SAP BTP Connectivity

Generated on: 2023-05-19 11:34:05 GMT+0000

SAP BTP Connectivity | Cloud

PUBLIC

Original content: https://help.sap.com/docs/CP_CONNECTIVITY/cca91383641e40ffbe03bdc78f00f681?locale=en-US&state=PRODUCTION&version=Cloud

Warning

Connectivity

SAP BTP Connectivity: overview, features, restrictions.

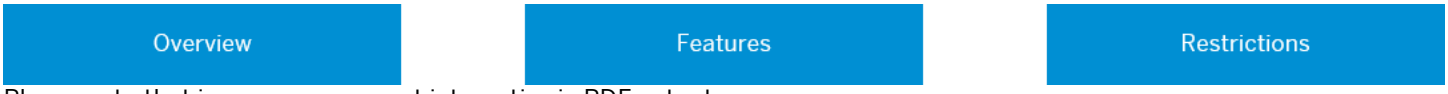
ℹ Note

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

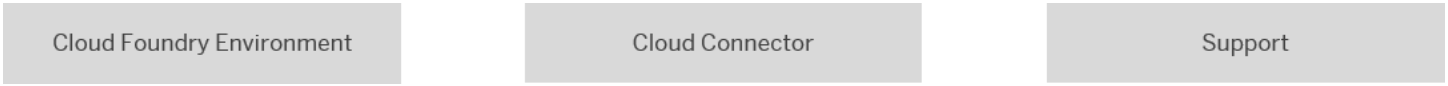
Hover over the elements for a description. Click an element for more information.



Please note that image maps are not interactive in PDF output.

In this Guide

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

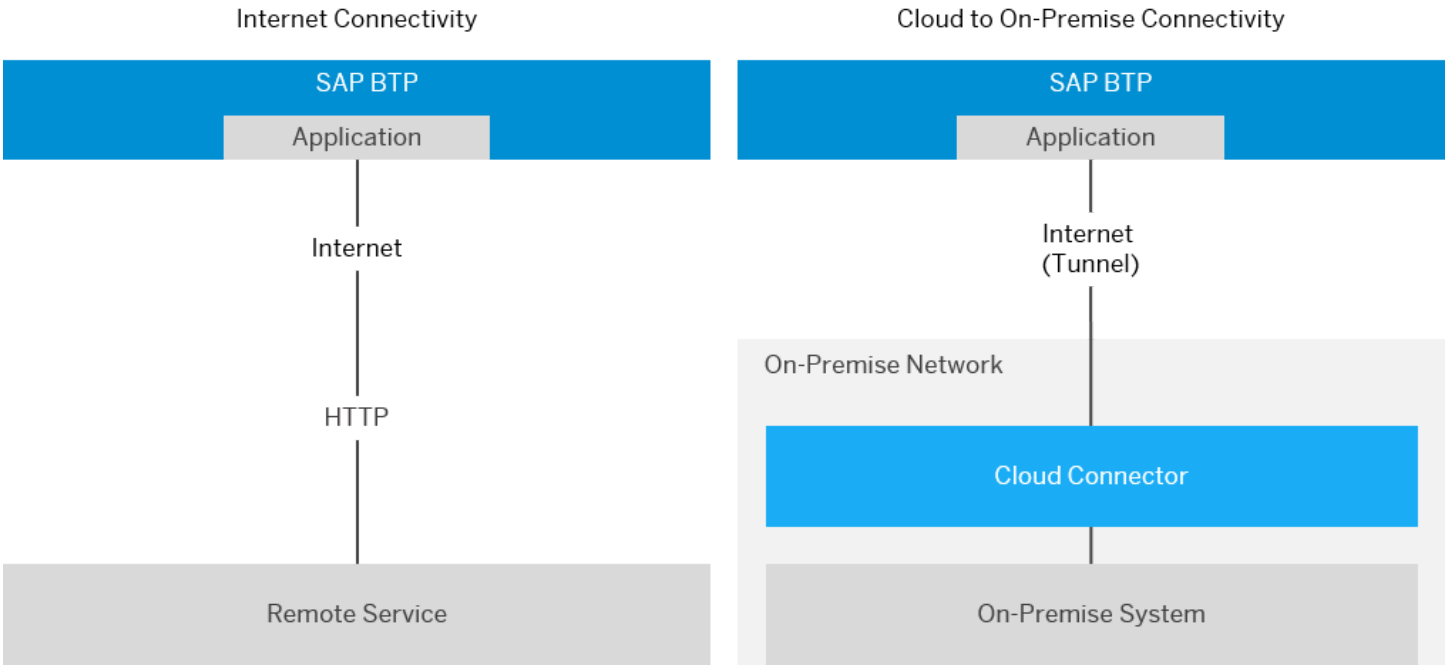


Please note that image maps are not interactive in PDF output.

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 (on-premise connections) or the respective API endpoints of your SAP BTP region (Internet connections).

Back to Content

Features

SAP BTP Connectivity supports the following protocols and scenarios:

Protocol	Scenario
HTTP(S)	Exchange data between your cloud application and Internet services or on-premise systems. • Create and configure HTTP destinations to make Web connections. • Connect to on-premise systems via HTTP, using the Cloud Connector.

Protocol	Scenario
RFC	Invoke on-premise ABAP function modules via RFC. • Create and configure RFC destinations. • Make connections to back-end systems via RFC, using the Cloud Connector.
TCP	Access on-premise systems via TCP-based protocols using a SOCKS5 proxy.

Back to Content

Restrictions

General

Protocols

Cloud Foundry Environment

Cloud Connector

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	• 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

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

Cloud Foundry Environment

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

Back to Restrictions

Cloud Connector

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

Back to Restrictions

Back to Content

Related Information

Connectivity in the Cloud Foundry Environment
Cloud Connector
Connectivity via Reverse Proxy
Connectivity Support
Connectivity Proxy for Kubernetes
Transparent Proxy for Kubernetes

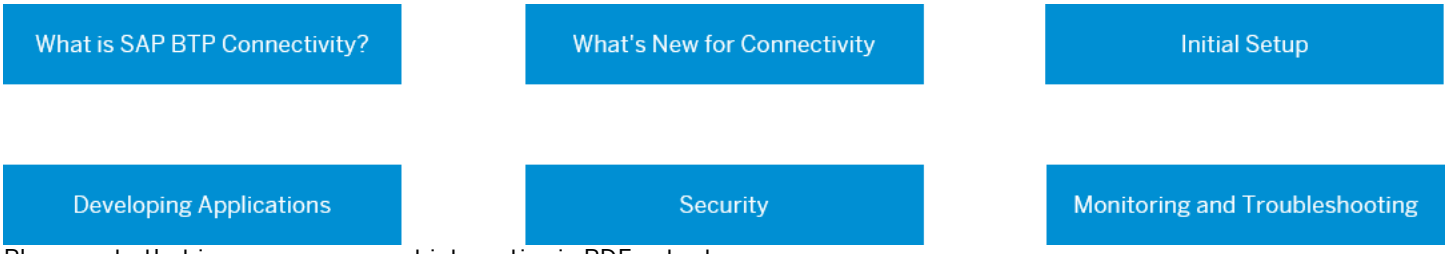
Connectivity in the Cloud Foundry Environment

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.



Please note that image maps are not interactive in PDF output.

Related Information

Resilience Recommendations

What Is SAP BTP Connectivity?

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

SAP BTP Connectivity lets you connect your SAP BTP applications to other Internet resources, or to your on-premise systems running in isolated networks. It provides an extensive set of features to choose different connection types and authentication methods. Using its configuration options, you can tailor access exactly to your needs.

ℹ Note

You can use SAP BTP Connectivity for the Cloud Foundry environment and for the Neo environment. 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

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

Please note that image maps are not interactive in PDF output.

Features

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

Use Cases

- 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.
- 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

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

Technical Roles

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 Administration.
- *Application developers* - develop a connectivity-enabled SAP BTP application by consuming the Connectivity service and/or the Destination service, see Developing Applications.
- *IT administrators* - set up the connectivity to SAP BTP in your on-premise network, using the Cloud Connector.

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]

Technical Roles [Feature Set B]

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 subaccount (Cloud Connector)	One of these roles: • ***Global Account*** member See Add Members to Your Global Account. • ***Security Administrator*** (must be Global Account member or Cloud Foundry Org/Space member) See Managing Security Administrators in Your Subaccount [Feature Set A].
	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)	
Subaccount	**View Cloud Connectors** connected to a subaccount (cockpit)	A Cloud Foundry org role containing the permission `readSCCTunnels`, for example, the role `Org Manager`. **ℹ Note** As a prerequisite, a Cloud Foundry org must be available.
Service instance	**Manage destinations** (all CRUD operations) on service instance level (cockpit)	One of these roles: • ***Org Manager*** • ***Space Manager*** • ***Space Developer*** See User and Member Management.
	View destinations (read operations) on service instance level (cockpit)	
	Manage certificates (all CRUD operations) on service instance level (cockpit)	
	View certificates (read operations) on service instance level (cockpit)	

Back to Technical Roles

Back to User Roles

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]

Default Role Collections [Feature Set B]

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
	Manage destinations (all CRUD operations) on subaccount level (cockpit)	One of these roles: • Destination Administrator • Destination Configuration Administrator
	View destinations (read operations) on subaccount level (cockpit)	One of these roles: • Destination Viewer • Destination Configuration Viewer
	Manage certificates (all CRUD operations) on subaccount level (cockpit)	One of these roles: • Destination Administrator • Destination Certificate Administrator
	View certificates (read operations) on subaccount level (cockpit)	One of these roles: • Destination Viewer • Destination Certificate Viewer
	Generate or renew the subaccount key pair for trust management (cockpit)	One of these roles: • Destination Administrator • Destination Subaccount Trust Administrator

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
	Download the subaccount key pair for trust management (cockpit)	One of these roles: • Destination Viewer • Destination Subaccount Trust Viewer
Subaccount	**View Cloud Connectors** connected to a subaccount (cockpit)	A role containing the permission `readSCCTunnels`, for example, the predefined role `Cloud Connector Administrator`.
Service instance	**Manage destinations** (all CRUD operations) on service instance level (cockpit)	One of these roles: • Destination Administrator • Destination Configuration Administrator • Destination Administrator Instance • Destination Configuration Instance Administrator ***plus*** one of these roles: • ***Org Manager*** • ***Space Manager*** • ***Space Developer*** See User and Member Management.
	View destinations (read operations) on service instance level (cockpit)	One of these roles: • Destination Viewer • Destination Configuration Viewer • Destination Viewer Instance • Destination Configuration Instance Viewer ***plus*** one of these roles: • ***Org Manager*** • ***Space Manager*** • ***Space Developer*** See User and Member Management.

Level	Operation (SAP BTP Cockpit or Cloud Connector)	Role
	Manage certificates (all CRUD operations) on service instance level (cockpit)	One of these roles: • Destination Administrator • Destination Certificate Administrator • Destination Administrator Instance • Destination Certificate Instance Administrator ***plus*** one of these roles: • ***Org Manager*** • ***Space Manager*** • ***Space Developer*** See User and Member Management.
	View certificates (read operations) on service instance level (cockpit)	One of these roles: • Destination Viewer • Destination Certificate Viewer • Destination Viewer Instance • Destination Certificate Instance Viewer ***plus*** one of these roles: • ***Org Manager*** • ***Space Manager*** • ***Space Developer*** See User and Member Management.

Back to Technical Roles [Feature Set B]

Default Role Collections [Feature Set B]

Default Role Collection	Connectivity Roles Included
Subaccount Administrator	• Cloud Connector Administrator • Destination Administrator
Subaccount Viewer	• Cloud Connector Auditor • Destination Viewer
Cloud Connector Administrator	Cloud Connector Administrator
Destination Administrator	Destination Administrator

Default Role Collection	Connectivity Roles Included
Connectivity and Destination Administrator	• Cloud Connector Administrator • Destination 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]

Back to Technical Roles

Back to User Roles

Back to Content

Related Information

What's New for Connectivity
Administration
Developing Applications
Security
Monitoring and Troubleshooting
Security Administration: Managing Authentication and Authorization

What's New for Connectivity

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

What's New for Connectivity

Related Information

2021 Connectivity (Archive)
2020 Connectivity (Archive)
2019 Connectivity (Archive)
2018 Connectivity (Archive)
2017 Connectivity (Archive)

2021 Connectivity (Archive)

2021

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - Cloud Connector - SAP Cloud PKI	SAP Cloud PKI (public key infrastructure) is enabled for technical communication between Cloud Connector and Connectivity service. • The change is transparent for the Cloud Connector - as soon as you renew your subaccount certificate in the Cloud Connector, the newly issued X.509 client certificate will be part of SAP Cloud PKI. • If you are using Connectivity proxy software components as part of your solution, make sure you use version 2.4.1 or higher. For scenarios with termination in the ingress, version 2.3.1 of the Connectivity proxy is sufficient.	Info only
Connectivity	Integration Suite	Cloud Foundry Neo	Destination Service - Mail Destinations	You can now configure destinations of type MAIL with any OAuth-based authentication type as available for HTTP destinations, including the option for mTLS via X.509 client certificate. **! Restriction** The Mail Java API (Neo environment) does not provide the `javax.mail.Session` object out of the box. It must be configured manually.	Info only

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry Neo	Cloud Connector 2.14.0.1 - Enhancements	Release of Cloud Connector version 2.14.0.1 introduces the following enhancements: • Cloud Connector can now use SAPMachine 11 as Java runtime. For more information, see Prerequisites. • Cloud Connector supports Windows Server 2022 as additional OS version. For more information, see Prerequisites. • Monitoring was extended to show usage information for service channels (on-premise to cloud scenarios). For more information, see Monitoring. • An administrator can now configure more connectivity-related parameters on the **Configuration** > **Advanced** screen instead of modifying the configuration files on OS level. For more information, see Configure Tunnel Connections. • You can define a different location for audit log and trace files. For more information, see Manage Audit Logs and Troubleshooting • Additional configuration REST APIs let you configure the Cloud Connector remotely. For more information, see Configuration REST APIs. • The additional role *Subaccount Administrator* lets you define authorizations limited to subaccount-related tasks. Cross-subaccount configuration can only be viewed by users having this role. For more information, see Use LDAP for Authentication. • Access control usage monitor data is now persisted and will survive a restart. • The connection check for HTTPS access control entries now reveals information about the causes for a failing check and potential configuration issues if principal propagation with x.509 certificates is used. Action: We recommend that you always use the latest Cloud Connector version. For more information, see Upgrade.	Recommended

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry Neo	Cloud Connector 2.14.0.1 - Fixes	Release of Cloud Connector version 2.14.0.1 provides the following bug fixes: • Audit log selection was not working correctly if the end of the time interval was before noon. This issue has been fixed. • If an RFC connection is broken while waiting for data from the ABAP system, the processing engine could get into an inconsistent state, causing wrong processing for succeeding requests sent from the cloud application, which eventually could make the cloud application hang. This issue has been fixed. • Fixed a race condition that could occur if many requests were sent from the cloud application over the same RFC connection, and if the network from the cloud application to the Cloud Connector was very fast. In such a situation, two threads were processing this single RFC connection, causing an inconsistency that could lead to a `NullPointerException` in `RfcBlock.populateRequestStatistics` when trying to access the field *<performanceStatistics>*. • When configuring an RFC SNC acccess control entry, but overall SNC configuration is incomplete, Cloud Connector now reports the configuration error at runtime instead of falling back to plain RFC, if offered by the backend.	Info only

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Features	• The **Destinations** UI in the cockpit lets you configure an X.509 client certificate for automatic token retrieval when using the relevant OAuth-based authentication types. The `AuthenticationHeaderProvider` Java client library, part of SAP Java Buildpack, was adapted as well. The REST API already supports it. • The `ProxyType` attribute now offers the new option `PrivateLink`, allowing you to configure a destination with URL, and optionally a token service URL, pointing to services consumed via the SAP Private Link service (beta). For more information, see also What Is SAP Private Link Service (Beta)?. **Note** The `CheckConnection` functionality as well as automatic token retrieval are not yet supported.	Info only
Connectivity	Integration Suite	Neo	Destination Service - Features	The **Destinations** UI in the cockpit lets you configure an X.509 client certificate for automatic token retrieval when using the relevant OAuth-based authentication types. The `AuthenticationHeaderProvider` Java client library, part of the Neo runtimes, was adapted as well.	Info only
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bug Fix	• A change has been applied that improves the stability and availability of the service during startup, for example, in case of a rolling update. • A request processing change has been applied, improving asyncronous handling of the processing load, ultimately improving overall service stability and availability.	Info only
Connectivity	Integration Suite	Cloud Foundry	Destination Service - OAuth	• Destinations with `ProxyType` set to `OnPremise` can now be configured with OAuth-based authentication types, both via the BTP cockpit UI and the Destination service REST API. • Automated token retrieval from OAuth servers residing on premise, exposed via Connectivity service and Cloud Connector, is now supported.	Info only

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry Kyma	Connectivity Proxy Version 2.4.1	Connectivity proxy version 2.4.1 is now available. • Connectivity CA is now automatically downloaded during help deployment. • Applies critical preparation for adoption of SAP Cloud PKI. • Improved server certificate validation towards remote targets. • Multiple open source software components reported as vulnerable have been replaced.	Info only
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - Bug Fix	An internal service exception could result in a *502 Bad Gateway* error on the client side, which was visible in the Cloud Connector logs during automatic reconnect. This issue has been fixed.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bug Fixes	• In some cases, a Destination service instance could not be deleted. The operation now works as expected. • A performance optimisation reduces the overall amount of remote calls to XSUAA when the Destination service is called with a user token. As a result, less load is put on XSUAA, and the Destination service responds faster. This fix contributes to improving overall stability.	
Connectivity	Integration Suite	Cloud Foundry	SAP Java Buildpack	SAP Java Buildpack has been updated from version 1.38.0 to 1.39.0. • The `com.sap.cloud.security.xsuaa` API has been updated to version 2.10.5. • The Connectivity API extension has been updated to version 3.12.0.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Authentication Types	When using authentication type `OAuth2ClientCredentials`, you can choose a tenant to perform automated token retrieval that is different from the tenant used to look up the destination configuration. This feature is especially useful for automated service scenarios, like running offline jobs, and so on. **Note** You cannot use this feature in combination with a passed user context. In this case, the tenant used to perform automated token retrieval is exclusively determined by the user context. For more information, see OAuth Client Credentials Authentication.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Neo	Destinations - Timeout Properties	You can configure timeout properties in the destination configuration, following a documented naming convention. This feature lets you manage timeouts externally, regardless of the cloud application's lifecycle. Using HttpDestination library version 2.15 , timeout properties are processed at runtime when pre-configuring the HTTP client instance for the cloud application. For more information, see HTTP Destinations.	
Connectivity	Integration Suite	Cloud Foundry Neo	Cloud Connector - Security Fixes	Multiple vulnerabilities in the Cloud Connector have been fixed. For more information, see SAP security note 3058553 .	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Client Certificates	When generating an X.509 client certificate (as announced on July 1), you can set a password to protect the private key. If you choose a PKCS12 file format, also the keystore is protected by the same password.	
Connectivity	Integration Suite	Neo	Destination Service - HTTP Headers	As of HttpDestination version 2.14.0 , any defined HTTP header in the destination configuration (see HTTP Destinations) is processed at runtime, that is, it is added to the request that is sent to the target server.	
Connectivity	Integration Suite	Neo	Connectivity Service - Bug Fix	A few stabilisation fixes have been applied in the Connectivity service to handle rare cases in which an abnormal amount of metering data was received by the Cloud Connector. This could cause a partial blockage of the service.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bug Fix	A few stabilisation fixes have been applied on the REST server-side logic, allowing more efficient parallel request handling under load.	
Connectivity	Integration Suite	Cloud Foundry Neo	Cloud Connector 2.13.2 - Enhancements	Release of Cloud Connector version 2.13.2 introduces the following improvement: • The HTML validation of login information has been improved.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry Neo	Cloud Connector 2.13.2 - Fixes	Release of Cloud Connector version 2.13.2 provides the following bug fixes: • A regression prevented that links provided in the login info widget (login screen) could be clicked. This issue has been fixed. • When rewriting a location header for redirect responses (status codes 30x), the lookup to determine the virtual host that replaces the internal one is now case insensitive. • When using custom attributes in JWTs (JSON web tokens) for principal propagation, the value can now be extracted even if represented as single element array. • A slow network could prevent a successful initial push, caused by an unintended timeout. As a consequence, the configuration on the shadow instance could be incomplete, even though the shadow showed a successful connection. This issue has been fixed. • CPIC traces can now be turned on and off multiple times without the need to restart. • Issues with restoring a 2.13.x backup into a fresh installation on Linux have been fixed.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service	• Using the Destination service REST API, you can configure a service-side issued X.509 client certificate as part of the SAP Cloud PKI (public key infrastructure) and formally choose *automatic renewal* of the certificate. • The same feature is available in the *Destinations* editor of the cloud cockpit (**Connectivity** > **Destinations**). • The *SAP Java Buildpack* now includes Java APIs as part of a client library for the Destination service. You can use the `ConnectivityConfiguration` Java API to retrieve destination and certificate configurations, and `AuthenticationHeaderProvider` Java API to provide prepared HTTP headers holding authentication tokens for various scenarios. For more information, see Destination Java APIs.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Destinations Editor	The *Destinations* UI (editor) in the cockpit lets you create a certificate configuration entry containing an X.509 client certificate part of SAP Cloud PKI (public key infrastructure). Optionally, you can specify values for the certificate common name (CN) as well as for the validity period (minimum value: one day, maximum value: one year). This feature has already been available via the Destination service REST API.	
Connectivity	Integration Suite	Cloud Foundry Neo	Destination Service - Destinations Editor	The *Destinations* UI (editor) in the cockpit has introduced a warning message as a reminder that the user's personal password should not be used when configuring the authentication type of a destination, for example, `BasicAuthentication` or `OAuth2Password`. The reason behind is that by design, destination configurations are meant to be used by one or more cloud applications which tipically are used by more than one person.	
Connectivity	Integration Suite	Cloud Foundry Neo	Connectivity Service - Bug Fix	In rare cases, the Cloud Connector version shown in the cockpit (**Connectivity → Cloud Connectors**) got lost. This issue has been fixed.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bug Fix	An issue has been resolved which prevented updating a service instance (previously created in the cockpit) via the Cloud Foundry CLI.	
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - Principal Propagation	You can configure a principal propagation scenario using a user access token issued by the Identity Authentication service (IAS), in addition to the scenarios based on XSUAA. For more information, see Principal Propagation via IAS Token.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - HTTP Destinations	A naming convention has been introduced, specifying how to properly configure HTTP headers and queries in a destination configuration. For more information, see HTTP Destinations.	
Connectivity	Integration Suite	Cloud Foundry Neo	Connectivity Service - Tunnel Connections	On the cloud side, the tunnel connection idle threshold has been increased to better match both older (yet supported) and latest Cloud Connector versions (versions lower or equal to 2.13). This ensures the internal heartbeat mechanism would work properly even in some special cases in which short interruptions have been observed.	
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bugfix	The service could have experienced delays in case of parallel load which caused requests to be executed unexpectedly slower on random basis. This issue has been fixed.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry Neo	Java Connector 3.1.4.0 - Enhancements	Release of Java Connector (JCo) version 3.1.4.0 introduces the following features and improvements: • JCo now offers the ABAP server processing time for JCo client scenarios via method `JCoThrougput.getServerTime()`. • `JCoRepository` methods were enhanced to ignore trailing blanks in passed structure, table, and function module names which are supposed to be looked up. • Performance was improved for setting `DATE` and `TIME` datatype fields when using *strings* as input values.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry Neo	Java Connector 3.1.4.0 - Bug Fixes	Release of Java Connector (JCo) version 3.1.4.0 provides the following bug fixes: • When invoking remote function modules (RFMs) via the t/q/bgRFC protocol to the same `JCoDestination` in multiple threads simultaneously, used TIDs, queue names and unit IDs could have been overwritten and used in the wrong thread context, which might have led to data loss in the target system. For example, IDocs that seemed to have been transferred correctly without an error, were not stored in the target system, because the TID contract was broken and several IDocs were erroneously sent at the same time with the same TID although different ones had been specified. This issue has been fixed. **Note** This regression bug was introduced with JCo 3.1.3. • When the new reentrance ticket technology, introduced as of S/4HANA 1909, was used to log on to the communication partner system, and `JCoCustomRepository` was configured to use query mode `DISABLE_REPOSITORY_POOL`, querying RFC metadata resulted in a logon failure (*invalid logon ticket*). This issue has been fixed. • If a JSON document contained numeric fields with negative values, for example, for a field of type `BCD` or `INT`, the JSON parser did not accept the sign character when analyzing the value for the given field. In this case, `JCoRecord.fromJSON()` threw an exception similar to *JCoSerializationException: (191) JCO_ERROR_SERIALIZATION: Digit(s) expected near position <###>*. This issue has been fixed.	
Connectivity	Integration Suite	Cloud Foundry	Cockpit: Cloud Connectors View	The **Cloud Connectors** view in the SAP BTP cockpit is now also available if your account is running on cloud management tools feature set B. For more information, see Monitoring (section *Monitoring from the Cockpit*).	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.13.1 - Enhancements	Release of Cloud Connector version 2.13.1 introduces the following features and improvements: • Additional audit log entries for changing the trace level are available. • You can open the support log assistant directly from the *Log And Trace Files* screen. For more information, see Troubleshooting, section *Log And Trace Files*. • The dependency on ping checks for connections to the LDAP system, which is used for UI authentication, has been minimized to avoid unnecessary role switches in high availability mode.	

Technical Component	Capability	Environment	Title	Description	Action

Connectivity	Integration Suite	Neo
		
		Cloud Foundry
			Cloud Connector 2.13.1 - Fixes	Release of Cloud Connector version 2.13.1 provides the following bug fixes:
				• Connecting a subaccount with several hundred access control entries is working again.
				• With release 2.13.1, restoring a backup created in version 2.13.0 works properly again for a Cloud Connector running on Linux.
				• Backups created in version 2.12.5 and older can be restored properly. Failures on restore led to a non-usable Cloud Connector setup.
				• The following high availability issues have been fixed:
				  ○ Improved implementation ensures that a high availability setup does not end up in a shadow/shadow situation. This issue could occur under rare circumstances.
				  ○ Errors could occur if subaccounts have a larger number of access control entries.
				  ○ Network issues could prevent the individual replication of configuration changes.
				  ○ After switching roles, connections can now always be reestablished correctly.
				• The connection test for LDAPS access control entries now works correctly.
				• A memory leak in the comprised netty library has been fixed by upgrading to a newer version.
				• A subaccount display issue has been fixed: In version 2.13.0, subaccounts on *eu2.hana.ondemand.com* were displayed as belonging to region Europe (Rot) instead of Europe (Frankfurt).	

Connectivity	Integration Suite	Cloud Foundry	Destination Service - Mobile Service Instances	You can create a destination configuration pointing to a mobile service instance, resulting in a fully functional destination configuration, including automatic token retrieval for the respective OAuth flows supported by the mobile service.	
Connectivity	Integration Suite	Kyma Other	Destination Service - Consumption from Kyma or Kubernetes Environments	Consumption of the Destination service from the Kyma or Kubernetes environments has been officially documented. For more information, see Create and Bind a Destination Service Instance.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry	Principal Propagation Authentication - OpenID Connect	When sending a user principal via the HTTP header `X-user-token`, you can use any *OpenID Connect*-compliant OAuth server and a related OpenID access token for passing the user identity. To enable this feature, you must specify either `x_user_token.jwks_uri` `x_user_token.jwks` as additional attribute, as described in the respective authentication type. or For more information, see HTTP Destinations.	
Connectivity	Integration Suite	Cloud Foundry	OAuth - X.509 Client Certificates	You can use X.509 client certificates for OAuth flows supported by the respective authentication types, see OAuth with X.509 Client Certificates.	
Connectivity	Integration Suite	Cloud Foundry	"Find Destination" REST API - Skip Credentials	The "Find Destination" REST API endpoint has been enhanced with a new feature enabling the client application to initiate a skip of credentials in the returned response. This parameter is useful especially for OAuth destinations (such as *OAuth2 User Token Exchange*, *OAuth2 JWT Bearer*, *OAuth2 SAML Bearer Assertion*). The client application may actually need only the auto-retrieved token by the service, which makes the credentials optional for the application, and in certain cases they are preferred not to be returned in the response. For more information, see SAP API Business Hub.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Cloud Foundry	Destinations - Authentication Types	• The property `SystemUser` is deprecated. The cockpit now shows an alert if this feature is still in use, suggesting what to do instead. Alternatives for technical user authentication are *Basic Authentication*, *OAuth2 Client Credentials*, or *Client Certificate Authentication*. See also OAuth SAML Bearer Assertion Authentication. **ⓘ Note** In general, we recommend that you work on behalf of specific (named) users rather than working with a technical user. To extend an OAuth access token's validity, consider using an OAuth refresh token. • Authentication type *SAP Assertion SSO* is deprecated. The cockpit now shows an alert if this feature is still in use, suggesting what to do instead. Authentication types *Principal Propagation* (for on-premise connections), *OAuth2 SAML Bearer Assertion* (Internet connections) or *SAML Assertion* (Internet connections) are the recommended mechanisms for establishing single sign-on (SSO). See SAP Assertion SSO Authentication.	
Connectivity	Integration Suite	Neo	Password Storage API	The Password Storage API documentation on SAP API Business Hub has been moved from the deprecated API package to SAP Cloud Platform Credential Store .	
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.13.0 - Enhancements	Release of Cloud Connector version 2.13.0 introduces the following features and improvements: • Cloud Connector 2.13 is based on a different runtime container. Up to version 2.12.x, the JavaWeb 1.x runtime based on Tomcat 7 was used. It now switches to JavaWeb 3.x based on Tomcat 8.5. As a consequence, the internal structure has changed and works differently. The upgrade will adjust these changes as much as possible for versions 2.9 and higher. • Linux on *ppc64 little endian* (ppc64le) is added as a supported platform for the Cloud Connector. For more information, see Prerequisites. • A set of new configuration REST APIs has been added.	

Technical Component	Capability	Environment	Title	Description	Action
				For more information, see Configuration REST APIs. • An additional screen in the subaccount-specific monitoring provides usage statistics of the various access control entries. Alternatively, you can access the same data using a new monitoring REST API. For more information, see Monitoring. • For access control entries of type TCP, you can configure a port range instead of a single port. For more information, see Configure Access Control (TCP). • You can configure a widget that shows information about the Cloud Connector on the login screen. For more information, see Configure Login Screen Information. • Improved high availability communication supersedes applying SAP note 2915578. • For new Cloud Connector versions, a notification and alert is shown to help you schedule the update. • The Cloud Connector supports JSON Web tokens (JWTs) based on OpenID-Connect (OIDC) for principal propgation authentication (Cloud Foundry environment). For more information, see Configure Principal Propagation via OIDC Token. • Client-side load balancing based on round-robin was introduced for Cloud Connector connections to SAP Cloud Platform to address its endpoints which are exposed on multiple IP addresses for high availability. • Scenarios based on the HTTP header *Expect: 100-continue* and response code *HTTP 100* are now supported.	

Technical Component	Capability	Environment	Title	Description	Action
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.13.0 - Fixes	Release of Cloud Connector version 2.13.0 provides the following bug fixes: • When doing a rollover at midnight, the initial audit log entry for a new file was not created correctly and the audit log checker wrongly assessed such files as corrupted. This issue has been fixed. • Incorrect host information could be used in audit logs related to access control audit entries. This issue has been fixed.	
Connectivity	Integration Suite	Cloud Foundry	Cloud Connector - Subaccount Configuration	For a subaccount that uses a custom identity provider (IDP), you can choose this IDP for authentication instead of the (default) SAP ID service when configuring the subaccount in the Cloud Connector. For more information, see Use a Custom IDP for Subaccount Configuration.	

2020 Connectivity (Archive)

2020

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Neo Cloud Foundry	Java Connector (JCo) - Client Certificates	JCo provides the new property `jco.client.tls_client_certificate_logon` 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
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
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

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Cloud Foundry	HTTP Destinations - Destination Properties	The destination property `SystemUser` for the authentication types: • 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	Announcement
Connectivity	Integration Suite	Neo Cloud Foundry	JCo Runtime - Enhancement	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 RFC_METADATA_GET, thus avoiding the need to provide additional authorizations for the repository user.	New
Connectivity	Integration Suite	Neo Cloud Foundry	JCo Runtime - 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 STRING and XSTRING was *null*. Since the initial value check in ABAP is different, JCo now behaves the same way and uses an emtpy string and an empty byte array, respectively.	Changed
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Automatic Token Retrieval	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
Connectivity	Integration Suite	Cloud Foundry	Documentation - Principal Propagation Scenarios	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.	Changed

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Cloud Foundry	Cloud Connector 2.12.5 - Enhancements	Release of Cloud Connector version 2.12.5 introduces the following improvements: • For principal propagation scenarios, custom attributes stored in `xs.user.attributes` 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
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.5 - Fixes	Release of Cloud Connector version 2.12.5 provides the following bug fixes: • 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.	Changed
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - Service Instances - Quota Management	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 Cloud Platform to simplify its usage. See also Create and Bind a Connectivity Service Instance.	Changed

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Service Instances - Quota Management	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 Cloud Platform to simplify its usage. See also Create and Bind a Destination Service Instance.	Changed
Connectivity	Integration Suite	Cloud Foundry	SAP Java Buildpack - Java Connector (JCo)	The SAP Java Buildpack has been updated from 1.27.3. to 1.28.0. • TomEE Tomcat has been updated from 7.0.104 to 7.0.105. • SAPJVM has been updated to 81.65.65. • The *com.sap.cloud.security.xsuaa* 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: *connectivity* to 3.3.3, *connectivity apiext* 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: ***<USE_JCO=true>***. **Note** The **previous activation process** for the JCo component is deprecated and **will expire after a transition period**.	Changed
Connectivity	Integration Suite	Cloud Foundry	Destination 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.	Changed
Connectivity	Integration Suite	Neo	Connectivity 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.	Changed
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bug Fix	During *Check Connection* 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.	Changed

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Bug Fix	Using authentication type `OAuth2SAMLBearerAssertion`, 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.	Changed
Connectivity	Integration Suite	Cloud Foundry	Destination Service REST API - Pagination Feature	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.	Changed
Connectivity	Integration Suite	Neo	HttpDestination Library - New Version	The `HttpDestination` v2 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
Connectivity	Integration Suite	Cloud Foundry	Destination 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.	Changed
Connectivity	Integration Suite	Cloud Foundry	Destinations - Authentication Types	You can use authentication type `OAuth2JWTBearer` when configuring a Destination. It is a simplified version of the authentication type `OAuth2UserTokenExchange` and represents the official OAuth grant type for exchanging OAuth tokens. See HTTP Destinations.	New
Connectivity	Integration Suite	Cloud Foundry	Destination Service - HTTP Header	The Destination service provides a prepared HTTP header that simplifies application and service development. See HTTP Destinations (code samples).	New
Connectivity	Integration Suite	Cloud Foundry	Destination 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.	Changed
Connectivity	Integration Suite	Cloud Foundry	HTTP Destinations - Authentication Types	The Cloud Foundry environment supports SAP `Assertion SSO` as authentication type for configuring destinations in the Destination service. See HTTP Destinations.	New
Connectivity	Integration Suite	Cloud Foundry	Destination 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
Connectivity	Integration Suite	Cloud Foundry	Destinations for Service Instances	For subscription-based scenarios, you can use an automated procedure to create a destination that points to your service instance. See Managing Destinations.	New

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Neo Cloud Foundry	Connectivity 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.	Changed
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.4 - Features	Release of Cloud Connector version 2.12.4 introduces the following features and enhancements: • You can activate the SSL trace in the Cloud Connector administration UI also for the shadow instance.	New
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.4 - Fixes	Release of Cloud Connector version 2.12.4 provides the following bug fixes: • 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
Connectivity	Integration Suite	Cloud Foundry	Connectivity 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
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Service Instances	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
Connectivity	Integration Suite	Cloud Foundry	Cockpit - Destination Management	The **Destinations** editor in the cockpit is available for accounts running on the cloud management tools feature set B. See Managing Destinations.	New
Connectivity	Integration Suite	Neo	Connectivity 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

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.3 - Features	Release of Cloud Connector version 2.12.3 introduces the following features and enhancements: • 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 `scc_daemon_extension.sh` 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 *Installer Scenario*.	New

Connectivity	Integration Suite	Neo
		
		Cloud Foundry
			Cloud Connector 2.12.3 - Fixes	Release of Cloud Connector version 2.12.3 provides the following bug fixes:
				• 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 *Received invalid block with negative size*. This issue has been fixed.
				• The Cloud Connector admin UI now shows the correct user information for installed Cloud Connector instances in the *About* window.
				• Fixes in the context of disaster recovery:
				  ◦ The location ID is now handled properly when setting it *after* 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
					Changed

Technical Component	Capability	Environment	Title	Description	Type
Connectivity	Integration Suite	Neo Cloud Foundry	JCo Runtime - WebSocket 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
Connectivity	Integration Suite	Cloud Foundry	Connectivity 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

2019 Connectivity (Archive)

2019

Technical Component	Capability	Environment	Title	Description
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.2 - Features	Release of Cloud Connector version 2.12.2 introduces the following features and enhancements: • You can turn on the TLS trace from the Cloud Connector administration UI instead of modifying the `props.ini` 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.
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.2 - Fixes	Release of Cloud Connector version 2.12.2 provides the following bug fixes: • 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.

Technical Component	Capability	Environment	Title	Description
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - Bug Fix	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.
Connectivity	Integration Suite	Neo	Region Europe (Frankfurt) - Change of Connectivity Service Hosts	The following IP addresses of the Connectivity service hosts for region Europe/Frankfurt (`eu2.hana.ondemand.com`) will change on **26 October 2019**: • `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) 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**. See also: Prerequisites: Network.
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Connection Check	Using the **Destinations** editor in the cockpit, you can check connections also for on-premise destinations. See Check the Availability of a Destination.
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector - Java Runtime	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. 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. See SAP Cloud Connector – Java 7 support will phase out and Update the Java VM.
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.1 - Features	Release of Cloud Connector version 2.12.1 introduces the following features and enhancements: • 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).

Technical Component	Capability	Environment	Title	Description
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12.1 - Fixes	Release of Cloud Connector version 2.12.1 provides the following bug fixes: • 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.
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Scope Attribute for OAuth-based Authentication Types	You can use the `scope` destination attribute for the OAuth-based authentication types `OAuth2ClientCredentials`, `OAuth2UserTokenExchange` and `OAuth2SAMLBearerAssertion`. 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. See HTTP Destinations.
Connectivity	Integration Suite	Neo	JCo Runtime for SAP Cloud Platform - Features	• Additional APIs have been added to `JCoBackgroundUnitAttributes`. 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. See API documentation of `JCoTable` and `JCoStructure`.
Connectivity	Integration Suite	Neo	JCo Runtime for SAP Cloud Platform - Fixes	• 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.
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - JCo Multitenancy	The Connectivity service supports multitenancy for JCo applications. This feature requires a runtime environment with SAP Java Buildpack version 1.9.0 or higher. See Scenario: Multitenancy for JCo Applications (Advanced).
Connectivity	Integration Suite	Cloud Foundry	Cloud Cockpit - Cloud Connector View	The Cloud Connector view is available also for Cloud Foundry regions. It lets you see which Cloud Connectors are connected to a subaccount.

Technical Component	Capability	Environment	Title	Description
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12 - Features	Release of Cloud Connector version 2.12 introduces the following features and enhancements: • 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 ◦ 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.

Technical Component	Capability	Environment	Title	Description
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.12 - Fixes	Release of Cloud Connector version 2.12 provides the following bug fixes: • 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. • `IllegalArgumentException` does not occur any more 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.
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Authentication Types	Authentication type `OAuth2SAMLBearerAssertion` provides two different types of `Token Service URL`: • `Dedicated`: used in the context of a single tenant, or • `Common`: used in the context of multiple tenants. For type `Common`, the tenant subdomain is automatically set to the target `Token Service URL`. In addition, cloud applications can use the `x-user-token` 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. See SAML Bearer Assertion Authentication.
Connectivity	Integration Suite	Neo Cloud Foundry	Connectivity Service - Fix	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. This issue has been fixed. The connection is now always closed right after sending the response.
Connectivity	Integration Suite	Cloud Foundry	Connectivity Service - Protocols	The Connectivity service supports TCP connections to on-premise 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.

Technical Component	Capability	Environment	Title	Description
Connectivity	Integration Suite	Neo	Connectivity Service - Fix	After receiving an on-premise system response with HTTP header *Connection: close*, 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.
Connectivity	Integration Suite	Neo	Cloud Connector - Certificate Update	For the Connectivity service (Neo environment), **a new, region-specific certificate authority** (X.509 certificate) is being introduced. If you use the Cloud Connector for on-premise connections to the Neo environment, you must import the new certificate authority into your trust configuration. • 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 deployments, 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 Connector deployment must be configured to use the new client certificates. See Update the Certificate for a Subaccount.
Connectivity	Integration Suite	Cloud Foundry	Destination Service - Authentication Types	The new authentication type `OAuth2UserTokenExchange` lets your applications use an automated exchange of user access tokens when accessing other applications or services. The feature supports single-tenant and multi-tenant scenarios. See OAuth User Token Exchange Authentication.
Connectivity	Integration Suite	Neo	RFC - Stateful Sequences	You can make a stateful sequence of function module invocations work across several request/response cycles. See Invoking ABAP Function Modules via RFC.
Connectivity	Integration Suite	Neo Cloud Foundry	Cloud Connector 2.11.3	A security note for Cloud Connector version 2.11.3 has been issued. See SAP note 2696233.
Connectivity	Integration Suite	Cloud Foundry	Protocols - RFC Communication	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.
Connectivity	Integration Suite	Cloud Foundry	Destinations - Renew Certificates	A button in the **Destinations** editor lets you update the validity period of an X.509 certificate. See Set up Trust Between Systems.

2018 Connectivity (Archive)

2018

Technical Component	Capability	Environment	Title	Description	Type	Ava as c

Technical Component	Capability	Environment	Title	Description	Type	Ava as c
Connectivity	Integration	Neo	Connectivity Service - Performance	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	201 20
Connectivity	Integration	Neo	Connectivity Service - Resilience	The Connectivity service has a better protection against zombie connections, which improves resilience and overall availability for the cloud applications consuming it.	Changed	201 20
Connectivity	Integration	Neo	Password Storage Service	A Password Storage REST API is available in the SAP API Business Hub, see Password Storage (Neo Environment).	New	201 06
Connectivity	Integration	Neo	Destination Configuration Service	A Destination Configuration service REST API is available in the SAP API Business Hub.	New	201 06
Connectivity	Integration	Cloud Foundry	Destination Service	A Destination service REST API is available in the SAP API Business Hub.	New	201 06
Connectivity	Integration	Neo	JCo Runtime for SAP Cloud Platform - Fixes	• When using `JCoRecord.fromJSON()` 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 any more. • Depending on the configuration of certain JCo properties, an internally managed connection pool could throw a `JCoException` (error group `JCO_ERROR_RESOURCE`). In a thread waiting for a free connection from this pool, an error message then erroneously reported that the pool was exhausted . This error situation could occur if the used destination was not configured with the property `jco.destination.max_get_client_time` set to 0 and the destination's `jco.destination.peak_limit` value was set higher than the `jco.destination.pool_capacity`. This issue has been fixed.	Changed	201 06
Connectivity	Integration	Neo	JCo Runtime for SAP Cloud Platform - Features	Support of the RFC fast serialization. Depending on the exchanged parameter and data types, the performance improvements for RFC communication can reach multiple factors. See SAP note 2372888 (prerequisites) and Parameters Influencing Communication Behavior (JCo configuration in SAP Cloud Platform).	Changed	201 06
Connectivity	Integration	Neo	JCo Runtime for SAP Cloud Platform - Information	Local runtimes on Windows must install the VS 2013 redistributables for x64, instead of VS 2010.	Changed	201 06

Technical Component	Capability	Environment	Title	Description	Type	Ava as c
Connectivity	Integration	Neo Cloud Foundry	Cloud Connector Fixes	Release of Cloud Connector 2.11.3: • An issue in RFC communication could cause the trace entry ***com.sap.scc.jni.CpicCommunicationException: no SAP ErrInfo available*** 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	201 06
Connectivity	Integration	Neo Cloud Foundry	Cloud Connector Enhancements	Release of Cloud Connector 2.11.3: • If the user ***sapadm*** exists on a system, the installation on Linux assigns it to the ***sccgroup***, which is a prerequisite for solution management integration to work properly, see Configure Solution Management Integration. • Restoring a backup has been improved. See Configuration Backup. • 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.	Changed	201 06
Connectivity	Integration	Neo	HTTP Destinations	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 runtimes. See OAuth Client Credentials Authentication.	Changed	201 11
Connectivity	Integration	Cloud Foundry	User Propagation	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. See Consuming the Connectivity Service (Cloud Foundry Environment) and Configure Principal Propagation via User Exchange Token (Cloud Foundry Environment).	Changed	201 27

Technical Component	Capability	Environment	Title	Description	Type	Ava as c
Connectivity	Integration	Neo Cloud Foundry	Cloud Connector 2.11.2	Release of Cloud Connector 2.11.2 • 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.	Changed	201 16
Connectivity	Integration	Neo Cloud Foundry	Cloud Connector 2.11.2	Fixes of Cloud Connector 2.11.2 • 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 ***Trust Store***. • You can now configure also the MOC (Multiple Origin Composition) OData service paths as resources. • The `Location` 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.	Changed	201 16
Connectivity	Integration	Neo	HttpDestination Library	The SAP `HttpDestination` library (available in the SDK and cloud runtime "Java EE 6 Web Profile") now creates Apache `HttpClient` instances which work with strict SNI (Server Name Indication) servers. Use cases with strict SNI configuration on the server side will no longer get the error message ***Failure reason: "peer not authenticated"***, that was raised either at runtime or while performing a connection test via the SAP Cloud Platform cockpit **Destinations** editor (**Check Connection** function).	Changed	201 16

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.

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.
- The limitation that only 100 subaccounts could be managed with the administration UI has been removed. See Managing Subaccounts.

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.
- 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.

- 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).
- 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 and Initial Configuration (HTTP).
- 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.
- On Windows platforms, Cloud Connector 2.10 now requires Visual Studio 2013 runtime libraries.

Fix

Cloud Connector

- The 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.

