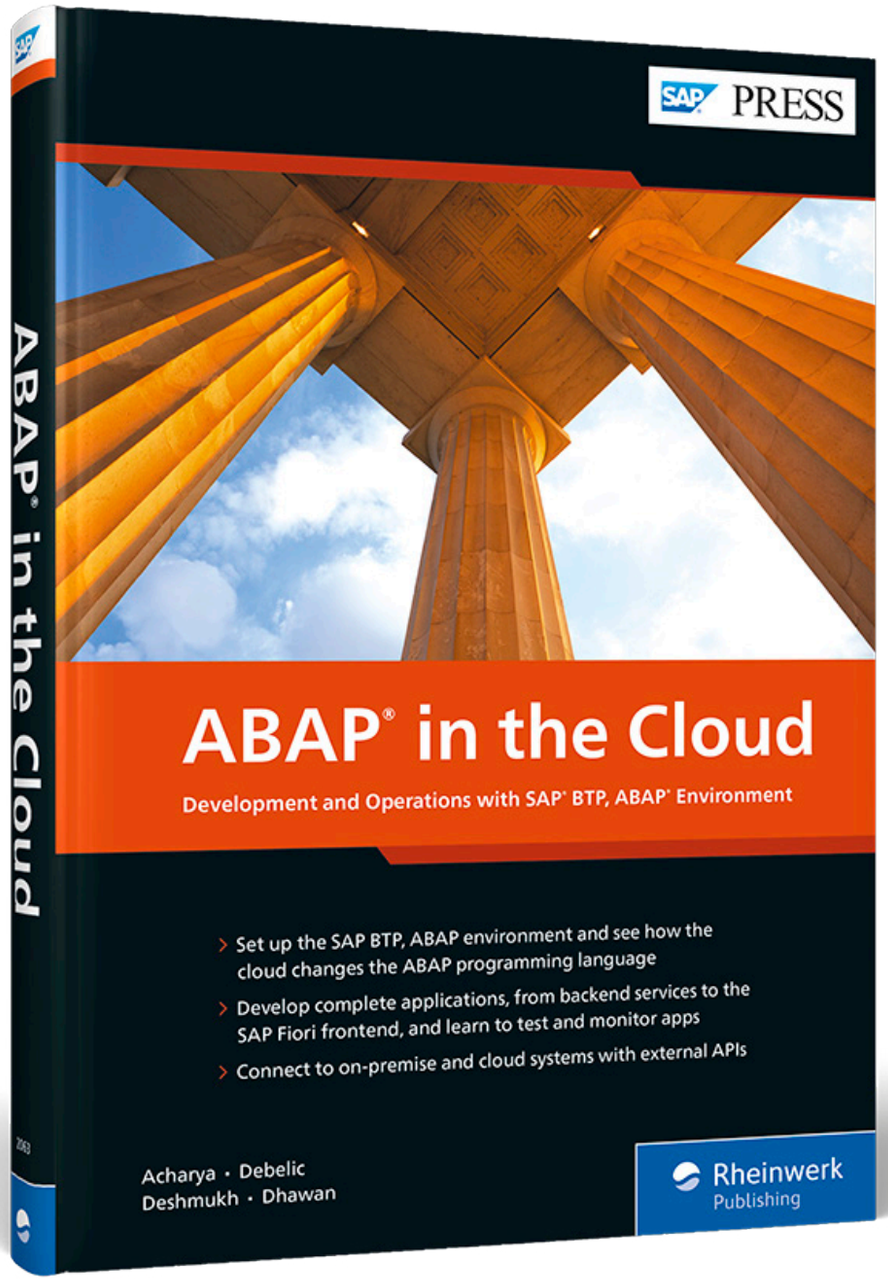


Browse the Book

This sample section details connecting your cloud development environment to an on-premise SAP system. It covers installing and configuring the cloud connector and granting RFC access in the cloud connector to the on-premise system. It also discusses defining the destinations in Cloud Foundry and testing the now-established connection to the on-premise system.

"Consuming External APIs"

Contents

Index

The Authors

Gairik Acharya, Aleksander Debelic, Shubhangi (Deshmukh) Joshi, Aayush Dhawan

ABAP in the Cloud: Development and Operations with SAP BTP, ABAP Environment

613 Pages, 2021, $79.95
ISBN 978-1-4932-2063-2

www.sap-press.com/5236

Chapter 3
Consuming External APIs

In this chapter, you'll learn why ready-to-consume business services are critical for an intelligent solution, and we'll discuss some best practices for connecting on-premise systems or cloud-based systems to business services. We'll also describe some example real-life projects to illustrate which services might be most suitable for your industry or line of business.

Traditionally, businesses have always faced make-or-buy decisions. One popular strategy adopted by successful organizations is a focus on core products that align with the organization's vision, while subcontracting non-core items. A similar concept was inherited by the services industry later, which led to a leap in outsourcing services. This approach paved way for business process outsourcing, knowledge process outsourcing, and many similar service outsourcing ventures. Extending this analogy in business solution development, application programming interfaces (APIs) are the easiest way to consume a business service without having to develop a service from scratch.

Enterprises that use SAP Business Technology Platform (SAP BTP) for ABAP development will need to open their architectures to benefit from the enormous range of APIs available from various service providers. This openness is essential so that the software solutions can be developed efficiently and in a flexible manner. More importantly, this openness reduces overall development costs and lowers the risk of failure since you'll rely on tried and tested APIs instead of custom developments.

In simple terms, an API is a software intermediary that applications (or programs) can use to communicate with each other. You can only leverage the true potential of API communication with proper connectivity and data exchange.

In this chapter, you'll learn about connecting your cloud-based ABAP applications to business systems and additional services. In Section 3.1, we'll walk you through all the steps required to connect on-premise systems, such as SAP S/4HANA or SAP ERP, to external APIs with HTTP and remote function call (RFC) connections. In Section 3.2, you'll also learn how to connect to cloud systems, like SAP S/4HANA, through SAP API Business Hub. We'll also discuss how to consume services that are already part of SAP BTP in Section 3.3 as well as how to consume several powerful, external, non-SAP cloud services in Section 3.4.

3.1 Connecting to On-Premise SAP Systems

Many SAP customers are still using on-premise SAP installations implemented long ago. To get existing business logic into an SAP cloud-based solution, you'll need to establish a connection in the cloud environment following one of two approaches: RFC destinations or HTTP destinations. These destinations must be specified properly to enable communication between cloud-based and on-premise SAP systems.

As a prerequisite, you must establish the connection to the on-premise system using SAP Connectivity service's cloud connector. The cloud connector provides secure communications, requiring only an outbound connection from the on-premise environment to SAP BTP.

> **Note**
>
> You can also use the cloud connector to connect to non-SAP systems and services on-premise.

To connect an on-premise system to the cloud, follow these steps:

1. Set up the cloud connector by installing it.
2. Configure the cloud connector.
3. Grant RFC access in the cloud connector to the on-premise SAP system.
4. For an RFC call, define the destinations in your Cloud Foundry subaccount.
5. Test the cloud connector connection to your on-premise system.

We'll describe all these steps in more detail in the following sections.

3.1.1 Configuring the HTTP and RFC Connection to an On-Premise System

In this section, we'll explain in detail the configuration process for HTTP and RFC connections to on-premise systems.

Cloud Connector Setup

First, we'll need to set up the cloud connector, which can be installed on almost any operating system, including Microsoft Windows, Linux, and macOS. The only prerequisite is that the machine on which the cloud connector is to be installed requires both access to SAP BTP and the on-premise system to which we want to connect. A step-by-step tutorial on how to install the cloud connector is available at *https://developers.sap.com/tutorials/cp-connectivity-install-cloud-connector.html*.

Now that you've installed cloud the connector, we're ready to set up access to our SAP BTP subaccount where SAP BTP, ABAP environment is running. In this stage, prepare the following information about the subaccount:

- **Region**
  The SAP BTP region where our subaccount is running.
- **Subaccount**
  The subaccount ID, which can be obtained by clicking the **Subaccount** tile under the global account and then clicking on the information button.
- **User name and password**
  In this context, the SAP BTP user name and password; this user should belong to the Cloud Foundry subaccount organization.

> **Note**
>
> Always use email IDs for Cloud Foundry subaccounts, unlike in Neo subaccounts, where the S-user ID is used.

Figure 3.1 shows what the filled-out form looks like.

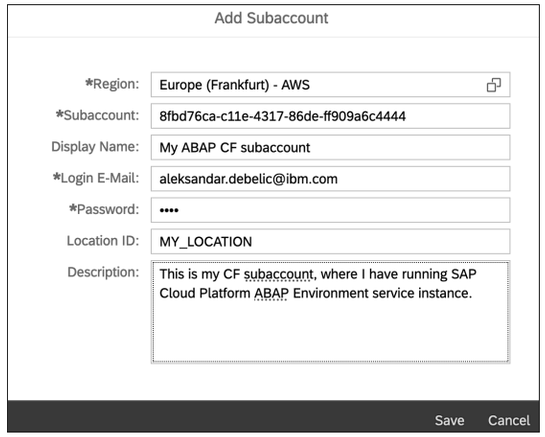


Figure 3.1 Adding a Subaccount

> **Note**
>
> We always recommend using the location ID to easily distinguish specific cloud connector instances when you have multiple cloud connector instances attached to the same SAP BTP subaccount. In fact, this field is mandatory whenever more than one cloud connector is attached to an SAP BTP subaccount.

Although the connectivity option is not visible in a trial SAP BTP subaccount, you can establish connections in a trial subaccount as well.

Now that the subaccount connection is set up, you'll need to establish a connection to the ABAP system, both via HTTP and RFC. A step-by-step guide on setting up the cloud

connector for HTTP access is available at *https://developers.sap.com/tutorials/cp-connectivity-create-secure-tunnel.html*. The process for RFC access is similar, but you'll provide an instance number that identifies the corresponding SAP Gateway service. Also, instead a URL path, you'll provide the RFC function module name/prefix.

Figure 3.2 shows several RFC and HTTP connections established with an on-premise system.

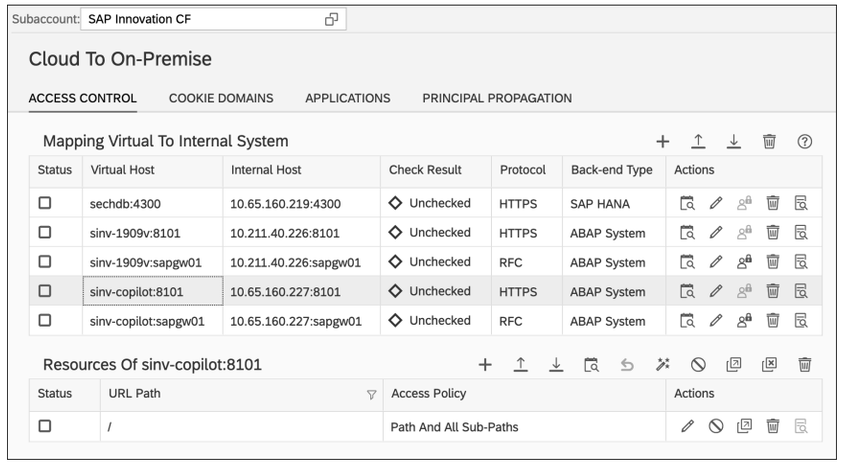


Figure 3.2 Cloud Connector Configuration Example (HTTPS)

Figure 3.3 shows a list of the Business Application Programming Interfaces (BAPIs), RFCs, and custom function modules to be called in the cloud. The objects included in this list can be consumed in the cloud.

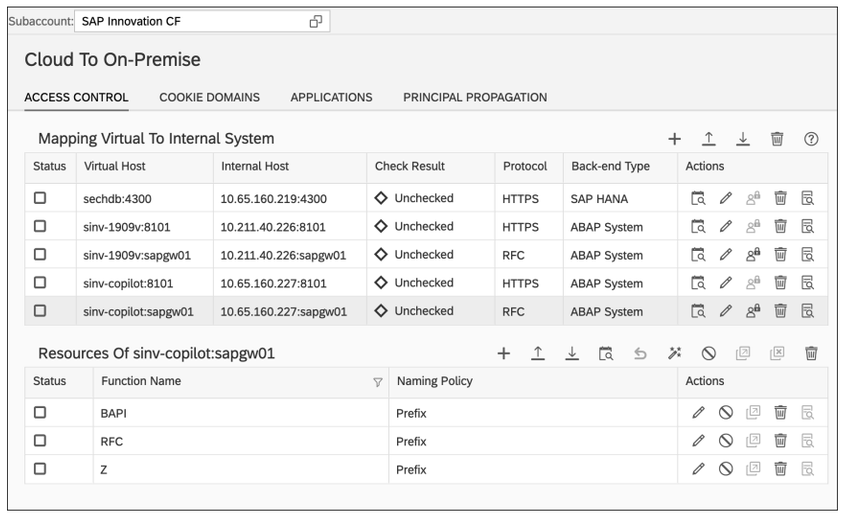


Figure 3.3 Cloud Connector Configuration Example (RFC)

Once the setup is complete on the cloud connector side, you can proceed with setting up destinations on SAP BTP. You can create a destination in one of two ways:

- At the Cloud Foundry subaccount level, under **Connectivity** menu item
- At the Cloud Foundry space level, as a destination service instance

As shown in Figure 3.4, you can navigate to destinations on the Cloud Foundry subaccount level.

