressing the [Reset](#) button resets all high availability settings to their initial state. As a result, high availability is disabled and the shadow host is cleared. Reset only works if no shadow is connected.

Installing and Setting Up a Shadow Instance

Install the shadow instance in the same network segment as the master instance. Communication between master and shadow via proxy is not supported. The same distribution package is used for master and shadow instance.

i Note

If you plan to use LDAP for the user authentication on both master and shadow, make sure you configure it **before** you establish the connection from shadow to master.

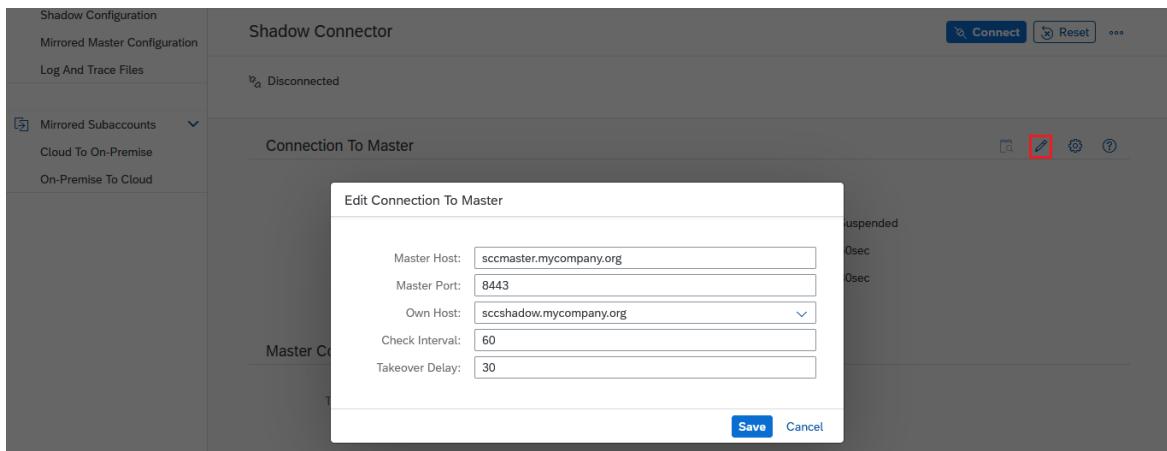
1. On first start-up of a Cloud Connector instance, a UI wizard asks you whether the current instance should be master or shadow. Choose [Shadow](#) and [Save](#):



Choose Installation Type

- [Master \(Primary Installation\)](#)
 [Shadow \(Backup Installation\)](#)

2. From the main menu, choose [Shadow Connector](#) and provide connection data for the master instance, that is, the master host and port. As of version 2.8.1.1, you can choose from the list of known host names, to use the host name under which the shadow host is visible to the master. You can specify a host name manually, if the one you want is not on the list. For the first connection, you must log on to the master instance, using the user name and password for the master instance. The master and shadow instances exchange X.509 certificates, which will be used for mutual authentication.



i Note

If you want to attach the shadow instance to a different master, press the [Reset](#) button. All your high availability settings will be removed, that is, reset to their initial state. This works only if the shadow is not connected.

3. Upon a successful connection, the master instance pushes the entire configuration plus some information about itself to the shadow instance. You can see this information in the UI of the shadow instance, but you can't modify it.
4. The UI on the master instance shows information about the connected shadow instance. From the main menu, choose [High Availability](#):

5. As of version **2.6.0**, the [High Availability](#) view includes an [Alert Messages](#) panel. It displays alerts if configuration changes have not been pushed successfully. This might happen, for example, if a temporary network failure occurs at the same time a configuration change is made. This panel lets an administrator know if there is an inconsistency in the configuration data between master and shadow that could cause trouble if the shadow needs to take over. Typically, the master recognizes this situation and tries to push the configuration change at a later time automatically. If this is successful, all failure alerts are removed and replaced by a warning alert showing that there had been trouble before. As of version **2.8.0.1**, these alerts have been integrated in the general [Alerting](#) section; there is no longer a separate [Alert Messages](#) panel.

If the master doesn't recover automatically, disconnect, then reconnect the shadow, which triggers a complete configuration transfer.

Related Information

[Initial Configuration \[page 322\]](#)

[Master and Shadow Administration \[page 521\]](#)

1.2.3.2.2 Master and Shadow Administration

Administration of Shadow Instances

There are several administration activities you can perform on the shadow instance. All configuration of tunnel connections, host mappings, access rules, and so on, must be maintained on the master instance; however, you can replicate them to the shadow instance for display purposes. You may want to modify the **check interval** (time between checks of whether the master is still alive) and the **takeover delay** (time the shadow waits to see whether the master would come back online, before taking over the master role itself).

As of Cloud Connector version 2.11.2, you can configure the timeout for the connection check, by pressing the gear icon in the section *Connection To Master* of the shadow connector main page.

Edit Timeout Settings

 Validate and/or change current timeout settings

Timeout Settings



Connection Timeout (in ms):

Request Timeout (in ms):

Timeout Validation



Number of requests:

Requests:

Connection Timeouts:

Errors:

Request Timeouts:

Save

Cancel

The `<Connection Timeout>` field specifies the maximum time allowed for establishing the technical connection to the master, the `<Request Timeout>` defines the maximum time allowed for executing the check over this connection. In the *Timeout Validation* section, you can check the current settings by pressing the *Execute* button. If the timeouts are hit, you might want to increase the settings accordingly.

Keep in mind the following points:

- The log level on master and shadow instances can be different.
- Configuration for check interval and takeover delay is maintained only on the shadow instance, and is transferred to the master for display purposes.
- Audit logs are only written on the master instance and are not transferred to the shadow. However, if the shadow becomes the master for some time, the audit log is potentially distributed over both master and shadow instances.

You can use the *Reset* button to drop all the configuration information on the shadow that is related to the master, but only if the shadow is not connected to the master.

Required Configuration on the Shadow Instance

Once connected to the master, the shadow instance receives the configuration from the master instance. Yet, there are some aspects you must configure on the shadow instance separately:

- **User administration** is configured separately on master and shadow instances. Generally, it is not required to have the same configuration on both instances. In most cases, however, it is suitable to configure master and shadow in the same way.
- The **UI certificate** is not shared. Each host can have its own certificate, so you must maintain the UI certificates on master and shadow. You can use the same certificate though.
- **SNC configuration:** If secure RFC communication or principal propagation for RFC calls is used, you must configure SNC on each instance separately.

Failover Process

The shadow instance regularly checks whether the master instance is still alive. If a check fails, the shadow instance first attempts to reestablish the connection to the master instance for the time period specified by the takeover delay parameter.

- If no connection becomes possible during the takeover delay time period, the shadow tries to take over the master role. At this point, it is still possible for the master to be alive and the trouble to be caused by a network issue between the shadow and master. The shadow instance next attempts to establish a tunnel to the given SAP BTP subaccount. If the original master is still alive (that is, its tunnel to the cloud subaccount is still active), this attempt is denied and the shadow instance remains in "shadow status", periodically pinging the master and trying to connect to the cloud, while the master is not yet reachable.
- If the takeover delay period has fully elapsed, and the shadow instance does make a connection, the cloud side opens a tunnel and the shadow instance takes over the role of the master. From this point, the shadow

instance shows the UI of the master instance and allows the usual operations of a master instance, for example, starting/stopping tunnels, modifying the configuration, and so on.

When the original master instance restarts, it first checks whether the registered shadow instance has taken over the master role. If it has, the master registers itself as a shadow instance on the former shadow (now master) instance. Thus, the two Cloud Connector installations, in fact, have switched their roles.

i Note

Only one shadow instance is supported. Any further shadow instances that attempt to connect are declined by the master instance.

The master considers a shadow as lost, if no check/ping is received from that shadow instance during a time interval that is equal to three times the check period. Only after this much time has elapsed can another shadow system register itself.

i Note

On the master, you can manually trigger failover by selecting the [Switch Roles](#) button. If the shadow is available, the switch is made as expected. Even if the shadow instance cannot be reached, the role switch of the master may still be enforced. Select [Switch Roles](#) only if you are absolutely certain it is the correct action to take for your current circumstances.

1.2.3.3 Change the UI Port

Context

By default, the Cloud Connector uses port **8443** for its administration UI. If this port is blocked by another process, or if you want to change it after the installation, you can use the `changeport` tool, provided with Cloud Connector version **2.6.0** and higher.

i Note

On Windows, you can also choose a different port during installation.

Procedure

1. Change to the installation directory of the Cloud Connector. To adjust the port and execute one of the following commands:
 - o Microsoft Windows OS:

```
changeport <desired_port>
```

- Linux OS, Mac OS X:

```
./changeport.sh <desired_port>
```

- When you see a message stating that the port has been successfully modified, restart the Cloud Connector to activate the new port.

1.2.3.4 Connect and Disconnect a Cloud Subaccount

The major principle for the connectivity established by the Cloud Connector is that the Cloud Connector administrator should have full control over the connection to the cloud, that is, deciding if and when the Cloud Connector should be connected to the cloud, the accounts to which it should be connected, and which on-premise systems and resources should be accessible to applications of the connected subaccount.

Using the administration UI, the Cloud Connector administrator can connect and disconnect the Cloud Connector to and from the configured cloud subaccount. Once disconnected, no communication is possible, either between the cloud subaccount and the Cloud Connector, or to the internal systems. The connection state can be verified and changed by the Cloud Connector administrator on the Subaccount Dashboard tab of the UI.

Status	Subaccount	Display Name	Location ID	Region	Actions
	trial	trial		Europe (Rot)	
	maasdemo	maasdemo		Europe (Rot)	

i Note

Once the Cloud Connector is freshly installed and connected to a cloud subaccount, none of the systems in the customer network are yet accessible to the applications of the related cloud subaccount. Accessible systems and resources must be configured explicitly in the Cloud Connector one by one, see [Configure Access Control \[page 379\]](#).

A Cloud Connector instance can be connected to multiple subaccounts in the cloud. This is useful especially if you need multiple subaccounts to structure your development or to stage your cloud landscape into development, test, and production. In this case, you can use a single Cloud Connector instance for multiple subaccounts. However, we recommend that you do not use subaccounts running in productive scenarios and subaccounts used for development or test purposes within the same Cloud Connector. You can add or delete a cloud account to or from a Cloud Connector using the [Add](#) and [Delete](#) buttons on the [Subaccount Dashboard](#) (see screenshot above).

Related Information

[Managing Subaccounts \[page 338\]](#)

1.2.3.5 Secure the Activation of Traffic Traces

For support purposes, you can trace HTTP and RFC network traffic that passes through the Cloud Connector.

Context

Traffic data may include business-critical information or security-sensitive data, such as user names, passwords, address data, credit card numbers, and so on. Thus, by activating the corresponding trace level, a Cloud Connector administrator might see data that he or she is not meant to. To prevent this behavior, implement the four-eyes principle, which is supported by the Cloud Connector release **1.3.2** and higher.

Once the four-eyes principle is applied, activating a trace level that dumps traffic data will require two separate users:

- An operating system user on the machine where the Cloud Connector is installed;
- An `Administrator` user of the Cloud Connector user interface.

By assigning these roles to two different people, you can ensure that both persons are needed to activate a traffic dump.

Four-Eyes Principle for Microsoft Windows OS

1. Create a file named `writeHexDump` in `<scc_install_dir>\scc_config`. The owner of this file must be a user other than the operating system user who runs the `cloud_connector` process.

i Note

Usually, this file owner is the user which is specified in the `Log On` tab in the properties of the `cloud_connector` service (in the Windows `Services` console). We recommend that you use a dedicated OS user for the `cloud_connector` service.

- Only the file owner should have write permission for the file.
 - The OS user who runs the `cloud_connector` process needs read-only permissions for this file.
 - Initially, the file should contain a line like `allowed=false`.
 - In the security properties of the file `scc_config.ini` (same directory), make sure that only the OS user who runs the `cloud_connector` process has write/modify permissions for this file. The most efficient way to do this is simply by removing all other users from the list.
2. Once you've created this file, the Cloud Connector refuses any attempt to activate the `Payload Trace` flag.

- To activate the payload trace, first the owner of `writeHexDump` must change the file content from `allowed=false` to `allowed=true`. Thereafter, the *Administrator* user can activate the payload trace from the Cloud Connector administration screens.

Four-Eyes Principle for Linux OS/Mac OS X

- Create a file named **writeHexDump** in `/usr/local/v1/base/cfg` (Cloud Connector 1.3.2) or `/opt/sap/scc/scc_config` (Cloud Connector 2.x). The owner of this file must be a user other than the `scctunnel1` user (that is, the operating system user under which the `cloud_connector` processes run) and not a member of the operating system user group `sccgroup`.
 - Only the file owner should have write permission for the file.
 - The `scctunnel1` user needs read-only permissions for this file.
 - Initially, the file should contain a line like `allowed=false`.
- Once you've created this file, the Cloud Connector refuses any attempt to set the trace level higher than Runtime (Cloud Connector 1.3.2) or to activate the *Payload Trace* flag (Cloud Connector 2.x).
- To set a higher trace level, which includes traffic Hex-dumps (Cloud Connector 1.3.2), or to activate the payload trace (Cloud Connector 2.x), the file owner mentioned above must first change the file content from `allowed=false` to `allowed=true`. Thereafter, the *Administrator* user can activate one of the higher trace levels (Cloud Connector 1.3.2) or the payload trace (Cloud Connector 2.x) from the Cloud Connector administration screens.

1.2.3.6 Monitoring

Learn how to monitor the Cloud Connector from the SAP BTP cockpit and from the Cloud Connector administration UI.

Checking the Operational State

The simplest way to verify whether a Cloud Connector is running is to try to access its administration UI. If you can open the UI in a Web browser, the `cloud_connector` process is running.

- On Microsoft Windows operating systems, the `cloud_connector` process is registered as a Windows service, which is configured to start automatically after a new Cloud Connector installation. If the Cloud Connector server is rebooted, the `cloud_connector` process should also auto-restart immediately. You can check the state with the following command:

```
sc query "SAP Cloud Connector"
```

The line state shows the state of the service.

- On Linux operating systems, the Cloud Connector is registered as a daemon process and restarts automatically each time the `cloud connector` process is down, for example, following a system restart. You can check the daemon state with the following command:

```
service scc_daemon status
```

To verify if a Cloud Connector is connected to a certain cloud subaccount, log on to the Cloud Connector administration UI and go to the [Subaccount Dashboard](#), where the connection state of the connected subaccounts is visible, as described in section [Connect and Disconnect a Cloud Subaccount \[page 524\]](#).

Monitoring from the Cockpit

The cockpit includes a [Connectivity](#) section, where users can check the status of the Cloud Connector(s) attached in the current subaccount, if any, as well as information about the Cloud Connector ID, version, used Java runtime, high availability setup (master and shadow instance), and so on (choose  [Connectivity](#)  [Cloud Connectors](#) ).

Access to this view is granted to:

- Neo environment: Users with a role containing the permission `readSCCTunnels`, for example, the predefined role `Cloud Connector Admin`.
- Cloud Foundry environment, feature set A: Users with a Cloud Foundry org role containing the permission `readSCCTunnels`, for example, the role `Org Manager`.

Note

As a prerequisite, a Cloud Foundry org must be available.

- Cloud Foundry environment, feature set B: Users with a role containing the permission `readSCCTunnels`, for example, the predefined role `Cloud Connector Administrator`.

Note

For more information on feature sets in the Cloud Foundry environment, see [Cloud Management Tools – Feature Set Overview](#).

SAP Cloud Platform Cockpit

Europe (Trial) / trial

trial - Cloud Connectors
Connected

Master Instance

Description:	my SCC
Connector Id:	B93D2570494911E4B
Connected since:	14.09.2015 12:18:22
Connected by:	here
Location Id:	here
Version:	2.9.0
Java Version:	1.7.0_55
High Availability:	active

Shadow Instance

Version:	2.9.0
Java Version:	1.7.0_55

Monitoring from the Cloud Connector Administration UI

The Cloud Connector offers various views for monitoring its activities and state.

You can check the overall state of the Cloud Connector through its [Hardware Metrics \[page 528\]](#), whereas subaccount-specific performance and usage data is available via [Subaccount-Specific Monitoring \[page 530\]](#). To provide external monitoring tools, you can use the [Monitoring APIs \[page 534\]](#).

Related Information

[Configure Solution Management Integration \[page 504\]](#)

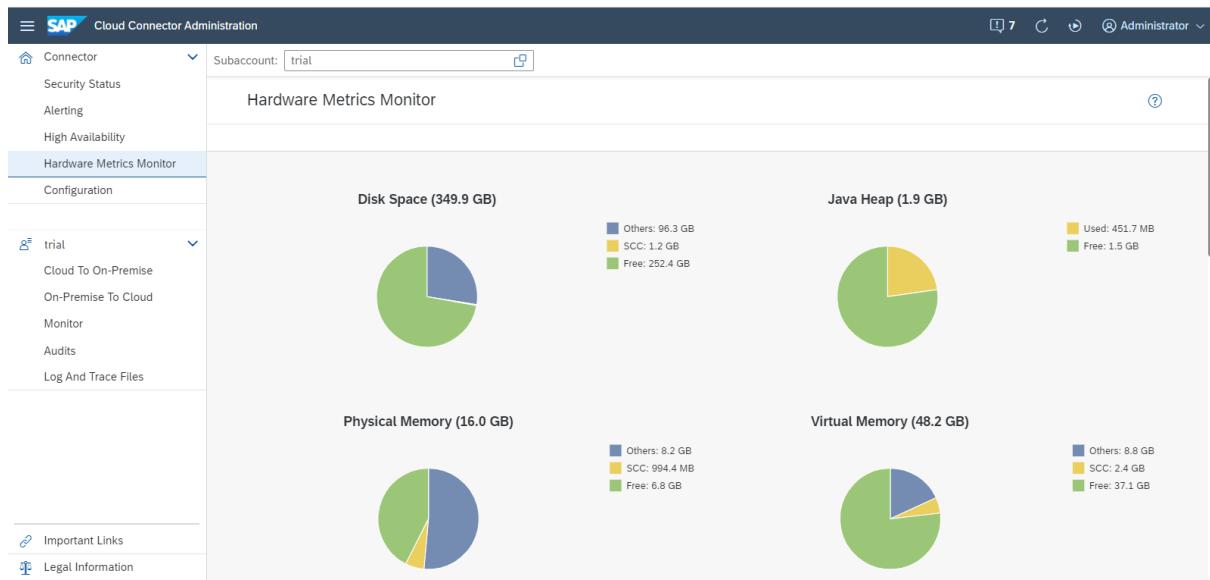
[High Availability Setup \[page 517\]](#)

1.2.3.6.1 Hardware Metrics

Check the current state of critical system resources in the Cloud Connector.

You can check the current state of critical system resources (disc space, Java heap, physical memory, virtual memory) using pie charts.

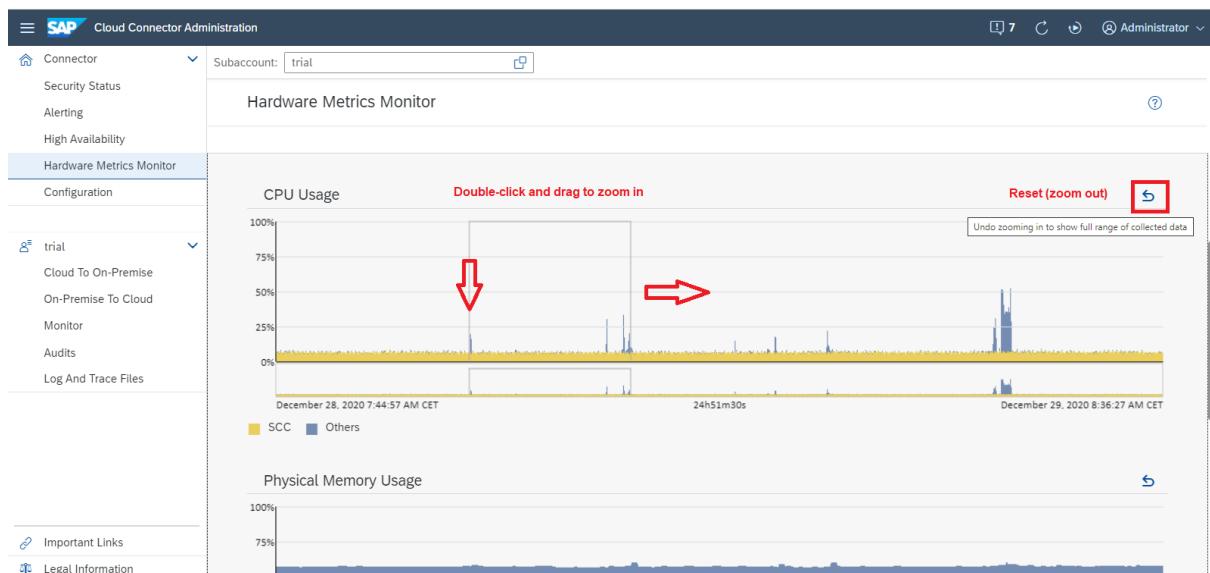
To access the monitor, choose [Hardware Metrics Monitor](#) from the main menu.



In addition, the history of CPU and memory usage (physical memory, Java heap) is shown in history graphs below the pie charts (recorded in intervals of 15 seconds).

You can view the usage data for a selected time period in each history graph:

- Double-click inside the main graph area to set the start (or end) point, and drag to the left or to the right to zoom in.
 - The entire timeline is always visible in the smaller bottom area right below the main graph.
 - A frame in the bottom area shows the position of the selected section in the overall timeline.
- Choose [Undo zooming in...](#) to reset the main graph area to the full range of available data.



1.2.3.6.2 Subaccount-Specific Monitoring

Use different monitoring views in the Cloud Connector administration UI to check subaccount-specific activities and data.

Content

[Performance Overview \[page 530\]](#)

[Most Recent Requests \[page 531\]](#)

[Resource Filter Settings \[page 532\]](#)

[Top Time Consumers \[page 533\]](#)

[Usage Statistics \[page 533\]](#)

[Backend Connections \[page 534\]](#)

Performance Overview

All requests that travel through the Cloud Connector to a backend system, as specified through access control, take a certain amount of time. You can check the duration of requests in a bar chart. The requests are not shown individually, but are assigned to buckets, each of which represents a time range.

For example, the first bucket contains all requests that took 10ms or less, the second one the requests that took longer than 10ms, but not longer than 20ms. The last bucket contains all requests that took longer than 5000ms.

In case of latency gaps, you may try to adjust the influencing parameters: number of connections, tunnel worker threads, and protocol processor worker threads. For more information, see [Configuration Setup \[page 300\]](#).

The collection of duration statistics starts as soon as the Cloud Connector is operational. You can delete all of these statistical records by selecting the button [Delete All](#). After that, the collection of duration statistics starts over.

i Note

[Delete All](#) deletes not only the list of most recent requests, but it also clears the top time consumers.

Back to [Content \[page 530\]](#)

Most Recent Requests

This option shows the most recent requests:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has sections for Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, and a trial subaccount with options for Cloud To On-Premise, On-Premise To Cloud, Monitor, Audits, and Log And Trace Files. The main area has tabs for PERFORMANCE, MOST RECENT REQUESTS (which is selected), TOP TIME CONSUMERS, USAGE STATISTICS, and BACK-END CONNECTIONS. Under MOST RECENT REQUESTS, there are 'Resource Filter Settings' and a table of requests. The table has columns for Date, Virtual Host, Resource, Protocol, Duration, and Actions. A specific row is highlighted with a red box. The 'Virtual Host' column shows 'sales-system.cloud:443' and the 'Resource' column shows '/PrincipalPropagation/PrincipalPropagationServlet/getUserInfo'. The 'Protocol' column shows 'RFC' and the 'Duration' column shows '68ms'.

The number of requests that are shown is limited to 50. You can either view all requests or only the ones destined for a certain virtual host, which you can select. You can select a row to see more detail.

The screenshot shows the 'Request Details' page for a selected request. It has tabs for Request Details, Basic Data, and Duration Breakdown (34ms). The Basic Data tab shows: Date: Dec 21, 2020 3:32:58 PM, Protocol: RFC, User: JCOTEST, Virtual Host: abapserver.hana.cloud:<port>, Bytes Sent: 304, Internal Host: <host>:<port>, and Bytes Received: 435. The Duration Breakdown chart is a horizontal stacked bar chart with segments for Latency Effects (8ms), Internal (SCC) (1ms), Open Connection (18ms), and External (Back-end) (7ms). A legend on the right identifies the colors: blue for External (Back-end), orange for Open Connection, green for Internal (SCC), red for SSO Handling, and purple for Latency Effects. A 'Close' button is at the bottom right.

A horizontal stacked bar chart breaks down the duration of the request into several parts: external (backend), open connection, internal (SCC), SSO handling, and latency effects. The numbers in each part represent milliseconds.

i Note

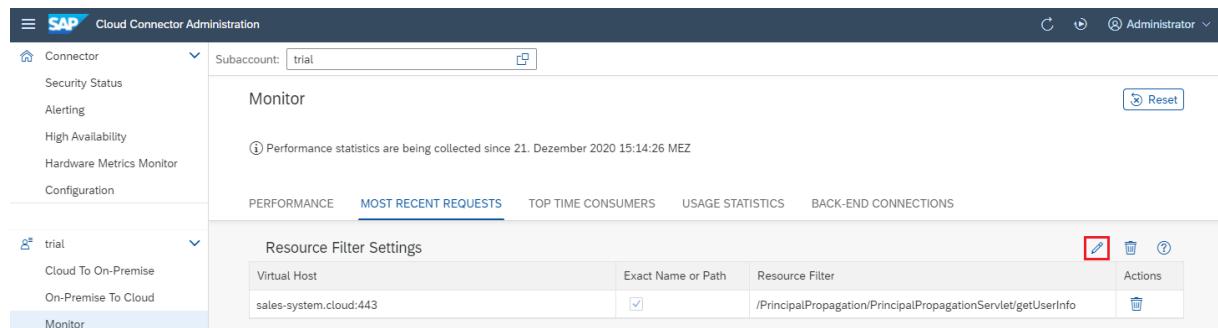
Parts with a duration of less than 1ms are not included.

In the above example, the selected request took 34ms, to which the Cloud Connector contributed 1ms. Opening a connection took 18ms. Backend processing consumed 7ms. Latency effects accounted for the remaining 8ms, while there was no SSO handling necessary and hence it took no time at all.

Back to [Content \[page 530\]](#)

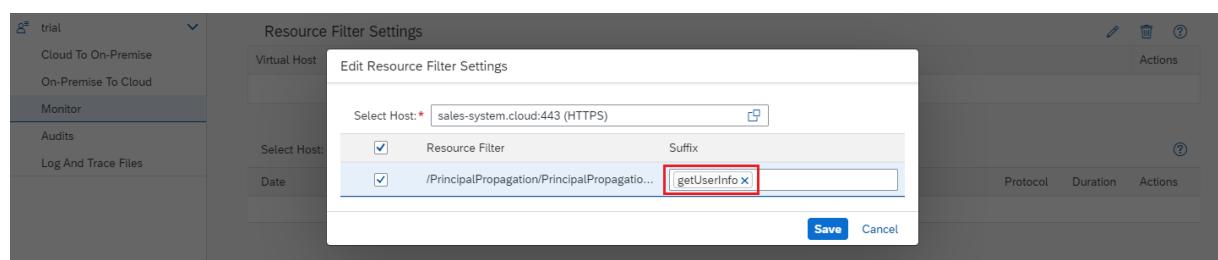
Resource Filter Settings

To further restrict the selection of the listed 50 most recent requests, you can edit the resource filter settings for each virtual host:



The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has sections for Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, and trial. Under trial, there are sub-options: Cloud To On-Premise, On-Premise To Cloud, and Monitor. The main area shows a subaccount 'trial' selected. A 'Monitor' section displays performance statistics. Below it is a tab bar with PERFORMANCE, MOST RECENT REQUESTS (which is selected), TOP TIME CONSUMERS, USAGE STATISTICS, and BACK-END CONNECTIONS. The MOST RECENT REQUESTS tab leads to a 'Resource Filter Settings' table. The table has columns: Virtual Host, Exact Name or Path, Resource Filter, and Actions. One row is shown: sales-system.cloud:443, checked, /PrincipalPropagation/PrincipalPropagationServlet/getUserInfo, and a delete icon.

In the [Edit](#) dialog, select the virtual host for which you want to specify the resource filter and choose one or more of the listed accessible resources. This list includes all resources that have been exposed during access control configuration (see also: [Configure Access Control \[page 379\]](#)). If the access policy for an accessible resource is set to `Path` and `all sub-paths`, you can further narrow the selection by adding one or more sub-paths to the resource as a suffix .



