

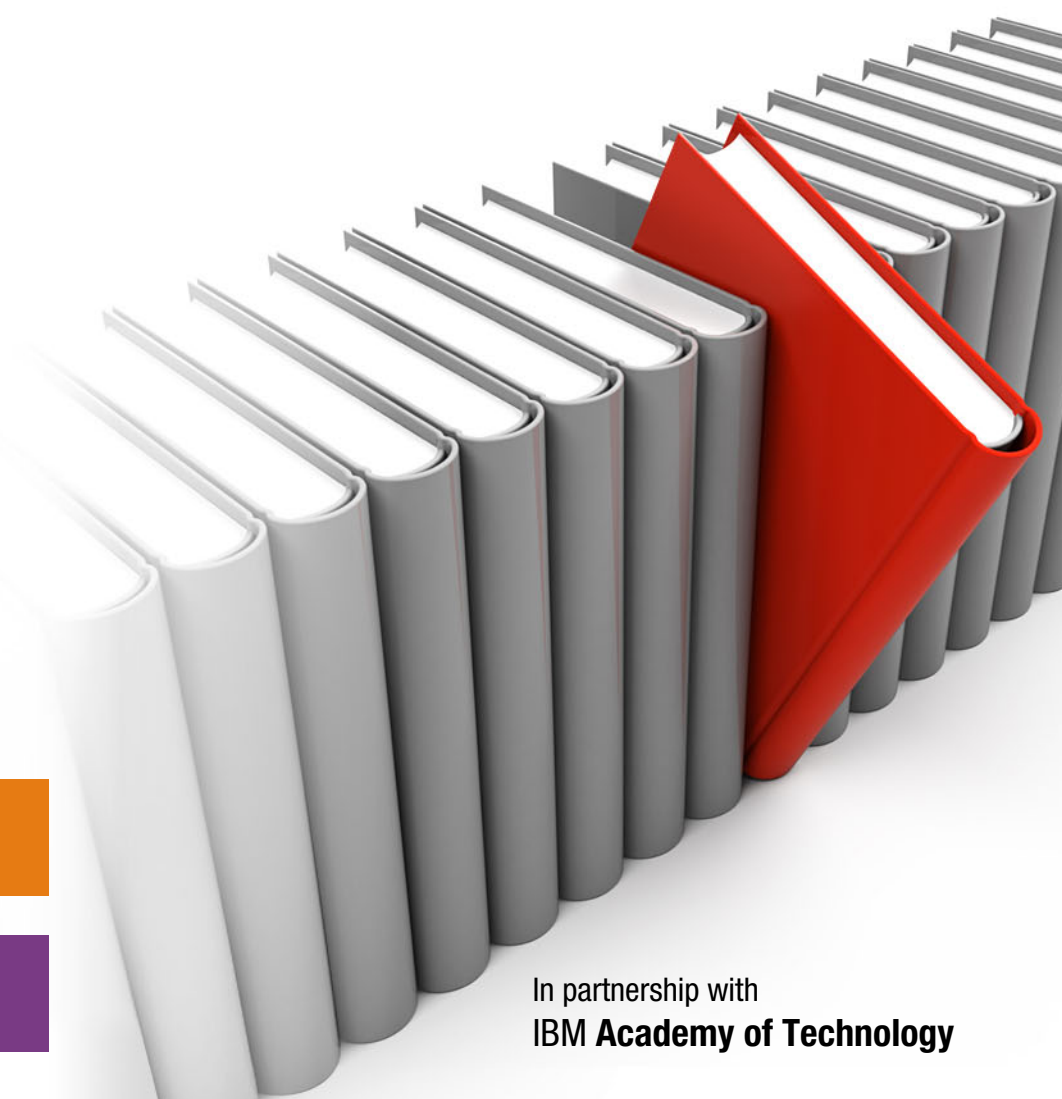
Modernize SAP Workloads on IBM Power Systems

Dino Quintero
Christian Bartels
Stefan Diederichs
Joerg Droste
Joachim Rese
Jochen Röhrig
Isgandar Valizada



Analytics

Power Systems



In partnership with
IBM Academy of Technology



IBM Redbooks

Modernize SAP Workloads on IBM Power Systems

May 2021

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (May 2021)

This edition applies to Version:

- ▶ ABAP SDK for SAP NetWeaver 7.50 and higher.
- ▶ SAP HANA 2.0 SPS 02.
- ▶ IBM Watson Machine Learning Accelerator 1.2.1.
- ▶ Red Hat OpenShift 3.11.
- ▶ TensorFlow Core v2.4.1.

This document was created or updated on May 19, 2021.

© Copyright International Business Machines Corporation 2021. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	ix
Now you can become a published author, too!	xi
Comments welcome	xi
Stay connected to IBM Redbooks	xi
Chapter 1. Hybrid system landscapes with SAP Business Suite and SAP S/4HANA	
introduction	1
1.1 Introduction	2
1.2 IBM Cloud	2
1.3 IBM Watson	3
Chapter 2. SAP connecting with cloud services: ABAP SDK for IBM Watson	5
2.1 General considerations	6
2.1.1 IBM Watson AI Services introduction	6
2.1.2 Integration of IBM Watson services into SAP applications	9
2.1.3 Network considerations	10
2.1.4 Licensing considerations	10
2.1.5 Administering credentials	11
2.1.6 ABAP SDK for IBM Watson	11
2.2 SAP system configuration	12
2.2.1 SAP profile parameters	12
2.2.2 Setting up secure Internet communication	12
2.3 Installing the ABAP SDK for IBM Watson	20
2.3.1 Introduction to abapGit	21
2.3.2 Installing the ABAP SDK for IBM Watson	28
2.4 Using the ABAP SDK for IBM Watson	32
2.4.1 ABAP SDK for IBM Watson API overview	32
2.4.2 Credentials	33
2.4.3 Configuration table	33
2.4.4 Identity and Access Management authentication	34
2.4.5 Using classes, methods, and data types of the ABAP SDK	35
Chapter 3. Cloud App connecting to SAP	41
3.1 Introduction	42
3.2 IBM Secure Gateway	42
3.3 OData and CDS Views	46
3.3.1 Sample data table	46
3.3.2 Creating Core Data Services View	47
3.3.3 Testing the OData service	48
3.4 Building a node.js application on IBM Cloud	51
3.4.1 Installing and running application locally	57
3.5 Using OData and IBM Watson services	59
3.5.1 Creating the back-end application	59
3.5.2 Node.js server code	59
3.5.3 Creating front-end code	60

3.5.4 Creating HTML index page	60
Chapter 4. Machine Learning on IBM Power Systems.	63
4.1 IBM Watson Machine Learning Accelerator	64
4.1.1 Installing on Red Hat Enterprise Linux	64
4.1.2 Installing on Red Hat OpenShift 3.11	64
4.2 Use case scenario and implementation	69
4.2.1 Naming scheme	69
4.2.2 Installing SAP HANA.	69
4.2.3 Target [A] machine	72
4.2.4 Adding EML to the installed SAP HANA.	72
4.2.5 Installing SAP HANA Studio	73
4.2.6 Connecting SAP HANA Studio to your SAP HANA database.	74
4.2.7 Checking the EML installation.	75
4.3 Preparing AI model for deployment.	77
4.3.1 Preparing a TMS instance	79
4.3.2 Communicating with TMS	80
4.3.3 Updating model configuration	80
4.3.4 Checking connectivity from SAP HANA	81
4.3.5 Preparing input images	81
4.3.6 Encoding input images as BASE64	82
4.3.7 Creating input and output types and a parameters table	83
4.3.8 Creating EML procedure wrapper.	83
4.4 Conclusion	85
Chapter 5. Benefits of modernizing SAP applications with IBM AI	87
5.1 Enterprise agenda	88
5.2 IBM Watson Studio	88
5.3 IBM Watson OpenScale	89
5.3.1 Monitors	89
5.3.2 Data sets.	90
5.3.3 Checking model drift	90
5.3.4 Trust in AI and IBM Watson OpenScale	90
5.3.5 Open Source Initiatives.	91
5.4 AutoAI	91
5.4.1 AutoAI in data science process.	91
5.4.2 AutoAI benefits	92
5.4.3 Creating a machine learning model by using AutoAI	92
Appendix A. node.js server code	105
node.js server code	106
Abbreviations and acronyms	117
Related publications	119
IBM Redbooks	119
Other publications	119
Online resources	119
Help from IBM	120

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.


Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Db2®
IBM®
IBM Cloud®
IBM Cloud Pak®

IBM Watson®
IBM Z®
OpenScale™
POWER®

Redbooks®
Redbooks (logo) ®
Watson OpenScale™

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

OpenShift, Red Hat, RHCE, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The goal of this IBM® Redpaper® publication is to describe how to modernize SAP workloads and run them on IBM Power Systems. By using theoretical knowledge, team experiences, and new technologies, the authors document sample scenarios to show such benefits.

This publication uses any available documentation, hardware, and software resources to accomplish the following tasks:

- ▶ Document guidelines to help the reader modernize SAP workloads to run on IBM Power Systems.
- ▶ Document guidelines to help provide an optimal system configuration and implementation to run the SAP workloads.
- ▶ Conduct and document an implementation case study.

This publication is targeted to architects, brand specialists, distributors, resellers, and anyone who wants integrate and run SAP workloads on IBM Power Systems. Moreover, this publication provides information to transfer the how-to-skills to the technical and sales teams.

Authors

This paper was produced in collaboration with the IBM SAP International Competence Center (ISICC) in Walldorf, SAP Headquarters in Germany and IBM Redbooks®.



Dino Quintero is an IT Management Consultant and an IBM Level 3 Senior Certified IT Specialist with IBM Redbooks in Poughkeepsie, New York. He has 24 years of experience with IBM Power Systems technologies and solutions. Dino shares his technical computing passion and expertise by leading teams developing technical content in the areas of enterprise continuous availability, enterprise systems management, high-performance computing, cloud computing, artificial intelligence including machine and deep learning, and cognitive solutions. He also is a Certified Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Christian Bartels is the team lead of the joint SAP and IBM development team for SAP on IBM i in Germany. He has 23 years of experience with SAP on IBM i and 30 years of experience with the IBM i platform, formerly known as AS/400. He holds a Diplom (comparable to a master's degree) in Electrical Engineering from the Technische Universität Braunschweig, Germany. He has been working in different areas of the SAP NetWeaver development, including the database interface for IBM Db2® for i, the SAP Computing Center Management System (CCMS), performance monitoring, performance analysis, and third-level customer support and education.

Stefan Diederichs started his career in 1987 in the IBM® development Lab in software development for banking after studying Electrical Engineering at the University of Cooperative Education with IBM. In 1999, Stefan joined SAP development in Walldorf and was responsible for porting the SAP Application Server to IBM mainframe running Linux. In 2001, he became a member of the technical pre-sales team in the Systems Innovations Center for SAP Proof of concepts and customer benchmarks. In 2016, Stefan joined the IBM Client Center Boeblingen, leading the Machine Learning and PowerAI team as an IT architect in the Europe region. Since 2018, he is part of SAP on IBM Z® and Analytics team, supporting SAP on IBM Platforms. Currently, he focuses on the area of IBM Cloud® Pak for data running on Red Hat OpenShift.

Joerg Droste is a Senior Software Engineer in Beaverton OR, US. He has more than 25 years of experience in systems performance and SAP software processing. His areas of expertise include hardware, operating system and SAP application administration, analysis, and tuning on-premise and in the cloud. He holds a bachelor's degree in computer science and business administration from the Berufsakademie Mannheim.

Joachim Rese is a Senior Software Engineer in Germany. He has more than two decades of experience in the field of enabling data driven processes with SAP. His areas of expertise include database technologies, analytics, artificial intelligence, and IBM Watson®. He holds degrees in Mathematics and Computer Science from the University of Paderborn, Germany.

Jochen Röhrig is a Senior Software Engineer in the joint IBM/SAP platform team for SAP on IBM Systems at SAP in Walldorf, Germany. Having worked on enabling SAP software on traditional IBM systems in the past, he is focusing on emerging topics, such as running SAP systems on Red Hat OpenShift, the use of IBM Watson services in ABAP, or connecting SAP systems to IBM Blockchain. Having worked for IBM for over 20 years, Jochen has experience in Linux and over 16 years of experience in SAP on IBM platforms. He holds a German and a French master's degree in Computer Science, and a Ph.D. in Computer Science from the Saarland University, Saarbrücken, Germany. He is a Red Hat Certified Engineer (RHCE, 2004) and holds certificates LPIC-1 (2006) and LPIC-2 (2008) of the Linux Professional Institute. His areas of expertise include emerging technologies, such as cloud computing, containerization, AI, and blockchain, and traditional topics, such as software development, open source software, operating systems, parallel computing, and SAP on IBM platforms.

Isgandar Valizada is a Software Engineer in IBM Germany since January 2016. He joined IBM after working for 4 and a half years as a Software Developer and Researcher in the area of Digital Preservation at the University of Freiburg. His areas of expertise include software architectural design and implementation, full stack development, algorithm analysis, and artificial intelligence. He holds a B.Sc. degree in Mathematics from Azerbaijan State Oil and Industry University and M.Sc. in Computer Science from the University of Freiburg in Germany.

Thanks to the following people for their contributions to this project:

Suraj Bharadwaj, for his work in Chapter 4, "Machine Learning on IBM Power Systems" on page 63

Wade Wallace
IBM Redbooks®, Poughkeepsie Center

Katharina Probst, Walter Orb, Tanja Scheller, Suraj Bharadwaj
IBM Germany

Reinaldo Katahira
IBM Brazil

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Hybrid system landscapes with SAP Business Suite and SAP S/4HANA introduction

This chapter provides an introduction to hybrid system landscapes.