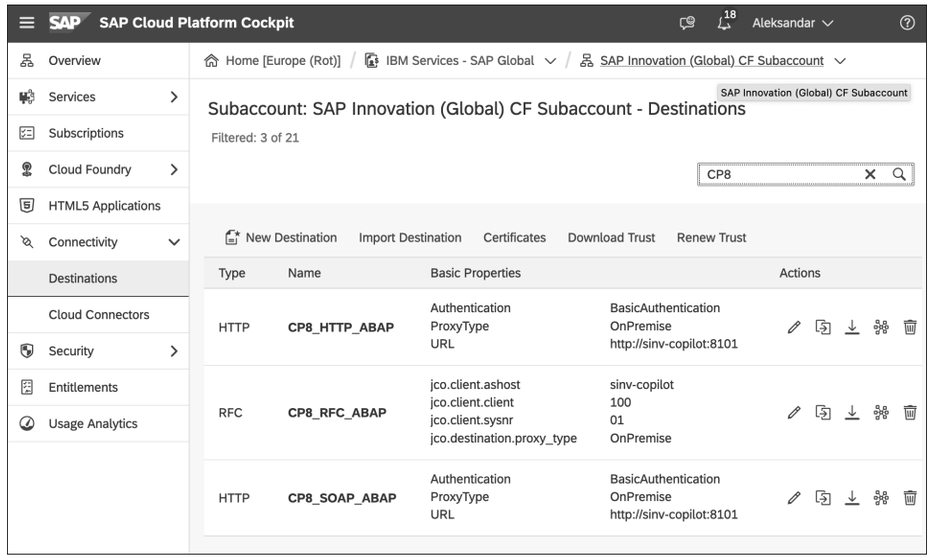


Figure 3.4 Destinations in the Cloud Foundry Subaccount

However, to access destinations defined under the destination service instance within the Cloud Foundry space, you'll need to follow these steps:

1. Access your Cloud Foundry space. This step is optional since you can select service instances directly on the Cloud Foundry subaccount level. However, with this approach, you'll get a list of all service instances in all Cloud Foundry spaces, which can sometimes be difficult to navigate.
2. Select the **Service Instances** menu item, as shown in Figure 3.5. On this screen, you can manage service instances, including creating new service instances or changing/deleting existing service instances.
3. By clicking on the selection icon (arrow pointing right), the menu on the right will open, as shown in Figure 3.6.
4. Note the text that says **More information is available for this instance**. Click on **See here**, and another browser session will open, where you can manage destinations belonging to a specific destination service, as shown in Figure 3.7.

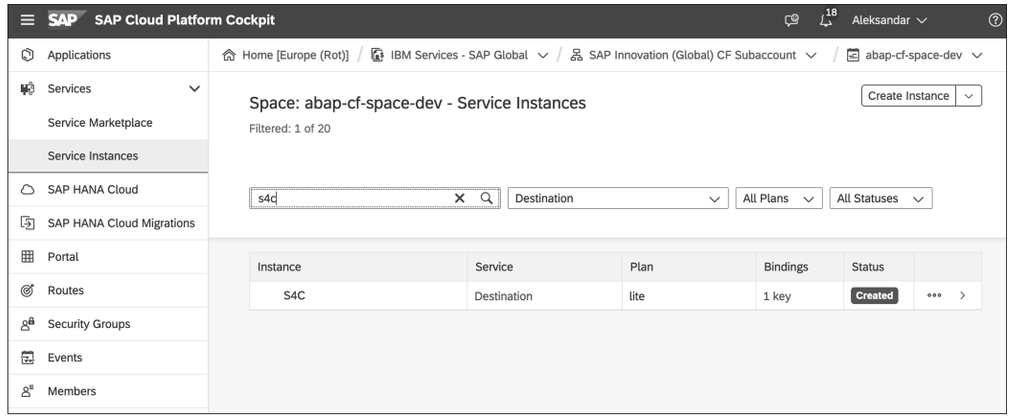


Figure 3.5 Service Instances

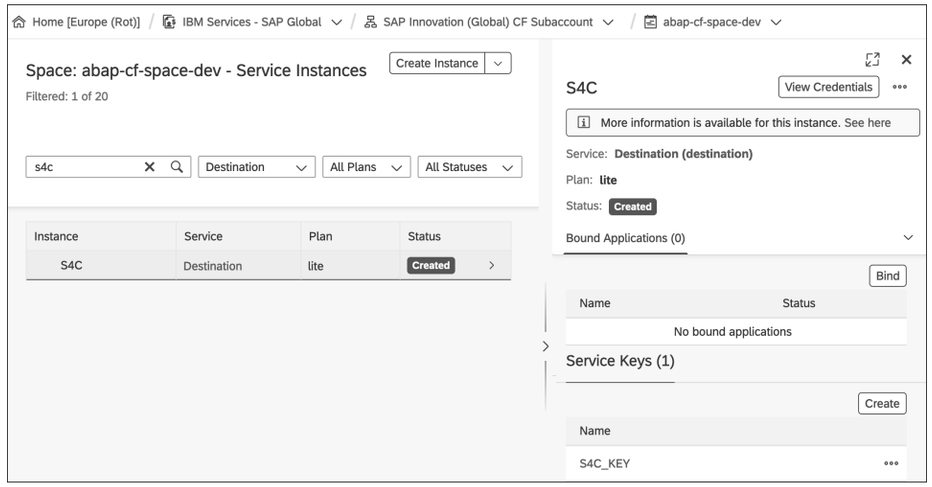


Figure 3.6 Destination Service Instance Details

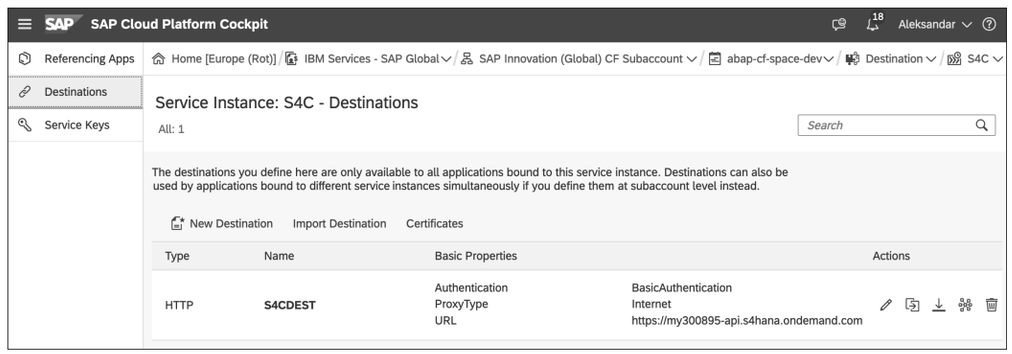


Figure 3.7 Destinations Maintenance

Now that you know how to navigate to a destination for maintenance, let's configure some destinations.

> **Note**
>
> For connections to on-premise systems, you'll use destinations defined on the Cloud Foundry subaccount level.
>
> For other types of connectivity (for example, to SAP S/4HANA Cloud), you'll use destinations defined under the destination service instance.

HTTP Destination Setup

To configure an HTTP destination in the SAP BTP cockpit, follow these steps:

1. Navigate to the relevant destination service instance, for example, the one we created in the previous step.
2. Select the **Destinations** menu item.
3. Select **New Destination**.
4. In the **Destination Configuration** section, shown in Figure 3.8, use the value help to select **HTTP** as **Type**.

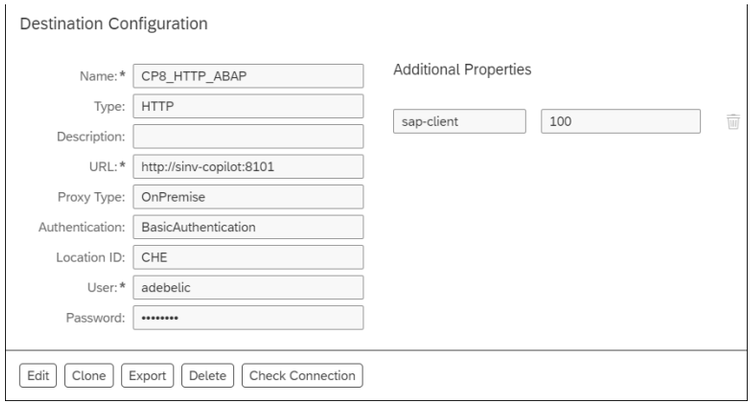


Figure 3.8 HTTP Destination

5. (Optional) If you're using more than one cloud connector in your subaccount, you must enter the location ID of the target cloud connector in the **Location ID** field.
6. For **Proxy Type**, select **OnPremise** from the value help.
7. For **Authentication**, select **BasicAuthentication** or **PrincipalPropagation**.
8. Fill in the required fields and select **Save**.
9. Open Eclipse and create and execute a runnable class.
10. To enable the HTTP communication, use, for example, the API shown in Listing 3.1.

```
DATA(lo_destination) = cl_http_destination_provider=>create_by_cloud_
destination(
 i_name = 'ERP_HTTP'
 i_authn_mode = if_a4c_cp_service=>service_specific
 ).
DATA(lo_client) = cl_web_http_client_manager=>create_by_http_destination(
lo_destination ).
DATA(lo_request) = lo_client->get_http_request( ).
DATA(lo_response) = lo_http_client->execute( i_method =
if_web_http_client=>get ).
out->write( lo_response->get_text( ) ).
```

Listing 3.1 Creating an HTTP Destination

In the code shown in Listing 3.1, the created instance `lo_destination` leverages the `cl_http_destination_provider` class. Similarly, the created instance `lo_client` leverages the `cl_web_http_client_manager` class. Then, we'll store the response from the `EXECUTE` method in the `lo_response` variable and output it.

The next step is to configure the RFC destination in the SAP BTP cockpit.

RFC Destination Setup

To configure an RFC destination in the SAP BTP cockpit, follow these steps:

1. Navigate to the relevant destination service instance.
2. Select the **Destinations** menu item.
3. Create a destination by selecting **New Destination**.
4. For **Type**, select **RFC** from the value help, as shown in Figure 3.9.

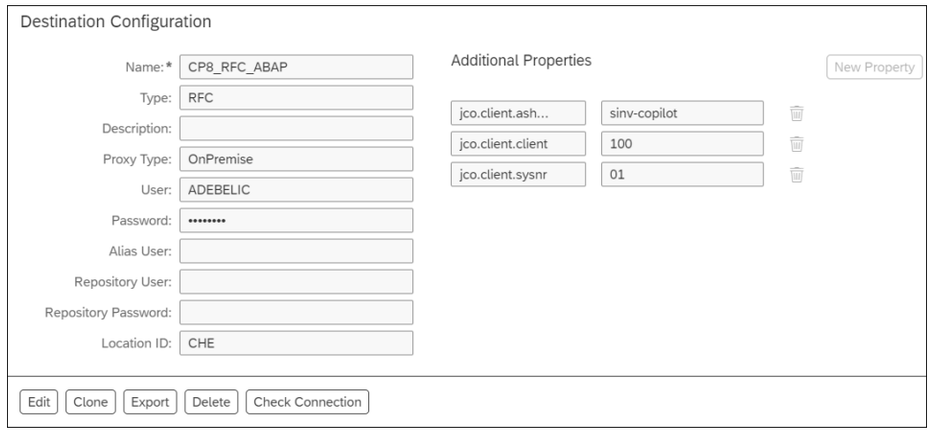


Figure 3.9 RFC Destination

5. (Optional) If you're using more than one cloud connector in your subaccount, you must enter the location ID of the target cloud connector in the **Location ID** field.
6. As authentication type, use basic authentication or set the property `jco.destination.auth_type=PrincipalPropagation`.
7. If you use basic authentication, set a user name and credentials for the destination.
8. To configure the RFC destination, choose one of the following options:
   - For a destination that uses load balancing (system ID and message server), proceed as follows:
     - Select **New Property**, choose **jco.client.r3name** from the value help, and enter the three-letter system ID of your backend system (as configured in the cloud connector) in the **property** field.
     - Create another property, select **jco.client.mshost**, and enter the message server host (as configured in the cloud connector) in the **property** field.
     - Add another property, choose **jco.client.group**, and enter a log group in the **property** field.
     - Create another property, select **jco.client.client**, and enter the three-digit ABAP client number.
   - For a destination without load balancing (application server and instance number), perform these steps:
     - Select **New Property**, choose **jco.client.ashost** from the value help, and enter the application server name of your backend system (as configured in the cloud connector) under the **Additional Properties** heading.
     - Add another property, choose **jco.client.sysnr**, and enter "00," which is the instance number of the application server (as configured in the cloud connector) under the **Additional Properties** heading.
     - Create another property, select **jco.client.client**, and enter the three-digit ABAP client number.
9. Click **Save**.

Call a remote function module with the command `CALL function 'RFC_SYSTEM_INFO' DESTINATION lv_destination`. Then, open Eclipse and create and execute a runnable class.

To execute a remote function module on your on-premise system, use the code shown in Listing 3.2.

```
DATA(lo_destination) = cl_rfc_destination_provider=>create_by_cloud_destination(
 i_name = 'ERP_RFC'
 ).
DATA(lv_destination) = lo_destination->get_destination_name( ).
DATA lv_result type c length 200.
CALL function 'RFC_SYSTEM_INFO'
```

```
DESTINATION lv_destination
 IMPORTING
 rfcsi_export = lv_result.
out->write( lv_result ).
```

Listing 3.2 Executing a Remote Function Module

As shown in Listing 3.2, you'll need to create the instance `lo_destination` leveraging the class `cl_rfc_destination_provider`. Then, you'll store the destination name in `lv_destination`. Utilize the function module '`RFC_SYSTEM_INFO`' to retrieve and output the resulting details.

3.1.2 Consuming the RFC Function Module

In this section, you'll learn in detail how to call a remote-enabled (RFC-enabled) function module from a backend system, and you'll learn in detail how to consume external APIs in the cloud in the following sections.