The screenshot shows the 'Edit Resource Filter Settings' dialog. It has a 'Select Host:' field with 'sales-system.cloud:443 (HTTPS)' selected. Below it are two checkboxes: 'Resource Filter' (checked) and 'Suffix' (checked). In the 'Suffix' field, the value '/PrincipalPropagation/PrincipalPropagationServlet/getUserInfo' is entered, with a red box highlighting the 'getUserInfo' part. At the bottom are 'Save' and 'Cancel' buttons.

Each selected resource/sub-path is listed separately in the resource filter list.

i Note

If you specify sub-paths for a resource, the request URL must match exactly one of these entries to be recorded. Without specified sub-paths (and the value `Path` and `all sub-paths` set for a resource), all sub-paths of a specified resource are recorded.

Back to [Content \[page 530\]](#)

Top Time Consumers

This option is similar to [Most Recent Requests](#); however, requests are not shown in order of appearance, but rather sorted by their duration (in descending order). Furthermore, you can delete top time consumers, which has no effect on most recent requests or the performance overview.

Back to [Content \[page 530\]](#)

Usage Statistics

To view the statistical data regarding the traffic handled by each **virtual host**, you can select a virtual host from the table. The detail view shows the traffic handled by each resource, as well as a *24 hour* overview of the throughput as a bar chart that aggregates the throughput (bytes received and bytes sent by a virtual host, respectively) on an hourly basis.

i Note

Currently, this feature is only available for the protocols HTTP(S) and RFC (SNC). Virtual hosts using other protocols are not listed.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar shows navigation links for Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, and a trial subaccount with options for Cloud To On-Premise, On-Premise To Cloud, Monitor, Audits, and Log And Trace Files. The main content area has tabs for PERFORMANCE, MOST RECENT REQUESTS, TOP TIME CONSUMERS, USAGE STATISTICS (which is selected), and BACK-END CONNECTIONS. The USAGE STATISTICS tab displays two tables: 'Virtual Systems Usage (1)' and 'Resources Usage Of (1)'. The first table shows a single entry for 'RFC' with 'Bytes Received' at 21.4 KB and 'Bytes Sent' at 12.8 KB. The second table shows a single entry for 'STFC_CONNECTION' with 'Bytes Received' at 6190 Bytes and 'Bytes Sent' at 4803 Bytes. Below the tables is a section titled '24H Throughput Overview For ldcibco.wdf.sap.corp:s...' with a date range from December 26, 2020 to December 27, 2020. A bar chart titled 'Bytes Received' shows throughput over time, with values ranging from 4k to 6k. The top right of the interface shows a toolbar with icons for search, refresh, and user information, and the text 'Administrator'.

The data that is collected includes the number of bytes received from cloud applications and the number of bytes sent back to cloud applications. The time of the most recent access is shown for the virtual hosts, and in the detail view also for the resources. If no access has taken place yet, the most recent access is shown as n.a. (not available). Similarly, the number of bytes received and sent of a virtual host is the sum of bytes received and sent of its resources.

The tables listing usage statistics of virtual hosts and their resources let you delete unused virtual hosts or unused resources. Use action [Delete](#) to delete such a virtual host or resource.

Caution

- Usage statistics are collected during runtime only and are not stored when stopping the Cloud Connector. That is, these statistics are lost when the Cloud Connector is stopped or restarted.
- Use care when taking the decision to delete a resource or virtual host based on its usage statistics.

For both virtual hosts and resources, you can use a classic filter button to reduce the virtual hosts or resources to those that have never been used (since the Cloud Connector started). For the virtual hosts, a second filter type is available that selects only those virtual hosts that have been used, but *include resources never used*. This feature facilitates locating obsolete resources of otherwise active virtual hosts.



Back to [Content \[page 530\]](#)

Backend Connections

This option shows a tabular overview of all active and idle connections, aggregated for each virtual host. By selecting a row (each of which represents a virtual host) you can view the details of all active connections as well as a graphical summary of all idle connections. The graphical summary is an accumulative view of connections based on the time the connections have been idle.

The maximum idle time appears on the rightmost side of the horizontal axis. For any point t on that axis (representing a time value ranging between 0ms and the maximal idle time), the ordinate is the number of connections that have been idle for no longer than t . You can click inside the graph area to view the respective abscissa t and ordinate.

Back to [Content \[page 530\]](#)

1.2.3.6.3 Monitoring APIs

Use the Cloud Connector monitoring APIs to include monitoring information in your own monitoring tool.

Context

You might want to integrate some monitoring information in the monitoring tool you use.

For this purpose, the Cloud Connector includes a collection of APIs that allow you to read various types of monitoring data.

i Note

This API set is designed particularly for monitoring the Cloud Connector via the SAP Solution Manager, see [Configure Solution Management Integration \[page 504\]](#).

Prerequisites

You must use *Basic Authentication* or *form field* authentication to read the monitoring data via API.

Users must be assigned to the roles `sccmonitoring` or `sccadmin`. The role `sccmonitoring` is restricted to managing the monitoring APIs.

i Note

The Health Check API does not require a specified user. Separate users are available through LDAP only.

URL Parameters

The API URLs adhere to the following pattern:

`https://<scchost>:<sccport>/xxx`

- Only HTTPS is supported
- `<scchost>:<sccport>` = cloud connector address
- `xxx` = path of the calling method

Available APIs

The following APIs are currently available.

- [Health Check \[page 536\]](#) (available as of version 2.6.0)
- [Subaccount data \[page 536\]](#) (as of 2.10.0)
- [Connection data \[page 538\]](#) (as of 2.10.0)
- [Performance data \[page 540\]](#) (as of 2.10.0)
- [Top Time Consumers \[page 542\]](#) (as of 2.11.0)
- [Memory Status \[page 544\]](#) (as of 2.13.0)
- [Certificate Status \[page 546\]](#) (as of 2.13.0)

- Usage Statistics [page 548] (as of 2.13.0)

Health Check (available as of version 2.6.0)

i Note

This API is relevant for the master instance as well as for the shadow instance.

Using the health check API, it is possible to recognize that the Cloud Connector is up and running. The purpose of this health check is only to verify that the Cloud Connector is not down. It does not check any internal state or tunnel connection states. Thus, it is a quick check that you can execute frequently:

URL	<code>https://<scc_host>:<scc_port>/exposed?action=ping</code>
-----	--

Expected Return Code	200
----------------------	-----

Back to [Available APIs \[page 535\]](#)

Back to [Context \[page 534\]](#)

List of Subaccounts (available as of version 2.10.0)

i Note

This API is relevant for the master instance only.

Using this API, you can read the list of all subaccounts connected to the Cloud Connector and view detail information for each subaccount:

URL	<code>https://<scchost>:<sccport>/api/monitoring/ subaccounts</code>
-----	--

Input	None
-------	------

Output

JSON document with list of detailed subaccount data, like:

- subaccounts: array of subaccounts for which the data is provided
 - regionHost: host of the region, in which the subaccount is residing
 - subaccount: name of subaccount
 - locationID: identifying the location of this Cloud Connector for a specific subaccount
 - tunnel: array of connection tunnels used by the subaccount
 - state: connected or disconnected
 - connectedSince: connection start time
 - connections: number of subaccount connections
 - applicationConnections: array of connections to application instances
 - serviceChannels: type and state of the service channels used (types: HANA database, Virtual Machine or RFC)
 - recoveryAccountState: state and more details about the disaster recovery subaccount, if configured
 - isActive: disaster recovery subaccount is working (true or false)
 - version: API version
-

Example:

```
▼ {
  ▼ "subaccounts": [
    ▼ {
      ▼ "tunnel": {
        "state": "Connected",
        "connectedSince": "2017-05-11T15:51:14.915 +0200",
        "connections": 0,
        "applicationConnections": [],
        ▼ "serviceChannels": [
          ▼ {
            "type": "VirtualMachine",
            "state": "ConnectFailure",
            "details": "dexlab"
          }
        ]
      },
      "regionHost": "hana.ondemand.com",
      "subaccount": "a117",
      "locationID": ""
    },
    ▼ {
      ▼ "recoveryAccountState": {
        "isActive": false,
        ▼ "tunnel": {
          "state": "Connected",
          "connectedSince": "2017-05-12T06:58:26.613 +0200",
          "connections": 0,
          "applicationConnections": [],
          "serviceChannels": []
        },
        "regionHost": "sapbydesign.com",
        "subaccount": "dev",
        "locationID": ""
      }
    }
  ]
}
```

Back to [Available APIs](#) [page 535]

Back to [Context](#) [page 534]

List of Open Connections (available as of version 2.10.0)

i Note

This API is relevant for the master instance only.

The list of connections lets you view all back-end systems connected to the Cloud Connector and get detail information for each connection:

URL	<code>https://<scchost>:<sccport>/api/monitoring/connections/backends</code>
Input	None
Output	<p>JSON document with list of all open connections and detailed information about back-end systems:</p> <ul style="list-style-type: none">• <code>subaccounts</code>: array of subaccounts for which the data is provided<ul style="list-style-type: none">◦ <code>backendConnections</code>: array of connections to a specified back-end system<ul style="list-style-type: none">◦ <code>virtualBackend</code>: virtual (external) back-end URL◦ <code>internalBackend</code>: internal back-end URL◦ <code>protocol</code>: type of protocol (RFC, HTTP, and so on)◦ <code>idle</code>: number of idle connections◦ <code>active</code>: number of active connections◦ <code>state</code>: connected or disconnected• <code>version</code>: API version

Example:

```
  "subaccounts": [
    {
      "backendConnections": [],
      "regionHost": "hana.ondemand.com",
      "subaccount": "a117",
      "locationID": ""
    },
    {
      "backendConnections": [
        {
          "virtualBackend": "abapserver.hana.cloud:",
          "internalBackend": "sap.corp:",
          "protocol": "RFC",
          "idle": 1,
          "active": 0
        }
      ],
      "regionHost": "hana.ondemand.com",
      "subaccount": "dev",
      "locationID": ""
    },
    {
      "backendConnections": [],
      "regionHost": "hanatrial.ondemand.com",
      "subaccount": "trial",
      "locationID": ""
    }
  ],
  "version": 1
}
```

Back to [Available APIs](#) [page 535]

Back to [Context](#) [page 534]

Performance Monitor Data (available as of version 2.10.0)

i Note

This API is relevant for the master instance only.

Using this API, you can read the data provided by the Cloud Connector performance monitor:

URL	<a href="https://<scchost>:<sccport>/api/monitoring/performance/backends">https://<scchost>:<sccport>/api/monitoring/ performance/backends
-----	--

Input	None
Output	<p>JSON document with a list providing the Cloud Connector performance monitor data with detailed information about back-end performance:</p> <ul style="list-style-type: none">● <code>subaccounts</code>: array of subaccounts for which data is provided<ul style="list-style-type: none">○ <code>VirtualHost</code>: host name of the back-end system○ <code>VirtualPort</code>: port of the back-end system○ <code>Protocol</code>: type of protocol (RFC, HTTP, and so on)○ <code>Buckets</code>: array of performance data related to back-end system<ul style="list-style-type: none">○ <code>numberOfCalls</code>: number of calls performed between Cloud Connector and back-end system○ <code>minimumCallDurationsMs</code>: minimum duration of the executed calls in milliseconds○ <code>SinceTime</code>: start of performance measurement● <code>version</code>: API version

Example:

```
▼ {
  ▼ "subaccounts": [
    ▼ {
      "backendPerformance": [],
      "sinceTime": "2017-05-11T15:48:42.084 +0200",
      "regionHost": "hana.ondemand.com",
      "subaccount": "a117",
      "locationID": ""
    },
    ▼ {
      "backendPerformance": [
        ▼ {
          "virtualHost": "abapserver.hana.cloud",
          "virtualPort": "2",
          "protocol": "RFC",
          "buckets": [
            ▼ {
              "numberOfCalls": 0,
              "minimumCallDurationMs": 0
            },
            ▼ {
              "numberOfCalls": 1,
              "minimumCallDurationMs": 10
            },
            ▼ {
              "numberOfCalls": 0,
              "minimumCallDurationMs": 20
            },
            ▼ {
              "numberOfCalls": 0,
              "minimumCallDurationMs": 30
            },
            ...
          ]
        }
      ]
    }
  ]
}
```

[Back to Available APIs \[page 535\]](#)

[Back to Context \[page 534\]](#)

Top Time Consumers (available as of version 2.11.0)

Note

This API is relevant for the master instance only.

Using this API, you can read the data of top time consumers provided by the Cloud Connector performance monitor:

URL	<code>https://<scchost>:<sccport>/api/monitoring/performance/toptimeconsumers</code>
Input	None
Output	<p>JSON document with list of top time consumer data:</p> <ul style="list-style-type: none"> • <code>subaccounts</code>: array of subaccounts for which the data is provided <ul style="list-style-type: none"> ◦ <code>requests</code>: array of requests sent to a specified backend system, including performance data ◦ <code>protocol</code>: type of protocol (RFC, HTTP, and so on) ◦ <code>virtualBackend</code>: virtual (external) back-end URL ◦ <code>internalBackend</code>: internal back-end URL ◦ <code>resource</code>: name of the request resource ◦ <code>sentBytes</code>: number of sent bytes ◦ <code>receivedBytes</code>: number of received bytes ◦ <code>user</code>: name of the request user ◦ <code>totalTime</code>: total request time in milliseconds ◦ <code>externalTime</code>: in milliseconds ◦ <code>genSsoTime</code>: in milliseconds ◦ <code>openRemoteTime</code>: in milliseconds ◦ <code>validationSsoTime</code>: time for SSO validation in milliseconds ◦ <code>latencyTime</code>: latency in milliseconds • <code>version</code>: API version

Example:

```

    {
      "subaccounts": [
        {
          "sinceTime": "2017-06-26T16:57:21.615 +0200",
          "requests": [
            {
              "startTime": "2017-06-26T16:57:25.426 +0200",
              "id": 1639964397,
              "protocol": "RFC",
              "virtualBackend": "abapserver.hana.cloud:sapgw42",
              "internalBackend": "ldciv9u:sapgw51",
              "resource": "RFCPING",
              "sentBytes": 307,
              "receivedBytes": 441,
              "user": "JCOUSER",
              "totalTime": 21,
              "externalTime": 6,
              "genSsoTime": 0,
              "openRemoteTime": 5,
              "validateSsoTime": 0,
              "latencyTime": 6
            },
            // 15 items
          ],
          "regionHost": "int.sap.hana.ondemand.com",
          "subaccount": "d039407sapdev",
          "locationID": "location"
        }
      ],
      "version": 1
    }
  
```

[Back to Available APIs \[page 535\]](#)

[Back to Context \[page 534\]](#)

Memory Status (available as of version 2.13.0)

i Note

This API is relevant for the master instance only.

This API provides a snapshot of the current memory status of the machine where the Cloud Connector is running:

URL	<code>https://<scchost>:<sccport>/api/monitoring/memory</code>
Input	None

Output

JSON document with memory data:

- `physicalKB`: usage of the physical memory, split into four categories (all sizes in KB):
 - `total`: total size of the physical memory
 - `CloudConnector`: size of the physical memory used by the Cloud Connector
 - `others`: size of the physical memory used by all other processes
 - `free`: size of the free physical memory
- `virtualKB`: usage of the virtual memory, split into four categories (all sizes in KB):
 - `total`: total size of the virtual memory
 - `CloudConnector`: size of the virtual memory used by the Cloud Connector
 - `others`: size of the virtual memory used by all other processes
 - `free`: size of the free virtual memory
- `cloudConnectorHeapKB`: usage of the Java heap, split into three categories (all sizes in KB):
 - `total`: total size of the Java heap
 - `used`: size of the Java heap used by the Cloud Connector
 - `free`: size of the free Java heap

Example:

```
{  
  "physicalKB": {  
    "total": 33380516,  
    "CloudConnector": 633932,  
    "others": 10960464,  
    "free": 21786120  
  },  
  "virtualKB": {  
    "total": 35477668,  
    "CloudConnector": 1267504,  
    "others": 14174208,  
    "free": 20035956  
  },  
  "cloudConnectorHeapKB": {  
    "total": 983040,  
    "used": 235457,  
    "free": 747583  
  }  
}
```

Back to [Available APIs \[page 535\]](#)

Back to [Context \[page 534\]](#)

Certificate Status (available as of version 2.13.0)

i Note

This API is relevant for the master instance only.

Using this API, you can get an overview of the certificates currently employed by the Cloud Connector:

URL	<code>https://<scchost>:<sccport>/api/monitoring/certificates</code>
-----	--

Input	None
-------	------

Output

JSON document with data reflecting the current status of all certificates. There are three categories or lists, each of them containing zero or more certificates:

- `expired`: list of all expired certificates
- `expiring`: list of all certificates that will expire in less than N days, where N is the number of days specified in the alerting setup regarding certificates that are close to their expiration date.
- `ok`: list of all certificates that continue to be valid for N days or more, where N is the number of days specified in the alerting setup regarding certificates that are close to their expiration date.

A certificate in any of those lists is represented by a JSON object with the following properties:

- `type`: type of the certificate which can be one of the following strings:
 - `UI` (for the UI certificate)
 - `System` (for the system certificate)
 - `CA` (for the certificate used in connection with principal propagation/certification authority)
 - `subaccount` (for subaccount certificates)
- `validTo`: end date of the respective certificate's validity (as a long integer, i.e. a UTC timestamp)
- `subjectDN`: subject DN of the respective certificate (included only for non-subaccount certificates)
- `subaccountId`: name of the subaccount (only for subaccount certificates)
- `subaccountRegion`: region or landscape host of the subaccount (only for subaccount certificates)
- `isDisasterRecoverySubaccount`: flag (Boolean value) that indicates that the subaccount is employed for disaster recovery. It can therefore be present only for subaccount certificates. Moreover, it is added only for disaster recovery subaccounts with its value set to true.

URL	<code>https://<scchost>:<sccport>/api/monitoring/certificates/expired</code>
Input	None
Output	JSON document (an array) holding the list of all expired certificates.
URL	<code>https://<scchost>:<sccport>/api/monitoring/certificates/expiring</code>
Input	None

Output	JSON document (an array) holding the list of all certificates that will expire in less than N days, where N is the number of days specified in the alerting setup regarding certificates that are close to their expiration date.
URL	<code>https://<scchost>:<sccport>/api/monitoring/certificates/ok</code>
Input	None
Output	JSON document (array) holding the list of all certificates that continue to be valid for N days or more, where N is the number of days specified in the alerting setup regarding certificates that are close to their expiration date.

Example:

```
{
  "expired": [
    {
      "type": "System",
      "subjectDN": "CN\u003dHugo, OU\u003dCSI, O\u003dSAP Trust Community, C\u003dDE",
      "validTo": 1458290342000
    }
  ],
  "expiring": [],
  "ok": [
    {
      "type": "UI",
      "subjectDN": "CN\u003dSCC, OU\u003dConnectivity, O\u003dSAP SE, C\u003dDE",
      "validTo": 1791826434000
    },
    {
      "type": "subaccount",
      "subaccountName": "d036325trial",
      "subaccountRegion": "hanatrial.ondemand.com",
      "validTo": 1613132865000
    }
  ]
}
```

Back to [Available APIs \[page 535\]](#)

Back to [Context \[page 534\]](#)

Usage Statistics (available as of version 2.13.0)

i Note

This API is relevant for the master instance only.

This API provides usage statistics regarding the systems and resources available in the Cloud Connector:

URL	<code>https://<scchost>:<sccport>/api/monitoring/ performance/toptimeconsumers</code>
-----	---

Input	None
-------	------

Output

JSON document (a JSON object) with usage data. There is one property:

- `subaccounts`: list of subaccounts (a JSON array)

Each subaccount in the subaccounts array is represented through an object with the following properties:

- `subaccountName`: name of the subaccount
- `subaccountRegion`: region host for the subaccount
- `sinceTime`: time at which collecting statistics started for this account (a UTC timestamp)
- `usageStatistics`: usage statistics, given as an array.

Each element of the `usageStatistics` array corresponds to one system (virtual host) and is represented through an object with the following properties:

- `virtualHost`: name of the virtual host
- `virtualPort`: port of the virtual host
- `protocol`: protocol this virtual host can process.

i Note

Currently, statistical data is collected only for protocols HTTP, HTTPS, RFC, and RFC SNC, and hence systems relying on other protocols are not listed here.

- `bytesReceived`: total number of bytes that were received through a call or request
- `bytesSent`: total number of bytes sent back as a response
- `calls`: total number of calls or requests
- `mostRecentAccess`: time of the most recent access (call or request) given as a UTC timestamp.

i Note

This property is only available if there has been at least one call or request.

- `resources`: usage statistics per resource, given as an array (i.e. the distribution of bytes received, sent, as well as number of calls/requests, across the resources of the respective virtual host).

Each element of the `resources` array holds the usage statistics of a resource, represented through an object with the following properties:

- `resourceName`: name of the resource (a URL path or the name of a remote function)

- `enabled`: Boolean flag that indicates whether the resource is currently active (true) or suspended (false).
- `bytesReceived`: total number of bytes that were received through a call or request and were handled by this resource.
- `bytesSent`: total number of bytes sent back as a response in the context of this resource.
- `calls`: total number of calls or requests handled by this resource
- `mostRecentAccess`: time of the most recent access (i.e., call or request) given as a UTC timestamp.

i Note

This property is only available if at least one call or request was handled by this resource.

i Note

Usage statistics are collected at runtime and stored in memory. They are lost when stopping or restarting the Cloud Connector.

Example:

```

    "subaccounts": [
      {
        "subaccountName": "lannister",
        "subaccountRegion": "westeros.com",
        "sinceTime": 1596810677689,
        "usageStatistics": [
          {
            "virtualHost": "iron.islands",
            "virtualPort": "1234",
            "protocol": "RFC",
            "bytesReceived": 0,
            "bytesSent": 0,
            "calls": 0,
            "resources": [
              {
                "resourceName": "TEST_FCT_1",
                "enabled": true,
                "bytesReceived": 0,
                "bytesSent": 0,
                "calls": 0
              },
              {
                "resourceName": "TEST_FCT_2",
                "enabled": true,
                "bytesReceived": 0,
                "bytesSent": 0,
                "calls": 0
              }
            ]
          },
          {
            "virtualHost": "the.north",
            "virtualPort": "8080",
            "protocol": "HTTP",
            "bytesReceived": 1482,
            "bytesSent": 705,
            "calls": 3,
            "mostRecentAccess": 1596810970078,
            "resources": [
              {
                "resourceName": "/invoke/giants",
                "enabled": true,
                "bytesReceived": 0,
                "bytesSent": 0,
                "calls": 0
              },
              {
                "resourceName": "/invoke/whiteWalkers",
                "enabled": true,
                "bytesReceived": 0,
                "bytesSent": 0,
                "calls": 0
              },
              {
                "resourceName": "/invoke/nightWatch",
                "enabled": true,
                "bytesReceived": 1482,
                "bytesSent": 705,
                "calls": 3,
                "mostRecentAccess": 1596810970078
              }
            ]
          }
        ]
      }
    ]
  }
}

```

[Back to Available APIs \[page 535\]](#)

[Back to Context \[page 534\]](#)

1.2.3.7 Alerting

Configure the Cloud Connector to send e-mail messages when situations occur that may prevent it from operating correctly.

To configure alert e-mails, choose *Alerting* from the top-left navigation menu.

You must specify the receivers of the alert e-mails (*E-mail Configuration*) as well as the Cloud Connector resources and components that you want to monitor (*Observation Configuration*). The corresponding *Alert Messages* are also shown in the Cloud Connector administration UI.

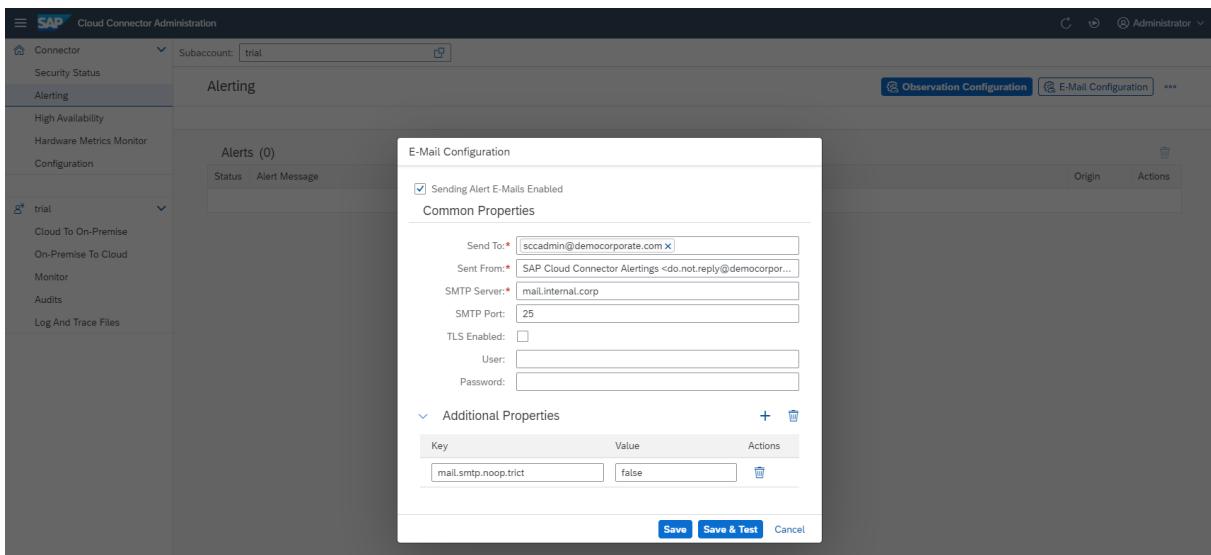
E-mail Configuration

1. Select *E-mail Configuration* to specify the list of em ail addresses to which alerts should be sent (*Send To*).

i Note

The addresses you enter here can use either of the following formats: john.doe@company.com or John Doe <j.doe@company.com>.

2. Enter the sender's e-mail address (*<Sent From>*).
3. In *<SMTP Server>* provide the host of the mail server.
4. You can specify an *<SMTP port>*, if the server is not using the default ports. For details, contact your e-mail administrator or provider.
5. Mark the *TLS Enabled* check box if you want to establish a TLS-encrypted connection.
6. If the SMTP server requires authentication, provide *<User>* and *<Password>*.
7. In the *Additional Properties* section you can provide any property supported by the *Java Mail library* ↗. All specified properties will be passed to the SMTP client.
8. Select *Save* to change the current configuration.



i Note

Connections to an SMTP server over SSL/TLS can cause SSL errors if the SMTP server uses an "untrusted" certificate. If you cannot use a trusted certificate, you must import the public part of the issuer certificate to the JDK's trust storage.

Usually, the trust storage is done in the file `cacerts` in the Java directory (`jre/lib/security/cacerts`). For import, you can use the `keytool` utility:

```
keytool -import -storepass changeit -file <certificate used by SMTP server> -keystore cacerts -alias <for example, SMTP_xyz>
```

For more information, see <https://docs.oracle.com/cd/E19830-01/819-4712/ablqw/index.html>.

Observation Configuration

Once you've entered the e-mail addresses to receive alerts, the next step is to identify the resources and components of the Cloud Connector: E-mail messages are sent when any of the chosen components or resources have malfunctioned or are in a critical state.

i Note

The Cloud Connector does not dispatch the same alert repeatedly. As soon as an issue has been resolved, an informational alert is generated, sent, and listed in [Alert Messages](#) (see section below).

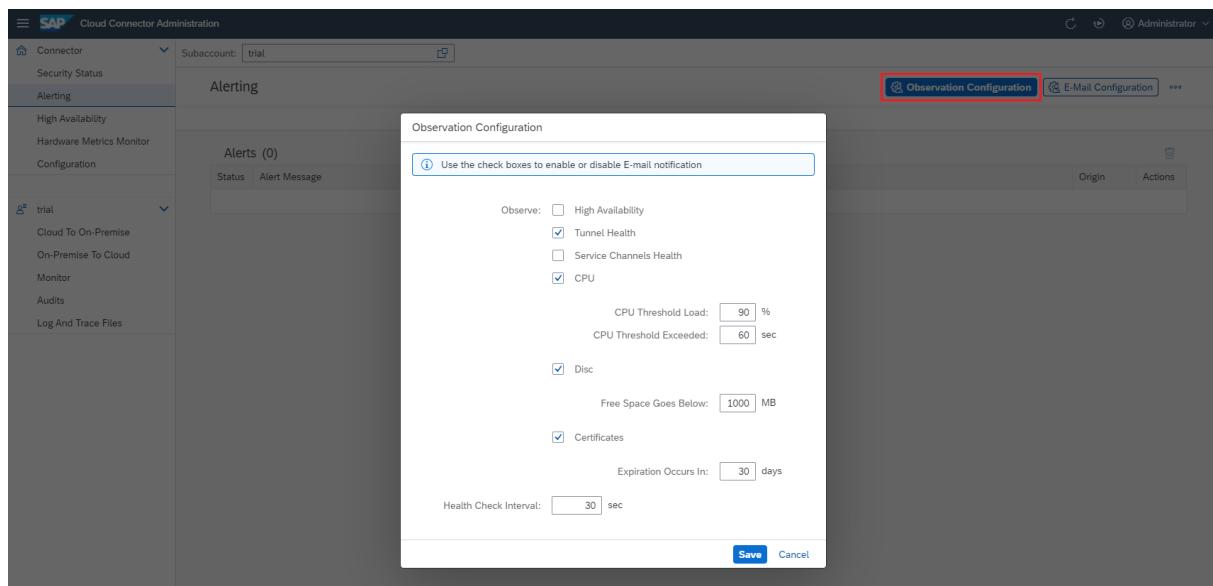
1. Select [Observation Configuration](#) from the top-right corner of the window.
2. Select the components or resources you want to monitor.
 - o [High Availability](#) alerts can occur in the context of an active high availability setup, meaning a shadow system is connected.