This chapter includes the following topics:

- ▶ 1.1, “Introduction” on page 2
- ▶ 1.2, “IBM Cloud” on page 2
- ▶ 1.3, “IBM Watson” on page 3

1.1 Introduction

Traditionally, applications were written by using procedural or object-oriented models that use functions, modules, and includes. These applications provide clearly defined functions. The source code defines every intended execution option. The data for these applications must be stored in defined data stores that provide specific access methods and ensure data integrity.

SAP is a leader in business application processing for many years. The core of SAP's applications is deeply embracing this computing paradigm and providing exceptional functions and data security for thousands of customers.

Over the last 5 - 10 years, a new type of computing emerged. To solve problems the traditional procedures and functions with many if/then/else structures cannot process, the rejuvenated area of Artificial Intelligence (AI) grew rapidly. The fundamental change in the computing methodology is not to tell the computer exactly what to do, but to show it examples of how things were seen or measured in the past and use smart algorithms to let the computer figure out how to use insight from these examples to solve a new or different problem.

The area of AI is not new; it has been around for some time. However, limitations in the availability of data and the amount of required compute cycles inhibited the broad adoption of AI.

The emergence of big data made vast amounts of data available that in most cases is unstructured and therefore difficult to digest for traditional applications. Rapid advances in the processing power of computer servers, including the use of graphics chips that are well suited for large matrix operations, made a wide adoption of AI possible today.

A second shift in the traditional computing landscape is the move of enterprise application processing to the cloud. Traditionally, companies purchase a server and some software, and hire consultants to create a solution that addresses a need in the customer's business. This effort required maintaining servers, infrastructure, software, and applications, and retaining skilled administrators.

The requirement of accessing enterprise functions from everywhere in the world with an increasing number of different devices, such as cell phones, tablets, and laptops fueled a move of more applications into the cloud. Cloud computing is a fundamental shift in the way applications are deployed. Large monolithic applications are broken up into micro-services that are easy to deploy and govern.

With enterprise customers increasingly moving their IT strategy toward cloud computing and AI, this document shows options to use these new technologies to create modern applications on-premises or in the cloud that are seamlessly integrated with SAP systems running mission critical business applications.

1.2 IBM Cloud

Although companies are moving workloads to the cloud, some of their mission-critical applications still run in monolithic applications in their data center. Some do not want to expose confidential data to the internet. The fact that many companies provide cloud infrastructure makes it difficult for businesses to choose a provider and effectively move the cloud.

IBM's strategy is to provide a hybrid cloud with open standards and full control over when and where to run applications: on-premises, in IBM Cloud, or on any other cloud.

Cloud computing includes the following terms:

- ▶ Public cloud: A broad set of computing options, including server infrastructures, serverless computing, and support for a large number of services.
- ▶ Private cloud: A large number of containerized services to create, operate, and maintain cloud native applications and services on-premises.
- ▶ Hybrid cloud: An integrated environment that includes public and private cloud services that are integrated with multi-cloud management tools.

IBM Cloud features the following benefits:

- ▶ Open platform: Build one Kubernetes container infrastructure for public and private cloud and run applications and workloads wherever they run best.
- ▶ Integration solutions: Use cloud tools, such as messaging, gateways, and API Management to bring together new applications with existing applications and workloads.
- ▶ DevOps: Accelerate the development and deployment of hybrid applications with synchronous deployment of multiple components on public and private cloud.

1.3 IBM Watson

IBM Watson is an IBM suite of enterprise-ready AI services, applications, and tools. It enables companies to unlock the value of their data in an entirely new and profound way. By using insights from IBM Watson, the shape of future business outcomes can be predicted, at the same time rethinking existing practices and workflows.

Many companies focus on developing applications that are infused with AI to solve specific problems. When developing world-class, enterprise-ready applications, it became clear that the main focus is not on the machine learning (ML) algorithm, but all of the lifecycle tasks that make up an enterprise application.

The following components must be in place to develop modern applications:

- ▶ Data collection: Data is key because it must be readily available to the application.
- ▶ Feature extraction: With large amounts of data, the information that is required for processing must be identified and defined so that it can be used by the application.
- ▶ Data verification: After the model is trained, the data that is processed by the application must be validated to ensure that it matches the data that is used for training.
- ▶ Machine resource management: No matter where an application is deployed, it must be run somewhere, and the required resources must be defined, identified, and allocated.
- ▶ Analysis tools: After the application is running, analysis must be done to ensure the correct execution; for example, ensure that no drift or bias exists.
- ▶ Process management tools: From design to customer deployment, the process of developing the application must be supported by the infrastructure, and the workflow must be clear and documented.
- ▶ Serving infrastructure: No matter where execution occurs (on-premises, in the IBM Cloud, or any other cloud), the application must be designed to run anywhere.
- ▶ Monitoring: When running an application, it is necessary to watch its execution and results. Sophisticated monitoring tools are required to ensure seamless operation.

In addition, these components of ML span multiple teams, including the following examples:

- ▶ Data Engineers
- ▶ Data Scientists
- ▶ Machine Learning Engineers
- ▶ Production Engineers

To address these enterprise requirements, IBM Watson was designed as an AI platform and full workflow orchestrator.



SAP connecting with cloud services: ABAP SDK for IBM Watson

Advanced Business Application Programming (ABAP) describes a programming language, development, and runtime environment that is used widely by SAP applications. It allows customers and Business Partners to modify and extend standard SAP applications.

IBM developed a software development kit (SDK) for ABAP to simplify the access of IBM Watson services from within applications that are written in ABAP and published it under an open source license.

This chapter describes the ABAP SDK for IBM Watson and provides an example of how to use it.

This chapter includes the following topics:

- ▶ 2.1, “General considerations” on page 6
- ▶ 2.2, “SAP system configuration” on page 12
- ▶ 2.3, “Installing the ABAP SDK for IBM Watson” on page 20
- ▶ 2.4, “Using the ABAP SDK for IBM Watson” on page 32

2.1 General considerations

With the introduction of the product SAP R/3 in 1992, SAP provided enterprise resource planning (ERP) software for large enterprises and reached out to small and medium businesses. Over time, these customers collected large amounts of data in their ERP systems and applied much business knowledge in their processes within their SAP systems.

Although they are interested in new technologies and want to apply those technologies for their benefit, they do not want to give up their existing systems and start over. As a consequence, customers and Business Partners are implementing hybrid solutions in which ERP systems exchange information with new services to use artificial intelligence (AI) and other modern technologies.

With the ABAP SDK for IBM Watson, customers can develop programs or extend business applications in their ERP systems (ABAP) so that they can use IBM Watson services with their business data. In this approach, the application logic is run in the SAP system based on SAP NetWeaver ABAP and accesses IBM Watson services through the HTTPS protocol. The new application components are integrated in the authorization model, user interface, and process flow of the accustomed, well-established SAP business applications. The application programmers continue to use their ABAP skills in their familiar environment.

The opposite approach, where most of the application logic is run in the cloud and accessing only the data from the SAP application, is described in Chapter 3, “Cloud App connecting to SAP” on page 41.

2.1.1 IBM Watson AI Services introduction

IBM Watson services can be accessed easily in the public cloud at [this website](#) (log in required).

In this section, we describe managed services for AI, which can be found in the services catalog of IBM Watson (see Figure 2-1).

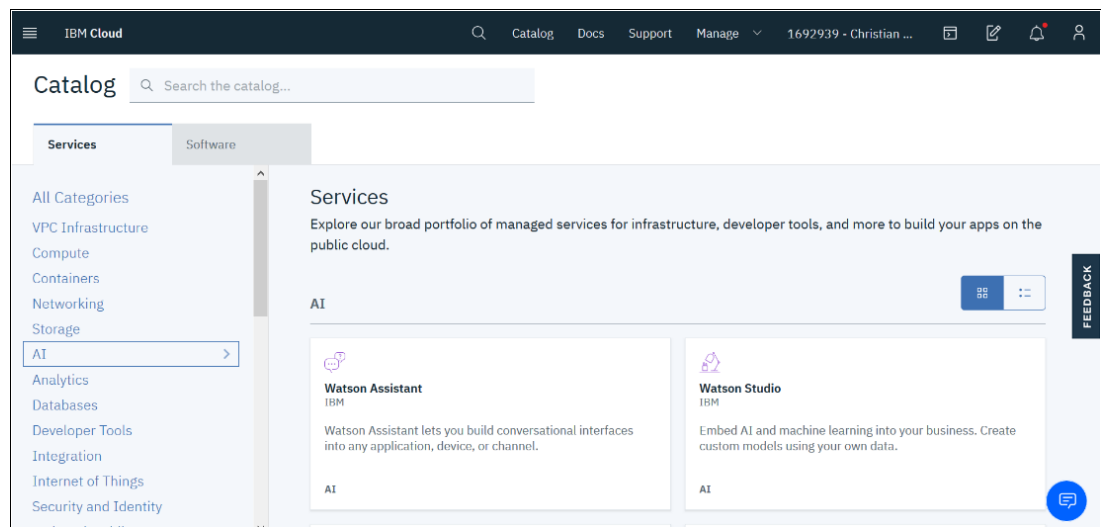


Figure 2-1 IBM Watson services catalog

To use a service, you must create a resource for that service. The attributes of the resource depend on the access control method. Historically, IBM offered Cloud Foundry services that were linked to an organization and a space and required a username and password to connect. Meanwhile, all IBM Watson services for AI support IBM Cloud Identity and Access Management (IAM) access control. For IAM services, the following attributes must be selected:

- **Region**

The region defines the data center for deployment of the service resource. Some services are not available in all data centers. When creating the resource, you can select the deployment region for your service from a pull-down menu. Here, you can choose a region that is close to your location.

The region defines the data center for deployment of the service resource. Some services are not available in all data centers. When creating the resource, you can select the deployment region for your service from a pull-down menu. Here, you can choose a region that is close to your location.

- **Pricing plan**

Each service offers various pricing plans. Most of the services that are supported by the ABAP SDK for IBM Watson offer a Lite plan at no cost. It includes limitations in supported methods, number of calls, and amount of data that is processed, but can be used for first trial versions.

For productive use, the services typically offer a Standard, Advanced, or Tiered plan. With these plans, charges are applied based on usage. The exact prices are shown in the pricing plan selection view. In addition, some services offer a Premium plan with special features for data privacy, high availability, and a single tenant environment.

- **Resource group**

The resource group is part of IBM Cloud Identity and Access Management (IAM) concept. A Default resource group always is available. Unless you have a Lite account or a 30-day trial, you can create resource groups. You can assign a specific resource group to your service resource to manage quota and view billing usage for a set of resources.

When the IBM Watson service resource is created with the selected attributes, its credentials are assigned. For IAM access control, the credentials consist of an API key and a URL. The API key is a character string that consists of uppercase and lowercase letters, digits, and special characters, similar to a password.

The URL is generated from the service name, selected region, and unique identifier. For example, for the Language Translation service in the region Frankfurt, you can get a URL as shown in the following example:

```
https://api.eu-de.language-translator.watson.cloud.ibm.com/instances/2ae347e6-3545-469a-99b3-fbbebdd7afd9
```

The structure of the URL can change over time, but you can copy and use whatever URL is generated when you create the service resource.

Note: To connect to the service resource from your application, you need the API key and the URL provided with the resource, *not* the IBM Cloud account and password of the owner.

You can display the main attributes of your service resource through the IBM Cloud browser interface, as shown in Figure 2-2.

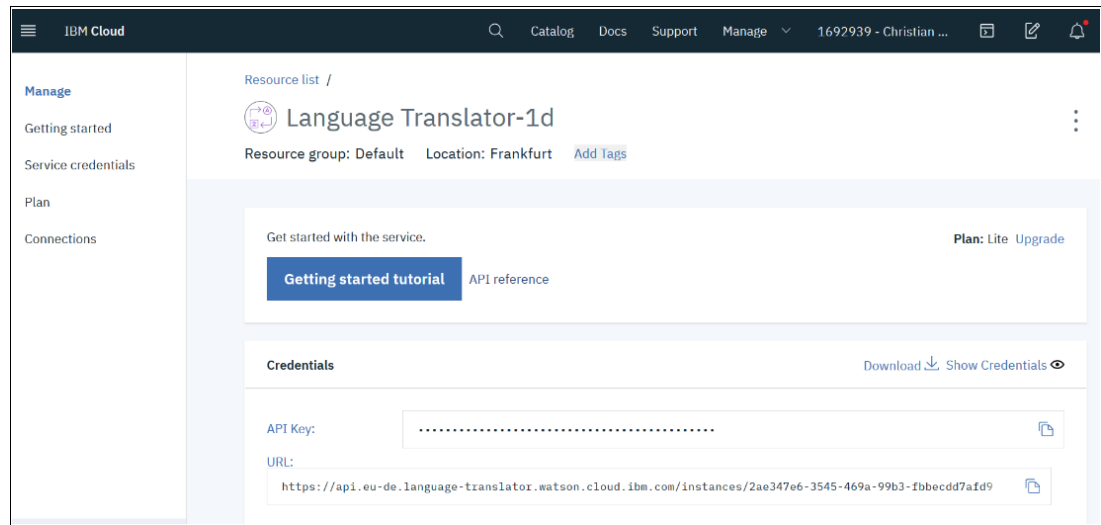


Figure 2-2 Service attributes of Language Translation resource

You can select Show Credentials to see the API Key value in clear text. In this view, you also find links to the service documentation, a Getting started tutorial with an introduction to the service features, and an API reference with detailed information about authentication, versioning, error handling, data handling, and the supported methods.

IBM Watson AI services are following the representational state transfer (REST) architectural style. Communication with the services is done through HTTP methods GET, POST, and DELETE. Data is exchanged between the service and its caller in the JavaScript Object Notation (JSON) format. You can easily use services interactively by using the curl utility, which is included as part of Microsoft Windows 10, Apple Mac OS X, and other operating systems. It also is available for download at [this website](#).