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

Administration

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

Task	Description
Managing Destinations	Manage HTTP destinations for Cloud Foundry applications in the SAP BTP cockpit.
HTTP Destinations	You can choose from a broad range of authentication types for HTTP destinations, to meet the requirements of your specific communication scenario.
RFC Destinations	Use RFC destinations to communicate with an on-premise ABAP system via the RFC protocol.
Principal Propagation	Use principal propagation (user propagation) to securely forward cloud users to a back-end system (single sign-on).
Set up Trust Between Systems	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	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	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	To use the Destination service in your application, you must first create and bind an instance of the service.
Configuring Backup	Create a backup of your destination configurations.

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	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Destination Service REST API	Yes	No	Yes	Yes	Yes	Yes	No	No
Create Destinations Using the MTA Descriptor	Yes	Limited	No	No	Yes	No	No	No
Create Destinations on Service Instance Creation	Yes	No	No	No	Yes	No	No	No

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.

Prerequisites

1. You are logged into the SAP BTP cockpit.
2. You have the required authorizations. See User Roles.
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.

- Subaccount level – no specific prerequisites.

For more information, see Access the Destinations Editor.

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 - provide data communication via the HTTP protocol and are used for both Internet and on-premise connections.
  - RFC Destinations - 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
- Create Destinations from a Template
- Check the Availability of a Destination
- Clone Destinations
- Edit and Delete Destinations
- Use Destination Certificates
- Import Destinations
- Export Destinations

Related Information

Destination Examples

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 specifiy 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.

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**.

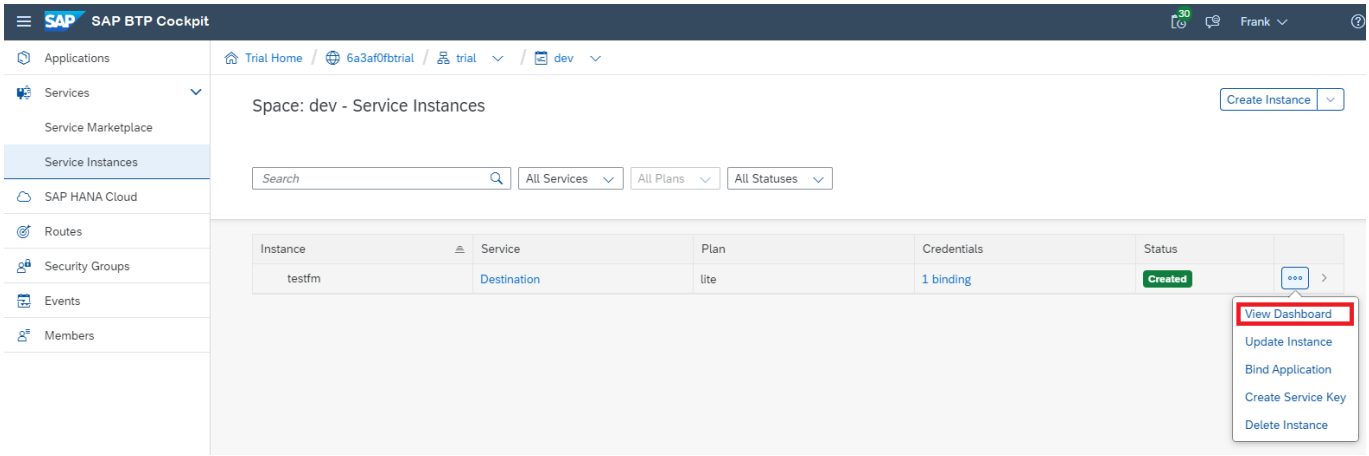
Access on Service Instance Level

> **Note**
>
> To perform these steps, you must have created a Destination service instance in your space, see Create and Bind a Destination Service Instance. On service instance level, you can set destinations only for Destination service instances.

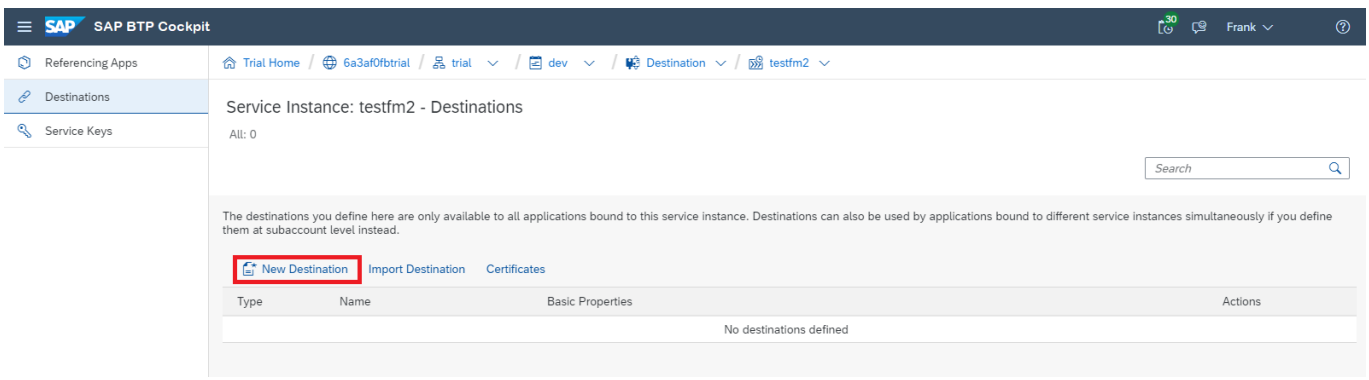
1. In the cockpit, choose your **Global Account** from the **Region Overview** and select a **Subaccount**.

   > **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**.



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



See also section ***Create and Bind a Service Instance from the Cockpit*** in Create and Bind a Destination Service Instance.

Related Information

Create Destinations from Scratch
Create Destinations from a Template
Check the Availability of a Destination
Clone Destinations
Edit and Delete Destinations
Use Destination Certificates
Import Destinations
Export Destinations

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, in contrast, you do not need this deeper knowledge for configuration, but edting options are limited by the available templates.

Related Information

Create HTTP Destinations
Create RFC Destinations
Create Mail Destinations
Destination Examples

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**.

   > **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.

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.

   > **Note**
   >
   > For more information, see also HTTP Destinations.

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

   For details, see HTTP Destinations.

   > **Note**
   >
   > If you set an HTTPS destination, you need to also add a Trust Store. For more information, see Use Destination Certificates.

8. (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 (section **Procedure**, step 4).

9. (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.

   c. To delete a property, choose the button next to it.

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

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

Related Information

Edit and Delete Destinations
Destination Examples

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**.

   > **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.

> **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.
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.
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 (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`	Three-letter system ID of your backend ABAP system (as configured in the Cloud Connector).
	`jco.client.mshost`	Message server host (as configured in the Cloud Connector).
	`jco.client.group`	(Optional) The group of application servers that is used (logon group). If not specified, the group `PUBLIC` is used.
	`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.

Proxy Type	Property	Description
	`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.

11. Press **Save**.

Related Information

Edit and Delete Destinations
Destination Examples
Cloud Connector

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**.

   > **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.

   > **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

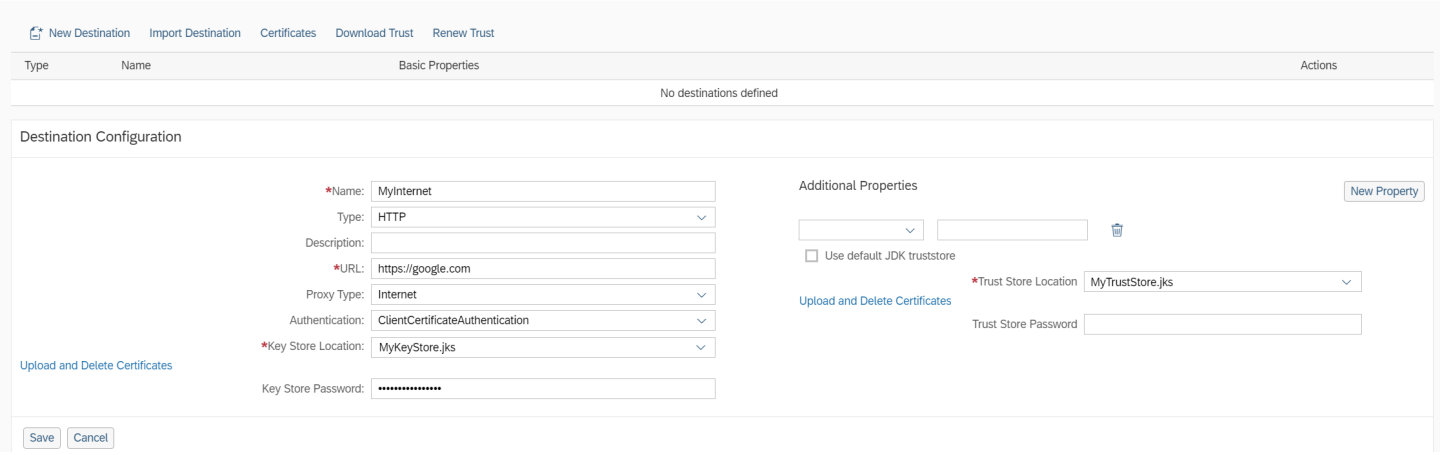
Edit and Delete Destinations
Destination Examples
Cloud Connector

Destination Examples

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

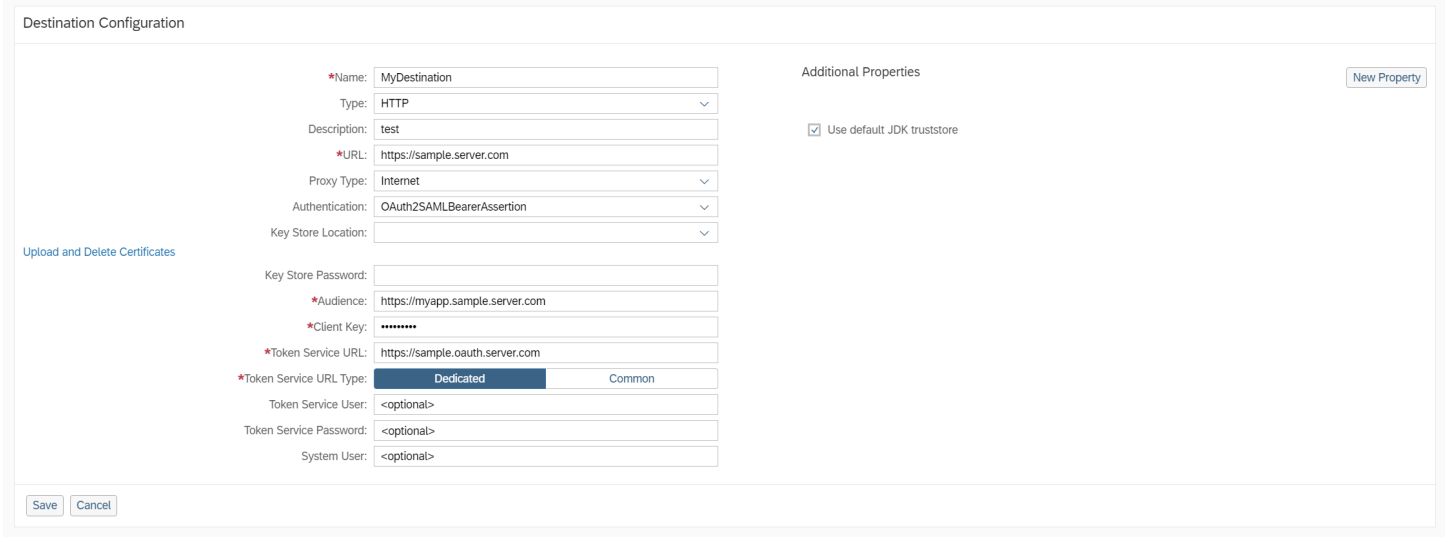
Content

Example: HTTP Destination (Internet, Client Certificate Authentication)



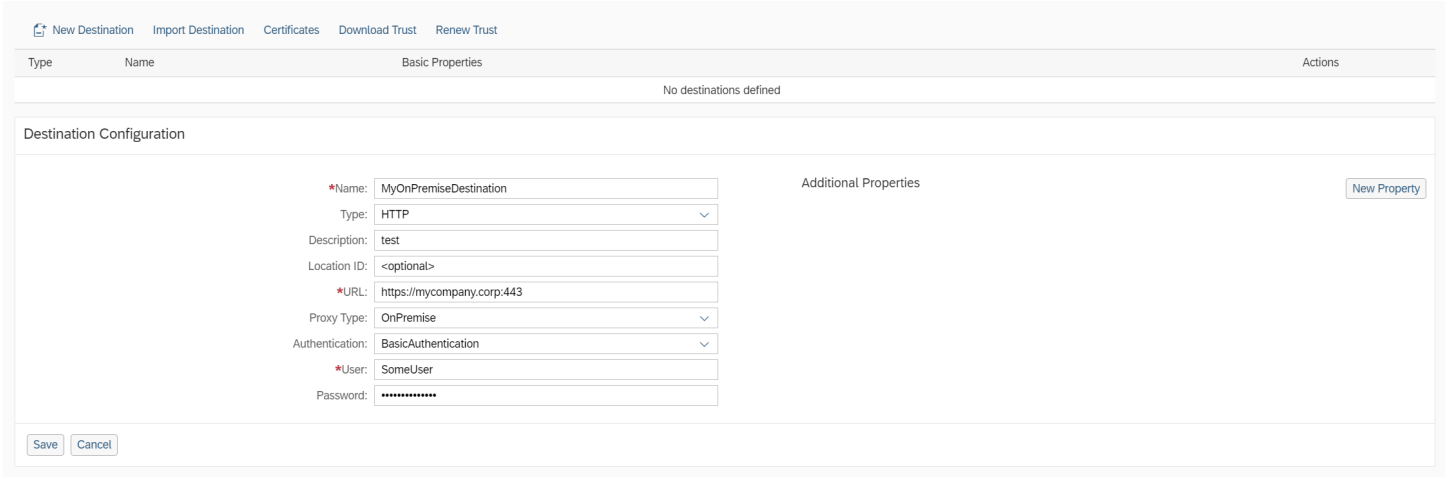
Back to Content

Example: HTTP Destination (Internet, OAuth2SAMLBearerAssertion)



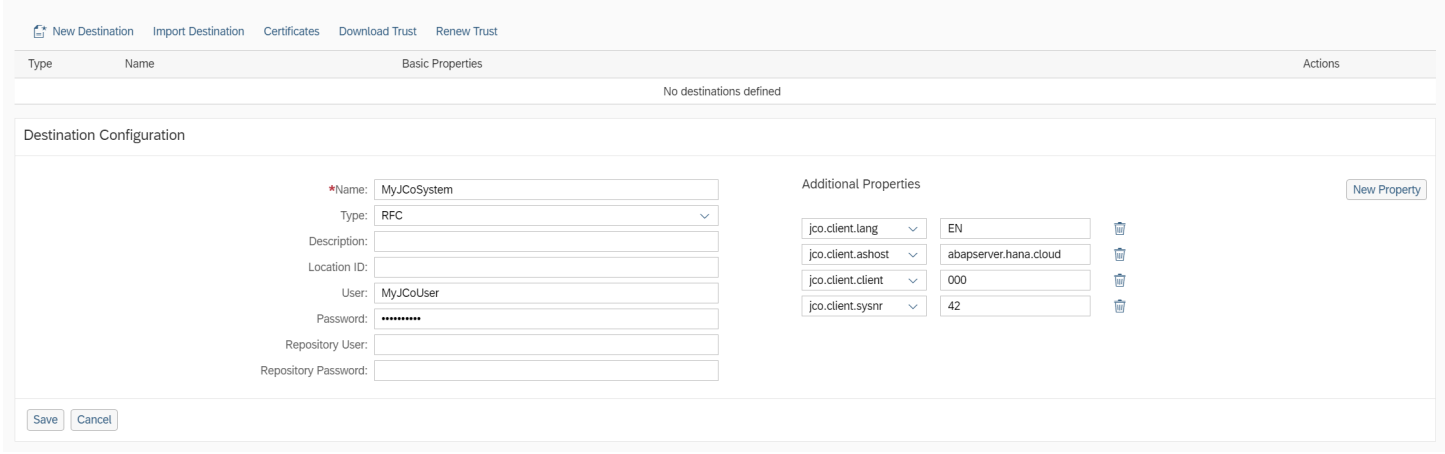
Back to Content

Example: HTTP Destination (On-Premise)



Back to Content

Example: RFC Destination



The following main properties correspond to the relevant additional properties:

User → `jco.client.user`

Password → `jco.client.passwd`

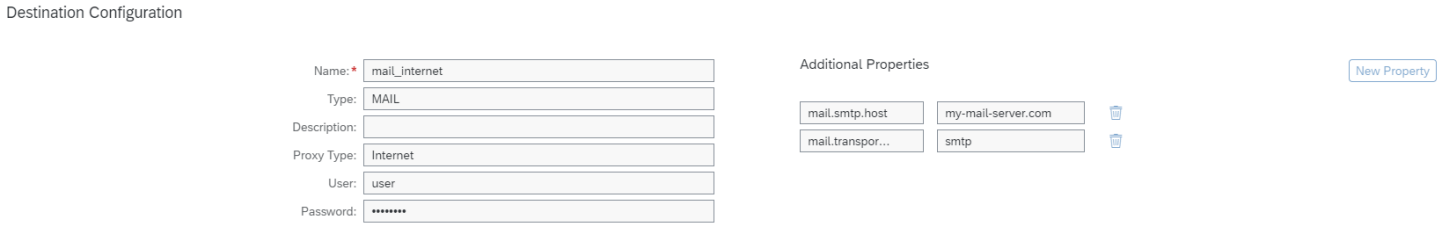
Repository password → `jco.destination.repository.passwd`

ℹ Note

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

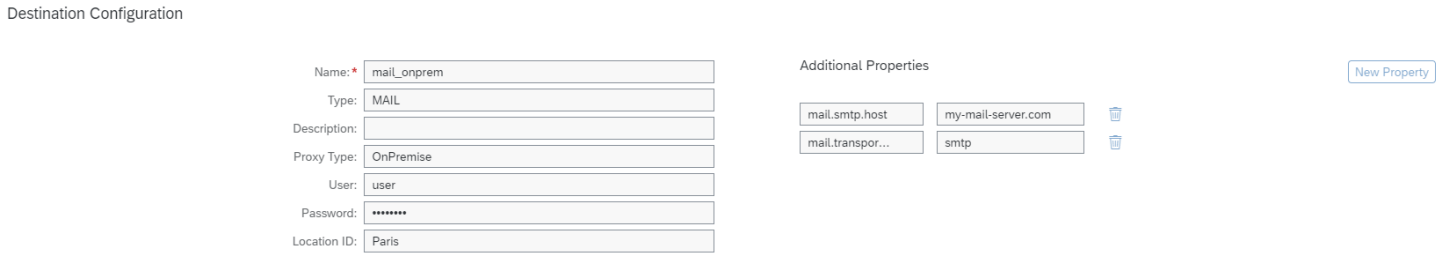
Back to Content

Example: Mail Destination (Internet)



Back to Content

Example: Mail Destination (On-Premise)



Back to Content

Related Information

HTTP Destinations
RFC Destinations

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

Destinations Pointing to Service Instances

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

> **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.
>
> In the meantime, you can follow the steps described in Create Destinations from Scratch.

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.

> **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.

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.

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**.

  > ❗ **Restriction**
  > The targeted URL cannot be a private endpoint (for example, *localhost*).

- 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
Backend status could not be determined.	• 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.	• 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
Backend is not available in the list of defined system mappings in Cloud Connector.	The Cloud Connector is not configured properly.	Check the basic Cloud Connector configuration steps: Initial Configuration
Resource is not accessible in Cloud Connector or backend is not reachable.	The Cloud Connector is not configured properly.	Check the basic Cloud Connector configuration steps: Initial Configuration
Backend is not reachable from Cloud Connector.	Cloud connector configuration is ok but the backend is not reachable.	Check the backend (server) availability.

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
Destination Examples

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
Destination Examples

Use Destination Certificates

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

Prerequisites

You have logged on to the cockpit and opened the **Destinations** editor. For more information, see Access the Destinations Editor.

Context

⚠ Caution

Uploaded certificates are accessible via the REST APIs, including any private data they may contain.

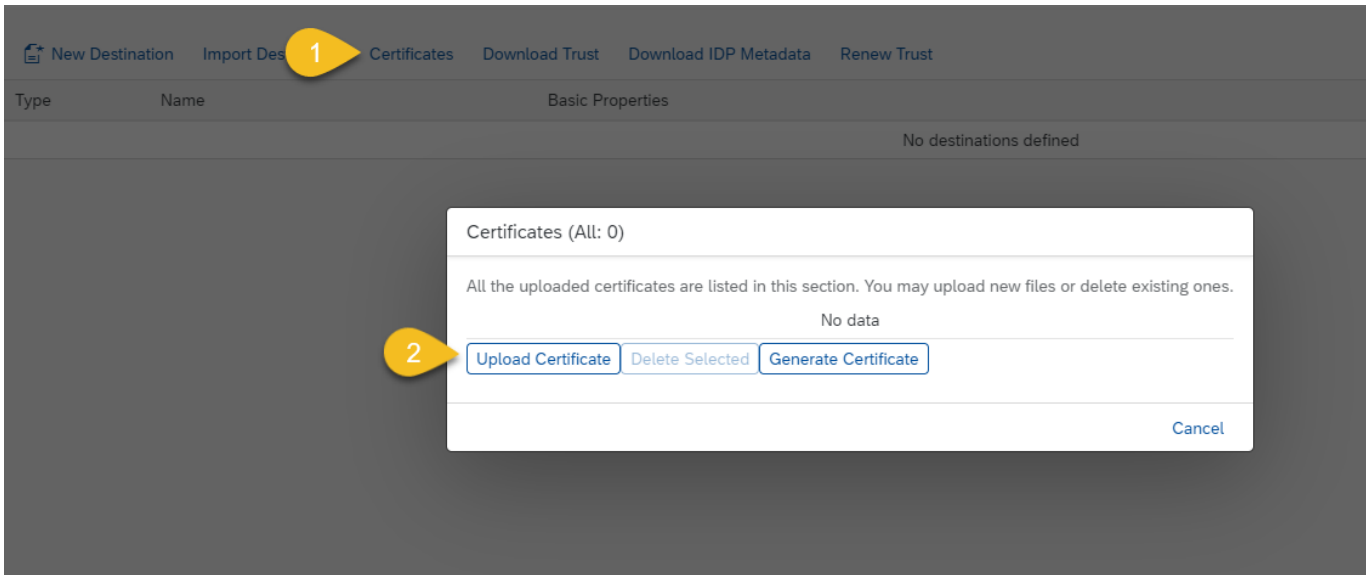
You can upload, add and delete certificates for your connectivity destinations.

- 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. A key store is available only for `ClientCertificateAuthentication` and OAuth authentication types (as a token service keystore, see OAuth with X.509 Client Certificates).
- An uploaded certificate file should contain the entire certificate chain.

Procedure

Upload Certificates

1. Choose **Certificates**.
2. Choose **Upload Certificate**.



3. Browse to the certificate file you need to upload.

   The certificate file is added.

ℹ Note

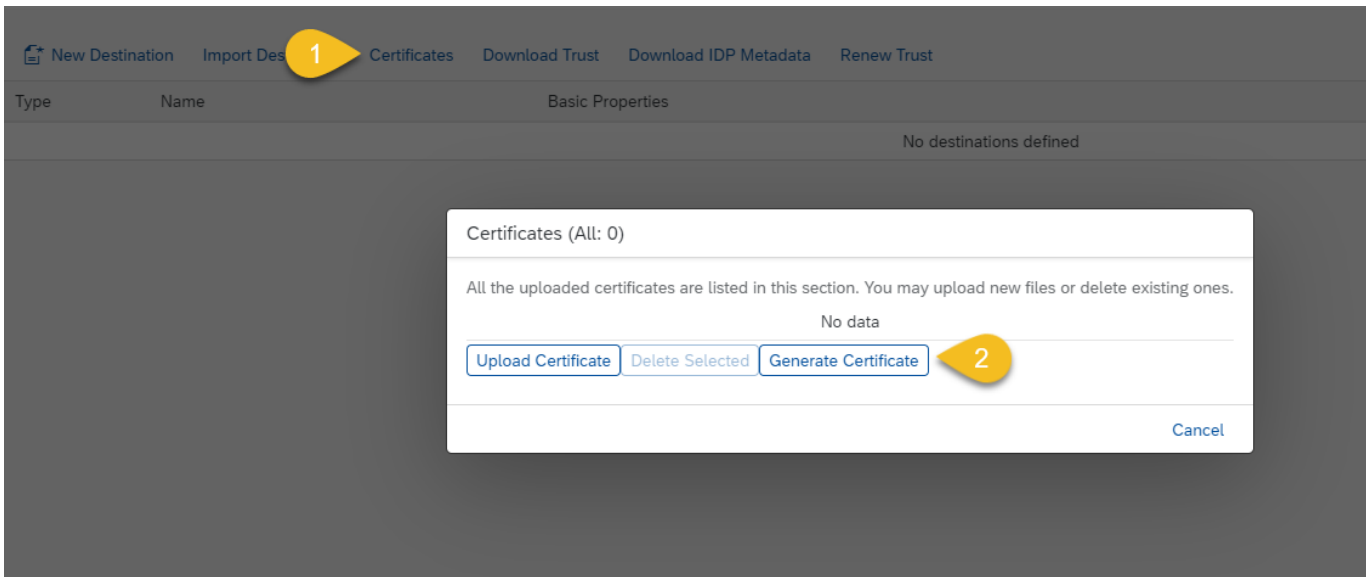
You can upload a certificate during creation or editing of a destination, by clicking the **Upload and Delete Certificates** link.

⚠ Caution

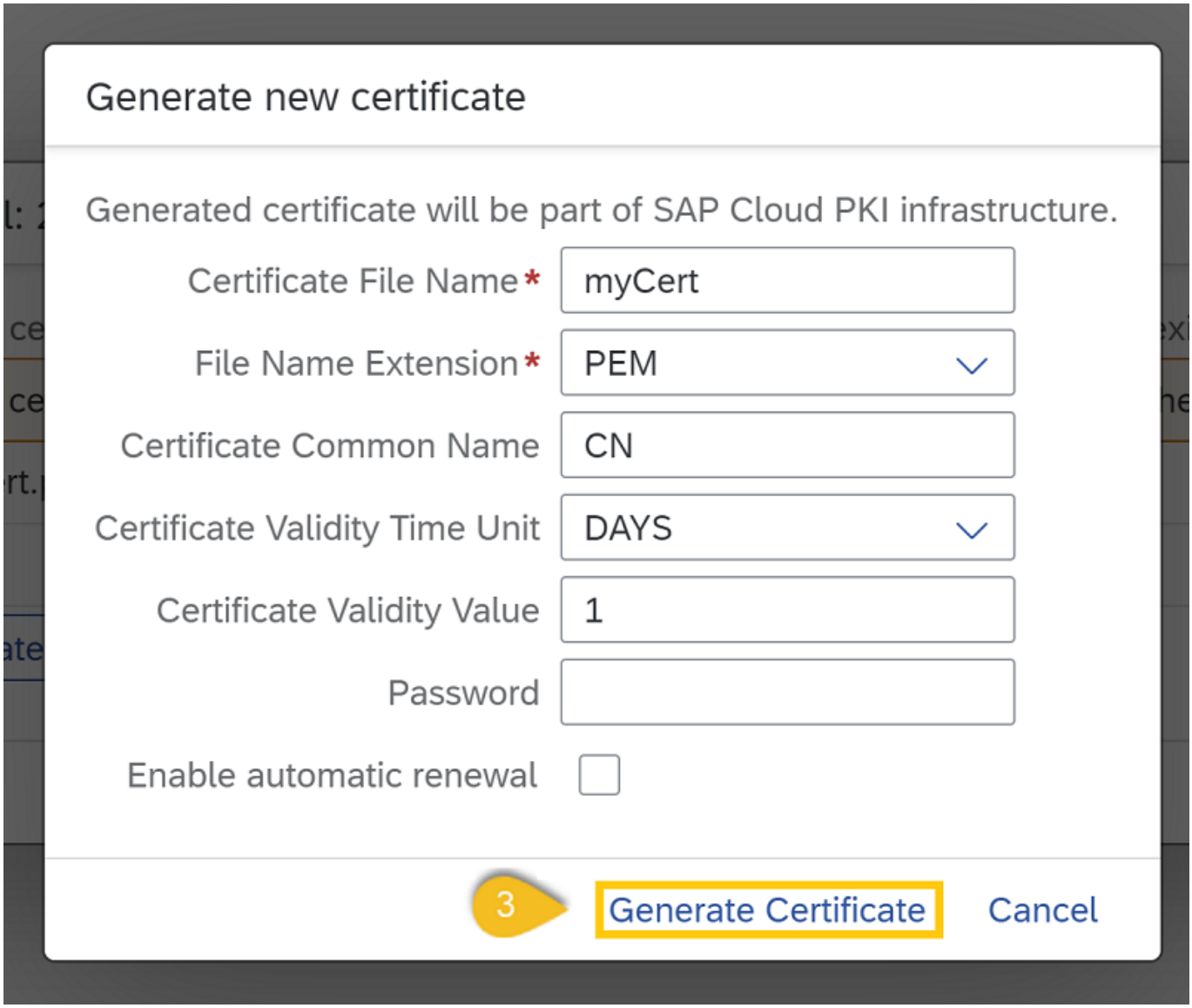
Certificates added through the **Upload Certificate** option cannot be renewed automatically.

Generate a SAP Cloud PKI Infrastructure Certificate

1. Choose **Certificates**.
2. Choose **Generate Certificate**.



3. In the pop-up window, enter certificate name and type. Optionally, you can enter certificate CN and certificate validity. (Optional) Additionally, you can select the **Enable automatic renewal** checkbox to renew the certificate automatically when it nears its expiration date. Choose **Generate Certificate** again. The certificate is generated, and you can use it in your destinations.



More Information

[Create HTTP Destinations]

[Edit and Delete Destinations]

[Import Destinations]

[Set up Trust Between Systems]

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

Destination Examples

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

Destination Examples

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.

Create Destinations Using the MTA Descriptor

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

Content

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

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

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.

> **Note**
> This function is equivalent to the Destinations Pointing to Service Instances 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.
>
> In the meantime, you can follow the steps described in Create Destinations from Scratch.

The descriptor may have the following structures:

Without client-provided service credentials:

```
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 service
ServiceInstanceName: <instance name> # the name of the service instance to which the destination wi
ServiceKeyName: <service key name> # the service key of the instance targeted by ServiceInstanceNam
AdditionalProp1: <value1> # optional
...
AdditionalPropN: <valueN> # optional
```

Applicable for:

- Service bindings of type "x509" which contain both a public and private key in the service key credentials
- Service bindings of type "clientsecret"

With client-provided service credentials:

```
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 service
ServiceInstanceName: <instance name> # the name of the service instance to which the destination wi
ServiceKeyName: <service key name> # the service key of the instance targeted by ServiceInstanceNam
ServiceCredentials: # the credentials of the service key which will be provided by the client
  PrivateKey: <value>
  ...
AdditionalProp1: <value1> # optional
...
AdditionalPropN: <valueN> # optional
```

Applicable for:

- Service bindings of type "x509" which contain only the public key in the service key credentials

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 may have the following structures:

Without client-provided service credentials:

```
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 service
URL: <url> # the URL of the target resource
TokenServiceInstanceName: <instance name> # the name of the service instance used for protecting th
TokenServiceKeyName: <service key name> # the service key of the instance targeted by ServiceInstan
AdditionalProp1: <value1> # optional
...
AdditionalPropN: <valueN> # optional
```

Applicable for:

- Service bindings of type "x509" which contain both a public and private key in the service key credentials
- Service bindings of type "clientsecret"

With client-provided service credentials:

```
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 service
URL: <url> # the URL of the target resource
TokenServiceInstanceName: <instance name> # the name of the service instance used for protecting th
TokenServiceKeyName: <service key name> # the service key of the instance targeted by ServiceInstan
TokenServiceCredentials: # the credentials of the service key which will be provided by the client
  PrivateKey: <value>
AdditionalProp1: value1 # optional
...
AdditionalPropN: valueN # optional
```

Applicable for:

- Service bindings of type "x509" which contain only the public key in the service key credentials

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

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.

Back to Content

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.

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.

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
```

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
- Principal Propagation SSO Authentication for HTTP
- OAuth SAML Bearer Assertion Authentication
- Client Authentication Types for HTTP Destinations
- OAuth Client Credentials Authentication
- OAuth User Token Exchange Authentication
- SAP Assertion SSO Authentication
- OAuth Password Authentication
- OAuth JWT Bearer Authentication
- SAML Assertion Authentication
- OAuth with X.509 Client Certificates
- OAuth Refresh Token Authentication
- OAuth Authorization Code Authentication
- OAuth Technical User Propagation Authentication
- Using Client Assertion with OAuth Flows

Custom Query Parameters and Headers

For most of these authentication types, you can add custom query parameters and headers to the URL of a destination. For more information, see the details of the respective authentication type.

Related Information

OAuth with X.509 Client Certificates
Create HTTP Destinations

Server Certificate Authentication

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

Context

The server certificate validation is applicable to all destinations with proxy type `Internet` and `PrivateLink` that use the HTTPS protocol.

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 versions 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: `TLSVersion=TLSv1.2` .
`TrustStoreLocation` 1. When used in local environment 2. When used in cloud environment	Path to the keystore file which contains trusted certificates (Certificate Authorities) for authentication against a remote client. To find the allowed keystore file formats, see Use Destination Certificates. 1. The relative path to the keystore file. The root path is the server's location on the file system. 2. The name of the keystore file. **Note** If the `TrustStoreLocation` property is not specified, the JDK trust store is used as a default trust store for the destination.
`TrustStorePassword`	Password for the JKS trust store file. This property is mandatory in case `TrustStoreLocation` is used.
`TrustAll`	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). In case `TrustAll` = TRUE, the `TrustStoreLocation` property is ignored so you can omit it. In case *<TrustAll>* = FALSE, the *<TrustStoreLocation>* property is mandatory to be used.

Property	Description
`HostnameVerifier`	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 `TrustAll` property is specified. The default value (used if no value is explicitly specified) is `Strict`. • `Strict HostnameVerifier` 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 "*.foo.com" matches only subdomains at the same level, for example "a.foo.com". It does not match deeper subdomains such as "a.b.foo.com". • `BrowserCompatible HostnameVerifier` 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. The only difference between `BrowserCompatible` and `Strict` is that a wildcard (such as ".foo.com") with `BrowserCompatible` matches all subdomains, including "a.b.foo.com". For more information about these Java classes, see Package org.apache.http.conn.ssl . In case *<TrustAll>* = TRUE, the *<HostnameVerifier>* property is ignored so you can omit it.

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

Related Information

Client Authentication Types for HTTP Destinations

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.

Properties

The following credentials need to be specified:

Property	Description
`Name`	Destination name. Must be unique for the destination level.
`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:
	
	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>",`
	`}`
	
	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=OnPremiseDestination
Type=HTTP
URL= http://virtualhost:80
Authentication=PrincipalPropagation
ProxyType=OnPremise
```

Related Information

Principal Propagation

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.

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 `HTTP` for all HTTP(S) destinations.
`URL`	URL of the target endpoint.
`ProxyType`	You can only use proxy type `Internet` or `OnPremise`. If `OnPremise` is used must be accessed through the Cloud Connector.
`Authentication`	Authentication type. Use `OAuth2SAMLBearerAssertion` as value.
`KeyStoreLocation`	Contains the name of the certificate configuration to be used for *per-destinati* signing. This certificate will be used instead of the standard subaccount-wide For more information, see Set up Trust Between Systems.
`KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when *destination* SAML assertion signing certificate.
`audience`	Intended audience for the assertion, which is verified by the OAuth authorizati information, see SAML 2.0 Bearer Assertion Profiles for OAuth 2.0 .
`clientKey`	Key that identifies the consumer to the authorization server.
`tokenServiceURL`	The URL of the token service, against which the token exchange is performed. `Token Service URL` type, this property is interpreted in different ways dur retrieval: • For `Dedicated`, the token service URL is taken as is. • For `Common`, the token service URL is searched for the tenant placehol `{tenant}` is resolved as the subdomain of the subaccount on behalf performing the call. If the placeholder is not found, `{tenant}` is inser the token service URL. The subaccount subdomain is mandated during creation of the subacc Subaccount. **Examples** of interpreting the token service URL for the token service URL type the Destination service is on behalf of a subaccount subdomain with value `myt` • `https://authentication.us10.hana.ondemand.com/oaut` `https://mytenant.authentication.us10.hana.ondemand` • `https://{tenant}.authentication.us10.hana.ondemand` → `https://mytenant.authentication.us10.hana.ondemand` • `https://authentication.myauthserver.com/tenant/{te` → `https://authentication.myauthserver.com/tenant/myt` • `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 `Co` service URL serves multiple tenants.
`tokenServiceUser`	User for basic authentication to OAuth server (if required).

Property	Description
`tokenServicePassword`	Password for `tokenServiceUser` (if required).
(Deprecated) `SystemUser`	User to be used when requesting an access token from the OAuth authorizatio is not specified, the currently logged-in user is used. **⚠ Caution** This property is deprecated and will be removed soon. We recommend that y specific (named) users instead of working with a technical user. As an alternative for technical user communication, we strongly recommend t these authentication types: • Basic Authentication (see Client Authentication Types for HTTP Dest • Client Certificate Authentication (see Client Authentication Types for • OAuth Client Credentials Authentication To extend an OAuth access token's validity, consider using an OAuth refresh t
`authnContextClassRef`	Value of the `AuthnContextClassRef` tag, which is part of generated `OAuth2SAMLBearerAssertion` authentication. For more information, see S .
Additional	
`nameQualifier`	Security domain of the user for which access token is requested.
`companyId`	Company identifier.
`assertionIssuer`	Issuer of the SAML assertion.
`assertionRecipient`	Recipient of the SAML assertion. If not set, the token service URL will be the as
`nameIdFormat`	Value of the `NameIdFormat` tag, which is part of generated `OAuth2SAMLBea` authentication. For more information, see SAML 2.0 specification .
`userIdSource`	When this property is set, the generated SAML2 assertion uses the currently lo for the `NameId` tag. See User Propagation via SAML 2.0 Bearer Assertion Flow
`scope`	The value of the OAuth 2.0 scope parameter, expressed as a list of space-deli strings.

`tokenServiceURL.headers.<header-key>`	A static key prefix used as a namespace grouping of the `tokenServiceUrl` values will be sent to the token service during token retrieval. For each HTTP h add a ***'tokenServiceURL.headers'*** prefix separated by dot delimiter. For exam
	
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.headers.<header-key-1>" : "<header-v`
	`...`
	`"tokenServiceURL.headers.<header-key-N>": "<header-va`
	`}`

`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service retrieval. The minimum va maximum is 60 seconds. If the value exceeds the allowed number, the default used.

Property	Description
`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. The minimum value all maximum is 600 seconds. If the value exceeds the allowed number, the defau used.

`tokenServiceURL.queries.<query-key>`	A static key prefix used as a namespace grouping of `tokenServiceUrl`'s q values will be sent to the token service during token retrieval. For each query p must add a ***'tokenServiceURL.queries'*** prefix separated by dot delimiter. For
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.queries.<query-key-1>" : "<query-val`
	`...`
	`"tokenServiceURL.queries.<query-key-N>": "<query-valu`
	`}`

`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of parameters which are ser request to the token service during token retrieval. For each request, a `tokenS` must be added to the parameter key, separated by dot-delimiter. For example
	Sample Code
	`{`
	`...`
	`"tokenService.body.<param-key-1>" : "<param-value`
	`...`
	`"tokenService.body.<param-key-N>": "<param-value-`
	`}`

`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used for mTLS towards This property is required when using OAuth with X.509 Client Certificates.
`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when Client Certificates.

Property	Description

`URL.headers.<header-key>`	A static key prefix used as a namespace grouping of the URL's HTTP headers sent to the target endpoint. For each HTTP header's key, you must add a `URL.` separated by dot-delimiter. For example:
	
	Sample Code
	`{`
	`...`
	`"URL.headers.<header-key-1>" : "<header-value-1>",`
	`...`
	`"URL.headers.<header-key-N>": "<header-value-N>",`
	`}`
	
	Note
	This is a naming convention. As the call to the target endpoint is performed c service only provides the configured properties. The expectation for the clier logic is to parse and use them. If you are using higher-level libraries and tools support this convention.

`URL.queries.<query-key>`	A static key prefix used as a namespace grouping of URL's query parameters sent to the target endpoint. For each query parameter's key, you must add a UR separated by dot-delimiter. For example:
	
	Sample Code
	`{`
	`...`
	`"URL.queries.<query-key-1>" : "<query-value-1>",`
	`...`
	`"URL.queries.<query-key-N>": "<query-value-N>",`
	`}`
	
	Note
	This is a naming convention. As the call to the target endpoint is performed c service only provides the configured properties. The expectation for the clier logic is to parse and use them. If you are using higher-level libraries and tools support this convention.

`x_user_token.jwks`	Base64-encoded *JSON web key set*, containing the signing keys which are use 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 val in the *X-User-Token* header. **Restriction** If the value is a private endpoint (for example, *localhost*), the Destination ser to perform the verification of the X-User-Token header when using the "Find D For more information, see OpenID Connect Discovery .

Property	Description
`skipUserAttributesPrefixInSAMLAttributes`	If set to true, any additional attributes taken from the OAuth server's user infor the *user_attributes* section, will be added to the assertion without the prefix th service would usually add to them. For more information, see User Propagation Assertion Flow.

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
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.

Sample Code

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

Related Information

Create HTTP Destinations
Destination Examples
User Propagation via SAML 2.0 Bearer Assertion Flow
User Propagation between Cloud Foundry Applications
Exchanging User JWTs via OAuth2UserTokenExchange Destinations

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 authentication type is used for destinations that refer to a service on the Internet, an on-premise system, or a *Private Link* endpoint 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

Used for destinations that refer to a service on the Internet, an on-premise system, or a *Private Link* endpoint 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.

Sample Code

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

Client Certificate Authentication

Used for destinations that refer to a service on the Internet or a *Private Link* endpoint. 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: • `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`.

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

Related Information

Server Certificate Authentication

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-renovated. 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.

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.
`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` or `OnPremise`. If `OnPremise` is used server must be accessed through the Cloud Connector.
`Authentication`	Authentication type. Use `OAuth2ClientCredentials` as value.
`clientId`	Client ID used to retrieve the access token.
`clientSecret`	Client secret for the Client ID.

Property	Description
`tokenServiceURL`	URL of the token service, against which token retrieval is performed. Dependir `tokenServiceURLType`, this property is interpreted in different ways during token retrieval: • For `Dedicated`, the `tokenServiceURL` is used as is. • For `Common`, the `tokenServiceURL` is searched for the tenant place ***{tenant}***. It is resolved as subdomain of the subaccount on whose beh performing the call to the Destination service API for fetching the desti If the placeholder is not found, ***{tenant}*** is processed as a subdomain `tokenServiceURL`. See the Destination service REST API to learn how the subaccount's specified. The subaccount subdomain is mandated during creation of subaccount, see Create a Subaccount (SAP BTP Core documentation). Examples of interpreting of the `tokenServiceURL` for `tokenServiceURL` if the call to the Destination service is done on behalf of a subaccount with sub ***mytenant***: • https://authentication.us10.hana.ondemand.com/oauth/token → https://**mytenant**.authentication.us10.hana.ondemand.com/oauth/tok • https://**{tenant}**.authentication.us10.hana.ondemand.com/oauth/toke https://**mytenant**.authentication.us10.hana.ondemand.com/oauth/tok • https://authentication.myauthserver.com/tenant/**{tenant}**/oauth/toke https://authentication.myauthserver.com/tenant/**mytenant**/oauth/tok • https://oauth.**{tenant}**.myauthserver.com/token → https://oauth.**mytenant**.myauthserver.com/token
`tokenServiceURLType`	Either `Dedicated` (if the `tokenServiceURL` serves only a single tenant), or 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-delir sensitive strings.
`tokenServiceURL.headers.<header-key>`	A static key prefix used as a namespace grouping of the `tokenServiceUrl` headers. Its values will be sent to the token service during token retrieval. For e header's key you must add a ***'tokenServiceURL.headers'*** prefix separated by For example: **Sample Code** `{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-v ... "tokenServiceURL.headers.<header-key-N>": "<header-va }`

Property	Description
`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service retrieval. The minimum va 0, the maximum is 60 seconds. If the value exceeds the allowed number, the d (10 seconds) is used.
`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. The minimum value all maximum is 600 seconds. If the value exceeds the allowed number, the defau seconds) is used.

`tokenServiceURL.queries.<query-key>`	A static key prefix used as a namespace grouping of `tokenServiceUrl`'s qu parameters. Its values will be sent to the token service during token retrieval. F paramester's key you must add a ***'tokenServiceURL.queries'*** prefix separated delimiter. For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.queries.<query-key-1>" : "<query-val`
	`...`
	`"tokenServiceURL.queries.<query-key-N>": "<query-valu`
	`}`

`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of parameters which are ser the token request to the token service during token retrieval. For each request, `tokenService.body` prefix must be added to the parameter key, separated delimiter. For example:
	Sample Code
	`{`
	`...`
	`"tokenService.body.<param-key-1>" : "<param-value`
	`...`
	`"tokenService.body.<param-key-N>": "<param-value-`
	`}`

`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used. This property is using client certificates for authentication. See OAuth with X.509 Client Certific
`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when certificates for authentication. See OAuth with X.509 Client Certificates.
`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be placed in the request body request, rather than the `Authorization` header. Default is `true`. **Note** If set to `false`, but `tokenServiceUser` / `tokenServicePassword` are `tokenServiceUser` / `tokenServicePassword` are taken with priority.

Property	Description
`clientAssertion.type`	When using this destination as a client assertion provider, you can specify the client assertion as defined by the authorization server. The supported values a • "urn:ietf:params:oauth:client-assertion-type:saml2-bearer" => indicati Bearer assertion. • "urn:ietf:params:oauth:client-assertion-type:jwt-bearer" => indicating token. This is used in case of automated client assertion fetching by the service. For more information, see Client Assertion with Automated Assertion Fetching
`clientAssertion.destinationName`	Name of the destination that provides client assertions when using client asse authentication mechanism. Must be on the same subaccount or service instan destination. This is used in case of automated client assertion fetching by the For more information, see Client Assertion with Automated Assertion Fetching
`URL.headers.<header-key>`	A static key prefix used as a namespace grouping of the URL's HTTP headers will be sent to the target endpoint. For each HTTP header's key, you must add `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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li 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 will be sent to the target endpoint. For each query parameter's key, you must 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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.