- *Tunnel Health* and *Service Channels Health* refer to the state of the respective connections. Whenever such a connection is lost, an alert is triggered.

i Note

These alerts are only triggered in case of an error or exception, but not upon intentional disconnect action.

- An excessively high *CPU* load over an extended period of time adversely affects performance and may be an indicator of serious issues that jeopardize the operability of the Cloud Connector. The CPU load is monitored and an alert is triggered whenever the CPU load exceeds and continues to exceed a given threshold percentage (the default is 90%) for more than a given period of time (the default is 60 seconds).
 - Although the Cloud Connector does not require nor consume large amounts of *Disk* space, running out of it is a circumstance that you should avoid. We recommend that you configure an alert to be sent if the disk space falls below a critical value (the default is 10 megabytes).
 - The Cloud Connector configuration contains various *Certificates*. Whenever one of those expires, scenarios might no longer work as expected so it's important to get notified about the expiration (the default is 30 days).
3. (Optional) Change the *Health Check Interval* (the default is 30 seconds).
 4. Select *Save* to change the current configuration.



Alert Messages

The Cloud Connector shows alert messages also on screen, in **Alerting > Alert Messages**.

You can remove alerts using *Delete* or *Delete All*. If you delete active (unresolved) alerts, they reappear in the list after the next health check interval.

1.2.3.8 Audit Logging

Audit log data can alert Cloud Connector administrators to unusual or suspicious network and system behavior.

Additionally, the audit log data can provide auditors with information required to validate security policy enforcement and proper segregation of duties. IT staff can use the audit log data for root-cause analysis following a security incident.

The Cloud Connector includes an auditor tool for viewing and managing audit log information about access between the cloud and the Cloud Connector, as well as for tracking of configuration changes done in the Cloud Connector. The written audit log files are digitally signed by the Cloud Connector so that their integrity can be checked, see [Manage Audit Logs \[page 556\]](#).

i Note

We recommend that you permanently switch on Cloud Connector audit logging in productive scenarios.

- Under normal circumstances, set the logging level to **Security** (the default configuration value).
- If legal requirements or company policies dictate it, set the logging level to **All**. This lets you use the log files to, for example, detect attacks of a malicious cloud application that tries to access on-premise services without permission, or in a forensic analysis of a security incident.

We also recommend that you regularly copy the audit log files of the Cloud Connector to an external persistent storage according to your local regulations. The audit log files can be found in the Cloud Connector root directory `/log/audit/<subaccount-name>/audit-log_<timestampl>.csv`.

1.2.3.8.1 Manage Audit Logs

Configure audit log settings and verify the integrity of audit logs.

Configure Audit Logs in the Cloud Connector

Choose [Audit](#) from your subaccount menu and go to [Settings](#) to specify the type of audit events the Cloud Connector should log at runtime. You can currently select between the following [Audit Levels](#) (for either `<subaccount>` and `<cross-subaccount>` scope):

- **Security:** Default value. The Cloud Connector writes an audit entry (`Access Denied`) for each request that was blocked. It also writes audit entries, whenever an administrator changes one of the critical configuration settings, such as exposed back-end systems, allowed resources, and so on.
- **All:** The Cloud Connector writes one audit entry for each received request, regardless of whether it was allowed to pass or not (`Access Allowed` and `Access Denied`). It also writes audit entries that are relevant to the **Security** mode.
- **Off:** No audit entries are written.

i Note

We recommend that you don't log all events unless you are required to do so by legal requirements or company policies. Generally, logging security events only is sufficient.

To enable automatic cleanup of audit log files, choose a period (14 to 365 days) from the list in the field `<Automatic Cleanup>`.

Audit entries for configuration changes are written for the following different categories:

- Account: A subaccount configuration was changed.
- RecoveryAccount: A disaster recovery subaccount configuration was changed.
- Configuration: A new subaccount was added or a disaster recovery switch happened.
- BackendMapping: Changes to the virtual to internal system mappings.
- AllowedResource: In a virtual system, changes in the accessible resources.
- DomainMapping: Changes to the domain mappings.
- ServiceChannelConfiguration: The configuration of a service channel was changed.
- SCCPassword: The Cloud Connector administration password was changed.
- SCCUser: The Cloud Connector administration user was changed.
- LDAPConfiguration: Changes to the LDAP settings.
- EmailConfiguration: The Email settings for alerts were changed.
- AlertConfiguration: The observation settings for alerts were changed.
- ScimConfiguration: Something changed in the settings for the cloud user store.
- KerberosConfiguration: The Kerberos configuration was changed.
- SNCSettings: SNC settings of Cloud Connector were changed.
- ProxySettings: The proxy settings were changed.
- SystemCertificate: The system certificate was changed.
- PpcaCertificate: The CA certificate was changed.
- PrincipalPropagationConfiguration: The principal propagation settings were changed.
- TrustSynchronization: The trust configuration for principal propagation was synchronized.
- IdentityProviderTrust: The trust configuration for a specific identity provider was changed.
- ApplicationTrust: The trust configuration to applications was changed.
- TrustedBackendCertificate: The trust store certificate was added or removed.
- SccCustomRoles: Custom role name settings were changed.
- BackendAuthority: RFC-specific user and client settings were adjusted.
- AdvancedConnectivity: Advanced connectivity configuration was changed.
- AdvancedJVM: Advanced JVM configuration was changed.
- ApplicationConfiguration: Application-specific connection configuration was changed.
- PayloadTrace: Payload trace (traffic data) was activated/deactivated.
- CPICTrace: The CPICTrace level was changed.
- AuditLogLevel: The subaccount-specific audit log level was changed.
- CrossAuditLogLevel: The cross-subaccount audit log level was changed.
- AuditLogCleanup: The audit log cleanup setting was changed.

In the [Audit Viewer](#) section, you can first define filter criteria, then display the selected audit entries.

- In the *Audit Type* field, you can select whether to view the audit entries for the following:
 - Any entries
 - Only denied requests
 - Only allowed requests
 - Service started
 - Service stopped
 - Cloud Connector changes
 - High Availability
 - Principal Propagation
- In the *Pattern* field, you can specify a certain string that the detail text of each selected audit entry must contain. The detail text contains information about the user name, requested resource/URL, and the virtual `<host>:<port>`. Wildcards are currently not supported. Use this feature to do the following:
 - Filter the audit log for all requests that a particular HTTP user has made during a certain time frame.
 - Identify all users who attempted to request a particular URL.
 - Identify all requests to a particular back-end system.
 - Determine whether a user has changed a certain Cloud Connector configuration. For example, a search for string **BackendMapping** returns all add-, delete- or modify- operations on the *Mapping Virtual To Internal System* page.
- The *Time Range* settings specify the time frame for which you want to display the audit events.

These filter criteria are combined with a logical AND so that all audit entries that match these criteria are shown. If you have modified one of the criteria, select *Refresh* to display the updated selection of audit events that match the new criteria.

Note

To prevent a single person from being able to both change the audit log level, and delete audit logs, we recommend that the operating system administrator and the SAP BTP administrator are different persons. We also suggest that you turn on the audit log at the operating system level for file operations.

The  *Check* button checks all files that are filtered by the specified date range.

Verify the Integrity of Audit Logs

To check the integrity of the audit logs, go to `<scc_installation>/auditor`. This directory contains an executable `go` script file (respectively, `go.cmd` on Microsoft Windows and `go.sh` on other operating systems).

If you start the `go` file without specifying parameters from `<scc_installation>/auditor`, all available audit logs for the current Cloud Connector installation are verified.

The auditor tool is a Java application, and therefore requires a Java runtime, specified in `JAVA_HOME`, to execute:

- For Microsoft Windows OS, set `JAVA_HOME=<path-to-java-installation>`
- For Linux OS and Mac OS X, export: `JAVA_HOME=<path-to-java-installation>`

Alternatively, to execute Java, you can include the Java `bin` directory in the `PATH` variable.

Example

In the following example, the [Audit Viewer](#) displays Any audit entries, at **Security** level, for the time frame between **December 18 2020, 00:00:00** and **December 19, 00:00:00**:

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has sections for Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, Subaccounts (with 'trial' selected), Cloud To On-Premise, On-Premise To Cloud, Monitor, Audits (selected), and Log And Trace Files. The main area is titled 'Audits' and shows settings for Subaccount Audit Level (Security), Cross-Subaccount Audit Level (Security), and Automatic Cleanup (Never). Below this is a table of audit logs:

Timestamp	Audit Log
2020-12-18 14:11:21 +0100	Audit Service is started for _crossaccount
2020-12-18 14:11:27 +0100	Audit Service is started for@hana.ondemand.com
2020-12-18 14:11:29 +0100	User during startup started Service Tunnel/account://.....

1.2.3.9 Troubleshooting

To troubleshoot connection problems, monitor the state of your open tunnel connections in the Cloud Connector, and view different types of logs and traces.

i Note

For information about a specific problem or an error you have encountered, see [Connectivity Support \[page 644\]](#).

Monitoring

To view a list of all currently connected applications, choose your *Subaccount* from the left menu and go to section [Cloud Connections](#):

The screenshot shows the SAP Cloud Connector Administration interface. On the left, there's a sidebar with navigation links like Connector, Security Status, Alerting, High Availability, Hardware Metrics Monitor, Configuration, and Log And Trace Files. The main area has a subaccount dropdown set to 'trial'. Below it, there's a 'Tunnel Information' section with status 'Connected', tunnel ID, and remote name. A red box highlights the 'Cloud Connections' section, which lists an application named 'jcodemo' with 1 connection, connected since 18. Dezember 2020 14:41:46 MEZ.

The provided information includes:

- **Application name:** The name of the application, as also shown in the cockpit, for your subaccount
- **Connections:** The number of currently existing connections to the application
- **Connected Since:** The earliest start time of a connection to this application
- **Peer Labels:** The name of the application processes, as also shown for this application in the cockpit, for your subaccount

Log and Trace Settings

The *Log and Trace Files* page includes some files for troubleshooting that are intended primarily for SAP Support. These files include information about both internal Cloud Connector operations and details about the communication between the local and the remote (SAP BTP) tunnel endpoint.

If you encounter problems that seem to be caused by some trouble in the communication between your cloud application and the on-premise system, choose *Log and Trace Files* from your *Subaccount* menu, go to section *Settings*, and activate the respective traces by selecting the *Edit* button:

- *Cloud Connector Loggers* adjusts the levels for Java loggers directly related to Cloud Connector functionality.
- *Other Loggers* adjusts the log level for all other Java loggers available at the runtime. Change this level only when requested to do so by SAP support. When set to a level higher than `Information`, it generates a large number of trace entries.
- *CPIC Trace Level* allows you to set the level between 0 and 3 and provides traces for the CPIC-based RFC communication with ABAP systems.
- When the *Payload Trace* is activated for a subaccount, all the HTTP and RFC traffic crossing the tunnel for that subaccount going through this Cloud Connector, is traced in files with names `traffic_trace_<subaccount_id>_on_<regionhost>.trc`.

i Note

Use payload and CPIC tracing at Level 3 carefully and only when requested to do so for support reasons. The trace may write sensitive information (such as payload data of HTTP/RFC requests and responses) to the trace files, and thus, present a potential security risk. As of version 2.2, the Cloud Connector supports an implementation of a "four-eyes principle" for activating the trace levels that

dump the network traffic into a trace file. This principle requires two users to activate a trace level that records traffic data. See [Secure the Activation of Traffic Traces \[page 525\]](#).

- **SSL Trace:** When the SSL trace is activated, the `1js_trace.log` file includes information for SSL-protected communication. To activate a change of this setting, a restart is required. Activate this trace only when requested by SAP support. It has a high impact on performance as it produces a large amount of traces.
- **Automatic Cleanup** lets you remove old trace files that have not been changed for a period of time exceeding the configured interval. You can choose from a list of predefined periods. The default is `Never`.

Edit Log Settings

Cloud Connector Loggers:	<input type="text" value="Information"/> ▼
Other Loggers:	<input type="text" value="Information"/> ▼
CPIC Trace Level:	<input type="text" value="0"/> ▼
Payload Trace:	<input type="checkbox"/>
SSL Trace:	<input type="checkbox"/>
Automatic Cleanup:	<input type="text" value="After 365 Days"/> ▼

Save [Cancel](#)

Log and Trace Files

View all existing trace files and delete the ones that are no longer needed.

The screenshot shows the SAP Cloud Connector Administration interface. The left sidebar has sections for Connector, trial (selected), and Log And Trace Files. The main area has a Subaccount dropdown set to 'trial'. Under 'Log And Trace Files', there's a 'Settings' tab with log levels and trace options. Below it is a table titled 'Log And Trace Files (4)' listing four files: 'localhost_http_access_2020-12-18.log', 'ljs_trace.log', 'vm_31400_gc.prf', and 'localhost_http_access_2020-12-17.log'. At the bottom of the table are buttons for 'All', '10', '12', and '175'.

To prevent your browser from being overloaded when multiple large files are loaded simultaneously, the Cloud Connector loads only one page into memory. Use the page buttons to move through the pages.

Use the [Download](#)/[Download All](#) icons to create a ZIP archive containing one trace file or all trace files. Download it to your local file system for convenient analysis.

i Note

If you want to download more than one file, but not all, select the respective rows of the table and choose [Download All](#).

When running the Cloud Connector with SAP JVM, it is possible to trigger the creation of a thread dump by pressing the [Thread Dump](#) button, which will be written to the JVM trace file `vm_${PID}_trace.log`. You will be requested by SAP support to create one if it is expected to help during incident analysis.

i Note

From the UI, you can't delete trace files that are currently in use. You can delete them from the Linux OS command line; however, we recommend that you do not use this option to avoid inconsistencies in the internal trace management of the Cloud Connector.

Two buttons may be helpful to solve issues on your own:

- [Guided Answers](#): A new tab or window opens, showing the Cloud Connector section in [Guided Answers](#). It helps you identify many issues that are classified through hierarchical topics. Once you found a matching issue, a solution is provided either directly, or by references to SAP Help Portal, Knowledge Base Articles (KBAs), and SAP notes.
- [Support Log Assistant](#): Opens the support log assistant. There, you can upload Cloud Connector log files and have them analyzed. After triggering the scan, the tool lists all issues for which a solution can be identified.

i Note

The support log assistant analyzes the complete log. Therefore, also older issues may be found that are no longer relevant.

Once a problem has been identified, you should turn off the trace again by editing the trace and log settings accordingly to not flood the files with unnecessary entries.

Use the [Refresh](#) button to update the information that appears. For example, you can use this button because more trace files might have been written since you last updated the display.

A screenshot of the SAP Cloud Connector Administration interface. The top navigation bar includes the SAP logo, the title "Cloud Connector Administration", and user information like "Administrator". The left sidebar has sections for "Connector", "Security Status", and "Alerting", with "Connector" currently selected. A dropdown menu shows "Subaccount: trial". The main content area is titled "Log And Trace Files" and contains buttons for "Thread Dump", "Guided Answers", "Support Log Assistant", and a help icon. There is also a refresh button in the top right corner of the content area.

Error Analysis and Support: Which Logs are Relevant?

If you contact SAP support for help, please always attach the appropriate log files and provide the timestamp or period, when the reported issue was observed. Depending on the situation, different logs may help to find the root cause.

Some typical settings to get the required data are listed below:

- <Cloud Connector Loggers> provide **details related to connections to SAP BTP and to backend systems as well as master-shadow communication in case of a high availability setup**. However, it does not contain any payload data. This kind of trace is written into `ljs_trace.log`, which is the most relevant log for the Cloud Connector.
- <Other Loggers> provide **details related to the tomcat runtime**, in which the Cloud Connector is running. The traces are written into `ljs_trace.log` as well, but they are needed only in very special support situations. If you don't need these traces, leave the level on `Information` or even lower.
- **Payload data** are written into the traffic trace file for HTTP or RFC requests if the payload trace is activated, or into the CPI-C trace file for RFC requests, if the CPI-C trace is set to level 3.
- <TLS trace> is helpful to **analyze TLS handshake failures** from Cloud Connector to Cloud or from Cloud Connector to backend. It should be turned off again as soon as the issue has been reproduced and recorded in the traces.
- Setting the audit log on level `ALL` for <Subaccount Audit Level> is the easiest way to **check if a request reached the the Cloud Connector and if it is being processed**.

Related Information

[Getting Support](#)

1.2.3.10 Process Guidelines for Hybrid Scenarios

A hybrid scenario is one, in which applications running on SAP BTP require access to on-premise systems. Define and document your scenario to get an overview of the required process steps.

Tasks

[Document the Landscape of a Hybrid Solution \[page 564\]](#)

[Document Administrator Roles \[page 565\]](#)

[Document Communication Channels \[page 565\]](#)

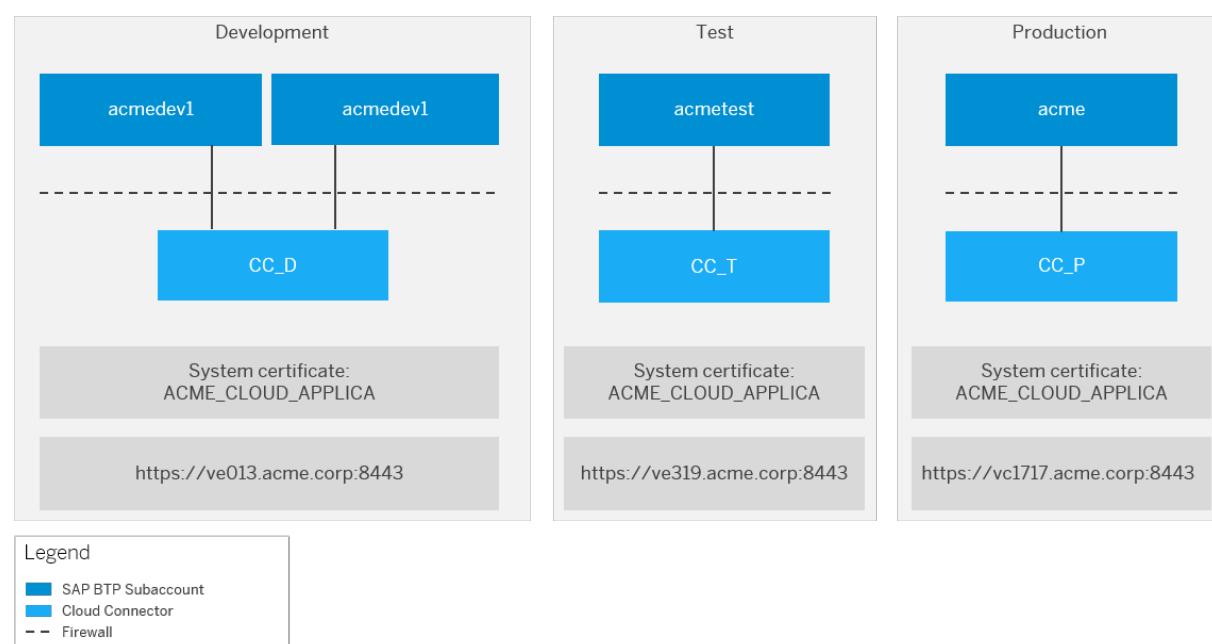
[Define Project and Development Guidelines \[page 566\]](#)

[Define How to Set a Cloud Application Live \[page 566\]](#)

Document the Landscape of a Hybrid Solution

To gain an overview of the cloud and on-premise landscape that is relevant for your hybrid scenario, we recommend that you diagrammatically document your cloud subaccounts, their connected Cloud Connectors and any on-premise back-end systems. Include the subaccount names, the purpose of the subaccounts (dev, test, prod), information about the Cloud Connector machines (host, domains), the URLs of the Cloud Connectors in the landscape overview document, and any other details you might find useful to include.

An example of landscape overview documentation could look like this:



Back to [Tasks \[page 564\]](#)

Document Administrator Roles

Document the users who have administrator access to the cloud subaccounts, to the Cloud Connector operating system, and to the Cloud Connector administration UI.

Such an administrator role documentation could look like following sample table:

Resource	john@acme.com	marry@acme.com	pete@acme.com	greg@acme.com
Cloud Subaccount (CA) Dev1	X			
CA Dev2		X		
CA Test			X	X
CA Prod				X
Cloud Connector Dev1 + Dev2	X	X		
Cloud Connector Test			X	X
Cloud Connector Prod				X
Cloud Connector Dev1 + Dev2 file system			X	X
Cloud Connector Test file system				X
Cloud Connector Prod file system				X

Back to [Tasks \[page 564\]](#)

Document Communication Channels

Create and document separate email distribution lists for both the cloud subaccount administrators and the Cloud Connector administrators.

An example of the documented communication channels could look like this:

Landscape	Distribution List
Cloud Subaccount Administrators	DL ACME HCP Subaccount Admins
Cloud Connector Administrators	DL ACME Cloud Connector Admins

[Back to Tasks \[page 564\]](#)

Define Project and Development Guidelines

Define and document mandatory project and development guidelines for your SAP BTP projects. An example of such a guideline could be similar to the following.

Every SAP BTP project in this organization requires the following:

- Use Maven, Nexus, Git-&-Gerrit for the application development.
- Align with accountable manager in projects (including the names).
- Align with accountable security officer in projects (including the names).
- For externally developed source code, an official handover to the organization.
- Fulfill connection restrictions in a three-system landscape, that is, use a staged landscape for dev, test and prod, and, for example, the dev landscape connects only to dev systems, and so on.
- Productive subaccounts cannot use the same Cloud Connector as a dev or test subaccount.

[Back to Tasks \[page 564\]](#)

Define How to Set a Cloud Application Live

Define and document how to set a cloud application live and how to configure needed connectivity for such an application.

For example, the following processes could be seen as relevant and should be defined and document in more detail:

1. Transferring application to production: Steps for transferring an application to the productive status on the SAP BTP.
2. Application connectivity: The steps for adding a connectivity destination to a deployed application for connections to other resources in the test or productive landscape.
3. Cloud Connector Connectivity: Steps for adding an on-premise resource to the Cloud Connector in the test or productive landscapes to make it available for the connected cloud subaccounts.
4. On-premise system connectivity: The steps for setting up a trusted relationship between an on-premise system and the Cloud Connector, and to configure user authentication and authorization in the on-premise system in the test or productive landscapes.
5. Application authorization: The steps for requesting and assigning an authorization that is available inside the SAP BTP application to a user in the test or productive landscapes.
6. Administrator permissions: Steps for requesting and assigning the administrator permissions in a cloud subaccount to a user in the test or productive landscape.

[Back to Tasks \[page 564\]](#)

1.2.4 Security

Learn how Cloud Connector features help you manage security.

Features

Security is a crucial concern for any cloud-based solution. It has a major impact on the business decision of enterprises whether to make use of such solutions. SAP BTP is a platform-as-a-service offering designed to run business-critical applications and processes for enterprises, with security considered on all levels of the on-demand platform:

Level	Features
Application Layer [page 568]	<ul style="list-style-type: none">Frontend securitySecurity standard-based application development
Service Layer [page 568]	<ul style="list-style-type: none">Identity and access managementData protectionRegulatory compliance management
Cloud Infrastructure Layer [page 573]	<ul style="list-style-type: none">Network infrastructure and communicationSandboxingIntrusion detection and prevention
Physical and Environmental Layer [page 574]	<ul style="list-style-type: none">Strict physical access controlHigh availability and disaster recovery capabilities

The Cloud Connector enables integration of cloud applications with services and systems running in customer networks, and supports database connections from the customer network to SAP HANA databases running on SAP BTP. As these are security-sensitive topics, this section gives an overview on how the Cloud Connector helps maintain security standards for the mentioned scenarios.

Target Audience

The content of this section concerns:

- System and IT administrators
- Technology consultants
- Solution architects

Related Information

[Security Guidelines \[page 574\]](#)

1.2.4.1 Application Layer

On application level, the main tasks to ensure secure Cloud Connector operations are to provide appropriate frontend security (for example, validation of entries) and a secure application development.

Product Security Standard

Basically, you should follow the rules given in the product security standard, for example, protection against cross-site scripting (XSS) and cross-site request forgery (XSRF).

Scope and Design

The scope and design of security measures on application level strongly depend on the specific needs of your application.

1.2.4.2 Service Layer

You can use SAP BTP Connectivity to securely integrate cloud applications with systems running in isolated customer networks.

Overview

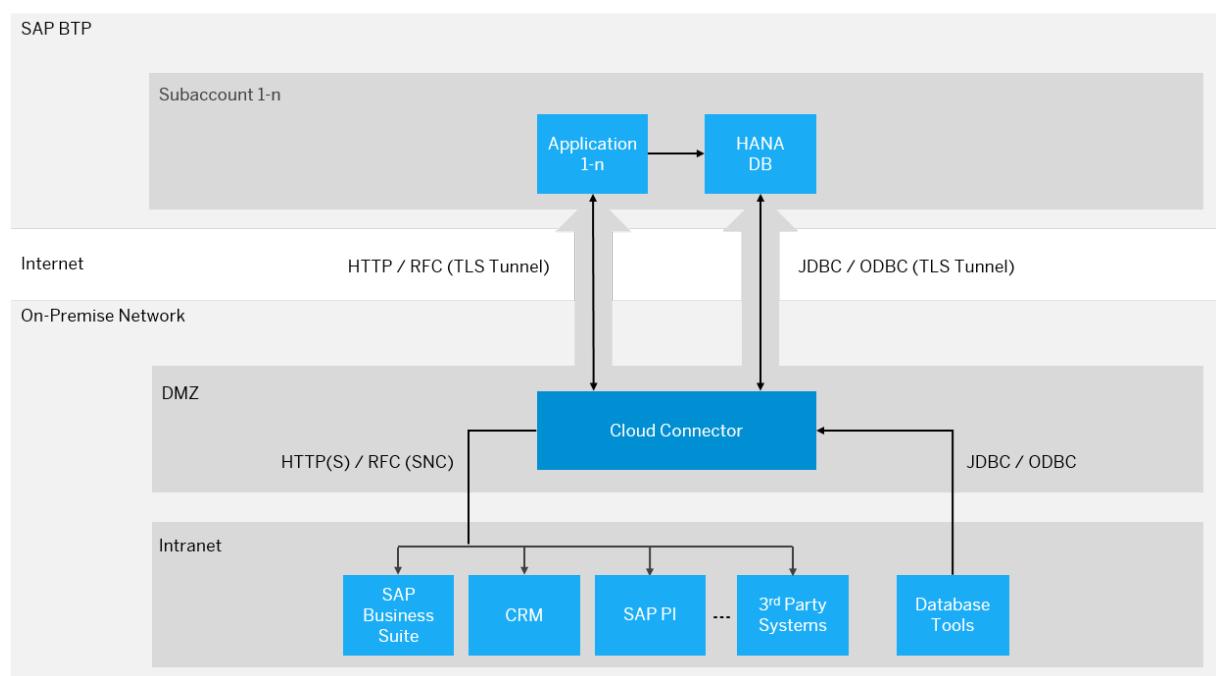
After installing the Cloud Connector as integration agent in your on-premise network, you can use it to establish a persistent TLS tunnel to SAP BTP subaccounts.

To establish this tunnel, the Cloud Connector administrator must authenticate himself or herself against the related SAP BTP subaccount of which he or she must be a member. Once established, the tunnel can be used by applications of the connected subaccount to remotely call systems in your network.

Architecture

The figure below shows a system landscape in which the Cloud Connector is used for secure connectivity between SAP BTP applications and on-premise systems.