We'll start with creating a custom class in our package, in Eclipse. To call the function module from the backend system, first declare a destination as an RFC destination, as shown in Listing 3.3. If you're familiar with SAP ERP, you can call function modules the same way as in that system, using the **Pattern** button on the menu bar, followed by entering the destination name.

```
class ZCL_GET_CUST definition

public
 final
create public .
public section.

INTERFACES if_oo_adt_classrun.
protected section.
private section.

ENDCLASS.

CLASS ZCL_GET_CUST IMPLEMENTATION.
METHOD if_oo_adt_classrun~main.
try.
DATA(lo_rfc_dest) = cl_rfc_destination_provider=>create_by_cloud_destination(
                                i_name = |CP8_RFC_ABAP| ).
DATA(lv_destination) = lo_rfc_dest->get_destination_name( ).
          TYPES : BEGIN OF ty_cust,
                    CUSTOMERID type C LENGTH 8 ,
```



```
                    CUSTNAME type c length 25 ,
                    FORM type c length 15,
                    STREET type c length 30,
                    POBOX type c length 10,
                    POSTCODe type c length 10,
                    CITY type c length 25,
                    COUNTR type c length 3,
                    COUNTR_ISO type c length 2,
                    REGION type c length 3,
                    PHONE type c length 30,
                    EMAIL type c length 40,
          END OF ty_cust.
DATA: msg TYPE c LENGTH 255.
DATA lt_cust TYPE STANDARD TABLE OF ty_cust.
DATA ls_cust TYPE ty_cust.

CALL FUNCTION 'BAPI_FLCUST_GETLIST'
          DESTINATION lv_destination
            tables
            customer_list  = lt_cust.
   CASE sy-subrc.
      WHEN 0.
        LOOP AT lt_cust INTO ls_cust.
         out->write( ls_cust-customerid && ',' &&  ls_cust-custname && ','
&& ls_cust-street && ',' &&  ls_cust-postcode && ',' && ls_cust-city ).
        ENDLOOP.
      WHEN 1.
        out->write ( |EXCEPTION SYSTEM_FAILURE | && msg ).
      WHEN 2.
        out->write ( |EXCEPTION COMMUNICATION_FAILURE | && msg ).
      WHEN 3.
        out->write( |EXCEPTION OTHERS| ).
    ENDCASE.

CATCH cx_root INTO DATA(lx_root).
          out->write( lx_root->get_text( ) ).
endtry.

endmethod.
ENDCLASS.
```

Listing 3.3 Consuming an RFC Function Module

As shown in Listing 3.3, you'll need to define the class ZCL_GET_CUST. Make sure you define the interface within the PUBLIC SECTION. Then, you'll need to write the class implementation. In the main method, you'll retrieve the destination name in lv_destination and then call/consume RFC 'BAPI_FLCUST_GETLIST' from that destination. This method will display a list of customers as the output, as shown in Figure 3.10.

```
00004668,Matthias Kramer,Gartenstr. 79,69123,Heidelberg
00004669,Guenther Simonen,Jacobistrasse 219,79104,Freiburg
00004670,Adam Neubasler,Heidelberger Str. 87,69483,Wald-Michelbach
00004671,Stephen Kreiss,Arionweg 30,68723,Schwetzingen
00004672,Kurt Schneider,Waldmann 185,69207,Kurt
00004673,Annemarie Barth,Stauboernchenstrasse 227,67663,Kaiserslautern
00004674,Ulla Marshall,Gruenlingweg 133,69180,Wiesloch
00004675,Andrej Eichbaum,43 Poklukarjeva,1000,Ljubljana
00004676,Ruth Kreiss,Melissenstr. 190,41466,Neuss
00004677,Johann Koller,Ausfallstr. 57,11111,Berlin
00004678,Amelie Deichgraeber,Froschstr. 61,68753,Amelie
00004679,Irmtraut Sudhoff,Max-Planck-Str. 213,63150,Heusenstamm
00004680,Allen Kreiss,6 Sagamore St.,17758,N. Massapequa
00004681,Holm Heller,Muehltalstr. 221,69121,Heidelberg
00004682,Roland Goelke,Gemeindestr. 247,79761,Waldshut
00004683,Achim D´Oultrement,Rankestr. 54,76137,Karlsruhe
00004684,Irmtraut Benjamin,Max-Planck-Str. 15,63150,Heusenstamm
00004685,Anna Detemple,Lerchenstr. 248,86343,Koenigsbrunn
00004686,Anna Buehler,Lerchenstr. 102,86343,Koenigsbrunn
00004687,Laura Ryan,Raupelsweg 27,60118,Mainz
```

Figure 3.10 Output

3.1.3 Consuming the OData Service from the Backend

The Open Data Protocol (OData) was introduced by Microsoft. Later, this protocol became the ISO/IEC-approved OASIS standard, which contains best practices for building and consuming RESTful APIs. You can find more details at *https://www.odata.org/*.

Assuming that an OData service has been created in your backend system (by creating a project through Transaction SEGW or simply added via Transaction /IWFND/MAINT_SERVICE), we'll show you how to consume an OData service from the backend into the cloud through the service consumption model (see Figure 3.11). This model enables communication based on OData and Simple Object Access Protocol (SOAP) client calls to achieve a higher level of abstraction than when using HTTP and RFC while also providing enhanced interfaces with the ABAP RESTful application programming model. To begin, follow these steps:

1. **Create a service consumption model for the OData call**
   Figure 3.11 shows the inputs required for creating a new service consumption model. You'll need to select the package and project where this consumption model is to be stored. The name of the model should follow the project's naming conventions. The **Description** field is a free text field for better documentation.

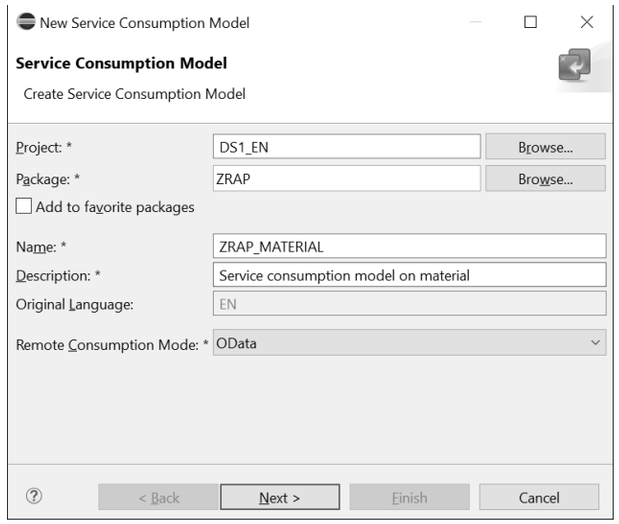


Figure 3.11 Service Consumption Model

2. **Get the service definition in XML format ($metadata file)**
   The *$metadata* file, shown in Figure 3.12, specifies the data that will be exposed from the backend system. The service definition can be obtained from a service provider, such as an SAP Gateway system or an SAP S/4HANA system, or from a public services inventory, such as the SAP API Business Hub, which lists all published APIs.

   This metadata file can be downloaded from an on-premise system and saved to a local system by executing Transaction SEGW.

```
<?xml version="1.0" encoding="UTF-8"?>
- <edmx:Edmx xmlns:sap="http://www.sap.com/Protocols/SAPData" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">
  - <edmx:Reference xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Uri="http://sinv-
    copilot.isl.edst.ibm.com:8001/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Vocabularies(TechnicalName='%2FIWBEP%
    2FVOC_COMMON',Version='0001',SAP__Origin='LOCAL')/$value">
      <edmx:Include Alias="Common" Namespace="com.sap.vocabularies.Common.v1"/>
    </edmx:Reference>
  - <edmx:Reference xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Uri="http://sinv-
    copilot.isl.edst.ibm.com:8001/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Vocabularies(TechnicalName='%2FIWBEP%
    2FVOC_CAPABILITIES',Version='0001',SAP__Origin='LOCAL')/$value">
      <edmx:Include Alias="Capabilities" Namespace="Org.OData.Capabilities.V1"/>
    </edmx:Reference>
  - <edmx:Reference xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Uri="http://sinv-
    copilot.isl.edst.ibm.com:8001/sap/opu/odata/IWFND/CATALOGSERVICE;v=2/Vocabularies(TechnicalName='%2FIWBEP%
    2FVOC_COMMUNICATION',Version='0001',SAP__Origin='LOCAL')/$value">
      <edmx:Include Alias="Communication" Namespace="com.sap.vocabularies.Communication.v1"/>
    </edmx:Reference>
  - <edmx:DataServices m:DataServiceVersion="2.0">
    - <Schema xml:lang="en" Namespace="API_MATERIAL_DOCUMENT_SRV" xmlns="http://schemas.microsoft.com/ado/2008/09/edm" sap:schema-
      version="1">
      - <EntityType sap:content-version="1" sap:label="API exposure Material Document Header" Name="A_MaterialDocumentHeaderType">
        - <Key>
            <PropertyRef Name="MaterialDocumentYear"/>
            <PropertyRef Name="MaterialDocument"/>
          </Key>
          <Property sap:label="Material Document Year" Name="MaterialDocumentYear" sap:display-format="NonNegative" MaxLength="4" Nullable="false"
            Type="Edm.String"/>
          <Property sap:label="Material Document" Name="MaterialDocument" sap:display-format="UpperCase" MaxLength="10" Nullable="false"
```

Figure 3.12 Metadata File

In the service consumption model, you'll need an input file to generate an OData consumption proxy. Figure 3.13 shows how you can select the metadata file, stored in your local drive in the previous step, as the service metadata file.

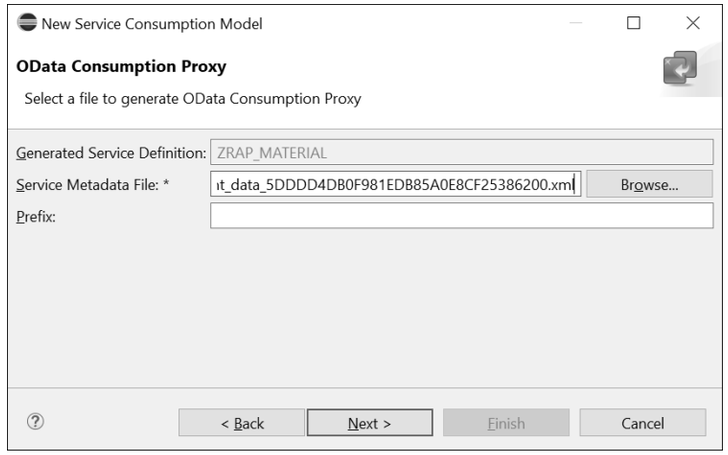


Figure 3.13 OData Service Call

In this step, as shown in Figure 3.13, you can also use a prefix if you plan to import the file several times. This option will help you prevent the framework from automatically creating repository object names when importing the same files (in different clients).

3. **Specify the entity sets to be included with the generation of the service consumption model**
   Figure 3.14 shows how to specify the entity sets to be included while creating the service consumption model.

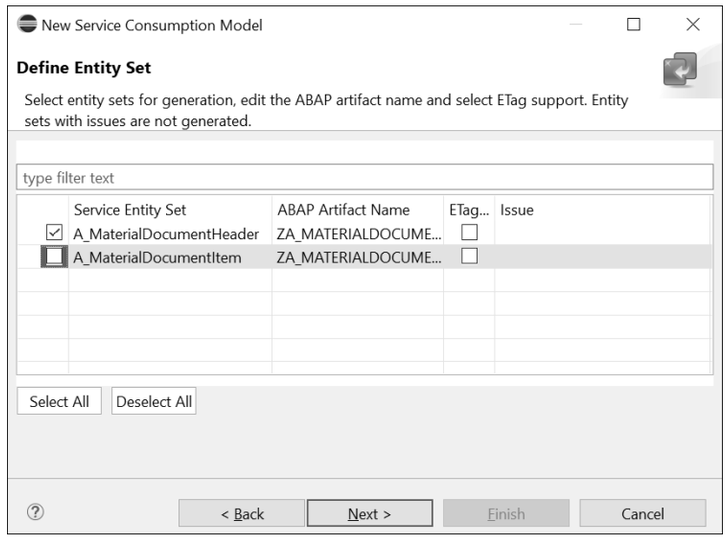


Figure 3.14 Entity Set

Service definitions, abstract entity data definitions, and behavior definitions will be created by a wizard, as shown in Figure 3.15.