Note

When the OAuth authorization server is called, it accepts the trust settings of the destination, see Server Certificate Authentication.

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
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
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.

Sample Code

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

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

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

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) or the Destination Service REST API.

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	The URL of the token service, against which the token excha `Token Service URL Type`, this property is interpreted token retrieval: • For `Dedicated`, the token service URL is taken as i • For `Common`, the token service URL is searched for t `{tenant}` is resolved as the subdomain of the sub performing the call. If the placeholder is not found, { the token service URL. See Automated Access Token Retrieval for informati The subaccount subdomain is mandated during crea Subaccount. **Examples** of interpreting the token service URL for the toker the Destination service is on behalf of a subaccount subdon • `https://authentication.us10.hana.onde` `https://mytenant.authentication.us10.` • `https://{tenant}.authentication.us10.` → `https://mytenant.authentication.us10.` • `https://authentication.myauthserver.c` → `https://authentication.myauthserver.c` • `https://oauth.{tenant}.myauthserver.c` `https://oauth.mytenant.myauthserver.c`
`Name`	Name	Name of the destination. Must be unique for the destination
`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

Field/Parameter	JSON Key	Input/Description
`Client ID`	clientId	OAuth 2.0 client ID to be used for the user access token exc
`Authentication`	Authentication	`OAuth2UserTokenExchange` in this case.
`Proxy Type`	ProxyType	You can only use proxy type `Internet` or `OnPremise`. If `O` must be accessed through the Cloud Connector.
`Type`	Type	Choose `HTTP` (for HTTP or HTTPS communication).
`Token Service URL Type`	tokenServiceURLType	• Choose `Dedicated` if the token service URL serves • Choose `Common` if the token service URL serves mu
Optional		
`Description`	Description	Description of the destination.
Additional		
	`scope`	The value of the OAuth 2.0 scope parameter expressed as a strings.
	`tokenServiceURL.headers.<header-key>`	A static key prefix used as a namespace grouping of the `to` values will be sent to the token service during token retrieva add a ***'tokenServiceURL.headers'*** prefix separated by dot **Sample Code** `{ ... "tokenServiceURL.headers.<header-key-1> ... "tokenServiceURL.headers.<header-key-N> }`
	`tokenServiceURL.queries.<query-key>`	A static key prefix used as a namespace grouping of `token` values will be sent to the token service during token retrieva must add a ***'tokenServiceURL.queries'*** prefix separated by **Sample Code** `{ ... "tokenServiceURL.queries.<query-key-1>" ... "tokenServiceURL.queries.<query-key-N>" }`

Field/Parameter	JSON Key	Input/Description

	`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of param request to the token service during token retrieval. For each must be added to the parameter key, separated by dot-deli
		
		Sample Code
		`{`
		`...`
		`"tokenService.body.<param-key-1>" :`
		`...`
		`"tokenService.body.<param-key-N>":`
		`}`

	`URL.headers.<header-key>`	A static key prefix used as a namespace grouping of the UR sent to the target endpoint. For each HTTP header's key, you separated by dot-delimiter. For example:
		
		Sample Code
		`{`
		`...`
		`"URL.headers.<header-key-1>" : "<header`
		`...`
		`"URL.headers.<header-key-N>": "<header-`
		`}`
		
		Note
		This is a naming convention. As the call to the target endp service only provides the configured properties. The expe logic is to parse and use them. If you are using higher-leve support this convention.

	`URL.queries.<query-key>`	A static key prefix used as a namespace grouping of URL's sent to the target endpoint. For each query parameter's key, separated by dot-delimiter. For example:
		
		Sample Code
		`{`
		`...`
		`"URL.queries.<query-key-1>" : "<query-v`
		`...`
		`"URL.queries.<query-key-N>": "<query-va`
		`}`
		
		Note
		This is a naming convention. As the call to the target endp service only provides the configured properties. The expe logic is to parse and use them. If you are using higher-leve support this convention.

Field/Parameter	JSON Key	Input/Description
	`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used client certificates for authentication. See OAuth with X.509 C
	`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if or certificates for authentication. See OAuth with X.509 Client C
	`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be placed in rather than the `Authorization` header. Default is `true`. **Note** If set to `false`, but `tokenServiceUser` / `tokenServ` `tokenServiceUser` / `tokenServicePassword` are t
	`x_user_token.jwks`	Base64-encoded *JSON web key set*, containing the signing 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 wh in the *X-User-Token* header. **Restriction** If the value is a private endpoint (for example, *localhost*), to perform the verification of the X-User-Token header whe For more information, see OpenID Connect Discovery .

Back to Content

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.

Sample Code

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

Back to Content

Related Information

Exchanging User JWTs via OAuth2UserTokenExchange Destinations

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 for on-premise connections.
- OAuth SAML Bearer Assertion Authentication or SAML Assertion Authentication for Internet connections.

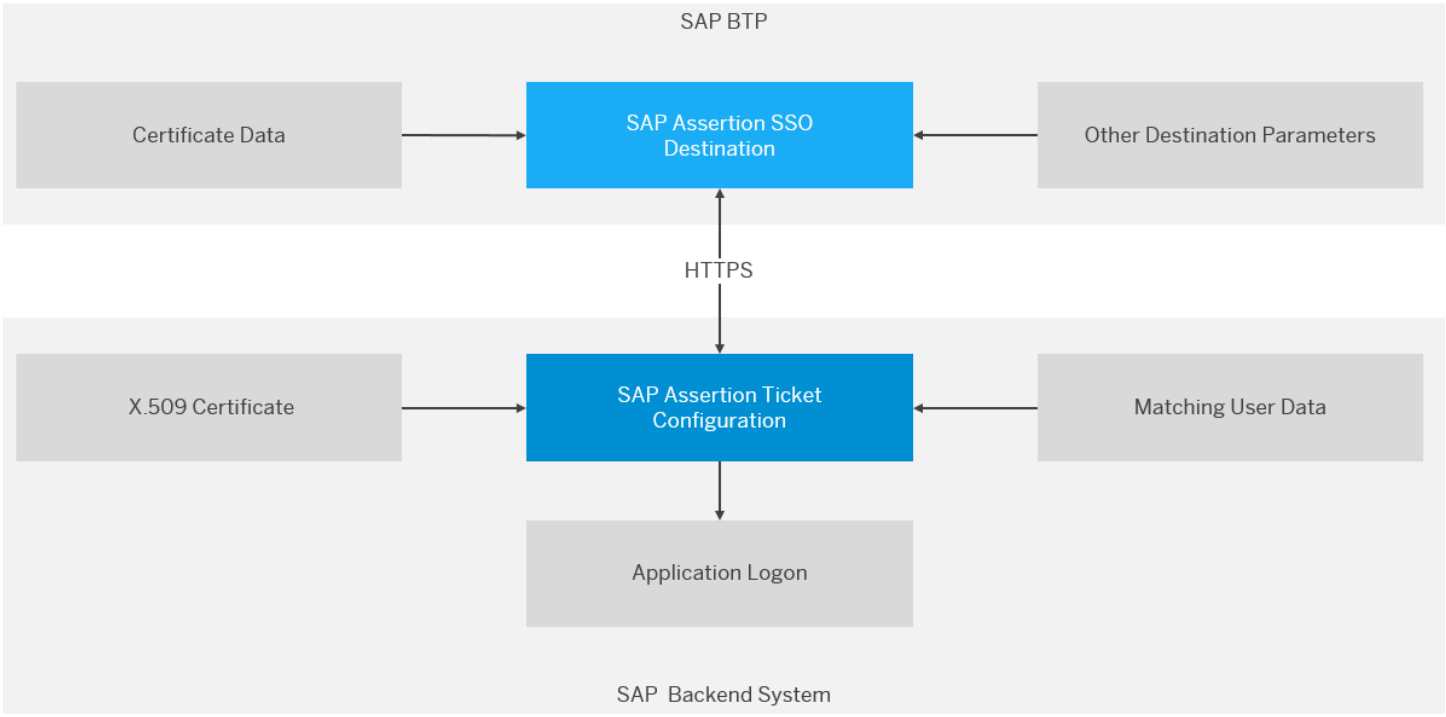
Context

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

ⓘ 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:

- Access the Destinations Editor
- Create HTTP Destinations
- 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, **Destinations** 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.
(Deprecated) `SystemUser`	**Note** Deprecated. This property will be removed soon. Optional property. • 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. Thus, if the current user needs to be propagated, do not use this property.
`ProxyType`	You can use both proxy types `Internet`, `PrivateLink`, and `OnPremise`.
`CloudConnectorLocationId`	Optional property. 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.
Additional	
`userIdSource`	When this property is set, you can choose which claim in a JWT (JSON Web token) to be considered as *<user ID>* field in the generated assertion.

Property	Description
`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 .
`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 .

`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>",`
	`}`
	
	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.

`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>",`
	`}`
	
	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.

Sample Code

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

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

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

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 tools, that is, the console client and **Destinations** 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` or `OnPremise`. If `OnPremise` is used server must be accessed through the Cloud Connector.
`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.
Additional	
`scope`	Value of the OAuth 2.0 `scope` parameter, expressed as a list of space-delimit sensitive strings.

`tokenServiceURL.headers.<header-key>`	Static key prefix used as a namespace grouping of the `tokenServiceUrl`'s headers. Its values will be sent to the token service during token retrieval. For e header's key you must add a ***'tokenServiceURL.headers'*** prefix separated by For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.headers.<header-key-1>" : "<header-v`
	`...`
	`"tokenServiceURL.headers.<header-key-N>": "<header-va`
	`}`

`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service retrieval. The minimum va 0, the maximum is 60 seconds. If the value exceeds the allowed number, the d (10 seconds) is used.
`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. The minimum value all maximum is 600 seconds. If the value exceeds the allowed number, the defau seconds) is used.

Property	Description

`tokenServiceURL.queries.<query-key>`	Static key prefix used as a namespace grouping of `tokenServiceUrl`'s que
	Its values will be sent to the token service during token retrieval. For each quer
	parameter's key you must add a ***'tokenServiceURL.queries'*** prefix separatec
	delimiter. For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.queries.<query-key-1>" : "<query-val`
	`...`
	`"tokenServiceURL.queries.<query-key-N>": "<query-valu`
	`}`

`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of parameters which are ser
	the token request to the token service during token retrieval. For each request,
	`tokenService.body` prefix must be added to the parameter key, separatec
	delimiter. For example:
	Sample Code
	`{`
	`...`
	`"tokenService.body.<param-key-1>" : "<param-value`
	`...`
	`"tokenService.body.<param-key-N>": "<param-value-`
	`}`

`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used. This property is using client certificates for authentication. See OAuth with X.509 Client Certifi
`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when certificates for authentication. See OAuth with X.509 Client Certificates.
`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be placed in the request body request, rather than the `Authorization` header. Default is `true`.
`clientAssertion.destinationName`	Name of the destination that provides client assertions when using client asse authentication mechanism. Must be on the same subaccount or service instan destination. This is used in case of automated client assertion fetching by the For more information, see Client Assertion with Automated Assertion Fetching I

Property	Description
`URL.headers.<header-key>`	Static key prefix used as a namespace grouping of the URL's HTTP headers wl will be sent to the target endpoint. For each HTTP header's key, you must add `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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.
`URL.queries.<query-key>`	Static key prefix used as a namespace grouping of URL's query parameters wł will be sent to the target endpoint. For each query parameter's key, you must 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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.

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.

Back to Content

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.

Sample Code

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

Back to Content

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

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

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), or using the Destination Service REST API. 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
`Client Secret`	clientSecret	OAuth 2.0 client secret to be used for the user access t
`Name`	Name	Name of the destination. Must be unique for the destina
`Proxy Type`	ProxyType	You can only use proxy type `Internet` or `OnPremise` must be accessed through the Cloud Connector.
`Token Service URL`	tokenServiceURL	The URL of the token service, against which the token ex `Token Service URL Type`, this property is interpre token retrieval: • For `Dedicated`, the token service URL is taken • For `Common`, the token service URL is searched `{tenant}` is resolved as the subdomain of the performing the call. If the placeholder is not fou the token service URL. See Automated Access Token Retrieval for infor The subaccount subdomain is mandated during Subaccount. **Examples** of interpreting the token service URL for the t the Destination service is on behalf of a subaccount sub • `https://authentication.us10.hana.o` `https://mytenant.authentication.us` • `https://{tenant}.authentication.us` → `https://mytenant.authentication.us` • `https://authentication.myauthserve` → `https://authentication.myauthserve` • `https://oauth.{tenant}.myauthserve` `https://oauth.mytenant.myauthserve`
`Token Service URL Type`	tokenServiceURLType	• Choose `Dedicated` if the token service URL se • Choose `Common` if the token service URL serves
`Type`	Type	Choose `HTTP` (for HTTP or HTTPS communication).
`URL`	URL	URL of the target endpoint.
Optional		

Field/Parameter (Cockpit)	JSON Key	Description
`Description`	Description	A human-readable description of the destination.
Additional		
	`scope`	The value of the OAuth 2.0 `scope` parameter, expresse strings.

	`tokenServiceURL.headers.<header-key>`	A static key prefix used as a namespace grouping of the values will be sent to the token service during token retr add a ***'tokenServiceURL.headers'*** prefix separated by
		Sample Code
		`{`
		`...`
		`"tokenServiceURL.headers.<header-key`
		`...`
		`"tokenServiceURL.headers.<header-key`
		`}`

	`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service ret maximum is 60 seconds. If the value exceeds the allow used.
	`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. maximum is 600 seconds. If the value exceeds the allo used.

	`tokenServiceURL.queries.<query-key>`	A static key prefix used as a namespace grouping of `to` values will be sent to the token service during token retr must add a ***'tokenServiceURL.queries'*** prefix separate
		Sample Code
		`{`
		`...`
		`"tokenServiceURL.queries.<query-key-`
		`...`
		`"tokenServiceURL.queries.<query-key-`
		`}`

	`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of pa request to the token service during token retrieval. For e must be added to the parameter key, separated by dot-
		Sample Code
		`{`
		`...`
		`"tokenService.body.<param-key-1>`
		`...`
		`"tokenService.body.<param-key-N>`
		`}`

Field/Parameter (Cockpit)	JSON Key	Description
	`x_user_token.jwks`	Base64-encoded *JSON web key set*, containing the sigr 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 key in the *X-User-Token* header. **! Restriction** If the value is a private endpoint (for example, *localhc* to perform the verification of the X-User-Token header For more information, see OpenID Connect Discovery

	`URL.headers.<header-key>`	A static key prefix used as a namespace grouping of the sent to the target endpoint. For each HTTP header's key separated by dot-delimiter. For example:
		
		Sample Code
		`{`
		`...`
		`"URL.headers.<header-key-1>" : "<hea`
		`...`
		`"URL.headers.<header-key-N>": "<head`
		`}`
		
		i Note
		This is a naming convention. As the call to the target e service only provides the configured properties. The e logic is to parse and use them. If you are using higher- support this convention.

	`URL.queries.<query-key>`	A static key prefix used as a namespace grouping of UF sent to the target endpoint. For each query parameter's separated by dot-delimiter. For example:
		
		Sample Code
		`{`
		`...`
		`"URL.queries.<query-key-1>" : "<quer`
		`...`
		`"URL.queries.<query-key-N>": "<query`
		`}`
		
		i Note
		This is a naming convention. As the call to the target e service only provides the configured properties. The e logic is to parse and use them. If you are using higher- support this convention.

Field/Parameter (Cockpit)	JSON Key	Description
	`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be client certificates for authentication. See OAuth with X.5
	`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (certificates for authentication. See OAuth with X.509 Cli
	`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be place rather than the `Authorization` header. Default is `tru`

Back to Content

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.

Sample Code

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

Back to Content

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 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

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`, `PrivateLink`, or `OnPremise`.
`CloudConnectorLocationId`	(only if `ProxyType=OnPremise`) Starting with Cloud Connector 2.9.0, you can connect multiple Cloud Connectors to an account , PrivateLinkas 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, seeSAML 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.
`assertionRecipient`	Recipient 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.
`KeyStoreLocation`	Contains the name of the certificate configuration to be used for *per-destination* SAML assertion signing. This certificate will be used instead of the standard subaccount-wide signing key. For more information, see Set up Trust Between Systems.
`KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when using a *per-destination* SAML assertion signing certificate.
`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 .

Property	Description
`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. **! Restriction** If the value is a private endpoint (for example, *localhost*), the Destination service will not be able to perform the verification of the X-User-Token header when using the "Find Destination" API. For more information, see OpenID Connect Discovery .

`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.

`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.

Property	Description
`skipUserAttributesPrefixInSAMLAttributes`	If set to true, any additional attributes taken from the OAuth server's user information endpoint, under the *user_attributes* section, will be added to the assertion without the prefix that the Destination service would usually add to them. For more information, see User Propagation via SAML 2.0 Bearer Assertion Flow.

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.

Sample Code

```
"authTokens": [
    {
        "type": "SAML2.0",
        "value": "PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZ...",
        "http_header": {
            "key":"Authorization",
            "value":"SAML2.0 PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZ..."
        }
    }
]
```

Related Information

Create HTTP Destinations
Destination Examples
Exchanging User JWTs via OAuth2UserTokenExchange Destinations

OAuth with X.509 Client Certificates

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

Caution

OAuth with X.509 is only supported for *<ProxyType>*=`Internet`.

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
- OAuth Password Authentication
- OAuth User Token Exchange Authentication
- OAuth JWT Bearer Authentication
- OAuth SAML Bearer Assertion Authentication

OAuth Refresh Token Authentication

Create and configure an ***OAuth refresh token*** destination for an application in the Cloud Foundry environment.

Overview

SAP BTP provides support for applications to use the *OAuth2 refresh token* flow for consuming OAuth-protected resources.

Refresh tokens are a common way to maintain certain levels of access, without requiring the use of credentials for getting a new access token. They have a longer validity compared to access tokens and can be used to fetch brand new access tokens without performing again the original flow.

The client credentials and a refresh token are used to request an access token from an OAuth server, referred to below as *token service*. This is automatically performed by the Destination service when using the "Find a destination" REST endpoint.

Properties

The table below lists the destination properties for the ***OAuth refresh token*** authentication type.

Property	Description

Property	Description
Required	
`Name`	Name of the destination. It must be the same as the destination name you use configuration tools, that is, the console client and **Destinations** editor (cockpit
`Type`	Destination type. Choose `HTTP` for all HTTP(S) destinations.
`URL`	The URL of the protected target resource.
`ProxyType`	Choose `Internet` or `OnPremise`. If `OnPremise` is used, the OAuth server n accessed through the Cloud Connector.
`Authentication`	Authentication type. Use `OAuth2RefreshToken` as value.
`clientId`	Client ID used to retrieve the access token.
`clientSecret`	Client secret for the Client ID.
`tokenServiceURL`	Token retrieval URL of the OAuth server.
`tokenServiceURLType`	Either `Dedicated` (if the `tokenServiceURL` serves only a single tenant), or the `tokenServiceURL` serves multiple tenants).
Additional	
`scope`	Value of the OAuth 2.0 `scope` parameter, expressed as a list of space-delimit sensitive strings.

`tokenServiceURL.headers.<header-key>`	Static key prefix used as a namespace grouping of the `tokenServiceUrl`'s headers. Its values will be sent to the token service during token retrieval. For e header's key you must add a ***'tokenServiceURL.headers'*** prefix separated by For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.headers.<header-key-1>" : "<header-v`
	`...`
	`"tokenServiceURL.headers.<header-key-N>": "<header-va`
	`}`

`tokenServiceURL.queries.<query-key>`	Static key prefix used as a namespace grouping of `tokenServiceUrl`'s que Its values will be sent to the token service during token retrieval. For each quer paramester's key you must add a ***'tokenServiceURL.queries'*** prefix separate delimiter. For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.queries.<query-key-1>" : "<query-val`
	`...`
	`"tokenServiceURL.queries.<query-key-N>": "<query-valu`
	`}`

Property	Description
`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of parameters which are ser the token request to the token service during token retrieval. For each request, `tokenService.body` prefix must be added to the parameter key, separatec delimiter. For example: **Sample Code** `{ ... "tokenService.body.<param-key-1>" : "<param-value ... "tokenService.body.<param-key-N>": "<param-value- }`
`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used. This property is using client certificates for authentication. See OAuth with X.509 Client Certific
`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when certificates for authentication. See OAuth with X.509 Client Certificates.
`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be placed in the request body request, rather than the `Authorization` header. Default is `true`.
`URL.headers.<header-key>`	Static key prefix used as a namespace grouping of the URL's HTTP headers wl will be sent to the target endpoint. For each HTTP header's key, you must add `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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.
`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service retrieval. The minimum va 0, the maximum is 60 seconds. If the value exceeds the allowed number, the d (10 seconds) is used.
`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. The minimum value all maximum is 600 seconds. If the value exceeds the allowed number, the defau seconds) is used.

Property	Description
`URL.queries.<query-key>`	Static key prefix used as a namespace grouping of URL's query parameters wh will be sent to the target endpoint. For each query parameter's key, you must 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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.

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.

Example: OAuth2 RefreshToken Destination

Sample Code

```
{
    "Name": "SapOAuthPassGrant",
    "Type": "HTTP",
    "URL": "https://myapp.cfapps.sap.hana.ondemand.com/mypath",
    "ProxyType": "Internet",
    "Authentication": "OAuth2RefreshToken",
    "clientId": "my-client-id",
    "clientSecret": "my-client-pass",
    "tokenServiceURL": "https://authentication.sap.hana.ondemand.com/oauth/token"
}
```

Calling "Find Destination"

When calling the destination an *X-refresh-token* is a required header parameter.

Curl call example

Sample Code

```
curl --location --request GET 'https://<destination>.<environment>.hanavlab.ondemand.com/destina
--header 'X-refresh-token: <refresh_token>' \ # mandatory parameter
--header 'Authorization: Bearer <destination_token>'
```

The response for *Find Destination* will contain an `authTokens` object in the format given below. For more information on the fields in `authTokens`, see "Find Destination" Response Structure.

Sample Code

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

OAuth Authorization Code Authentication

Create and configure an ***OAuth Authorization Code*** destination for an application in the Cloud Foundry environment.

Overview

The *OAuth Authorization Code* flow is a standard mechanism for business user login. It is a two-step procedure. In a first step, business users authenticate themselves towards an authorization server, which grants users an authorization code. The second step exchanges the authorization code for an access token through a token service. Applications can use this flow to access OAuth-protected resources.

The client credentials and an authorization code are used to request an access token from an OAuth server, referred to below as *token service*. This is performed automatically by the Destination service when using the "Find a destination" REST endpoint.

! Restriction

This authentication type is not yet available for destination configuration via the cockpit.

Properties

The table below lists the destination properties for the ***OAuth2AuthorizationCode*** authentication type.

Property	Description
Required	
`Name`	Name of the destination. It must be the same as the destination name you use configuration tools, that is, the console client and **Destinations** editor (cockpit
`Type`	Destination type. Choose `HTTP` for all HTTP(S) destinations.
`URL`	The URL of the protected target resource.
`ProxyType`	Choose `Internet` or `OnPremise`. If `OnPremise` is used, the OAuth server n accessed through the Cloud Connector.

Property	Description
`Authentication`	Authentication type. Use `OAuth2AuthorizationCode` as value.
`clientId`	Client ID of the application.
`clientSecret`	Client secret for the Client ID.
`tokenServiceURL`	Token retrieval URL of the OAuth server.
`tokenServiceURLType`	Either `Dedicated` (if the `tokenServiceURL` serves only a single tenant), or the `tokenServiceURL` serves multiple tenants).
Additional	
`scope`	Value of the OAuth 2.0 `scope` parameter, expressed as a list of space-delimit sensitive strings.

`tokenServiceURL.headers.<header-key>`	Static key prefix used as a namespace grouping of the `tokenServiceUrl`'s headers. Its values will be sent to the token service during token retrieval. For e header's key you must add a ***'tokenServiceURL.headers'*** prefix separated by For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.headers.<header-key-1>" : "<header-v`
	`...`
	`"tokenServiceURL.headers.<header-key-N>": "<header-va`
	`}`

`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service retrieval. The minimum va 0, the maximum is 60 seconds. If the value exceeds the allowed number, the d (10 seconds) is used.
`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. The minimum value all maximum is 600 seconds. If the value exceeds the allowed number, the defau seconds) is used.

`tokenServiceURL.queries.<query-key>`	Static key prefix used as a namespace grouping of `tokenServiceUrl`'s que Its values will be sent to the token service during token retrieval. For each quer paramester's key you must add a ***'tokenServiceURL.queries'*** prefix separated delimiter. For example:
	Sample Code
	`{`
	`...`
	`"tokenServiceURL.queries.<query-key-1>" : "<query-val`
	`...`
	`"tokenServiceURL.queries.<query-key-N>": "<query-valu`
	`}`

Property	Description
`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of parameters which are ser the token request to the token service during token retrieval. For each request, `tokenService.body` prefix must be added to the parameter key, separatec delimiter. For example: **Sample Code** `{ ... "tokenService.body.<param-key-1>" : "<param-value ... "tokenService.body.<param-key-N>": "<param-value- }`
`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used. This property is using client certificates for authentication. See OAuth with X.509 Client Certific
`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when certificates for authentication. See OAuth with X.509 Client Certificates.
`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be placed in the request body request, rather than the `Authorization` header. Default is `true`.
`clientAssertion.destinationName`	Name of the destination that provides client assertions when using client asse authentication mechanism. Must be on the same subaccount or service instanc destination. This is used in case of automated client assertion fetching by the For more information, see Client Assertion with Automated Assertion Fetching E
`URL.headers.<header-key>`	Static key prefix used as a namespace grouping of the URL's HTTP headers wl will be sent to the target endpoint. For each HTTP header's key, you must add `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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.

Property	Description
`URL.queries.<query-key>`	Static key prefix used as a namespace grouping of URL's query parameters wh will be sent to the target endpoint. For each query parameter's key, you must 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>", }` **Note** This is a naming convention. As the call to the target endpoint is performed c side, the service only provides the configured properties. The expectation fo side processing logic is to parse and use them. If you are using higher-level li tools, please check if they support this convention.

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.

Example: OAuth2 Authorization Code Destination

Sample Code

```
{
    "Name": "SapOAuth2AuthorizationCodeDestination",
    "Type": "HTTP",
    "URL": "https://myapp.cfapps.sap.hana.ondemand.com/mypath",
    "ProxyType": "Internet",
    "Authentication": "OAuth2AuthorizationCode",
    "clientId": "my-client-id",
    "clientSecret": "my-client-pass",
    "tokenServiceURL": "https://authentication.sap.hana.ondemand.com/oauth/token"
}
```

Calling "Find Destination"

When calling the destination, an *X-code* is a required header parameter. *X-redirect-uri* and *X-code-verifier* are optional header parameters. They depend on the call for the authorization code fetch. If a redirect URI was specified in that call, the same redirect URI must be used as value for the *X-redirect-uri* header. If a code challenge was presented in the authorization code fetch request, a code verifier must be given as value for the *X-code-verifier* header.

Curl call example

Sample Code

```
curl --location --request GET 'https://<destination>.<environment>.hanavlab.ondemand.com/destinat
--header 'X-code: <authorization_code>' \ # mandatory parameter
--header 'X-redirect-uri: <redirect_uri>' \ # optional parameter
--header 'X-code-verifier: <code_verifier>' \ # optional parameter
--header 'Authorization: Bearer <destination_token>'
```

The response for *Find Destination* will contain an `authTokens` object in the format given below. For more information on the fields in `authTokens`, see "Find Destination" Response Structure.

Sample Code

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

OAuth Technical User Propagation Authentication

Learn about the *OAuth2TechnicalUserPropagation* authentication type for HTTP destinations in the Cloud Foundry environment: use cases, supported properties and examples.

Overview

SAP BTP supports the propagation of technical users from the cloud application towards on-premise systems. In the Destination service, an access token representing the technical user is retrieved, which can then be sent in a header to the Connectivity service. This is similar to *principal propagation*, but in this case, a technical user is propagated instead of a business user.

The retrieval of the access token performs the OAuth 2.0 client credentials flow, according to the token service configurations in the destination. The token service is called from the Internet, not from the Cloud Connector.

Note

The retrieved access token is cached for the duration of its validity.

Restriction

This authentication type is not yet available for destination configuration via the cockpit.

Properties

The table below lists the destination properties for the ***OAuth2TechnicalUserPropagation*** authentication type.

Property	Description
Required	
`Name`	Name of the destination. It must be the same as the destination name you use tools, that is, the console client and **Destinations** editor (cockpit).
`Type`	Destination type. Choose `HTTP` for all HTTP(S) destinations.
`URL`	The URL of the protected target resource.
`ProxyType`	You can only use proxy type `OnPremise`. **→ Remember** The token service is not accessed through the Cloud Connector, but through t
`Authentication`	Authentication type. Use `OAuth2TechnicalUserPropagation` as value.
`clientId`	Client ID of the application.
`clientSecret`	Client secret for the Client ID.
`tokenServiceURL`	The URL of the token service, against which the token exchange is performed. `Token Service URL Type`, this property is interpreted in different ways d token retrieval: • For `Dedicated`, the token service URL is taken as is. • For `Common`, the token service URL is searched for the tenant placehol `{tenant}` is resolved as the subdomain of the subaccount on behalf performing the call. If the placeholder is not found, `{tenant}` is inser the token service URL. Consult the Destination Service REST API to see how the subdomain of specified. The subaccount subdomain is mandated during creation of Create a Subaccount. **Examples** of interpreting the `tokenServiceURL` for `tokenServiceURLTy` to the Destination service is on behalf of a subaccount subdomain with value m • `https://authentication.us10.hana.ondemand.com/oaut` `https://mytenant.authentication.us10.hana.ondemand` • `https://{tenant}.authentication.us10.hana.ondemand` → `https://mytenant.authentication.us10.hana.ondemand` • `https://authentication.myauthserver.com/tenant/{te` → `https://authentication.myauthserver.com/tenant/myt` • `https://oauth.{tenant}.myauthserver.com/token` → `https://oauth.mytenant.myauthserver.com/token`
`tokenServiceURLType`	Either `Dedicated` (if the `tokenServiceURL` serves only a single tenant), or `tokenServiceURL` serves multiple tenants).
`tokenServiceUser`	User for basic authentication to the OAuth server (if required).
`tokenServicePassword`	Password for `tokenServiceUser` (if required).
Additional	

Property	Description
`scope`	Value of the OAuth 2.0 `scope` parameter, expressed as a list of space-delimit strings.
`tokenServiceURL.headers.<header-key>`	Static key prefix used as a namespace grouping of the `tokenServiceUrl`'s values will be sent to the token service during token retrieval. For each HTTP h add a ***'tokenServiceURL.headers'*** prefix separated by dot delimiter. For exam **Sample Code** `{ ... "tokenServiceURL.headers.<header-key-1>" : "<header-v ... "tokenServiceURL.headers.<header-key-N>": "<header-va }`
`tokenServiceURL.ConnectionTimeoutInSeconds`	Defines the connection timeout for the token service retrieval. The minimum va maximum is 60 seconds. If the value exceeds the allowed number, the default used.
`tokenServiceURL.SocketReadTimeoutInSeconds`	Defines the read timeout for the token service retrieval. The minimum value all maximum is 600 seconds. If the value exceeds the allowed number, the defau used.
`tokenServiceURL.queries.<query-key>`	Static key prefix used as a namespace grouping of `tokenServiceUrl`'s que values will be sent to the token service during token retrieval. For each query p must add a ***'tokenServiceURL.queries'*** prefix separated by dot delimiter. For **Sample Code** `{ ... "tokenServiceURL.queries.<query-key-1>" : "<query-val ... "tokenServiceURL.queries.<query-key-N>": "<query-valu }`
`tokenService.body.<param-key>`	A static key prefix used as a namespace grouping of parameters which are ser request to the token service during token retrieval. For each request, a `tokenS` must be added to the parameter key, separated by dot-delimiter. For example **Sample Code** `{ ... "tokenService.body.<param-key-1>" : "<param-value ... "tokenService.body.<param-key-N>": "<param-value- }`
`tokenService.KeyStoreLocation`	Contains the name of the certificate configuration to be used. This property is client certificates for authentication. See OAuth with X.509 Client Certificates.

Property	Description
`tokenService.KeyStorePassword`	Contains the password for the certificate configuration (if one is needed) when certificates for authentication. See OAuth with X.509 Client Certificates.
`tokenService.addClientCredentialsInBody`	Specifies whether the client credentials should be placed in the request body rather than the `Authorization` header. Default is `true`. **Note** If set to `false`, but `tokenServiceUser` / `tokenServicePassword` are `tokenServiceUser` / `tokenServicePassword` will be taken with priori
`clientAssertion.destinationName`	Name of the destination that provides client assertions when using client asse mechanism. Must be on the same subaccount or service instance as this desti case of automated client assertion fetching by the service. For more information, see Client Assertion with Automated Assertion Fetching I

`URL.headers.<header-key>`	Static key prefix used as a namespace grouping of the URL's HTTP headers wl to the target endpoint. For each HTTP header's key, you must add a `URL.heac` by dot-delimiter. For example:
	
	Sample Code
	`{`
	`...`
	`"URL.headers.<header-key-1>" : "<header-value-1>",`
	`...`
	`"URL.headers.<header-key-N>": "<header-value-N>",`
	`}`
	
	Note
	This is a naming convention. As the call to the target endpoint is performed c service only provides the configured properties. The expectation for the clier logic is to parse and use them. If you are using higher-level libraries and tools support this convention.

`URL.queries.<query-key>`	Static key prefix used as a namespace grouping of URL's query parameters wh to the target endpoint. For each query parameter's key, you must add a `URL.q` separated by dot-delimiter. For example:
	
	Sample Code
	`{`
	`...`
	`"URL.queries.<query-key-1>" : "<query-value-1>",`
	`...`
	`"URL.queries.<query-key-N>": "<query-value-N>",`
	`}`
	
	Note
	This is a naming convention. As the call to the target endpoint is performed c service only provides the configured properties. The expectation for the clier logic is to parse and use them. If you are using higher-level libraries and tools support this convention.

ℹ Note

When the OAuth authorization server is called, it accepts the trust settings of the destination. For more information, see Server Certificate Authentication.

⚠ Caution

When using the OAuth service of the SAP BTP *Neo* environment (*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 also set the `clientId` and `clientSecret` values for `tokenServiceUser` and `tokenServicePassword` properties.

Example: OAuth Technical User Propagation Destination

Sample Code

```
Name=technical-user-example
Type=HTTP
URL=http://protected-url.example.com
ProxyType=OnPremise
Authentication=OAuth2TechnicalUserPropagation
clientId=clientId
clientSecret=secret1234
tokenServiceURL=http://authserver.example.com/oauth/token
```

Example: authTokens Object in *Find Destination* Response

The response for *Find Destination* will contain an `authTokens` object in the format given below. For more information on the fields in `authTokens`, see "Find Destination" Response Structure.

For usage of the `SAP-Connectivity-Technical-Authentication` header, see Authentication Types.

Sample Code

```
"authTokens": [
    {
        "type": "Bearer",
        "value": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiI...",
        "http_header": {
            "key":"SAP-Connectivity-Technical-Authentication",
            "value":"Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiI..."
        }
    }
]
```

Using Client Assertion with OAuth Flows

Replace client secrets with client assertions in OAuth flows for destinations in the Cloud Foundry environment.

Client assertion is one of the possible client authentication mechanisms when requesting an access token from an authorization server. When using client assertion, you must provide the `client_assertion` and `client_assertion_type` parameters in the request to an authorization server that supports client assertion for client authentication.

The Destination service supports client assertion as a client authentication mechanism. It sends the client assertion to the authorization server whenever requesting an OAuth token. The assertion can be either generated externally and provided to the Destination service, or it can be retrieved by the Destination service via an additional destination.

For more information, see Provide Client Assertion Properties as Headers and Client Assertion with Automated Assertion Fetching by the Service.

Prerequisites

You must ensure that the target authorization server supports client assertion as authentication mechanism and accepts the provided client assertions after validation. This might require additional setup in the authorization server and the assertion provider.

Provide Client Assertion Properties as Headers

Provide client assertion properties as headers when using client assertion with OAuth flows for a destination.

You can provide the following headers to use client assertion authentication:

Header	Value	Description
`X-client-assertion-type`	Absolute URI	Format of the assertion as defined by the authorization server. The value is an absolute URI. A URN of the form *urn:ietf:params:oauth:client-assertion-type:** is suggested. Examples: • "urn:ietf:params:oauth:client-assertion-type:saml2-bearer" => indicating a SAML Bearer assertion. • "urn:ietf:params:oauth:client-assertion-type:jwt-bearer" => indicating a JWT Bearer token.
`X-client-assertion`	Token	Assertion being used to authenticate the client.

Example: Destination for an OAuth Service Accepting Client Assertion instead of Client Secret

Sample Code

```
Name=sap_Destination
Type=HTTP
URL= https://xxxx.example.com
ProxyType=Internet
Authentication=OAuth2ClientCredentials
```

```
clientId=clientId
tokenServiceURL=https://authserver.example.com/oauth/token/
```

Example: Client Assertion Properties as Headers

To use the client assertion mechanism in the *Find Destination* API, you must add two mandatory headers as shown in the following example:

Sample Code

```
curl --location --request GET 'https://<destination_service_host>/destination-configuration/v1/d
--header 'X-client-assertion-type: urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \ # ma
--header 'X-client-assertion: eyJraWQiOiJKMWpC...' \ # mandatory parameter
--header 'Authorization: Bearer <access_token>'
```

Client Assertion with Automated Assertion Fetching by the Service

Use the *Find Destination* API to fetch client assertions automatically when using client assertion with OAuth flows for a destination.

The *Find Destination* API lets you fetch client assertions automatically and then use them for retrieving tokens from authorization servers that accept client assertions as a client authentication mechanism.

To apply this mechanism, you must use the following of destination configurations:

- Destination that provides the client assertion - with specified token service that issues client assertions
- Destination that uses the client assertion - with specified token service that uses client assertion as a client authentication mechanism

⚠ Caution

Only *OAuth2ClientCredentials* authentication type is allowed for destinations that provide client assertions.The following authentication types are supported for destinations that use client assertions: *OAuth2Password*, *OAuth2ClientCredentials*, *OAuth2AuthorizationCode*, *OAuth2TechnicalUserPropagation*.

Define the Client Assertion Type

The client assertion type must be defined as a property in the destination that provides the client assertion:

Property	Value	Description

Property	Value	Description
`clientAssertion.type`	Absolute URI	The format of the assertion as defined by the authorization server. The supported values are: • "urn:ietf:params:oauth:client-assertion-type:saml2-bearer" => indicating a SAML Bearer Assertion. • "urn:ietf:params:oauth:client-assertion-type:jwt-bearer" => indicating a JWT Bearer Token.

Use a Property to Add a Reference to the Destination that Provides the Client Assertion

The destination that provides the client assertion can be specified in a property of the destination that uses client assertions:

Property	Value	Description
`clientAssertion.destinationName`	Name of a destination	Name of the destination that provides the client assertion. Must be on the same subaccount or service instance as the destination that uses client assertions.

⚠ Caution

If headers `X-client-assertion` and `X-client-assertion-type` are specified in the *Find Destination* API call, the `clientAssertion.destinationName` property will not be used for an automated assertion fetching mechanism.

Use a Header to Specify the Destination that Provides the Client Assertion

Alternatively, the destination that provides the client assertion can also be specified in a header in the *Find Destination* API.

Header	Value	Description
`X-client-assertion-destination-name`	Name of a destination	Name of the destination that provides the client assertion. Must be on the same subaccount or service instance as the destination that uses client assertions.

If specified, this header overrides the property `clientAssertion.destinationName` in the destination that uses client assertions.

⚠ Caution

Usage of headers `X-client-assertion` and `X-client-assertion-type` is not allowed if this header is present.

Example: Destination that Provides Client Assertions

Sample Code

```
Name=Provides_Client_Assertion_Destination
Type=HTTP
```

```
URL= https://xxxx.example.com
ProxyType=Internet
Authentication=OAuth2ClientCredentials
clientId=clientId
clientSecret=secret1234
tokenServiceURL=https://authserver1.example.com/oauth/token/
clientAssertion.type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer
```

Example: Destination that Uses Client Assertions

Sample Code

```
Name=Uses_Client_Assertion_Destination
Type=HTTP
URL= https://xxxx.example.com
ProxyType=Internet
Authentication=OAuth2ClientCredentials
clientId=clientId
tokenServiceURL=https://authserver2.example.com/oauth/token/
clientAssertion.destinationName=Provides_Client_Assertion_Destination
```

Example: *Find Destination* API Call for Client Assertion Authentication with Automated Assertion Fetching

curl call:

Sample Code

```
curl --location --request GET 'https://<destination_service_host>/destination-configuration/v1/d
--header 'Authorization: Bearer <access_token>'
```

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
- Pooling Configuration
- Repository Configuration
- Target System Configuration
- Parameters Influencing Communication Behavior

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

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

Property	Description
`jco.client.passwd`	Represents the password of the user that is used. **Note** 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. **Note** When working with the **Destinations** editor in the cockpit, enter this password in the *<Password>* field. Do not enter it as additional property.
`jco.client.tls_client_certificate_logon`	When set to `1`, the client certificate provided by the *KeyStore*, 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). The default value is `0`. **Note** When working with the Destinations editor in the cockpit, the *<User>*, *<Alias User>* and *<Password>* fields are hidden when setting the property to `1`. For more information on WebSocket RFC, see also: WebSocket RFC
`jco.destination.auth_type`	Optional property. • 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. **Note** 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.

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`.
`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. The default value is 30000 (30 seconds).
`jco.destination.expiration_time`	Represents the time in milliseconds after which idle connections that are available in the pool can be closed. The default value is 60000 (60 seconds).
`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. The default value is 60000 (60 seconds).
`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.

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
`jco.destination.repository_destination`	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.
`jco.destination.repository.user`	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. **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.
`jco.destination.repository.passwd`	Represents the password for a repository user. If you use such a user, this property is mandatory. **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.

Target System Configuration

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

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

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)

> **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

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 communciation is done over a WebSocket connection.

Back to Content

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
`jco.client.ashost`	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 **Access Control** configuration. The property indicates that a direct connection is established.

Property	Description
`jco.client.sysnr`	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 **Access Control** configuration. **Note** The virtual port in the above access control entry must be named `sapgw<##>`, where *<##>* is the value of `sysnr`.
`jco.client.client`	Three-digit ABAP client number. Defines the client of the target ABAP system.

Back to Content

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 **Access Control** 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 **Access Control** configuration. **Note** The virtual port in the above access control entry must be named `sapms<###>`, where *<###>* is the value of `r3name`.
`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 **Access Control** configuration. You can therefore avoid lookups 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

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`. **Note** We recommend that you **do not use value `1` ("trust all") in productive scenarios**, but only for demo/test purposes.
<Trust Store Location> 1. When used in local environment 2. When used in cloud environment	If you don't want to use the default JDK trust store (option **Use default JDK truststore** 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. 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. **Note** If the *<Trust Store Location>* is not specified, the JDK trust store is used as a default trust store for the destination.
<Trust Store Password>	Password for the JKS trust store file. This field is mandatory if *<Trust Store Location>* is used.

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.

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

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. **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.
`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 ***fast RFC serialization*** is used, as long as the partner system supports it, see SAP Note 2372888. When choosing the `rowBased` option, ***classic*** or ***basXML*** 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`.

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: The user is propagated from a cloud application to an on-premise system using a destination configuration with authentication type `PrincipalPropagation`.

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

- Scenario: Cloud to Cloud: 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
- Create RFC Destinations

Scenario: Cloud to On-Premise

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

Concept

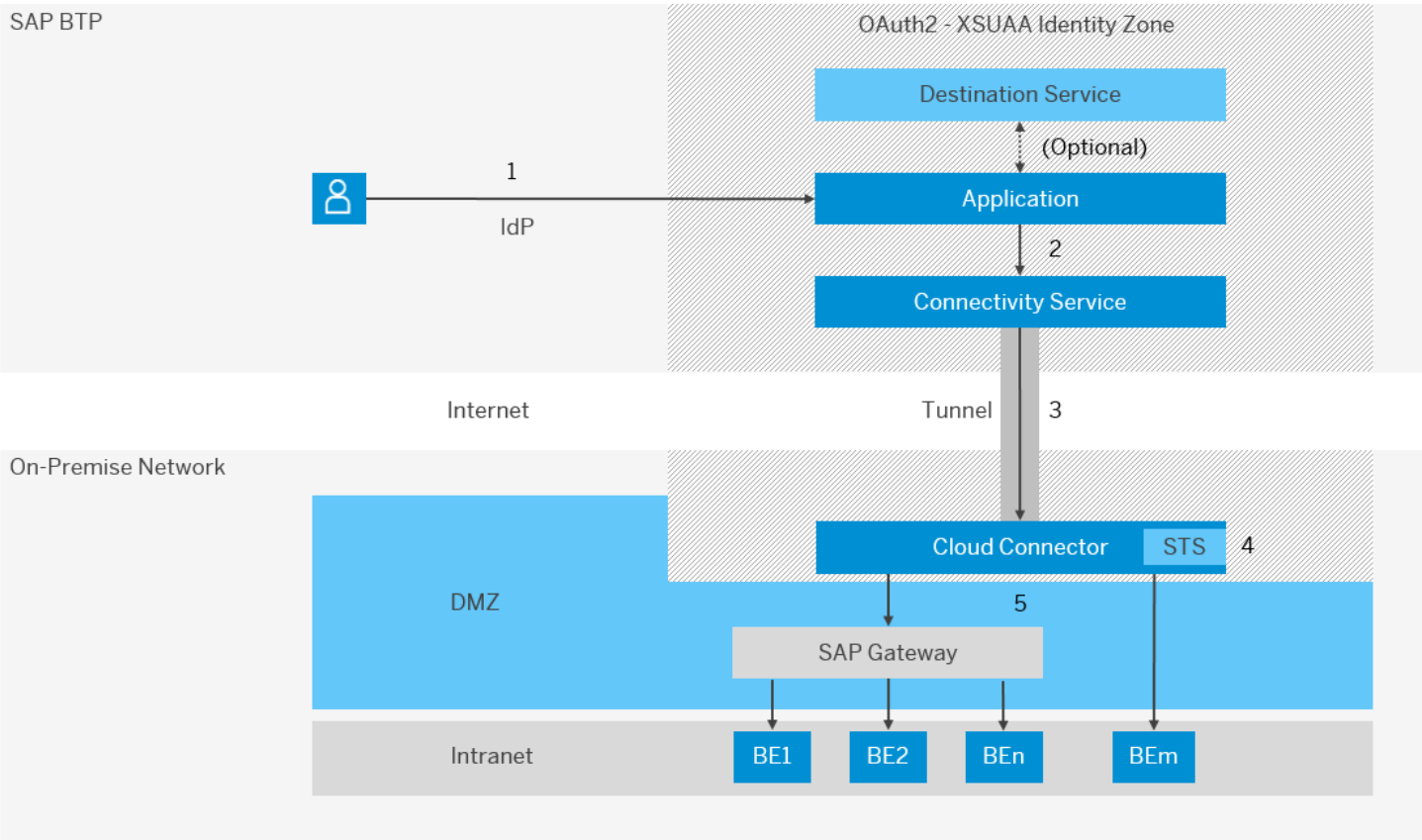
The Connectivity service lets you connect your cloud applications to on-premise systems through the Cloud Connector and forward the identity of a cloud user. This process is called principal propagation (also known as user propagation). The JSON Web token (JWT) representing the cloud user identity is 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.