- A single Cloud Connector instance can connect to multiple SAP BTP subaccounts, each connection requiring separate authentication and defining an own set of configuration.
- You can connect an arbitrary number of SAP and non-SAP systems to a single Cloud Connector instance.
- The on-premise system does not need to be touched when used with the Cloud Connector, unless you configure trust between the Cloud Connector and your on-premise system. A trust configuration is required, for example, for principal propagation (single sign-on), see [Configuring Principal Propagation \[page 350\]](#).
- You can operate the Cloud Connector in a high availability mode. To achieve this, you must install a second (redundant) Cloud Connector (shadow instance), which takes over from the master instance in case of a downtime.
- The Cloud Connector also supports the communication direction from the on-premise network to the SAP BTP subaccount, using a database tunnel that lets you connect common ODBC/JDBC database tools to SAP HANA as well as other available databases in SAP BTP.



Related Information

- [Network Zones \[page 570\]](#)
- [Inbound Connectivity \[page 570\]](#)
- [Outbound Connectivity \[page 572\]](#)

1.2.4.2.1 Network Zones

Choosing a network zone for the Cloud Connector installation.

A company network is usually divided into multiple network zones according to the security level of the contained systems. The DMZ network zone contains and exposes the external-facing services of an organization to an untrusted network, typically the Internet. Besides this, there can be one or multiple other network zones which contain the components and services provided in the company's intranet.

You can set up the Cloud Connector either in the DMZ or in an inner network zone. Technical prerequisites for the Cloud Connector to work properly are:

- The Cloud Connector must have access to the SAP BTP landscape host, either directly or via HTTPS proxy (see also: [Prerequisites \[page 283\]](#)).
- The Cloud Connector must have direct access to the internal systems it shall provide access to. I.e. there must be transparent connectivity between the Cloud Connector and the internal system.

It's a company's decision, whether the Cloud Connector is set up in the DMZ and operated centrally by an IT department or set up in the intranet and operated by the line of business.

Related Information

[Network Zones \[page 295\]](#)

1.2.4.2.2 Inbound Connectivity

For inbound connections into the on-premise network, the Cloud Connector acts as a reverse invoke proxy between SAP BTP and the internal systems.

Exposing Resources

Once installed, none of the internal systems are accessible by default through the Cloud Connector: you must configure explicitly each system and each service and resource on every system to be exposed to SAP BTP in the Cloud Connector.

You can also specify a virtual host name and port for a configured on-premise system, which is then used in the cloud. Doing this, you can avoid that information on physical hosts is exposed to the cloud.

TLS Tunnel

The TLS (Transport Layer Security) tunnel is established from the Cloud Connector to SAP BTP via a so-called **reverse invoke** approach. This lets an administrator have full control of the tunnel, since it can't be established from the cloud or from somewhere else outside the company network. The Cloud Connector administrator is the one who decides when the tunnel is established or closed.

The tunnel itself is using TLS with strong encryption of the communication, and mutual authentication of both communication sides, the client side (Cloud Connector) and the server side (SAP BTP).

The X.509 certificates which are used to authenticate the Cloud Connector and the SAP BTP subaccount are issued and controlled by SAP BTP. They are kept in secure storages in the Cloud Connector and in the cloud. Having encrypted and authenticated the tunnel, confidentiality and authenticity of the communication between the SAP BTP applications and the Cloud Connector is guaranteed.

Restricting Allowed Applications

As an additional level of control, the Cloud Connector optionally allows restricting the list of SAP BTP applications which are able to use the tunnel. This is useful in situations where multiple applications are deployed in a single SAP BTP subaccount while only particular applications require connectivity to on-premise systems.

Isolation on Subaccount Level

SAP BTP guarantees strict isolation on subaccount level provided by its infrastructure and platform layer. An application of one subaccount is not able to access and use resources of another subaccount.

Supported Protocols

The Cloud Connector supports inbound connectivity for HTTP and RFC, any other protocol is not supported.

- The payload sent via these protocols is encrypted on TLS/tunnel-level.
- For the route from the Cloud Connector to the on-premise systems, Cloud Connector administrators have the choice for each configured on-premise system whether to use HTTP, HTTPS, RFC or RFC over SNC.
- For HTTPS, you can configure a so-called system certificate in the Cloud Connector which is used for the trust relationship between the Cloud Connector and the connected on-premise systems.
- For RFC over SNC, you can configure an SNC PSE in the Cloud Connector respectively.

Principal Propagation

The Cloud Connector also supports principal propagation of the cloud user identity to connected on-premise systems (single sign-on). For this, the system certificate (in case of HTTPS) or the SNC PSE (in case of RFC) is

mandatory to be configured and trust with the respective on-premise system must be established. Trust configuration, in particular for principal propagation, is the only reason to configure and touch an on-premise system when using it with the Cloud Connector.

Related Information

[Configuring Principal Propagation \[page 350\]](#)

1.2.4.2.3 Outbound Connectivity

The Cloud Connector supports the communication direction from the on-premise network to SAP BTP, using a database tunnel.

The database tunnel is used to connect local database tools via JDBC or ODBC to the SAP HANA DB or other databases on SAP BTP, for example, SAP Business Objects tools like Lumira, BOE or Data Services.

- The database tunnel only allows JDBC and ODBC connections from the Cloud Connector into the cloud. A reuse for other protocols is not possible.
- The tunnel uses the same security mechanisms as for the inbound connectivity:
 - TLS-encryption and mutual authentication
 - Audit logging

To use the database tunnel, two different SAP BTP users are required:

- A platform user (member of the SAP BTP subaccount) establishes the database tunnel to the HANA DB.
- A HANA DB user is needed for the ODBC/JDBC connection to the database itself. For the HANA DB user, the role and privilege management of HANA can be used to control which actions he or she can perform on the database.

Related Information

[Using Service Channels \[page 492\]](#)

1.2.4.2.4 Audit Log

As audit logging is a critical element of an organization's risk management strategy, the Cloud Connector provides audit logging for the complete record of access between cloud and Cloud Connector as well as of configuration changes done in the Cloud Connector.

Integrity Check

The written audit log files are digitally signed by the Cloud Connector so that they can be checked for integrity (see also: [Manage Audit Logs \[page 556\]](#)).

Alerting

The audit log data of the Cloud Connector can be used to alert Cloud Connector administrators regarding unusual or suspicious network and system behavior.

Additional Use Cases

- The audit log data can provide auditors with information required to validate security policy enforcement and proper segregation of duties.
- IT staff can use the audit log data for root-cause analysis following a security incident.

Related Information

[Audit Logging \[page 556\]](#)

1.2.4.3 Cloud Infrastructure Layer

Infrastructure and network facilities of the SAP BTP ensure security on network layer by limiting access to authorized persons and specific business purposes.

Isolated Network

The SAP BTP landscape runs in an isolated network, which is protected from the outside by firewalls, DMZ, and communication proxies for all inbound and outbound communications to and from the network.

Sandboxed Environments

The SAP BTP infrastructure layer also ensures that platform services, like the SAP BTP Connectivity, and applications are running isolated, in sandboxed environments. An interaction between them is only possible over a secure remote communication channel.

1.2.4.4 Physical and Environmental Layer

Learn about data center security provided for SAP BTP Connectivity.

SAP BTP runs in SAP-hosted data centers which are compliant with regulatory requirements. The security measures include, for example:

- strict physical access control mechanisms using biometrics, video surveillance, and sensors
- high availability and disaster recoverability with redundant power supply and own power generation

1.2.4.5 Security Guidelines

Find a checklist of recommended security measures for the Cloud Connector.

Topics

Hover over the elements for a description. Click an element to find the recommended actions in the table below.

Network Zone	Administration UI	High Availability
On-Premise Configuration	OS-Level Protection	Protocol Security
Audit Logging	Instances	

- [#unique_167/unique_167_Connect_42_network \[page 575\]](#)
- [#unique_167/unique_167_Connect_42_ui \[page 575\]](#)
- [#unique_167/unique_167_Connect_42_availability \[page 577\]](#)

- [#unique_167/unique_167_Connect_42_protocols \[page 577\]](#)
- [#unique_167/unique_167_Connect_42_os \[page 575\]](#)
- [#unique_167/unique_167_Connect_42_oP \[page 577\]](#)
- [#unique_167/unique_167_Connect_42_instances \[page 578\]](#)
- [#unique_167/unique_167_Connect_42_audit \[page 576\]](#)

Recommended Actions

Topic	Description	Recommended Action
Network Zone Back to Topics [page 574]	<p>Depending on the needs of the project, the Cloud Connector can be either set up in the DMZ and operated centrally by the IT department or set up in the intranet and operated by the line-of-business.</p>	<p>To access highly secure on-premise systems, operate the Cloud Connector centrally by the IT department and install it in the DMZ of the company network.</p> <p>Set up trust between the on-premise system and the Cloud Connector, and only accept requests from trusted Cloud Connectors in the system.</p>
OS-Level Protection Back to Topics [page 574]	<p>The Cloud Connector is a security-critical component that handles the inbound access from SAP BTP applications to systems of an on-premise network.</p> <p>Methods to secure the operating system, on which the Cloud Connector is running, should be applied.</p>	<p>Restrict access to the operating system on which the Cloud Connector is installed to the minimal set of users who should administrate the Cloud Connector.</p> <p>Use the machine which runs the Cloud Connector only for this purpose and don't reuse it for other scenarios.</p>
		<p>Use hard-drive encryption for the machine that runs the Cloud Connector. This ensures that the Cloud Connector configuration data cannot be read or modified by unauthorized users, even if they obtain access to the hard drive.</p>
		<p>Turn on the audit log on operating system level to monitor the file operations.</p>
Administration UI Back to Topics [page 574]	<p>After installation, the Cloud Connector provides an initial user name and password and forces the user (Administrator) to change the password upon initial logon.</p>	<p>Change the password of the Administrator user immediately after installation. Choose a strong password for the user (see also Recommendations for Secure Setup [page 309]).</p>

Topic	Description	Recommended Action
		<p>Configure a corporate LDAP system for the user management of the Cloud Connector administrator users. This guarantees that users of the Cloud Connector administration UI are named users and can be traced via the Cloud Connector audit log (see Use LDAP for Authentication [page 513]).</p>
	<p>You can access the Cloud Connector administration UI remotely via HTTPS. After installation, it uses a self-signed X.509 certificate as SSL server certificate, which is not trusted by default by Web browsers.</p>	<p>Exchange the self-signed X.509 certificate of the Cloud Connector administration UI by a certificate that is trusted by your company and the company's approved Web browser settings (see [Deprecated] Replace the Default SSL Certificate [page 316]).</p>
		<p>For high-security scenarios, limit the access to the Cloud Connector administration UI to localhost (see also Recommendations for Secure Setup [page 309]).</p>
		<p>Use a JVM that allows to limit the ciphers to a set considered safe (see Recommendations for Secure Setup [page 309]). A list of cipher suites, which are currently considered safe, is available in Use Secure Cipher Suites.</p>
Audit Logging		
Back to Topics [page 574]		
	<p>For end-to-end traceability of configuration changes in the Cloud Connector, as well as communication delivered by the Cloud Connector, switch on audit logging for productive scenarios.</p>	<p>Switch on audit logging in the Cloud Connector: set audit level to "All" (see Recommendations for Secure Setup [page 309] and Manage Audit Logs [page 556]).</p>
		<p>Cloud Connector administrators must ensure that the audit log files are properly archived and are not lost, to conform to the local regulations.</p>
		<p>To gain end-to-end traceability, you should switch on audit logging also in the connected on-premise systems.</p>

Topic	Description	Recommended Action
High Availability	To guarantee high availability of the connectivity for cloud integration scenarios, run productive instances of the Cloud Connector in high availability mode, that is, with a second (redundant) Cloud Connector in place.	Use the high availability feature of the Cloud Connector for productive scenarios (see Install a Failover Instance for High Availability [page 518]).
Supported Protocols	<p>HTTP, HTTPS, RFC and RFC over SNC are currently supported as protocols for the communication direction from the cloud to on-premise.</p> <p>The route from the application VM in the cloud to the Cloud Connector is always encrypted.</p> <p>You can configure the route from the Cloud Connector to the on-premise system to be encrypted or unencrypted.</p>	The route from the Cloud Connector to the on-premise system should be encrypted using TLS (for HTTPS) or SNC (for RFC).
Configuration of On-Premise Systems	When configuring the access to an internal system in the Cloud Connector, map physical host names to virtual host names to prevent exposure of information on physical systems to the cloud.	Use hostname mapping of exposed on-premise systems in the access control of the Cloud Connector (see Configure Access Control (HTTP) [page 380] and Configure Access Control (RFC) [page 387]).
	When configuring the access to an internal system, restrict access to those resources which are actually required by the cloud applications. Do not expose the complete system.	Narrow access to on-premise systems to resources required by the relevant cloud applications in the access control of the Cloud Connector (see Configure Access Control (HTTP) [page 380] and Configure Access Control (RFC) [page 387]).
	To allow access only for trusted applications of your SAP BTP subaccount to on-premise systems, configure the list of trusted applications in the Cloud Connector.	Narrow the list of cloud applications which are allowed to use the on-premise tunnel to the ones that need on-premise connectivity (see Set Up Trust for Principal Propagation [page 351]).

Topic	Description	Recommended Action
Cloud Connector Instances Back to Topics [page 574]	<p>You can connect a single Cloud Connector instance to multiple SAP BTP subaccounts.</p> <p>Subaccounts can be created by SAP BTP users as a self service. Different subaccounts are often used to separate development, test and production.</p> <p>Do not mix productive Cloud Connector usages with development or test scenarios.</p>	Use different Cloud Connector instances to separate productive and non-productive scenarios.

Related Information

[Recommendations for Secure Setup \[page 309\]](#)

[Secure the Activation of Traffic Traces \[page 525\]](#)

1.2.5 Upgrade

Upgrade your Cloud Connector and avoid connectivity downtime during the update.

The steps for upgrading your Cloud Connector are specific to the operating system that you use. Previous settings and configurations are automatically preserved.

⚠ Caution

Upgrade is supported only for *installer* versions, not for *portable* versions, see [Installation \[page 282\]](#).

Before upgrading, please check the [Prerequisites \[page 283\]](#) and make sure your environment fits the new version. We recommend that you create a [Configuration Backup \[page 508\]](#) before starting an upgrade.

Avoid Connectivity Downtime

If you have a single-machine Cloud Connector installation, a short downtime is unavoidable during the upgrade process. However, if you have set up a master and a shadow instance, you can perform the upgrade without downtime by executing the following procedure:

1. Shut down the shadow instance.

2. Perform the upgrade on the shadow instance. (Follow the relevant procedure below.)
3. Restart the shadow instance and wait until it has connected to the master instance.
4. Perform a *Switch Roles* operation by pressing the corresponding button in the master administration UI.
The master instance has now changed into a shadow instance.

 **Caution**

After upgrading the former shadow instance from a version prior to 2.13 and having switched its role to be the new master instance, reset high availability settings in *both* instances *now*, before continuing to upgrade the second instance from a version prior to 2.13 as well. The master-shadow connection must be re-established after both instances have been upgraded from versions prior to 2.13 to versions 2.13 or higher.

5. Shut down the new shadow instance and perform the upgrade procedure on it as well.
6. Restart the new shadow instance and wait until it has connected to the already upgraded current master instance.
7. Perform again the *Switch Roles* operation if you want the previous master instance to act as the new master instance again.

Result: Both instances have now been upgraded without connectivity downtime and without configuration loss.

For more information, see [Install a Failover Instance for High Availability \[page 518\]](#).

Microsoft Windows OS

1. Uninstall the Cloud Connector as described in [Uninstallation \[page 581\]](#) and make sure to retain the existing configuration.
2. Reinstall the Cloud Connector within the same directory. For more information, see [Installation on Microsoft Windows OS \[page 302\]](#).
3. Before accessing the administration UI, clear your browser cache to avoid any unpredictable behavior due to the upgraded UI.

Linux OS

1. Execute the following command: :

```
rpm -U com.sap.scc-ui-<version>.rpm
```

 **Note**

Daemon extensions (as of Cloud Connector version 2.12.3)

All extensions to the daemon provided via `scc_daemon_extension.sh` mechanism will survive a version update. An upgrade to version 2.12.3 will already consider an existing file, even though previous versions were not supporting that feature.

- Before accessing the administration UI, clear your browser cache to avoid any unpredictable behavior due to the upgraded UI.

1.2.6 Update the Java VM

How to update the Java VM used by the Cloud Connector.

Sometimes you must update the Java VM used by the Cloud Connector, for example, because of expired SSL certificates contained in the JVM, bug fixes, deprecated JVM versions, and so on.

- If you make a replacement in the same directory, shut down the Cloud Connector, upgrade the JVM, and restart the Cloud Connector when you are done.
- If you change the installation directory of the JVM, follow the steps below** for your operating system.

Make sure that the JVM has been installed successfully.

i Note

A Java Runtime Environment (JRE) is not sufficient. You must use a JDK or SAP JVM.

Microsoft Windows OS

- Make sure that the current user has administrative privileges.
- Shutdown the Cloud Connector (for example, by stopping the corresponding Windows service or by double-clicking the *Stop SAP Cloud Connector* shortcut).
- Open the *registry editor* (regedit32) and locate the following registry entry: `HKEY_LOCAL_MACHINE\SOFTWARE\SAP\Cloud Connector`.
- Change the value `JavaHome` to reflect the installation directory of the new Java VM.

i Note

The `bin` subdirectory must not be part of the `JavaHome` value.

If the `JavaHome` value does not yet exist, create it here with a "String Value" (REG_SZ) and specify the full path of the Java installation directory, for example: `C:\Program Files\sapjvm`.

- Close the registry editor and restart the Cloud Connector.

Linux OS

- Open a shell as root user.
- In this shell, set the environment variable `JAVA_HOME`, pointing to the installation directory of the new Java VM, for example, in tcsh:

```
setenv JAVA_HOME /opt/sap/sapjvm
```

3. Execute the command

```
System V init distributions: service scc_daemon stop  
systemd distributions: systemctl stop scc_daemon
```

4. Execute the command

```
System V init distributions: /opt/sap/scc/daemon.sh reinstall  
systemd distributions: /opt/sap/scc/daemon.sh reinstallSystemd
```

5. Execute the command

```
System V init distributions: service scc_daemon start  
systemd distributions: systemctl start scc_daemon
```

Both Operating Systems

After executing the above steps, the Cloud Connector should be running again and should have picked up the new Java version during startup. You can verify this by logging in to the Cloud Connector with your favorite browser, opening the [About](#) dialogue and checking that the field <Java Details> shows the version number and build date of the new Java VM. After you verified that the new JVM is indeed used by the Cloud Connector, delete or uninstall the old JVM.

1.2.7 Uninstallation

Uninstall an installer version or portable version of the Cloud Connector.

- If you have installed an installer variant of the Cloud Connector, follow the steps for your operating system to uninstall the Cloud Connector.
- To uninstall a developer version, proceed as described in section **Portable Variants**.

Microsoft Windows OS

1. In the Windows software administration tool, search for **Cloud Connector** (formerly named **SAP HANA cloud connector 2.x**).
2. Select the entry and follow the appropriate steps to uninstall it.
3. When you are uninstalling in the context of an upgrade, make sure to retain the configuration files.

Linux OS

To uninstall Cloud Connector 2.x, execute the following command:

```
rpm -e com.sap.scc-ui
```

⚠ Caution

This command also removes the configuration files.

Mac OS X

There is no installer variant for Mac OS X, only a portable one.

Portable Variants

(Microsoft Windows OS, Linux OS, Mac OS X) If you have installed a portable version (zip or tgz archive) of the Cloud Connector, simply remove the directory in which you have extracted the Cloud Connector archive.

Related Information

[Installation \[page 282\]](#)

1.2.8 Frequently Asked Questions

Answers to the most common questions about the Cloud Connector.

Technical Issues

Does the Cloud Connector send data from on-premise systems to SAP BTP or the other way around?

The connection is opened from the on-premise system to the cloud, but is then used in the other direction.

An on-premise system is, in contrast to a cloud system, normally located behind a restrictive firewall and its services aren't accessible thru the Internet. This concept follows a widely used pattern often referred to as *reverse invoke proxy*.

Is the connection between the SAP BTP and the Cloud Connector encrypted?

Yes, by default, TLS encryption is used for the tunnel between SAP BTP and the Cloud Connector.

If used properly, TLS is a highly secure protocol. It is the industry standard for encrypted communication and also, for example, as a secure channel in HTTPS.

Keep your Cloud Connector installation updated and we will make sure that no weak or deprecated ciphers are used for TLS.

Can I use a TLS-terminating firewall between Cloud Connector and SAP BTP?

This is not possible. Basically, this is a desired man-in-the-middle attack, which does not allow the Cloud Connector to establish a mutual trust to the SAP BTP side.

What is the oldest version of SAP Business Suite that's compatible with the Cloud Connector?

The Cloud Connector can connect an SAP Business Suite system version 4.6C and newer.

Which JRE versions are supported to run the Cloud Connector?

	Supported JRE Version			
	6	7	8	
Cloud Connector Version	< 2.7.2	Yes	Yes	No
	= 2.7.2	Yes	Yes	Yes
	>= 2.8	No	Yes	Yes
	>= 2.12.3	No	No	Yes

! Restriction

Support for Java 7 has been discontinued. For more information, see [Prerequisites \[page 285\]](#).

→ Tip

We recommend that you always use the latest supported JRE version.

⚠ Caution

Version 2.8 and later of the Cloud Connector may have problems with ciphers in Google Chrome, if you use the JVM 7. For more information read [this SCN Article](#).

Which configuration in the SAP BTP destinations do I need to handle the user management access to the Cloud User Store of the Cloud Connector?

See [Configure an On-Premise User Store \[page 490\]](#).

Is the Cloud Connector sufficient to connect the SAP BTP to an SAP ABAP back end or is SAP BTP Integration needed?

It depends on the scenario: For pure point-to-point connectivity to call on-premise functionality like BAPIs, RFCs, OData services, and so on, that are exposed via on-premise systems, the Cloud Connector might suffice.

However, if you require advanced functionality, for example, n-to-n connectivity as an integration hub, SAP BTP Integration – Process Integration is a more suitable solution. SAP BTP Integration can use the Cloud Connector as a communication channel.

How much bandwidth does the Cloud Connector consume?

The amount of bandwidth depends greatly on the application that is using the Cloud Connector tunnel. If the tunnel isn't currently used, but still connected, a few bytes per minute is used simply to keep the connection alive.

What happens to a response if there's a connection failure while a request is being processed?

The response is lost. The Cloud Connector only provides tunneling, it does not store and forward data when there are network issues.

Where should I install the Cloud Connector?

For productive instances, we recommend installing the Cloud Connector on a single purpose machine. This is relevant for [Security \[page 567\]](#). For more details on which network zones to choose for the Cloud Connector setup, see [Network Zones \[page 295\]](#).

How many servers do I need to deploy the Cloud Connector?

We recommend that you use at least three servers, with the following purposes:

- Development
- Production master
- Production shadow

i Note

Do not run the production master and the production shadow as VMs inside the same physical machine. Doing so removes the redundancy, which is needed to guarantee high availability. A QA (Quality Assurance) instance is a useful extension. For disaster recovery, you will also need two additional instances; another master instance, and another shadow instance.

What are the hardware requirements to deploy the Cloud Connector?

See: [Prerequisites \[page 283\]](#).

Can I send push messages from an on-premise system to the SAP BTP through the Cloud Connector?

No, this is not supported by the Cloud Connector.

Is NTLM supported for authorization against the proxy server?

No, the Cloud Connector currently supports only basic authentication.

What operating systems are supported by the Cloud Connector?

See [Prerequisites \[page 283\]](#).

What processor architectures are supported by the Cloud Connector?

We currently support 64-bit operating systems running only on an x86-64 processor (also known as x64, x86_64 or AMD64).

See: [Prerequisites \[page 283\]](#).

Can I use the Cloud Connector without an ABAP back end?

Yes, you should be able to connect almost any system that supports the HTTP Protocol, to the SAP BTP, for example, Apache HTTP Server, Apache Tomcat, Microsoft IIS, or Nginx.

Can I authenticate with client certificates configured in SAP BTP destinations at HTTP services that are exposed via the Cloud Connector?

No, this is not possible. For client certificate authentication, an end-to-end TLS communication is required. This is not the case, because the Cloud Connector needs to inspect incoming requests in order to perform access control checks.

How can I do connection pooling for HTTP services that are exposed via the Cloud Connector?

The Cloud Connector itself does not perform connection pooling, but provides a 1-to-1 mapping for each logical connection received through the tunnel.

By this mapping, a new connection to the backend system is opened, and kept open until closed either by the backend or by the client on cloud side.

The actual connection pooling is defined by the application client on cloud side:

- If a connection is re-used in the client library, it is re-used on the Cloud Connector side as well.
- If it is closed immediately, also the mapped one on Cloud Connector side will be closed immediately.

Administration

Are there Audit Logs for changes in the Cloud Connector?

Yes, find more details here: [Manage Audit Logs \[page 556\]](#).

Is it possible to split authorization?

No, currently there is only one role that allows complete administration of the Cloud Connector.

Can I configure multiple administrative subaccounts?

Yes, to enable this, you must configure an LDAP server. See: [Use LDAP for Authentication \[page 513\]](#).

How can I reset the Cloud Connector's administrator password when not using LDAP for authentication?

Visit <https://tools.hana.ondemand.com/#cloud> to download the portable version of the Cloud Connector. Extract the `users.xml` file in the `config` directory to the `config` directory of your Cloud Connector installation, then restart the Cloud Connector.

This resets the password and user name to their default values.

You can manually edit the file; however, we strongly recommend that you use the `users.xml` file.

How do I create a backup of the Cloud Connector configuration?

Starting with Cloud Connector version 2.11, you can use a dedicated backup feature, either from the administration UI (see [Configuration Backup \[page 508\]](#)) or via REST API (see [Backup \[page 447\]](#)).

Can I create a backup of the complete installation?

Yes, you can create an archive file of the installation directory to create a full backup. Before you restore from a backup, note the following:

- If you restore the backup on a different host, the UI certificate will be invalidated.
- Before you restore the backup, you should perform a “normal” installation and then replace the files. This registers the Cloud Connector at your operating systems package manager.

Why do I need a user ID during configuration?

This user opens the tunnel and generates the certificates that are used for mutual trust later on.

The user is not part of the certificate that identifies the Cloud Connector.

In both the Cloud Connector UI and in the SAP BTP cockpit, this user ID appears as the one who performed the initial configuration (even though the user may have left the company).

What happens to a Cloud Connector connection if the user who created the tunnel leaves the company?

This does not affect the tunnel, even if you restart the Cloud Connector.

What do changes in major or minor version numbers mean?

The semantics of Cloud Connector versions are explained in detail [here](#).

For how long does SAP continue to support older Cloud Connector versions?

Each Cloud Connector version is supported for 12 months, which means the cloud side infrastructure is guaranteed to stay compatible with those versions.

After that time frame, compatibility is no longer guaranteed and interoperability could be dropped. Furthermore, after an additional 3 month, the next feature release published after that period will no longer support an upgrade from the deprecated version as a starting release.

What is the difference between “subaccount name” and “subaccount user”?

SAP BTP customers can purchase subaccounts and deploy applications into these subaccounts.

Additionally, there are users, who have a password and can log in to the cockpit and manage all subaccounts they have permission for.

- A single subaccount can be managed by multiple users, for example, your company may have several administrators.
- A single user can manage multiple subaccounts, for example, if you have multiple applications and want them (for isolation reasons) to be split over multiple subaccounts.

Find your account name by taking the following steps:

1. Open the SAP BTP cockpit.
2. Log in with your subaccount user.
3. You'll see the subaccount name in the top left section of the screen.

For trial users, the account name is typically your user name, followed by the suffix “trial”:

The screenshot shows the SAP BTP Cloud Foundry interface. The left sidebar has sections for Overview, Applications (selected), Java Applications, HTML5 Applications, HANA XS Applications, Subscriptions, Services, Persistence, and Connectivity. The top navigation bar shows 'Europe (Trial)' and the current user 'userXYtrial'. The main area is titled 'System Status' and contains two cards: 'JAVA' and 'HTML5'. The JAVA card shows 'Overall Health' with a yellow diamond icon, '15 Applications' (all stopped), and 'N/A'. The HTML5 card shows 'Overall Health' with a green checkmark icon, 'OK', and 'N/A'.

Features

Does the Cloud Connector work with the SAP BTP Cloud Foundry environment?

As of version 2.10, the Cloud Connector can establish a connection to regions based on the SAP BTP Cloud Foundry environment. Newer regions, however, require a Cloud Connector version 2.11 or higher.

Does the Cloud Connector work with SAP S/4HANA Cloud?

As of version 2.10, the Cloud Connector offers a Service Channel to S/4HANA Cloud instances, given that they are associated with the respective SAP BTP subaccount. For more information, see [Using Service Channels \[page 492\]](#).