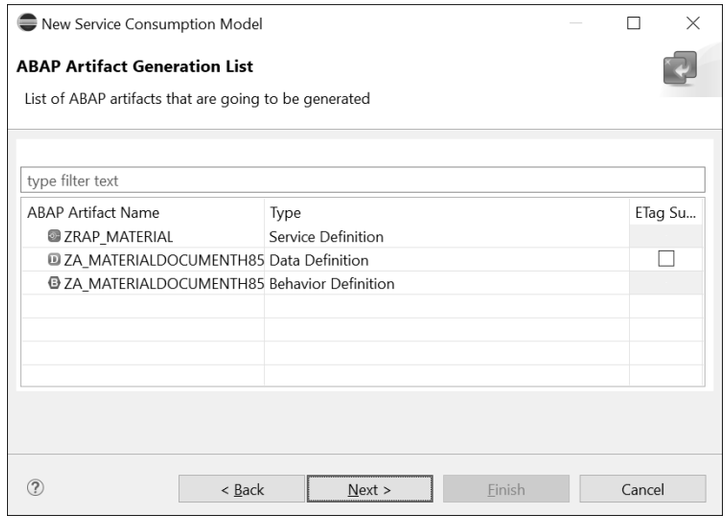


Figure 3.15 ABAP Artifact

Listing 3.4 shows the detailed code that is generated by the wizard tool as part of a service definition.

```
/*Service definition generated by wizard tool*/
@EndUserText.label: 'ZRAP_MATERIAL'
@OData.schema.name: 'API_MATERIAL_DOCUMENT_SRV'
define service ZRAP_MATERIAL {
 expose ZA_MATERIALDOCUMENTH856A03A9E0;
 }

/********** GENERATED on 10/24/
2020 at 09:50:04 by CB0000000002*************/
 @OData.entitySet.name: 'A_MaterialDocumentHeader'
 @OData.entityType.name: 'A_MaterialDocumentHeaderType'
 define root abstract entity ZA_MATERIALDOCUMENTH856A03A9E0 {
 key MaterialDocumentYear : abap.numc( 4 ) ;
 key MaterialDocument : abap.char( 10 ) ;
 @Odata.property.valueControl: 'InventoryTransactionType_vc'
 InventoryTransactionType : abap.char( 2 ) ;
 InventoryTransactionType_vc : RAP_CP_ODATA_VALUE_CONTROL ;
 @Odata.property.valueControl: 'DocumentDate_vc'
 DocumentDate : RAP_CP_ODATA_V2_EDM_DATETIME ;
 DocumentDate_vc : RAP_CP_ODATA_VALUE_CONTROL ;
 @Odata.property.valueControl: 'PostingDate_vc'
```

```
PostingDate : RAP_CP_ODATA_V2_EDM_DATETIME ;
PostingDate_vc : RAP_CP_ODATA_VALUE_CONTROL ;
@Odata.property.valueControl: 'CreationDate_vc'
CreationDate : RAP_CP_ODATA_V2_EDM_DATETIME ;
CreationDate_vc : RAP_CP_ODATA_VALUE_CONTROL ;
@Odata.property.valueControl: 'CreationTime_vc'
CreationTime : RAP_CP_ODATA_V2_EDM_TIME ;
CreationTime_vc : RAP_CP_ODATA_VALUE_CONTROL ;
@Odata.property.valueControl: 'CreatedByUser_vc'
CreatedByUser : abap.char( 12 ) ;
CreatedByUser_vc : RAP_CP_ODATA_VALUE_CONTROL ;
@Odata.property.valueControl: 'MaterialDocumentHeaderText_vc'
MaterialDocumentHeaderText : abap.char( 25 ) ;
MaterialDocumentHeaderText_vc : RAP_CP_ODATA_VALUE_CONTROL ;
@Odata.property.valueControl: 'ReferenceDocument_vc'
ReferenceDocument : abap.char( 16 ) ;
ReferenceDocument_vc : RAP_CP_ODATA_VALUE_CONTROL ;
@Odata.property.valueControl: 'GoodsMovementCode_vc'
GoodsMovementCode : abap.char( 2 ) ;
GoodsMovementCode_vc : RAP_CP_ODATA_VALUE_CONTROL ;
}
```

Listing 3.4 Auto-Generated Service Definition from the Wizard

The results screen of the service consumption model, shown in Figure 3.16, shows an automatically generated code template that can read the entity set.

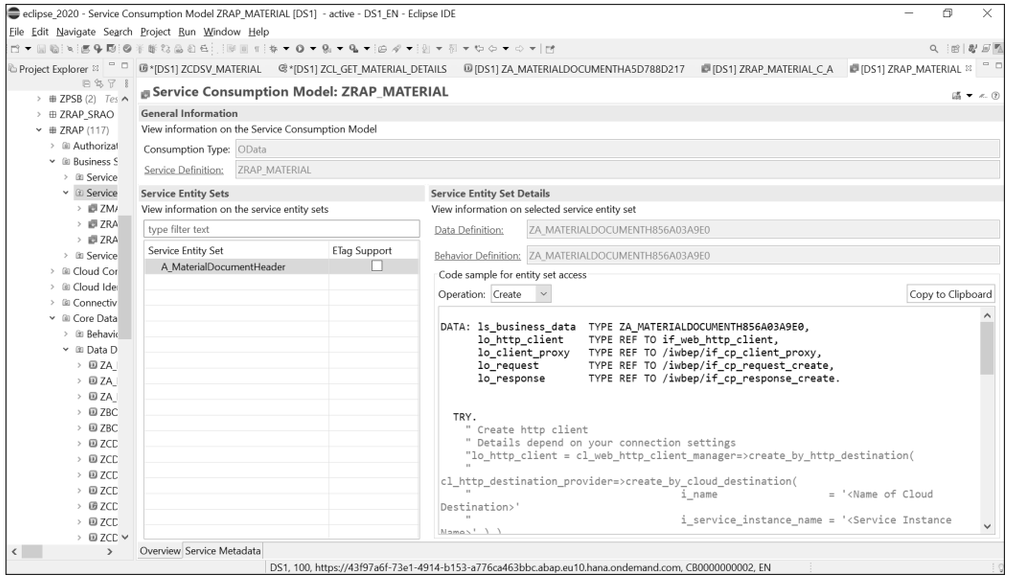


Figure 3.16 Service Consumption Model

4. **Create a custom class to embed this template**
   The class shown in Listing 3.5 will produce the console output and list all the materials available from the backend SAP Gateway system.


```
class ZCL_MATERIAL_ODATA_CALL definition
  public
  final
  create public .

public section.
interfaces IF_OO_ADT_CLASSRUN.
protected section.
private section.
ENDCLASS.

CLASS ZCL_MATERIAL_ODATA_CALL IMPLEMENTATION.

method if_oo_adt_classrun~main.
DATA: lt_business_data  TYPE TABLE OF ZA_MATERIALDOCUMENTH856A03A9E0,
      lo_http_client    TYPE REF TO if_web_http_client,
      lo_client_proxy   TYPE REF TO /iwbep/if_cp_client_proxy,
      lo_request        TYPE REF TO /iwbep/if_cp_request_read_list,
      lo_response       TYPE REF TO /iwbep/if_cp_response_read_lst.
  TRY.
  DATA(http_destination) = cl_http_destination_provider=>create_by_cloud_
destination(
        i_name = 'CP8_HTTP_ABAP'
        i_authn_mode = if_a4c_cp_service=>service_specific
        ).
       lo_http_client = cl_web_http_client_manager=>create_by_http_
destination( http_destination ).
    lo_client_proxy = cl_web_odata_client_factory=>create_v2_remote_proxy(
      EXPORTING
        iv_service_definition_name = 'ZRAP_MATERIAL'
        io_http_client             = lo_http_client
        iv_relative_service_root   = '/sap/opu/odata/sap/API_MATERIAL_
DOCUMENT_SRV' ).
    " Navigate to the resource and create a request for the read operation
    lo_request = lo_client_proxy->create_resource_for_entity_set(
'A_MATERIALDOCUMENTHEADER' )->create_request_for_read( ).
    lo_request->set_top( 1 )->set_skip( 0 ).
    " Execute the request and retrieve the business data
    lo_response = lo_request->execute( ).
```

```
    lo_response->get_business_data( IMPORTING et_business_data = lt_business_
data ).

    CATCH cx_web_http_client_error INTO DATA(lx_response).
        out->write( lx_response->get_text(  ) ).

      CATCH cx_http_dest_provider_error INTO DATA(lx_destexception).
        out->write( lx_destexception->get_text(  ) ).

  CATCH /iwbep/cx_cp_remote INTO DATA(lx_remote).
    " Handle remote Exception
    " It contains details about the problems of your http(s) connection

  CATCH /iwbep/cx_gateway INTO DATA(lx_gateway).
    " Handle Exception
ENDTRY.
ENDMETHOD .
ENDCLASS.
```

Listing 3.5 Custom Class for the OData Call

As shown in Listing 3.5, you'll need to define the class `ZCL_MATERIAL_ODATA_CALL`. Make sure you define the interface within the `public` section. Then, you'll need to write the class implementation. In the main method, you'll need to retrieve destination using `cl_http_destination_provider` and then consume OData services from that destination. This example will retrieve material document information.

3.1.4 Consuming the SOAP Web Service

The Simple Object Access Protocol (SOAP) is a messaging protocol that serves as the foundation for the implementation of web services and has been around for more than two decades. Its first release was back in 1998, when it was called "XML-RPC."

SAP introduced support for SOAP web services in 2002, with SAP NetWeaver 6.20 (then called the SAP Web Application Server), which was foundation for SAP R/3 Enterprise 4.70. SOAP web services are supported both in SAP S/4HANA and in all SAP ERP releases, unlike OData, which is not supported in older SAP ERP systems.

To consume SOAP web services from SAP BTP, ABAP environment, you must fulfill two prerequisites:

- **Configuration in the cloud connector**
  An HTTP connection to the on-premise system.
- **Web service configuration**
  A web service configured and exposed via Transaction SOAMANAGER.

For this example, we'll use the *Business User – Read* web service. For more details on this service, refer to *https://api.sap.com/api/QUERYBUSINESSUSERIN/documentation*.

First, download the Web Service Definition Language (WSDL) file via Transaction SOAMANAGER. Locate the web service QUERYBUSINESSUSERIN and open the **Configuration** tab for the service definition, as shown in Figure 3.17.

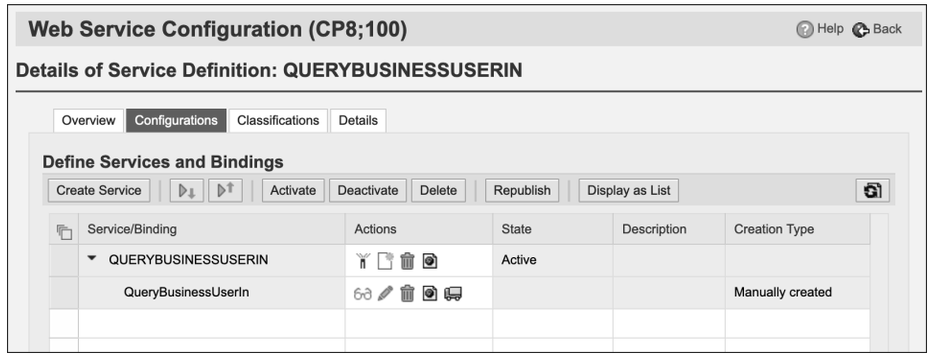


Figure 3.17 Web Service Configuration

By selecting the **Open Binding WSDL Generation** option, a popup window will open. As shown at the bottom of Figure 3.18, locate and copy the WSDL URL. Alternatively, you can simply open the WSDL document directly.

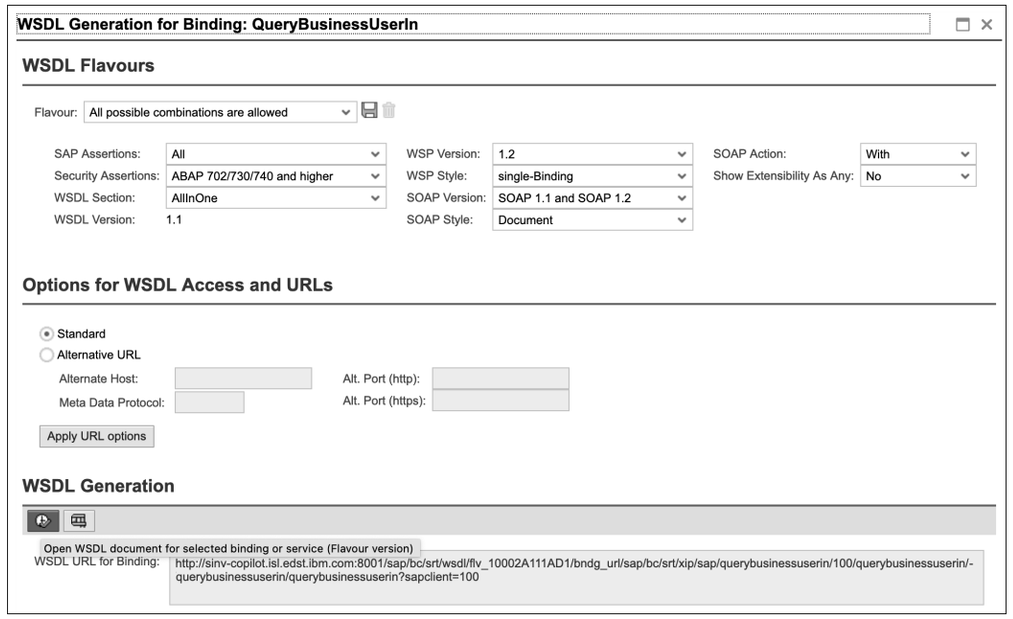


Figure 3.18 Add WSDL URL

The SAP system will ask you to authenticate. After successful authentication, you'll see the WSDL document in the web browser, as shown in Figure 3.19.

Figure 3.19 WSDL Document

You'll need to download the WSDL document as a text file by selecting the **View Source** option in the web browser, which will display the unformatted version of the WSDL. (This option may be labeled differently depending on the web browser being used.) Copy the content of the web browser window into a text editor and adjust the host name and port so that these details correspond to the cloud connector connection definition. The host name and port information can be located in SAP BTP, as shown in Figure 3.20.

Having completed these steps, the WSDL file is ready.

> **Note**
>
> Make sure that the content is pasted and saved as a plain text file, without any formatting.

Now, moving to Eclipse, we'll create the service consumption model. Right-click on the package where you want to create service consumption model and, in context menu, select **New • Other ABAP Repository Object**. On the next screen, select the **Service Consumption Model** option and click **Next**, as shown in Figure 3.21.