For example, to translate a sample sentence from English into German by using the Language Translator service, run following command:

```
curl --user apikey:UZ5E17jKADciPoRYIwjWakFUCysfk-HjRxJa_C8Fdpf --request POST
--header "Content-Type: application/json" --data "{\"text\":[\"This is a sample
sentence in English language\"],\"model_id\":\"en-de\"}"
https://api.eu-de.language-translator.watson.cloud.ibm.com/instances/2ae347e6-3545
-469a-99b3-fbbecdd7afd9/v3/translate?version=2018-05-01
```

This request returns the translated sentence plus some statistics in JSON format:

```
{
  "translations" : [ {
    "translation" : "Dies ist ein Mustersatz in englischer Sprache"
  } ],
  "word_count" : 8,
  "character_count" : 45
}
```

Note: In the previous example, an invalid value was provided for the apikey. If you paste the command, you receive the following error information:

```
{"code":401, "error": "Unauthorized"}
```

To run the example on your own, create your own resource for the Language Translator service and use its credentials (API key and URL).

The IBM Watson services contain versioning information. When significant changes to the services are implemented, a new version of the service is supplied. By providing the version information with the REST call, applications that were created for earlier versions of the service still can be run.

2.1.2 Integration of IBM Watson services into SAP applications

Since the introduction of SAP NetWeaver Application Server 6.20 in 2002, SAP applications that were written in the ABAP programming language can communicate with web applications. The ABAP REST Library can be used to communicate with IBM Watson services from within an ABAP application server. To use the ABAP REST Library, an HTTP client object of type CL_HTTP_CLIENT must be created. An HTTP connection can be established by using one of the following methods:

- Provide a URL by using the CL_HTTP_CLIENT → CREATE_BY_URL method.
- By using a predefined HTTP connection (transaction SM59) through the CL_HTTP_CLIENT → CREATE_BY_DESTINATION method.

The ABAP SDK for IBM Watson services uses the first method because it offers more flexibility. Through interface IF_REST_CLIENT, the ABAP REST Library offers the methods GET, POST, DELETE, and others to communicate with RESTful web services, such as the IBM Watson services.

Exchanging data between the SAP application server ABAP and the IBM Watson services poses a challenge. ABAP programs process data in elementary data types or composite data types, such as structures or internal tables. IBM Watson services exchange data in JSON format, which is a flexible, text-based format that follows a standardized grammar (ECMA-404 or IETF RFC 8259). The main differences between the properties of ABAP data and JSON data are shown in Table 2-1.

Table 2-1 Properties of ABAP and JSON data

ABAP data types and data objects	JSON grammar
Case-insensitive names	Case-sensitive names
Strict types (numeric, character, and Boolean)	Unicode character string representation
Static data types (fields, structures, and internal tables), Runtime Type Information (RTTI)	Dynamically generated character string
Maximum name length: 30 characters	Unlimited name length

Conversion from JSON to ABAP and vice versa can be done in different ways. A convenient way is provided through ABAP class /UI2/CL_JSON, which is available through the UI2 Add-on and can be applied from SAP NetWeaver 7.0 onwards. As of SAP NetWeaver 7.40, it is part of the standard SAP NetWeaver shipment.

For more information about this class, see in the [SAP Community Wiki](#).

When the ABAP SDK is used for IBM Watson services, you do not need to be concerned with HTTP communication details and ABAP to JSON conversion. The SDK manages those issues by encapsulating all necessary detail operations in the methods that are provided with the SDK.

The HTTP protocol defines status codes to indicate the success of an operation. A status code of 200 is sent upon successful completion of the request. Other status code values indicate some type of abnormal processing, but not necessarily an error. The ABAP REST library provides an exception class `CX_REST_CLIENT_EXCEPTION`, which turns the HTTP status code into an error text.

2.1.3 Network considerations

The communication between SAP NetWeaver ABAP and the internet is processed by the Internet Communication Manager (ICM), which is running in the application server as a separate process. ICM supports HTTP and HTTPS as protocols, and a proxy configuration, if required. It is configured through profile parameters and can be controlled and monitored through the ICM Monitor, which is available in SAP transaction SMICM.

HTTPS is based on the Transport Layer Security (TLS) protocol, also known as Secure Sockets Layer (SSL). SSL employs certificates to encrypt communication. In the SAP system, certificates are maintained through the Trust Manager, which is available in SAP transaction STRUST.

For more information about setting up your network and SAP system, see 2.2.2, “Setting up secure Internet communication” on page 12.

2.1.4 Licensing considerations

When developing and using applications in your SAP system that use IBM Watson services, you must consider the licensing terms and conditions of SAP *and* IBM. Many on-premises SAP applications that are based on the SAP NetWeaver platform include a user-based license model, in which the license costs depend on the number of named users that use the SAP application.

With the growing number of hybrid scenarios in which SAP systems exchange data with non-SAP applications, the concept of licenses that are based on named SAP users reached its limit. Access from a non-SAP front-end system to the SAP business data was often authorized through a *technical* user so that many users accessed the SAP system with only one named SAP user. Because of this issue, SAP introduced the concepts of Indirect Access Licensing and the SAP NetWeaver Foundation for Third-party Applications.

You must check with SAP if the development of your own application is included in your license agreement or requires another license agreement with SAP. Technically, you do not notice whether you violate the license agreement with SAP.

IBM offers various pricing plans for the IBM Watson AI services in the public cloud. Most pricing plans are based on a subscription model in which you pay license fees that are based on the number of calls or amount of data that is being processed.

The IBM Cloud account that owns the service resource is charged with the accumulated fees. If you change your plan, the service resource must be restarted, and you receive new credentials for your service resource.

Some pricing plans, especially in the Premium section, cannot be selected online and require assistance by the IBM sales organization. If your plan does not allow a specific method, or if you exceeded the number of calls that was allowed in your plan, you receive an HTTP status code of 403: Forbidden.

2.1.5 Administering credentials

Credentials are assigned when you create an IBM Watson service resource with your account. Several sets of credentials can exist for a single service resource.

Through the credentials that are used when connecting to a service, fees are charged to the account that owns the service resource. You do not need individual credentials for each SAP application user, but you can use one set of credentials for all your SAP application users.

If you want to split up fees (for example, between different departments), you can create separate accounts for each department. In each account, you create a service resource with its own separate credentials. Based on the application user's department, you then pick the credentials for the department's service resource.

Although the service credentials do not include sensitive, personal data, they must be kept secret to avoid unauthorized access to IBM Watson services, which can produce costs for the owner of the service.

However, you do not want to require each user of your application to memorize the credentials and enter them when accessing the service. Therefore, you must store the credentials somewhere in your system.

Because the credentials are processed in ABAP in clear text, anyone with debug authority in the SAP system can see the credentials in the debugger. If you store the credentials in an unencrypted format in a file or database table, anyone with access to that file or table can access the data.

For legal reasons, you cannot use the secure storage that is provided by SAP to store the credentials in an encrypted way. Instead, you can use the Secure Store & Forward (SSF) mechanism with a third-party security product. However, even if you encrypt the credentials externally before saving them, it is still possible to see them in clear text in the ABAP debugger.

Consider regenerating new credentials periodically to reduce the opportunity of credential fraud.

2.1.6 ABAP SDK for IBM Watson

Various SDKs are offered to use the IBM Watson services with your own applications. The following programming environments are supported:

- ▶ Android
- ▶ Go
- ▶ Java
- ▶ Node.js
- ▶ Python
- ▶ Ruby
- ▶ .NET
- ▶ Salesforce
- ▶ Swift
- ▶ Unity

In 2019, an SDK for SAP ABAP was added into the section Community SDKs. The SDKs are available at [this web page](#).

The ABAP SDK for IBM Watson provides a set of ABAP classes that hide complexity when accessing IBM Watson services. One ABAP class is available for each supported service and one ABAP method for each corresponding method of the service.

The SDK also provides ABAP fields, data structures, and internal tables to hold the data that is sent to or received from the service. The conversion between ABAP data types and the JSON format and the execution of the REST calls is done within the classes and is hidden from the user. The ABAP SDK also provides token management for authentication, which is described 2.4.4, “Identity and Access Management authentication” on page 34.

2.2 SAP system configuration

To install and use the ABAP SDK for IBM Watson, you must configure your SAP system so that it supports the secure HTTPS protocol. This includes the configuration of the Internet Communication Manager (ICM) through SAP profile parameters, the installation of certificates through the SAP Trust Manager and, if required by your network configuration, the definition of a proxy server. In this section, we describe the necessary configuration steps.

2.2.1 SAP profile parameters

To use ABAP SDK for IBM Watson services, the SAP profile parameters that are shown in Table 2-2 are recommended in the application server.

Table 2-2 SAP recommended profile parameters

Parameter	Recommended value
icm/HTTPS/client_sni_enabled	TRUE
ssl/ciphersuites	135:PFS:HIGH::EC_P256:EC_HIGH
ssl/client_ciphersuites	150:PFS:HIGH::EC_P256:EC_HIGH
wdisp/ssl_ignore_host_mismatch	TRUE

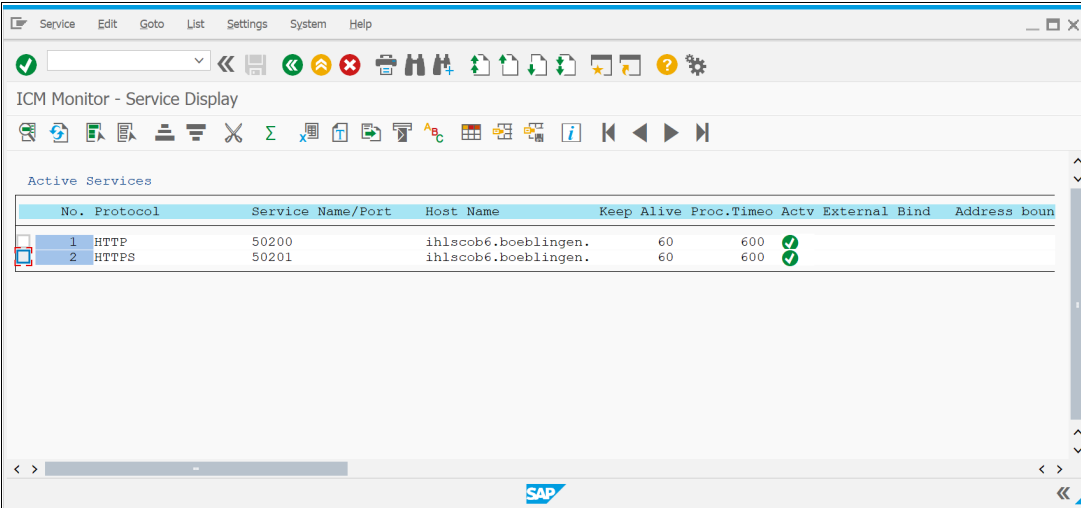
2.2.2 Setting up secure Internet communication

Secure internet communication is used during the installation of the ABAP SDK for IBM Watson and when connecting to IBM Watson services through the ABAP SDK. Secure communication is established by way of the encrypted HTTPS protocol.

HTTPS is based on the Transport Layer Security (TLS) protocol, also known as Secure Sockets Layer (SSL). SSL uses certificates to encrypt communication. Therefore, you must ensure that HTTPS is enabled in your SAP system and that all required certificates are installed on the SAP application server in the Personal Security Environment (PSE).

Enabling HTTPS

You must verify that the HTTPS service were started on your SAP system. To display the status of the services in your SAP system, run transaction SMICM and select **Goto** → **Services**. A window opens that is similar the example that is shown in Figure 2-3.



No.	Protocol	Service Name/Port	Host Name	Keep Alive	Proc.Timeo	Actv	External Bind	Address bound
1	HTTP	50200	ihlscob6.boeblingen.	60	600	✓		
2	HTTPS	50201	ihlscob6.boeblingen.	60	600	✓		

Figure 2-3 ICM Monitor: Service Display

If the HTTPS service is unavailable, you can configure it by selecting **Service** → **Create**. Alternatively, you can set the following SAP profile parameter:

```
icm/server_port_<n> = PROT=HTTPS,PORT=443$$,PROCTIMEOUT=600,TIMEOUT=60
```

Replace <n> with the next available number. For example, if profile parameters icm/server_port_1 and icm/server_port_2 are defined, choose, icm/server_port_3.

Setting up SSL certificates

To use the encrypted HTTPS protocol, you also must install the required SSL certificates in your SAP system. The certificates can be downloaded from trusted sources on the internet.

In the SAP system, certificates are maintained through the Trust Manager (transaction STRUST). If your system was configured to use SSL before, you likely need only to install the DigiCert Global Root CA certificate. Otherwise, you must check whether specific personal security environments (PSE) exist and create them if necessary, in particular the System PSE and the SSL Client PSE.

Downloading required certificates

Download the following certificates and save them locally:

- ▶ [DigiCert High Assurance EV Root CA](#) (needed during installation of the ABAP SDK)
- ▶ [DigiCert SHA2 Extended Validation Server CA](#) (needed during installation of the ABAP SDK)
- ▶ [DigiCert Global Root CA](#) (needed when accessing IBM Watson services through the ABAP SDK)

If the direct download links do not work, see [this web page](#), search for the certificates by the provided certificate name, right-click the download link, and then, save the certificate on the local file system.

Installing required certificates

To install the downloaded certificates, you need to add them to your SAP system's Standard SSL Client Personal Security Environment (PSE). Complete the following steps:

1. Call transaction STRUST.
2. Switch to edit mode (click the toolbar).
3. If a local PSE file does not exist (which is indicated by **X**), create it by right-clicking **SSL client SSL Client (Standard)** → **Create**, as shown in Figure 2-4.