Also supported as of version 2.10: S/4HANA Cloud communication scenarios invoking HTTP services or remote-enabled function modules (RFMs) in on-premise ABAP systems.

Does the Cloud Connector work with the SAP BTP ABAP environment?

As of version 2.11, the Cloud Connector supports communication from and to the SAP BTP ABAP environment, when using the **Neo** Connectivity service. Using the **Cloud Foundry** Connectivity service requires a Cloud Connector version 2.12.3 or higher.

How do I bind multiple Cloud Connectors to one SAP BTP subaccount?

As of version 2.9, you can connect multiple Cloud Connectors to a single subaccount. This lets you assign multiple separate corporate network segments.

Those Cloud Connectors are distinguishable based on the location ID, which you must provide to the destination configuration on the cloud side.

i Note

During an upgrade, location IDs provided in earlier versions of the Cloud Connector are dropped to ensure that running scenarios are not disturbed.

Is WebSocket communication through the Cloud Connector supported?

Yes, this is possible as of version 2.12.

Is there any plan to add traffic management functionality in Cloud Connector?

No, this functionality is not currently planned.

Can I use the Cloud Connector for any protocol?

As of version 2.10, this is possible using the TCP channel of the Cloud Connector, if the client supports a SOCKS5 proxy to establish the connection. However, only the HTTP and RFC protocols currently provide an additional level of access control by checking invoked resources.

You can also use the Cloud Connector as a JDBC or ODBC proxy to access the HANA DB instance of your SAP BTP subaccount (service channel). This is sometimes called the "HANA Protocol".

Can I check the communication of the service channel?

No, the audit log monitors access only from SAP BTP to on-premise systems.

Troubleshooting

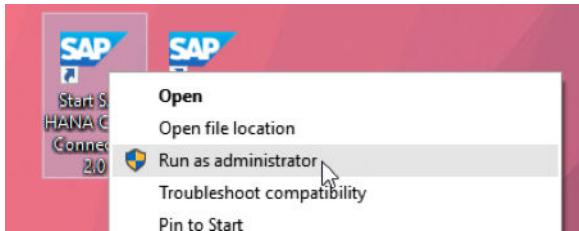
How do I fix the “Could not open Service Manager” error message?

You are probably seeing this error message due to missing administrator privileges. Right-click the cloud connector shortcut and select Run as administrator.

If you don't have administrator privileges on your machine you can use the portable variant of the Cloud Connector.

i Note

The portable variants of the Cloud Connector are meant for nonproductive scenarios only.



How do I set JAVA_HOME and PATH correctly?

For the portable versions, JAVA_HOME must point to the installation directory of your JRE, while PATH must contain the *bin* folder inside the installation directory of your JRE.

The installer versions automatically detect JVMs in these locations, as well as in other places.

When I try to open the Cloud Connector UI, Google Chrome opens a Save as dialog, Firefox displays some cryptic signs, and Internet Explorer shows a blank page, how do I fix this?

This happens when you try to access the Cloud Connector over HTTP instead of HTTPS. HTTP is the default protocol for most browsers.

Adding "https://" to the beginning of your URL should fix the problem. For localhost, you can use `https://localhost:8443/`.

1.3 Connectivity Proxy for Kubernetes

Use the connectivity proxy for Kubernetes to connect workloads on a Kubernetes cluster to on-premise systems.

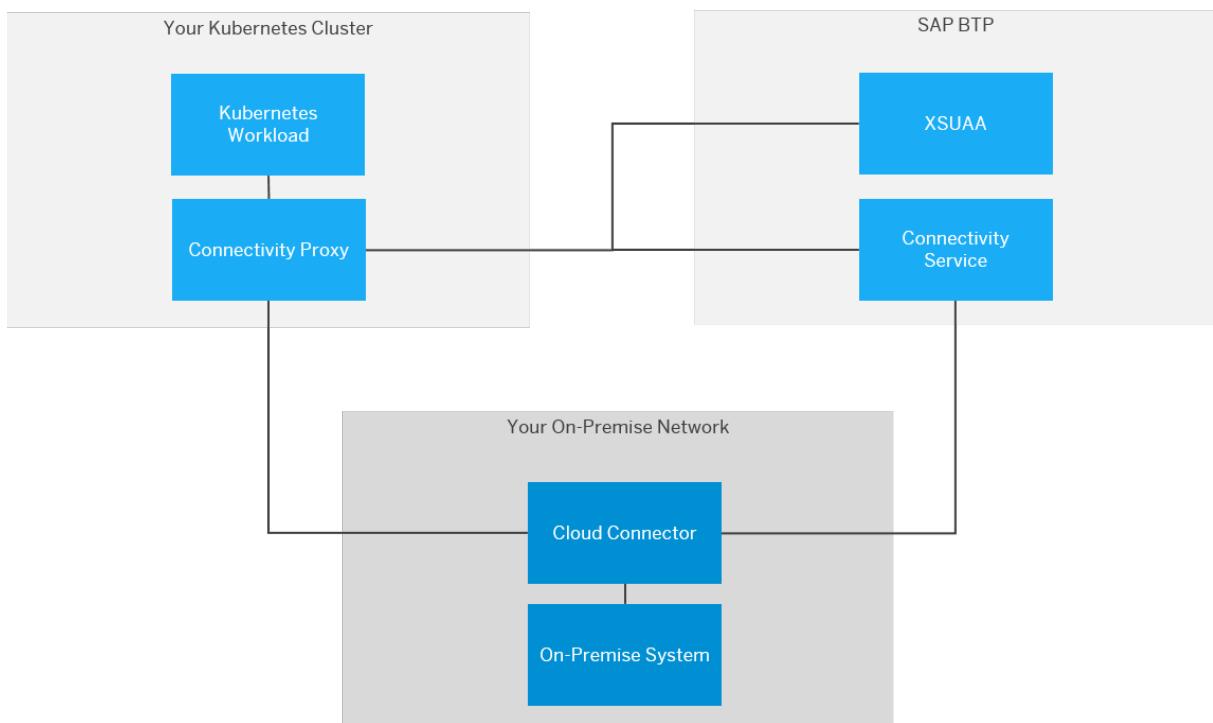
The connectivity proxy is a Kubernetes component that connects workloads running on a Kubernetes cluster to on-premise systems, which are exposed via the [Cloud Connector \[page 276\]](#). The connectivity proxy must be paired to an SAP BTP region to grant access to the Cloud Connectors connected to that region. The SAP BTP domain model (subaccounts) is used to target a particular Cloud Connector.

The connectivity proxy is delivered as a Docker image and a Helm chart. You need to run the image on your Kubernetes cluster with appropriate configurations. The Helm chart simplifies the installation process. See [Lifecycle Management \[page 612\]](#) for more details.

You can find information about new versions of the connectivity proxy via SAP BTP [release notes](#). To request a new feature, you can use [Influence SAP](#).

i Note

The connectivity proxy is only available for clusters provisioned by [Gardener](#).



Related Information

[Concepts \[page 591\]](#)

[Lifecycle Management \[page 612\]](#)

[Verification and Testing \[page 629\]](#)

[Monitoring \[page 631\]](#)

[Using the Connectivity Proxy \[page 632\]](#)

[Troubleshooting \[page 635\]](#)

[Frequently Asked Questions \[page 641\]](#)

1.3.1 Concepts

Find an overview of important concepts for working with the connectivity proxy for Kubernetes.

[How the Connectivity Proxy Works \[page 592\]](#)

Learn about the connectivity proxy for Kubernetes: Scenario and configuration steps.

[Operational Modes \[page 595\]](#)

Details about the different operational modes of the connectivity proxy.

Mutual TLS [page 599]	Use Transport Layer Security (TLS) with the connectivity proxy.
External Health Checking [page 603]	Perform external health checks for the connectivity proxy.
High Availability [page 606]	Run the connectivity proxy for Kubernetes in high availability mode.
Audit Logging [page 609]	Using audit logging for the connectivity proxy.
Integration with SAP Services [page 610]	Integrate the connectivity proxy with SAP services.

1.3.1.1 How the Connectivity Proxy Works

Learn about the connectivity proxy for Kubernetes: Scenario and required configuration.

[Glossary \[page 592\]](#)

[Scenario \[page 593\]](#)

- [Prerequisites \[page 593\]](#)
- [Runtime Flow in Steps \[page 594\]](#)
- [Involved Parties \[page 594\]](#)

[Configuration \[page 594\]](#)

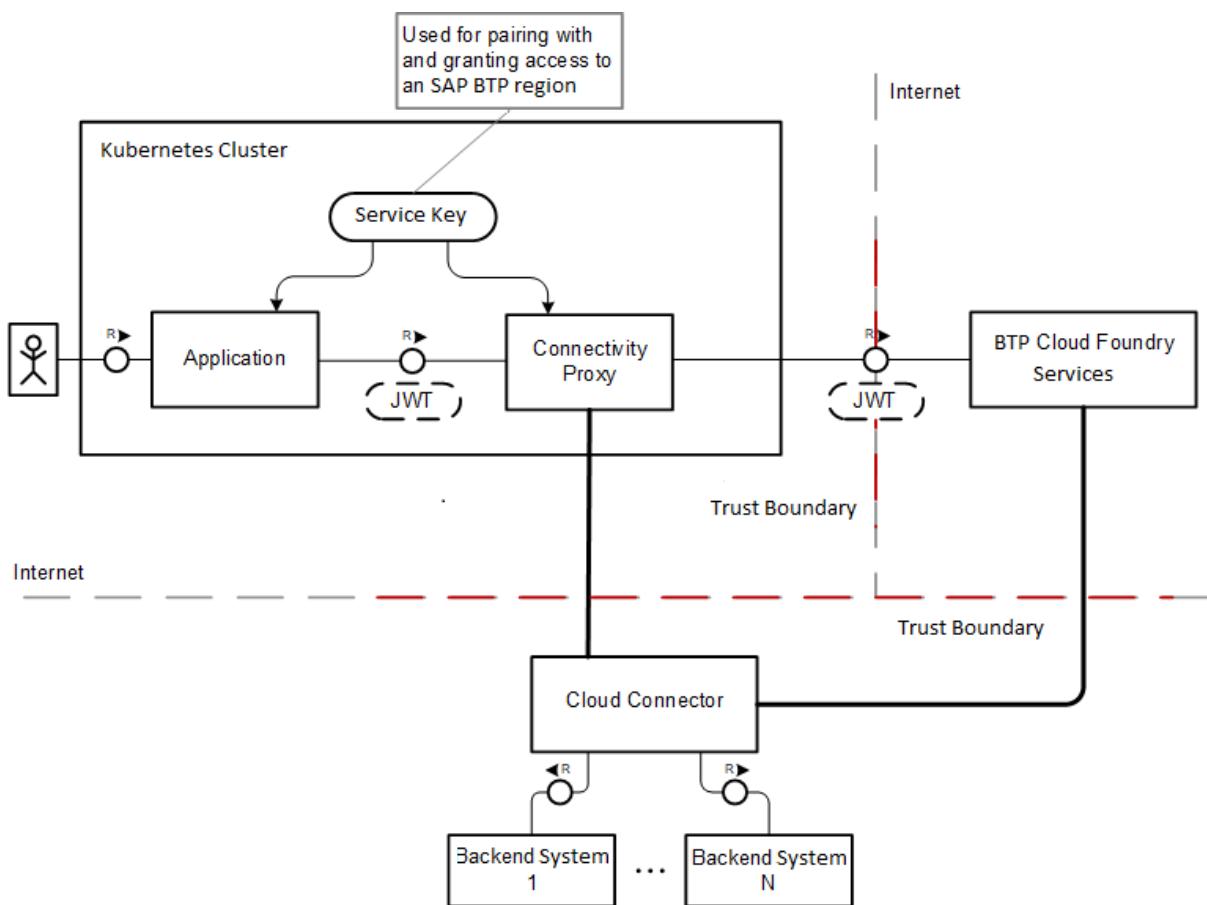
Glossary

- **SAP Business Technology Platform** (SAP BTP): SAP platform for cloud services (replaces SAP Cloud Platform).
- **SAP Connectivity service**: Core platform service, offering a secure tunnelling solution between your on-premise network and the cloud.
- **Cloud Connector**: On-premise client of the Connectivity service, deployed and lifecycle-managed in your local network. The initiator of the secure tunnel to the platform, that is, to the Connectivity service.
- **Connectivity proxy**: Software component (logically part of the Connectivity service), deployed locally to the consuming part (usually a cloud application or a service component). It can work in multiple operational modes, depending on the exact requirement of the consuming party.
- **SAP UAA** (aka XSUAA): SAP Authorization service, issues client credentials and access tokens, associated with the platform tenancy model.

[Back to Top \[page 592\]](#)

Scenario

An end user works with a cloud application or solution. To complete the task, the application or solution needs to connect to an on-premise system (hosted either by the consumer tenant or the cloud application provider tenant). The system is not accessible directly via Internet, but securely exposed by the Cloud Connector. Only selected parts of the system functionality may be exposed to the cloud application. For more information, see [Cloud Connector \[page 276\]](#).



Prerequisites

- The connectivity proxy is deployed and configured in the Kubernetes cluster (see [Lifecycle Management \[page 612\]](#)).
- A cloud application is deployed on Kubernetes, next to the connectivity proxy, and it is configured to connect to the proxy (see [Using the Connectivity Proxy \[page 632\]](#)). The cloud application is up and running and accessible by end users.
- The Cloud Connector is installed and configured in your local network and connected to the cloud (that is, to the Connectivity service), and stays in a ready-to-be-used mode (see [Cloud Connector \[page 276\]](#)).
- The on-premise systems that the cloud application needs to connect to are properly exposed via / configured in the Cloud Connector (see [Configure Access Control \[page 379\]](#)).

[Back to Scenario \[page 593\]](#)

[Back to Top \[page 592\]](#)

Runtime Flow in Steps

When all prerequisites are met, the cloud application can be properly used by end users:

- An end user works with a *client tool*, for example, a browser or a REST client.
- The *client tool* connects to a *cloud application*, in this case hosted in a Kubernetes cluster.
- The *cloud application* knows it needs to connect to the on-premise system, therefore it connects to the connectivity proxy. Depending on the specific requirements of the cloud application environment, the *cloud application* might need to obtain an access token which is used to authenticate the application as a client, as well as to authorize it to connect to the respective tenant-specific Cloud Connector exposing the target system (see [Operational Modes \[page 595\]](#)).
- The *connectivity proxy* accepts the client proxy request (by the *cloud application*) and routes the traffic via a TLS secure tunnel, which has been already initiated by the Cloud Connector and successfully established.
- The Cloud Connector receives the *request data* from the *cloud application* and performs the related access control checks. A connection is established to the target on-premise system, and the request data is forwarded to the on-premise system.
- The *response data* from the on-premise system is routed back to the *cloud application*.
- The *cloud application* processes the response data retrieved by the on-premise system and shows the result to the end user.

[Back to Scenario \[page 593\]](#)

[Back to Top \[page 592\]](#)

Involved Parties

- **Cloud application:** Business workload initiated by end users (or a background job) of the business solution.
- **SAP BTP Services:** The connectivity proxy cannot operate on its own. It needs to connect to other services for key operations, namely:
 - *authorization - XSUAA:* Ensure any operation is properly secured.
 - *pairing/integration with SAP Connectivity service:* Secure access control to Cloud Connectors.
- **On-Premise systems or services:** The target system, securely hosted in your local network, usually behind a firewall, and exposed via the Cloud Connector.

[Back to Scenario \[page 593\]](#)

[Back to Top \[page 592\]](#)

Configuration

The connectivity proxy for Kubernetes is meant to

- Be deployed as a Kubernetes [StatefulSet](#). As it is accessed both by other Kubernetes deployments and the Cloud Connector, you need to configure access accordingly.
- Connect to other (remote) services.
- Enforce authorization on its proxy endpoints.

For all these points, proper configuration is required. For more information, see [Lifecycle Management \[page 612\]](#).

[Back to Top \[page 592\]](#)

1.3.1.2 Operational Modes

Learn about the different operational modes of the connectivity proxy for Kubernetes.

The connectivity proxy can run in four different operational modes, based on two main categories:

- Trust for the surrounding environment and the callers of the proxy
- Tenant usage of the proxy

Find below the details for each operational mode:

[Single-Tenant Usage: Non-Trusted Environment \[page 595\]](#)

[Multi-Tenant Usage: Non-Trusted Environment \[page 596\]](#)

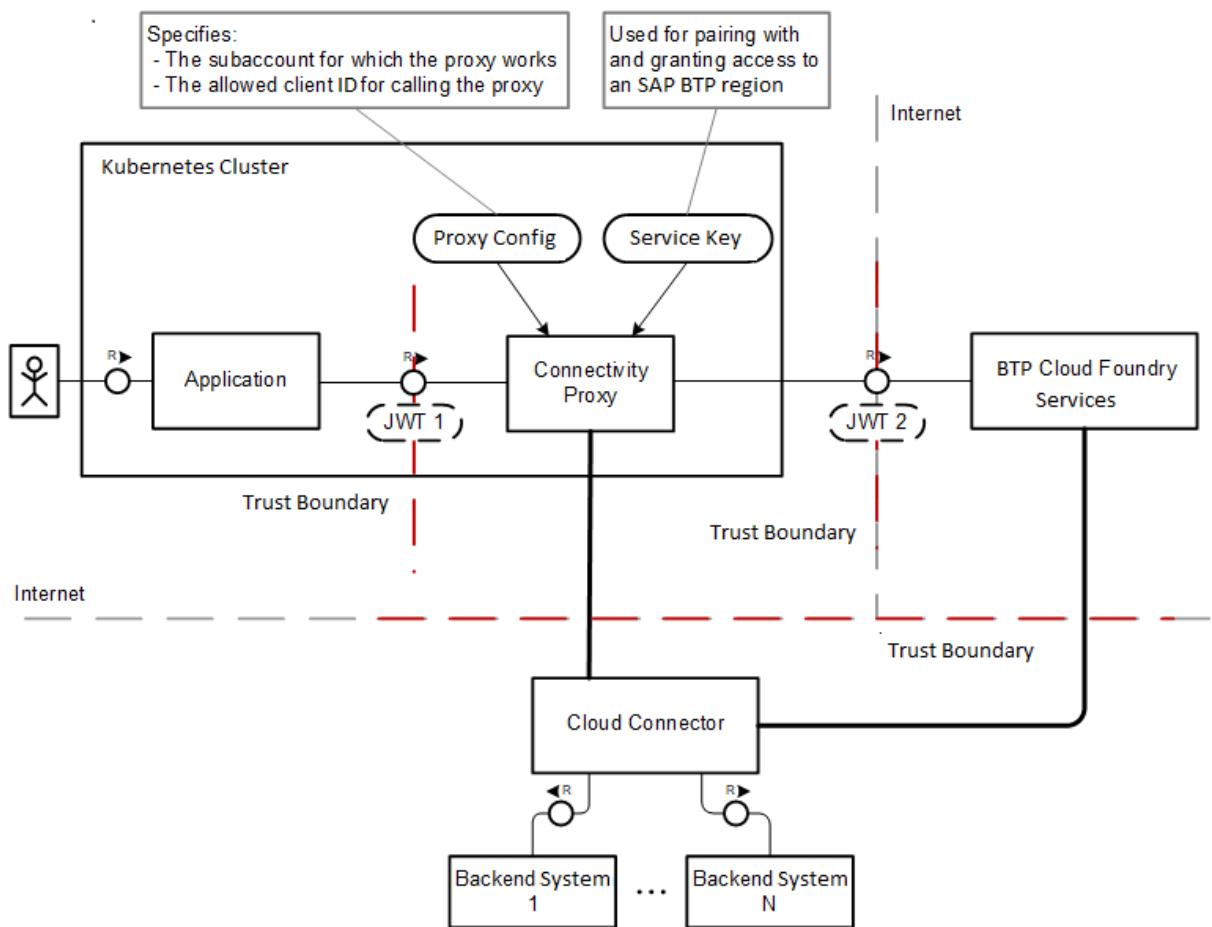
[Single-Tenant Usage: Trusted Environment \[page 597\]](#)

[Multi-Tenant Usage: Trusted Environment \[page 598\]](#)

Single-Tenant Usage: Non-Trusted Environment

The connectivity proxy operates on behalf of a single, statically configured tenant. Applications cannot be trusted. The connectivity proxy is configured as follows:

- Proxy authorization is *enabled*.
- Tenant mode is *dedicated*.
- Uses a statically configured service key of the `connectivity_proxy` service instance of the `connectivity` service.
- Connects to the central Connectivity service on behalf of the statically configured tenant.

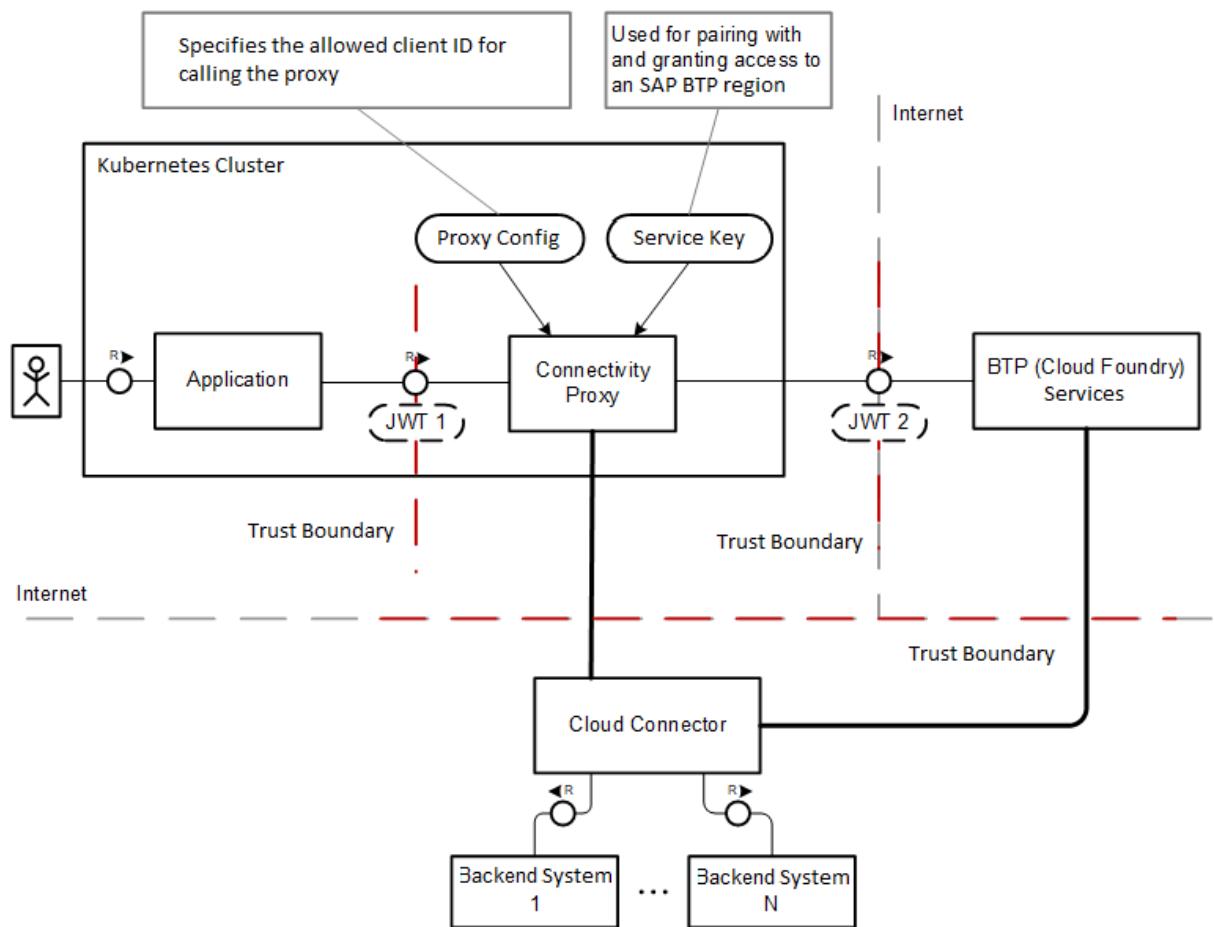


[Back to Top \[page 595\]](#)

Multi-Tenant Usage: Non-Trusted Environment

The connectivity proxy operates on behalf of multiple tenants. Applications cannot be trusted. The connectivity proxy is configured as follows:

- Proxy authorization is *enabled*.
- Tenant mode is *shared*.
- Uses a statically configured service key of the `connectivity_proxy` service instance of the `connectivity` service.
- Connects to the central Connectivity service on behalf of the dynamically determined tenant, based on the OAuth access token (JWT) forwarded by the application.

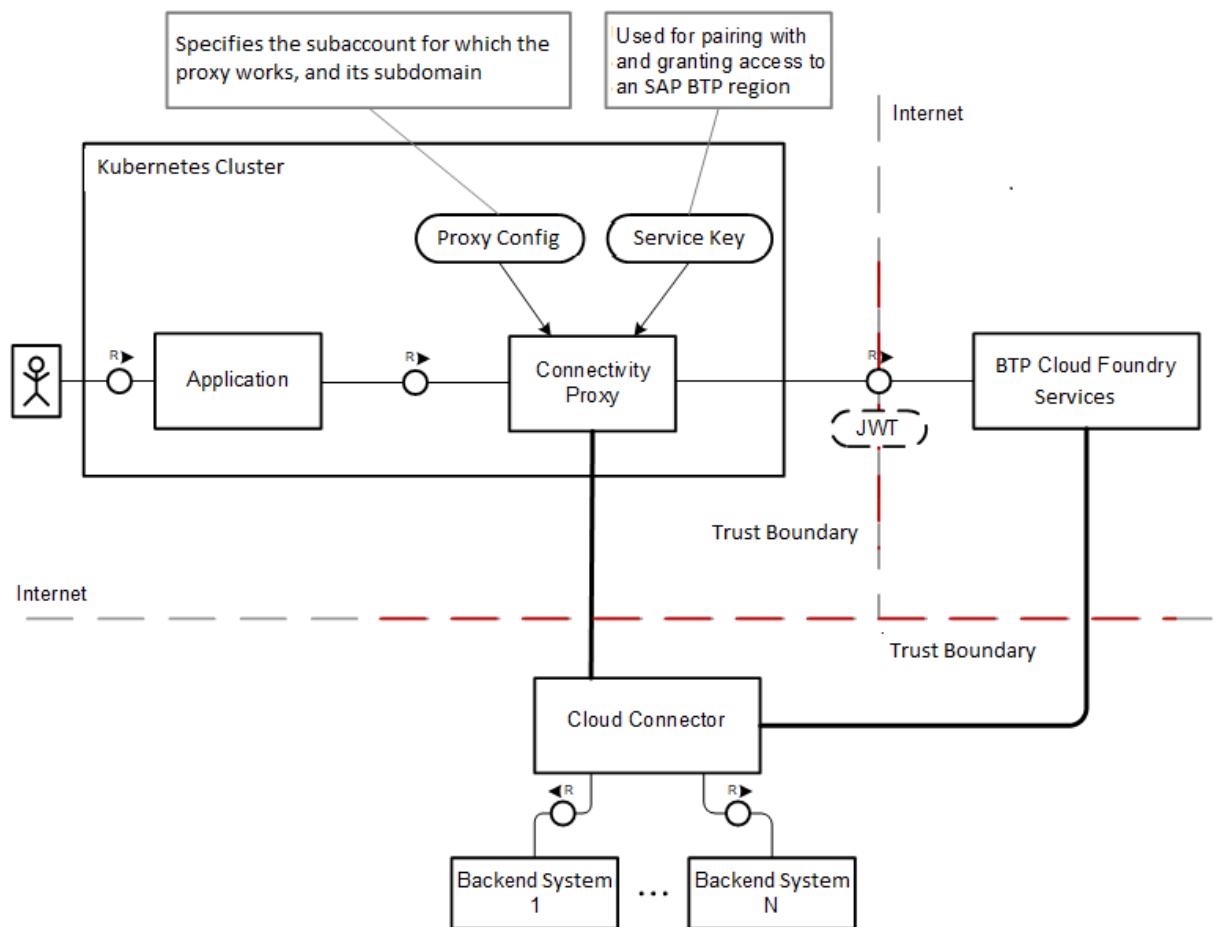


[Back to Top \[page 595\]](#)

Single-Tenant Usage: Trusted Environment

The connectivity proxy operates on behalf of a single, statically configured tenant. Applications are trusted. The connectivity proxy is configured as follows:

- Proxy authorization is *disabled*.
- Tenant mode is *dedicated*.
- Uses a statically configured service key of the `connectivity_proxy` service instance of the `connectivity` service.
- Connects to the central Connectivity service on behalf of the statically configured tenant.