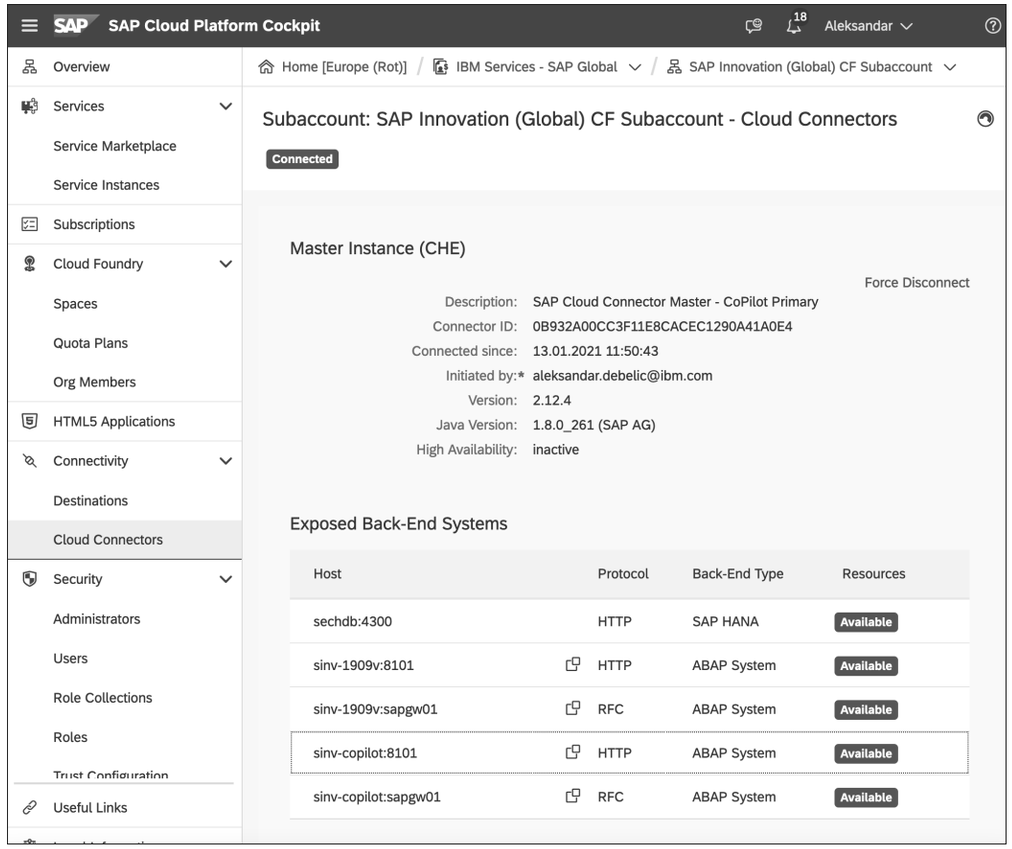


Figure 3.20 Cloud Connector Configuration

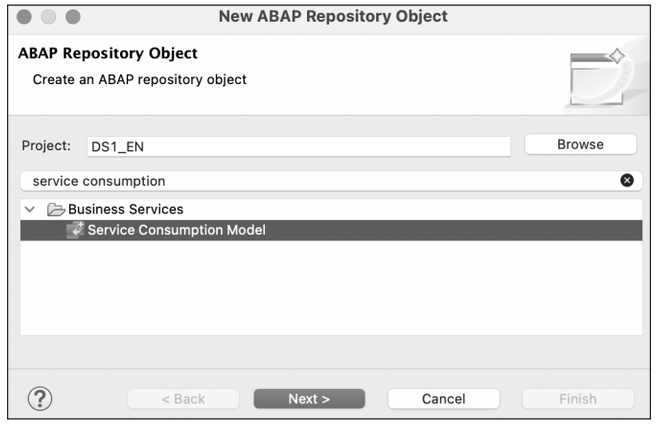


Figure 3.21 ABAP Repository Object: Service Consumption Model

On the next screen, enter a name and a description. From the **Remote Consumption Mode** dropdown list, select **Web Service**, as shown in Figure 3.22.

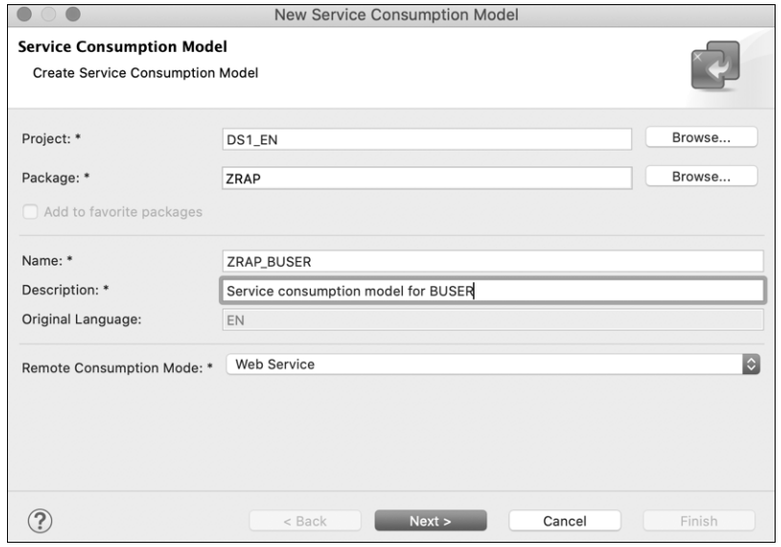


Figure 3.22 Creating a Service Consumption Model

On the next screen, shown in Figure 3.23, provide the WSDL file and define the prefix.

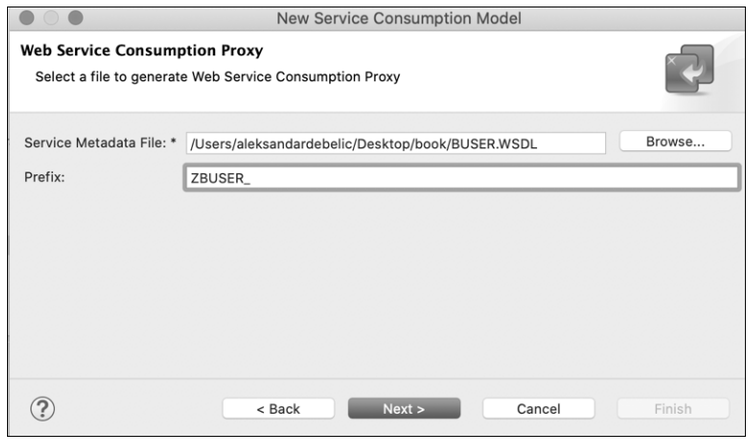


Figure 3.23 Creating a Service Consumption Model, Continued

These changes can be stored in a transport request. If a transport request already exists for this project, you can utilize that transport request. If no transport request is available, then you must create a new transport request.

The service consumption model has now been created, and now, all you need to do is activate it by clicking on the **Activate** button on the menu bar. Now, you'll be presented

with the service consumption model maintenance screen, where you'll see properties and metadata as well as a code sample that you can reuse in your development projects, as shown in Figure 3.24.

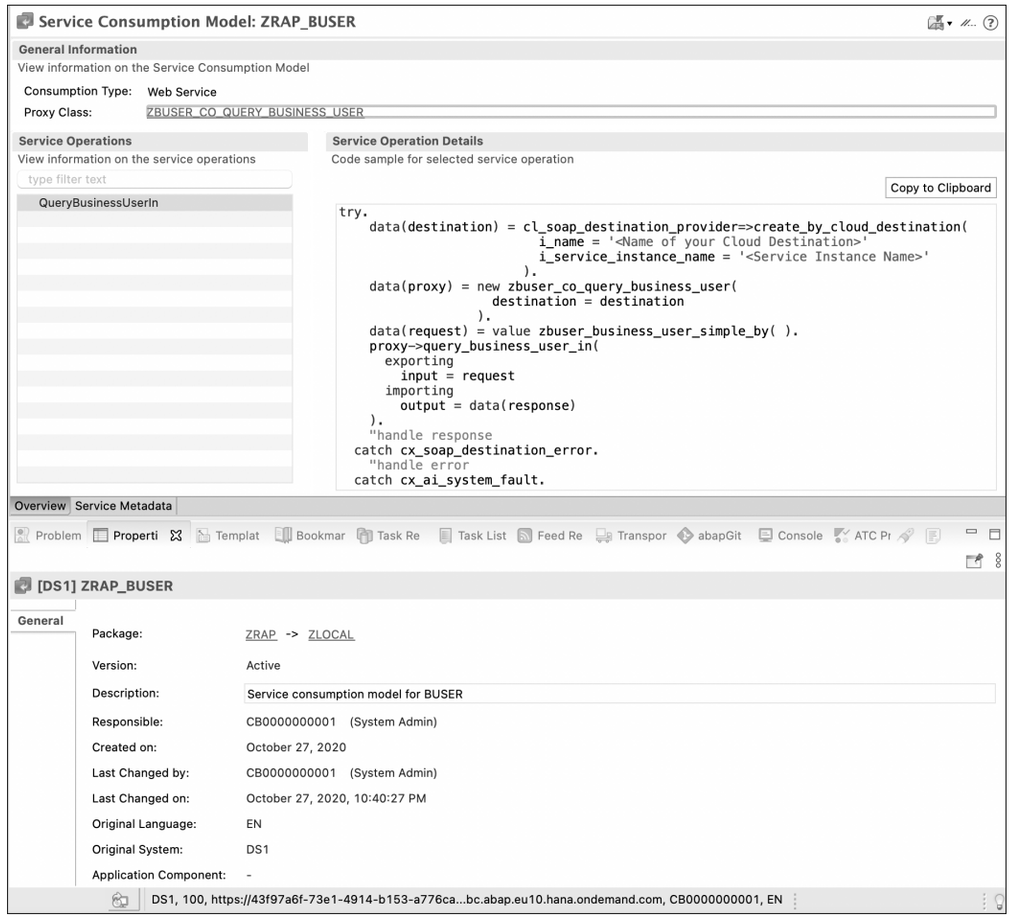


Figure 3.24 Service Consumption Model Maintenance Screen

Alongside the service consumption model, you'll notice that helper classes have been created as well. These proxy classes, shown in Figure 3.25, are required for SOAP service consumption.

Figure 3.25 Generated Proxy Classes

Your new SOAP service is ready for the consumption. Simply use the sample code available in the service consumption model to accelerate development.

3.1.5 Inbound RFC Concepts

Inbound RFC support was introduced in SAP BTP, ABAP environment release 2008. In this section, we'll walk through the process flow to help you understand the prerequisites and required steps to enable external systems to call RFC modules developed in SAP BTP, ABAP environment.

Figure 3.26 shows the inbound RFC architecture, outlining how to configure inbound connectivity to the ABAP environment and to consume a remote-enabled function module from an on-premise ABAP system through the cloud connector. RFC connections from on-premise systems to the ABAP environment can be performed through the cloud connector service channel. From SAP S/4HANA 2019 onwards, WebSocket RFCs, which do not need the cloud connector because they can be opened directly on the Internet, can be used.

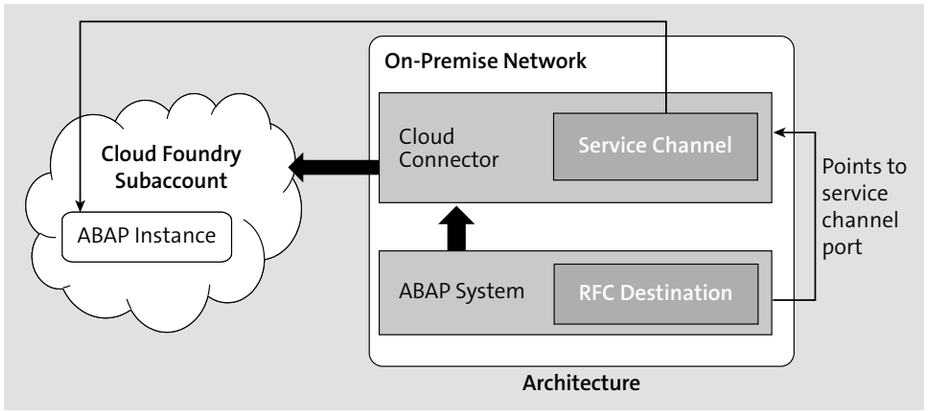


Figure 3.26 Inbound RFC Architecture

Let's now dive more deeply into the technical steps involved in the implementation of the inbound RFC concept. To establish a connection from an on-premise system to the cloud, you must have a service channel in place. Then, you'll need to create a remote-enabled function module. You'll also need a communication scenario, a proper communication arrangement, and a RFC destination already set up. With all these elements in place, finally, you can call the function module to receive the desired output.

Adding a Service Channel

In addition to the standard configuration in the cloud connector for HTTP and RFC connections from SAP BTP to an on-premise system, you must add a service channel with a local instance number. In an on-premise ABAP system, you would thus create an RFC destination of type 3 through Transaction SM59 with the host name of the cloud connector in your on-premise network and the same instance number. The load balancing status should be **NO**.

Now, let's add a service channel to establish the connection from the on-premise system to the cloud. From your subaccount menu, select **On Premise To Cloud**. Then, click the **Add** (**+**) icon, which will open the screen shown in Figure 3.27.

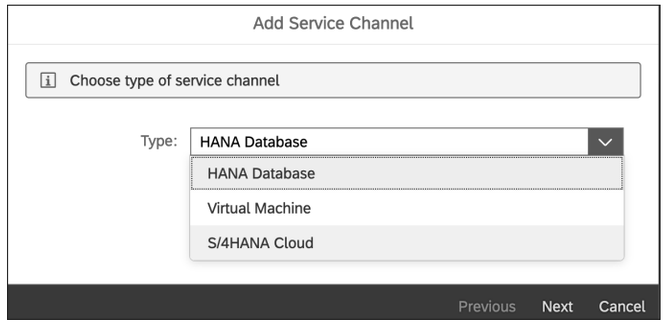


Figure 3.27 Service Channel

On this screen, select **S/4HANA Cloud** from the **Type** dropdown list.