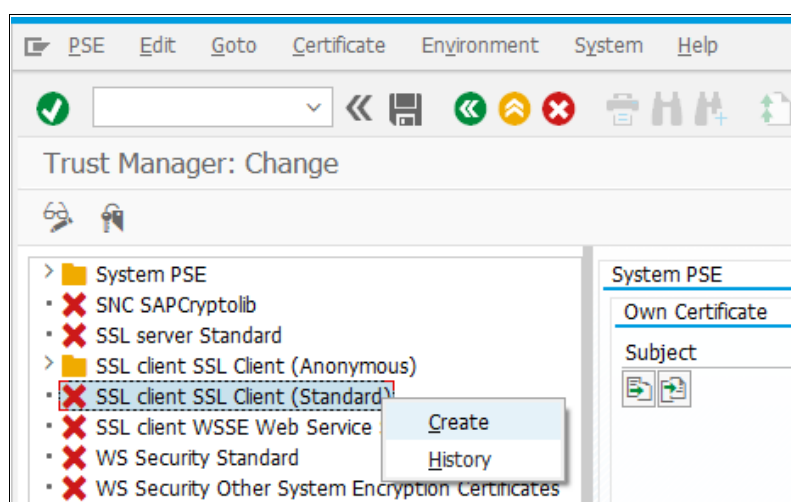


Figure 2-4 Create local PSE file

4. Keep all default settings that are shown in the next window and press **Enter** or click **Continue** to create the standard SSL client PSE.
5. Click **Import certificate** in the Certificate section of the standard SSL client PSE. Alternatively, select **Certificate** → **Import** (see Figure 2-5).

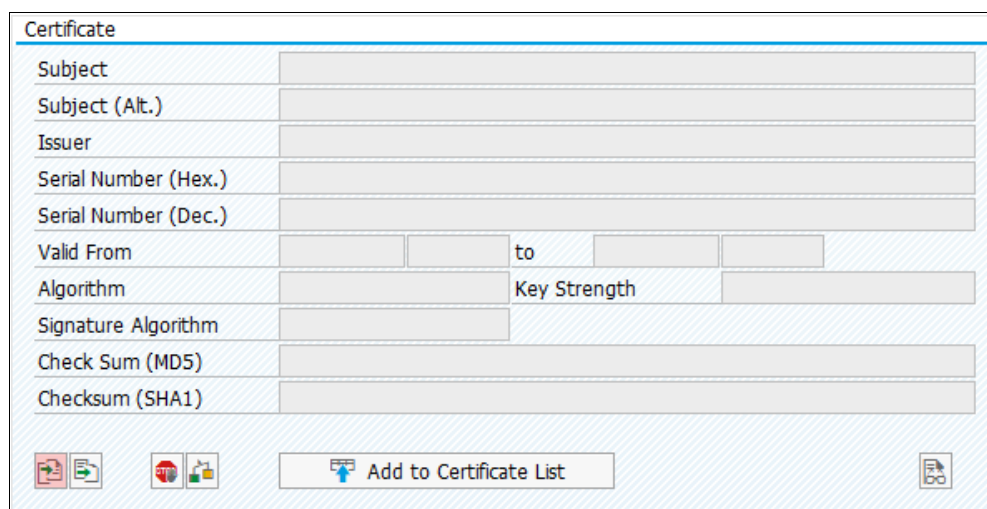





Figure 2-5 Import certificate

6. Choose the certificate file that you downloaded and import the certificate.

7. Add the certificate to the certificate list by clicking **Add to Certificate List**, as shown in Figure 2-6.

Certificate			
Subject	CN=DigiCert High Assurance EV Root CA, OU=www.digicert.com, O=Digi...		
Subject (Alt.)			
Issuer	CN=DigiCert High Assurance EV Root CA, OU=www.digicert.com, O=Digi...		
Serial Number (Hex.)	02:AC:5C:26:6A:0B:40:9B:8F:0B:79:F2:AE:46:25:77		
Serial Number (Dec.)	3553400076410547919724730734378100087		
Valid From	10.11.2006 00:00:00	to	10.11.2031 00:00:00
Algorithm	RSA	Key Strength	2048
Signature Algorithm	RSA+SHA1		
Check Sum (MD5)	D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A		
Checksum (SHA1)	5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25		

 Add to Certificate List




Figure 2-6 Add to certificate list

8. Click **Save** or press **F3** \ to save the newly added certificate.
9. Repeat steps 5- 8 to create the two other certificate files.

The certificate list of the standard SSL client PSE contains the newly imported certificates, as shown in Figure 2-7.

Certificate List	
	<div style="border: 1px solid #ccc; padding: 5px;"> <div style="background-color: #f0f0f0; padding: 2px;">Subject</div> <div style="padding: 2px;">CN=DigiCert High Assurance EV Root CA, OU=www.digicert.com, O=..</div> <div style="padding: 2px;">CN=DigiCert SHA2 Extended Validation Server CA, OU=www.digicert..</div> <div style="padding: 2px;">CN=DigiCert Global Root CA, OU=www.digicert.com, O=DigiCert Inc.,</div> </div>




Figure 2-7 Certificate list

Restarting the Internet Communication Manager (ICM)

It is recommended to restart the ICM after a new SSL certificate is applied to the PSE.
Complete the following steps:

1. Call transaction SMICM.
2. Select menu item Administration → ICM → Restart → Yes as shown in Figure 2-8 on page 16.

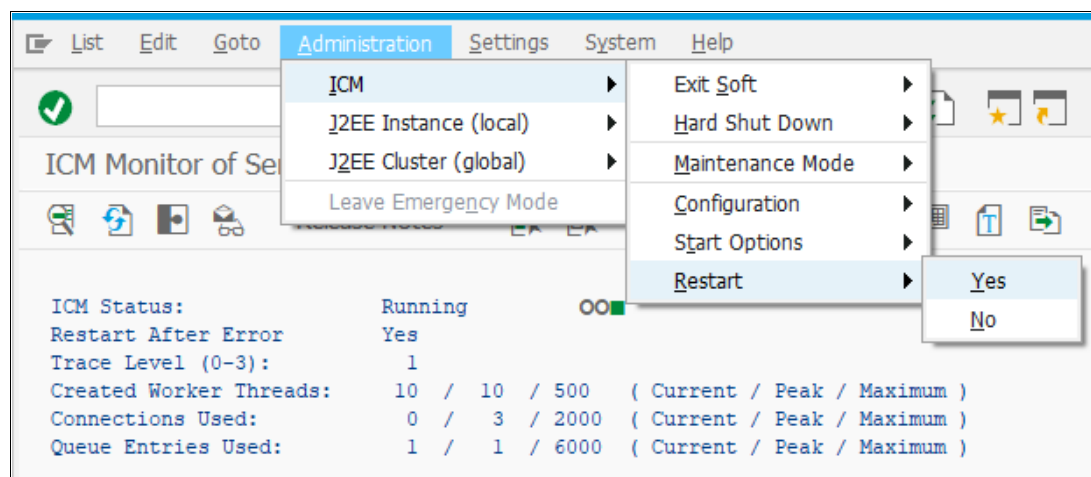


Figure 2-8 Restart ICM

Determining required certificates (identify missing certificates)

It is possible that in the future, more or other certificates are needed to install the ABAP SDK for IBM Watson or connect to IBM Watson services through the SDK.

If a call to an IBM Watson service fails because of a missing SSL certificate, you can check the ICM trace file to find the required certificate. To do so, call transaction SMICM and select **Goto → Trace File → Display End**.

Alternatively, call transaction AL11 and browse to file DIR_HOME/dev_icm. Find the last entry in trace file that indicates error SSSLERR_PEER_CERT_UNTRUSTED, as shown in Figure 2-9.

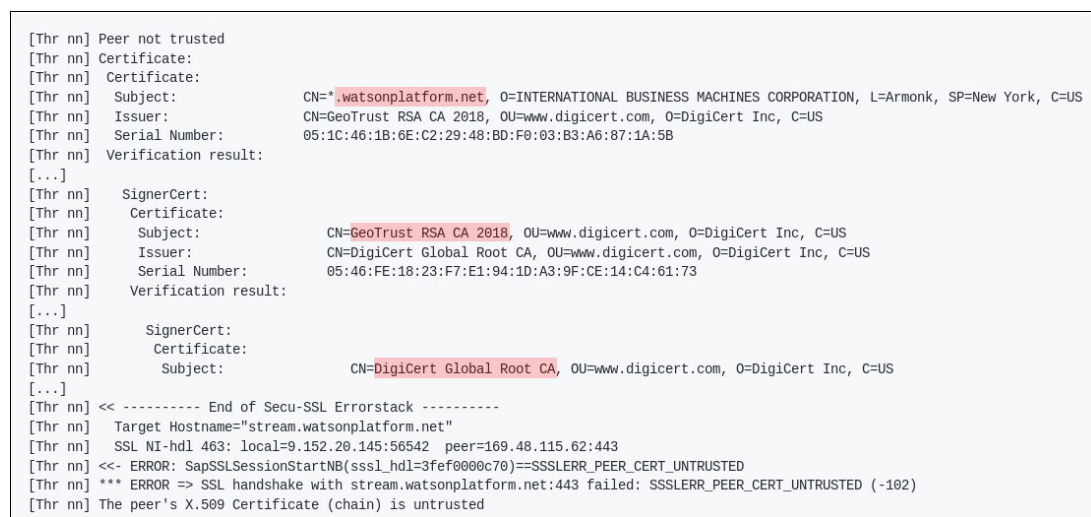


Figure 2-9 Certificate errors in trace file

The SSL error stack shows the SSL certificate chain, which in Figure 2-9 looks as listed in Table 2-3.

Table 2-3 SSL certificate chain

Certificate authority type	Certificate authority
End-user	*.watsonplatform.net
Intermediate CA	GeoTrust RSA CA 2018
Root CA	DigiCert Global Root CA

Download the certificates from a trusted source and install them as described in “Setting up SSL certificates” on page 13.

Configuring proxy server

If your network configuration uses a proxy server, you can configure your SAP system to use the proxy server. To define the proxy server in your SAP system, complete the following steps:

1. Call transaction SICF and click **Execute** (or press **F8**), as shown in Figure 2-10.

Figure 2-10 SICF entry window

2. Select **Client** → **Proxy Settings**, as shown in Figure 2-11.

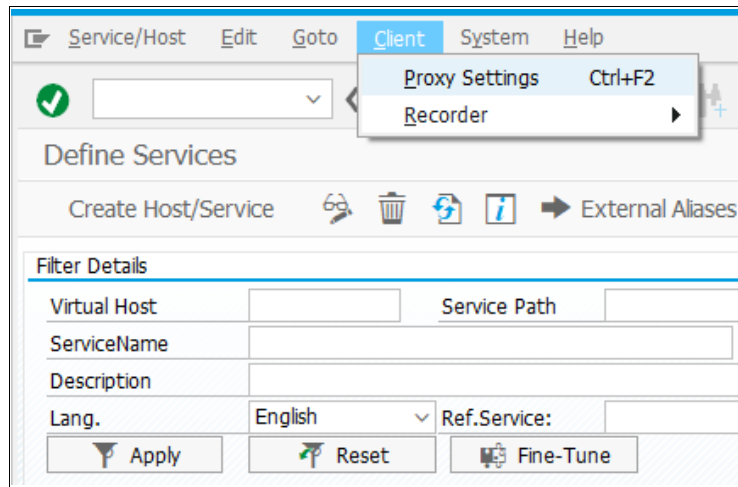


Figure 2-11 Jump to proxy settings

3. Complete the following steps in the Proxy Configuration for HTTP Client window:
- On the Global Settings tab, select the **Proxy Setting is Active** and **No Proxy Setting for Local Server** options, as shown in Figure 2-12.

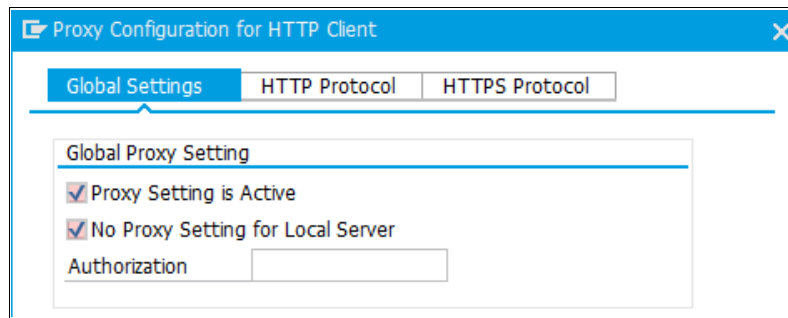
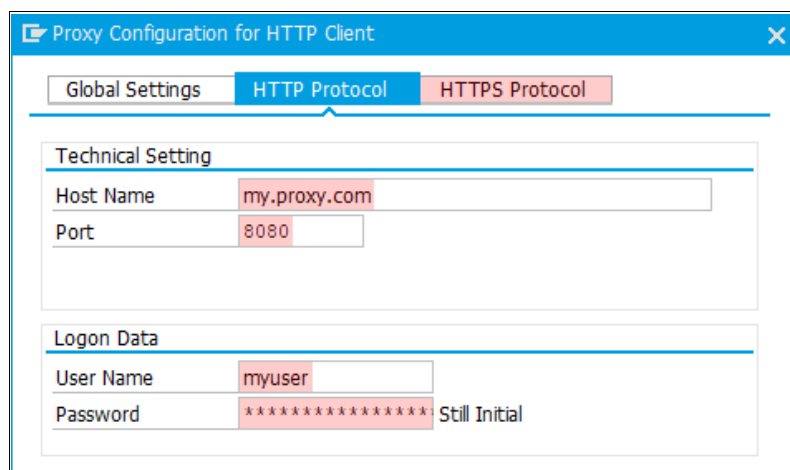


Figure 2-12 Activate proxy

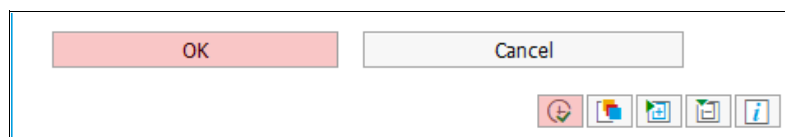
- b. On the HTTP Protocol and HTTPS Protocol tabs, specify the proxy Host Name and Port. If the proxy server requires log-in credentials, also specify the User Name and Password, as shown in Figure 2-13.