> **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.
    - 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.
    - If you are using RFC as communication protocol usa 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.
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
Operator	Configuring Principal Propagation (Cloud Connector)

Task Type	Task
Developer	Use cases: • HTTP communication: Configure Principal Propagation via User Exchange Token (Connectivity service) • RFC communication: Configure Principal Propagation for RFC

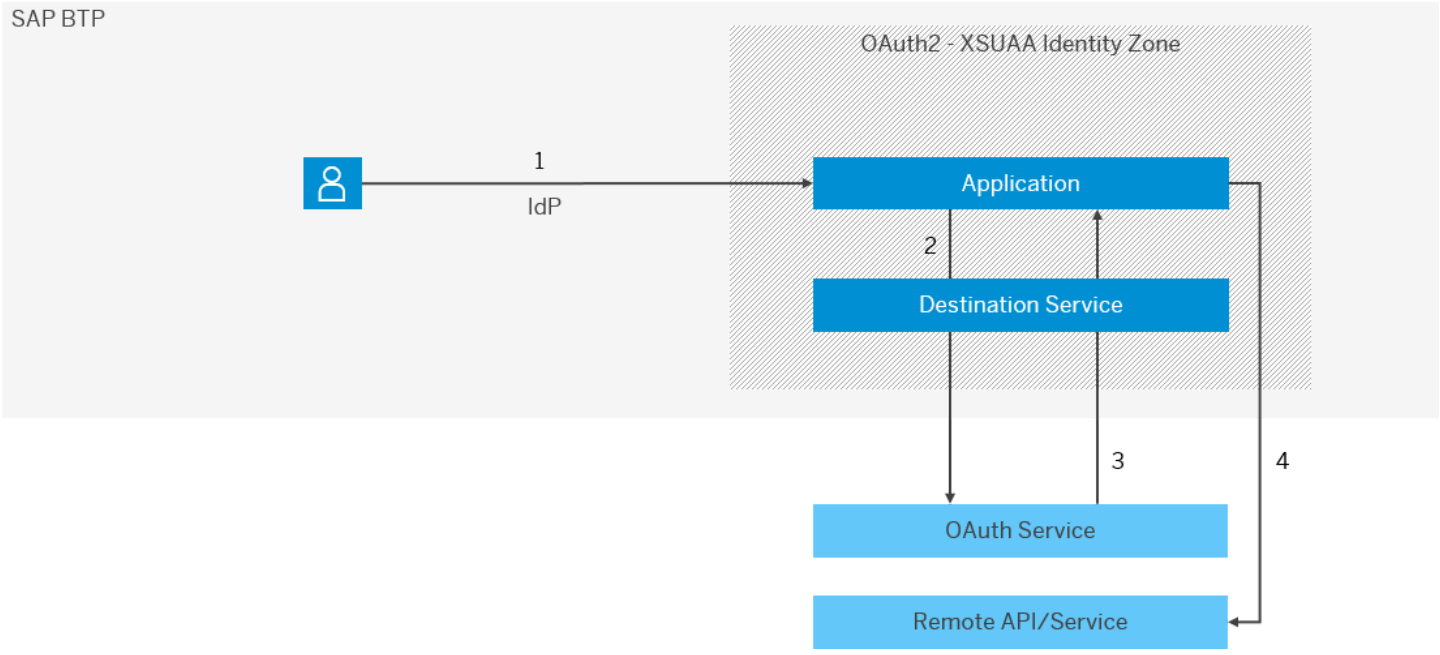
Scenario: Cloud to Cloud

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
Operator	Set up Trust Between Systems
Operator and/or Developer	User Propagation via SAML 2.0 Bearer Assertion Flow (Destination service)

Use Cases: Cloud to Cloud

- User Propagation from the Cloud Foundry Environment to SAP S/4HANA Cloud
- User Propagation from the Cloud Foundry Environment to SAP SuccessFactors
- User Propagation between Cloud Foundry Applications
- User Propagation from the Cloud Foundry Environment to the Neo Environment

Set up Trust Between Systems

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

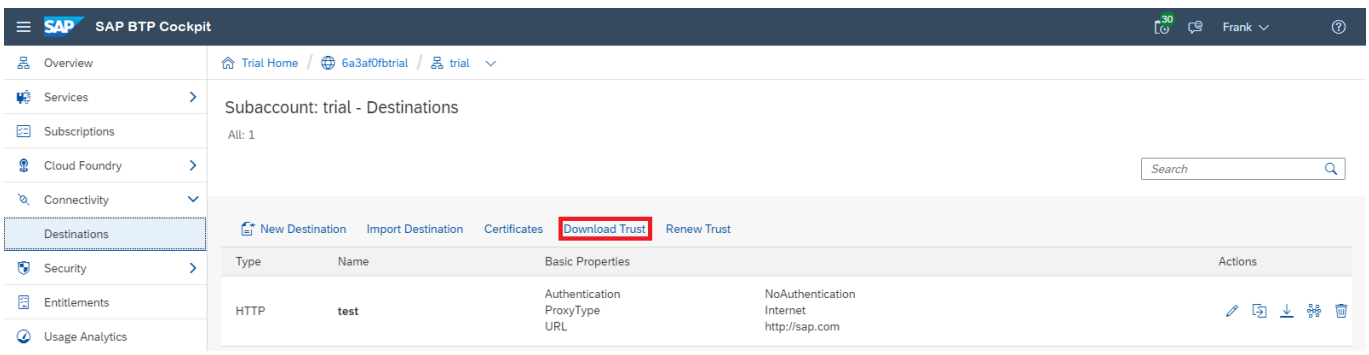
Setting up a trust scenario for user propagation requires the exchange of public keys and certificates between the affected systems, as well as the respective trust configuration within these systems. This enables you to use an HTTP destination with authentication type `OAuth2SAMLBearerAssertion` for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

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

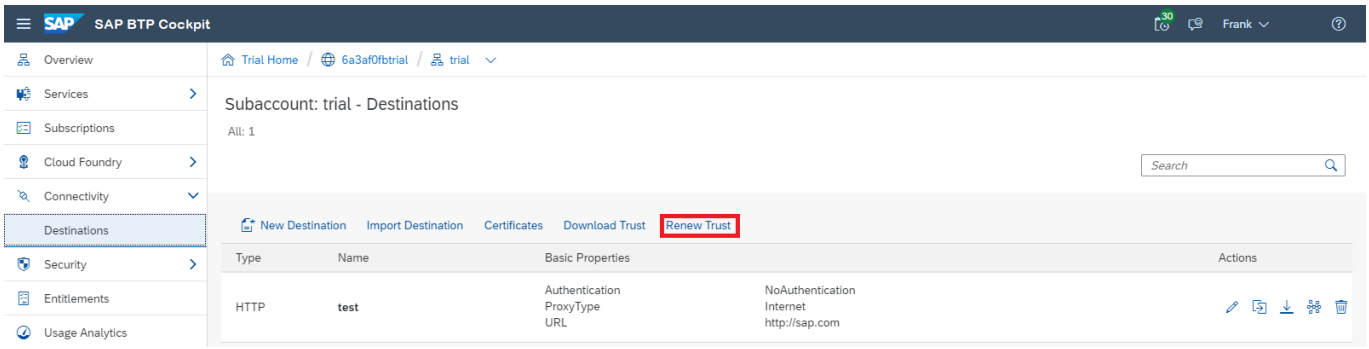


5. Configure the downloaded X.509 certificate in the target system to which you want to propagate the user.

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.

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**.
4. Choose the **Renew Trust** button to trigger a renewal of the existing X509 certificate.



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

Rotate Certificates

You can rotate the identifying X.509 certificate of the subaccount. Rotation is done by creating a passive X.509 certificate for the subaccount, configuring it in the target system to which you want to propagate the user, and rotating it with the active one. After rotation is performed, the active X.509 certificate becomes passive and the passive one active.

ℹ Note

The passive X.509 certificate and the certificate rotation can be managed only via the Destination service REST API. For more information, see Destination Service REST API.

Procedure

1. Generate or renew the passive X.509 certificate: `POST /saml2Metadata/certificate/passive`.
2. Download and save locally the passive X.509 certificate: `GET /saml2Metadata/certificate/passive`.
3. Configure the downloaded X.509 passive certificate in the target system you want to propagate the user to.
4. Rotate the active certificate, making the active one passive and the passive one active: `POST /saml2Metadata/rotateCertificate`.
5. Check that user propagation is working correctly. If it is not, you can rotate the certificates again until fixing the issue.
6. (Optional): Delete the passive X.509 certificate, which used to be active before rotation: `DELETE /saml2Metadata/certificate/passive`.
7. (Optional): Delete the passive X.509 certificate, which used to be active before rotation, from the target system.

Related Information

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

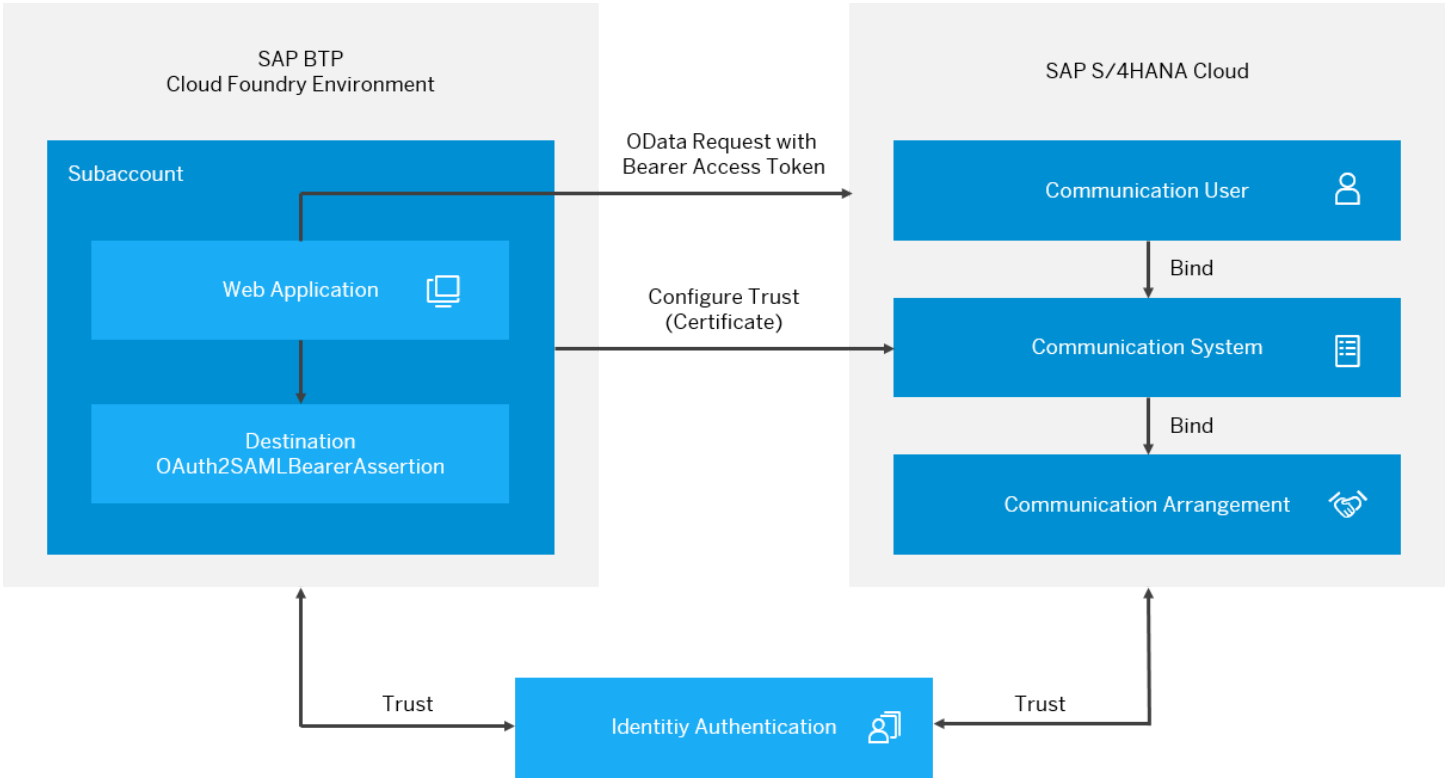
Configure user propagation (single sign-on), using OAuth communication from the SAP BTP Cloud Foundry environment to S/4HANA Cloud. As OAuth mechanism, you use the ***OAuth 2.0 SAML Bearer Assertion Flow***.

Steps

Scenario

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

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
2. Configure OAuth Communication
3. Configure Communication Settings in S/4HANA Cloud
4. Configure Communication Settings in SAP BTP
5. Consume the Destination and Execute the Scenario

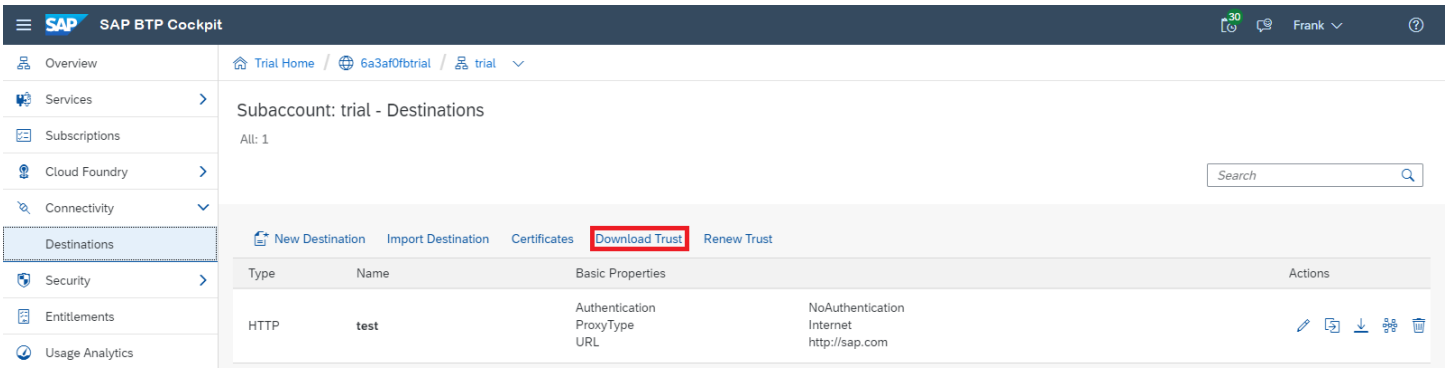
Configure Single Sign-On between S/4HANA Cloud and the Cloud Foundry Organization on SAP BTP

To configure SSO with S/4HANA you must configure trust between the S/4HANA IAS tenant and theCloud 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.



Back to Tasks

Configure Communication Settings in S/4HANA Cloud

1. **Create a Communication User**

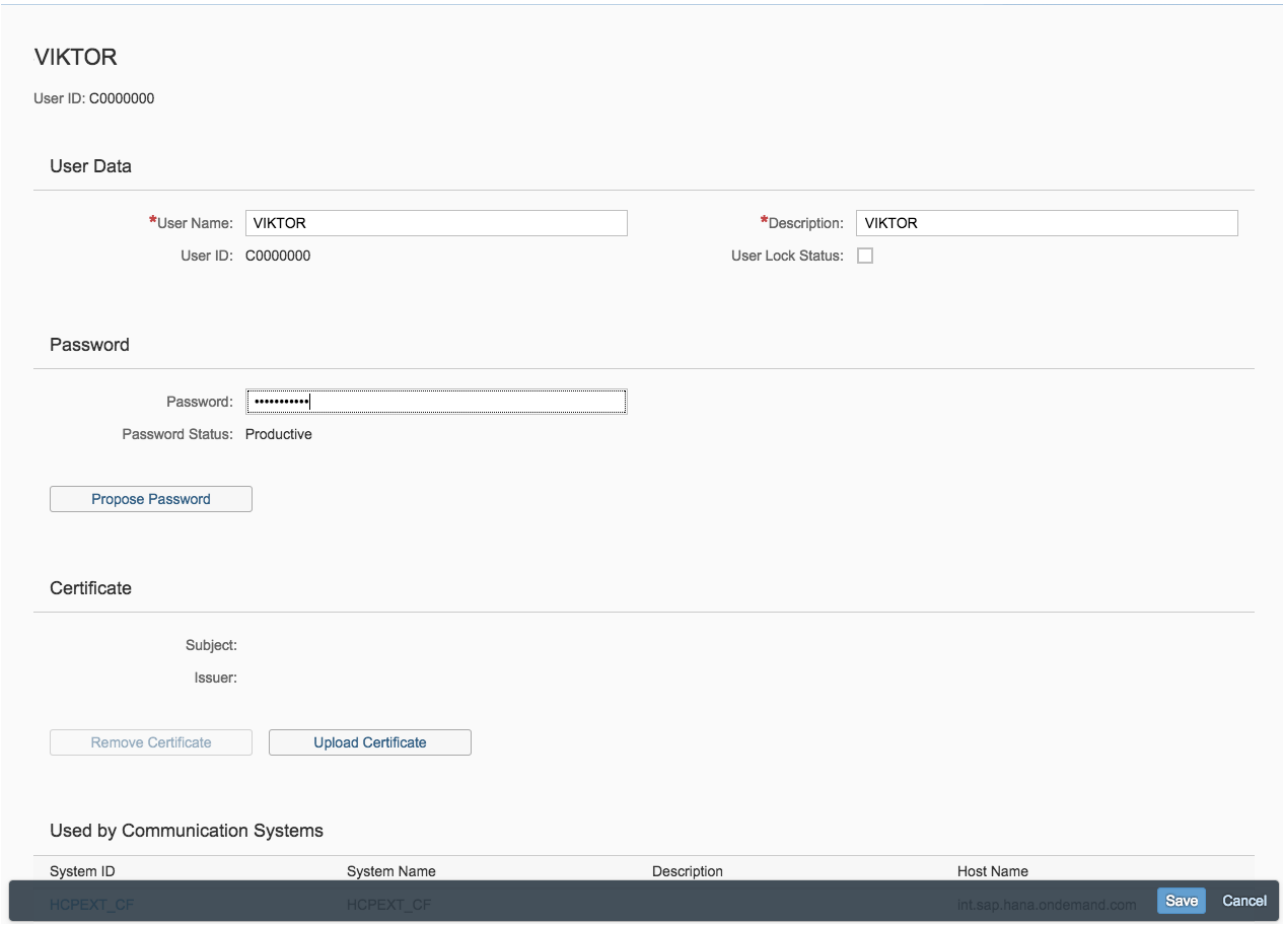
   a. In your S/4HANA Cloud launchpad, choose the application **Maintain Communication Users**.

   b. From the **User List** view, create a new user.

   c. Set *<User Name>*, *<Password>* and *<Description>*.

   d. Copy this password, you will need it in a later step.

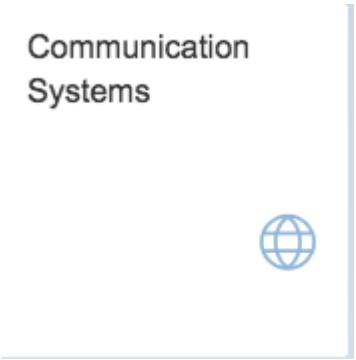
   e. Press the **Save** on the bottom of the screen.



f. Close the **Communication Users** application.

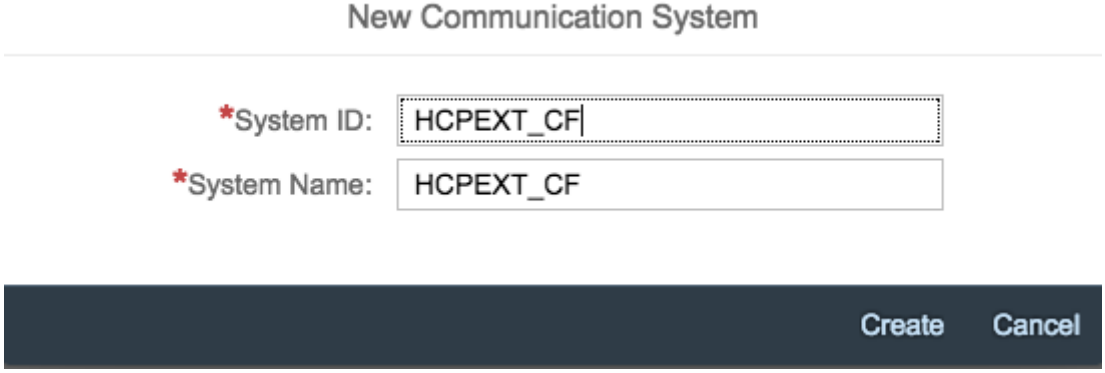
2. **Set up a Communication System for OAuth**

a. From the launchpad, choose the application **Communication Systems**.



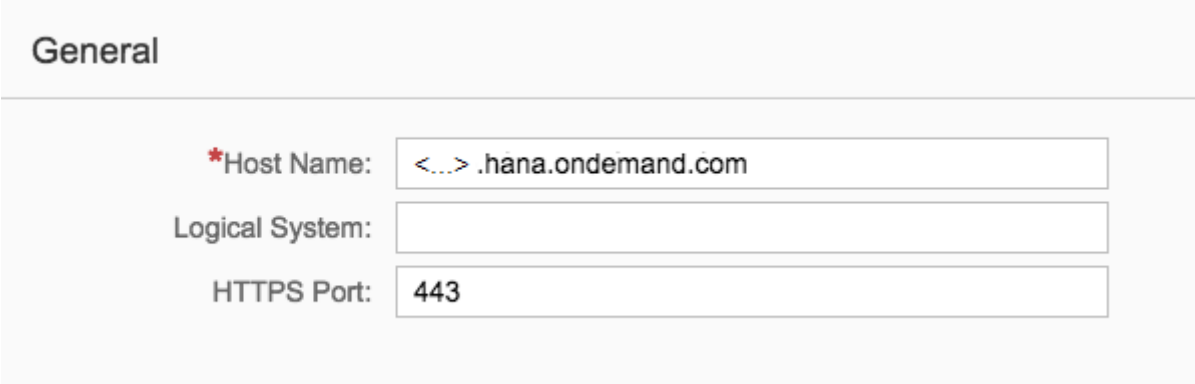
b. From the list view, select **New**.

c. A popup window appears. Enter the *<System ID>* and the *<System Name>*, then choose **Create**.

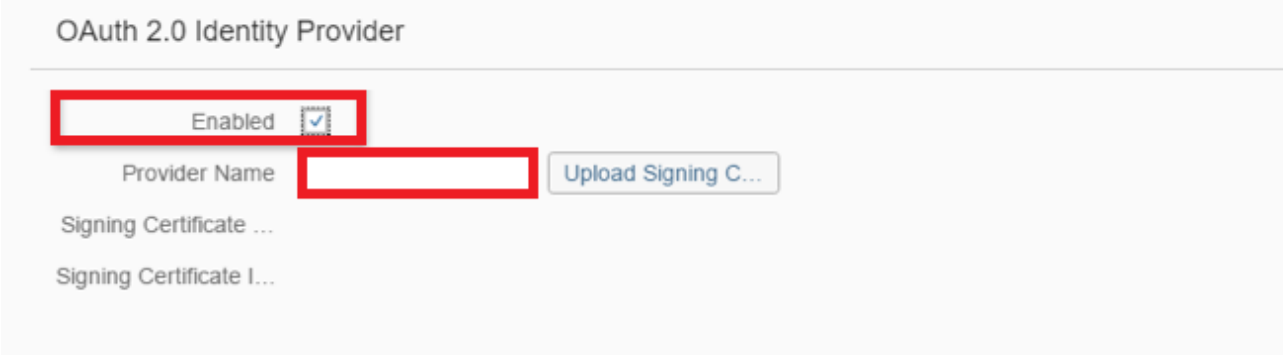


d. Enter the host name. This is your Cloud Foundry region, for example: `cf.eu10.hana.ondemand.com` for Europe (Frankfurt).

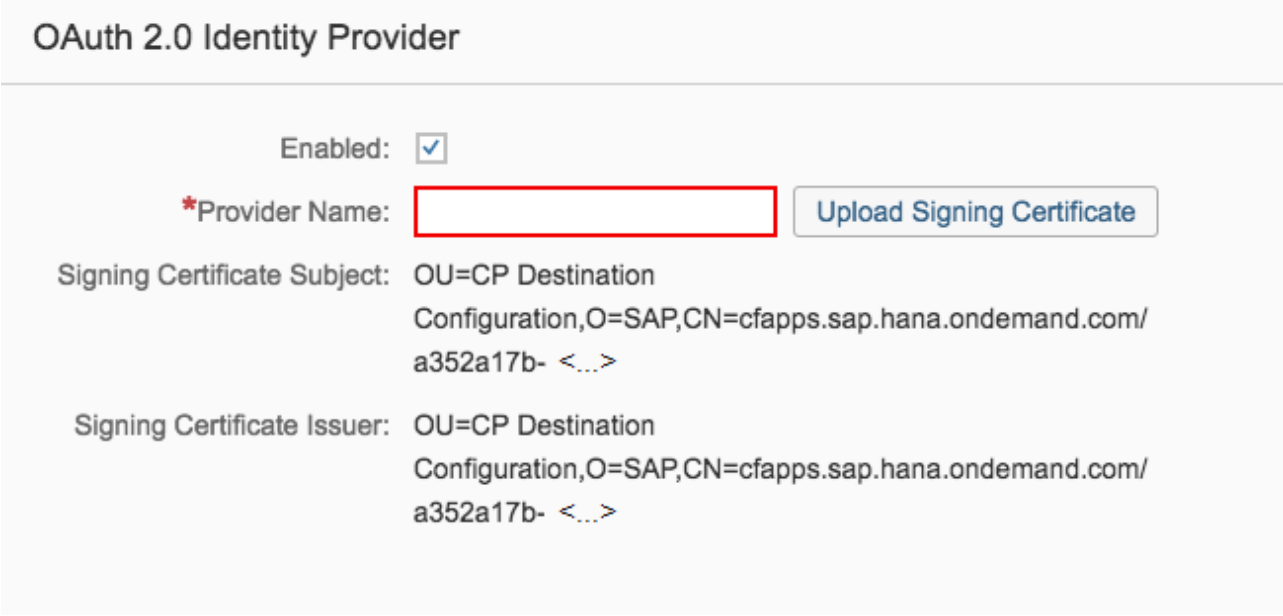
> **Note**
>
> For the complete list of standard regions, see Regions.



e. Enable the OAuth Identity Provider.

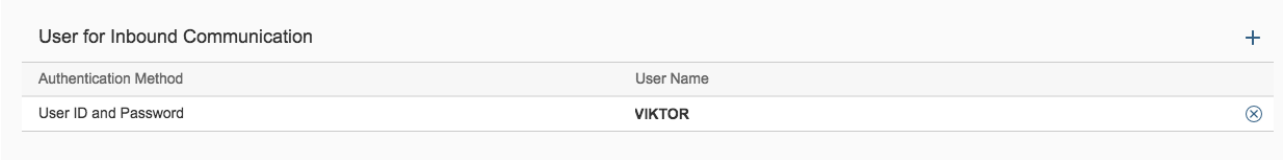


f. Upload the subaccount certificate that you have downloaded before from the SAP BTP cockpit.



g. From *<Signing Certificate Subject>*, copy the `CN` value (for example: `cfapps.sap.hana.ondemand.com/a352a17b-<...>`) and paste it in the field *<Provider Name>*.

h. Add the **Communication User** you have created in the previous step.



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

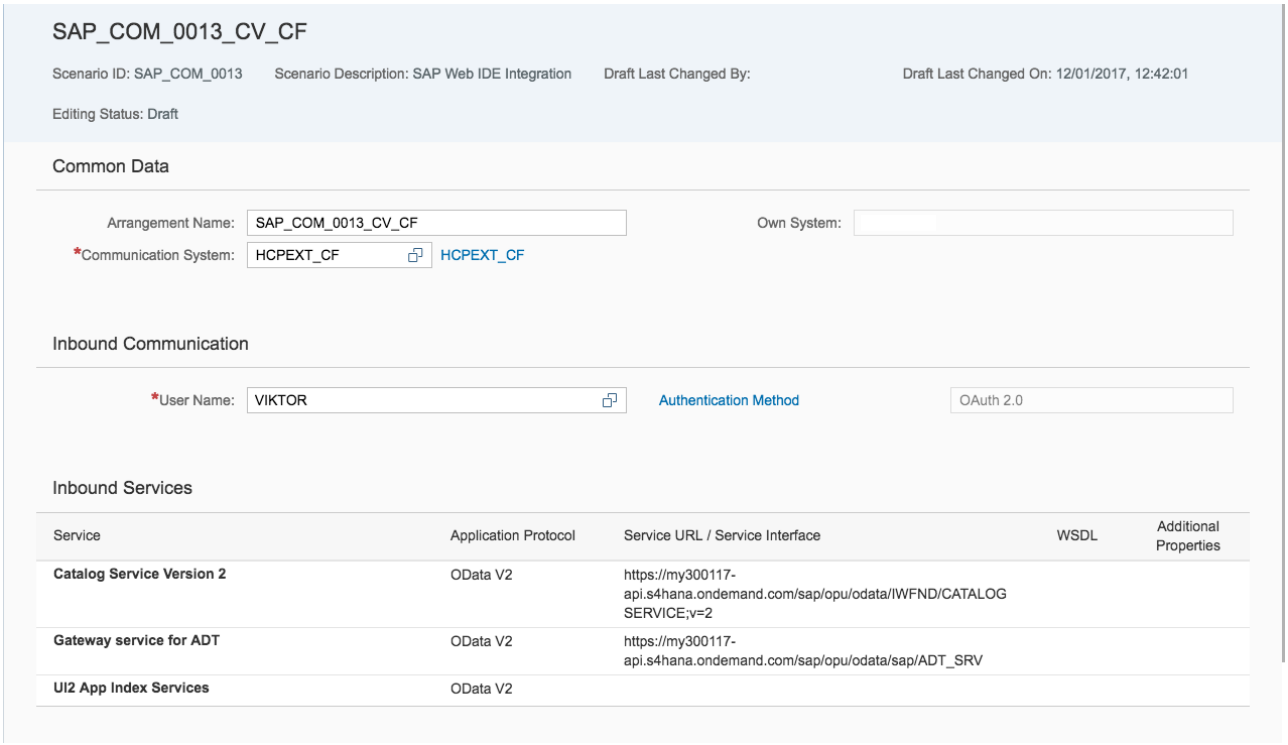
3. **Create a Communication Arrangement**

a. Start the Communication Arrangements application.

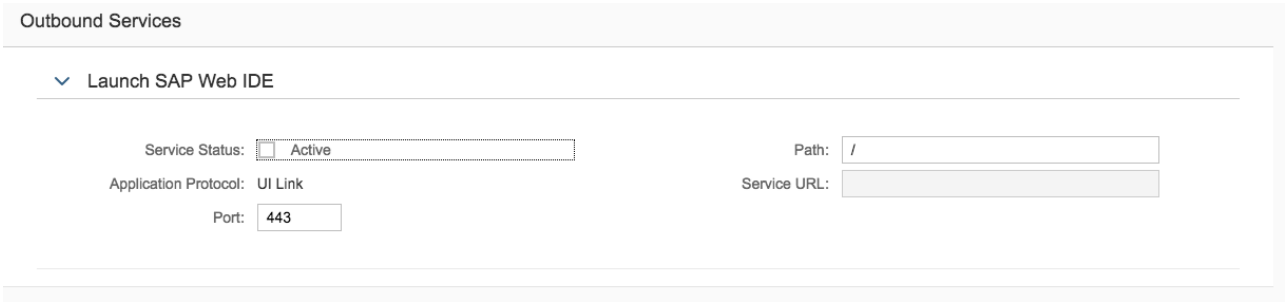
b. From the list view, select New.

c. 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`.

d. 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`.



e. In the Outbound Services section, go to Launch SAP Web IDE and uncheck the ***Active*** checkbox of the field *<Service Status>*.



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

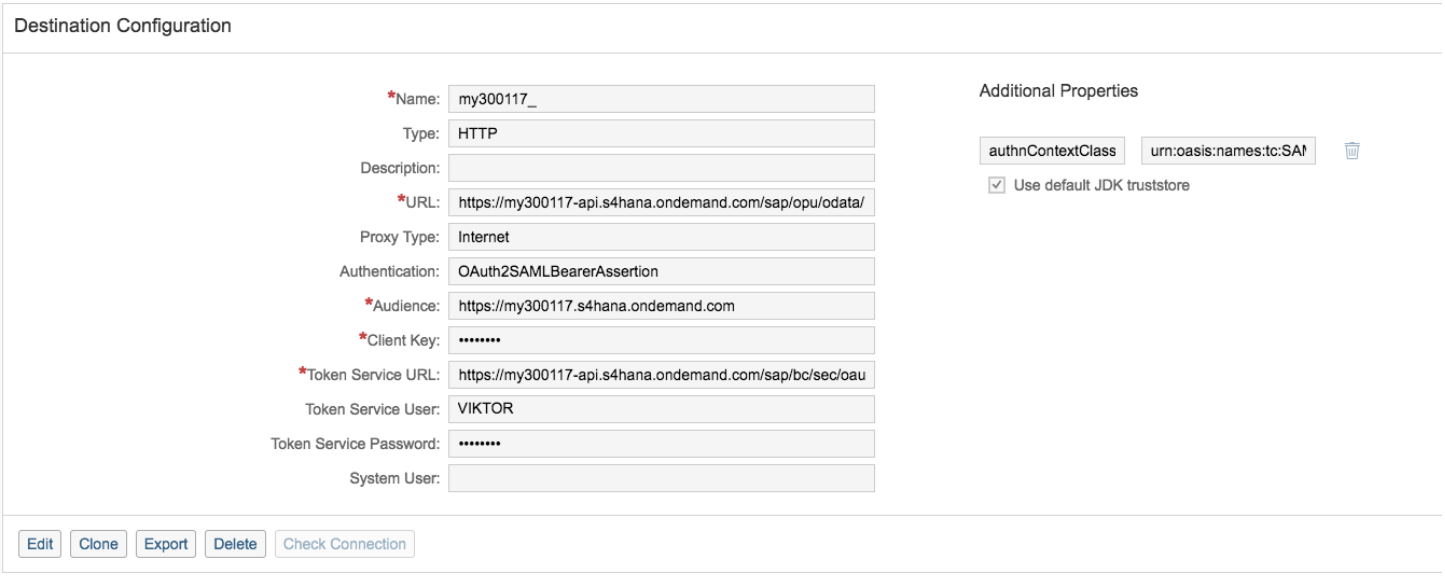
Back to Tasks

Configure Communication Settings in SAP BTP

1. From the SAP BTP cockpit, choose **Cloud Foundry environment** > **your global account**.
2. Choose your subaccount, and from the left-side subaccount menu, go to **Connectivity** > **Destinations**.
3. Press the **New Destination** button.
4. Enter the following parameters for your destination:

Parameter	Value
`Name`	Enter a meaningful name.
`Type`	`HTTP`
`Description`	(Optional) Enter a meaningful description.
`URL`	The OData URL, for example `https://my300117-api.s4hana.ondemand.com/sap/opu/odata/IWFND/CATALOGSERVICE;v=2?$format=json`
`Proxy Type`	`Internet`
`Authentication`	`OAuth2SAMLBearerAssertion`
`Audience`	The URL of your SAP S/4HANA Cloud account. To get it, log on to your SAP S/4HANA Cloud account. Select the profile picture. Then choose **Settings** and copy the value from the *<Server>* field. Add `https://` to the beginning of this string, for example, `https://my300117.s4hana.ondemand.com`. **ⓘ Note** This URL does not contain `my300117-`***`api`***, but only `my300117`.
`Client Key`	The name of the communication user you have in the SAP S/4HANA ABAP tenant, e.g VIKTOR.
`Token Service URL`	For this field, you need the part of the URL before `/sap/...` that you copied before from **Communications Arrangements** service URL/service interface: `https://my300117-api.s4hana.ondemand.com/sap/bc/sec/oauth2/token`
`Token Service User`	The same user as for the `Client Key` parameter.
`Token Service Password`	The password for the communication user.
`System User`	This parameter is not used, leave the field empty.

Parameter	Value
authnContextClassRef	urn:oasis:names:tc:SAML:2.0:ac:classes:X509



Back to Tasks

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. 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 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 for details on the structure of the response from the Destination service.

Back to Tasks

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

Prerequisites

Concept Overview

Create an OAuth Client in SAP SuccessFactors

Create and Consume a Destination for the Cloud Foundry Application

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

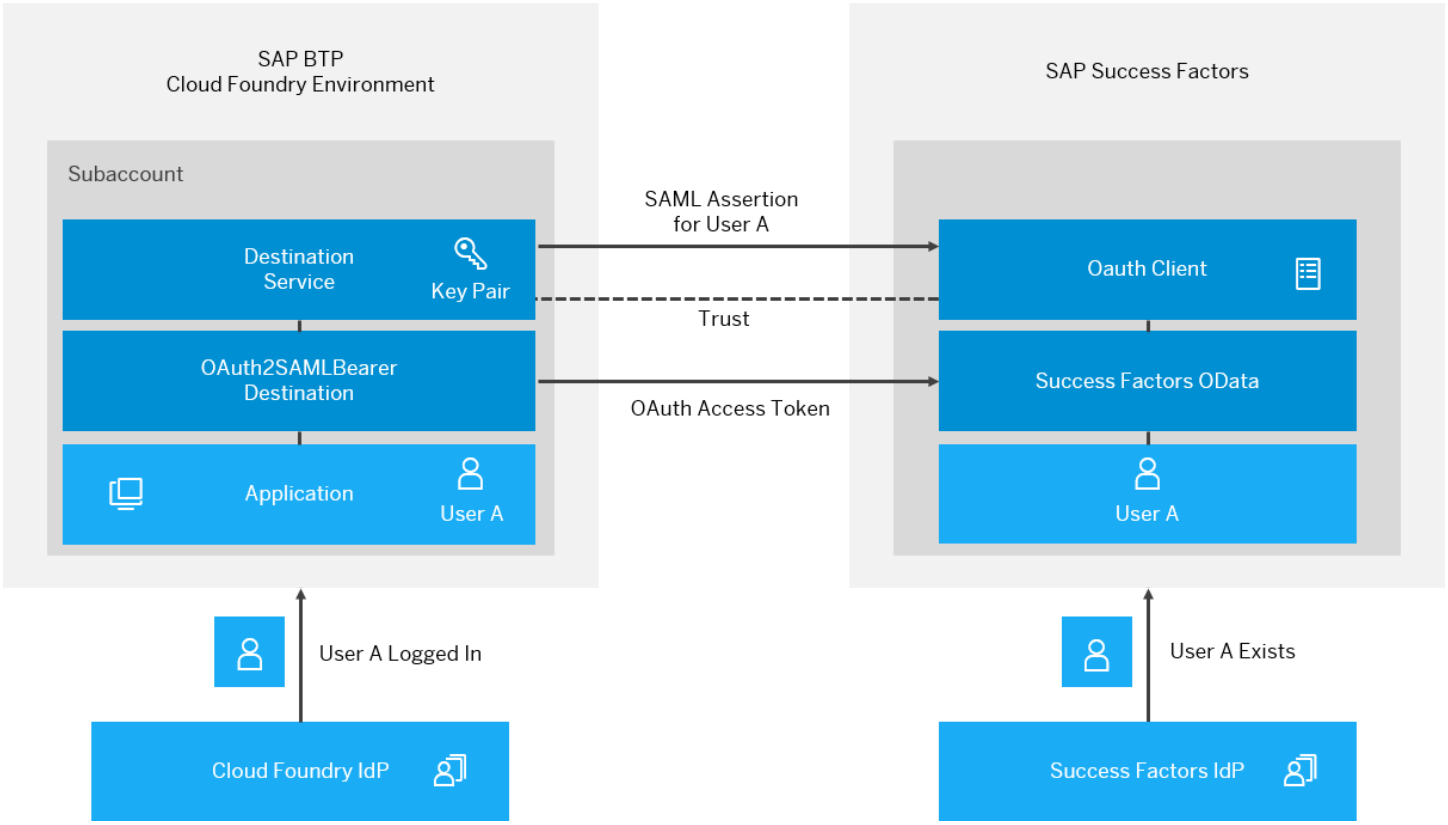
A user logs in to the Cloud Foundry application. Its identity is established by an **Identity Provider** (IdP). This could be the default IdP for the Cloud Foundry subaccount or a trusted IdP, for example the SuccessFactors IdP.

When the application retrieves an `OAuth2SAMLBearer` 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

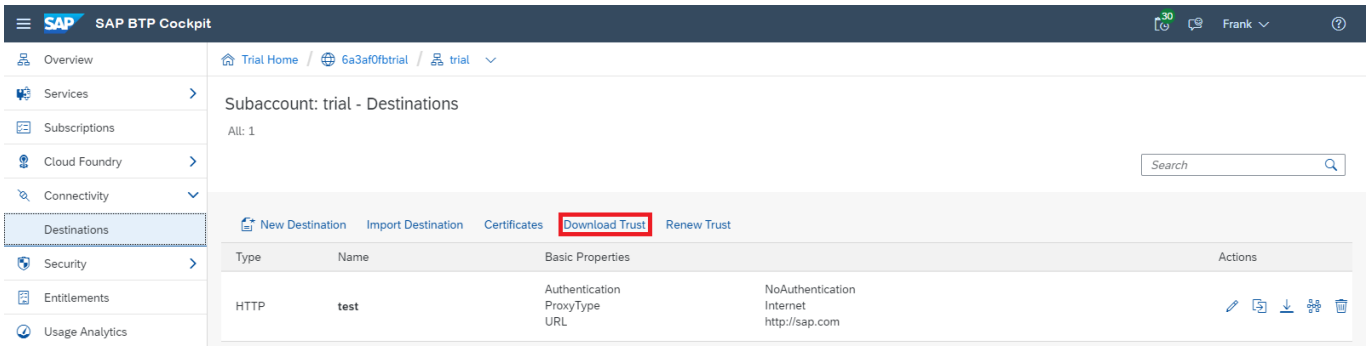
- Create an OAuth Client in SAP SuccessFactors
- Create and Consume a Destination for the Cloud Foundry Application

Create an OAuth Client in SAP SuccessFactors

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

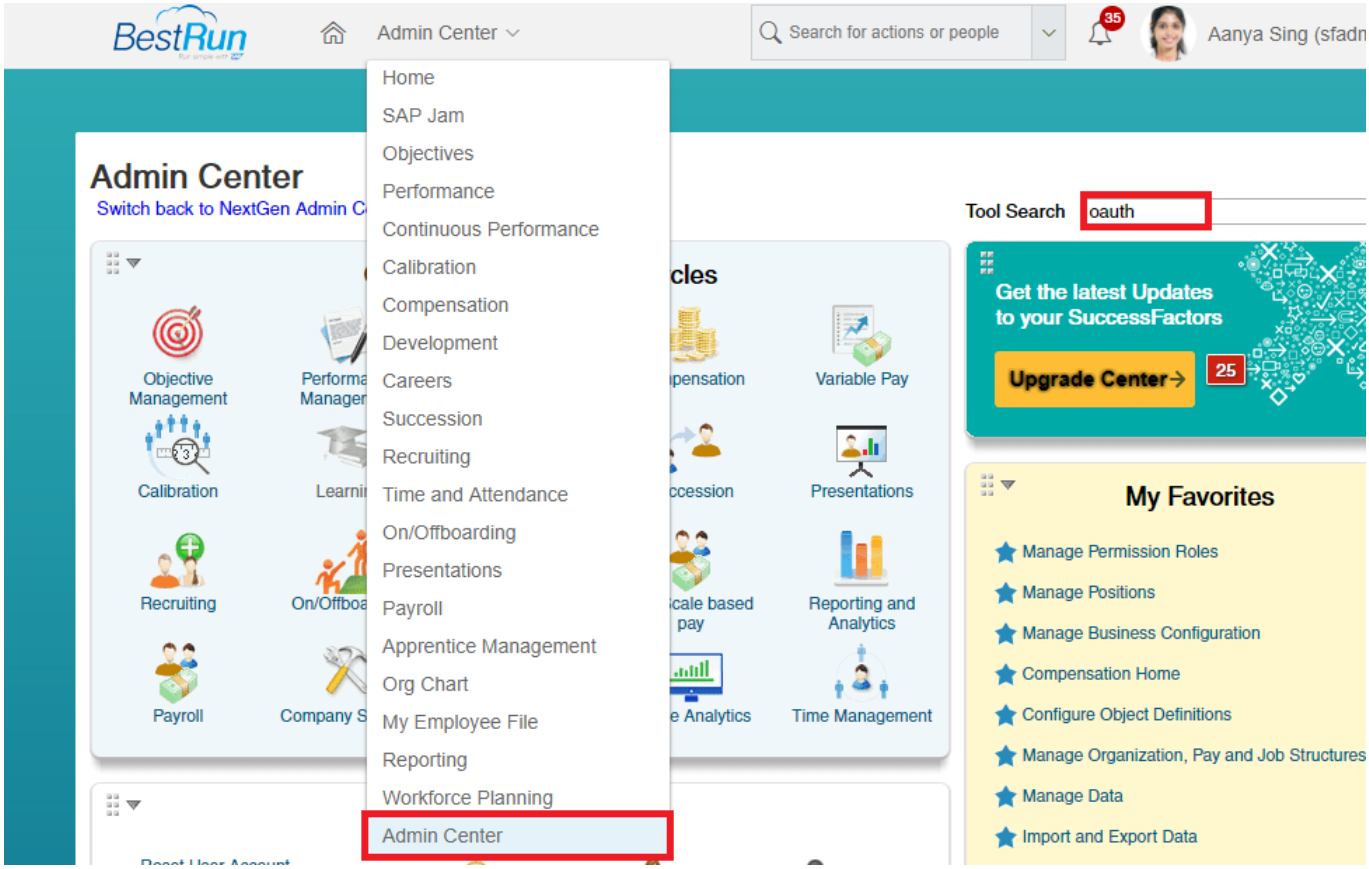
1. Download the X.509 certificate from your Cloud Foundry subaccount:

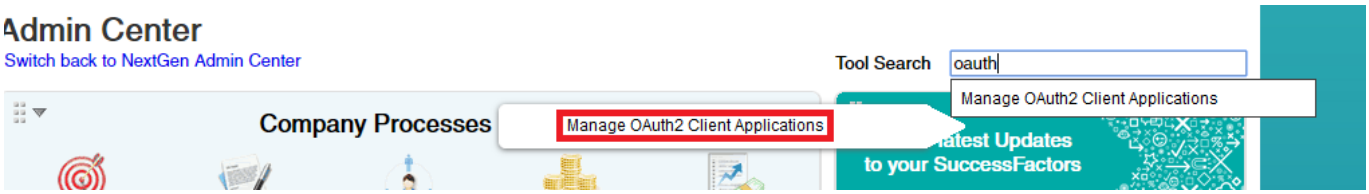
   In the cloud cockpit, navigate to your Cloud Foundry subaccount and from the left-side subaccount menu, choose **Connectivity** > **Destinations**. Choose **Download Trust** to get the certificate for this subaccount.