Next, provide the cloud tenant host, local instance number, and connections. Then, select the **Enabled** checkbox, as shown in Figure 3.28. The cloud tenant host information can be obtained from the service dashboard.

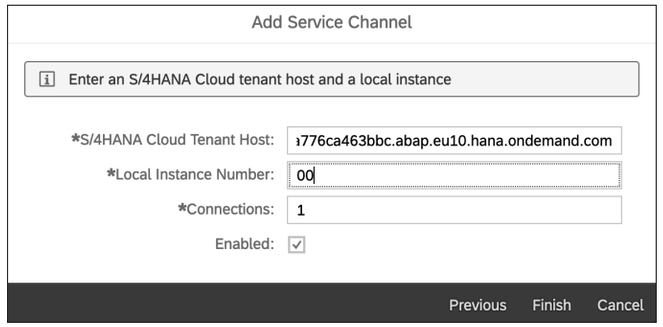


Figure 3.28 Service Channel Details

> **Note**
>
> You can assign a random instance number, and this information will later be used in RFC destination.

This step completes the configuration of the cloud connector. The result looks like the screen shown in Figure 3.29.

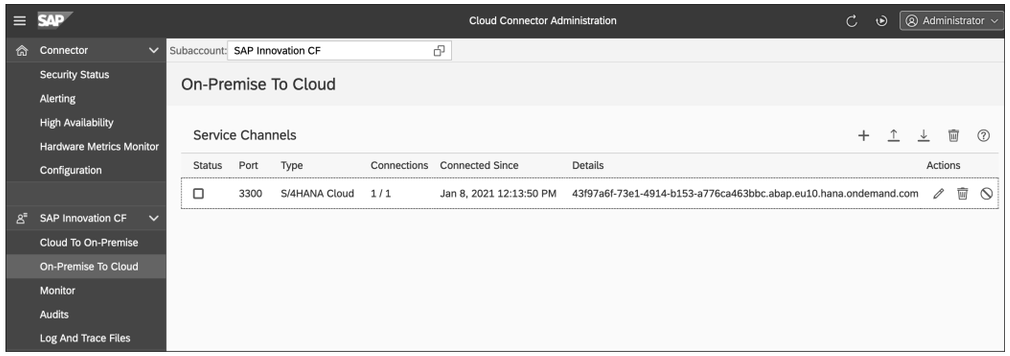


Figure 3.29 Cloud Connector Configuration

A green indicator under **Status** means that the connection with SAP BTP, ABAP environment has been established and the service channel is ready for use.

Creating a Remote-Enabled Function Module

Now, let's use SAP BTP, ABAP environment to create a remote-enabled function module and the communication scenario that will be used for its consumption.

First, create the remote-enabled function module in ABAP Development Tools (ADT) using the code shown in Listing 3.6.

```
FUNCTION ZFM_GET_SO
  IMPORTING
    VALUE(LV_KG) TYPE ZKG
  EXPORTING
    VALUE(LV_POUNDS) TYPE ZLB.

DATA : LV_CON TYPE ZKG.
DATA : GT_LIKP TYPE TABLE OF ZLIKP,
       GS_LIKP TYPE ZLIKP,
       lv_cnt type N,
       CNT TYPE I,
       msg type string .

       LV_CON = '2.204' .
       LV_POUNDS = LV_CON * LV_KG.

ENDFUNCTION.
```

Listing 3.6 Remote-Enabled Function Module Example

As shown in Listing 3.6, this sample code creates a remote-enabled function module in ADT. In our example, for simplicity, the logic converts kilograms to pounds, but any business logic can be similarly embedded.

Figure 3.30 shows in detail how to update the properties of a remote-enabled function module. Make sure you select the **RFC** processing type option from the **Processing Type** dropdown list to make the function module remote-enabled

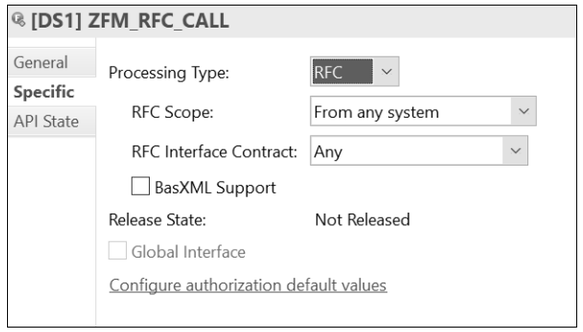


Figure 3.30 Remote Function Module Properties

Creating a Communication Scenario

Now, let's proceed with creating a communication scenario. In ADT, right-click on desired package and select **New Communication Scenario**, as shown in Figure 3.31.

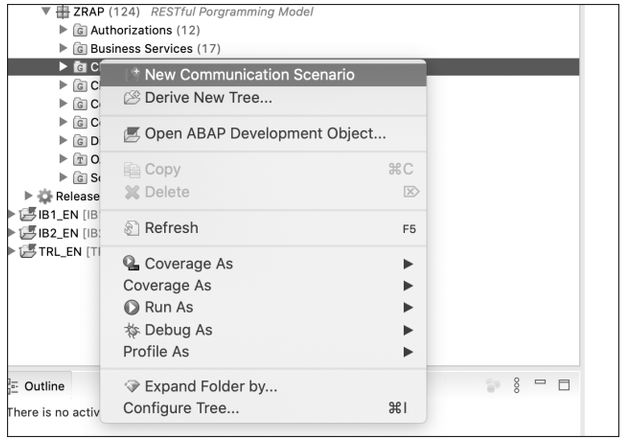


Figure 3.31 Communication Scenario

As shown in Figure 3.32, you'll create the communication scenario in ADT.

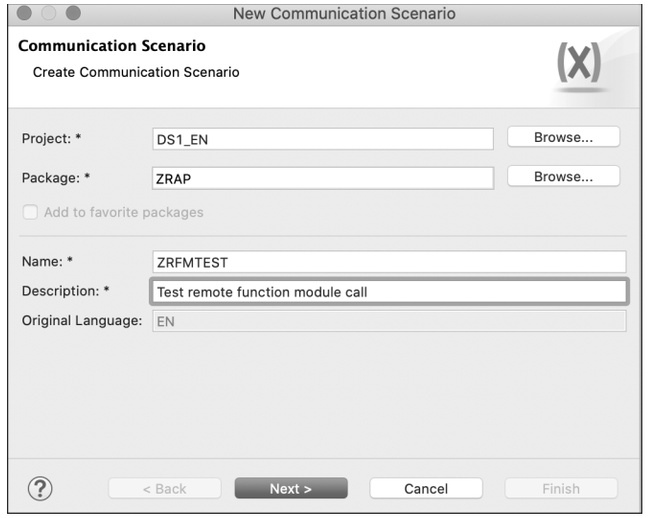


Figure 3.32 Creating a Communication Scenario

Once it's created, on the **Overview** tab you will find the dropdown list for **Allowed Instances**. Choose the **One instance per scenario & communication system** option, as shown in Figure 3.33.

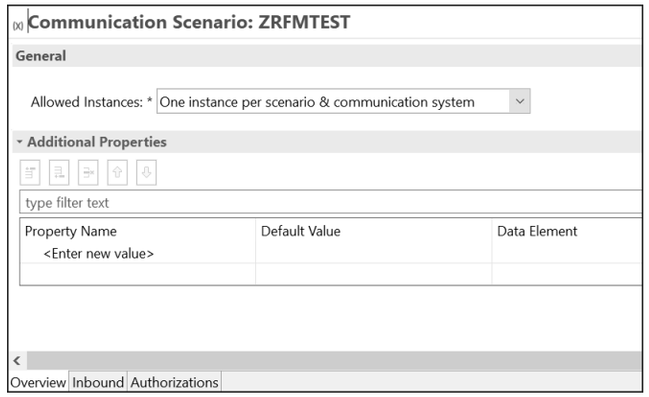


Figure 3.33 Communication Scenario Overview Tab

Select the **Inbound** tab and assign the basic authentication method by selecting the **Basic** checkbox in the **Inbound Settings** section, as shown in Figure 3.34.

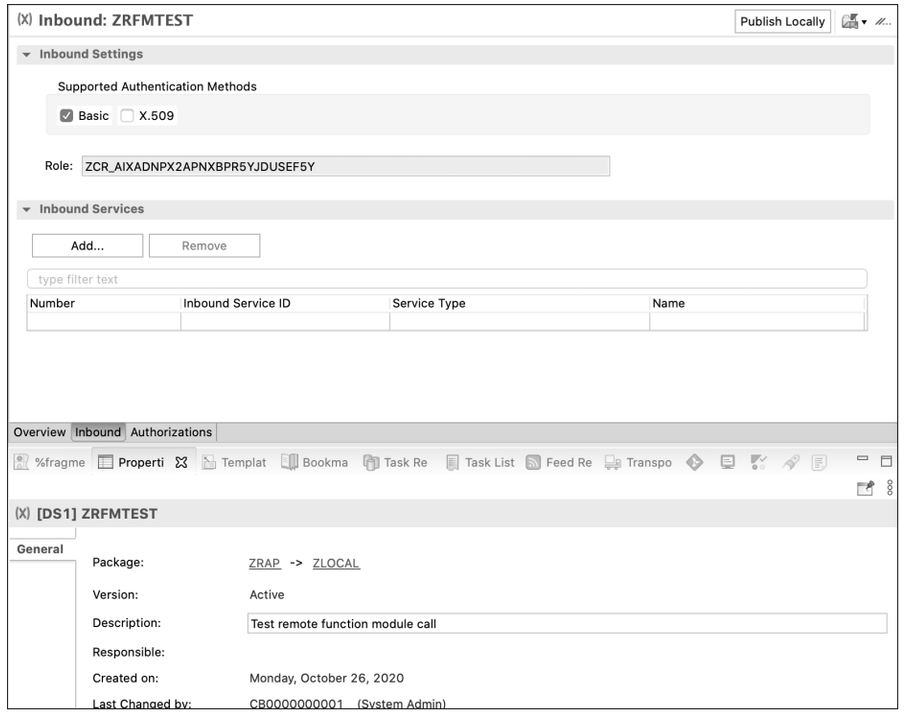


Figure 3.34 Authentication Method Assignment

Now, add the inbound services that you want to include in this communication scenario in the **Inbound Service ID** field, as shown in Figure 3.35.

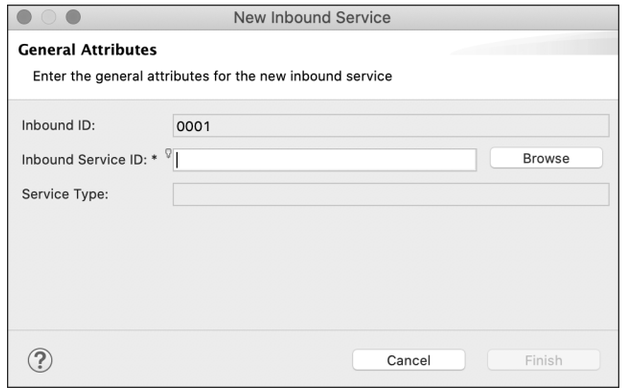


Figure 3.35 General Attributes

Next, assign one or more authorization objects, as shown in Figure 3.36. This step is optional, but if authorization objects need to be assigned, you'll need to provide the authorization objects.

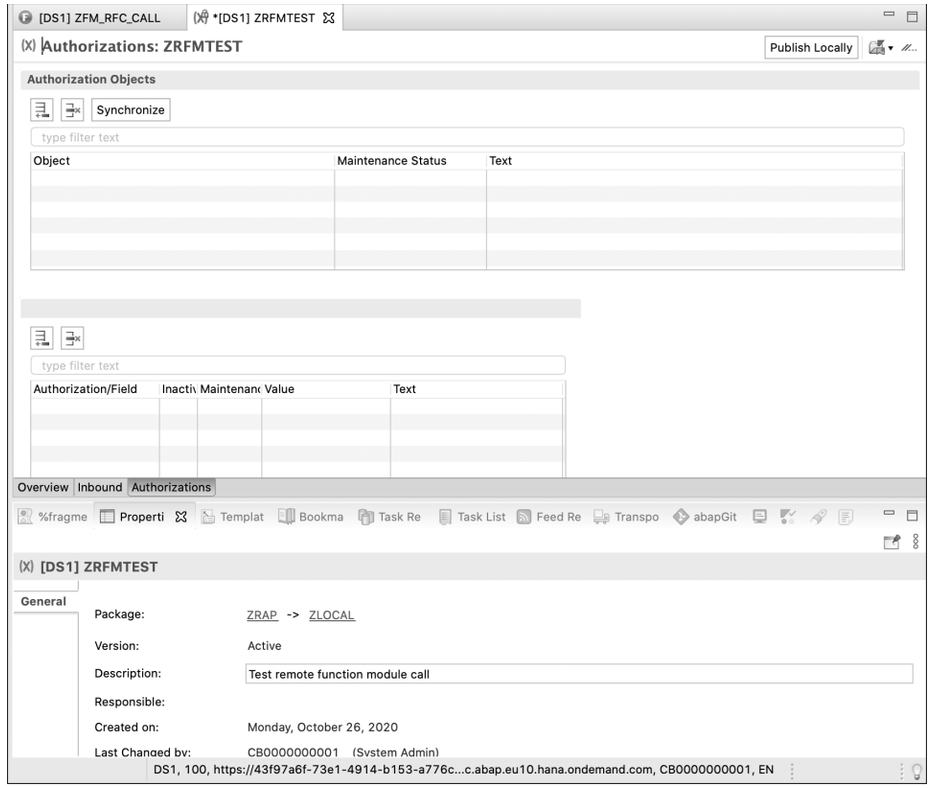


Figure 3.36 Authorization Assignment

If you assign authorization objects, they will be automatically assigned to the communication user later. Once this step is complete, you're ready to publish the communication scenario, as shown in Figure 3.37.