The image shows a dialog box titled "Proxy Configuration for HTTP Client". It has three tabs: "Global Settings", "HTTP Protocol", and "HTTPS Protocol". The "HTTP Protocol" tab is selected. Inside the dialog, there are two sections: "Technical Setting" and "Logon Data". In the "Technical Setting" section, the "Host Name" field contains "my.proxy.com" and the "Port" field contains "8080". In the "Logon Data" section, the "User Name" field contains "myuser" and the "Password" field contains "*****" with the text "Still Initial" to its right.

Figure 2-13 Proxy settings

4. Click **Execute** (or press **F8**) and then, click **OK** (see Figure 2-14) to save the changed settings.



The image shows a small confirmation dialog box with two buttons: "OK" and "Cancel". Below the buttons is a row of five small icons: a red circle with a white 'X', a blue square with a white 'X', a green square with a white 'X', a yellow square with a white 'X', and a blue square with a white 'X'.

Figure 2-14 Confirmation of proxy settings

Debugging network communication

If communication errors occur, you can see detailed error messages in the ICM trace file `dev_i cm`, which is in the work directory of the instance. To display the trace file, run transaction **SMICM** (ICM Monitor) and click **Goto** → **Trace File** → **Display All**.

For enhanced debugging of your own application, you can lock an SAP work process exclusively and increase the trace level for component ICF to 3. At that level, all communication between the application server and the IBM Watson service is logged in the ICM trace file. To increase the trace level for your tests, run ABAP program **RSTRC000** through transaction **SE38** and make the following selections (as shown in Figure 2-15):

- ▶ Increase the trace level value to 3.
- ▶ Mark the field **Keep Work process** with the letter **x**.
- ▶ Clear the trace components **Taskhandler** and **VM Container**.
- ▶ Select the trace component **ICF**.
- ▶ Click the **Save** icon or press **F5** to activate your changes.

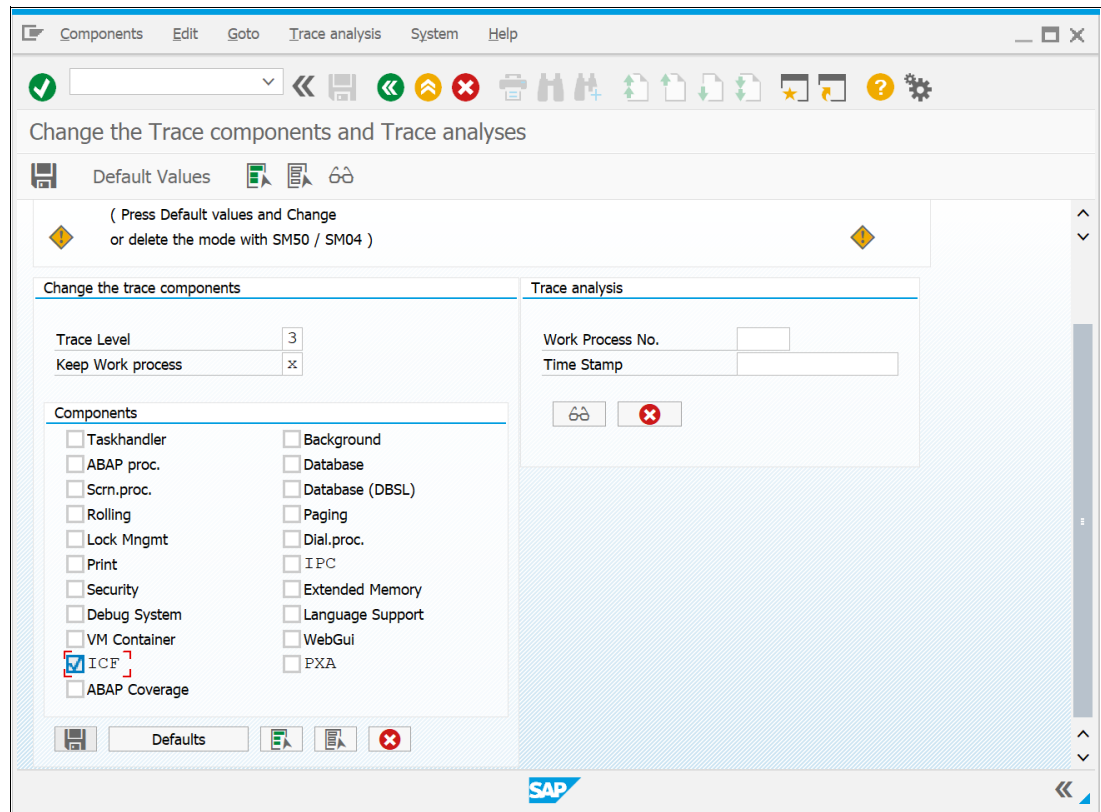


Figure 2-15 Activate trace level 3 for the Internet communication

If you now run your application in the same GUI window, all network traffic between your application and the internet are logged in the ICM trace file. To reset the trace level to its defaults, run ABAP program RSTRC000 in the same GUI session again. Then, click **Defaults** and then, click the **Save** icon.

2.3 Installing the ABAP SDK for IBM Watson

The ABAP SDK for IBM Watson is provided under an open source license (Apache License 2.0) in GitHub. For technical reasons, it is available in the following versions:

- ▶ [ABAP SDK for SAP NetWeaver 7.50 and higher](#), such as SAP ECC or SAP S/4HANA on-premises
- ▶ [ABAP SDK for the SAP Cloud Platform ABAP Environment](#) ("Steampunk")

From a programmer's perspective, both versions provide the same functions and interfaces. The system setup is different for the two versions. All the information that is in this document is related to the ABAP SDK for SAP NetWeaver 7.50 and higher, such as SAP ECC or SAP S/4HANA on-premises.

2.3.1 Introduction to abapGit

To install the ABAP SDK for IBM Watson in an SAP system, the abapGit¹ tool is used. The abapGit tool is a Git² client for ABAP and offers an alternative to the traditional SAP transport system to import packages into an SAP system. All relevant information about an SAP package (including its source code) is stored in a Git repository; for example, on GitHub³.

The abapGit tool can be configured to remotely connect to a Git repository, pull the repository's content into the target SAP system, and install it into a package on the target system.

The abapGit tool is an open source tool and is freely available on GitHub at this [GitHub web page](#). The documentation is accessible at [this web page](#).

For Application Development Tools (ADT) in Eclipse users, an abapGit plug-in for Eclipse is available at [this web page](#). The use of ADT and the plug-in is required for the SCP ABAP version of the SDK. For the on-premises versions of the SDK, the SAP GUI-based abapGit version provides more functions and is preferred.

Installing abapGit

Complete the following steps to install abapGit:

1. Call transaction SE38 and create report ZABAPGIT_FULL, as shown in Figure 2-16.

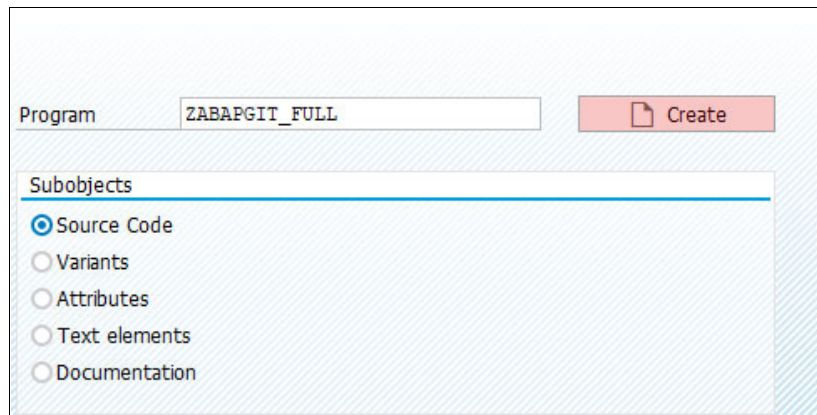


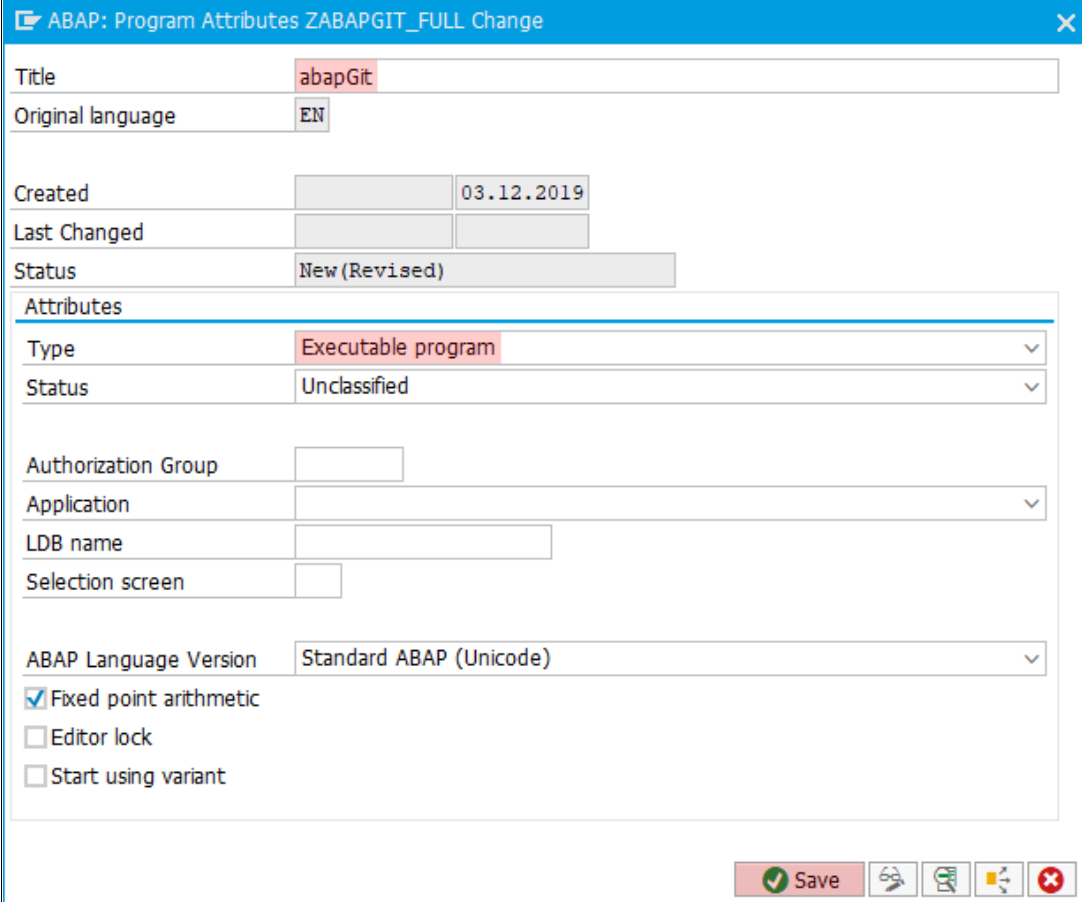
Figure 2-16 Creating full report ZABAPGIT_FULL

2. In the ABAP: Program Attributes pop-up window, choose **abapGit** as the Title, **Executable Program** as the Type, and leave the rest of the fields unchanged. Click **Save**, as shown in Figure 2-17 on page 22.

¹ <http://www.abapgit.org>

² <https://git-scm.com>

³ <https://github.com>



ABAP: Program Attributes ZABAPGIT_FULL Change

Title: **abapGit**

Original language: **EN**

Created: **03.12.2019**

Last Changed:

Status: **New (Revised)**

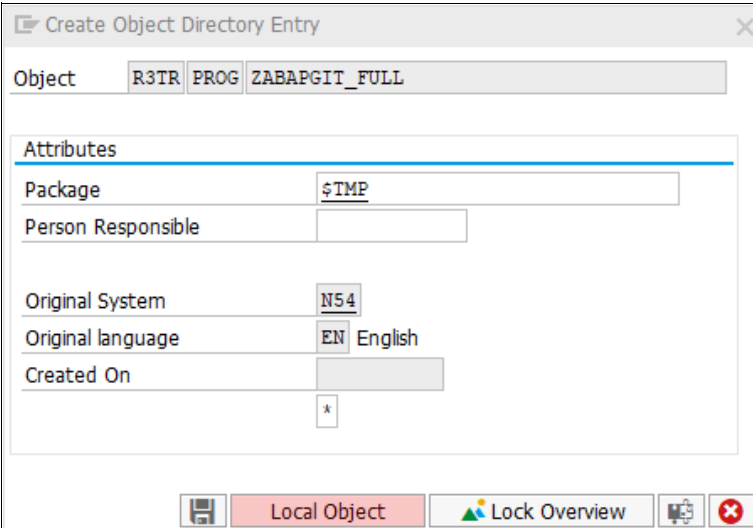
Attributes

Type	Executable program
Status	Unclassified
Authorization Group	
Application	
LDB name	
Selection screen	
ABAP Language Version	Standard ABAP (Unicode)
<input checked="" type="checkbox"/> Fixed point arithmetic	
<input type="checkbox"/> Editor lock	
<input type="checkbox"/> Start using variant	

Save

Figure 2-17 Attributes for program ZABAPGIT_FULL

3. In the Create Object Directory Entry window, click **Local Object** to create the report, as shown in Figure 2-18.



Create Object Directory Entry

Object: **R3TR** **PROG** **ZABAPGIT_FULL**

Attributes

Package: **\$TMP**

Person Responsible:

Original System: **N54**

Original language: **EN** English

Created On:

*

Local Object

Lock Overview

Figure 2-18 Package selection for program ZABAPGIT_FULL

An empty editor window opens, as shown in Figure 2-19.