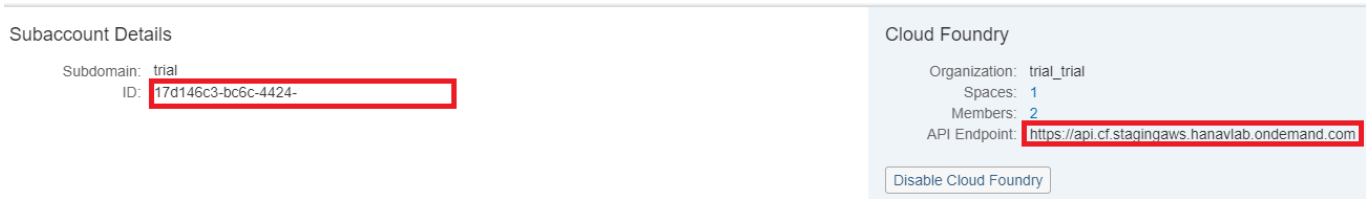
2. Create a SuccessFactors OAuth Client:

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





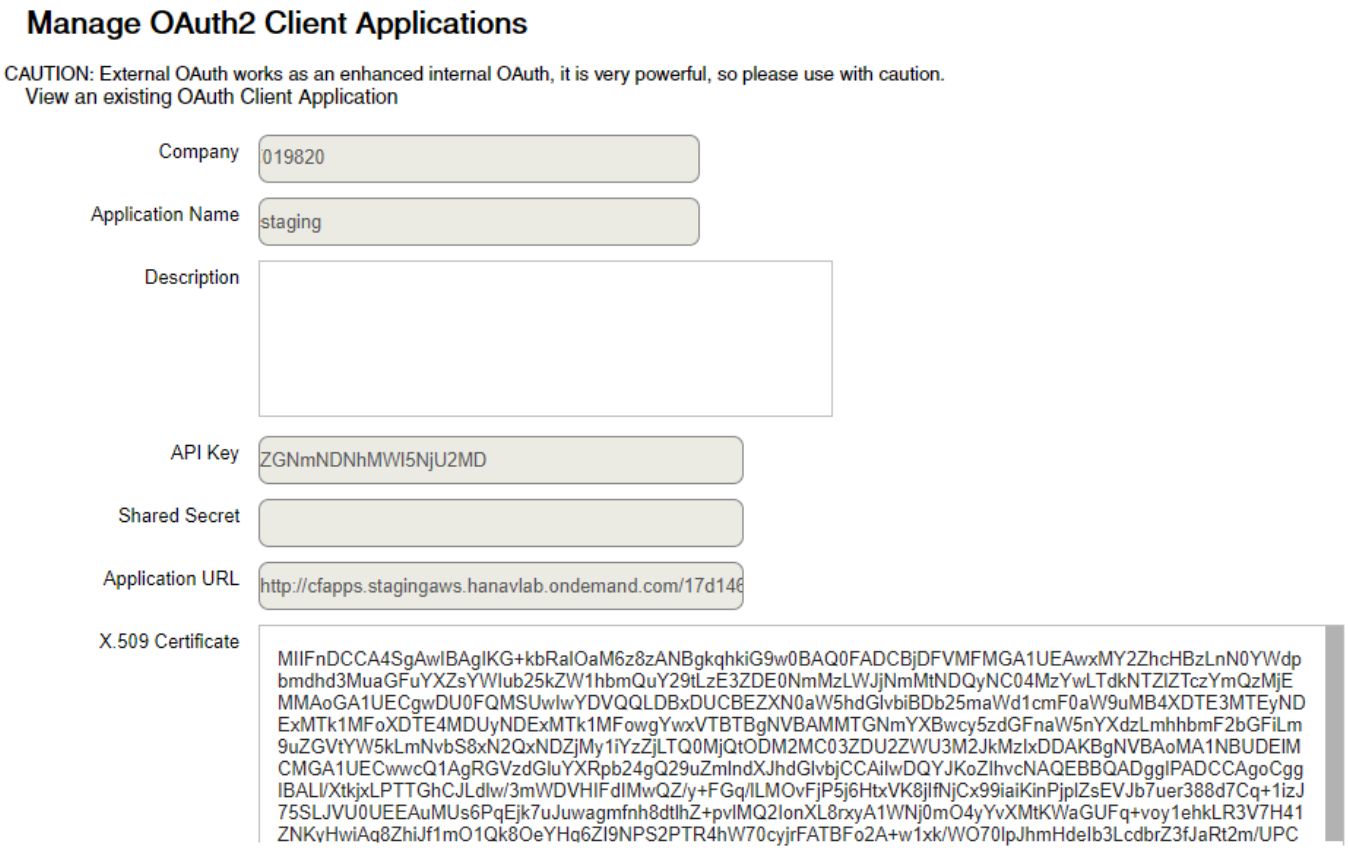
3. Press the **Register Client Application** button on the right. In the *<Application Name>* field, provide some arbitrary descriptive name for the client. For *<Application URL>*, enter the Cloud Foundry host of the application, followed by the subaccount GUID, for example `cfapps.stagingaws.hanavlab.ondemand.com/17d146c3-bc6c-4424-8360-7d56ee73bd32`.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



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.



Next Step

- Create and Consume a Destination for the Cloud Foundry Application

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 **Connectivity** > **Destinations**. 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.

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. 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 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 for details on the structure of the response from the Destination service.

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

Prerequisites

Concept

Procedure

1. Assemble IdP Metadata for Subaccount 1
2. Establish Trust between Subaccount 1 and Subaccount 2
3. Create an OAuthSAMLBearerAssertion Destination for Application 1

4. Consume the Destination and Execute the Scenario

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

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

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.

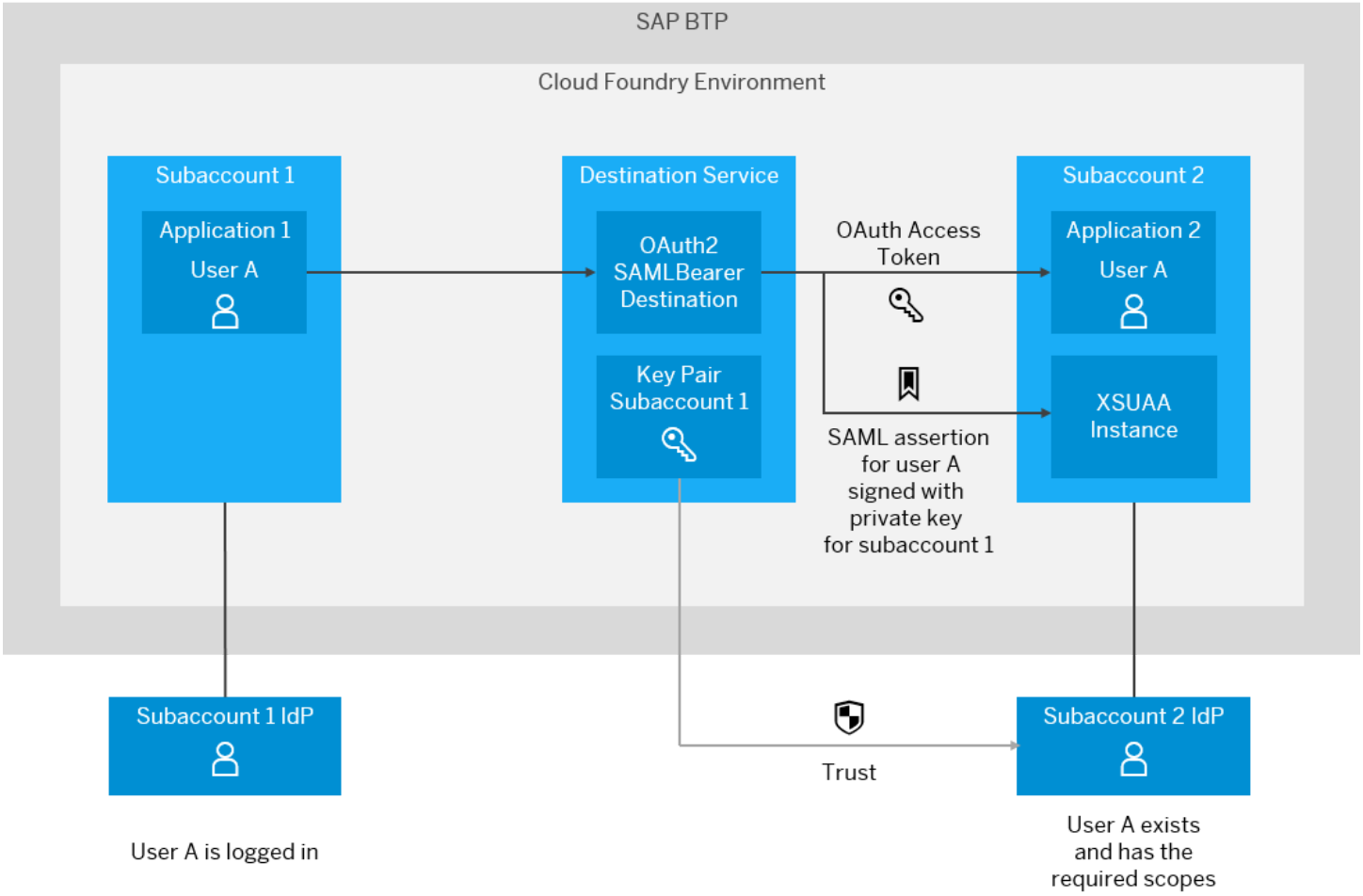
The service then wraps the user identity in a SAML assertion, signs it with subaccount 1's private key (which is part of the special key pair for the subaccount, maintained by the Destination service) and sends it to the authentication endpoint of **subaccount 2**, which hosts application 2.

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.

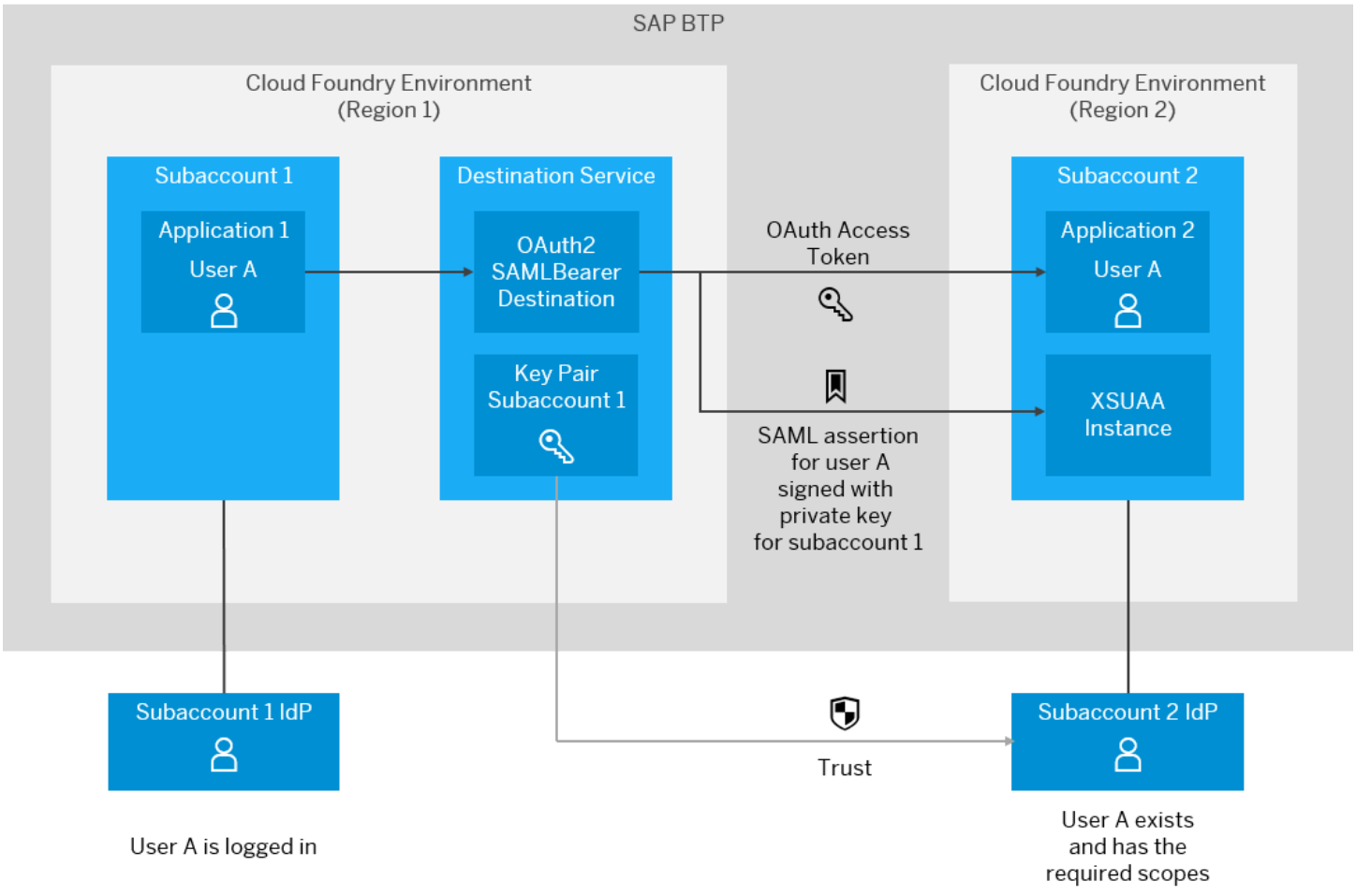
This way, users propagated from application 1 can be verified by subaccount 2's IdP before granting them access tokens with their respective scopes in the context of subaccount 2.

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

Procedure

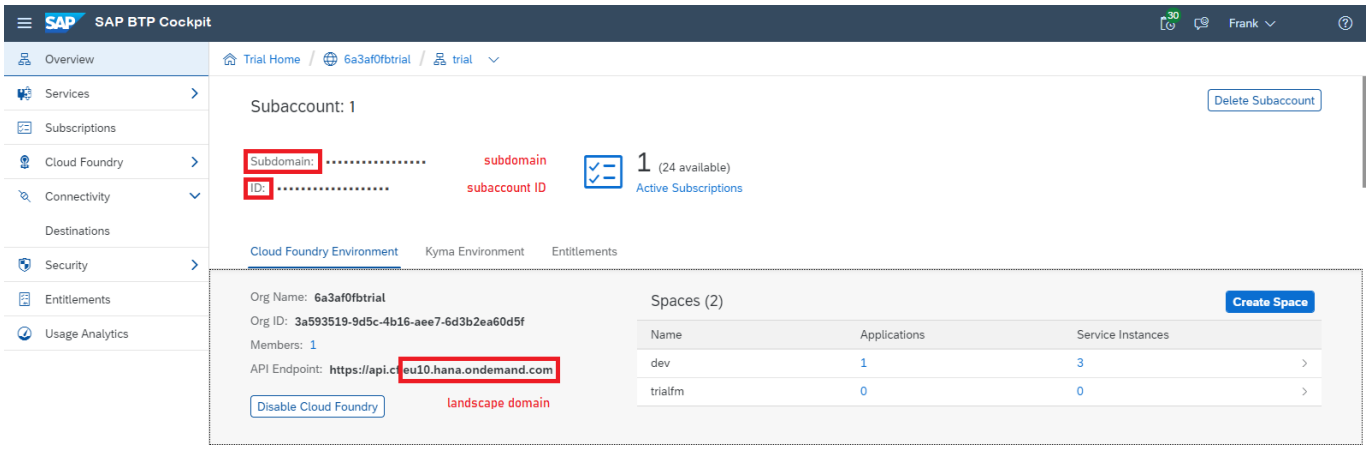
Assemble IdP Metadata for Subaccount 1

1. Download the X.509 certificate of subaccount 1. For instructions, see Set up Trust Between Systems. 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="h
   ...
   ```

   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

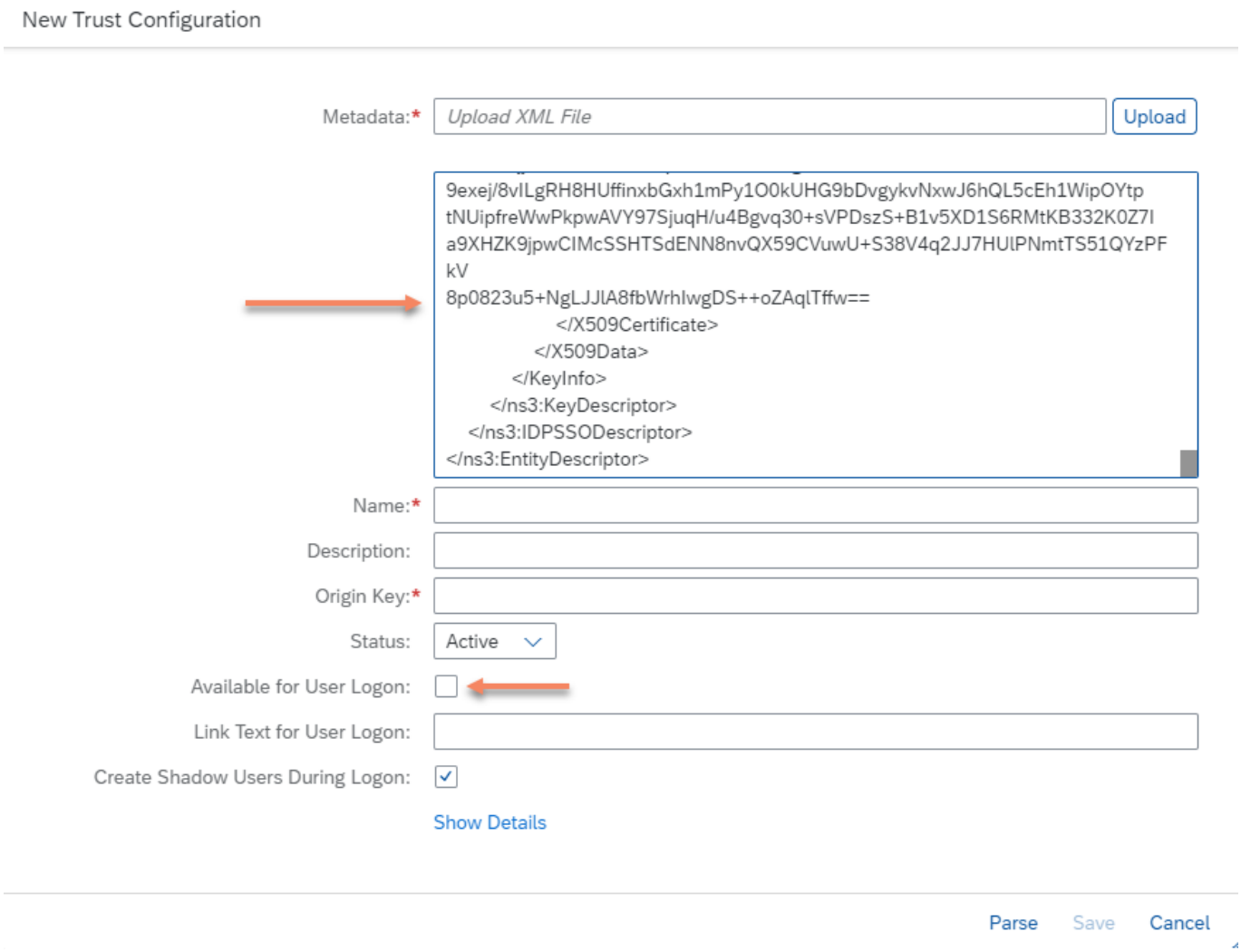
   ```
   <ns3:EntityDescriptor
       ID="cfapps.${S1_LANDSCAPE_DOMAIN}/${S1_SUBACCOUNT_ID}"
       entityID="cfapps.${S1_LANDSCAPE_DOMAIN}/${S1_SUBACCOUNT_ID}"
       xmlns="http://www.w3.org/2000/09/xmldsig#"
       xmlns:ns2="http://www.w3.org/2001/04/xmlenc#"
       xmlns:ns4="urn:oasis:names:tc:SAML:2.0:assertion"
       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

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 **Security** > **Trust Configuration**. 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**.



4. Choose **Parse**.

5. Enter a *<Name>* for the trust configuration and choose **Save**.

ℹ Note

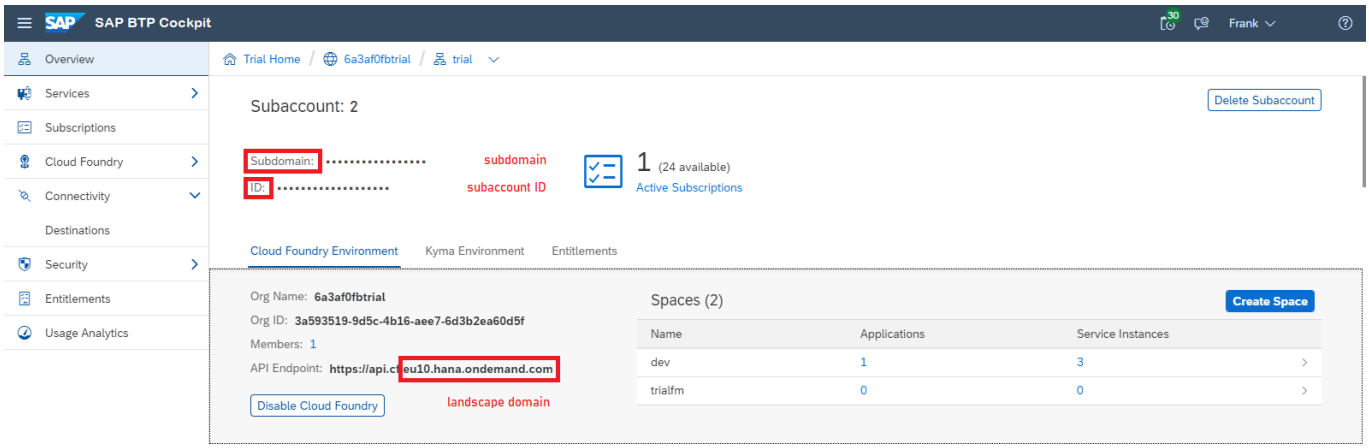
Additionally, you must add users to this new trust configuration and assign appropriate scopes to them.

Back to Steps

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}`.



3. 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" Locatio
...
```

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

4. In cockpit, navigate to subaccount 1.

5. From the left panel, select **Connectivity** > **Destinations**.

6. Choose **New Destination** and configure the values as described below. Replace the `${...}` placeholders with the values you determined in the previous steps and sections.

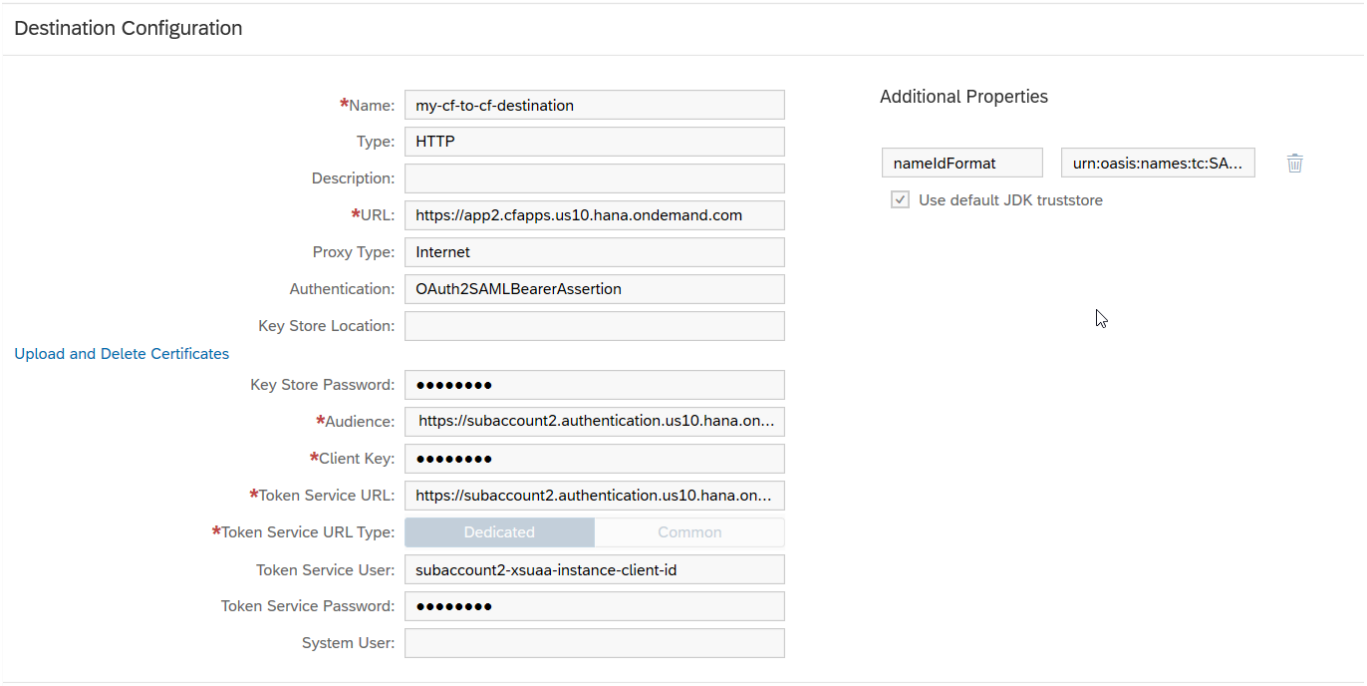
Property	Value
`Name`	Choose any name for your destination. You will use this name to request the destination from the Dest
`Type`	`HTTP`
`URL`	The URL of application 2, identifying the resource you want to consume.
`Proxy Type`	`Internet`
`Authentication`	`OAuth2SAMLBearerAssertion`
`Audience`	`${S2_AUDIENCE}`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

Property	Value
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



7. Choose **Save**.

Back to Steps

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. 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 and the REST API documentation.
3. From the Destination service response, extract the access token and URL, and construct your request to application 2. See "Find Destination" Response Structure for details on the structure of the response from the Destination service.

Back to Steps

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

- 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

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

Concept

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

ⅈ Note

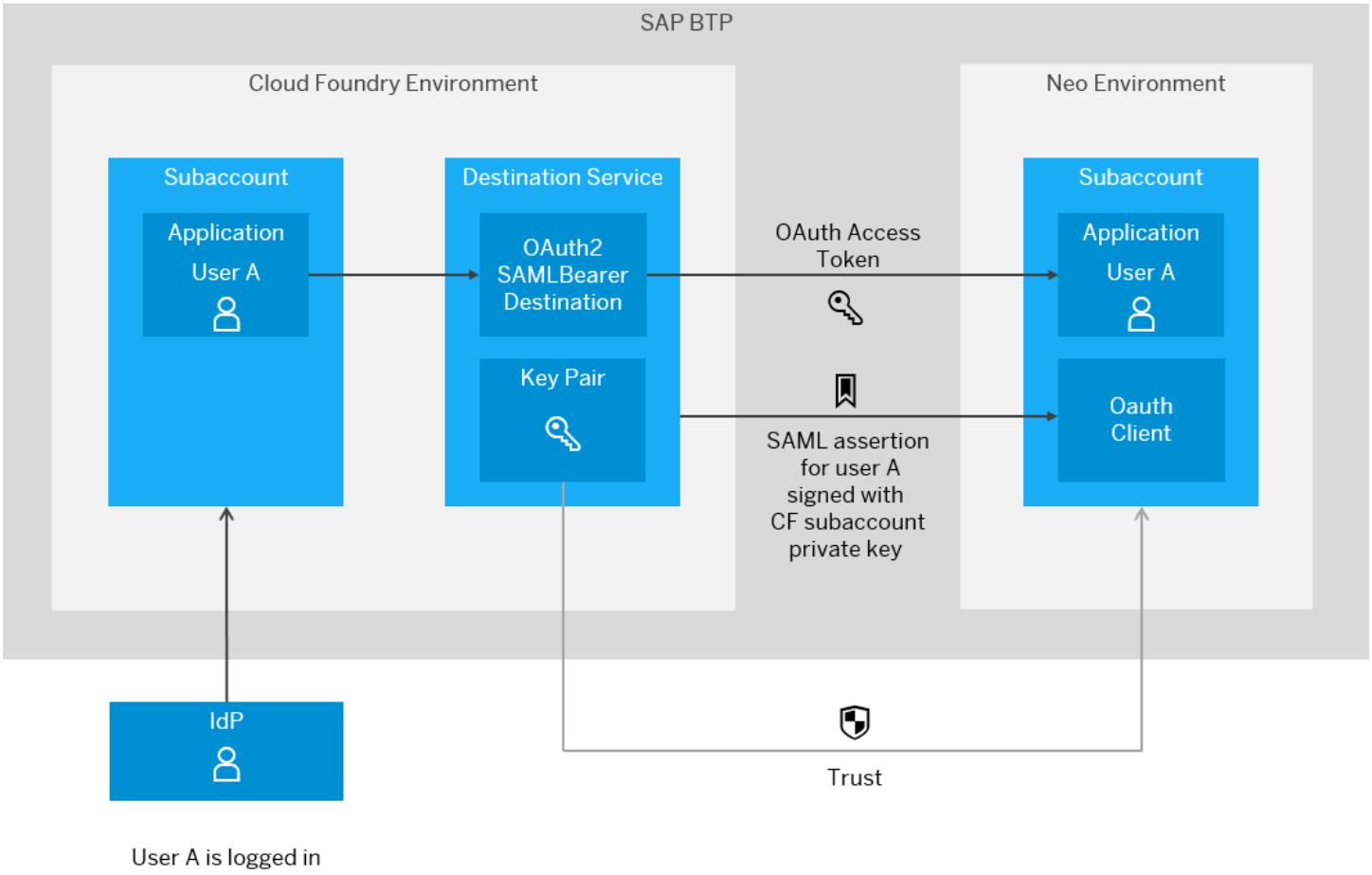
You can use the default IdP of the Cloud Foundry subaccount or any trusted IdP, for example, the Neo subaccount IdP.

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.

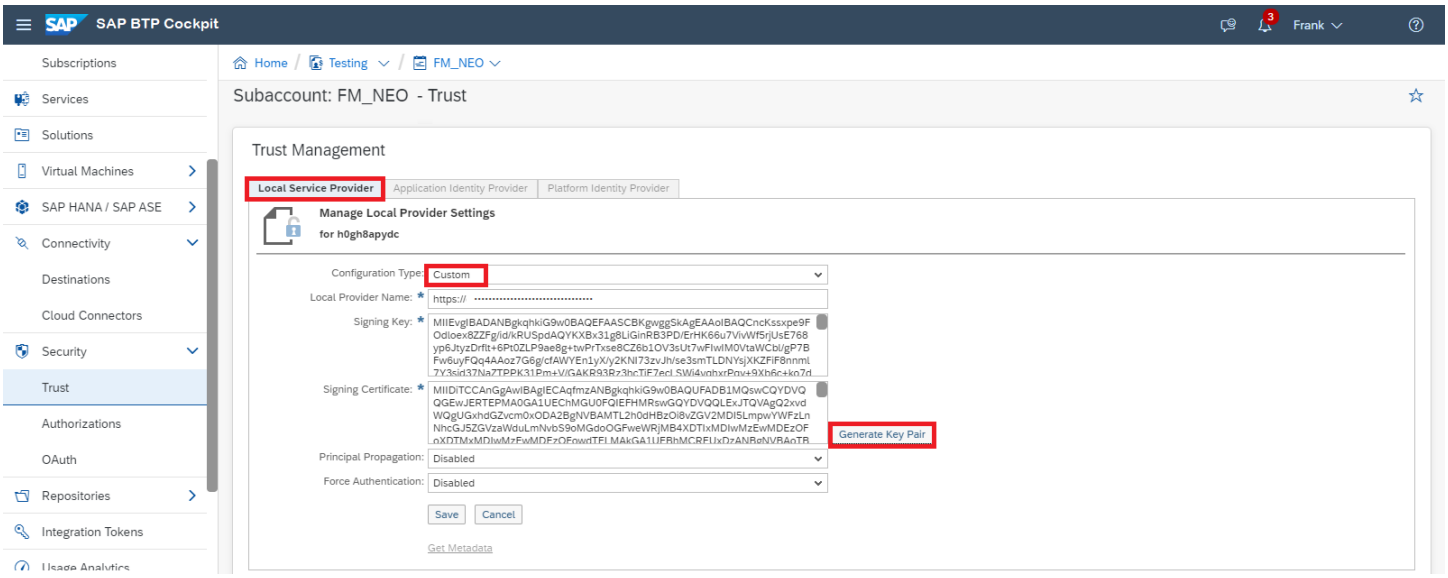


Back to Steps

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

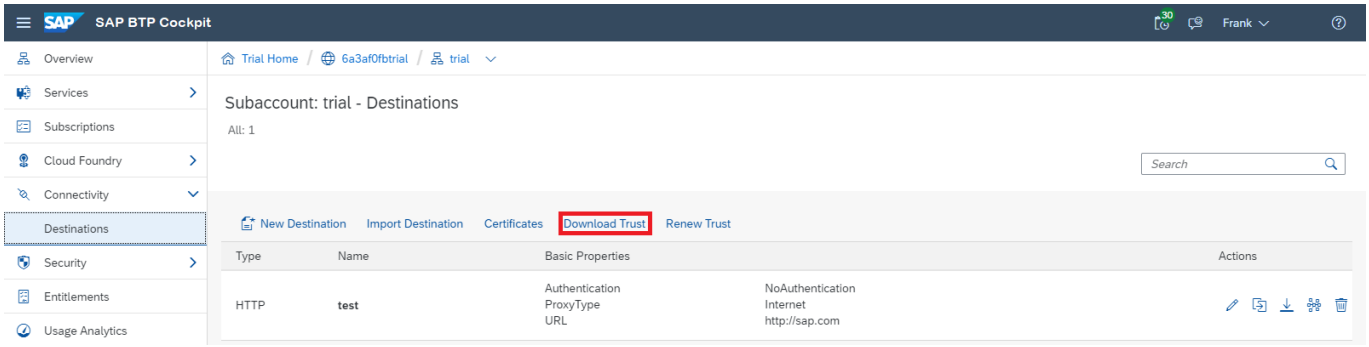
IMPORTANT: When you choose `Custom` for the *Local Service Provider* type, the default IdP (SAP ID service) will no longer be available. If your scenario requires login to the SAP ID service as well, you can safely skip this step and leave the default settings for the Local Service Provider.

Back to Steps

Establish Trust between Cloud Foundry and Neo Subaccounts

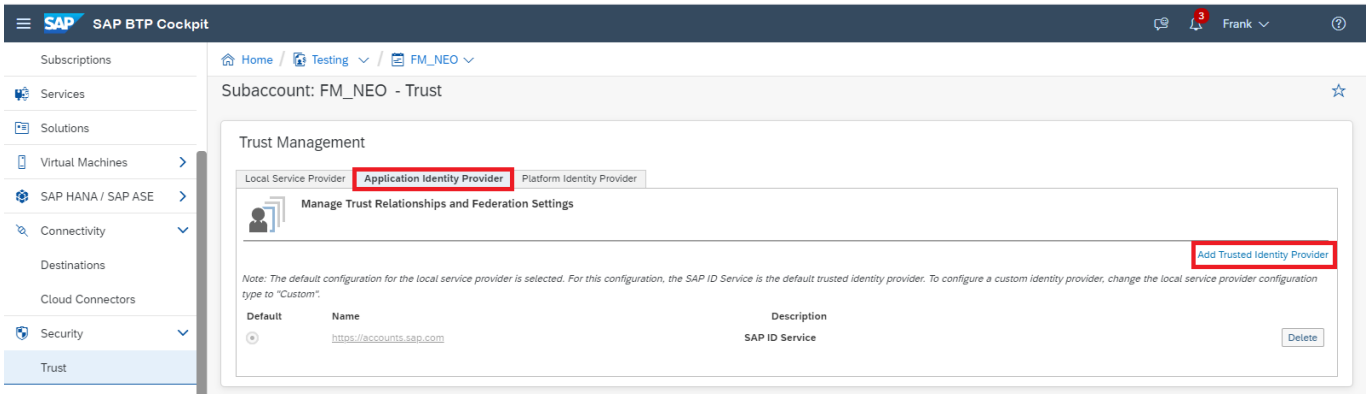
1. Download the X.509 certificate of the Cloud Foundry subaccount:

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

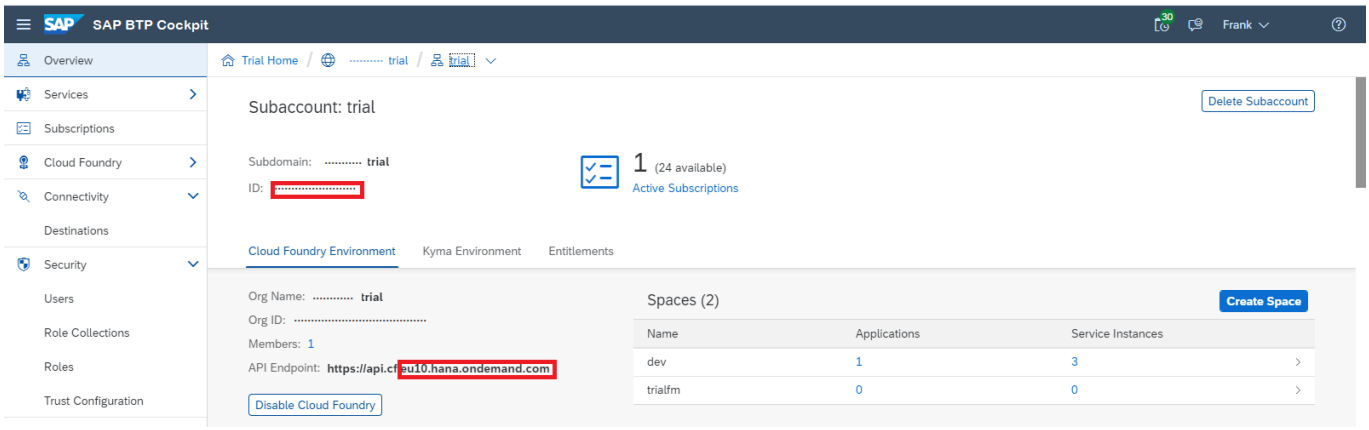


2. Configure trust in the Neo subaccount:

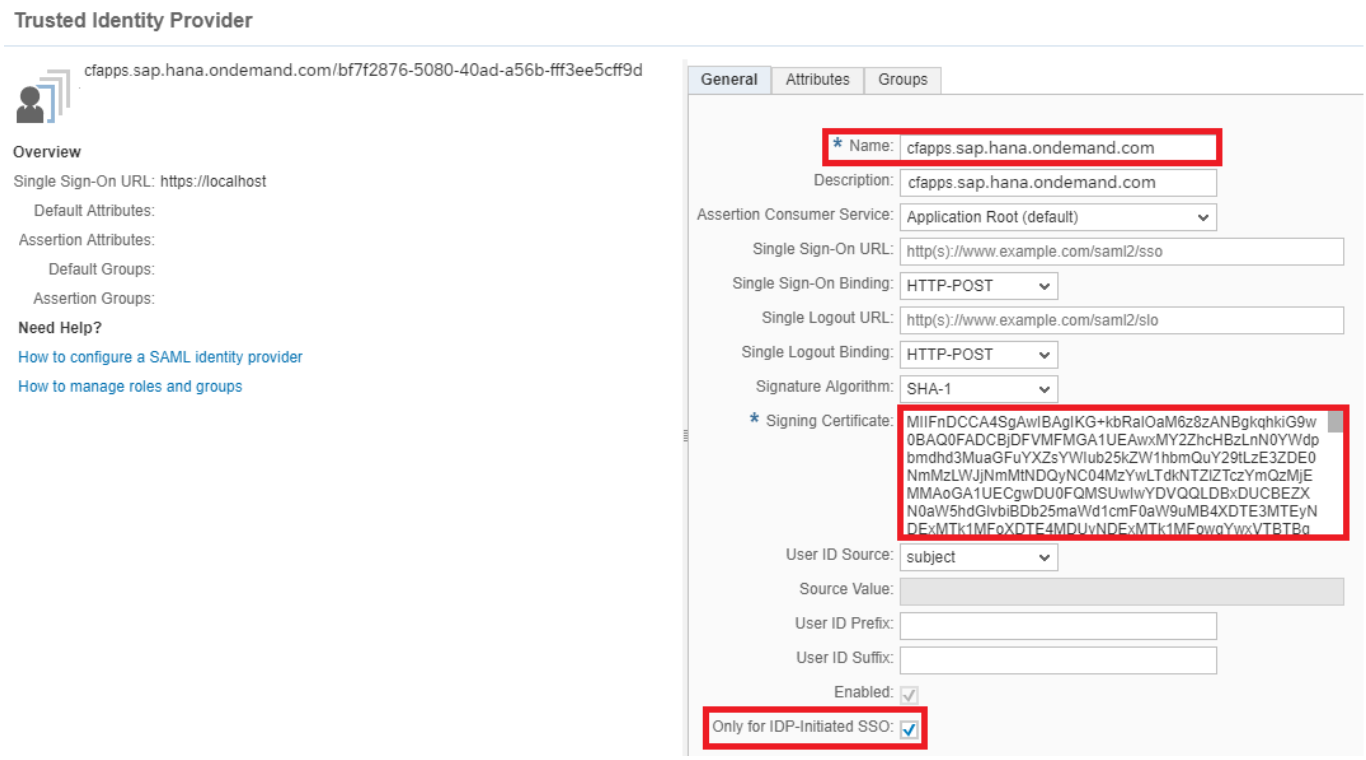
   In the cockpit, navigate to your Neo subaccount, choose **Trust** from the left menu, and select the **Application Identity Provider** tab on the right. Then choose **Add Trusted Identity Provider**.



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:



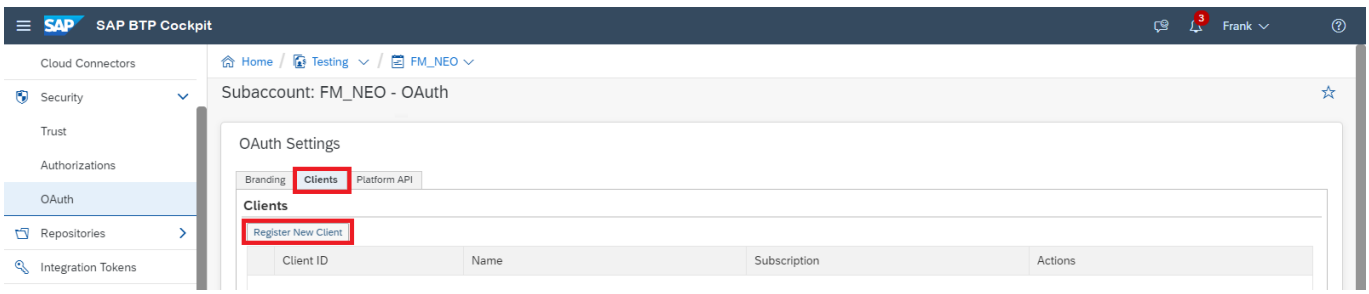
In the *<Signing Certificate>* field, paste the X.509 certificate you downloaded in step 1. Make sure you remove the *BEGIN CERTIFICATE* and *END CERTIFICATE* strings. Then check **Only for IDP-Initiated SSO** and save the configuration:



Back to Steps

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**:



2. Enter a *<Name>* for the client.

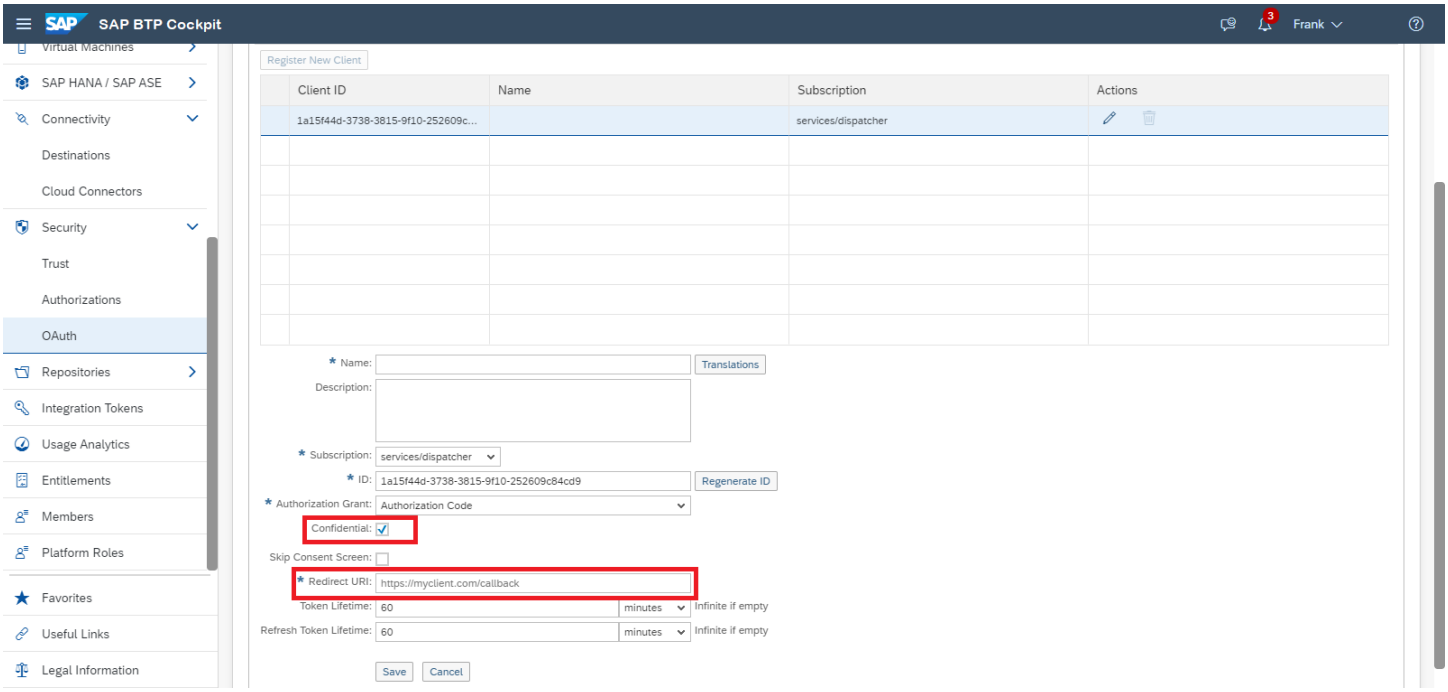
3. In the *<Subscription>* field, select your Neo application.

4. For *<Authorization Grant>* select `Authorization Code`.

5. Check the **Confidential** checkbox and provide a secret for the OAuth client.

> **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.



Back to Steps

Create an OAuth2SAMLBearerAssertion Destination for the Cloud Foundry Application

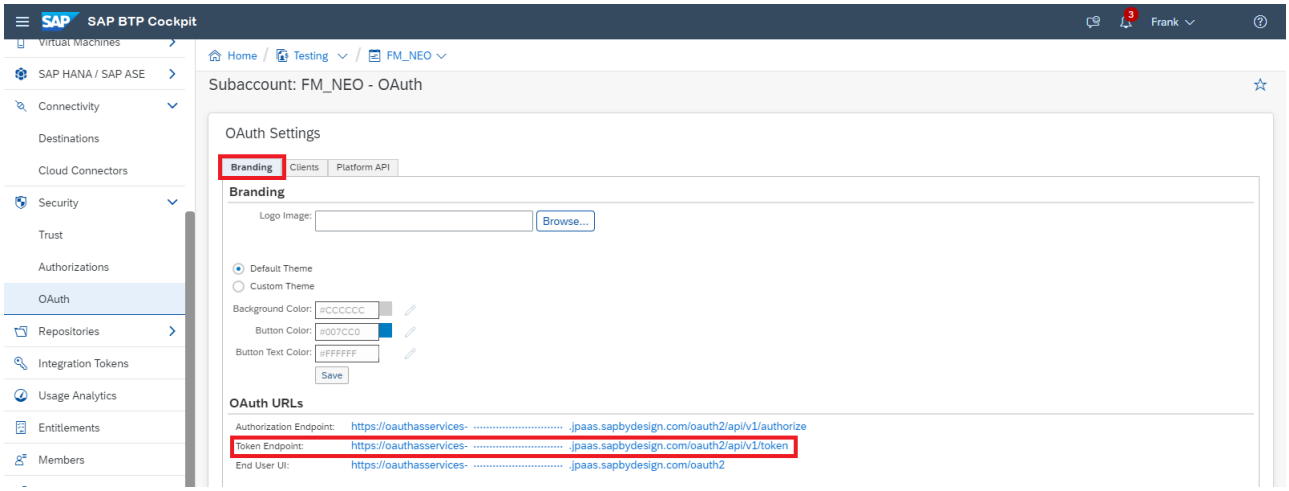
1. In the cockpit, navigate to the Cloud Foundry subaccount, choose **Connectivity** > **Destinations** from the left menu, select the **Client** tab and press **New Destination**.

2. Enter a *<Name>* for the destination, then provide:

   - *<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>*:

   - *<Client Key>*: the ID of the OAuth client for the Neo application

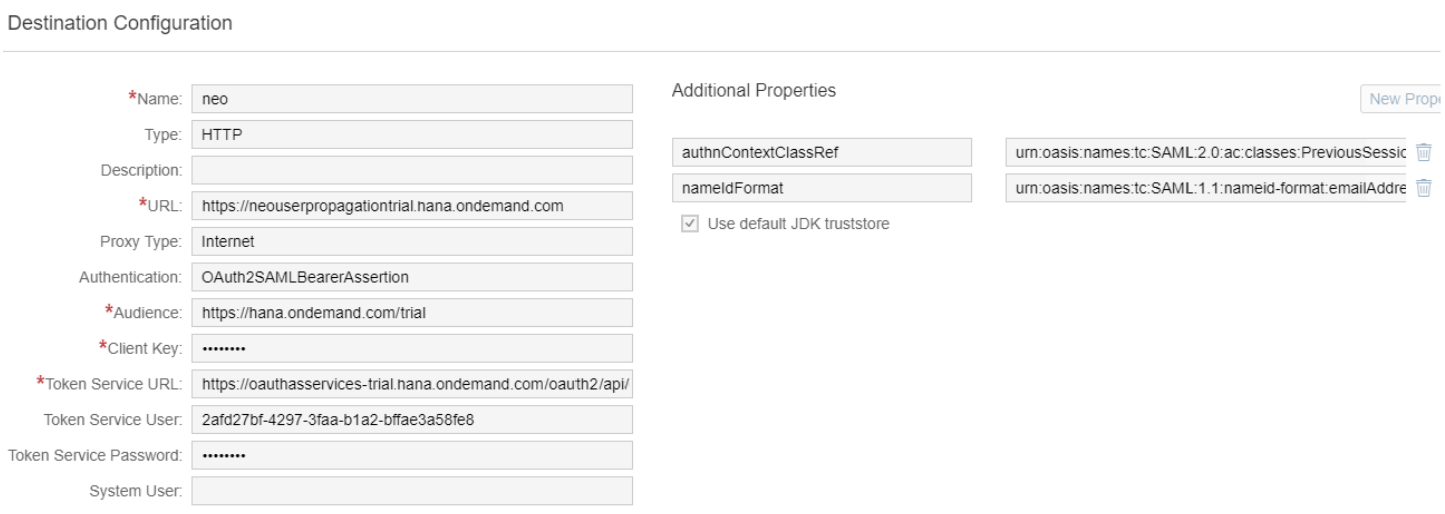
- *<Token Service URL>*: can be taken from the **Branding** tab in the Neo subaccount (choose **Security** > **OAuth** from the left menu):



- *<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.



Back to Steps

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. 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 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 for details on the structure of the response from the Destination service.

Back to Steps

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), or by each application subscriber (Subscriber-Specific Destination).

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.

Destination lookup according to the level, when configured on:

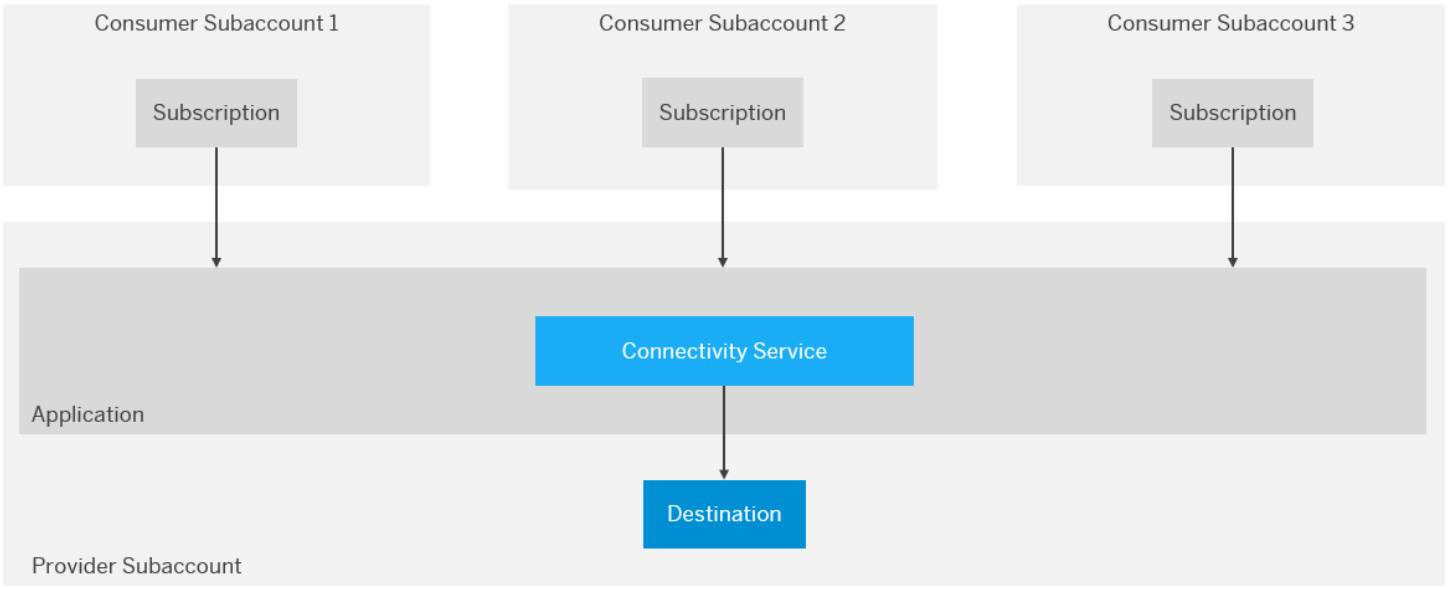
Level	Lookup
Subaccount level	Looked up on subaccount level, no matter which destination service instance is used.
Service instance level	Lookup via particular service instance (in provider or subscriber subaccount associated with this service instance).

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.

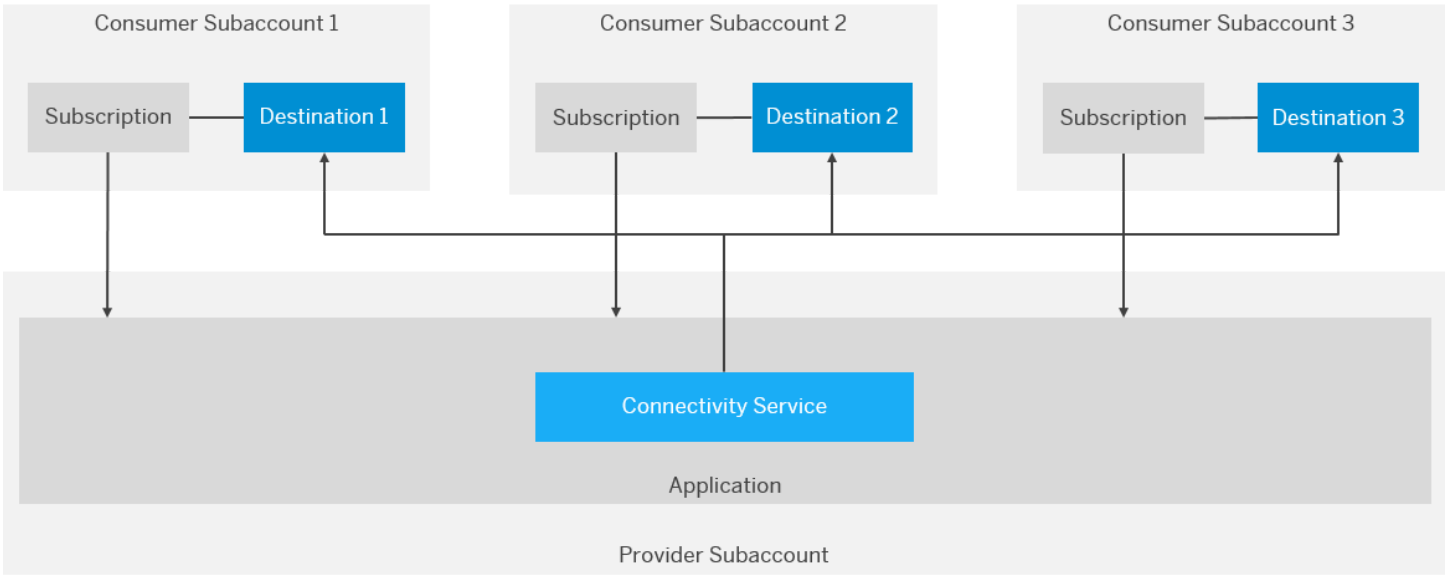
Back to Top

Provider-Specific Destination



Back to Top

Subscriber-Specific Destination



Back to Top

Related Information

Developing Multitenant Applications in the Cloud Foundry Environment

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
  - Example
  - Result
- Create and Bind a Service Instance from the Cockpit
  - Result

For more information on using X.509 certificates, see:

X.509 Bindings

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> [-c <config json>]*

Back to Procedure

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

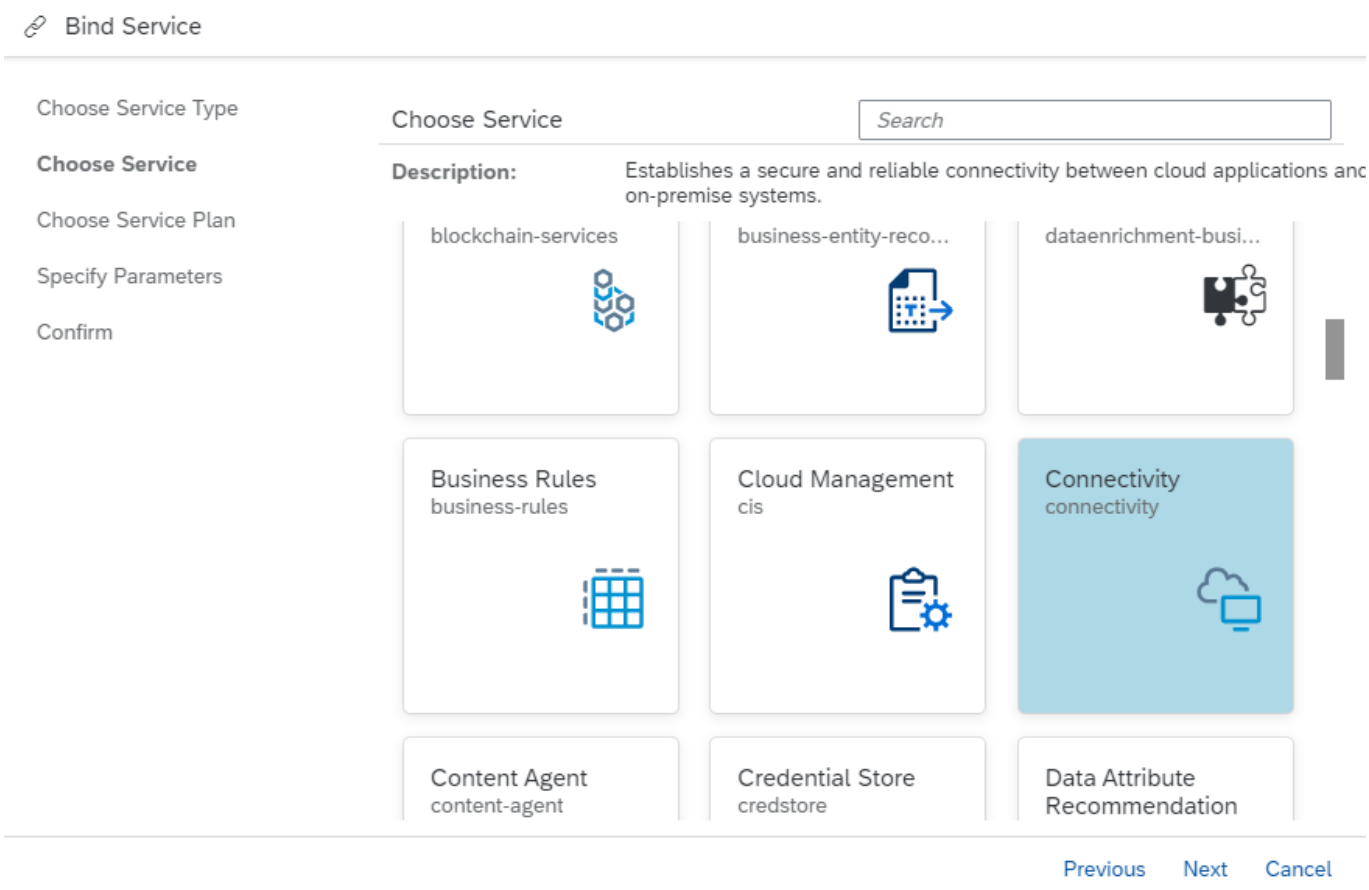
Back to Procedure

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

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": []
    }
    ],
```