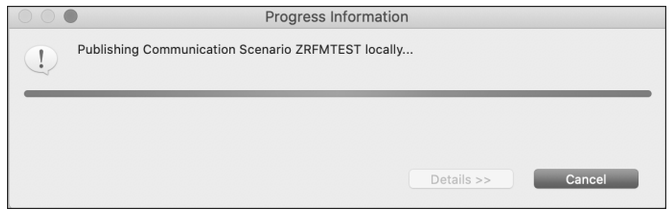


Figure 3.37 Publishing a Communication Scenario

This step concludes the setting up of the communication scenario, which is now ready for use in SAP BTP, ABAP environment.

Creating a Communication Arrangement

Let's now proceed with setting up the SAP BTP, ABAP environment dashboard. In this section, we'll create a communication arrangement via the **Communication Arrangements** tile, shown in Figure 3.38.

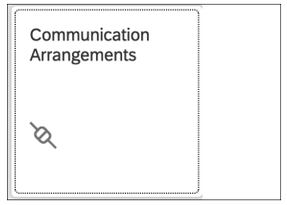


Figure 3.38 Communication Arrangements Tile

But, first, you'll need to complete two prerequisites: creating a communication user and creating a communication system. First, create a communication user via the Create Communication User app. Enter a user name, password, and description, as shown in Figure 3.39.

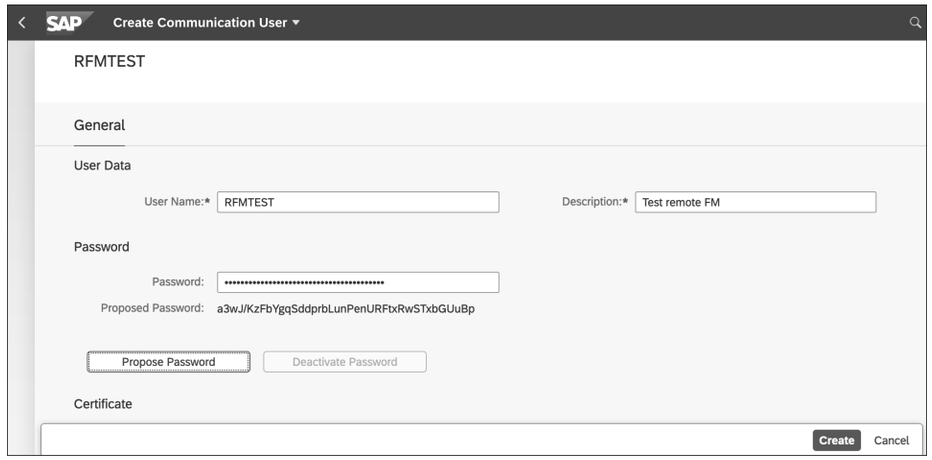


Figure 3.39 Creating a Communication User

The next step is to create a communication system via the Communication Systems app. As shown in Figure 3.40, enter a system ID, name, and business system.

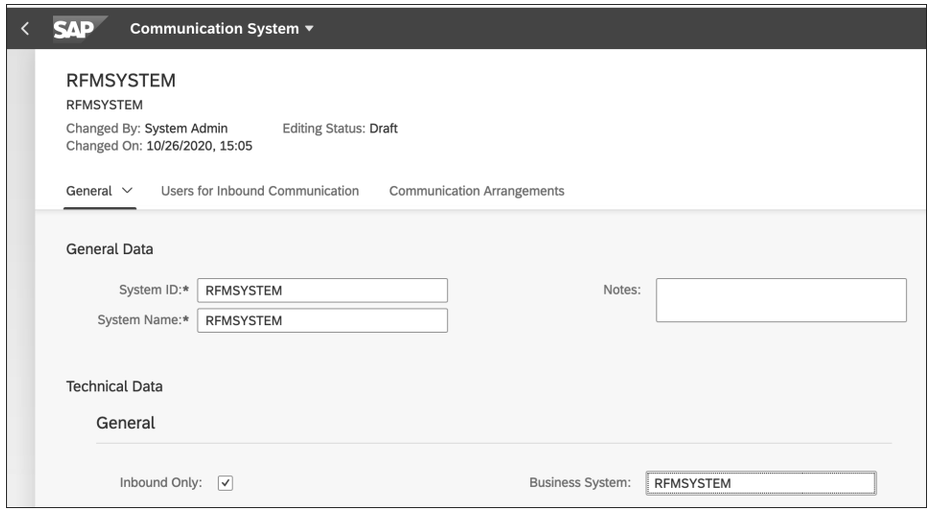


Figure 3.40 Creating a Communication System

In the communication system, we'll enter the previously created communication user in the **Users for Inbound Communication** section, as shown in Figure 3.41.

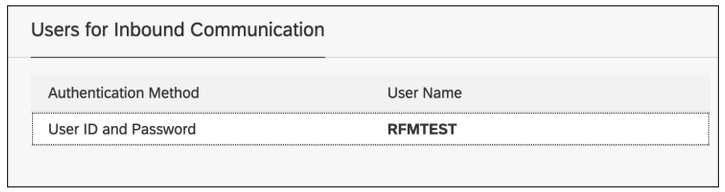


Figure 3.41 Assigning a Communication User to a Communication System

Now, we're ready to create a new communication arrangement. For this step, we'll use the Communication Arrangements app, shown in Figure 3.42.

To create a communication arrangement, we'll refer to communication scenario we configured previously in ADT.

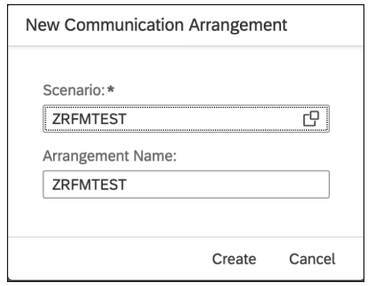


Figure 3.42 New Communication Arrangement Popup Window

Now, maintain the **Communication System** field, and all the relevant data, including communication user and the inbound services, will automatically be populated in the **User Name** field and under the **Inbound Services** heading, respectively, as shown in Figure 3.43.

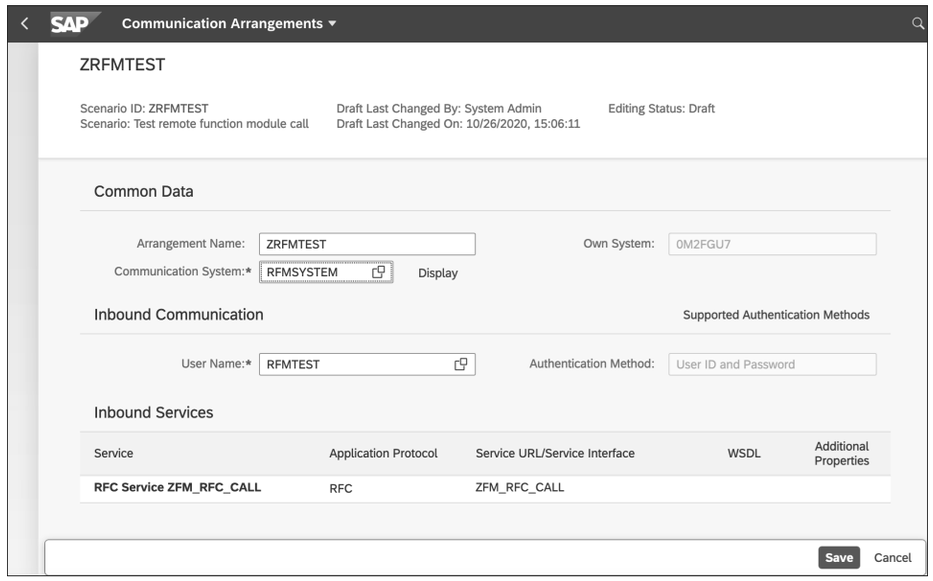


Figure 3.43 Creating a Communication Arrangement

Save your communication arrangement, which is now ready for use.

Let's say we want to add another inbound service for the inbound function module, to be called by on-premise system. On the **Communication Scenario** screen, go to the **Inbound** tab and maintain the fields under the **Inbound Settings** heading, as shown in Figure 3.44.

Figure 3.44 Maintaining Inbound Settings

The next step is to add an inbound service by clicking the **Add...** button. In the **General Attributes** window, click **Browse** to reach the screen shown in Figure 3.45.

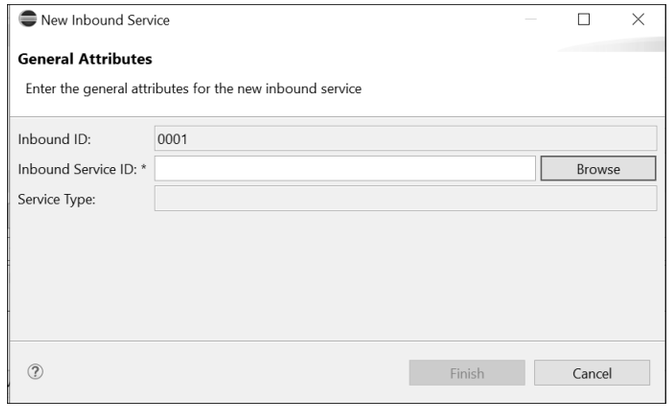


Figure 3.45 General Attributes

From the dropdown menu, select the inbound service by its name, as shown in Figure 3.46.

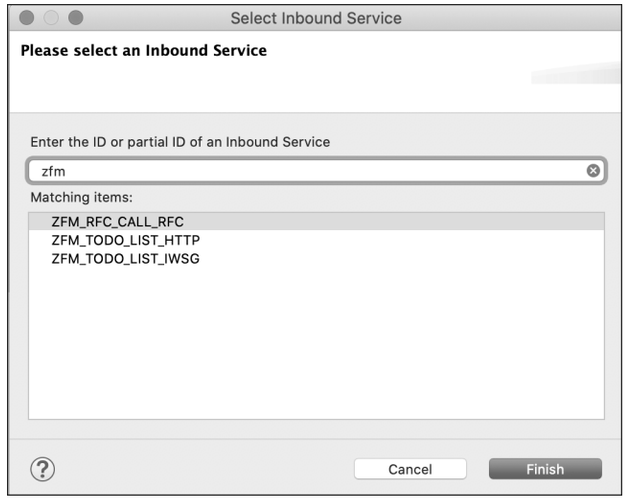


Figure 3.46 Inbound Service

Now, the **General Attributes** window should be populated with the service name in the **Inbound Service ID** field, as shown in Figure 3.47.

Finally, the inbound settings of the communication scenario will be populated with details about the inbound service, as shown in Figure 3.48.

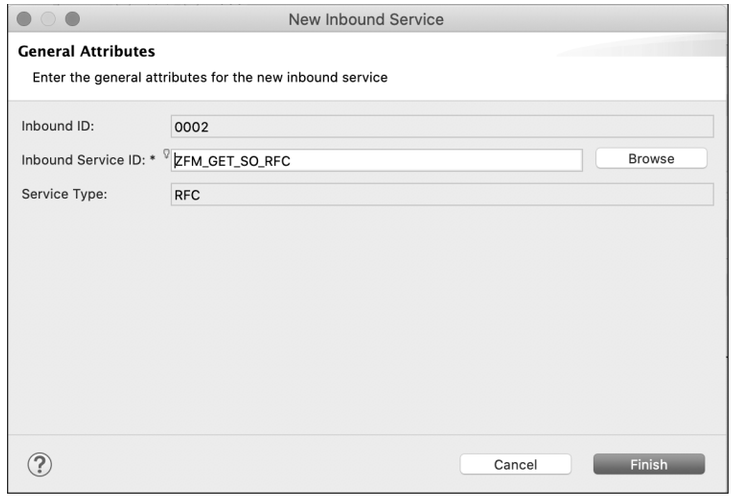


Figure 3.47 General Attributes

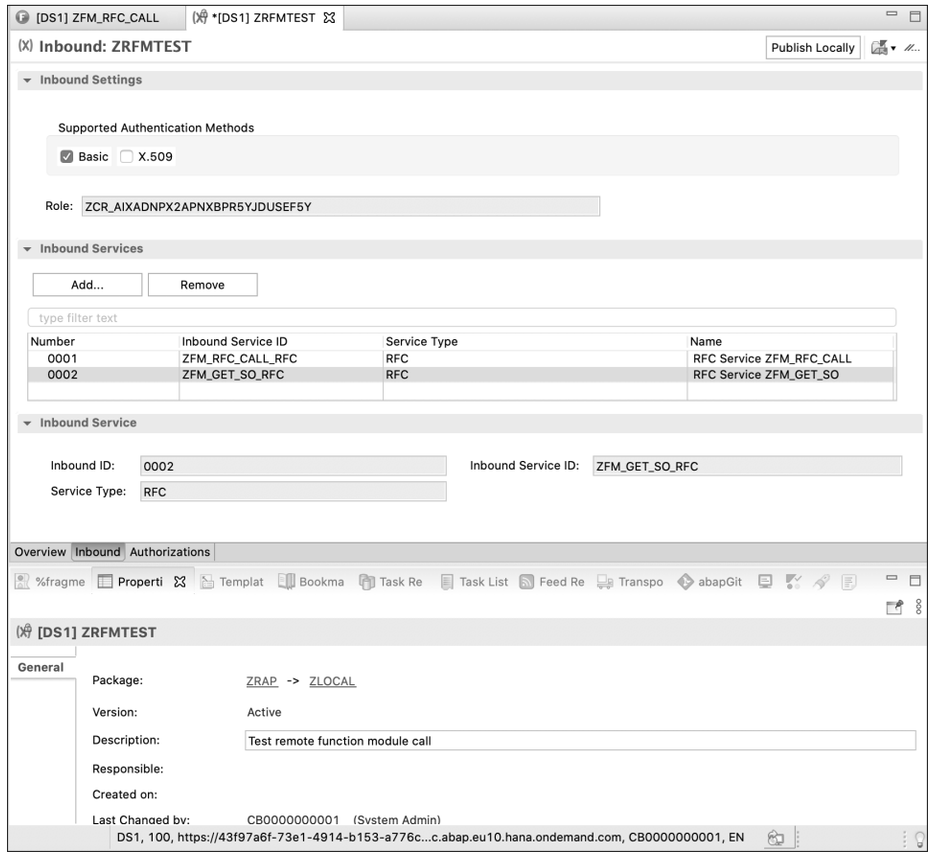


Figure 3.48 Inbound Settings

Once you publish the new configuration, it becomes automatically visible in our communication arrangement, as shown in Figure 3.49.