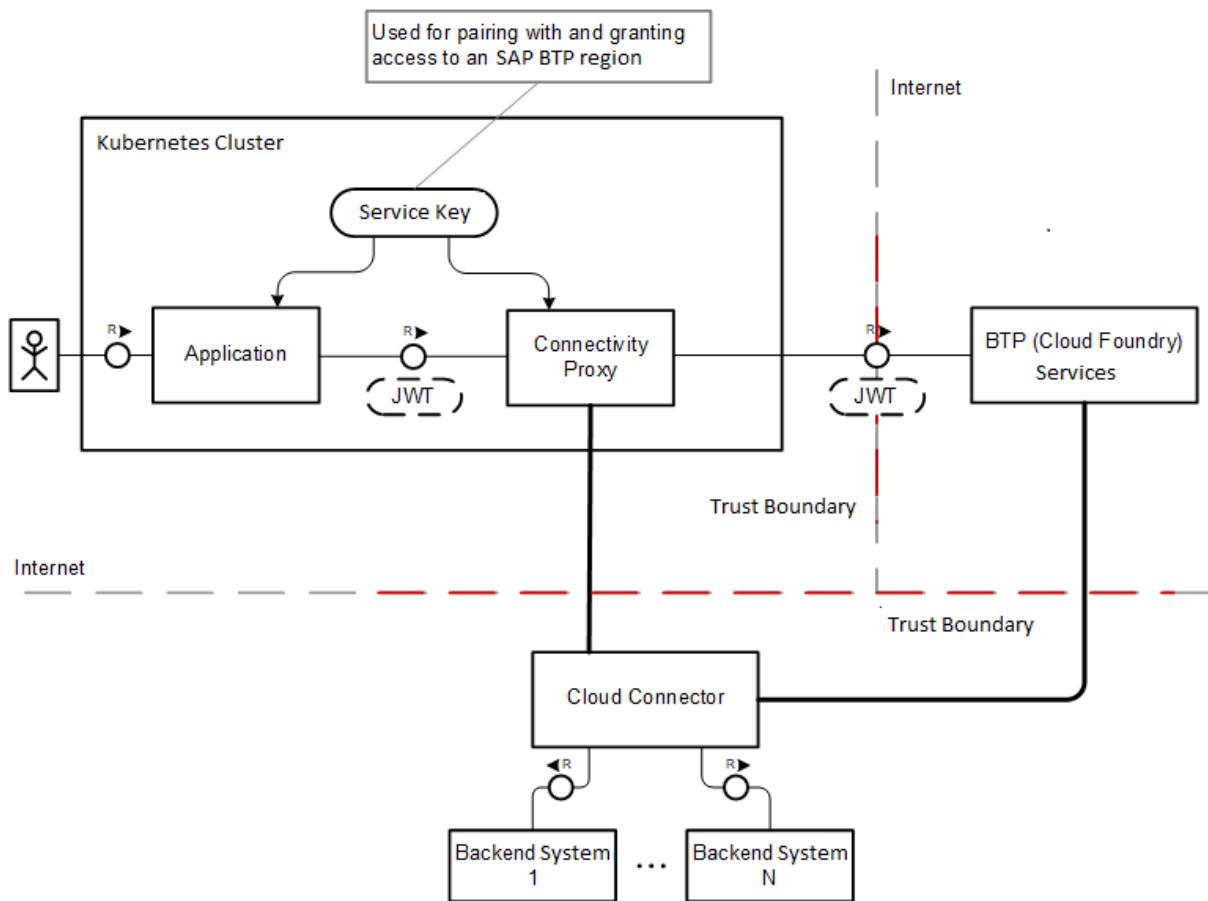


[Back to Top \[page 595\]](#)

Multi-Tenant Usage: Trusted Environment

The connectivity proxy operates on behalf of multiple tenants. Applications are trusted. Applications use service keys of the `connectivity_proxy` service instance of the `connectivity` service. The connectivity proxy is configured as follows:

- Proxy authorization is *disabled*.
- Tenant mode is *shared*.
- Connects to the central Connectivity service on behalf of the dynamically determined tenant, using the OAuth access token (JWT) forwarded by the application.



[Back to Top \[page 595\]](#)

1.3.1.3 Mutual TLS

Use Transport Layer Security (TLS) for the connectivity proxy for Kubernetes.

TLS encrypts the connection between client and server, following the TLS specification. When using mutual TLS, both the TLS client and the TLS server authenticate each other through X.509 certificates.

In an on-premise network, the TLS client is represented by the Cloud Connector. On the cloud side, the direct TLS server may be:

- The Kubernetes Ingress: In this case, the Ingress is terminates the TLS connection and establishes a new TLS connection to the connectivity proxy.
- The connectivity proxy: In this case, the Ingress does not terminate the TLS, but transparently forwards the TLS traffic to the connectivity proxy.

Connectivity proxy deployment provides two options to configure end-to-end mutual TLS, that is, the TLS communication between Cloud Connector and connectivity proxy:

- [With TLS termination in the Ingress \[page 600\]](#): TLS configuration has to be made on both the Ingress controller (TLS client) and connectivity proxy (TLS server).

- [Without TLS termination in the Ingress \[page 602\]](#): TLS configuration has to be made only on the connectivity proxy side (TLS server).

End-to-End Mutual TLS with Termination of the TLS Connection in the Ingress

Perform the following steps:

1. Enable the TLS communication on the connectivity proxy (TLS server) side, adding the following configuration in the `values.yaml` file:

```
config:
  servers:
    businessDataTunnel:
      enableTls: true
```

2. Add the required TLS configuration for the connectivity proxy (TLS server). There are two options to do this:

- If you already have an appropriate secret for this purpose in the cluster, add the following configuration in the `values.yaml` file:

```
secretConfig:
  servers:
    businessDataTunnel:
      secretName: <secret name>
```

i Note

The Kubernetes secret must contain the following properties, used for authentication to the Ingress resource:

- `tls.key`: Base 64-encoded private key in PEM format.
- `tls.crt`: Base 64-encoded certificate in PEM format.
- `ca.crt`: Base 64-encoded full certificate authority (CA) chain in PEM format.

- If you don't have such a Kubernetes secret, it can be automatically generated. Add the following configuration in the `values.yaml` file:

```
secretConfig:
  servers:
    businessDataTunnel:
      secretName: <secret name>
      secretData:
        key: <base 64 encoded private key in PEM format>
        certificate: <base 64 encoded certificate in PEM format>
        caCertificate: <base 64 encoded full Certificate Authority chain in PEM format>
```

i Note

The certificate must be issued for the external host specified on the configuration path `config.servers.businessDataTunnel.externalHost` or for the domain to which the

external host belongs. For example, if the external host is "ingress.mycluster.com", the certificate CN or SAN must contain "ingress.mycluster.com" or "*mycluster.com".

3. Add the required TLS configuration for the Ingress. There are three options to do this:

- If you already have an appropriate Kubernetes secret for this purpose in the cluster, add the following configuration in the `values.yaml` file:

```
ingress:  
  tls:  
    proxy:  
      secretName: <secret name>
```

i Note

The Kubernetes secret must contain the following properties, used for authentication to the Ingress resource:

- `tls.key`: Base 64-encoded private key in PEM format.
 - `tls.crt`: Base 64-encoded certificate in PEM format.
 - `ca.crt`: Base 64-encoded full certificate authority (CA) chain in PEM format.
- If you don't have such a secret, it can be automatically generated. Add the following configuration in the `values.yaml` file:

```
ingress:  
  tls:  
    proxy:  
      secretName: <secret name>  
      secretData:  
        key: <base 64 encoded private key in PEM format>  
        certificate: <base 64 encoded certificate in PEM format>  
        caCertificate: <base 64 encoded full Certificate Authority chain  
in PEM format>
```

- If you didn't add any TLS configuration for the Ingress, the TLS configuration of the connectivity proxy is reused for the Ingress, that is, the connectivity proxy and the Ingress will use the same TLS configuration to communicate to each other.

i Note

In this case, the specified certificate must be issued by the specified certificate authority.

i Note

The certificate must be issued for the external host specified on the configuration path `config.servers.businessDataTunnel.externalHost` or for the domain to which the external host belongs. For example, if the external host is "ingress.mycluster.com", the certificate CN or SAN must contain "ingress.mycluster.com" or "*mycluster.com".

i Note

In this case, the **minimum required version** for the NGINX ingress controller is 0.31.0.

End-to-End Mutual TLS without Termination of the TLS Connection in the Ingress

Perform the following steps:

1. Enable the TLS communication on the connectivity proxy (TLS server) side, adding the following configuration in the `values.yaml` file:

```
config:  
  servers:  
    businessDataTunnel:  
      enableTls: true
```

2. Add the required TLS configuration for the connectivity proxy (TLS server). There are two options to do this:
 - If you already have an appropriate secret for this purpose in the cluster, add the following configuration in the `values.yaml` file:

```
secretConfig:  
  servers:  
    businessDataTunnel:  
      secretName: <secret name>
```

i Note

The Kubernetes secret must contain the following properties, used for authentication to the Ingress resource:

- `tls.key`: Base 64-encoded private key in PEM format.
- `tls.crt`: Base 64-encoded certificate in PEM format.
- `ca.crt`: Base 64-encoded full certificate authority (CA) chain in PEM format.

- If you don't have such a secret, it can be automatically generated. Add the following configuration in the `values.yaml` file:

```
secretConfig:  
  servers:  
    businessDataTunnel:  
      secretName: <secret name>  
      secretData:  
        key: <base 64 encoded private key in PEM format>  
        certificate: <base 64 encoded certificate in PEM format>  
        caCertificate: <base 64 encoded full Connectivity Certificate Authority chain in PEM format>
```

i Note

The certificate must be issued for the external host specified on the configuration path `config.servers.businessDataTunnel.externalHost` or for the domain to which the

external host belongs. For example, if the external host is "ingress.mycluster.com", the certificate CN or SAN must contain "ingress.mycluster.com" or "*mycluster.com".

3. Skip the TLS configuration on the Ingress resource, that is, skip the configuration path `ingress.tls` in the `values.yaml` file.

i Note

In this case, you must set the configuration parameter `config.servers.businessDataTunnel.strictSniEnabled` to `false`, because this strict SNI configuration is relevant for a TLS termination in the Ingress. In case of direct TLS connection, strict SNI is supported by default.

i Note

By default, the NGINX ingress controller does not support communication without termination of the TLS connection. To support such communication, the `--enable-ssl-passthrough` flag must be part of the configuration with which the ingress controller is started. For more information, see [Command line arguments](#) (Kubernetes documentation on github).

[Back to Top \[page 599\]](#)

1.3.1.4 External Health Checking

Perform external health checks for the connectivity proxy for Kubernetes.

External health checking lets you check and monitor the health of the connectivity proxy using an external application. To achieve this, you must call one of the following URLs:

- `https://healthcheck.<config.servers.businessDataTunnel.externalHost>/healthcheck`
- `https://healthcheck.<config.servers.businessDataTunnel.externalHost>`, that will redirect the request to the first URL.

The external health checking configuration depends on the mutual TLS configuration. Currently, the connectivity proxy deployment provides three mutual TLS configuration options:

- [Mutual TLS only to the Ingress \[page 604\]](#)
- [End-to-end mutual TLS with termination of the TLS connection in the Ingress \[page 604\]](#)
- [End-to-end mutual TLS without termination of the TLS connection in the Ingress \[page 605\]](#)

For more information, see [Mutual TLS \[page 599\]](#).

Mutual TLS Only to the Ingress or End-to-End Mutual TLS with Termination of the TLS Connection in the Ingress

In these cases, you have two options to configure the external health checking:

- Reuse the TLS secret that is used for communication with the Cloud Connector. In this case, you don't need to do any extra configuration.

i Note

The certificate from the TLS secret must be issued for both the external host specified on the configuration path `<config.servers.businessDataTunnel.externalHost>` and the host `healthcheck.<config.servers.businessDataTunnel.externalHost>`. For example, if the external host is "`ingress.mycluster.com`", the certificate CN could be "`ingress.mycluster.com`" and certificate SAN must contain "`healthcheck.ingress.mycluster.com`".

- Use a specific TLS secret only for the communication with the external health checking application. In this case, there are two options to do this:
 - If you already have an appropriate secret for this purpose in the cluster, add the following configuration in the `values.yaml` file:

```
ingress:
  healthcheck:
    tls:
      secretName: <secret name>
```

i Note

The Kubernetes secret must contain the following properties, used for authentication to the Ingress resource:

- `tls.key`: Base 64-encoded private key in PEM format.
- `tls.crt`: Base 64-encoded certificate in PEM format.

- If you don't have such a Kubernetes secret, it can be automatically generated. Add the following configuration in the `values.yaml` file:

```
ingress:
  healthcheck:
    tls:
      secretName: <secret name>
      secretData:
        key: <base 64 encoded private key in PEM format>
        certificate: <base 64 encoded certificate in PEM format>
```

i Note

The certificate from the TLS secret must be issued for the host `healthcheck.<config.servers.businessDataTunnel.externalHost>`. For example, if the external host is "`ingress.mycluster.com`", then the certificate CN or SAN must contain "`healthcheck.ingress.mycluster.com`".

i Note

The certificate chain of the CA that issues the certificate from the TLS secret must be imported to the trust store of the external health checking application.

[Back to Top \[page 603\]](#)

End-to-end Mutual TLS without Termination of the TLS Connection in the Ingress

In this case, execute the following steps:

1. Download the *plugins* archive and unzip it (for more information, see [Lifecycle Management \[page 612\]](#)).
2. Navigate to the *plugins* folder and install *connectivity-certificate-plugin* by executing the following command:

```
helm plugin install connectivity-certificate-plugin
```

3. Generate a TLS secret by executing the following command:

```
helm get-connectivity-certificates <region_domain> <subaccount> <user>
<password> <namespace> <secret_name>
```

i Note

- `region_domain`: BTP region to which you are pairing the connectivity proxy.
- `subaccount`: BTP subaccount, on whose behalf the connectivity proxy is running.
- `user`: BTP subaccount user.
- `password`: Password of the BTP subaccount user.
- `namespace`: Kubernetes namespace to which the secret is created.
- `secret_name`: Name of the Kubernetes secret to be created.

4. Specify a TLS secret that will be used for the communication with the external health checking application. There are two options to do this:
 - If you already have an appropriate secret for this purpose in the cluster, add the following configuration in the `values.yaml` file:

```
ingress:
  healthcheck:
    tls:
      secretName: <secret_name>
```

i Note

The Kubernetes secret must contain the following properties, used for authentication to the Ingress resource:

- `tls.key`: Base 64-encoded private key in PEM format.
- `tls.crt`: Base 64-encoded certificate in PEM format.

- If you don't have such a Kubernetes secret, it can be automatically generated. Add the following configuration in the `values.yaml` file:

```
ingress:  
  healthcheck:  
    tls:  
      secretName: <secret name>  
      secretData:  
        key: <base 64 encoded private key in PEM format>  
        certificate: <base 64 encoded certificate in PEM format>
```

i Note

The certificate from the TLS secret must be issued for the host `healthcheck.<config.servers.businessDataTunnel.externalHost>`. For example, if the external host is "`ingress.mycluster.com`", then the certificate CN or SAN must contain "`healthcheck.ingress.mycluster.com`".

i Note

The certificate chain of the CA that issues the certificate from the TLS secret must be imported to the trust store of the external health checking application.

5. Specify the TLS secret that was generated in step 3, to be used for the communication between the Ingress controller and the connectivity proxy. Add the following configuration in the `values.yaml` file:

```
ingress:  
  healthcheck:  
    tls:  
      proxy:  
        secretName: <name of the secret generated on step 3>
```

[Back to Top \[page 603\]](#)

1.3.1.5 High Availability

Run the connectivity proxy for Kubernetes in a high-availability setup.

Content

[Overview \[page 607\]](#)

[High-Availability Modes \[page 607\]](#)

[Advantages and Drawbacks \[page 608\]](#)

[High-Availability Mode: *Path* \[page 608\]](#)

[High-Availability Mode: *Subdomain* \[page 608\]](#)

Overview

The connectivity proxy can work in an active-active high-availability setup. In this setup, there are at least two connectivity proxy instances, both running actively and simultaneously.

The main purpose of an active-active deployment is to provide high availability and allow zero-downtime maintenance as well as horizontal-scaling capabilities.

The Kubernetes service exposing the connectivity proxy pods distributes and load-balances the traffic from the workloads across all running connectivity proxy pods.

i Note

The load balancing strategy for distributing traffic to the connectivity proxy pods depends on the *kube-proxy* mode. The strategies used by the different *kube-proxy* modes are described in the [Kubernetes documentation](#). This configuration is done on cluster level.

→ Tip

[MultiAZ with Pod Anti-Affinity](#) and [PodDisruptionBudget](#) are supported out of the box in the Helm chart.

[Back to Content \[page 607\]](#)

High-Availability Modes

There are two technical options (modes) for the active-active deployment:

- *Path*
- *Subdomain*

The difference between the two modes is in the way the routing information of the current connectivity proxy (which is being used for the current connection) is passed to the Cloud Connector.

Depending on the perspective, as well as on the concrete requirements and boundaries you have, both modes may have advantages and drawbacks, so you should carefully choose the one which best suits your needs.

The number of connectivity proxy instances that run simultaneously depends on the *replicaCount* configuration in the `values.yml` file.

```
deployment:  
  replicaCount: <number_of_replicas>
```

[Back to Content \[page 607\]](#)

Advantages and Drawbacks

High-Availability Mode	Path	Subdomain
Compatibility with end-to-end mutual TLS without termination	No	Yes
Compatibility with secure and inexpensive certificates	Yes	No

[Back to Content \[page 607\]](#)

High-Availability Mode: Path

To configure the connectivity proxy for the high-availability mode *Path*, add the following configuration in the `values.yml` file:

```
config:  
  highAvailabilityMode: "path"
```

[Back to Content \[page 607\]](#)

High-Availability Mode: Subdomain

To configure the connectivity proxy for the high-availability mode *Subdomain*, add the following configuration in the `values.yml` file:

```
config:  
  highAvailabilityMode: "subdomain"
```

i Note

High-Availability mode *Subdomain* uses either wildcard certificates or SAN (Subject Alternative Name) certificates. Wildcard certificates cover all possible subdomains. However, they are insecure since when the certificate key that is installed on one subdomain gets compromised, it will be compromised on all of the subdomains it is installed on. A safer option is to use SAN certificates, although they may be expensive and require each subdomain to be specified explicitly.

[Back to Content \[page 607\]](#)

1.3.1.6 Audit Logging

Use audit logging for the connectivity proxy for Kubernetes.

At runtime, the connectivity proxy produces audit messages for the relevant auditable events. These audit messages help you gather audit data which can be useful for identifying possible attacks or unwanted access attempts. The audit messages are written on the standard output and it is up to the operator of the connectivity proxy to configure the surrounding infrastructure in a way that these audit messages are collected, stored, protected from tampering, rotated when necessary, and so on.

i Note

When written to the standard system output, the audit log messages are preceded by the prefix [AUDIT-EVENT] to distinguish them from the other logs.

Audit Messages

In the table below, the left column shows a description of the security event, the right column shows the corresponding audit message. These messages are written on behalf of the consumer tenant.

i Note

The <message> field of the audit message should be treated as case insensitive.

Operation	Audit Message
<p>[Security Event]</p> <p>Successfully opened tunnel</p> <p>from Cloud Connector to connectivity proxy.</p>	<pre>{"user":"YYY","data":{"action":"open connectivity tunnel","message":"tunnel handshake success","tunnelId":"NNN","objectName":"connectivity tunnel management","clientId":"FFF","loggedBy Class":"com.sap.core.connectivity.tunn el.server.TunnelServerHandshaker","han dshakeDetails":"success"},"tenant":"TTT"}</pre>
<p>[Security Event]</p> <p>Failed attempt to open tunnel</p> <p>from Cloud Connector to connectivity proxy.</p>	<pre>{"user":"YYY","data":{"action":"open connectivity tunnel","message":"Tunnel handshake failure","tunnelId":"NNN","objectName":"connectivity tunnel management","clientId":"FFF","loggedBy Class":"com.sap.core.connectivity.tunn el.server.TunnelServerHandshaker","han dshakeDetails":"UUU"},"tenant":"TTT"}</pre>

In the table above, there are multiple keys with UPPER CASE values. At runtime, these keys hold real scenario values:

- YYY: User executing the operation, if known. If the request is not done on behalf of a user, the value will be `fallback_for_missing_user`.
- NNN: Secure tunnel between the connectivity proxy and the Cloud Connector.
- TTT: Tenant on whose behalf the operation is executed.
- FFF: Remote client (unique identifier of a particular installation) of the Cloud Connector.
- UUU: Additional information about the event.

1.3.1.7 Integration with SAP Services

Integrate the connectivity proxy for Kubernetes with SAP services.

Connectivity Service [page 610]	Use the connectivity proxy for Kubernetes with the Connectivity service to connect to on-premise systems.
Security (Proxy Authorization) [page 611]	Perform proxy authorization for the connectivity proxy.

1.3.1.7.1 Connectivity Service

Use the connectivity proxy for Kubernetes with the Connectivity service to connect to on-premise systems.

On-premise systems are exposed via the Cloud Connector. The Cloud Connector is connected to (registered in) the SAP Connectivity service and ready to serve. The connectivity proxy is the software component to which, at runtime, the Cloud Connector connects and establishes the business data tunnel.

To use the connectivity proxy, it must be paired with the SAP Connectivity service, that is, configured to connect to the central service to which initially the Cloud Connector was connected. This enables the connectivity proxy to dynamically (on-demand) serve its purpose, that is, to securely establish business data tunnel connections to the Cloud Connector, hosted in an on-premise network next to the backend systems.

Prerequisites

You have an account on SAP BTP, Cloud Foundry environment, and your SAP BTP account user has the authorization to create service instances.

Procedure

1. Create a Connectivity service instance with service plan `connectivity_proxy`.
2. Create a service key of the created service instance.
3. Follow the instructions under [Lifecycle Management \[page 612\]](#) and set up the Kubernetes secret for pairing with the central SAP Connectivity service, providing the content of the created service key.
4. Check the [Configuration Guide \[page 618\]](#) for `config.integration.connectivityService*` parameters.

1.3.1.7.2 Security (Proxy Authorization)

Perform proxy authorization for the connectivity proxy for Kubernetes.

An important aspect of using the connectivity proxy is the proxy authorization, that is, how the access to the proxy is controlled (protected), enabling the callers to reach out to on-premise systems exposed via the Cloud Connector.

As described in [Operational Modes \[page 595\]](#), there are two major categories for consideration: The level of trust in the environment, and the *type of tenancy* (single vs multi-tenant). This helps you prevent unwanted callers and is especially useful in scenarios where calls to the proxy can originate from workloads outside of your control, or if you simply want to apply stricter rules.

The authorization of the connectivity proxy is based on OAuth and it is provided via integration with the SAP UAA (aka XSUAA) service acting as an OAuth server. For a successful authorization request, a JSON Web token (JWT) must be sent to the connectivity proxy, for which the following is valid:

- Issued by XSUAA

- Not expired
- Passes the signature verification (the connectivity proxy takes care to get the token keys from XSUAA for offline JWT verification).
- Its `client_id` (the client ID of the OAuth client which is part of the respective XSUAA service instance credentials) matches the allowed client ID in the connectivity proxy configuration.

```
config:
  servers:
    proxy:
      authorization:
        oauth:
          allowedClientId: "the client id which is allowed as JWT issuer for
proxy authorization"
```

→ Tip

The service instance used to protect the connectivity proxy can be any XSUAA service instance.

i Note

Based on the above info, the connectivity proxy can be protected with the very same service instance you use to perform the user login flow for your application. In this case, you can reuse the login token. However, to achieve separation of concerns, we recommend that you use a dedicated service instance to protect the connectivity proxy.

Multi-Tenancy and Subscriptions

An important aspect of the proxy authorization is that, when enabled, it is also used to pass the tenant context for the request to the connectivity proxy. When you fetch a token for proxy authorization, make sure you do so on behalf of the *correct* tenant, over which you want to call the connectivity proxy.

In cases of multi-tenancy, or single-tenancy where the proxy-protecting instance is created in a tenant that is *different* from the one for which the proxy is dedicated, you must *ensure subscriptions* between this tenant and the proxy-protecting instance to fetch a token on behalf of the correct tenant.

1.3.2 Lifecycle Management

Use the connectivity proxy image and the connectivity proxy Helm chart to manage the life cycle of the connectivity proxy for Kubernetes.

The connectivity proxy delivery includes the following main components:

- [Connectivity Proxy Image \[page 613\]](#): the functional component that packages all the binaries and utilities.

- [Connectivity Proxy Helm Chart \[page 613\]](#): used for configuring and managing the life cycle of the connectivity proxy via the popular Helm package manager. For more information, see [Operations via Helm \[page 614\]](#).

Connectivity Proxy Image

The connectivity proxy image is a standard Docker image containing all the required binaries for the connectivity proxy. This includes the connectivity proxy binaries themselves, a JVM (SapMachine), and other important utilities.

Connectivity Proxy Image in the Repository-Based Shipment Channel (RBSC)

The connectivity proxy image is available via the RBSC docker image repository. This requires having an S-user with associated licenses.

i Note

Contact your account manager to clarify access to the repository.

Pull information (for the latest version):

- **Registry:** 73554900100900005672.dockersrv.repositories.sap.ondemand.com
- **Repository:** com.sap.cloud.connectivity/connectivity-proxy
- **Tag:** 2.3.1
- **Authorization:** see [Manage Technical Users in SAP Repositories Management](#) (RBSC documentation).

Example:

```
docker pull 73554900100900005672.dockersrv.repositories.sap.ondemand.com/
com.sap.cloud.connectivity/connectivity-proxy:2.3.1
```

[Back to Top \[page 2\]](#)

Connectivity Proxy Helm Chart

The connectivity proxy delivery also includes a Helm chart that you can use for life cycle management. It is the recommended way to perform life cycle management. The connectivity proxy Helm provides full configuration capabilities via the standard Helm method of a `values.yaml` file (see [Configuration Guide \[page 618\]](#)).

Connectivity Proxy Helm Chart in the Repository-Based Shipment Channel (RBSC)

The connectivity proxy Helm chart is available via the RBSC Helm repository. This requires having an S-user with associated licenses.

i Note

Contact your account manager to clarify access to the repository.

Pull information (for the latest version):

- **Registry:** 73554900100900005672.helmsrv.repositories.sap.ondemand.com
- **Repository:** com.sap.cloud.connectivity/connectivity-proxy
- **Tag:** 2.3.1
- **Authorization:** see [Manage Technical Users in SAP Repositories Management](#) (RBSC documentation).

Example:

```
helm repo add connectivity https://73554900100900005672.dev.helmsrv.repositories.sap.ondemand.com --username <user> --password <pass>
helm pull connectivity/connectivity-proxy --version=2.3.1
```

[Back to Top \[page 2\]](#)

Related Information

[Operations via Helm \[page 614\]](#)

[Operations via Separate YAML Files \[page 616\]](#)

[Configuration Guide \[page 618\]](#)

[Additional Security Aspects \[page 627\]](#)

[Sizing Recommendations \[page 628\]](#)

1.3.2.1 Operations via Helm

Use the Helm chart to configure and manage the life cycle of the connectivity proxy for Kubernetes.

i Note

Out of the box, the Helm chart only supports the [NGINX Ingress Controller](#).

[Deploy \[page 615\]](#)

[Update / Upgrade / Downgrade \[page 615\]](#)

[Undeploy \[page 616\]](#)

Deploy

To deploy the connectivity proxy on a cluster that does not have the Helm chart yet, for example, in a new namespace, follow these steps:

1. Get the connectivity proxy Helm chart, as described in [Lifecycle Management \[page 612\]](#).
2. Download the list of trusted CAs for the SAP BTP region to which you want to pair the connectivity proxy. Create a Kubernetes secret from it together with a previously generated public/private TLS key pair (for the Ingress public endpoint). For generating the TLS key pair, you can use [Gardener Certificate resources](#), [openssl](#) or some other tool. Example:

```
# download the list of trusted CAs
curl https://connectivity.cf.{region_domain}/api/v1/CAs -o connectivity_ca.crt
# create a Kubernetes secret
kubectl create secret generic connectivity-tls --from-
file=ca.crt=connectivity_ca.crt --from-file=tls.key=private.key --from-
file=tls.crt=public.crt --namespace my-namespace
```

Where `private.key` is the private key and `public.crt` is the public key of a TLS certificate, generated for the Ingress public endpoint of the connectivity proxy (the one which the Cloud Connector connects to).

→ Remember

The content of `/api/v1/CAs` may change over time. Make sure you update it regularly.