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

X.509 Bindings

Note

Older instances of the service do not support the X.509 credentials binding type and an attempt to create one will result in an error. To overcome this, you just need to update the service instance, so it picks up the X.509 support.

The service supports X.509 bindings as described in Retrieving Access Tokens with Mutual Transport Layer Security (mTLS). This lets you choose if your binding / service key will be with a client secret X.509 certificate generated by SAP or an X.509 certificate provided by you. To do so, add a config JSON when creating your binding:

```
cf bind-service <app-name> <service-name> [-c <config json>]
```

The structure of the config JSON should be like this:

```
{
    "xsuaa":
    {
        <X.509 properties>
    },
    <other config parameters>
}
```

For a list of the supported X.509 properties, see:

- Parameters for X.509 Certificates Managed by SAP Authorization and Trust Management Service

- Parameters for Self-Managed X.509 Certificates

> **Note**
>
> By default, without providing a config JSON, a binding with client secret will be created.

Back to Procedure

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 SAP BTP 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.

Troubleshooting

If you get this failure message:

```
Failed to create all provided configurations. Will delete all on instance level, for configurations
```

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.

X.509 Bindings

ℹ Note

Older instances of the service do not support the X.509 credentials binding type and an attempt to create one will result in an error. To overcome this, you just need to update the service instance, so it picks up the X.509 support.

The service supports X.509 bindings as described in Retrieving Access Tokens with Mutual Transport Layer Security (mTLS). This lets you choose if your binding / service key will be with a client secret X.509 certificate generated by SAP or an X.509 certificate provided by you. To do so, add a config JSON when creating your binding:

```
cf bind-service <app-name> <service-name> [-c <config json>]
```

The structure of the config JSON should be like this:

```
{
    "xsuaa":
    {
        <X.509 properties>
    },
    <other config parameters>
}
```

For a list of the supported X.509 properties, see:

- Parameters for X.509 Certificates Managed by SAP Authorization and Trust Management Service
- Parameters for Self-Managed X.509 Certificates

ℹ Note

By default, without providing a config JSON, a binding with client secret will be created.

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
`init_data`	JSON	The data (destinations, certificates) to initialise or update the service instance with. The data can be stored on both *service instance* data and *subaccount* data.

Parameter	Value	Description
`HTML5Runtime_enabled`	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": "update|fail|ignore",
            "destinations" : [
                {
                    ...
                }
            ],
            "certificates" : [
                {
                    ...
                }
            ]
        },
        "instance" : {
            "existing_destinations_policy": "update|fail|ignore",
            "existing_certificates_policy": "update|fail|ignore",
            "destinations" : [
                {
                    ...
                }
            ],
            "certificates" : [
                {
                    ...
                }
            ]
        }
    }
}
```

Related Information

HTTP Destinations

Configuring Backup

To create a backup of your destination configurations, choose the procedure Export Destinations.

Notifications and Alerts

Subscribe to the SAP Alert Notification service to receive notifications and alerts for the Destination service on SAP BTP, Cloud Foundry environment.

You can subscribe for notifications and alerts for different Destination service events via SAP Alert Notification Service for SAP BTP.

For more information, see What Is SAP Alert Notification Service for SAP BTP?.

For details on Destinaton service events, see SAP BTP Destination Service Events.

Developing Applications

Consume the Connectivity service and the Destination service from an application in the Cloud Foundry environment.

Task	Description
Consuming the Connectivity Service	Connect your Cloud Foundry application to an on-premise system.
Consuming the Destination Service	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	Call a remote-enabled function module (RFM) in an on-premise or cloud ABAP server from your Cloud Foundry application, using the RFC protocol.

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
- Using the TCP Protocol for Cloud Applications

Tasks

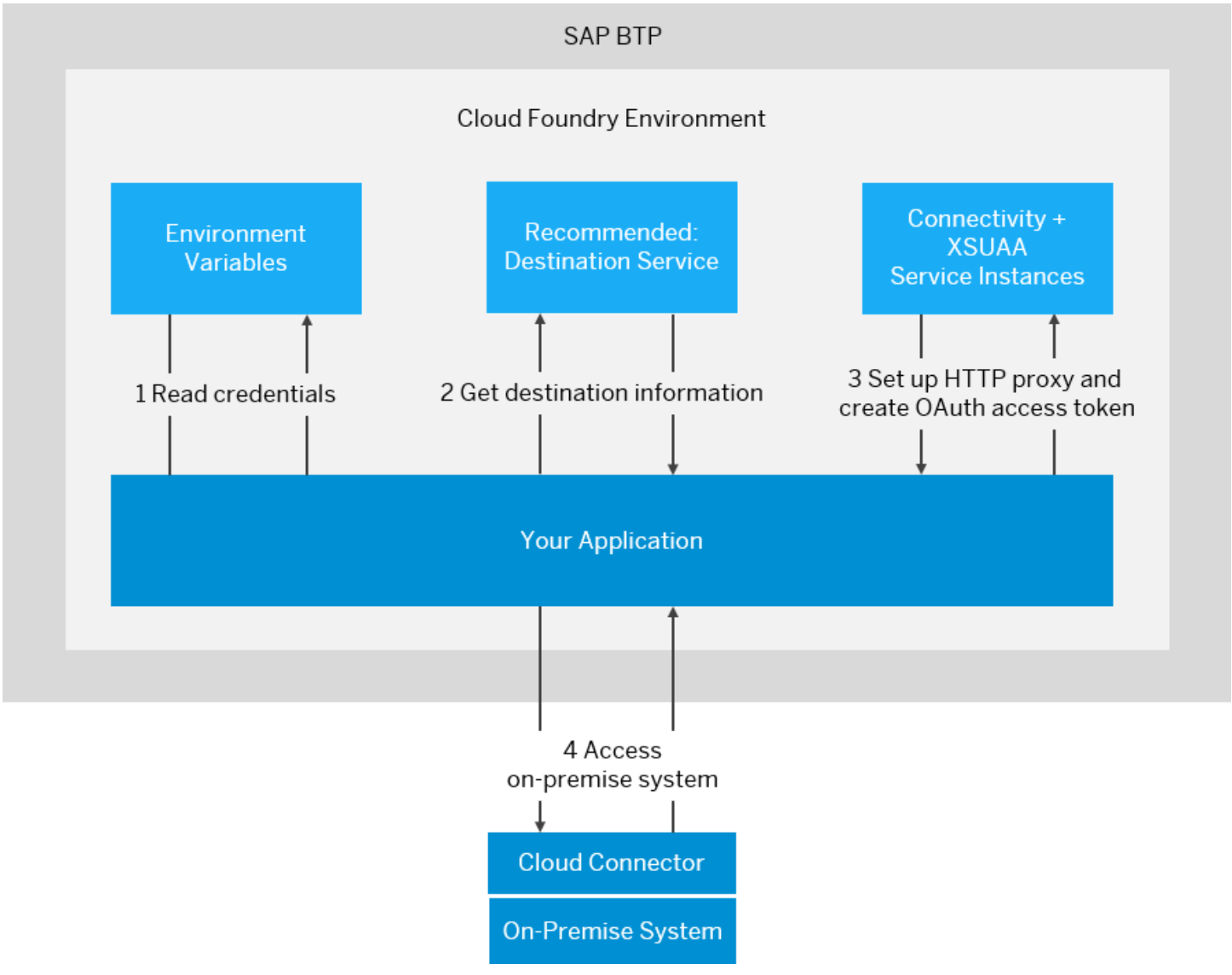
Task Type	Task
	Overview
	Prerequisites

Operator and/or Developer

Task Type	Basic Steps
Developer	1. Read Credentials from the Environment Variables 2. Provide the Destination Information 3. Set up the HTTP Proxy for On-Premise Connectivity
Operator and/or Developer	Additional Steps • Authentication against the On-Premise System • Specify a Cloud Connector Location ID • Multitenancy in the Connectivity Service

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

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].

  > **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 to the scenario you want to use. See Installation and Configuration.
- You have deployed an application in a landscape of the Cloud Foundry environment that complies with the Business Application Pattern.
- 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.
- 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.

> **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.

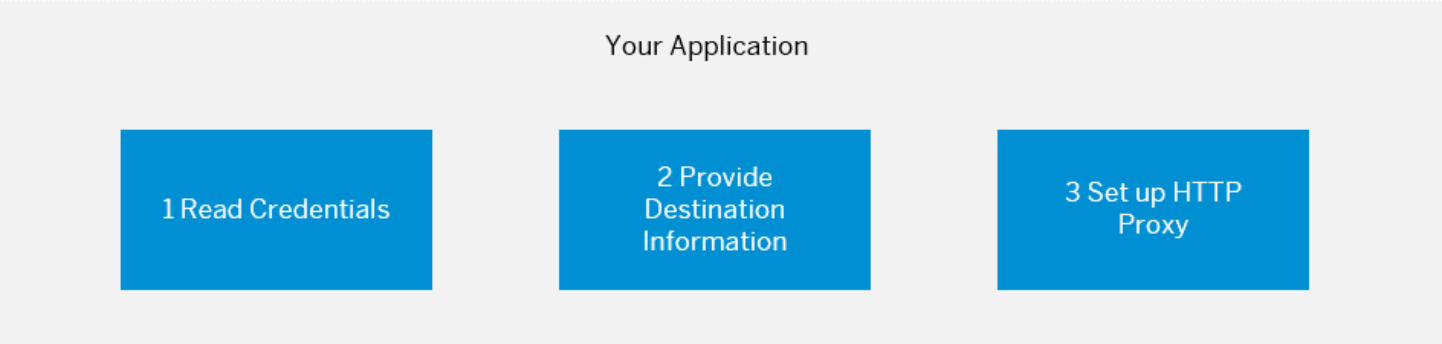
> **Caution**
>
> There is a limit of **8192** bytes for the size of the HTTP lines (for example, request line or header) that you send via the Connectivity service. If this limit is exceeded, you receive an HTTP error of type 4xx. This issue is usually caused by the size of the *path* + *query* string of the request.

Back to Tasks

Basic Steps

To consume the Connectivity service from your Cloud Foundry application, perform the following basic steps:

1. Read Credentials from the Environment Variables
2. Provide the Destination Information
3. Set up the HTTP Proxy for On-Premise Connectivity



Back to Tasks

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");
```

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.

Note

We refer to the result of the above code block as `connectivityCredentials`, when called for ***connectivity***, and `xsuaaCredentials` for ***xsuaa***.

Back to Tasks

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).
- The required authentication type for the on-premise system. See HTTP Destinations.

- 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.
- (Optional) You can use a ***location Id***. For more details, see section Specify a Cloud Connector Location ID.

We recommend that you use the ***Destination service*** (see Consuming the Destination Service) 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

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 varia
// 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 Require
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 la
urlConnection = (HttpURLConnection) url.openConnection(proxy);
```

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 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.ClientCredentialsOAuth2AuthorizedClientProvide
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(clie
    principal(new AbstractAuthenticationToken(null) {
        @Override
        public Object getPrincipal() {
            return null;
        }

        @Override
        public Object getCredentials() {
            return null;
        }

        @Override
        public String getName() {
            return "dummyPrincipalName";    // There is no principal in the client credentials a
        }
    }).build();

OAuth2AuthorizedClientProvider clientCredentialsAccessTokenProvider = new ClientCredentialsOAuth2
OAuth2AccessToken token = clientCredentialsAccessTokenProvider.authorize(xsuaaContext).getAccess

// set access token as Proxy-Authorization header in the URL connection
urlConnection.setRequestProperty("Proxy-Authorization",  token.getTokenType().getValue() + " " +
```

Note

`xsuaaCredentials.getString("token_service_url")` replaces the deprecated property `xsuaaCredentials.getString("url")`, which will be removed soon.

Back to Tasks

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.

Back to Tasks

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, 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

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

Related Information

Cloud Connector
Set Up an Application as a Sample Backend System
Create and Bind a Connectivity Service Instance
Authentication to the On-Premise System
Consuming the Destination Service
What Is the SAP Authorization and Trust Management Service?
Multitarget Applications in the Cloud Foundry Environment
Invoking ABAP Function Modules via RFC
Using the TCP Protocol for Cloud Applications

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
- Authentication Types

The SAP-Connectivity-Authentication Header

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.

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

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.

ⓘ Note

Do not use the `Authorization` header in *principal propagation* scenarios.

Technical User Propagation

As of Cloud Connector version 2.15, consumers of the Connectivity service can propagate technical users from the cloud application towards the on-premise systems. To make use of this feature, provide a JWT (usually obtained via `client_credentials` OAuth flow), representing the technical user, via the `SAP-Connectivity-Technical-Authentication` header. This is similar to *principal propagation*, but in this case, a technical user is propagated instead of a business user.

```
urlConnection.setRequestProperty("SAP-Connectivity-Technical-Authentication", "Bearer " + technical
```

For more information, see Configuring Principal Propagation.

ⓘ Note

Do not use the `Authorization` header in *technical user propagation* scenarios.

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

Related Information

Configure Principal Propagation via Corporate IdP Embedded Token
Configure Principal Propagation via User Exchange Token
Configure Principal Propagation via IAS Token
Configure Principal Propagation via OIDC Token

Configure Principal Propagation via Corporate IdP Embedded Token

Configure a corporate IdP embedded token for principal propagation (user propagation) from your Cloud Foundry application to an on-premise system.

Tasks

Task Type	Task
 Operator and/or Developer	Scenario
 Operator	Prerequisites
 Developer	Solutions

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 via token from your trusted corporate IdP.

For more information, see Principal Propagation.

Back to Tasks

Prerequisites

- Cloud Connector 2.13 (or newer) must be used.

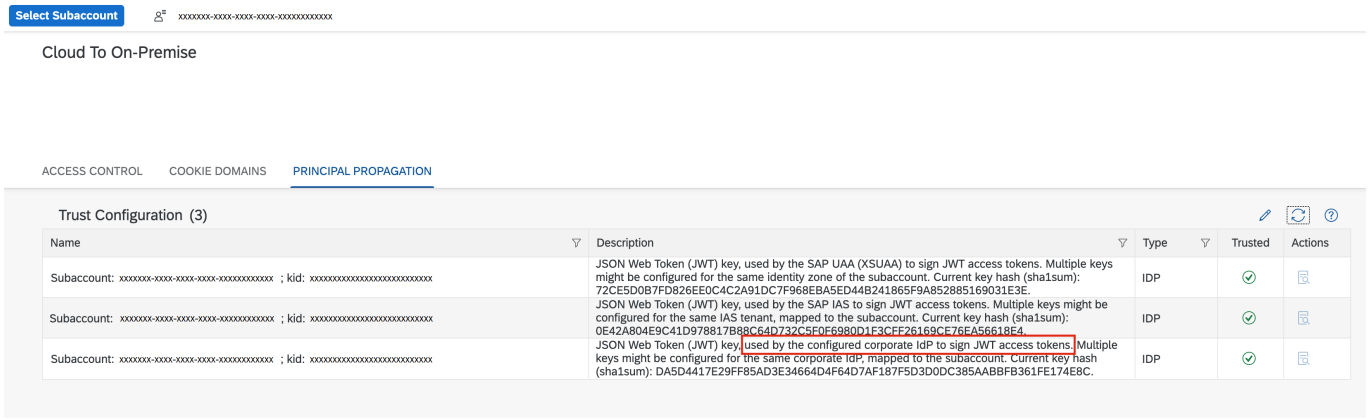
- The Cloud Connector must be connected to a subaccount that is configured with the corporate IdP, issuing the tokens.

  After the configuration, the issued XSUAA token contains an embedded token, which is extracted and propagated by the Connectivity service.

  For more information, see SAP Authorization and Trust Management Service in the Cloud Foundry Environment.

- Trust configuration for that subaccount must be synchronized in the Cloud Connector to obtain the JSON web key set from the configured corporate IdP that is used to verify the token.

  For more information, see Set Up Trust for Principal Propagation.



Back to Tasks

Solutions

You have two options to implement the user propagation via embedded corporate IdP token:

1. **Recommended**. The application sends one header containing the user exchange token to the Connectivity proxy:
   - The application sends one HTTP header `Proxy-Authorization`.
   - The original *user token* is exchanged for a special *user exchange access token* following the OAuth2 JWT Bearer grant type.

     For more information, see Using JWTs as Authorization Grants .
   - The *user exchange access token* is send through the `Proxy-Authorization` header to consume the Connectivity service.
   - The embedded corporate token which contains the user details must be present in the obtained *user exchange access token*.

   ```
   Header: "Proxy-Authorization" : "Bearer <userExchangeAccessToken>"
   ```

   For more information on how to obtain a user exchange access token with an embedded IAS corporate IdP token, see SAP Authorization and Trust Management Service in the Cloud Foundry Environment and Generate the Authentication Token.

2. The application sends two headers to the Connectivity proxy:
   - The application sends two HTTP headers: `SAP-Connectivity-Authentication` and `Proxy-Authorization`.
   - The embedded corporate token which contains the user details must be present in the user token provided via `SAP-Connectivity-Authentication`.
   - The access token is provided via `Proxy-Authorization`.

```
Header: "SAP-Connectivity-Authentication" : "Bearer <userToken>"
Header: "Proxy-Authorization" : "Bearer <accessToken>"
```

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.

Back to Tasks

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
 Operator and/or Developer	Scenario
 Developer	Solutions
	Generate the Authentication Token

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.

Back to Tasks

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 <userExchangeAcessToken>"
   ```

   This option is described in detail in Generate the Authentication Token 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.

ℹ Note

This solution is supported to guarantee backward compatibility.

Back to Tasks

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
- Example: Calling the Connectivity Proxy with the Exchanged User Token

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>/oa
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 `userExchangeAcessToken` 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 XsuaaDefau
String userExchangeAcessToken = tokenFlows.userTokenFlow().token(<jwtToken_to_exchange>).execute
```

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

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.

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);
```

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

Back to Tasks

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 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

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
 Operator and/or Developer	Scenario
 Developer	Solution

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 and the OIDC specification .

Back to Tasks

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": "11122233344455566677788899900",
  "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 Subject Patterns for Principal Propagation.

Back to Tasks

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

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.

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, 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 (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

Restrictions

- You cannot use the provided SOCKS5 proxy as general-purpose proxy.
- Proxying UDP traffic is not supported.

Back to Content

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

    executeSOCKS5AuthenticationRequest(outputStream); // 2. Negotiate authentication sub-version

    executeSOCKS5ConnectRequest(outputStream, (InetSocketAddress) endpoint); // 3. Initiate conn
}
```

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.getBytes().length
        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(sccLocat
    }

    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", 
```

```
        }
    }

    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, bu
        }

        int authenticationMethodValue = inputStream.read();
        if (SOCKS5_JWT_AUTHENTICATION_METHOD_UNSIGNED_VALUE != authenticationMethodValue) {
            throw new SocketException(String.format("Unsupported authentication method value - e
                    SOCKS5_JWT_AUTHENTICATION_METHOD_UNSIGNED_VALUE, authenticationMethodValue))
        }
    }
```

```
    private void executeSOCKS5AuthenticationRequest(OutputStream outputStream) throws IOExceptio
        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).arra
            byteArraysStream.write(jwtToken.getBytes());
            byteArraysStream.write(ByteBuffer.allocate(1).put((byte) sccLocationId.getBytes().le
            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 -
                    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 endpoi
        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)
                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, bu
        }

        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: " + commandConnectStatusTra
        }

        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 n
                        }
                        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 + portBy
            }
        }
    }
```

Back to Example

Back to Content

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:

SOCSK5 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).
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 and Managing Subaccounts, section *Procedure*.
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.
0x08	ADDRESS_UNSUPPORTED		Only the SOCKS5 `DOMAIN` and `IPv4` commands are supported.

Back to Content

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
	Prerequisites

Operator and/or Developer

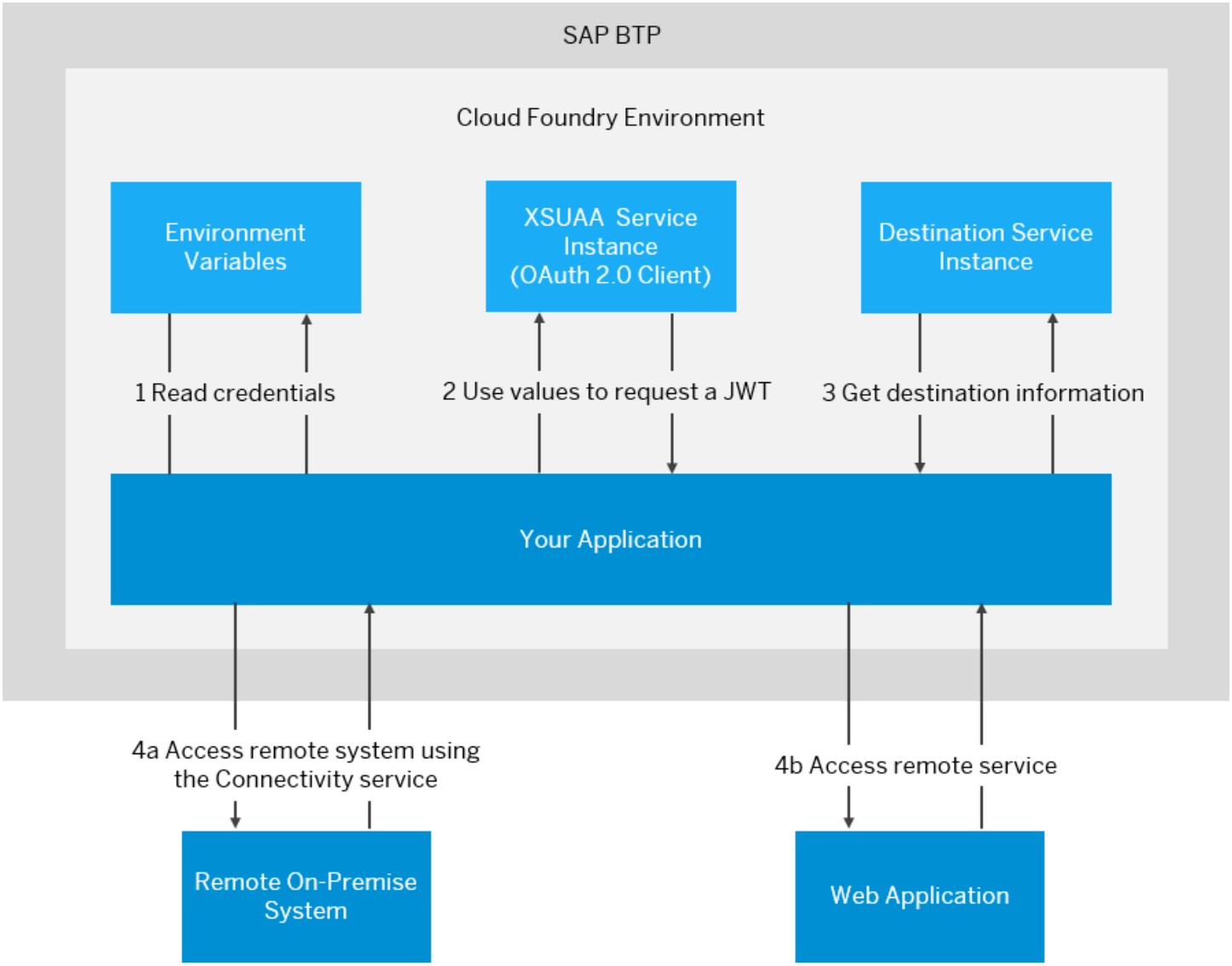
Task Type	Steps
 Developer	1. Read Credentials from the Environment Variables 2. Generate a JSON Web Token (JWT) 3. Call the Destination Service
 Operator and/or Developer	Destination Configuration Attributes

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.
- 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

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`.

  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.
- 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.

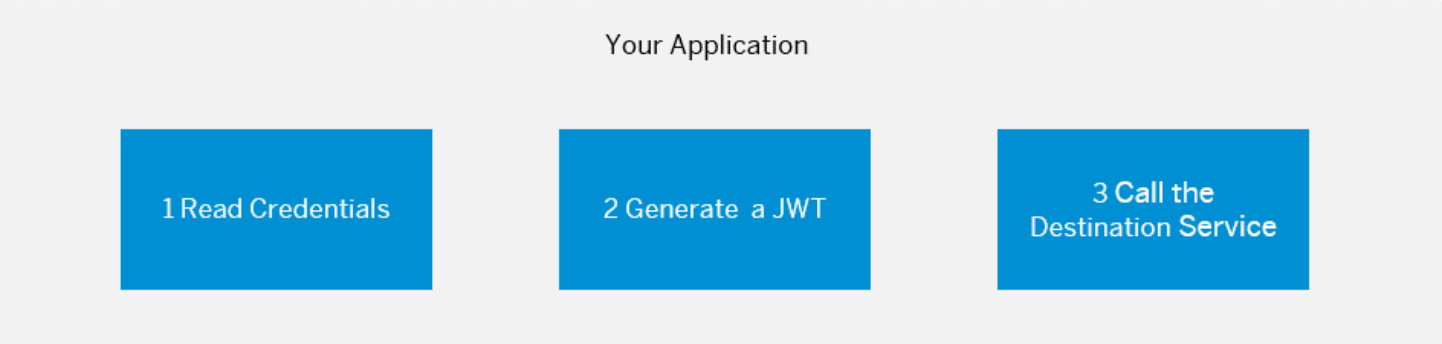
To manage destinations via REST API, see Destination Service REST API.

Back to Tasks

Steps

To consume the Destination service from your application, perform the following basic steps:

1. Read Credentials from the Environment Variables
2. Generate a JSON Web Token (JWT)
3. Call the Destination Service



Back to Tasks

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.

Note

Below, we refer to the ***JSONObjects***, containing the instance credentials as `destinationCredentials` (for the Destination service) and `xsuaaCredentials` (for xsuaa).

Back to Tasks

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 XsuaaDefau

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

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")

Read a Destination Associated with a Subaccount

Get All Destinations Associated with a Subaccount

Response Codes

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.

Back to Call the Destination Service

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

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": "myname123",
"Password": "123456"
}
]
```

Back to Call the Destination Service

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

Back to Tasks

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.

Back to Tasks

Related Information

User Propagation via SAML 2.0 Bearer Assertion Flow
Destination Service REST API
Exchanging User JWTs via OAuth2UserTokenExchange Destinations
Use Cases
Multitenancy in the Destination Service
Destination Java APIs

"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.
- The actual destination configuration.
- (Optional) Authentication tokens that are relevant to the destination.
- (Optional) Certificates 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.

See also Call the Destination Service.

Destination Owner

- **Key:** `owner`

  The JSON object that serves as the value of this property contains two properties itself: `SubaccountId` and `InstanceId`. `SubaccountId` has as its value the ID of the subaccount instance to which the destination belongs. Depending on where the destination was found (as a subaccount destination or a service instance destination) `InstanceId` can have the value `null`, or the ID of the service instance to which the destination belongs.

- **Example:**

  ### Sample Code

```
"owner": {
        "SubaccountId": "9acf4877-5a3d-43d2-b67d-7516efe15b11",
        "InstanceId": null
    }
```

Back to Response Structure

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.

- **Example:**

  ### Sample Code

```
"destinationConfiguration": {
        "Name": "TestBasic",
        "Type": "HTTP",
        "URL": "http://sap.com",
        "Authentication": "BasicAuthentication",
        "ProxyType": "OnPremise",
        "User": "test",
        "Password": "pass12345"
    }
```

Back to Response Structure

Authentication Tokens

Note

This property is only applicable to destinations that use the following authentication types: *BasicAuthentication*, *OAuth2SAMLBearerAssertion*, *OAuth2ClientCredentials*, *OAuthUserTokenExchange*, *OAuth2JWTBearer*, *OAuth2Password*, *SAMLAssertion*, *OAuth2RefreshToken*, *OAuth2RefreshToken*, *OAuth2TechnicalUserPropagation*.

Restriction

The section will contain an error if the `tokenServiceUrl` is a private endpoint (like *localhost*) and `ProxyType` is `Internet`, as the automation cannot be performed on the server side in this case.

- **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`.

- `refresh_token` (optional) (only in OAuth2 destinations): A refresh token, returned by the OAuth service. It can be used to renew the access token via OAuth Refresh Token Authentication.

- **Example:**

Sample Code

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

Back to Response Structure

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

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

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
Operator and/or Developer	Prerequisites

Task Type	Task
Developer	Automated Access Token Retrieval • Determine the Propagated User ID • Propagate User Attributes • Scenarios

Prerequisites

- You have configured an `OAuth2SAMLBearerAssertion` destination. See OAuth SAML Bearer Assertion Authentication.
- Unless using the destination property `SystemUser`, the user's identity should be represented by a JSON Web token (JWT).

  ### ⚠ Caution

  The `SystemUser` 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:

  - Basic Authentication (see Client Authentication Types for HTTP Destinations)
  - Client Certificate Authentication (see Client Authentication Types for HTTP Destinations)
  - OAuth Client Credentials Authentication

  To extend an OAuth access token's validity, consider using an OAuth refresh token.

  ### ℹ 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

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.

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
System User	The system user is a special user ID that is hardcoded in your destination as value of the `SystemUser` property. If you set this property, its value is used as the propagated user ID.

Field in the JWT	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:
	• If the `userIdSource` 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:
	  ◦ plain string: the exact match is searched on the root-level element keys of the JWT.
	  ◦ JsonPath expression: lets you use non-root-level elements of the JWT.
	• If the `userIdSource` property is missing, the flow falls back to the destination property `nameIdFormat`. It must have one of the following two values (or not be set at all), otherwise an exception is thrown:
	  ◦ **urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress**: the `email` element of the JWT is the user ID. If there is no such element in the JWT, an exception is thrown.
	  ◦ **urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified** (or not set): the `user_name` element of the JWT is the user ID. If there is no such element in the JWT, an exception is thrown.