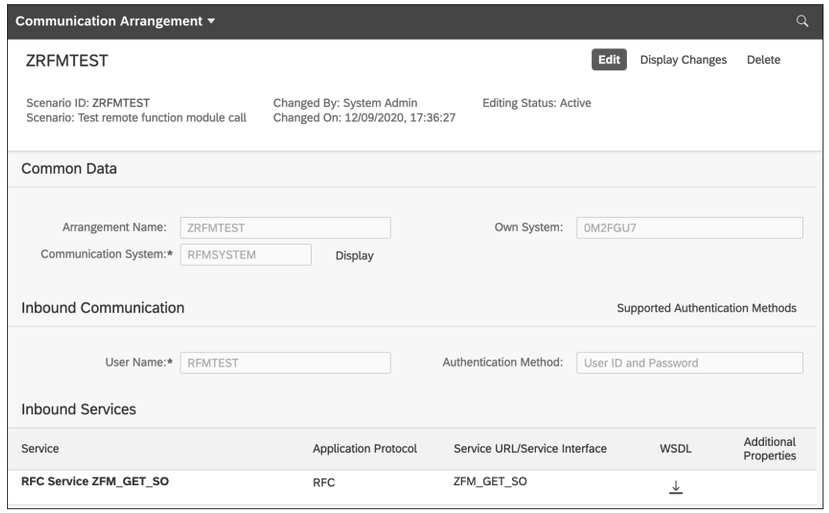


Figure 3.49 Communication Arrangement

Restrictions

Some restrictions regarding communication arrangements you'll need to keep in mind include the following:

- Only remote function modules that are part of a communication scenario can be called from an on-premise system. Most others are blocked, except for the system function module RFCPING.
- An inbound RFC connection supports only *basic authentication* as the authentication type.
- The user used for authentication must be a *communication user*; a *business user* is not allowed.
- The user name you enter in an RFC destination of an on-premise ABAP system is limited to 12 characters.

Setting Up the RFC Destination

Before you can call the service from an on-premise function module, you must set up the RFC destination in Transaction SM59. The destination is of type **3** (**ABAP Connection**).

As shown in Figure 3.50, under the **Technical Settings** tab, in the **Target Host** field, enter the cloud connector host. You'll use the instance number defined in the cloud connector itself, in the previous step, in the **Instance No.** field.

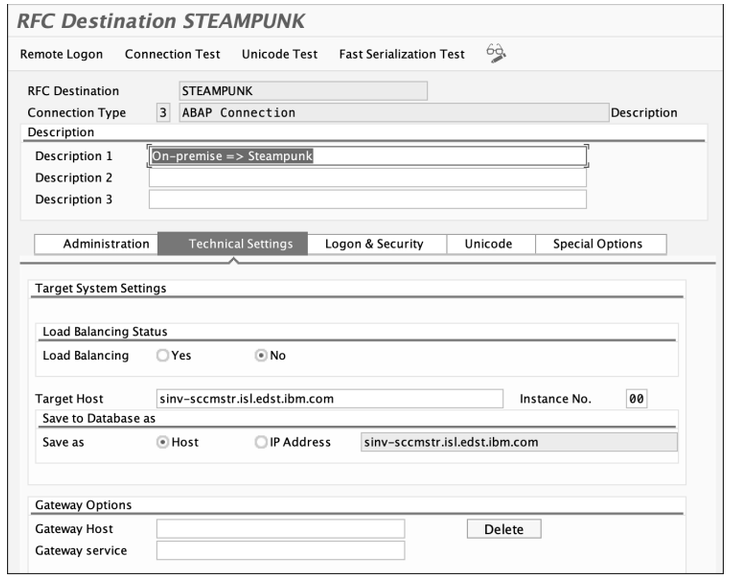


Figure 3.50 RFC Destination Details

As shown in Figure 3.51, under the **Logon & Security** tab, enter the communication user and password that was previously defined and shown earlier in Figure 3.39.

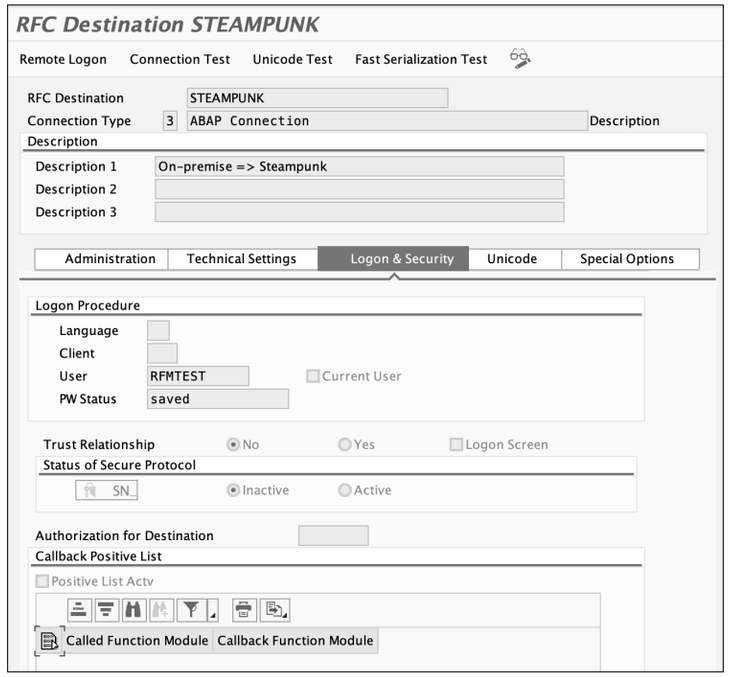


Figure 3.51 RFC Destination: Logon & Security

Now, test the connection and ensure no errors arise. In the case of successful execution, the result will be similar to the screen shown in Figure 3.52.

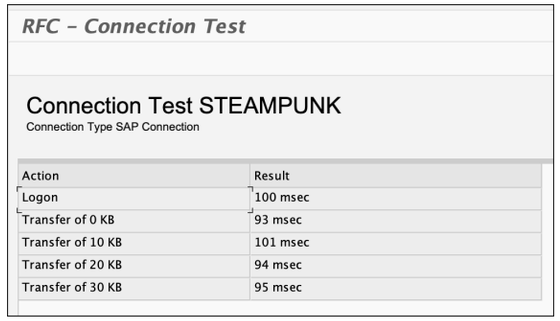


Figure 3.52 Test Connection

> **Note**
>
> When calling an RFC, write its name in uppercase.

Calling the Function Module

Now, let's create a custom report to call this function module in your on-premise system by using the code shown in Listing 3.7.

```
REPORT zre_call_rfc.

DATA : lv_lb TYPE dzmeng.

SELECTION-SCREEN BEGIN OF BLOCK b1 WITH FRAME TITLE TEXT-001.
PARAMETERS : p_kg TYPE dzmeng OBLIGATORY.
SELECTION-SCREEN END OF BLOCK b1.

CALL FUNCTION 'ZFM_GET_SO'
  DESTINATION 'STEAMPUNK'
  EXPORTING
    lv_kg     = p_kg
  IMPORTING
    lv_pounds = lv_lb.

WRITE :  lv_lb.
```

Listing 3.7 Report to Consume Function Module in On-Premise System

Notice how the syntax for the function module call is standard. Provide the function module details (name, importing/exporting parameters, etc.) and RFC destination.

Thus concludes this section where you learned how to consume RFC function modules, as well as SOAP and OData services, in SAP BTP, ABAP environment from an on-premise system. You also learned how to configure communication scenarios and communication arrangements and how to consume RFC modules from an on-premise system on SAP BTP, ABAP environment.

In the next section, you'll learn how to connect cloud systems and how to consume an API on SAP S/4HANA Cloud.

3.2 Connecting to Cloud Systems

In the previous section, we covered the consumption of services that are exposed from on-premise systems.

When connecting to cloud-based systems, however, the main difference is in connectivity itself. Since the source system is available via a public IP address, you don't have to enforce additional security measures or create VPN tunnels (like cloud connector); you can simply establish a point-to-point connection.

Although SAP's software as a service (SaaS) portfolio includes multiple products, like the SAP Customer Experience product line, SAP SuccessFactors, Qualtrics, etc., from an SAP BTP, ABAP environment perspective, for building side-by-side extensions, the focus is SAP S/4HANA Cloud.

> **Note**
>
> More information about SAP S/4HANA Cloud features and capabilities can be found at *https://www.sap.com/products/s4hana-erp/features/cloud-release.html*.

Due to limited in-app extensibility options with SAP S/4HANA Cloud, side-by-side extensions are important to bridge the gaps, when business requirements exist that cannot be fulfilled otherwise. SAP BTP, ABAP environment plays an important role in building side-by-side extensions for SAP S/4HANA Cloud.

In this section, we'll focus on building extensions for the public cloud version of SAP S/4HANA Cloud, highlighting several differences from extensions for on-premise systems.

3.2.1 SAP API Business Hub

We'll start with SAP API Business Hub, which is a central location for exploring, discovering, and consuming APIs, prepackaged integrations, business services, and even sample apps created by SAP and selected partners.

Contents

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

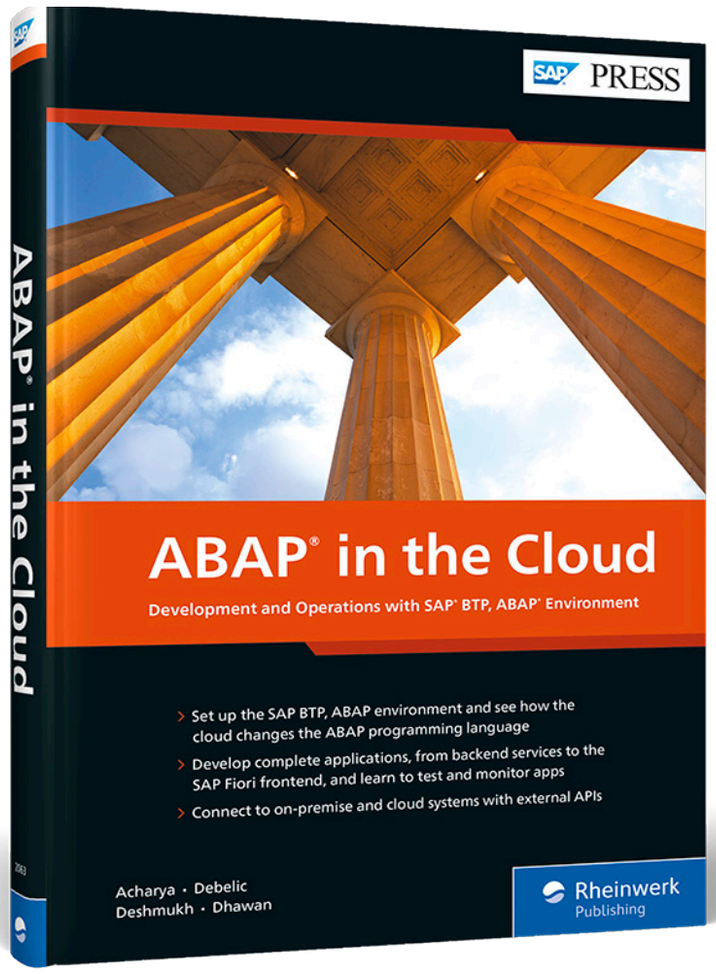
U

V

W

X

Y



Gairik Acharya, Aleksander Debelic, Shubhangi (Deshmukh) Joshi, Aayush Dhawan

ABAP in the Cloud: Development and Operations with SAP BTP, ABAP Environment

613 Pages, 2021, $79.95
ISBN 978-1-4932-2063-2

www.sap-press.com/5236

Gairik Acharya is a senior technical architect and associate partner at IBM with more than 20 years of IT experience. He is a recognized expert in ABAP, SAP HANA, OData, SAP Gateway, SAP S/4HANA, SAP Fiori, SAPUI5, SAP BTP, and SAP Mobile Platform.

Aleksandar Debelic is an SAP technical architect at IBM. He has more than 17 years of SAP experience, primarily as a technical solution architect and technology team lead. He is a managing consultant and member of the Global SAP Center of Competence and a global development team lead at SAP Innovation by IBM.

Shubhangi (Deshmukh) Joshi is an SAP technical architect at IBM with more than 15 years of SAP experience in the areas of project delivery, asset building, and SAP S/4HANA migration projects. In addition to her ABAP experience, her main areas of expertise are SAP S/4HANA, machine learning with SAP, SAP Conversational AI, SAP CoPilot, SAP BTP, and SAP Analytics Cloud.

Aayush Dhawan is a managing consultant at IBM. He has more than 14 years of SAP experience in the areas of technical development, functional consulting, and asset building. As part of the SAP Innovation team at IBM, he has contributed towards many co-innovation projects on emerging technologies including cloud development, digital advisors, blockchain, machine learning, and data science.