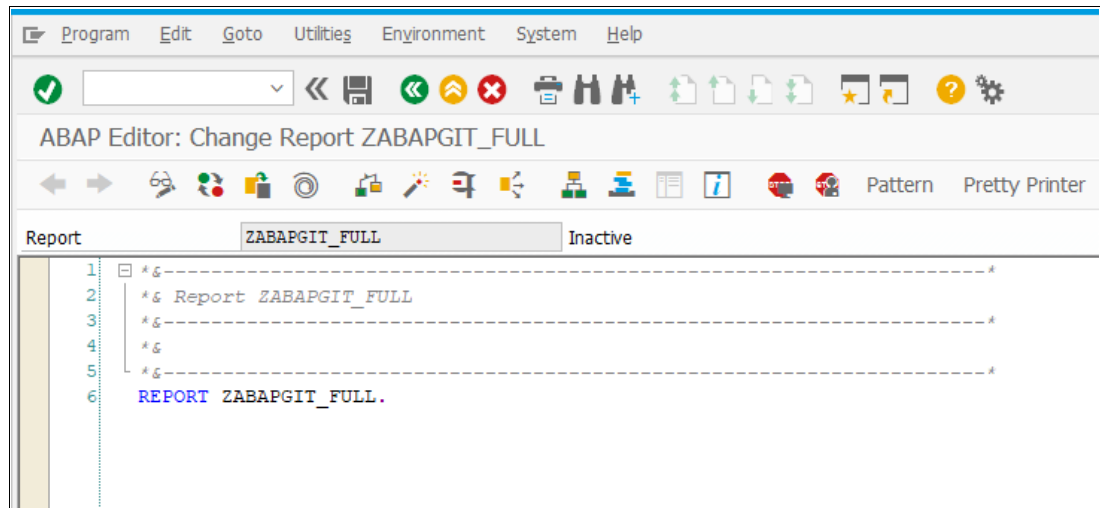


Figure 2-19 ABAP editor for ZABAPGIT_FULL

4. Download [the abapGit code](#) and save it to a local file.
5. In the editor window, click **Utilities** → **More Utilities** → **Upload/Download** → **Upload** and upload the downloaded abapGit code into the editor, as shown in Figure 2-20.

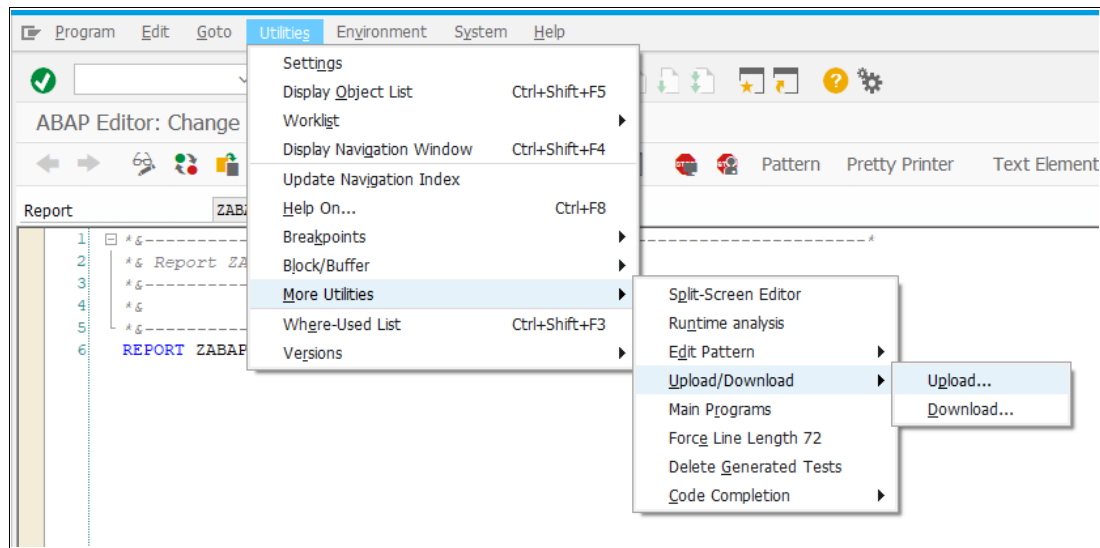


Figure 2-20 ABAP editor upload function

6. Save, activate, and run the ZABAPGIT_FULL report, as shown in Figure 2-21.

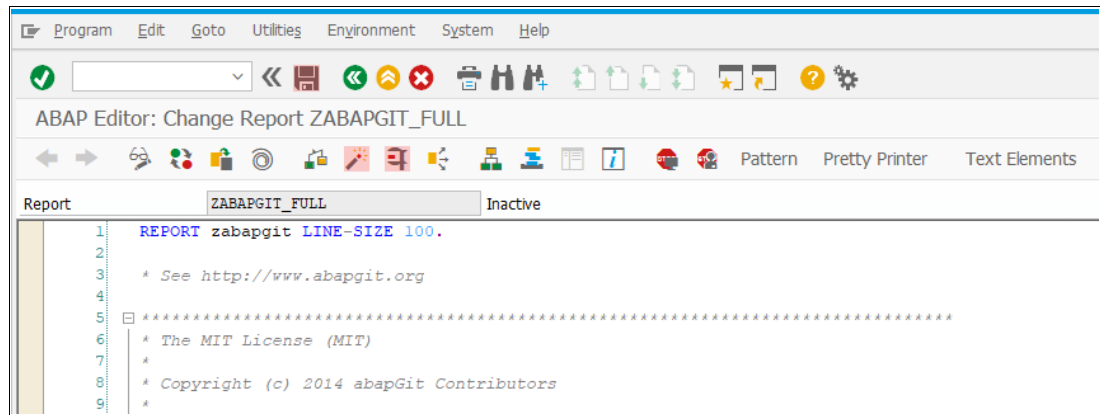


Figure 2-21 Final program ZABAPGIT_FULL

7. The abapGit home window is displayed, as shown in Figure 2-22.



Figure 2-22 abapGit entry window

Updating the abapGit tool to the latest release level

Before importing the ABAP SDK for IBM Watson, the abapGit tool must be updated to the latest release level. Complete the following steps:

1. Call transaction SE38 and start program ZABAPGIT_FULL, as shown in Figure 2-23.

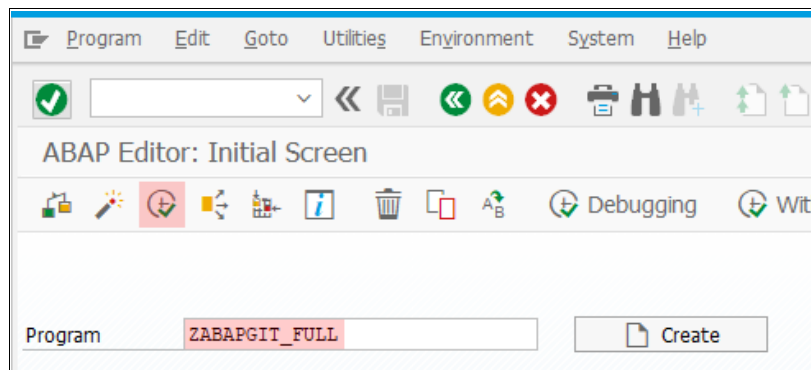


Figure 2-23 ZABAPGIT_FULL execution in SE38

2. Click **+Online** in the abapGit home window, as shown in Figure 2-24.

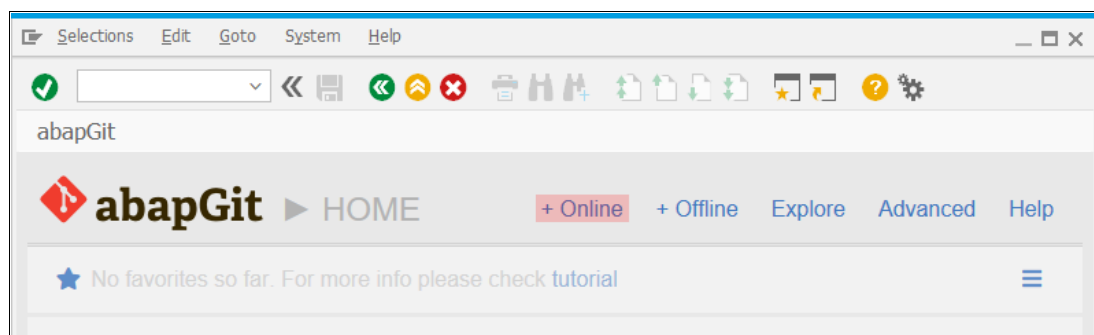


Figure 2-24 Add online project

3. In the New Online Project window, enter the following settings:

- Git clone URL: <https://github.com/larshp/abapGit.git>
- Package: \$ABAPGIT
- Display name: abapGit

Click **Create package** (see Figure 2-25).

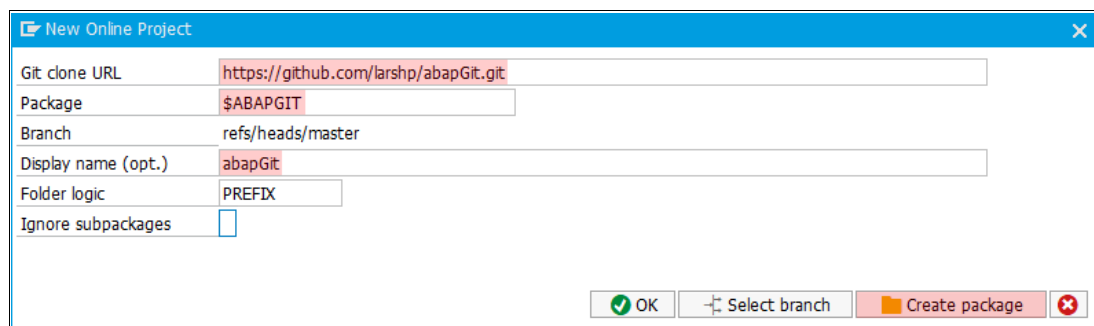


Figure 2-25 Add project abapGit

4. In the Create Package window, set a Short Description to abapGit and click **Continue**, as shown in Figure 2-26.

Figure 2-26 Create \$ABAPGIT

5. In the New Online Project window, click **OK**. abapGit now starts to fetch information about the latest abapGit code from GitHub. An announcement is displayed that can be safely closed by click the **X** in the upper right corner, as shown in Figure 2-27.

Figure 2-27 abapGit change overview

The state of the abapGit installation on your SAP system compared to the state of the abapGit repository at GitHub is displayed.

6. Click **Pull** to download the latest version of the abapGit code to your SAP system, as shown in Figure 2-28.

Figure 2-28 abapGit pull request

It can take several minutes until abapGit finishes downloading the code and installing it into package \$ABAPGIT.

7. In the Inactive Objects window, click **Continue** to activate all of the abapGit objects, as shown in Figure 2-29.

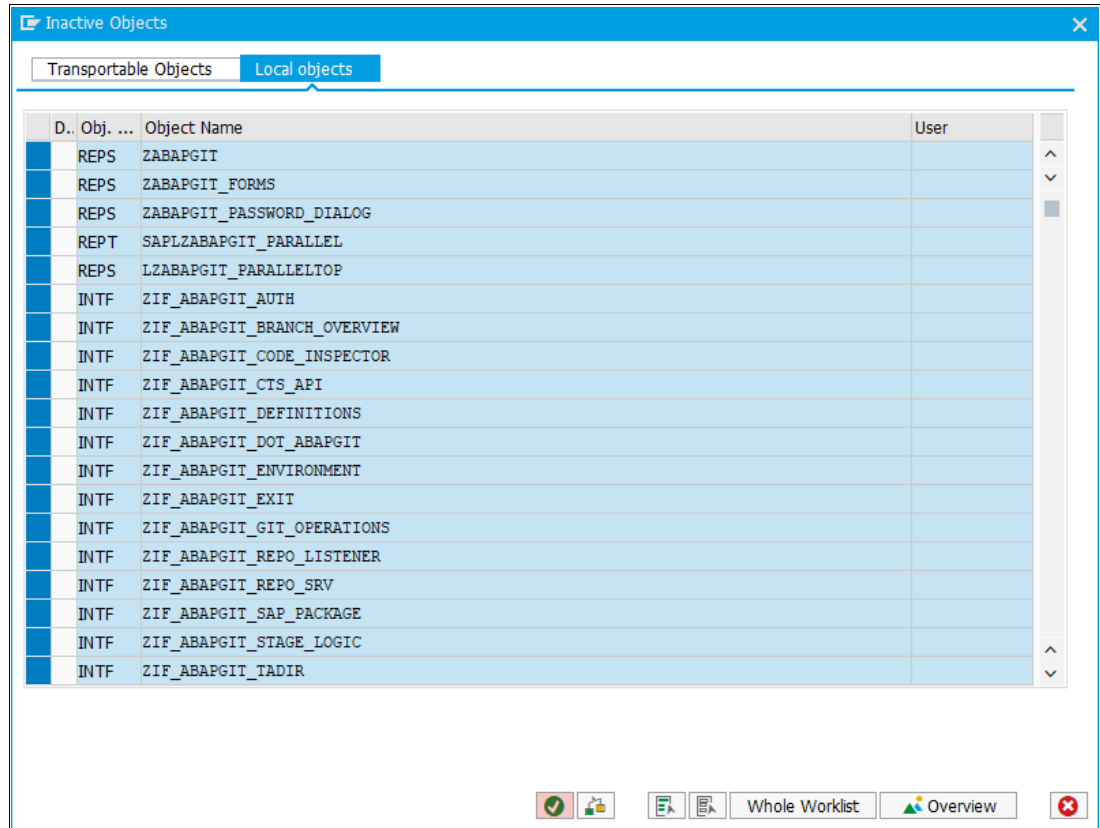


Figure 2-29 abapGit activation window

Again, it can take some time until abapGit finishes activating the objects.