Custom User Attribute	Like ***Field in the JWT***, 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 `user_attributes` scope. Otherwise the custom attributes cannot be retrieved, and the operation results in an error. • If the `userIdSource` property is configured in the destination, the same logic applies as for ***Field in the JWT***, but this time on the JSON containing the custom user attributes. • If `userIdSource` is missing or the desired key is not found in the custom attributes, the operation fails (`user ID could not be determined`).

Back to Tasks

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 .

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. If you want to avoid this `user_attributes.` prefix, you can set the `skipUserAttributesPrefixInSAMLAttributes` additional property of the destination to `true`. If you do so, the above example will result in an attribute called `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']`

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

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 `SystemUser` property of an `OAuth2SAMLBearerAssertion` destination maintained in the **subscriber subaccount**, and used by the **provider application**.	Access token, retrieved • via the client credentials of the Destination service instance (bound to the application). • using the subscriber's tenant-specific `Token Service URL`.

Scenario	Authorization Header
Propagate a **technical user** principal, using the `SystemUser` property of an `OAuth2SAMLBearerAssertion` destination maintained in the **same subaccount** where the application is deployed.	Access token, retrieved • 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 `OAuth2SAMLBearerAssertion` 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 • 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 `OAuth2SAMLBearerAssertion` 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 • 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

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.

Exchanging User JWTs via OAuth2UserTokenExchange Destinations

Automatic token retrieval using the `OAuth2UserTokenExchange` authentication type for HTTP destinations.

Content

Prerequisites

You have configured an OAuth2UserTokenExchange destination. See OAuth User Token Exchange Authentication.

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

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.

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.

Back to Content

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: • 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: • 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: • 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.

Multitenancy in the Destination Service

Establilsh multitenancy in the Destination service using subscription-level destinations.

Concept

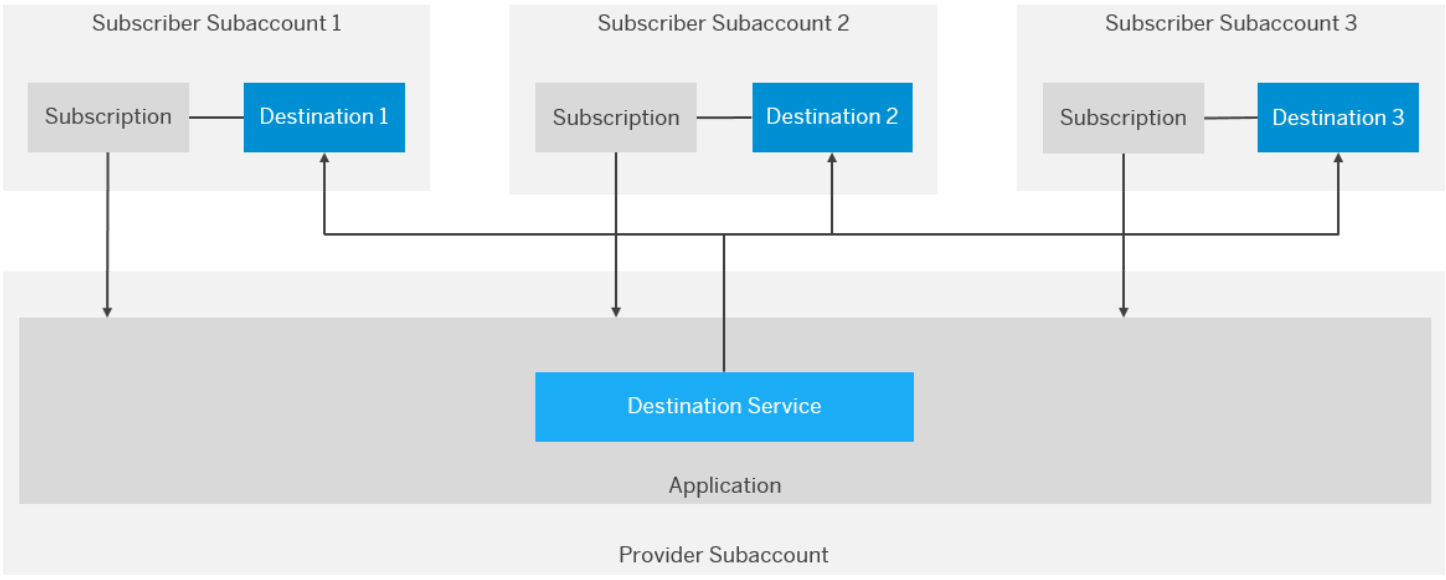
When developing a provider application (SaaS application) that consumes the Destination service, you can choose between the following destination levels:

Level	Who has Access and How?
`Subaccount`	*Any* application using *any* 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 *provider* subaccount (the subaccount in which the instance is provisioned). Each service instance has its own level.
`Subscription`	*Any* application using the *concrete* destination service instance in the context of the *subscribed* 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

Consume a Subscription-Level Destination

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

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

Related Information

Multitenancy in the Connectivity Service

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: Retrieve destination configurations.
- AuthenticationHeaderProvider API: Retrieve prepared authentication headers, ready to be used towards the remote target system.

Add the *Connectivity Apiext* dependency in the `pom.xml` file:

```
<dependency>
    <groupId>com.sap.cloud.connectivity.apiext</groupId>
    <artifactId>com.sap.cloud.connectivity.apiext</artifactId>
    <version>${connectivity-apiext.version}</version>
    <scope>provided</scope>
</dependency>
```

For more information on *SAP Java Buildpack*, see Developing Java in the Cloud Foundry Environment.

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:

   src/main/webapp/META-INF/context.xml

   ### 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. You also need to enable Connectivity `ApiExt` with an environment variable and bind the appropriate services in `manifest.yml`:

   manifest.yml

   ### Sample Code

   ```
   applications:
     - ...
       env:
         USE_CONNECTIVITY_APIEXT: true
       services:
         - xsuaa-instance
         - destination-instance
         - connectivity-instance
         ...
   ```

3. 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
```

4. 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("myDestinationN

// get a single destination property
String authenticationType = destConfiguration.getProperty("Authentication");

// get all destination properties
Map<String, String> allDestinationPropeties = destConfiguration.getAllProperties();
```

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:

a. Instance/subscription level

b. Subaccount level

5. 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("myDestinationN

// get the configured keystore
KeyStore keyStore = destConfiguration.getKeyStore();

// get the configured truststore
KeyStore trustStore = destConfiguration.getTrustStore();

// create sslcontext
TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlg
tmf.init(trustStore);
```

```
KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance(KeyManagerFactory.getDe

// 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();
...
```

AuthenticationHeaderProvider API

Use the `AuthenticationHeaderProvider` API for applications in the Cloud Foundry environment to retrieve prepared authentication headers that are ready to be used to 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.AuthenticationHeaderProvid
                 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 "myAuthHeaderPro
Context ctx = new InitialContext();
AuthenticationHeaderProvider authHeaderProvider = (AuthenticationHeaderProvider) ctx.lookup
```

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
- Create and Bind a Connectivity Service Instance
- 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, proxyI

// look up the connectivity authentication header provider resource called "authHeaderProvider"
Context ctx = new InitialContext();
AuthenticationHeaderProvider authHeaderProvider = (AuthenticationHeaderProvider) ctx.lookup("java

// get header for principal propagation
AuthenticationHeader principalPropagationHeader = authHeaderProvider.getPrincipalPropagationHead
```

```
//insert the necessary headers in the request
HttpGet request = new HttpGet("http://virtualhost:1234");
request.addHeader(principalPropagationHeader.getName(), principalPropagationHeader.getValue());

// 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 destination
```

For more information, see also Principal Propagation SSO Authentication for HTTP.

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 destinationConfigur
```

For more information, see:

- Principal Propagation SSO Authentication for HTTP
- OAuth SAML Bearer Assertion Authentication
- OAuth Client Credentials Authentication

Related Information

Principal Propagation

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
Operator	Prerequisites
Operator and/or Developer	About JCo
	Installation Prerequisites for JCo Applications
	Consume Connectivity via RFC
	Restrictions

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.

- (Only for on-premise backend systems) RFC connectivity between a backend system and the application. To do this, you must install the Cloud Connector 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) and Configure Access Control (RFC).

Back to Tasks

About JCo

To learn in detail about the SAP JCo API, see the JCo 3.0 documentation on SAP Support Portal.

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

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

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.

Back to Tasks

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.

- Your application must not bundle the JCo 3.1 standalone Java archives nor the native library. JCo is already embedded properly in the build pack.
- **JCoServer** functionality cannot be used within SAP BTP.
- **Environment embedding**, such as offered by JCo standalone 3.1, is not possible. This is, however, similar to SAP NetWeaver AS Java.
- A **stateful sequence of function module invocations** must be done in a single HTTP request/response cycle.
- **Logon authentication** only supports user/password credentials (basic authentication) and principal propagation. See Authentication to the On-Premise System.
- The supported set of **configuration properties** is restricted. For details, see RFC Destinations.

Back to Tasks

Related Information

- Use Cases
- Developing Java in the Cloud Foundry Environment
- SAP Java Connector

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	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	Call a function module in a cloud ABAP system via RFC, using a sample Web application (Cloud Foundry environment).
Multitenancy for JCo Applications (Advanced)	Learn about the required steps to make your Cloud Foundry JCo application tenant-aware.
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.

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.

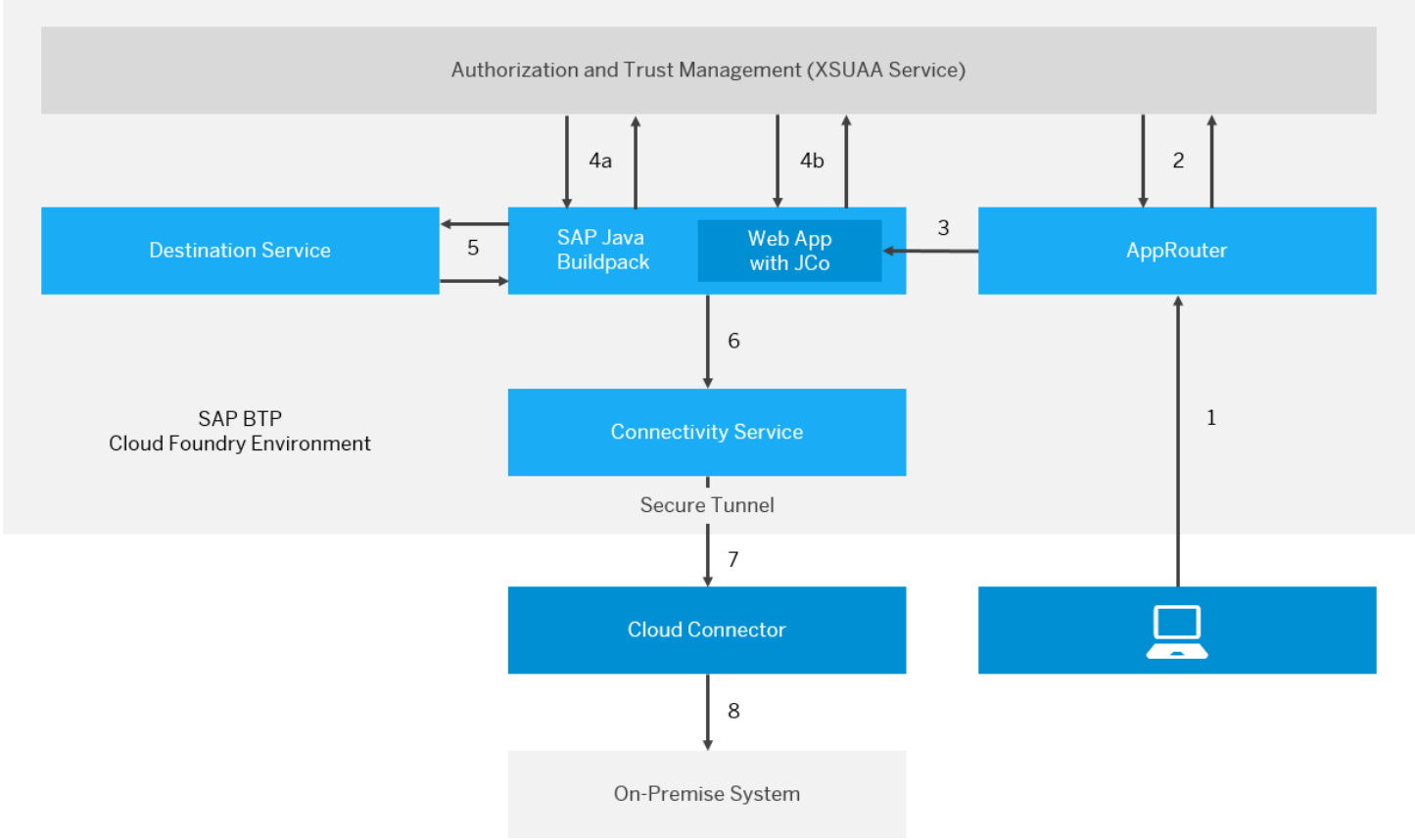
Tasks

Task Type	Task
	Scenario Overview

Task Type	Task
Operator and/or Developer	Connectivity User Roles
	Installation Prerequisites
	Used Values
 Developer	Develop a Sample Web Application
 Operator and/or Developer	Create and Bind Service Instances
 Developer	Deploy the Application
 Operator	Configure Roles and Trust
 Operator and/or Developer	Set Up an Application Router
 Operator	Configure the RFC Destination
	Configure the Cloud Connector
 Operator and/or Developer	Monitoring Your Web Application (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).

   > **Note**
   > AppRouter is only required 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:
   a. Buildpack requests JWT2 to access the Destination service instance (JCo call).
   b. 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.

Back to Tasks

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

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

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

Next Steps

- Develop a Sample Web Application
- Create and Bind Service Instances
- Deploy the Application
- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination
- Configure the Cloud Connector
- Monitoring Your Web Application (Optional)

Related Information

Multitenancy for JCo Applications (Advanced)

Develop a Sample Web Application

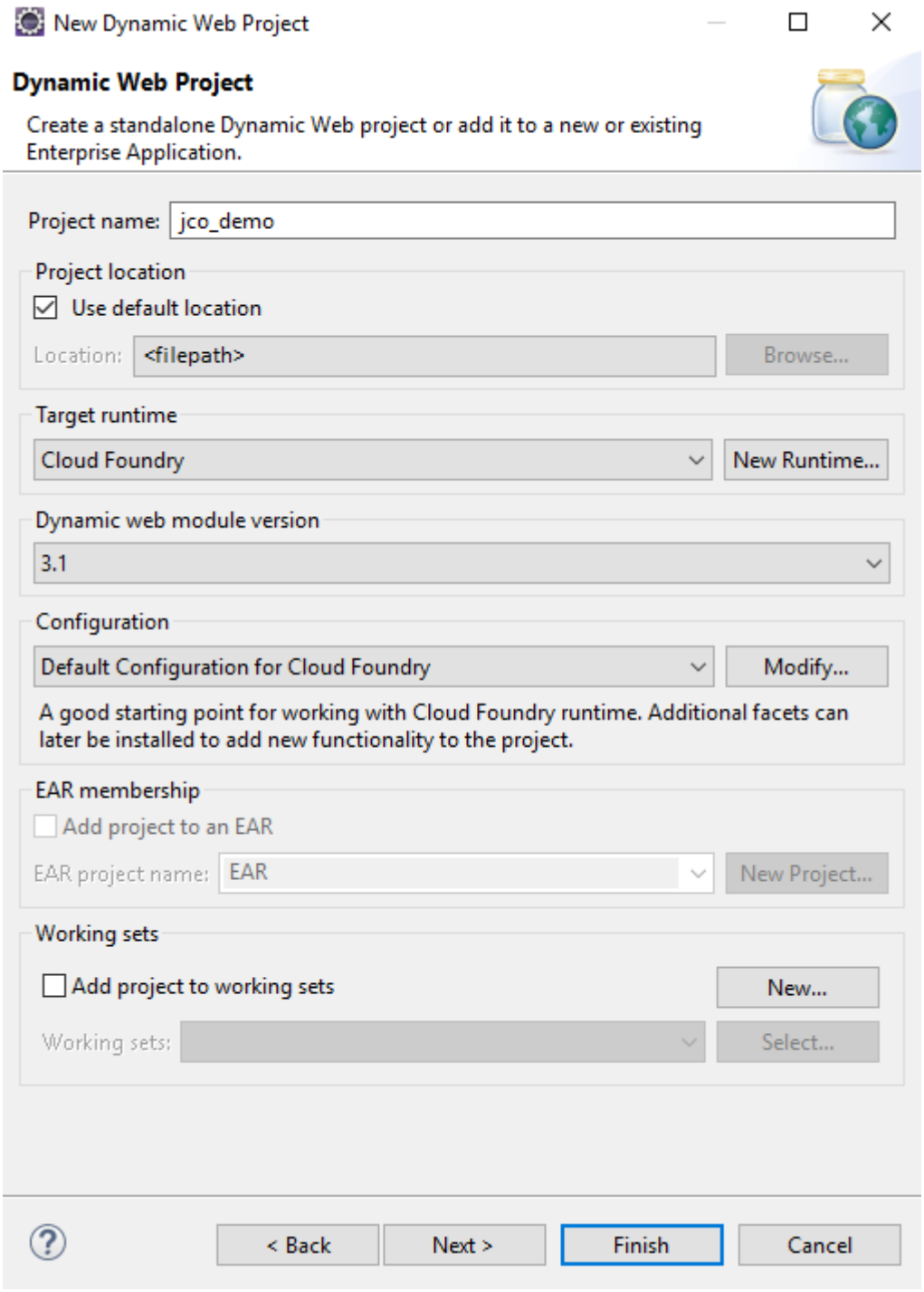
Create a Web application to call an ABAP function module via RFC.

Steps

1. Create a Dynamic Web Project
2. Include JCo Dependencies
3. Create a Sample Servlet

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**.



Back to Steps

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

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 `ConnectivityRFCExample`Java 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("JCoDemoSyste



            // 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</h
            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 sys
            responseWriter.println("<pre>");
            e.printStackTrace(responseWriter);
            responseWriter.println("</pre>");
```

```
                responseWriter.println("</body></html>");

            }

        }

    }
```

5. Save the Java editor and make sure that the project compiles without errors.

Back to Steps

Next Steps

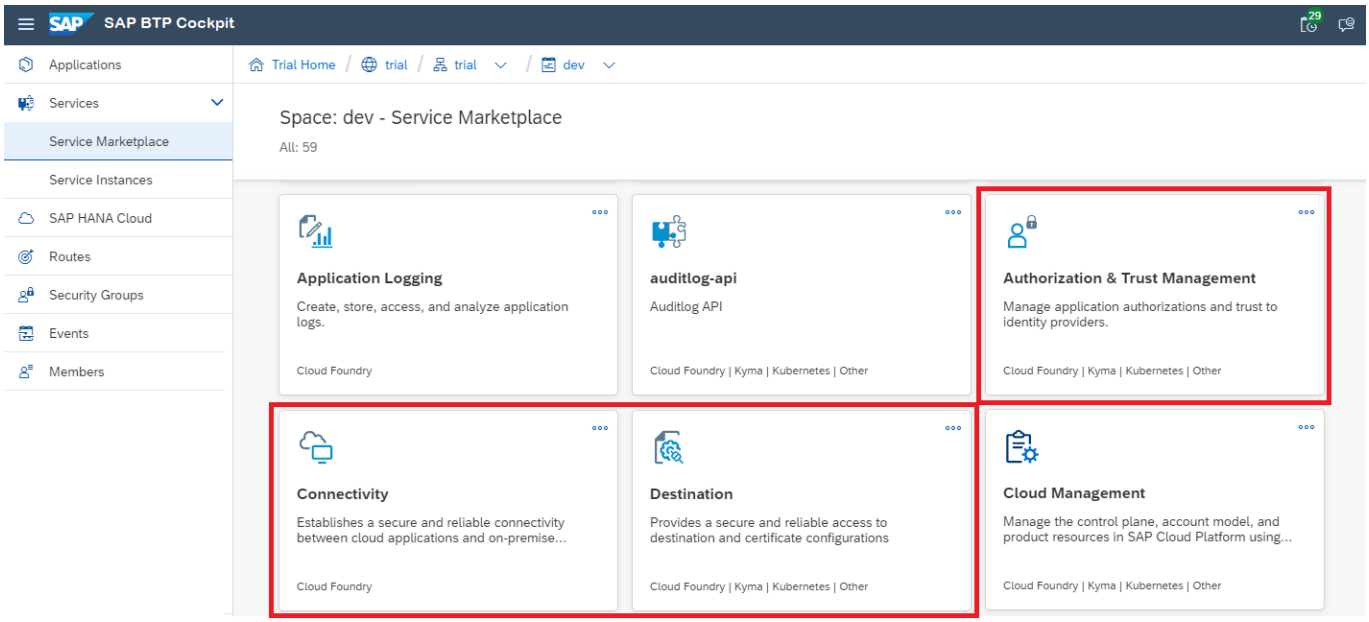
- Create and Bind Service Instances
- Deploy the Application
- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination
- Configure the Cloud Connector
- Monitoring Your Web Application (Optional)

Create and Bind Service Instances

You must create and bind several service instances, before you can use your application.

Procedure

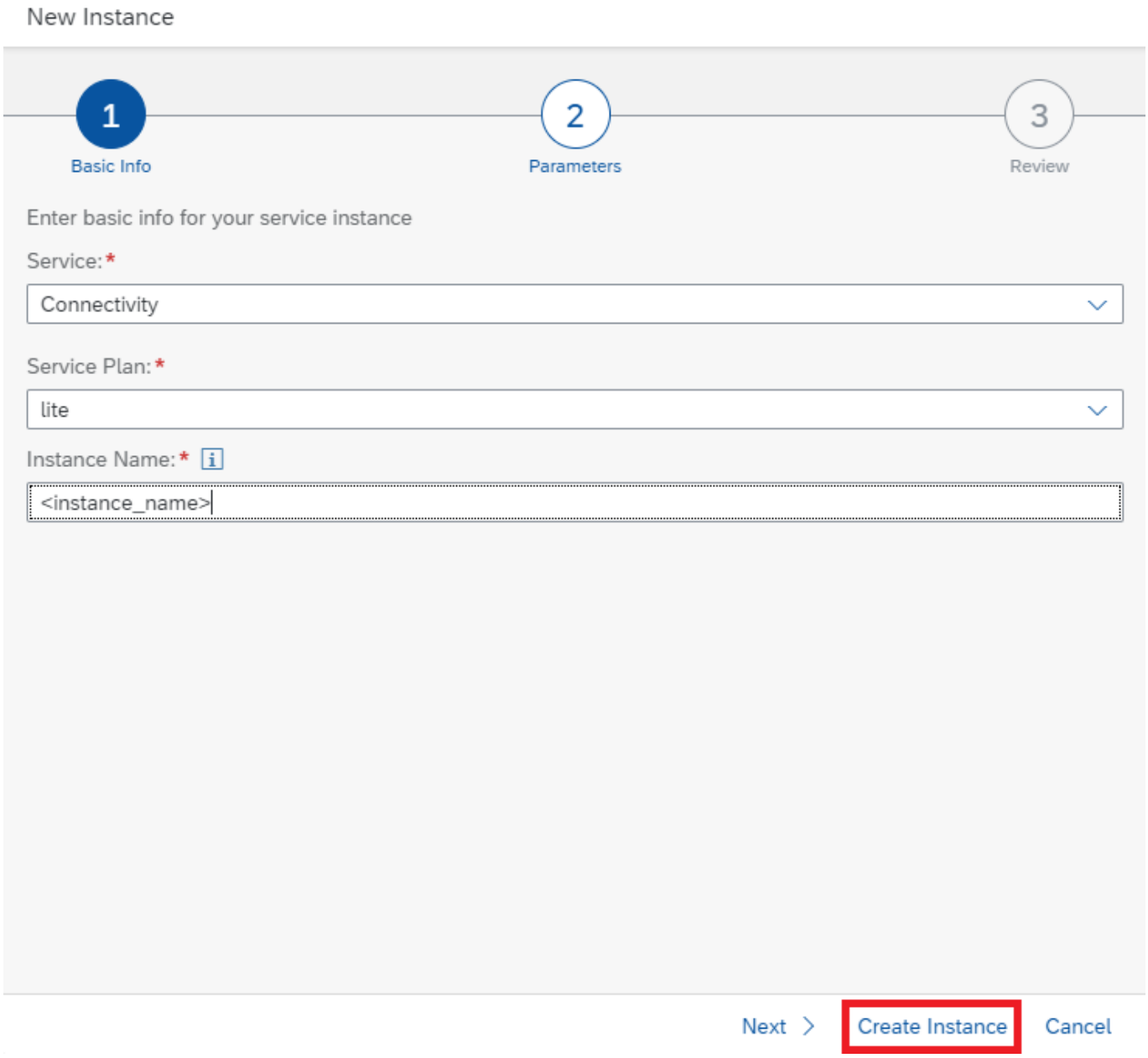
1. Logon to the cloud cockpit and choose your subaccount.
2. Choose the space where you want to deploy your demo application.
3. Choose **Service Marketplace** and find these 3 services:
    - **Connectivity**
    - **Destination**
    - **Authorization & Trust Management** (XSUAA)



4. Create and bind a service instance for each of these services.
    - Connectivity service
    - Destination service
    - Authorization & Trust Management (XSUAA service)

Connectivity Service

1. Choose **Connectivity** > **Create Instance**.
2. Insert an instance name (for example, `connectivity_jco`) and choose **Create Instance** .



Back to Procedure

Destination Service

1. Choose Destination > Create Instance.
2. Insert an instance name (for example, `destination_jco`) and choose Create Instance .

Back to Procedure

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

Next Steps

- Deploy the Application
- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination

- Configure the Cloud Connector
- Monitoring Your Web Application (Optional)

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.

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.

   Note

   JCo supports the usage of X.509 secrets for communication with the Destination/Connectivity service. If you want to use it, you must specify the binding accordingly.

   For more information, see Binding Parameters of SAP Authorization and Trust Management Service.

   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

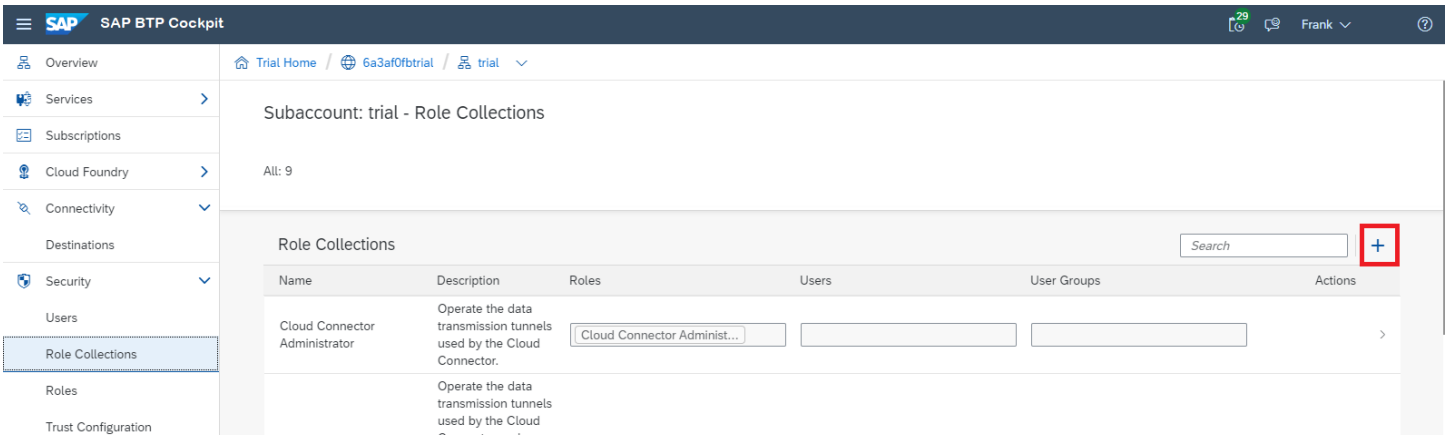
- Configure Roles and Trust
- Set Up an Application Router

- Configure the RFC Destination
- Configure the Cloud Connector
- Monitoring Your Web Application (Optional)

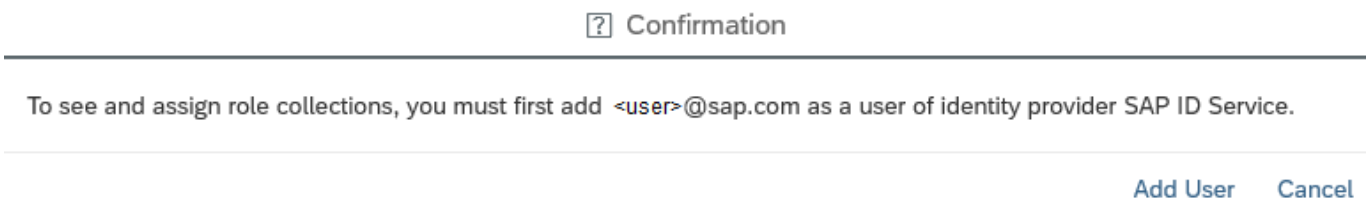
Configure Roles and Trust

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**.



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.



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
- Configure the RFC Destination
- Configure the Cloud Connector
- Monitoring Your Web Application (Optional)

Related Information

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

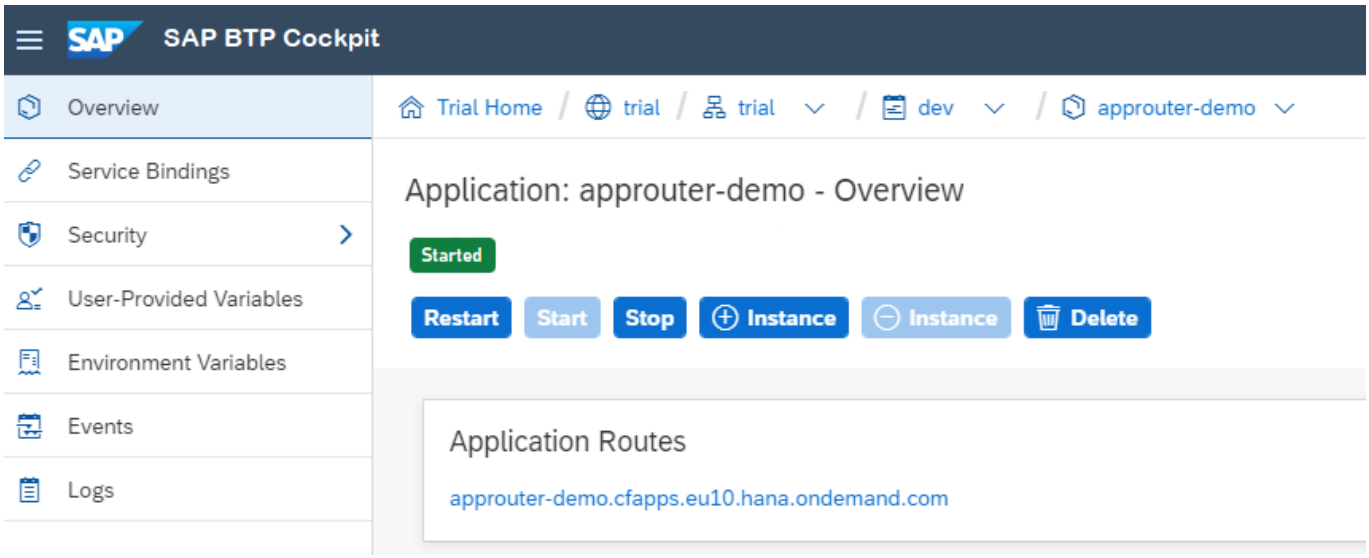
```
---

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.ondeman

      ]

  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) JCO_ERROR_RESOURCE: Destination JCoDemoSystem does not ex
    at com.sap.conn.jco.rt.DefaultDestinationManager.update(DefaultDestinationManager.java:223
    at com.sap.conn.jco.rt.DefaultDestinationManager.searchDestination(DefaultDestinationManag
    at com.sap.conn.jco.rt.DefaultDestinationManager.getDestinationInstance(DefaultDestinatio
    at com.sap.conn.jco.JCoDestinationManager.getDestination(JCoDestinationManager.java: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 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.getDestinatio

                // 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 J
                responseWriter.println("<p>Export parameter ECHOTEXT of STFC_CONN
                responseWriter.println(echotext);
                responseWriter.println("<p>Export parameter RESPTEXT of STFC_CONN
                responseWriter.println(resptext);
                responseWriter.println("</body></html>");
        } catch (AbapException ae) {
                // just for completeness: As this function module does not have a
                // in its signature, this exception cannot occur. But you should
                // take care of AbapExceptions
        } catch (JCoException e) {
                response.addHeader("Content-type", "text/html");
                responseWriter.println("<html><body>");
                responseWriter
                                .println("<h1>Exception occurred while executing
                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
- Configure the Cloud Connector
- Monitoring Your Web Application (Optional)

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.

    a. From the subaccount menu, choose **Connectivity** > **Destinations** > **Import Destination** and upload this file.

    b. 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 **<your space>** > **Services** > **Service Instances** > **<your destination_instance>** 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 fa

    at com.sap.conn.jco.rt.MiddlewareJavaRfc.generateJCoException(MiddlewareJavaRfc.java:487)
    at com.sap.conn.jco.rt.MiddlewareJavaRfc$JavaRfcClient.connect(MiddlewareJavaRfc.java:121(
    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(DefaultConnectionManager.jav
    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
- Monitoring Your Web Application (Optional)

Related Information

Target System Configuration

Configure the Cloud Connector

Configure the system mapping and the function module in the Cloud Connector.

Steps

1. Configure Host Mapping
2. Configure the Function Module

Configure Host Mapping

The Cloud Connector only allows access to trusted backend systems. To configure this, follow the steps below:

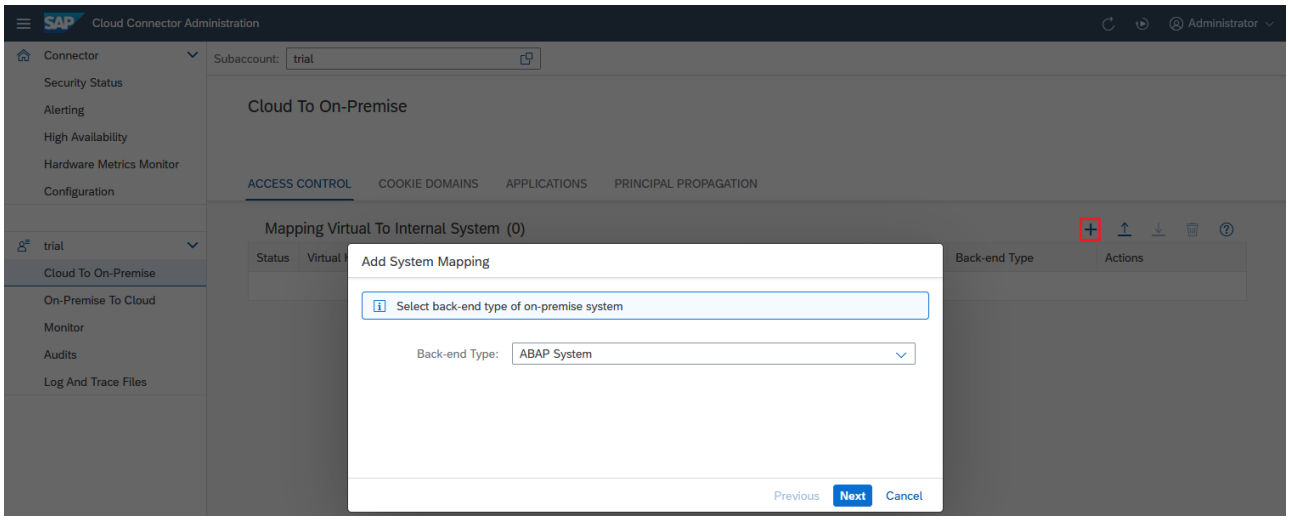
1. Optional: In the Cloud Connector administration UI, you can check under **Audits** whether access has been denied:

   ```
   Denying access for user DEMOUSER to system abapserver.hana.cloud:sapgw42
   [connectionId=-1547299395]
   ```

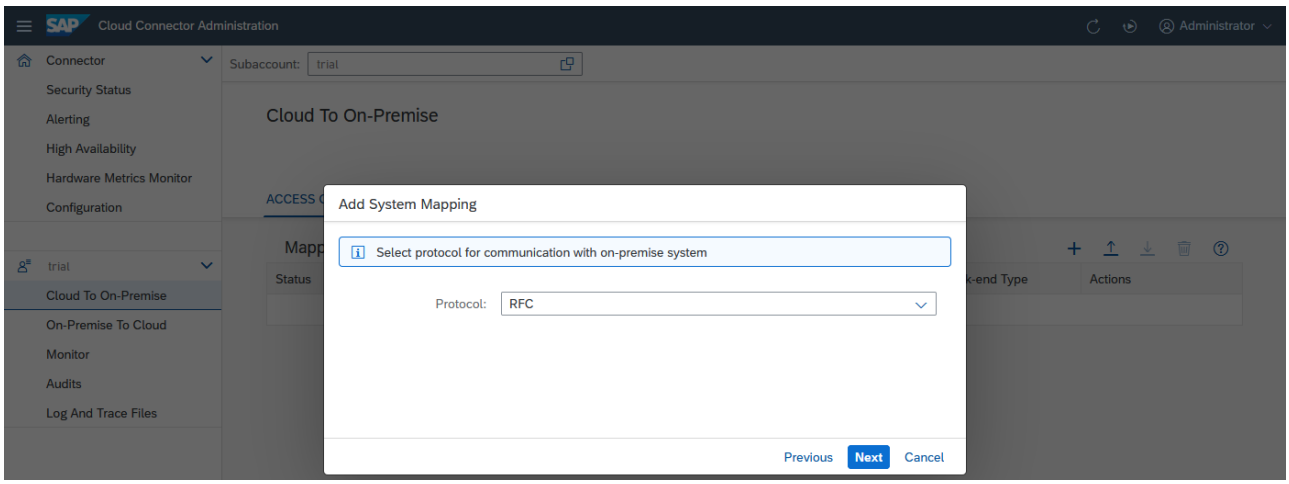
2. In the Cloud Connector administration UI, choose **Cloud To On-Premise** from your **Subaccount** menu, tab **Access Control**.

3. In section **Mapping Virtual To Internal System** choose **Add** to define a new system.

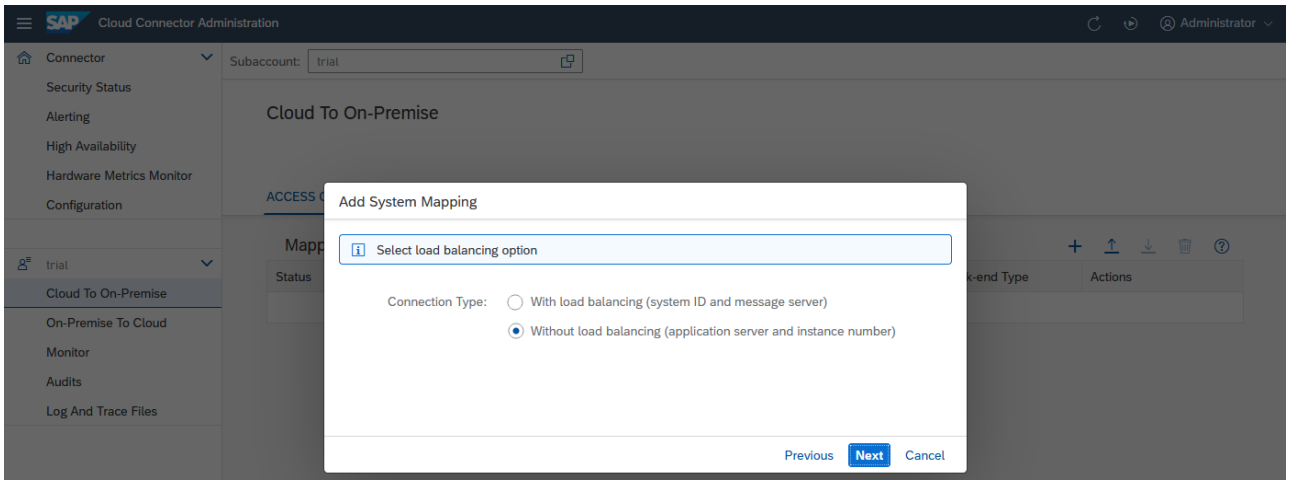
   a. For **Backend Type**, select ABAP System and choose **Next**.



b. For **Protocol**, select RFC and choose **Next**.

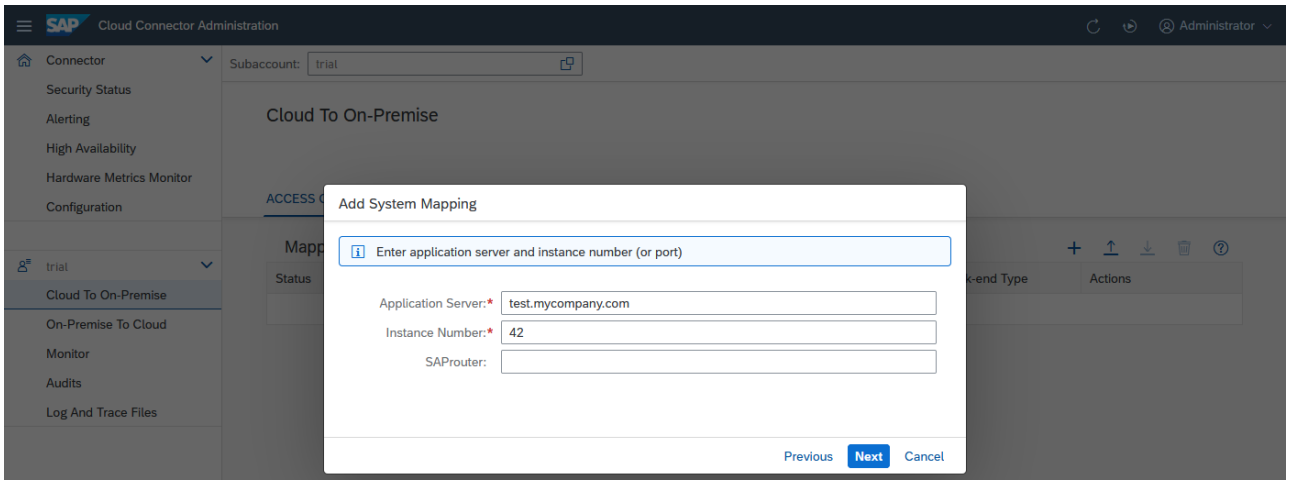


c. Choose option **Without load balancing**.



d. 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:

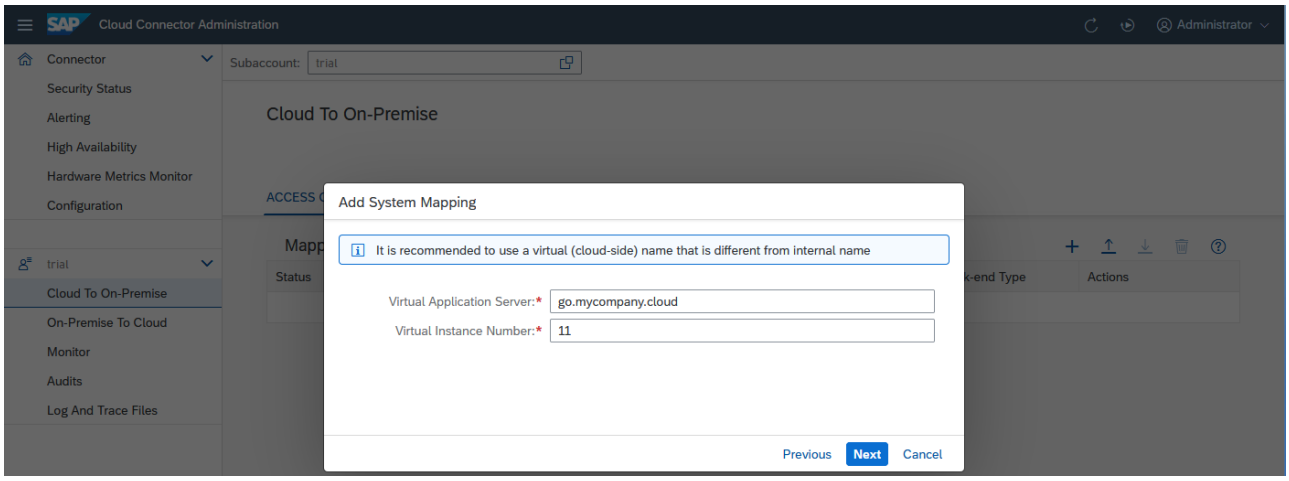


e. Enter server and instance number for virtual mapping.

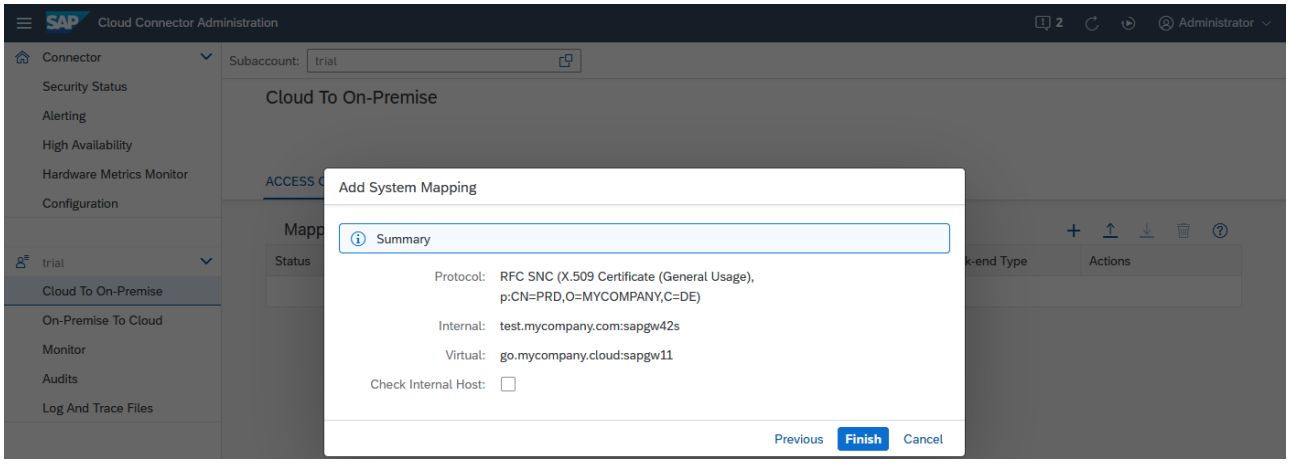
ⓘ Note

The values must match with the ones of the destination configuration in the cloud cockpit.