3. Prepare the `values.yaml` file for your scenario, as described in [Configuration Guide \[page 618\]](#).
4. Use the Helm CLI to deploy the connectivity proxy. Example:

```
helm install connectivity-proxy ./connectivity-proxy-<version>.tgz -f
values.yaml --namespace my-namespace
```

[Back to Top \[page 614\]](#)

Update / Upgrade / Downgrade

When you have a connectivity proxy deployed on the cluster, you may want to maintain it by changing configurations and/or changing its version. To do so, follow these steps:

1. Get the connectivity proxy Helm chart, as described in [Lifecycle Management \[page 612\]](#). It can be the same version as the one currently installed or a different version, when you want to upgrade or downgrade.
2. Prepare the `values.yaml` file for your scenario, as described in [Configuration Guide \[page 618\]](#). Here you can just modify the one you used previously by applying the changes you desire.
3. Use the Helm CLI to upgrade the connectivity proxy. Example:

```
helm upgrade connectivity-proxy ./connectivity-proxy-<version>.tgz -f  
values.yaml --namespace my-namespace
```

i Note

Each Helm upgrade will result in a restart of the connectivity proxy pod(s). This is done to ensure that configuration changes are picked up immediately.

[Back to Top \[page 614\]](#)

Undeploy

If you need to remove the connectivity proxy from your cluster or from a namespace, you can do it almost completely via normal Helm tools. However, there are some additional actions required. Please follow these steps:

1. Use the Helm CLI to undeploy the connectivity proxy. Example:

```
helm uninstall connectivity-proxy --namespace my-namespace
```

2. Delete the Kubernetes secret representing the TLS certificate for the connectivity proxy public endpoint. Example:

```
kubectl delete secret connectivity-tls --namespace my-namespace
```

[Back to Top \[page 614\]](#)

1.3.2.2 Operations via Separate YAML Files

Using separate YAML files to configure and manage the life cycle of the connectivity proxy for Kubernetes.

⚠ Caution

As operating with separate YAML files that you modify and maintain is considered error prone, we do not recommend this method. Consider using [Operations via Helm \[page 614\]](#) instead. If you do need to use separate YAML files, we recommend that you generate them via Helm template.

[Deploy \[page 617\]](#)

[Update / Upgrade / Downgrade \[page 617\]](#)

[Undeploy \[page 618\]](#)

Deploy

To deploy the connectivity proxy on a cluster that does not have it yet, for example, in a new namespace, follow these steps:

1. Get the connectivity proxy YAML files via Helm template
2. Download the list of trusted CAs for the SAP BTP region to which you want to pair the connectivity proxy. Create a Kubernetes secret from it together with a previously generated public/private TLS key pair (for the Ingress public endpoint). For generating the TLS key pair, you can use [Gardener Certificate resources](#), [openssl](#) or some other tool. Example:

```
# download the list of trusted CAs
curl https://connectivity.cf.{region_domain}/api/v1/CAs -o connectivity_ca.crt
# create a Kubernetes secret
kubectl create secret generic connectivity-tls --from-
file=ca.crt=connectivity_ca.crt --from-file=tls.key=private.key --from-
file=tls.crt=public.crt --namespace my-namespace
```

Where `private.key` is the private key and `public.crt` is the public key of a TLS certificate, generated for the Ingress public endpoint of the connectivity proxy (the one which the Cloud Connector connects to).

→ Remember

The content of `/api/v1/CAs` may change over time. Make sure you update it regularly.

3. Use the `kubectl` CLI to deploy the connectivity proxy. Example:

```
kubectl apply -f my_changed_resource.yaml --namespace my-namespace
```

[Back to Top \[page 616\]](#)

Update / Upgrade / Downgrade

When you have a connectivity proxy deployed on the cluster, you may want to maintain it by changing configurations and/or changing its version. To do so, follow these steps:

1. Get the YAML files (or only those you want to modify) you used to deploy the connectivity proxy. You can also export them from the cluster via `kubectl`.
2. Make the desired changes. For example, you can modify the subaccount ID in the `config map` or the connectivity proxy version in the deployment.
3. Use the `kubectl` CLI to apply your changes. Example:

```
kubectl apply -f my_changed_resource.yaml --namespace my-namespace
```

→ Tip

You can also use `kubectl edit` to directly modify the resources on the cluster.

i Note

When updating secrets or config maps, you must restart the pod(s) to activate the changes. You can do this by running `kubectl rollout restart statefulset/connectivity-proxy`.

[Back to Top \[page 616\]](#)

Undeploy

If you need to remove the connectivity proxy from your cluster or from a namespace, follow these steps:

1. Get the YAML files you used to deploy the connectivity proxy. You can also export them from the cluster via `kubectl`.
2. Use the `kubectl` CLI to undeploy the connectivity proxy. Example:

```
kubectl delete -f ./dir_with_yaml_files --namespace my-namespace
```

→ Tip

You can also use `kubectl delete <resource type> <resource name>`.

3. Delete the Kubernetes secret representing the TLS certificate for the connectivity proxy public endpoint. Example:

```
kubectl delete secret connectivity-tls --namespace my-namespace
```

[Back to Top \[page 616\]](#)

1.3.2.3 Configuration Guide

Find an overview of configuration parameters for the connectivity proxy for Kubernetes.

Refer to the table below for the configurations available in the `values.yaml` file of the connectivity proxy.

i Note

Make sure you become familiar with the semantics, restrictions and interoperability aspects of each property before using it.

[Parameter Overview \[page 619\]](#)

[Example \[page 625\]](#)

Parameter Overview

Parameter	Description	Default
chart.nameSuffix	Custom string used to suffix your resources.	"" (empty string)
config.displayName	Display name for the connectivity proxy installation. It is displayed in the Cloud Connector UI in Cloud Connections section.	None
config.integration.connectivityService.serviceCredentialsKey	Key in the Connectivity service secret resource that holds the base64-encoded value of the connectivity proxy service key.	"service_key"
config.servers.businessDataTunnel.enableTls	Enables end-to-end mutual TLS. See Mutual TLS [page 599] for details.	false
config.servers.businessDataTunnel.externalHost	External ingress host of the business data tunnel server.	None
config.servers.businessDataTunnel.externalPort	External ingress port of the business data tunnel server.	443
config.servers.businessDataTunnel.strictSniEnabled	Flag for enabling and disabling strict SNI verification in the business data tunnel server.	true
config.servers.businessDataTunnel.port	Port on which to start the business data tunnel server.	8042
config.servers.proxy.authorization.oauth.allowedClientId	Oauth client ID that is authorized to pass through the connectivity proxy when authorization is enabled.	None
config.servers.proxy.http.allowRemoteConnections	Flag for enabling and disabling calls from outside the pod to the HTTP proxy server.	true
config.servers.proxy.http.enabled	Flag for enabling and disabling the HTTP proxy server.	false
config.servers.proxy.http.enableProxyAuthorization	Flag for enabling and disabling the HTTP proxy authorization. See Operational Modes [page 595] for details.	true
config.servers.proxy.http.port	Port on which to start the HTTP proxy server.	20003
config.servers.proxy.rfcAndLdap.allowRemoteConnections	Flag for enabling and disabling calls from outside the pod to the RFC and LDAP proxy server.	true
config.servers.proxy.socks5.allowRemoteConnections	Flag for enabling and disabling calls from outside the pod to the SOCKS5 proxy server.	true
config.servers.proxy.socks5.enabled	Flag for enabling and disabling the SOCKS5 proxy server.	true

Parameter	Description	Default
config.servers.proxy.socks5.enableProxyAuthorization	Flag for enabling and disabling the SOCKS5 proxy authorization. See Operational Modes [page 595] for details.	true
config.servers.proxy.socks5.port	Port on which to start the SOCKS5 proxy server.	20004
config.jvm.errorFilePath	Directory for JVM error logs.	"/var/log/connectivity"
config.jvm.heapDumpPath	Directory for JVM heap dumps.	"/var/log/connectivity"
config.jvm.memory.maxDirectSize	Maximum direct memory size to be set to the Java process of the connectivity proxy. If not set, the value is calculated, based on the available resources.	None
config.jvm.memory.maxHeapSize	Maximum heap size to be set to the Java process of the connectivity proxy. If not set, the value is calculated, based on the available resources.	None
config.jvm.memory.minHeapSize	Minimum heap size to be set to the Java process of the connectivity proxy. If not set, the value is calculated, based on the available resources.	None
config.highAvailabilityMode	High availability mode in which the connectivity proxy works. The possible values are off , subdomain , and path . See High Availability [page 606] for details.	"off"
config.subaccountId	ID of the subaccount, on whose behalf the connectivity proxy is running. This is mandatory only for the <i>dedicated</i> tenant mode. It is ignored when running in <i>shared</i> tenant mode.	None
config.subaccountSubdomain	Subdomain of the subaccount, on whose behalf the connectivity proxy is running. This is mandatory only for the <i>dedicated</i> tenant mode when there is at least one proxy with disabled authorization. It is ignored when running in <i>shared</i> tenant mode or if authorization is enabled for all proxies.	None
config.tenantMode	Mode in which the connectivity proxy works. The possible values are <i>dedicated</i> and <i>shared</i> . See Operational Modes [page 595] for details.	"dedicated"
secretConfig.integration.auditlogService.secretName	Name of the secret that holds the credentials for the Auditlog service. See Audit Logging [page 609] for details.	"auditlog-service-key"

Parameter	Description	Default
secretConfig.integration.auditlogService.secretData	Base64-encoded value of the service key, obtained from the Auditlog service instance. See Audit Logging [page 609] for details.	None
secretConfig.servers.businessDataTunnel.secretName	Name of the secret that is used for TLS handshake with The Ingress proxy or Cloud Connector. See Mutual TLS [page 599] for details.	None
secretConfig.servers.businessDataTunnel.secretData.caCertificate	CA certificate part of the secret that is used for TLS handshake with the Ingress proxy or Cloud Connector. See Mutual TLS [page 599] for details.	None
secretConfig.servers.businessDataTunnel.secretData.certificate	Certificate part of the secret that is used for TLS handshake with the Ingress proxy or Cloud Connector. See Mutual TLS [page 599] for details.	None
secretConfig.servers.businessDataTunnel.secretData.key	Key part of the secret that is used for TLS handshake with the Ingress proxy or Cloud Connector. See Mutual TLS [page 599] for details.	None
deployment.image.pullPolicy	One of Always , Never , IfNotPresent . For more information, see: Updating images (Kubernetes documentation).	"IfNotPresent"
deployment.image.pullSecret	Secret used for authentication against the repository.	None
deployment.image.registry	Name of the registry from which the Connectivity deployment is downloaded.	<the image repository>
deployment.image.repository	Name of the repository.	"com.sap.cloud.connectivity/connectivity-proxy"
deployment.image.tag	Version of the connectivity proxy to be deployed.	1
deployment.resources.maxFileDescriptorCount	Maximum number of file descriptors that will be used. This value should be based on the overall scenario in which the connectivity proxy is used and on the load of the proxy (user requests to the proxy).	64000
deployment.resources.limits.cpu	The <i>kubelet</i> enforces the limit so that the running container is not allowed to use more CPU resource than set as limit. If the limit is crossed, the process is throttled.	1

Parameter	Description	Default
deployment.resources.limits.memory	The <i>kubelet</i> enforces the limit so that the running container is not allowed to use more memory than set as limit. If the limit is crossed, the process is terminated with an out-of-memory (OOM) error.	1024M
deployment.resources.requests.cpu	The Kubernetes scheduler uses this information to decide which node to place the pod on. If there are no nodes with the specified amount of CPU resources, the pod won't be scheduled (and therefore not started).	0.1
deployment.resources.requests.memory	The Kubernetes scheduler uses this information to decide which node to place the pod on. If there are no nodes with the specified amount of memory, the pod won't be scheduled (and therefore not started).	256M
deployment.autoscaling.horizontal.enabled	Enables or disables the <i>Horizontal Pod Autoscaler</i> mechanism, see Horizontal Pod Autoscaler (Kubernetes documentation).	256M
deployment.autoscaling.horizontal.maxReplicaCount	Upper limit for the number of connectivity proxy replicas to which the autoscaler can scale up. It should be higher than <code>deployment.replicaCount</code> .	2
deployment.autoscaling.horizontal.metrics.cpuAverageUtilization	Target value of the average CPU metric across all connectivity proxy pods, represented as a percentage of the requested value of the CPU for the pods.	80
deployment.autoscaling.horizontal.metrics.memoryAverageUtilization	Target value of the average memory metric across all connectivity proxy pods, represented as a percentage of the requested value of the memory for the pods.	80
deployment.autoscaling.vertical.enabled	Enables or disables the <i>Vertical Pod Autoscaler</i> mechanism.	false
<div style="background-color: #f0f0f0; padding: 10px;"> <p>⚠ Caution</p> <p>To take effect, the <i>Vertical Pod Autoscaling</i> mechanism for the cluster must be enabled. See Vertical Pod Auto-Scaling for details (GitHub Gardener Kubernetes documentation).</p> </div>		

Parameter	Description	Default
deployment.autoscaling.vertical.updateMode	Mode in which the Vertical Pod Auto-scale should operate. See Vertical Pod Autoscaler Quick start for details (Github Kubernetes documentation).	"Off"
ingress.annotations	Annotations to add to the Ingress resource.	{}
ingress.healthcheck.tls.proxy.secretName	Name of the secret that is used for TLS handshake with the connectivity proxy. See External Health Checking [page 603] for details.	None
ingress.healthcheck.tls.proxy.secretData.caCertificate	CA certificate part of the secret that is used for TLS handshake with the connectivity proxy. See External Health Checking [page 603] for details.	None
ingress.healthcheck.tls.proxy.secretData.certificate	Certificate part of the secret that is used for TLS handshake with the connectivity proxy. See External Health Checking [page 603] for details.	None
ingress.healthcheck.tls.proxy.secretData.key	Key part of the secret that is used for TLS handshake with the connectivity proxy. See External Health Checking [page 603] for details.	None
ingress.healthcheck.tls.secretName	Name of the secret that is used for TLS handshake with external health checking application. See External Health Checking [page 603] for details.	None
ingress.healthcheck.tls.secretData.certificate	Certificate part of the secret that is used for TLS handshake with external health checking application. See External Health Checking [page 603] for details.	None
ingress.healthcheck.tls.secretData.key	Key part of the secret that is used for TLS handshake with external health checking application. See External Health Checking [page 603] for details.	None
ingress.timeouts.proxy.connect	Connect timeout (in seconds), for the Ingress controller.	10

Parameter	Description	Default
ingress.timeouts.proxy.read	Read timeout (in seconds), for the Ingress controller. <div style="border-left: 3px solid #e67e22; padding-left: 10px;">⚠ Caution Make sure the load balancer exposing the Ingress controller is also configured to allow longer times. For example, for NGINX with AWS, you need to add a special annotation for the LB service of the Ingress controller.</div>	120
ingress.timeouts.proxy.send	Send timeout (in seconds), for the Ingress controller.	120
ingress.tls.proxy.secretName	Name of the secret that is used for TLS handshake with the connectivity proxy. See Mutual TLS [page 599] for details.	None
ingress.tls.proxy.secretData.caCertificate	CA certificate part of the secret that is used for TLS handshake with the connectivity proxy. See Mutual TLS [page 599] for details.	None
ingress.tls.proxy.secretData.certificate	Certificate part of the secret that is used for TLS handshake with the connectivity proxy. See Mutual TLS [page 599] for details.	None
ingress.tls.proxy.secretData.key	Key part of the secret that is used for TLS handshake with the connectivity proxy. See Mutual TLS [page 599] for details.	None
ingress.tls.secretName	Name of the secret that is used for TLS handshake with the Cloud Connector.	None
ingress.tls.secretData.caCertificate	CA certificate part of the secret that is used for TLS handshake with the Cloud Connector.	None
ingress.tls.secretData.certificate	Certificate part of the secret that is used for TLS handshake with the Cloud Connector.	None
ingress.tls.secretData.key	Key part of the secret that is used for TLS handshake with the Cloud Connector.	None

[Back to Top \[page 618\]](#)

Example

⚠ Caution

The following example shows the full structure and does not represent a productive values.yaml file. Many properties listed here are mutually exclusive and should *not be used together* in real situations.

values.yaml structure example

```
chart:
  nameSuffix: "a string to append to resource names"

config:
  integration:
    auditlog:
      mode: "service"/"console"
      serviceCredentialsKey: "specifies the filename of the mounted auditlog secret"
    connectivityService:
      serviceCredentialsKey: "specifies the filename of the mounted connectivity service secret"
    servers:
      businessDataTunnel:
        enableTls: true/false
        externalHost: "myconnproxy.mycluster.com"
        externalPort: 443
        strictSniEnabled: true/false
        port: 8042
    proxy:
      authorization:
        oauth:
          allowedClientId: "the client id which is allowed as JWT issuer for proxy authorization"
      http:
        allowRemoteConnections: true/false
        enabled: true/false
        enableProxyAuthorization: true/false
        port: 20003
    rfcAndLdap:
      allowRemoteConnections: true/false
      enabled: true/false
      enableProxyAuthorization: true/false
      port: 20001
    socks5:
      allowRemoteConnections: true/false
      enabled: true/false
      enableProxyAuthorization: true/false
      port: 20004
  jvm:
    errorFilePath: "directory for JVM error logs"
    heapDumpPath: "directory for JVM heap dumps"
    memory:
      maxHeapSize: 256m
      minHeapSize: 16m
  highAvailabilityMode: "off"/"subdomain"/"path"
  subaccountId: "id of the subaccount, for which the proxy is running"
  subaccountSubdomain: "subdomain of the subaccount, for which the proxy is running"
  tenantMode: dedicated/shared

secretConfig:
  integration:
```

```

auditlogService:
  secretData: "base64 encoded auditlog service key"
  secretName: "name of the secret resource, holding the auditlog secret"
connectivityService:
  secretData: "base64 encoded connectivity_proxy service key"
  secretName: "name of the secret resource, holding the connectivity service
secret"
servers:
  businessDataTunnel:
    secretData:
      caCertificate: "base64 encoded CA certificate"
      certificate: "base64 encoded certificate"
      key: "base64 encoded private key"
    secretName: "name of the secret"

deployment:
  image:
    pullPolicy: "IfNotPresent"
    pullSecret: "name of the pull secret fot the registry"
    registry: "the official SAP image repo"
    repository: "com.sap.cloud.connectivity/connectivity-proxy"
    tag: X.Y.Z
  replicaCount: 1
  resources:
    maxFileDescriptorCount: 64000
    limits:
      cpu: 1
      memory: 1024M
    requests:
      cpu: 0.1
      memory: 256M
  autoscaling:
    horizontal:
      enabled: true/false
      maxReplicaCount: 2
    metrics:
      cpuAverageUtilization: 80
      memoryAverageUtilization: 80
    vertical:
      enabled: true/false
      updateMode: "Off"

ingress:
  annotations:
    annotation1: value1
    annotation2: value2
  healthcheck:
    tls:
      proxy:
        secretData:
          caCertificate: "base64 encoded CA certificate"
          certificate: "base64 encoded certificate"
          key: "base64 encoded private key"
        secretName: "name of the secret"
      secretData:
        certificate: "base64 encoded certificate"
        key: "base64 encoded private key"
        secretName: "name of the secret"
  timeouts:
    proxy:
      connect: 10
      read: 120
      send: 120
  tls:
    proxy:
      secretData:
        caCertificate: "base64 encoded CA certificate"
        certificate: "base64 encoded certificate"

```

```
key: "base64 encoded private key"
secretName: "name of the secret"
secretData:
  caCertificate: "base64 encoded CA certificate"
  certificate: "base64 encoded certificate"
  key: "base64 encoded private key"
secretName: "name of the secret"
```

[Back to Top \[page 618\]](#)

1.3.2.4 Additional Security Aspects

Considerations on security for the traffic flow and configuration of the connectivity proxy for Kubernetes.

Security of Traffic Flow *within* a Cluster

This section refers to the traffic flow between a workload from a Kubernetes cluster and the connectivity proxy running in the same cluster.

The traffic between a workload running in the cluster and the HTTP and LDAP/RFC proxies is not encrypted. The traffic to the SOCKS5 proxy can be SSL-encrypted if the client application initiates an SSL connection. The HTTP and LDAP/RFC proxies are disabled by default for a connectivity proxy installation. If you want to enable them, add the following configuration in the `values.yaml` file:

```
config:
  servers:
    proxy:
      rfcAndLdap:
        enabled: true
      http:
        enabled: true
```

i Note

All business data that is transmitted between the connectivity proxy in the cluster and the Cloud Connector is sent over an SSL-encrypted tunnel.

Security of Traffic Flow *from Outside* a Cluster

This section refers to the traffic flow between the Cloud Connector and the connectivity proxy running in a Kubernetes cluster.

By default, the connections between the Cloud Connector and the Ingress load balancer in the Kubernetes cluster are SSL-encrypted. If you want to enable mutual TLS for the connections from the Ingress to the

connectivity proxy or have end-to-end mutual TLS between the Cloud Connector and the connectivity proxy, follow the procedures described in [Mutual TLS \[page 599\]](#).

Security of Connectivity Proxy Configuration Data

This section refers to any security-sensitive configuration data which is required for the connectivity proxy.

The connectivity proxy installation requires some configuration data to be supplied via a Kubernetes secret.

By default, the Kubernetes secrets are stored as unencrypted base64-encoded strings and are transmitted in base64-encoded format, so they are basically accessible by persons with access to the cluster.

See the [Kubernetes documentation](#) for details on how to secure the usage of secrets in a cluster.

1.3.2.5 Sizing Recommendations

Find basic sizing guidance for the connectivity proxy for Kubernetes.

Sizing Options

The following table gives basic sizing guidance for different usage scenarios. The values listed in the **CPU** and **Memory** columns correspond to the properties you should define in the `values.yaml` file (for more information, see [Configuration Guide \[page 618\]](#)):

Size (S/M/L)	CPU	Memory
S: The expected load is small - request concurrency and size is low	<ul style="list-style-type: none">• <code>deployment.resources.requests.cpu: 0.1</code>• <code>deployment.resources.limits.cpu: 1</code>	<ul style="list-style-type: none">• <code>deployment.resources.requests.memory: 256 M</code>• <code>deployment.resources.limits.memory: 1024 M</code>
M: The expected load is medium - request concurrency and size is medium	<ul style="list-style-type: none">• <code>deployment.resources.requests.cpu: 1</code>• <code>deployment.resources.limits.cpu: 2</code>	<ul style="list-style-type: none">• <code>deployment.resources.requests.memory: 512 M</code>• <code>deployment.resources.limits.memory: 2048 M</code>
L: The expected load is large - request concurrency and size is medium or high	<ul style="list-style-type: none">• <code>deployment.resources.requests.cpu: 2</code>• <code>deployment.resources.limits.cpu: 4</code>	<ul style="list-style-type: none">• <code>deployment.resources.requests.memory: 1024 M</code>• <code>deployment.resources.limits.memory: 4096 M</code>

Relation to Operational Modes

The connectivity proxy can operate in multiple [Operational Modes \[page 595\]](#).

A major criteria is the type of tenancy used: single or multi-tenant. If multiple tenants are served, we recommend that you choose a slightly bigger size to make sure each tenant is served without precedence to any other tenant at the same time. Based on the chosen tenancy mode, the following general recommendations apply:

Tenancy Mode	Sizes to Consider
Single-Tenant	S, M
Multi-Tenant	M, L

i Note

The above-mentioned sizing recommendations are related to the connectivity proxy software component, also acting as a tunnel server to which Cloud Connector instances connect. This means the Cloud Connector must be sized properly as well, see [Sizing Recommendations \[page 296\]](#) (Cloud Connector).

→ Remember

These sizing recommendation are just a direction point. There are many factors that affect the performance of the tunneling between Cloud Connector and connectivity proxy. They are closely related to the specifics of your scenario, expected regular and intermittent load, and so on.

1.3.3 Verification and Testing

Check if the connectivity proxy for Kubernetes is operational.

Once you have installed the connectivity proxy in your cluster, you can perform the following checks to verify it is running successfully.

i Note

Before starting the checks, you must wait for a few seconds until all the components are started and can be consumed.

1. Execute the following command and verify that the status of the pod (pods) is **running**.

```
kubectl run verify-connectivity-proxy --image=nginx --restart=Never -it --rm  
-- curl -v -x "connectivity-proxy:20003" "http://virtual:1234/"
```

2. Call the external health check endpoint of the connectivity proxy to verify it is returning a successful response. You can call the endpoint in a web browser or execute the following command from the command line:

```
kubectl run verify-connectivity-proxy --image=nginx --restart=Never -it --rm  
-- curl -v -x "connectivity-proxy:20003" "http://virtual:1234/" -H "Proxy-  
Authorization: Bearer <proxy-authorization-jwt-value>"
```

⚠ Caution

If the host of the Ingress in your cluster is configured with a self-signed certificate, add the `-k` flag to curl to disable the SSL certificate verification.

3. Make sure you have connected a Cloud Connector to your cloud subaccount. For more information, see [Managing Subaccounts \[page 338\]](#).

1. If you have deployed the connectivity proxy in a **single-tenant trusted mode** and have **enabled** the HTTP proxy in the `values.yml` file, execute the following command:

```
kubectl run verify-connectivity-proxy --image=nginx --restart=Never -it --  
rm -- curl -v -x "connectivity-proxy:20003" "http://virtual:1234/"
```

2. If you have deployed the connectivity proxy in a **single-tenant non-trusted** or **multi-tenant non-trusted mode** and have **enabled** the HTTP proxy in the `values.yml` file, execute the following command:

```
kubectl run verify-connectivity-proxy --image=nginx --restart=Never -it --  
rm -- curl -v -x "connectivity-proxy:20003" "http://virtual:1234/" -H  
"Proxy-Authorization: Bearer <proxy-authorization-jwt-value>"
```

3. If you have deployed the connectivity proxy in a **single-tenant trusted mode** and have **enabled** the HTTP proxy in the `values.yml` file, execute the following command:

```
kubectl run verify-connectivity-proxy --image=nginx --restart=Never -it --  
rm -- curl -v -x "connectivity-proxy:20003" "http://virtual:1234/" -H "SAP-  
CP-Connectivity-Service-Token: <connectivity-service-jwt-value>"
```

For more information on the process of fetching a JWT, see [Consuming the Connectivity Service \[page 165\]](#).

For more information on the different operational modes, see [Operational Modes \[page 595\]](#).

If the connectivity proxy is working as expected, you get one of these responses:

- Access denied to system virtual:1234: If this was a valid request, make sure you expose the system correctly in your Cloud Connector.

ℹ Note

This response indicates that the business data from the cluster is successfully reaching your Cloud Connector, but the system `virtual:1234` is not exposed there.

- Response from your backend system if you have exposed it in the Cloud Connector.

ℹ Note

This response confirms that the business data from the cluster is successfully reaching your backend system exposed in the Cloud Connector.

If you encounter problems with any of the above steps, please refer to [Troubleshooting \[page 635\]](#) and [Recommended Actions \[page 638\]](#) for further investigation.

1.3.4 Monitoring

Check operability, scenarios and metrics of the connectivity proxy for Kubernetes.

Basic Availability Monitoring

The basic availability check is the minimal verification you can do to make sure the connectivity proxy is alive. This check only shows if the process of the connectivity proxy is running and if it is able to handle requests. This check is also what is configured as the *liveness probe* for the Kubernetes deployment resource.

You can perform this check on-demand by invoking a simple HTTP GET request to the healthcheck endpoint of the connectivity proxy. If the response is 200 *OK*, the check was *successful*. Any other response means the check *failed*. There are two ways to perform this:

- From **within the cluster**: Call `connectivity-proxy-tunnel:8042/healthcheck` and observe the result
- From **outside the cluster**: Call `https://healthcheck.<ingress host of the connectivity proxy>/healthcheck` and observe the result. See [External Health Checking \[page 603\]](#) for more details.

Scenario Monitoring

On top of the availability monitoring of the component itself, it is useful to also monitor entire scenarios. This, however, cannot be provided out of the box by the connectivity proxy as it is specific to the way you use the component. Some options you can explore for this are:

- Monitor the failure rate of requests to the connectivity proxy.
- Monitor the amount of error logs in the connectivity proxy.
- Set up a scheduled execution of a full scenario that performs end-to-end verification, including the operations done via the connectivity proxy.

Metrics

Currently, the connectivity proxy does not provide any dedicated support for metrics monitoring via [full metrics pipelines](#).

However, you may want to use the [resource metrics pipeline](#) to collect basic metrics and observe them, or configure alerts based on those metrics.

1.3.5 Using the Connectivity Proxy

Use the connectivity proxy for Kubernetes with different communication protocols and principal propagation (SSO).