Now, abapGit is installed into package \$ABAPGIT on your SAP system. Also, you do not have to start abapGit by way of SE38; instead, you can call it directly by way of the transaction ZABAPGIT.

Your system is now prepared for installing the ABAP SDK for IBM Watson.

2.3.2 Installing the ABAP SDK for IBM Watson

After abapGit is installed on your SAP system, the ABAP SDK for IBM Watson can be installed on the system by completing the following steps:

1. Call transaction ZABAPGIT and click **+ Online** in the abapGit home window, as shown in Figure 2-30 on page 28.

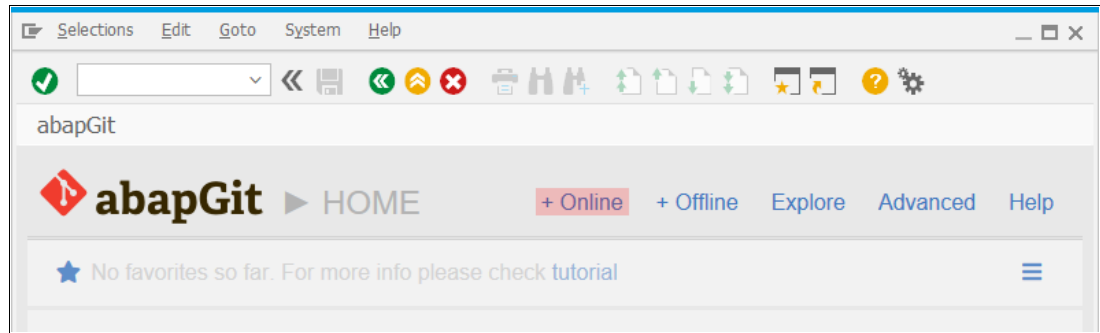


Figure 2-30 Add project abap-sdk-nwas

2. In the New Online Project window, enter the following settings:
 - Git clone URL: `https://github.com/watson-developer-cloud/abap-sdk-nwas.git`
 - Package: ZIBMC
 - Display name: ABAP SDK for IBM Watson

Click **Create package**, as shown in Figure 2-31.

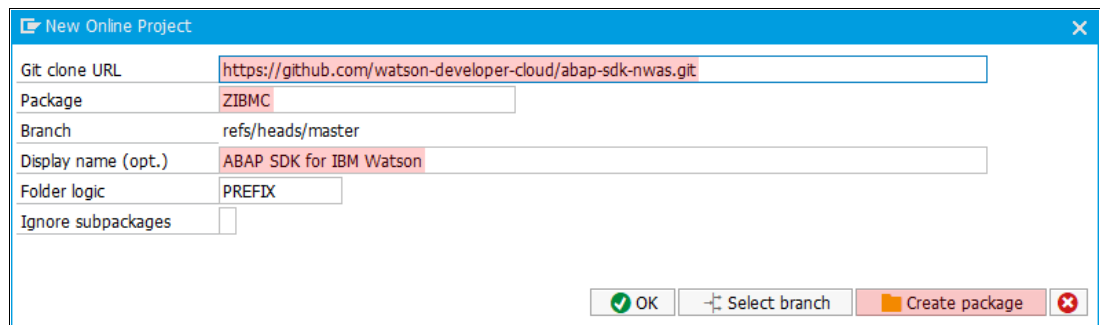


Figure 2-31 Specify project path and package name ZIBMC

3. In the Create Package window, set a Short Description to **ABAP SDK for IBM Watson**, and leave all other fields unchanged. Then, click **Continue**, as shown in Figure 2-32.

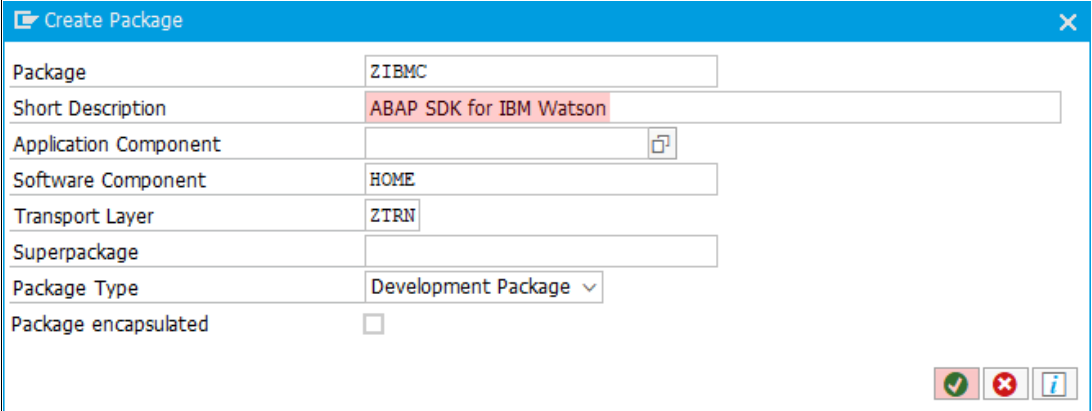


Figure 2-32 shows the 'Create Package' dialog box. The fields are as follows:

Field	Value
Package	ZIBMC
Short Description	ABAP SDK for IBM Watson
Application Component	
Software Component	HOME
Transport Layer	ZTRN
Superpackage	
Package Type	Development Package
Package encapsulated	<input type="checkbox"/>

Figure 2-32 Create package ZIBMC

4. When prompted for a transportable workbench request, create one by clicking **Create Request**, as shown in Figure 2-33.

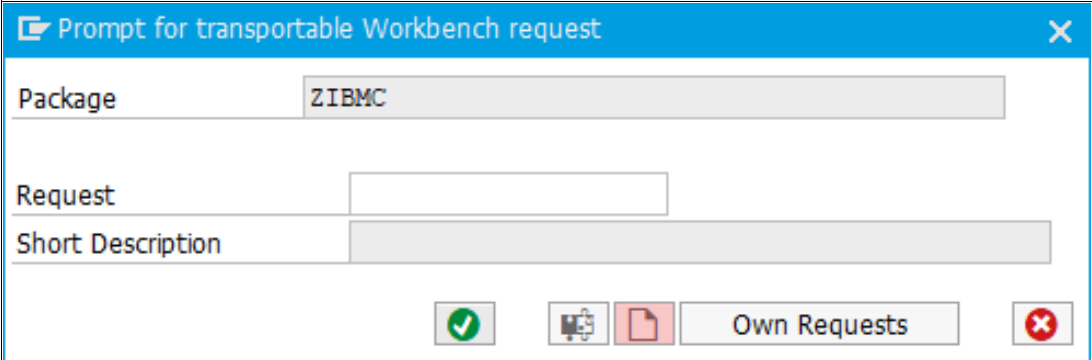


Figure 2-33 shows the 'Prompt for transportable Workbench request' dialog box. The fields are as follows:

Field	Value
Package	ZIBMC
Request	
Short Description	

Figure 2-33 Assign transport request

5. In the Create Request window, set a Short Description to **ABAP SDK for IBM Watson** and leave all other fields unchanged. Click **Save** to create the request, as shown in Figure 2-34.

Figure 2-34 Create new transport request

6. In the Prompt for transportable Workbench request window, click **Continue**, as shown in Figure 2-35.

Figure 2-35 Confirm transport request selection

7. In the New Online Project pop-up window, click **OK**.

abapGit now fetches information about the latest ABAP SDK for IBM Watson code from GitHub.

The state of the ABAP SDK for IBM Watson installation on your SAP system compared to the state of the abapGit repository at GitHub is displayed. Because the SDK is not yet installed on your system, all components are marked as new.

- Click **Pull** to download the latest version of the ABAP SDK for IBM Watson code to your SAP system, as shown in Figure 2-36.

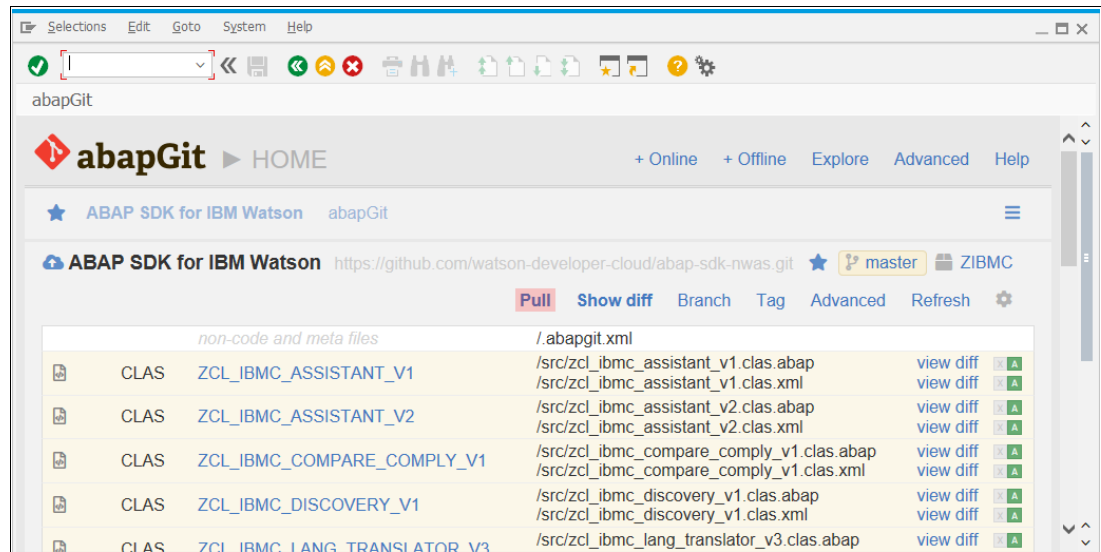


Figure 2-36 abapGit project overview before pull

- When prompted for a workbench request, confirm the request that you created, as shown in Figure 2-37.

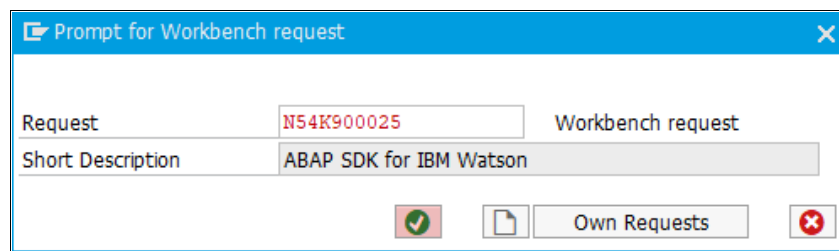


Figure 2-37 Confirm previous transport request selection

It can take some time until abapGit finishes downloading the code and installing it into package ZIBMC.

When abapGit finishes, the ABAP SDK for IBM Watson is ready for use on your system.

2.4 Using the ABAP SDK for IBM Watson

This section describes the use of ABAP SDK for IBM Watson.

2.4.1 ABAP SDK for IBM Watson API overview

The ABAP SDK for IBM Watson is delivered as package ZIBMC. After the Git Repository is cloned to the SAP system, an IBM Watson service instance is wrapped by an ABAP class instance.

Figure 2-38 shows the ABAP SDK for IBM Watson class hierarchy.

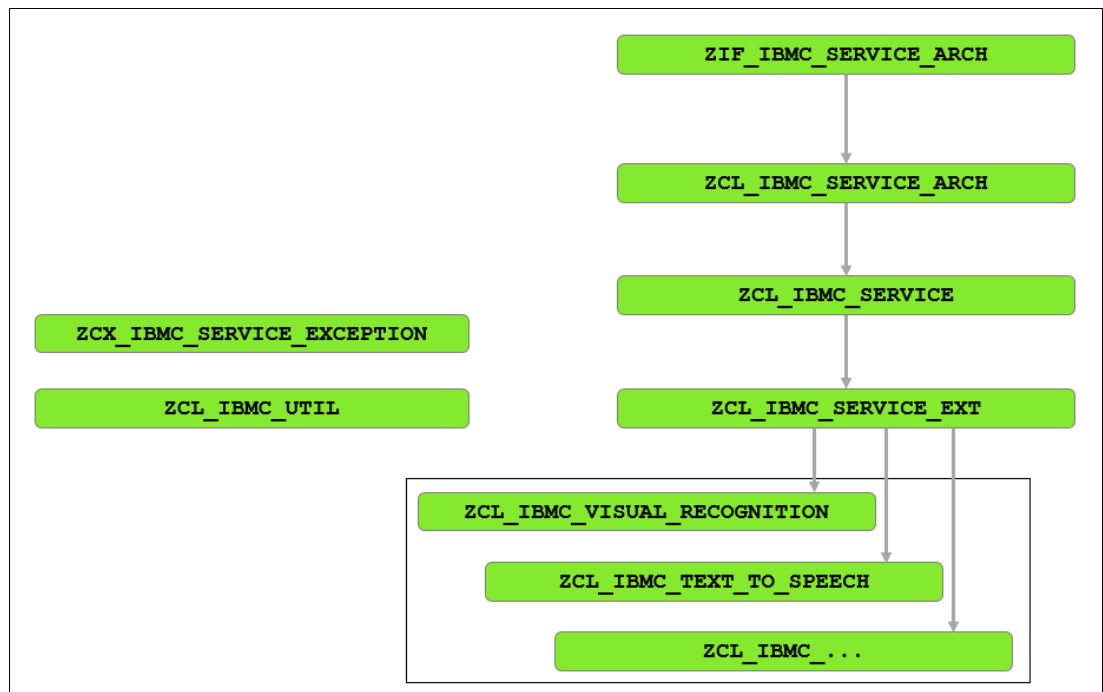


Figure 2-38 IBM Watson SDK class hierarchy

The box in the lower right of Figure 2-38 includes the ABAP classes that map to IBM Watson services. The other classes are base classes in the class hierarchy, helper classes (ZCL_IBMC_UTIL), and exception classes (ZCX_IBMC_SERVICE_EXCEPTION).