Example:



f. Summary (example):



4. 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: Acces

    at com.sap.conn.jco.rt.MiddlewareJavaRfc.generateJCoException(MiddlewareJavaRfc.java:632)
    at com.sap.conn.jco.rt.MiddlewareJavaRfc$JavaRfcClient.execute(MiddlewareJavaRfc.java:176
```

```
        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

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:

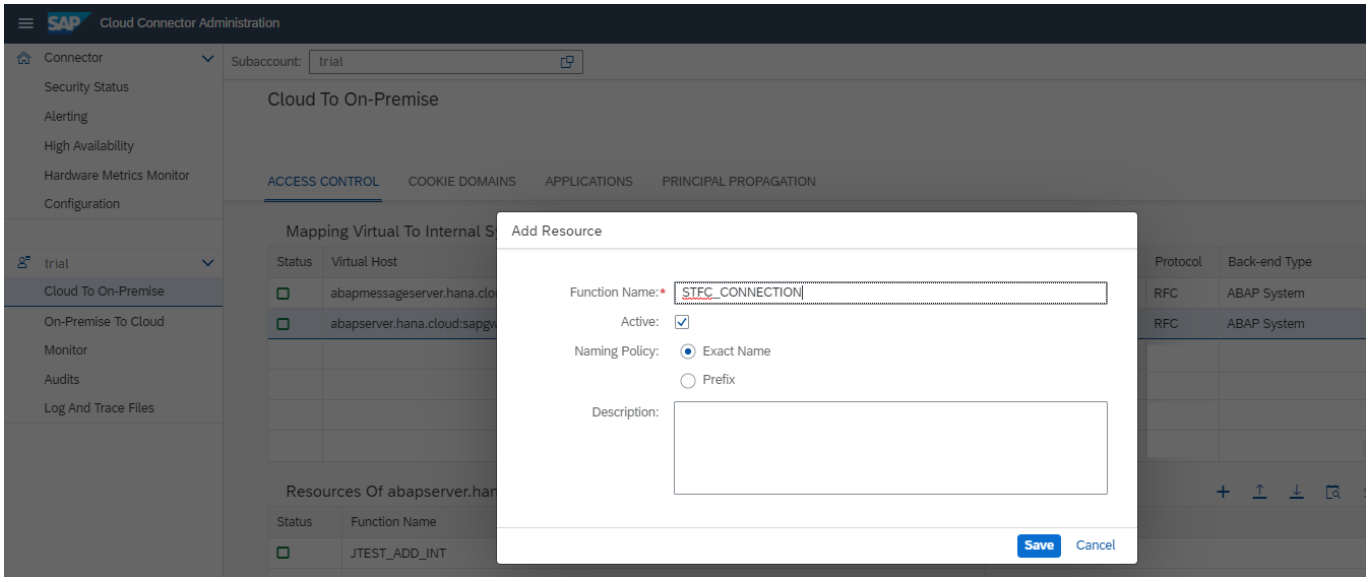
1. Optional: In the Cloud Connector administration UI, you can check under **Monitor** > **Audit** whether access has been denied:

   ```
   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.



5. 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).

Back to Steps

Next Step (Optional)

- Monitoring Your Web Application

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.

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.secu
```

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.

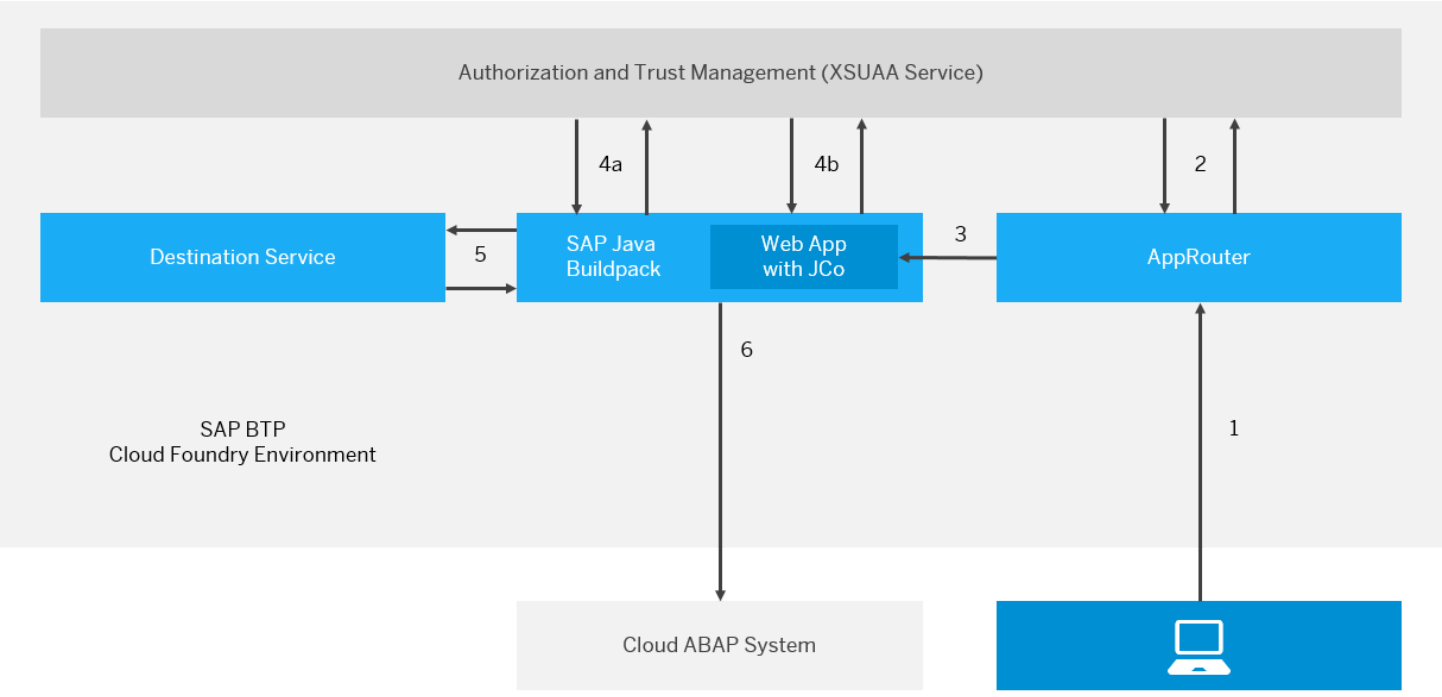
Tasks

Task Type	Task
 Operator and/or Developer	Scenario Overview
	Used Values
	Connectivity User Roles
	Installation Prerequisites
 Developer	Develop a Sample Web Application

Task Type	Task
Operator and/or Developer	Create and Bind Service Instances
Developer	Deploy the Application
Operator	Configure Roles and Trust
Operator and/or Developer	Set Up an Application Router
Operator	Configure the RFC Destination
Operator and/or Developer	Monitoring Your Web Application (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).

   > **Note**
   >
   > AppRouter is only required 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.

Back to Tasks

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

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

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

Next Steps

- Develop a Sample Web Application
- Create and Bind Service Instances
- Deploy the Application
- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination
- Monitoring Your Web Application (Optional)

Related Information

Multitenancy for JCo Applications (Advanced)

Develop a Sample Web Application

Create a Web application to call an ABAP function module via RFC.

Steps

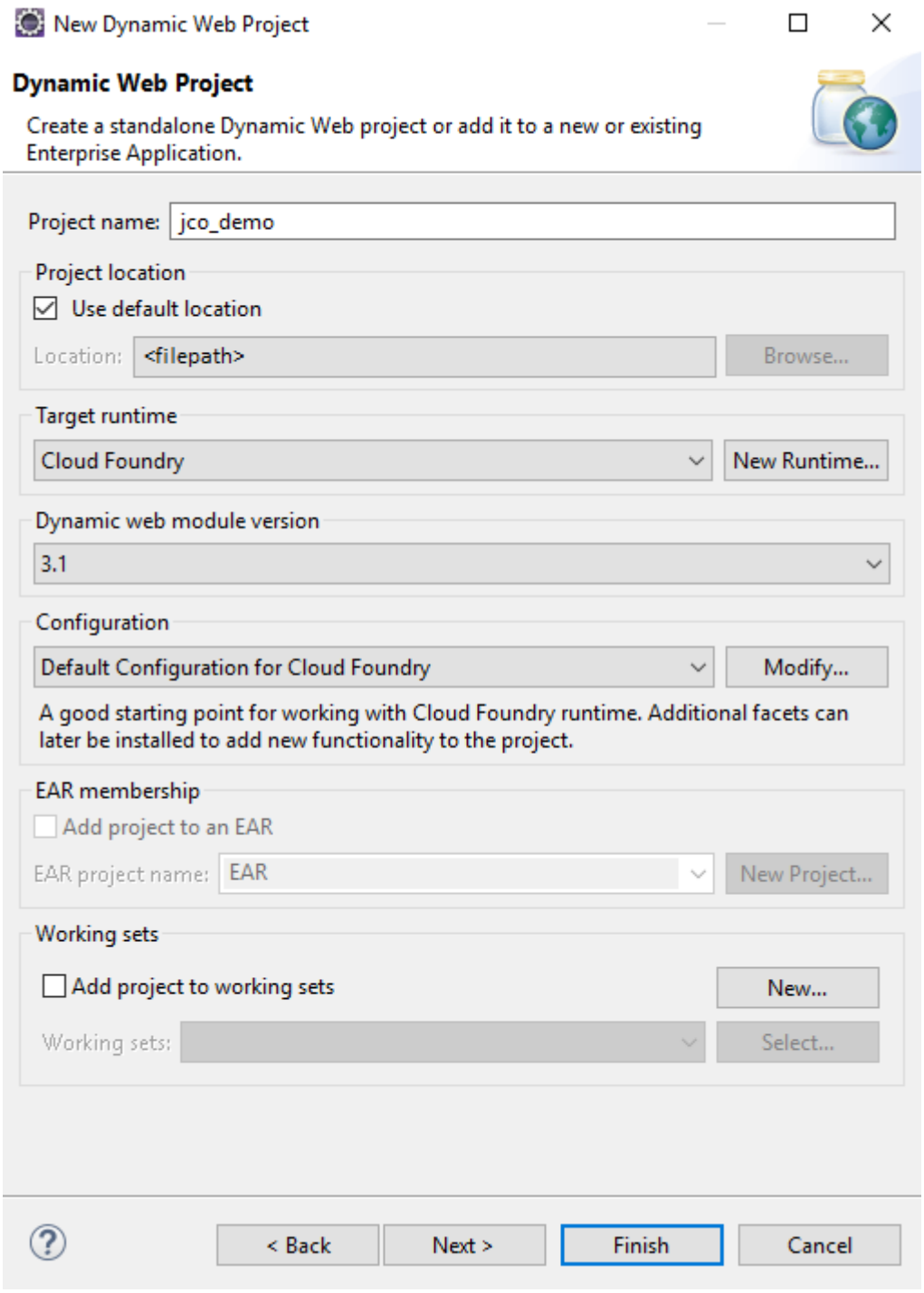
1. Create a Dynamic Web Project
2. Include JCo Dependencies
3. Create a Sample Servlet

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**.



Back to Steps

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

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 `ConnectivityRFCExample`Java 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 AB
 * 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("JCoDemoSyste


            // 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</h
            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 sys
            responseWriter.println("<pre>");
            e.printStackTrace(responseWriter);
```

```
                responseWriter.println("</pre>");

                responseWriter.println("</body></html>");

            }

        }

    }
```

5. Save the Java editor and make sure that the project compiles without errors.

Back to Steps

Next Steps

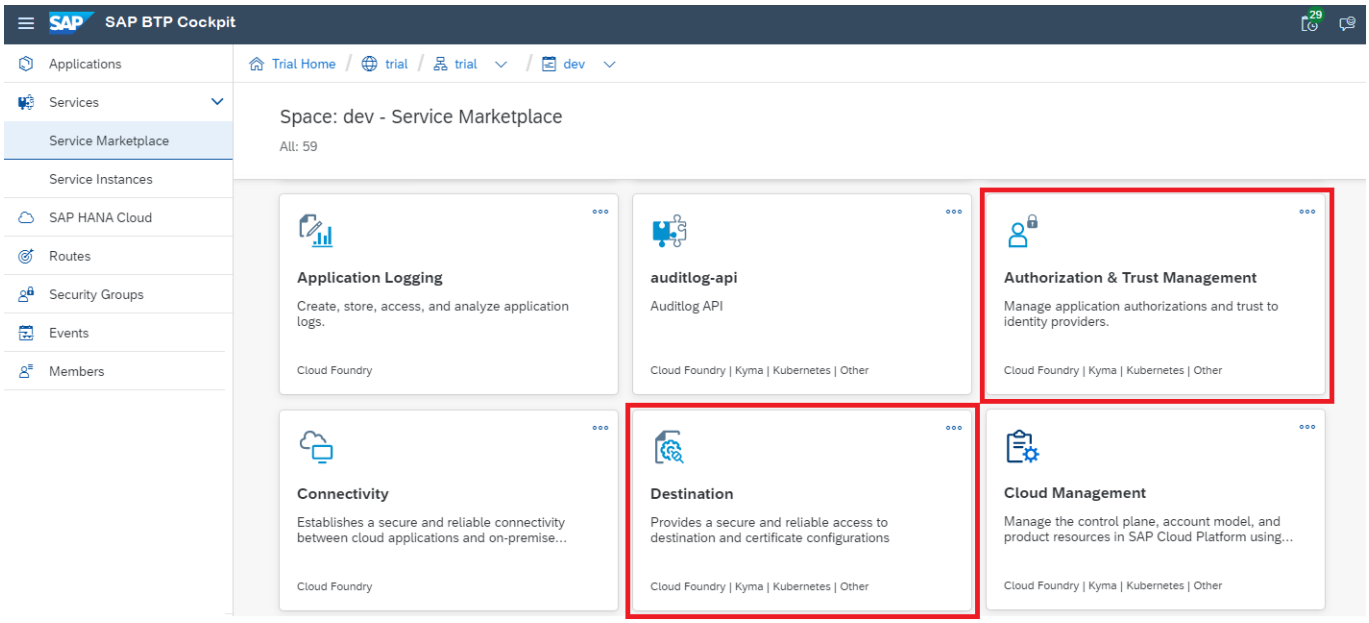
- Create and Bind Service Instances
- Deploy the Application
- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination
- Monitoring Your Web Application (Optional)

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)



4. Create and bind a service instance for each of these services.
    - Destination service
    - Authorization & Trust Management (XSUAA service)

Destination Service

1. Choose **Destination** > **Create Instance** >.
2. Insert an instance name (for example, `destination_jco`) and choose **Create Instance**.

Back to Procedure

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

Next Steps

- Deploy the Application
- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination
- Monitoring Your Web Application (Optional)

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.

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.

ℹ Note

JCo supports the usage of X.509 secrets for communication with the Destination/Connectivity service. If you want to use it, you must specify the binding accordingly.

For more information, see Binding Parameters of SAP Authorization and Trust Management Service.

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.

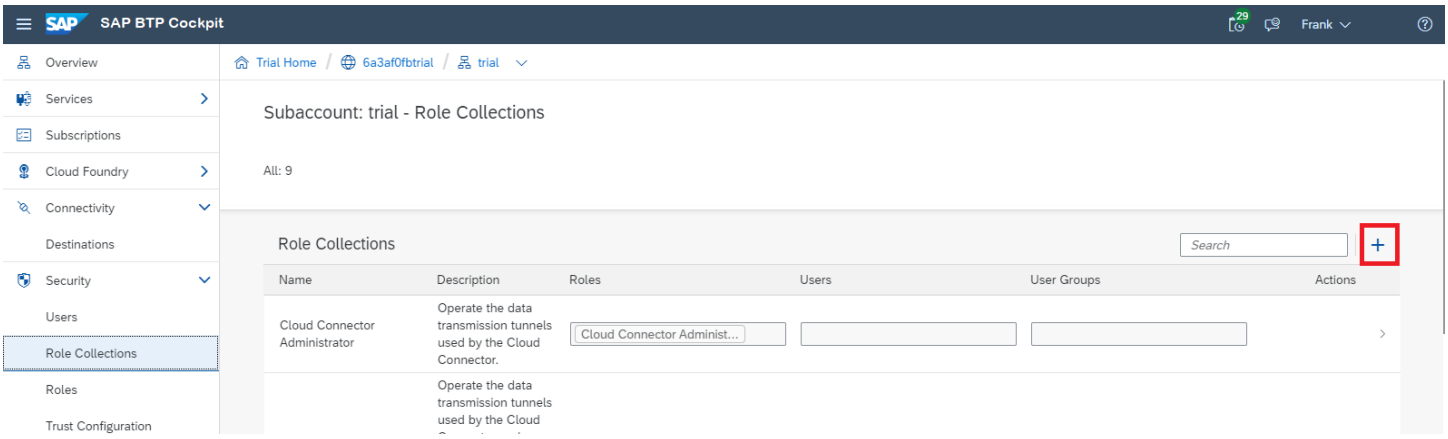
Next Steps

- Configure Roles and Trust
- Set Up an Application Router
- Configure the RFC Destination
- Monitoring Your Web Application (Optional)

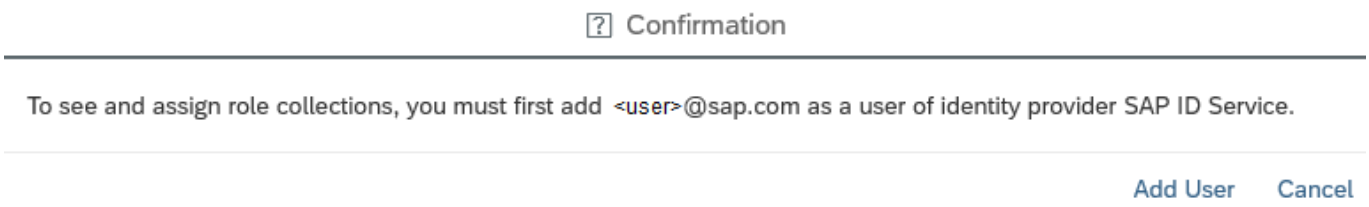
Configure Roles and Trust

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**.



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.



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
- Configure the RFC Destination
- Monitoring Your Web Application (Optional)

Related Information

Working with Role Collections

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

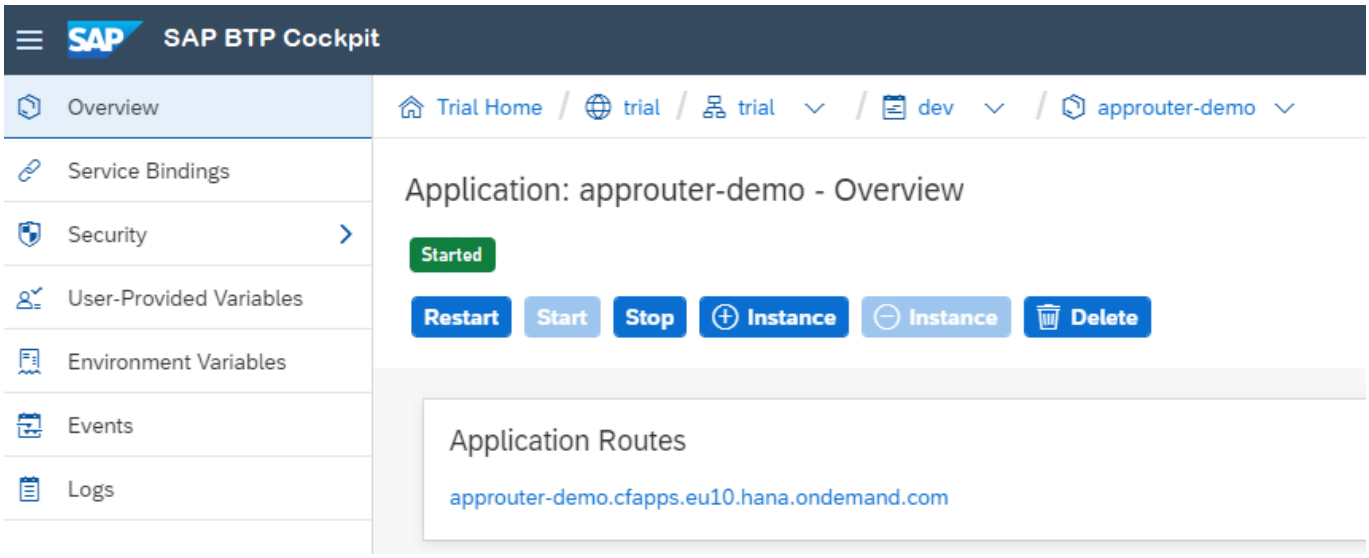
```
---

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.ondeman

      ]

  services:

    - xsuaa_jco
```

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) JCO_ERROR_RESOURCE: Destination JCoDemoSystem does not ex
    at com.sap.conn.jco.rt.DefaultDestinationManager.update(DefaultDestinationManager.java:223
    at com.sap.conn.jco.rt.DefaultDestinationManager.searchDestination(DefaultDestinationManag
    at com.sap.conn.jco.rt.DefaultDestinationManager.getDestinationInstance(DefaultDestinatio
    at com.sap.conn.jco.JCoDestinationManager.getDestination(JCoDestinationManager.java: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 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.getDestinatio

                        // 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 J
                        responseWriter.println("<p>Export parameter ECHOTEXT of STFC_CONN
                        responseWriter.println(echotext);
                        responseWriter.println("<p>Export parameter RESPTEXT of STFC_CONN
                        responseWriter.println(resptext);
                        responseWriter.println("</body></html>");
                } catch (AbapException ae) {
                        // just for completeness: As this function module does not have a
                        // in its signature, this exception cannot occur. But you should
                        // take care of AbapExceptions
                } catch (JCoException e) {
                        response.addHeader("Content-type", "text/html");
                        responseWriter.println("<html><body>");
                        responseWriter
                                        .println("<h1>Exception occurred while executing
                        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
- Monitoring Your Web Application (Optional)

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.
   a. From the subaccount menu, choose **Connectivity** > **Destinations** > **Import Destination** and upload this file.
   b. 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 **<your space>** > **Services** > **Service Instances** > **<your destination_instance>** and choose **Destinations**.
3. Specify a trust/key store or security information, see WebSocket Connection.

Next Steps

- Monitoring Your Web Application (Optional)

Related Information

Target System Configuration

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.

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.secu
```

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 or Invoke ABAP Function Modules in Cloud ABAP Systems, for multitenancy.

Steps

1. Prerequisites
2. Adjust the Application Router
3. Adjust the XSUAA Service Instance and Roles
4. Make the Application Subscribable
5. Create the SAAS Provisioning Service
6. Subscribe to the Application
7. Create a New Route

Prerequisites

- Your runtime environment uses SAP Java Buildpack version 1.9.0 or higher.
- You have successfully completed one of these precedures:

  Invoke ABAP Function Modules in On-Premise ABAP Systems

  Invoke ABAP Function Modules in Cloud ABAP Systems
- You have created a second subaccount (in the same global account), that is used to subscribe to your application.

Back to Steps

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
      ]
  services:
    - xsuaa_jco
```

Back to Steps

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":"$XSAPPNAME.Callback",
         "description":"With this scope set, the callbacks for tenant onboarding, offboarding an
         "grant-as-authority-to-apps":[
            "$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 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

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
- PUT tenant subscription to an application

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 Servle
        try {
            DependantServiceDto destinationService = createLPSDependency(DESTINATION_SERVICE_NAM
            DependantServiceDto connectivityService = createLPSDependency(CONNECTIVITY_SERVICE_N
            List<DependantServiceDto> dependenciesList = Arrays.asList(destinationService, conne
```

```
            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 JSONExcept
        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_

    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#VC
     */
    public static JSONObject getServiceCredentials(String serviceName) throws JSONException
    {
        return new JSONObject(VCAP_SERVICES).getJSONArray(serviceName).getJSONObject(0).getJSONObje
    }
```

```
    /**
     * 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#VC
     */
    public static JSONObject getServiceCredentials(String serviceName, String serviceInstanceName)
    {
        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 bo
    }

    /**
     * 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#VC
     */
    public static String getServiceCredentialsAttribute(String serviceName, String attributeName) t
    {
        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_CONN
    }
}
```

Back to Make the Application Subscribable

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 Servle
        PayloadDataDto payload = new Gson().fromJson(request.getReader(), PayloadDataDto.class);
        response.getWriter().println("https://" + payload.getSubscribedSubdomain() + ".approuter
    }

    @Override
    protected void doDelete(HttpServletRequest req, HttpServletResponse resp) throws ServletExce
        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[] dependantServiceInstanceAppIds = null;

        private Map<String, String> additionalInformation;

        public PayloadDataDto() {}

        public PayloadDataDto(String subscriptionAppName, String subscriptionAppId,
                        String subscribedTenantId, String subscribedSubdomain, String subscriptionA
                        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[] getDependantServiceInstanceAppIds() {
                return dependantServiceInstanceAppIds;
        }

        public void setDependantServiceInstanceAppIds(
                        String[] dependantServiceInstanceAppIds) {
                this.dependantServiceInstanceAppIds = dependantServiceInstanceAppIds;
        }

        public String toString() {
                return String.format("Payload data: subscriptionAppName=%s, subscriptionAppId=%s, s
                                + "subscribedSubdomain=%s subscriptionAppPlan=%s subscriptionAppAmo
                                this.subscribedTenantId, this.subscribedSubdomain, this.subscriptio
        }
}
```

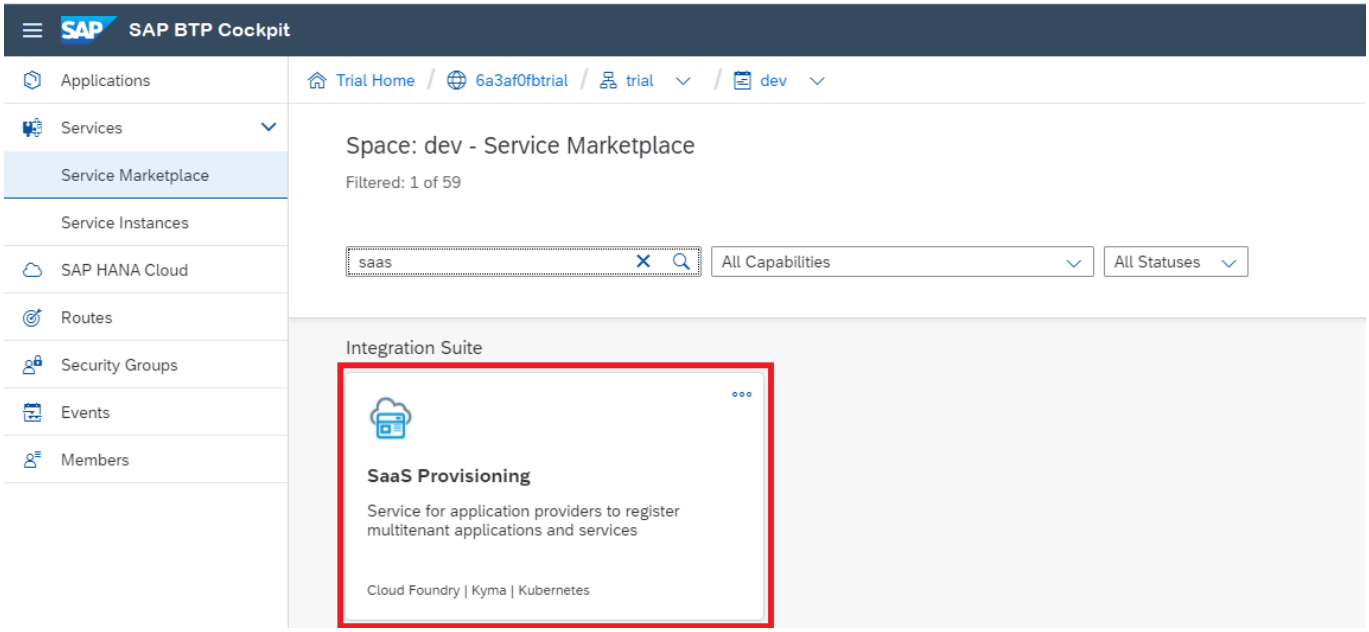
Back to Make the Application Subscribable

Back to Steps

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

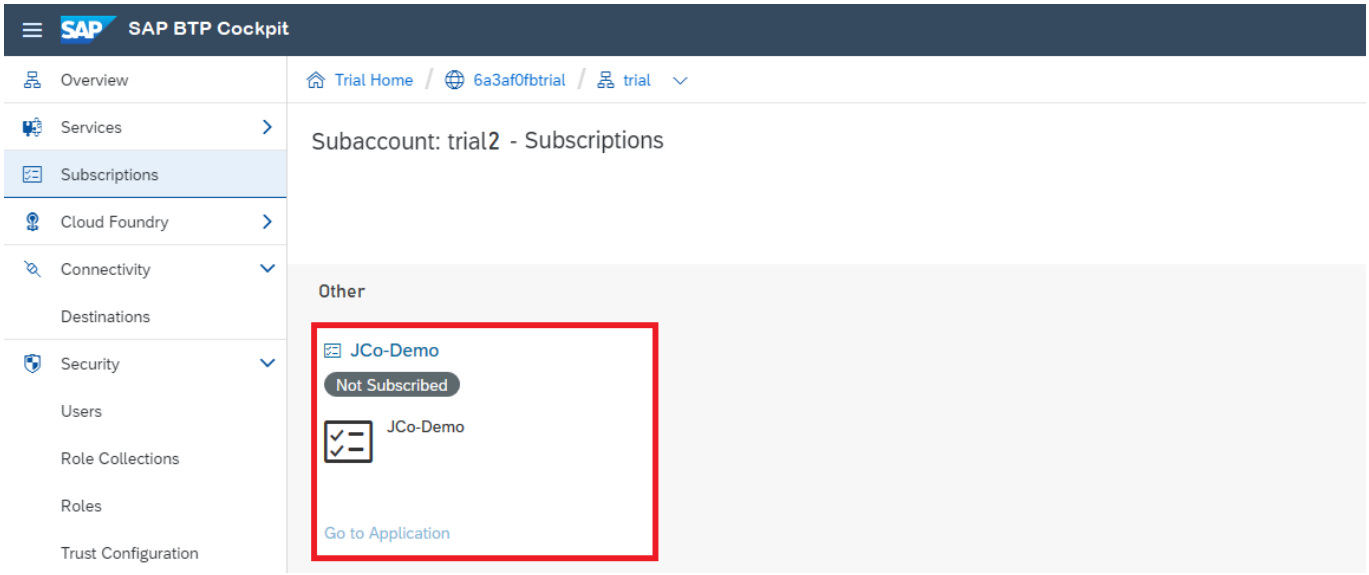
```
{
    "xsappname" : "jco-demo-p42424242",
    "appName" : "JCo-Demo",
    "appUrls": {
            "getDependencies": "https://jco-demo-p42424242.cfapps.eu10.hana.ondemand.com/ca
            "onSubscription" : "https://jco-demo-p42424242.cfapps.eu10.hana.ondemand.com/ca
    }
}
```

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**.

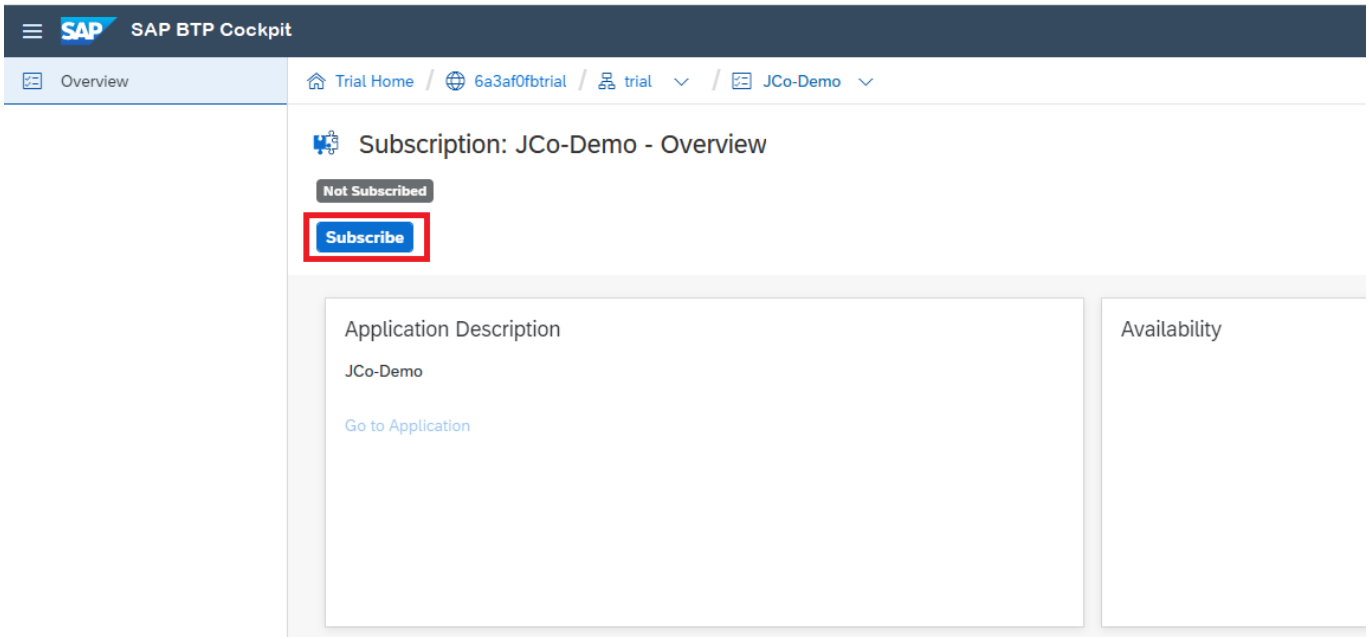
Back to Steps

Subscribe to the Application

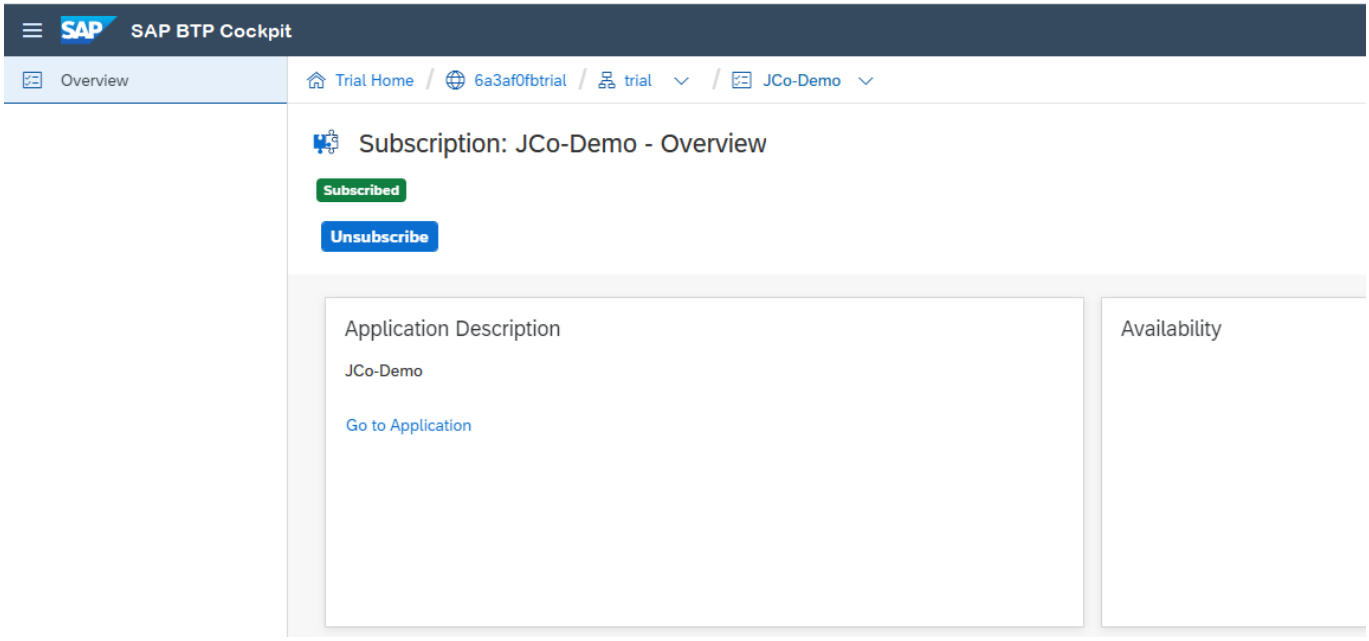
1. To subscribe the new application from a different subaccount, go to **Subscriptions** in the cockpit:



2. Click on **JCo-Demo**.
3. In the next window, choose **Subscribe**:



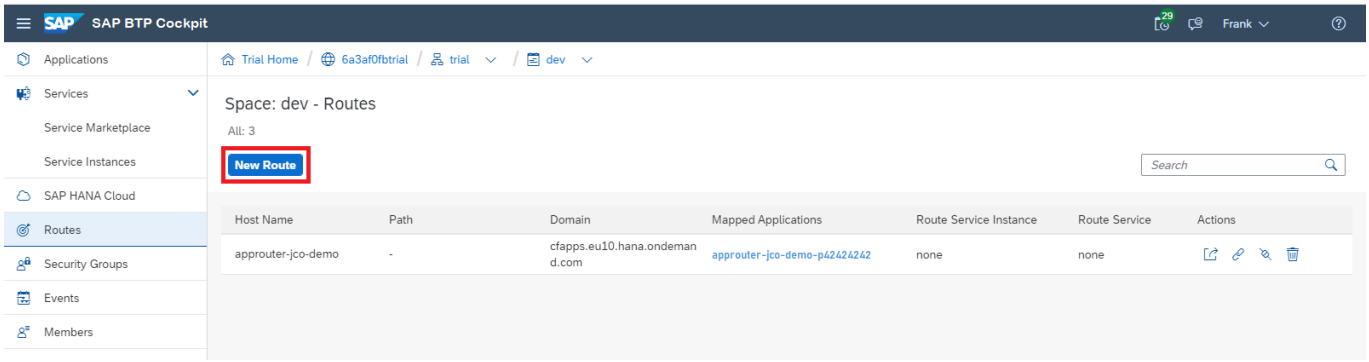
4. If the subscription was successful, your window should look like that:



Back to Steps

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**:

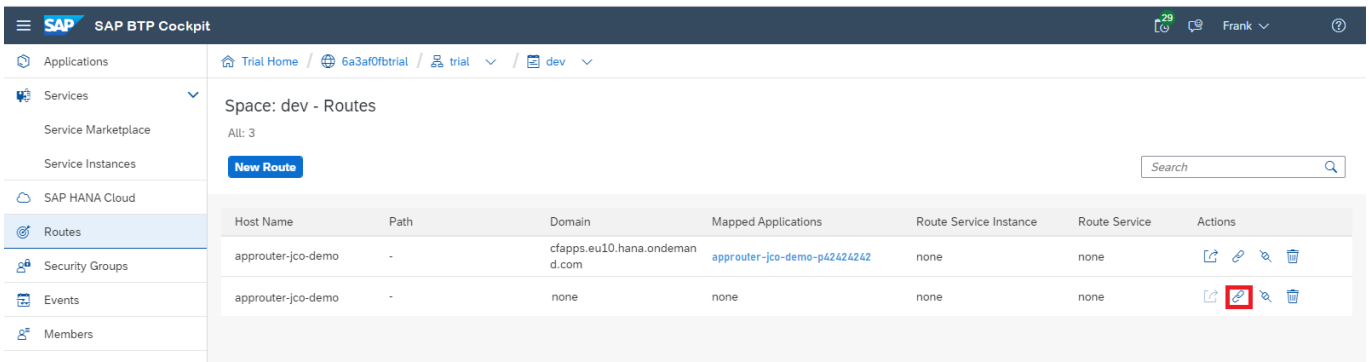


2. For *<Domain>*, select the landscape your application is deployed in (e.g. `cfapps.eu10.hana.ondemand.com`).

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):



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

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 informatioon, see Configuring Principal Propagation.

Procedure

- Make sure your RFC destination is configured with `jco.destination.auth_type=PrincipalPropagation`. For more informatioon, see User Logon Properties.
- 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 informatioon, see Set Up an Application Router.

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
Set up and run the Cloud Connector according to the highest security standards.	Security Guidelines

More Information

SAP BTP Security Recommendations collects information that lets you secure the configuration and operation of SAP BTP services in your landscape.

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 for SAP BTP Connectivity incident, see Connectivity Support (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
Troubleshoot connection problems and view different types of logs and traces in the Cloud Connector.	Troubleshooting
Detailed support information for SAP Connectivity service and the Cloud Connector.	Connectivity Support

Resilience Recommendations

Improve resilience of your SAP BTP applications.

While SAP strives to ensure the highest possible availability of the provided services, true resilience actually is a two-way collaboration between client and server. Thus, it is important to implement any applications using the Connectivity and/or Destination services (or any other services for that matter) in a resilient way. This page gives suggestions on measures to take in order to endure short disruptions, network issues, slowdowns or other abnormal situations that might arise. By doing this, these application can withstand such disruptions, ensuring business continuity in the face of underlying issues in the platform.

ℹ Note

When using client libraries like the BTP security library , the Cloud SDK , and so on, many of these recommendations may already be included. However, we recommend that you double-check these features, as they might require additional configuration.

Caching

Caching is an important pillar of resilience. It is the act of storing data (acquired from an external resource or as a result of a heavy computation) for future use. Caching operations must be done within a reasonable timespan to avoid the risk of data becoming invalid or outdated. So, caching lets you reduce the number of risky or heavy operations that go over the network towards an external resource. By doing this, the risk of failure is reduced, and caching additionally improves the performance of the application.

ℹ Note

Be careful what kind of data you cache and for how long. It is especially important to consider the handling of sensitive data (personal data, security objects, and so on).

⚠ Caution

Make sure you limit the size of the caches to avoid memory issues. There are different options when the maximum number of entities in the cache is reached. For example, you can drop the oldest entity from the cache in favor of the new one.

Token Caching

Caching access tokens to the services is highly recommended for the period of their validity. Access tokens to the service include the timestamp on which the token is no longer valid (as per JWT specification). Thus the tokens can be reused at any time before said timestamp. Keep in mind that tokens are issued for the specific clients and in the context of the specific tenant. Thus this should be taken into account when designing the caching mechanism.

Destination/Certificate Caching

The entities of the Destination service (be it destination configurations or certificate configuration) can also be cached. This includes both responses for single entities, as well as the list of entities responses. The drawback of this is that any changes in the configurations will not become immediately available to the application, thus a reasonable caching time (3-5 minutes is usually a good choice) must be set.

If you do not expect time critical changes in the destination configuration, you can set the caching time even higher (for example, an hour). Also, we recommend that you refresh the cache on-demand instead of calling the Destination service every 3-5 minutes.

By caching applications, they can withstand temporary issues with the service or the network. We also recommend that in cases where the cache has expired but retrieval of the updated entities does not succeed to still use the cached value as a further resilience measure.

Retry Logic

When requests (whether it is the token call, the service call or the business call) fail, it is always a good idea to retry the call over a reasonable time. It also makes sense to have some delay between retries, for example, at least 100 milliseconds. You can even make this delay increase with each subsequent retry. By doing this, you can handle short intermittent issues without having business impact.

Pagination

The APIs of the Destination service that return multiple entities support pagination. We highly recommended that you make use of pagination if you use these APIs and have a high number of entities (more than 100 entities). By doing this, you ease the pressure on the service but also make processing lighter on client side as the list can be handled in batches. See more about pagination in the documentation of the Destination Service REST API.

Timeouts

Connect and read timeouts help you keep your application resources from getting stuck in abnormally long processing. In some cases, one attempt to get data might get stuck, but abandoning it and trying again might immediately succeed.

We recommend you use appropriate *connect* and *read* timeouts when communicating with the services. The *connect* timeout is recommended to be on the low end - a couple of seconds. The *read* timeout would depend on the semantic of the call being made. A call to the Destination service REST API would not require a high read timeout. Around half a minute would be sufficient. The same is true for token retrieval calls. For business calls, including calls to on-premise systems via the Connectivity service, the timeout would be based on the type of the call and is heavily scenario-specific.

Circuit Breaker

The circuit-breaker architecture pattern can be a powerful tool for handling a misbehaving dependency. It reduces the effort done by your application when communicating to a service which is identified as not working or unstable, while waiting for the dependency to come back online.

For more information on this pattern, see https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker .

Cloud Connector High Availability

See High Availability Setup.

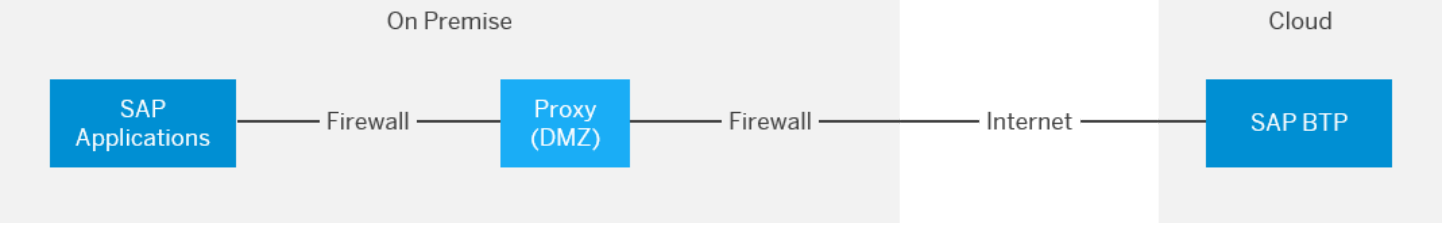
Connectivity via Reverse Proxy

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.

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 (Cloud Connector)
- Administration (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 *cloud-side* issues with **cloud to on-premise** connectivity, where: • The **environment is unknown** or • The issue is **not related** to a **specific environment**
BC-CP-CON-CF	For *cloud-side* issues with **cloud to on-premise** connectivity in the SAP BTP **Cloud Foundry environment**.
BC-CP-CON-S4HC	For *cloud-side* issues with **cloud to on-premise** connectivity in an **S/4HANA Cloud** system.
BC-CP-CON-K8S-PROXY	For cloud-side 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 *cloud-side* issues with **cloud to on-premise** connectivity in the SAP BTP **ABAP environment**.
BC-NEO-CON	For *cloud-side* issues with **cloud to on-premise** connectivity in the SAP BTP **Neo environment**.
Destinations	
BC-CP-DEST	For issues with **destination configurations**, where: • 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: • REST API • Instance creation, etc.
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: • 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

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.

For detailed support strategy information, check SAP note 3302250.

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.

> **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.