[Overview \[page 632\]](#)

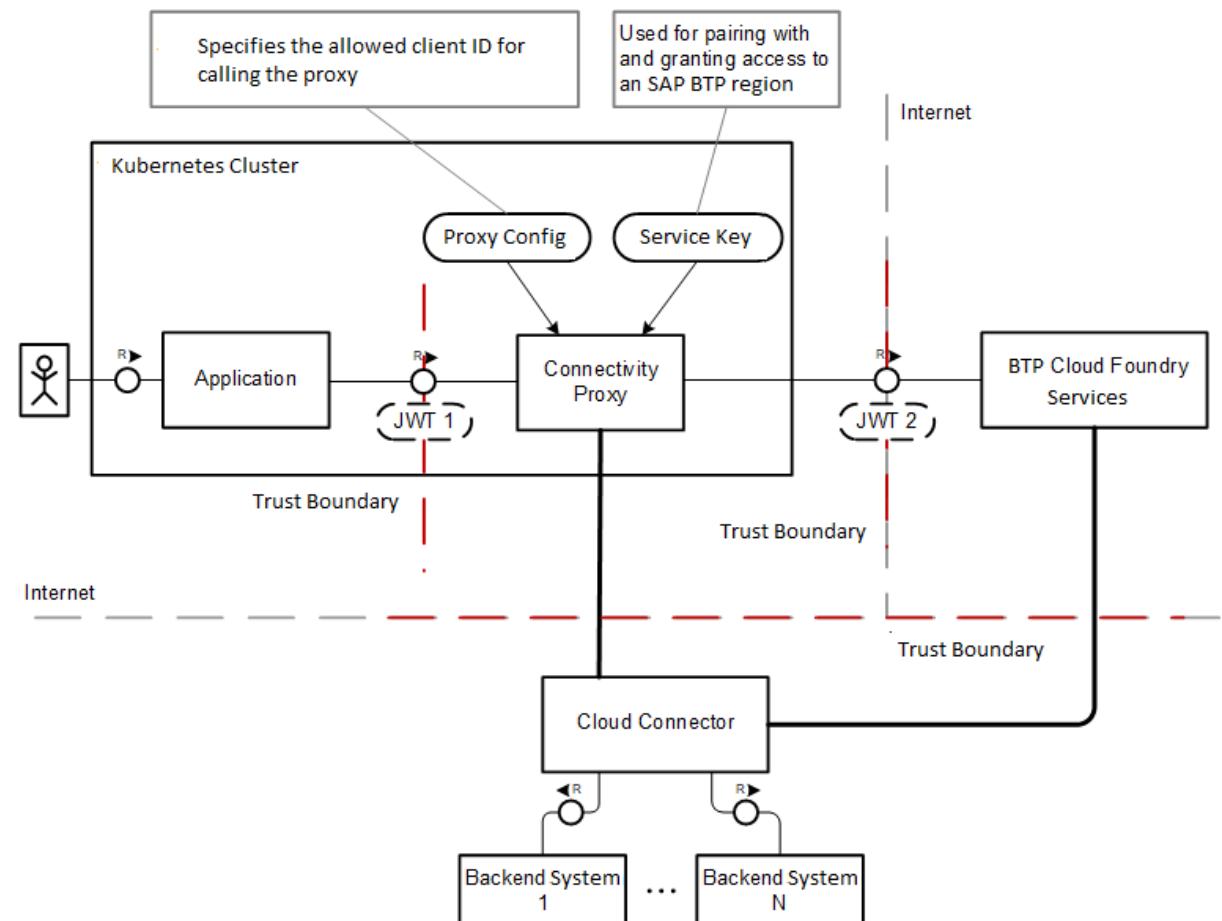
[TCP \[page 633\]](#)

[HTTP \[page 633\]](#)

[Principal Propagation \[page 634\]](#)

Overview

The connectivity proxy offers multiple proxy endpoints which are communication protocol-specific: TCP (via SOCKS5), HTTP, RFC (invoking ABAP functions) and LDAP. Depending on the operational mode, the usage may involve technical authorization when connecting to the proxy, see [Operational Modes \[page 595\]](#).



Basically, the application needs to connect to the target on-premise system, and select the respective host and port as a protocol-standard proxy (with optional authorization) to get the request routed to the target on-premise system via the Cloud Connector.

i Note

- The tunnel channel (between Cloud Connector and connectivity proxy) always uses TLS, that is, it is encrypted.
- The actual connection from the application to the connectivity proxy may be encrypted or not, depending on the exact scenario used.

i Note

If a TLS connection is attempted by the application, the SOCKS5 proxy endpoint must be used.

- The actual connection from the Cloud Connector towards the on-premise system is established and controlled in the Cloud Connector. As a result, TLS can be enabled or disabled, regardless of the connection from the application to the connectivity proxy.

[Back to Top \[page 2\]](#)

TCP

TCP connectivity is achieved via the [SOCKS5](#) proxy protocol. As the authorization is based on OAuth, we provide a custom authorization scheme for SOCKS5 that lets the application pass the required OAuth tokens and establish technical authorization with the connectivity proxy. For more information, see [Using the TCP Protocol for Cloud Applications \[page 182\]](#).

[Back to Top \[page 2\]](#)

HTTP

Uses a standard HTTP proxy, just like using a corporate proxy to reach out to the Internet. For more information, see [Authentication to the On-Premise System \[page 174\]](#).

```
curl --proxy "connectivity-proxy-host:20003" /  
http://my-virtual-host:1234/my-path#my-fragment?my-query=my-value /  
--header "Proxy-Authorization: Bearer <token>"
```

[Back to Top \[page 2\]](#)

Principal Propagation

Principal propagation, also known as *user propagation*, lets you perform single sign-on (SSO) authentication of the cloud user towards an on-premise system.

The cloud user identity is passed as a token represented by a JSON Web token (JWT). It is forwarded via the connectivity proxy to the Cloud Connector, which validates and further processes it to establish SSO with the on-premise system.

For more information, see [Authenticating Users against On-Premise Systems \[page 349\]](#) and [Set Up Trust for Principal Propagation \[page 351\]](#).

As of connectivity proxy release 2.1.1, support for principal propagation with IAS tokens is added.

Prerequisites:

- Cloud Connector 2.13 (or higher) must be used.
- The Cloud Connector must be connected to a subaccount, that is, configured with the IAS tenant issuing the tokens (for more information, see [Establish Trust and Federation Between UAA and Identity Authentication](#)).

Principal propagation is supported for both HTTP and RFC communication protocols and can be used in all [Operational Modes \[page 595\]](#) of the connectivity proxy:

- **Single-tenant in trusted environment**

The user should be propagated via `SAP-Connectivity-Authentication` header.

- **Multi-tenant in trusted environment**

The user should be propagated only via `SAP-Connectivity-Authentication` header.

The `SAP-CP-Connectivity-Service-Token` header *must* be sent for handling the tenant context.

- **Single or multi-tenant in non-trusted environment**

The user should be propagated via either `Proxy-Authorization` or `SAP-Connectivity-Authentication` header, exclusively. If `Proxy-Authorization` is used, it must hold a JWT that results from a token exchange (see [JWT Bearer Grant Type](#)), to include both the context for technical access toward the Connectivity Proxy as well as the user identity.

i Note

If IAS tokens are used, they can *only* be provided via the `SAP-Connectivity-Authentication` header.

i Note

If in a **non-trusted environment** the user context is provided via the `Proxy-Authorization` header, the `SAP-Connectivity-Authentication` *must not* be sent.

For more information, see also [XSUAA Token Client and Token Flow API](#).

[Back to Top \[page 2\]](#)

1.3.6 Troubleshooting

Find procedures to troubleshoot issues with the connectivity proxy for Kubernetes.

[Get Logs of the Connectivity Proxy \[page 635\]](#)

[Changing Log Level\(s\) of the Connectivity Proxy \[page 635\]](#)

[Viewing Logger\(s\) of the Connectivity Proxy \[page 636\]](#)

[Troubleshooting the Cloud Connector \[page 637\]](#)

[Common Issues and Solutions \[page 637\]](#)

Get Logs of the Connectivity Proxy

As the connectivity proxy workload is represented as a [standard Kubernetes StatefulSet](#), fetching the logs is done in the standard Kubernetes way. Example via `kubectl`:

```
kubectl logs statefulset/connectivity-proxy
```

[Back to Top \[page 2\]](#)

Changing Log Level(s) of the Connectivity Proxy

When the default logging level is not sufficient for debugging the issue you are facing, you can change the log level to get more insight about the problem.

i Note

Changing a log level to something more verbose will have a negative impact on the performance of the connectivity proxy. Thus, we recommend that you do not keep such a log level for a long period of time.

Changing a log level is done without any downtime and requires no restarts. All you need to do is invoke a simple command on a pod of the connectivity proxy. Here are some examples:

Put all loggers on DEBUG (full command)

```
kubectl exec <pod> -it -- connectivity-proxy-operations logging change-log-level DEBUG
```

Put all loggers on DEBUG (shortcut)

```
kubectl exec <pod> -it -- change-log-level DEBUG
```

Put only some loggers on DEBUG (shortcut)

```
kubectl exec <pod> -it -- change-log-level com.sap.core.connectivity.tunnel.k8s  
DEBUG # Will affect all loggers with names, starting with  
com.sap.core.connectivity.tunnel.k8s
```

Restore log level to default (shortcut)

```
kubectl exec <pod> -it -- change-log-level INFO
```

For more information, check the help text of the command:

```
kubectl exec <pod> -it -- change-log-level help
```

[Back to Top \[page 2\]](#)

Viewing Logger(s) of the Connectivity Proxy

List all loggers (full command)

```
kubectl exec <pod> -it -- connectivity-proxy-operations logging list-loggers
```

List all loggers (shortcut)

```
kubectl exec <pod> -it -- list-loggers  
# or  
kubectl exec <pod> -it -- get-loggers
```

List only some specific loggers (shortcut)

```
kubectl exec <pod> -it -- list-loggers com.sap.core.connectivity.tunnel.k8s #  
Will show all loggers with names, starting with  
com.sap.core.connectivity.tunnel.k8s
```

For more information, check the help text of the command:

```
kubectl exec <pod> -it -- list-loggers help
```

[Back to Top \[page 2\]](#)

Troubleshooting the Cloud Connector

For troubleshooting the Cloud Connector, see [Troubleshooting \[page 559\]](#) (Cloud Connector).

[Back to Top \[page 2\]](#)

Common Issues and Solutions

Issue	Solution
You use the HTTP proxy (port 20003) and get a 405 response.	Make sure the URL you are calling is <code>http://<virtual host>:<virtual port></code> and not <code>https://<virtual host>:<virtual port></code> .
You use the HTTP proxy (port 20003) and get a 407 response (in a non-trusted environment / proxy authorization turned on).	Make sure you set the <code>Proxy-Authorization</code> header with a value in the format " <code>Bearer <valid JWT token></code> ".
You use the HTTP proxy (port 20003) and get a 503 response stating that there is no Cloud Connector for your subaccount.	<ul style="list-style-type: none">• Make sure the Cloud Connector you are targeting is still connected.• Make sure its location ID matches the one used in the request.• Make sure the Cloud Connector is connected to the same subaccount, on whose behalf the token for the request is issued.
Out of nowhere, the connectivity proxy stops working / monitors indicate a failure (considered an outage if it happens in production).	See Recommended Actions [page 638] .
An SSL error is shown in the Cloud Connector when trying to send a request through the connectivity proxy.	Assuming the SSL error occurs during the call from the Cloud Connector to the public endpoint of the connectivity proxy, there are two options: <ul style="list-style-type: none">• Error is due to a connection termination during the SSL handshake. In this case, check your intermediate components (proxies, firewalls, etc.).• The JVM on which the Cloud Connector is running does not trust the TLS certificate of the public endpoint of the connectivity proxy. In this case, import the certificate (or its CA) into the trust store of the JVM.

[Back to Top \[page 2\]](#)

Related Information

[Recommended Actions \[page 638\]](#)

1.3.6.1 Recommended Actions

Find procedures to resolve an outage of the connectivity proxy for Kubernetes functionality.

⚠ Caution

Before performing any of the steps below, make sure there is really an outage of the connectivity proxy and not just a general problem on the entire cluster.

[Pre-Intervention Steps \[page 638\]](#)

[Recovery Attempt \[page 639\]](#)

[Request Help from SAP \[page 640\]](#)

[Request Root Cause Analysis \(RCA\) \[page 641\]](#)

Pre-Intervention Steps

Before doing any restarts or modifications, it is important that you collect all relevant information.

1. Check the status of the connectivity proxy pods and collect the outcome. Example via `kubectl`:

```
kubectl get pods | grep 'connectivity-proxy'
```

2. Perform basic availability monitoring from *within* the cluster, as described in [Monitoring \[page 631\]](#), and collect the outcome. This can be done from an existing container or by spinning up a container for the check. Example:

```
kubectl run perform-hc --image=curlimages/curl -it --rm --restart=Never --  
curl -vvv 'connectivity-proxy-tunnel:8042/healthcheck'
```

3. Perform basic availability monitoring from *outside* the cluster, as described in [Monitoring \[page 631\]](#), and collect the outcome. This can be done from your browser or via REST clients like cUrl or Postman.

Example:

```
curl -vvv 'https://healthcheck.connectivitytunnel.ingress.mycluster.com/  
healthcheck'
```

4. Collect the logs of the connectivity proxy (see [Troubleshooting \[page 635\]](#)).
5. Proceed to **Recovery Attempt**.

[Back to Top \[page 638\]](#)

Recovery Attempt

The exact action to take here depends on the check result in steps 2 and 3 of section [Pre-Intervention Steps \[page 638\]](#). Choose one of the four options below, according to the outcome of your checks.

→ Tip

Check if the cause of the outage might be insufficient resources. For more information, see [Sizing Recommendations \[page 628\]](#). If this is the possible cause, try scaling the connectivity proxy vertically and/or horizontally (see [Configuration Guide \[page 618\]](#)).

- Option 1: [Check Succeeds from within the Cluster and Fails from outside the Cluster \[page 639\]](#)
- Option 2: [Check Fails from within the Cluster and Succeeds from outside the Cluster \[page 639\]](#)
- Option 3: [Check Fails from within the Cluster and Fails from outside the Cluster \[page 640\]](#)
- Option 4: [Check Succeeds from within the Cluster and Succeeds from outside the Cluster \[page 640\]](#)

Check Succeeds from within the Cluster and Fails from outside the Cluster

This indicates some sort of issue with the Ingress configuration. Some possible reasons:

- TLS certificate has expired.
- Something went wrong with the Ingress controller.

Such a situation is likely not an issue with the connectivity proxy. Next steps should be to stop following the steps here and shift focus towards the Ingress configuration and Ingress controller.

[Back to Recovery Attempt \[page 639\]](#)

[Back to Top \[page 638\]](#)

Check Fails from within the Cluster and Succeeds from outside the Cluster

This indicates some sort of issue with the exposure of the connectivity proxy to internal pods. Some possible reasons:

- Some unwanted network policy came into effect, preventing calls to the connectivity proxy from where you are executing them.

Such a situation is likely not an issue with the connectivity proxy. Next steps should be to stop following the steps here and shift focus towards cluster configurations and the network policies that affect access to the connectivity proxy.

[Back to Recovery Attempt \[page 639\]](#)

[Back to Top \[page 638\]](#)

Check Fails from within the Cluster and Fails from outside the Cluster

This indicates that the connectivity proxy itself is indeed having issues. Please perform the following steps:

1. Restart the connectivity proxy deployment. Example via kubectl:

```
kubectl rollout restart statefulset/connectivity-proxy
```

2. Collect logs from the connectivity proxy after the restart completes.
3. Check if outage is still ongoing:
 1. If no, issue is resolved, proceed with [Request Root Cause Analysis \(RCA\) \[page 641\]](#).
 2. If yes, issue is not resolved, proceed with [Request Help from SAP \[page 640\]](#).

[Back to Recovery Attempt \[page 639\]](#)

[Back to Top \[page 638\]](#)

Check Succeeds from within the Cluster and Succeeds from outside the Cluster

This indicates that the connectivity proxy is currently considered operational, however it might still have trouble when used for real scenarios (depends on how you detect the outage).

1. Check if the outage is still ongoing:
 - If no, issue is resolved, proceed with [Request Root Cause Analysis \(RCA\) \[page 641\]](#).
 - If yes, issue is not resolved, proceed with the next step.
2. Restart the connectivity proxy deployment. Example via kubectl:

```
kubectl rollout restart statefulset/connectivity-proxy
```

3. Collect logs from the connectivity proxy after the restart completes.
4. Check if outage is still ongoing:
 - If no, issue is resolved, proceed with [Request Root Cause Analysis \(RCA\) \[page 641\]](#).
 - If yes, issue is not resolved, proceed with [Request Help from SAP \[page 640\]](#).

[Back to Recovery Attempt \[page 639\]](#)

[Back to Top \[page 638\]](#)

Request Help from SAP

If you cannot resolve the issue and require help from SAP, follow this procedure:

1. Open an incident on the support component (see [Connectivity Support \[page 644\]](#)) for the connectivity proxy (with the appropriate priority and impact stated).
2. Provide all the collected information in the incident, including all the logs, timestamps of the events that occurred, summary of the taken actions, version of the connectivity proxy you are using, and so on.
3. Engage your SAP contacts to help with this.

4. Continue working in parallel to identify as much information as possible or to find a temporary measure to mitigate the outage.

[Back to Top \[page 638\]](#)

Request Root Cause Analysis (RCA)

Once the issue is resolved, the next step is figuring out what exactly caused the issue and if there is something that can be done to prevent it from happening in the future. Follow this procedure for requesting RCA for the issue you experienced:

1. Open an incident on the support component for the connectivity proxy (see [Connectivity Support \[page 644\]](#)).
2. Provide all the collected information in the incident, including all the logs, timestamps of the events that occurred, summary of the taken actions, version of the connectivity proxy you are using, and so on.
3. The incident will be handled according to the SAP incident SLAs.

[Back to Top \[page 638\]](#)

1.3.7 Frequently Asked Questions

Answers to the most common questions about the connectivity proxy for Kubernetes.

When using one of the two untrusted operational modes, what is the purpose of the allowedClientId property? What token should I provide in the Proxy-Authorization header?

The `Proxy-Authorization` header serves as a way for workloads calling the proxy to authenticate against it. To do this, they need to provide this header with a JSON Web token (JWT) as value.

To accept the JWT, it must be issued by the OAuth client, specified via the `allowedClientId`. By configuring the `allowedClientId`, you determine which OAuth client protects the proxy endpoints.

The OAuth client must be XSUAA-based.

How is the Proxy-Authorization header verified?

There are several aspects that are being verified:

- The signature is being verified by calling XSUAA to get the public keys for the tenant, on behalf of which the JWT is issued, and using those keys to check if it is valid.
- The token validity is also verified and expired tokens are rejected.
- Also, we allow only tokens issued by a specified client ID (via the `allowedClientId` configuration).

Why do I need the connectivity:connectivity_proxy service key?

The connectivity proxy is a distributed software component that needs to connect to an instance of the central Connectivity service to function.

This pairing is achieved via a `connectivity:connectivity_proxy` service key, which contains both routing information and credentials for this pairing.

Why do I need a public endpoint for the connectivity proxy?

To preserve the integrity of an on-premise landscape and not expose anything from there to the Internet, the flow for establishing the connection between connectivity proxy and Cloud Connector is initiated by the Cloud Connector.

The public endpoint is used by the Cloud Connector to call the connectivity proxy and enable the data exchange.

What is the relation of the connectivity proxy to the Connectivity service in SAP BTP?

The connectivity proxy is a distributed component that must be paired to an instance of the Connectivity service in SAP BTP in order to function.

Cloud Connectors would still connect to the Connectivity service on SAP BTP and the connectivity proxy will make use of those Cloud Connectors via the established pairing.

What is the relation of the connectivity proxy to the Destination service in SAP BTP?

There is no dependency or tight integration.

You can use the Destination service to store and retrieve on-premise destination configurations which can then be used to construct a request to on-premise systems through the connectivity proxy.

Are there any client libraries that I can use with the connectivity proxy?

Please check [Using the Connectivity Proxy \[page 632\]](#).

When do I need a subscription to the connectivity:connectivity_proxy instance?

Please check [Connectivity Service \[page 610\]](#).

Can I port a cloud SDK application from the Cloud Foundry environment to Kubernetes and use the connectivity proxy?

It is a bit clunky, but yes.

If you set up the connectivity proxy in an untrusted operational mode, the way the SDK works is well suited for it.

However, since the SDK is created with the Cloud Foundry environment in mind, you would need to simulate the VCAP_SERVICES environment to get it working.

Is there an equivalent to the lite plan from the Cloud Foundry environment? Is the lite plan relevant for the connectivity proxy?

The lite plan is only relevant for the Cloud Foundry environment.

There is no lite plan for the connectivity proxy. Instead, you use an XSUAA-based OAuth client of your choice to protect the connectivity proxy, and use it to issue tokens.

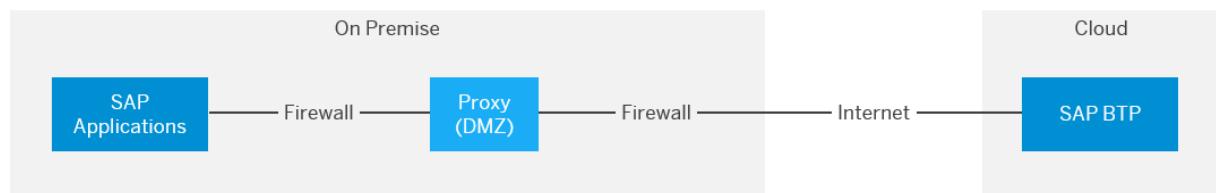
1.4 Connectivity via Reverse Proxy

An alternative approach compared to the SSL VPN solution that is provided by the Cloud Connector is to expose on-premise services and applications via a reverse proxy to the Internet. This method typically uses a reverse proxy setup in a customer's "demilitarized zone" (DMZ) subnetwork. The reverse proxy setup does the following:

- Acts as a mediator between SAP BTP and the on-premise services
- Provides the services of an Application Delivery Controller (ADC) to, for example, encrypt, filter, route, or check inbound traffic

The figure below shows the minimal overall network topology of this approach.

On-premise services that are accessible via a reverse proxy are callable from SAP BTP like other HTTP services available on the Internet. When you use destinations to call those services, make sure the configuration of the ProxyType parameter is set to [Internet](#).



Advantages

Depending on your scenario, you may benefit from the reverse proxy:

- Network infrastructure (such as a reverse proxy and ADC services): since it already exists in your network landscape, you can reuse it to connect to SAP BTP. There's no need to set up and operate new components on your (customer) side.
- A reverse proxy is independent of the cloud solution you are using.

- It acts as single entry point to your corporate network.

Disadvantages

- The reverse proxy approach leaves exposed services generally accessible via the Internet. This makes them vulnerable to attacks from anywhere in the world. In particular, **Denial-of-Service attacks** are possible and difficult to protect against. To prevent attacks of this type and others, you must implement the highest security in the DMZ and reverse proxy. For the productive deployment of a hybrid cloud/on-premise application, this approach usually requires intense involvement of the customer's IT department and a longer period of implementation.
- If the reverse proxy allows filtering, or restricts accepted source IP addresses, you can set only one IP address to be used for all SAP BTP outbound communications.
A reverse proxy does not exclusively restrict the access to cloud applications belonging to a customer, although it does filter any callers that are not running on the cloud. Basically, any application running on the cloud would pass this filter.
- The SAP-proprietary RFC protocol is supported only if WebSocket RFC can be used for communication with the ABAP system. WebSocket RFC is available as of S/4HANA release 1909. A cloud application cannot call older on-premise ABAP systems directly without using application proxies on top of ABAP in between.
- No easy support of principal propagation authentication, which lets you forward the cloud user identity to on-premise systems.
- You cannot implement projects close to your line of business (LoB).

i Note

Using the Cloud Connector mitigates all of these issues. As it establishes the SSL VPN tunnel to SAP BTP using a reverse invoke approach, there is no need to configure the DMZ or external firewall of a customer network for inbound traffic. Attacks from the Internet are not possible. With its simple setup and fine-grained access control of exposed systems and resources, the Cloud Connector allows a high level of security and fast productive implementation of hybrid applications. It also supports multiple application protocols, such as HTTP and RFC.

1.5 Connectivity Support

Support information for SAP BTP Connectivity and the Cloud Connector.

Troubleshooting

Locate the problem or error you have encountered and follow the recommended steps:

- [Frequently Asked Questions \[page 582\]](#) (Cloud Connector)

- [Operations \[page 512\]](#) (Cloud Connector)
- [Cloud Connectivity: Guided Answers](#)
- [Getting Support](#) (SAP Support Portal, SAP BTP community)

SAP Support Information

If you cannot find a solution to your issue, collect and provide the following specific, issue-relevant information to SAP Support:

- The Java EE code that throws an error (if any)
- A screenshot of the error message for the failed operation or the error message from the `HttpResponse` body
- Access credentials for your cloud or on-premise location

You can submit this information by creating a customer ticket in the SAP CSS system using the following components:

Component	Purpose
Connectivity Service	
BC-CP-CON	For <i>cloud-side</i> issues with cloud to on-premise connectivity, where: <ul style="list-style-type: none"> • The environment is unknown or • The issue is not related to a specific environment
BC-CP-CON-CF	For <i>cloud-side</i> issues with cloud to on-premise connectivity in the SAP BTP Cloud Foundry environment.
BC-CP-CON-S4HC	For <i>cloud-side</i> issues with cloud to on-premise connectivity in an S/4HANA Cloud system.
BC-CP-CON-K8S-PROXY	For <i>cloud-side</i> issues with cloud to on-premise connectivity in a Kubernetes cluster (or Kubernetes-based product), using the connectivity proxy software component.
BC-CP-CON-ABAP	For <i>cloud-side</i> issues with cloud to on-premise connectivity in the SAP BTP ABAP environment .
BC-NEO-CON	For <i>cloud-side</i> issues with cloud to on-premise connectivity in the SAP BTP Neo environment .
Destinations	
BC-CP-DEST	For issues with destination configurations , where: <ul style="list-style-type: none"> • The environment is unknown or • The issue is not related to a specific environment
BC-CP-DEST-CF	For general issues with the Destination service in the SAP BTP Cloud Foundry environment , like: <ul style="list-style-type: none"> • REST API • Instance creation, etc.

Component	Purpose
BC-CP-DEST-CF-CLIBS	For client library issues with the Destination service in the SAP BTP Cloud Foundry environment.
BC-CP-DEST-CF-TOOLS	For issues with the management of destination configurations via tools like the SAP BTP cockpit (Cloud Foundry environment).
BC-CP-DEST-NEO	For issues with destination configurations or: <ul style="list-style-type: none"> • Management tools • Client libraries, etc. related to destinations in the SAP BTP Neo environment .
Cloud Connector	
BC-MID-SCC	For connectivity issues related to installing and configuring the Cloud Connector , configuring tunnels, connections, and so on.

If you experience a more serious issue that cannot be resolved using only traces and logs, SAP Support may request access to the Cloud Connector. Follow the instructions in these SAP notes:

- To provide access to the Administration UI via a browser, see [592085](#).
- To provide SSH access to the operating system of the Linux machine on which the connector is installed, see [1275351](#).

Related Information

[Release and Maintenance Strategy \[page 646\]](#)

1.5.1 Release and Maintenance Strategy

Find information about SAP BTP Connectivity releases, versioning and upgrades.

Release Cycles

Updates of the Connectivity service are published as required, within the regular, bi-weekly SAP BTP release cycle.

New releases of the Cloud Connector are published when new features or important bug fixes are delivered, available on the [Cloud Tools](#) page.

Cloud Connector Versions

Cloud Connector versions follow the <major>. <minor>. <micro> versioning schema. The Cloud Connector stays fully compatible within a major version. Within a minor version, the Cloud Connector will stay with the same feature set. Higher minor versions usually support additional features compared to lower minor versions. Micro versions generally consist of patches to a <master>. <minor> version to deliver bug fixes.

For each supported major version of the Cloud Connector, only one <major>. <minor>. <micro> version will be provided and supported on the Cloud Tools page. This means that users must upgrade their existing Cloud Connectors to get a patch for a bug or to make use of new features.

Cloud Connector Upgrade

New versions of the Cloud Connector are announced in the [Release Notes](#) of SAP BTP. We recommend that Cloud Connector administrators regularly check the release notes for Cloud Connector updates. New versions of the Cloud Connector can be applied by using the Cloud Connector upgrade capabilities. For more information, see [Upgrade \[page 578\]](#).

i Note

We recommend that you first apply upgrades in a test landscape to validate that the running applications are working.

There are no manual user actions required in the Cloud Connector when the SAP BTP is updated.

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.