Instances of the IBM Watson service classes are created by using the `zcl_ibmc_service_ext → get_instance()` method, which is provided by class ZCL_IBMC_SERVICE_EXT, as described in 2.4.5, “Using classes, methods, and data types of the ABAP SDK” on page 35.

Referring to the service classes (the box in the lower right in Figure 2-38), the IBM Watson services that are listed in Table 2-4 are supported by the ABAP SDK for IBM Watson.

Table 2-4 Supported IBM Watson services with ABAP SDK

Service	ABAP class name
Compare and Comply	ZCL_IBMC_COMPARE_COMPLY_V1
Discovery	ZCL_IBMC_DISCOVERY_V1

Service	ABAP class name
Language Translator	ZCL_IBMC_LANG_TRANSLATOR_V3
Natural Language Classifier	ZCL_IBMC_NAT_LANG_CLASS_V1
Natural Language Understanding	ZCL_IBMC_NAT_LANG_UNDRSTND_V1
Personality Insights	ZCL_IBMC_PERSONAL_INSIGHTS_V3
Speech to Text	ZCL_IBMC_SPEECH_TO_TEXT_V1
Text to Speech	ZCL_IBMC_TEXT_TO_SPEECH_V1
Tone Analyzer	ZCL_IBMC_TONE_ANALYZER_V3
Visual Recognition	ZCL_IBMC_VISUAL_RECOGNITION_V3
	ZCL_IBMC_VISUAL_RECOGNITION_V4
Watson Assistant	ZCL_IBMC_ASSISTANT_V1
	ZCL_IBMC_ASSISTANT_V2

For more information about ABAP classes, see [this web page](#).

2.4.2 Credentials

All supported services support IAM authentication (see 2.4.4, “Identity and Access Management authentication” on page 34). Service credentials consist of an API Key and a URL. The API Key and the URL are character values that can be viewed through the IBM dashboard and must be provided as parameters `i_apikey` and `i_url` to method `zcl_ibmc_service_ext → get_instance()`.

You can store the values with your application, but it is suggested to do so in an encrypted format. The use of cloud services often creates costs that are based on the use for the owner of the service instance. Also, anyone with the credentials can use the service instance at the owner’s expenses. If you want to distribute the costs over multiple cost centers, you must create a service instance and provide service credentials for each cost center separately.

2.4.3 Configuration table

Service credentials and other parameters that must be specified at IBM Watson service wrapper ABAP class instantiation can also be provided in table `ZIBMC_CONFIG`. This table includes the keys that are listed in Table 2-5.

Table 2-5 *ZIBMC_CONFIG table*

Table key	Description
SERVICE	The ABAP class name without prefix <code>ZCL_IBMC_</code> .
INSTANCE_UID	ID chosen by application developer that must be provided by application as parameter to method <code>zcl_ibmc_service_ext → get_instance()</code> .
PARAM	The parameter name.

The list of configuration parameters in table ZIBMC_CONFIG are listed in Table 2-6.

Table 2-6 Configuration parameters

Parameter name	Default value	Description
URL		IBM Watson service URL.
APIKEY		IBM Watson service API keys.
PROXY_HOST		Proxy server (optional).
PROXY_PORT		Proxy server port (optional).
AUTH_NAME	IAM	Authorization, IAM, or basicAuth.
SSL_ID	CLIENT	SSL Identity, defines PSE for SSL certificates: CLIENT or ANONYM.

2.4.4 Identity and Access Management authentication

Identity and Access Management (IAM) is a token-based authentication method. For more information about IAM, see this [IBM Cloud Docs web page](#).

You use the API key to generate a token, which is then used to authenticate the REST calls to the IBM Watson service. The token is valid for 60 minutes. When it expires, you must generate a new one by using the API key again.

Although the ABAP SDK provides a mechanism to manage the token generation and expiration automatically, you can also choose to manage the token manually.

If the `apikey` is provided for method `zcl_ibmc_service_ext=>get_instance()`, the ABAP SDK automatically generates a bearer-token when needed and refreshes it when it is about to expire. This procedure is hidden from the SDK user. You get an instance of the service class as shown in Example 2-1.

Example 2-1 Service class instance

```
data lo_lang_translator type ref to zcl_ibmc_lang_translator_v3.
```

```
zcl_ibmc_service_ext=>get_instance(  
  exporting  
    i_url      = 'https://api.eu-de.language-translator.watson.cloud.ibm.com' &  
                '/instances/2ae347e6-3545-469a-99b3-fbbeadd7afd9'  
    i_apikey   = 'UZ5E17jKADciPoRYIvwjWakFUCysfk-HjRxJa_C8Fdpf'  
    i_version  = '2018-05-01'  
  importing  
    eo_instance = lo_lang_translator ).
```

The method `zcl_ibmc_service_ext=>get_instance()` also supports input parameters `i_username` and `i_password` for Cloud Foundry services. However, all IBM Watson services that are supported by the ABAP SDK support the IAM authentication method; therefore, they are no longer needed.

Note: The API key that is shown in Example 2-1 on page 34 is provided to show the method call only. It is *not* a valid key, and its usage results in an error.

Neither the user name and password nor apikey are provided for method `zcl_ibmc_service_ext=>get_instance()`, the ABAP SDK user must implement an individual token management.

Before a service method is called for the first time, a valid bearer-token must be generated by accessing `https://iam.cloud.ibm.com/identity/token` and providing the API key. The returned token can then be specified in the IBM Watson service wrapper ABAP class instance as shown in the following example:

```
lo_service_class->set_bearer_token( i_bearer_token = '...' ).
```

Where `lo_service_class` is the instance of the class in question that was created by calling `zcl_ibmc_service_ext=>get_instance()` without API key.

Later, service methods can be called if the provided token is valid. When the token expires, the method that is shown must be called again with a new (refreshed) bearer-token as parameter.

2.4.5 Using classes, methods, and data types of the ABAP SDK

By using the IBM Watson Language Translator service as an example, we show some principles when the ABAP SDK is used in an application. The Language Translator service is supported through ABAP class `ZCL_IBMC_LANG_TRANSLATOR_V3` in the current version of the SDK.

The class is providing methods, such as `TRANSLATE`, `IDENTIFY`, or `LIST_IDENTIFIABLE_LANGUAGES` and data types, such as `T_TRANSLATE_REQUEST`, `T_TRANSLATION_RESULT`, or `T_IDENTIFIED_LANGUAGES`.

For more information about the function of the methods and the meaning of the data types, see [the online documentation](#) for the IBM Watson service.

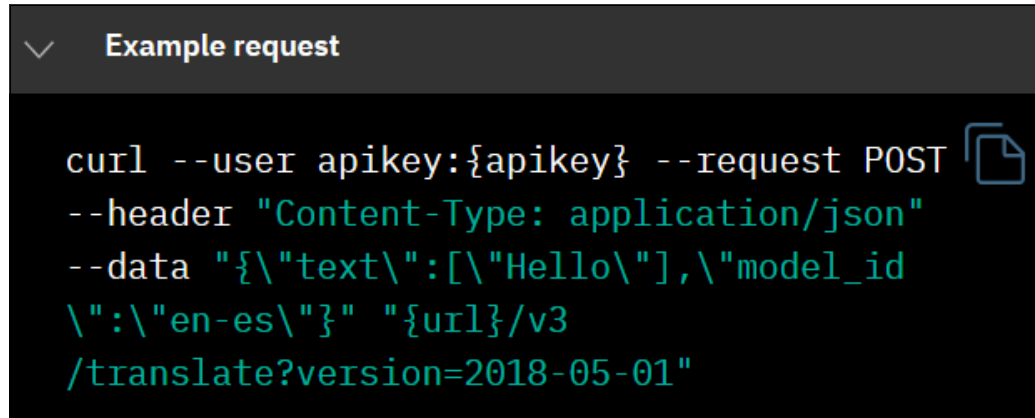
At that web page, you get information about the supported methods, and how to start them. Required and optional parameters, and the format of the response, also are explained, and an example call in different programming languages is provided, as shown in Figure 2-39.

The screenshot displays the IBM Cloud API documentation for the 'Translate' method. The left sidebar shows the navigation menu with categories like Overview, Authentication, Service endpoint, Versioning, Error handling, Data handling, Related information, Methods, Translation, Identification, Models, and Document translation. The main content area is titled 'Translate' and describes the method's purpose: 'Translates the input text from the source language to the target language.' It details the request parameters, including 'version' (string), 'text' (string[]), 'model_id' (string), 'source' (string), and 'target' (string). The response body is shown as 'word_count' (integer). The right sidebar contains a 'Curl' section with an example request and a 'Status 200' response.

Figure 2-39 API documentation for translate method

For each method in the API documentation, one corresponding method in ABAP class ZCL_IBMC_LANG_TRANSLATOR_V3 is available. The method name can be shortened in ABAP because ABAP allows only method names up to 30 characters in length. Typically, an example for the method execution with the expected result is shown to the right of the parameter description. By default, the example is showing curl syntax with the parameters that are provided in JSON notation.

An example call for the Translate method in curl syntax is shown in Figure 2-40.



```
curl --user apikey:{apikey} --request POST
--header "Content-Type: application/json"
--data "{\"text\": [\"Hello\"], \"model_id\": \"en-es\"}" \"{url}/v3/translate?version=2018-05-01"
```

Figure 2-40 Example request

The method expects some text in string format as input. You can provide multiple text strings in an array and receive an array with the translated texts in return. In addition, you can provide a model ID to indicate source and target language of the translation. In Figure 2-40, the model ID “en-es” indicates that a translation from English to Spanish is requested.

In ABAP class ZCL_IBMC_LANG_TRANSLATOR_V3 and the data type T_TRANSLATE_REQUEST is provided with the following declaration:

```
types:
  begin of T_TRANSLATE_REQUEST,
    TEXT type STANDARD TABLE OF STRING WITH NON-UNIQUE DEFAULT KEY,
    MODEL_ID type STRING,
    SOURCE type STRING,
    TARGET type STRING,
  end of T_TRANSLATE_REQUEST.
```

To run the example, you enter the internal table TEXT with the text to be translated and the field MODEL_ID with the requested model. The other fields are optional and can remain empty.

The returned data for the documented example is shown in Figure 2-41.



Figure 2-41 Example response

In ABAP class ZCL_IBMC_LANG_TRANSLATOR_V3, the data type T_TRANSLATION_RESULT is provided with the following declarations:

```
types:
    begin of T_TRANSLATION,
        TRANSLATION type STRING,
    end of T_TRANSLATION.
types:
    begin of T_TRANSLATION_RESULT,
        WORD_COUNT type INTEGER,
        CHARACTER_COUNT type INTEGER,
        TRANSLATIONS type STANDARD TABLE OF T_TRANSLATION WITH NON-UNIQUE DEFAULT KEY,
    end of T_TRANSLATION_RESULT.
```

All classes of the ABAP SDK for IBM Watson services use a common exception class ZCX_IBMC_SERVICE_EXCEPTION. With that exception class, the example from the documentation in ABAP syntax can be implemented as shown in Example 2-2.

Example 2-2 ABAP syntax

```
data:
    lv_apikey          type string value '...', " replace ... with credentials
    lo_lang_translator type ref to zcl_ibmc_lang_translator_v3,
    lo_service_exception type ref to zcx_ibmc_service_exception,
    ls_request         type zcl_ibmc_lang_translator_v3=>t_translate_request,
    lv_text            type string,
    ls_trans           type zcl_ibmc_lang_translator_v3=>t_translation_result.

zcl_ibmc_service_ext=>get_instance(
    exporting
```

```

i_url      = 'https://api.eu-de.language-translator.watson.cloud.ibm.com' &
              '/instances/2ae347e6-3545-469a-99b3-fbbecdd7afd9'
i_apikey   = lv_apikey
i_version  = '2018-05-01'
importing
eo_instance = lo_lang_translator ).

lv_text = 'Hello'.
append lv_text to ls_request-text.
ls_request-model_id = 'en-es'.

try.
  lo_lang_translator->translate(
    exporting
      i_request      = ls_request
      i_contenttype = 'application/json'
    importing
      e_response = ls_trans ).
catch zcx_ibmc_service_exception into lo_service_exception.
  message lo_service_exception type 'E'.
endtry.

```

At times, IBM Watson services require to upload a file or document as input. You must provide an image if you want to perform image recognition. Also, the Language Translation service offers a method to translate complete documents with one service call.

The method Translate Document supports several document types. In the curl example of the API reference documentation, the input data is provided in the Portable Document Format (PDF), as shown in Figure 2-42.

Example request

```

curl --user apikey:{apikey} --request POST
--form "model_id=en-fr" --form
"file=@en.pdf" "{url}/v3
/documents?version=2018-05-01"

```

Figure 2-42 Example for document translation

To provide the document in an ABAP program, you must first upload the document into a field of type XSTRING (for example, with the help of ABAP function module GUI_UPLOAD), which is part of the GUI front-end services. The method call in ABAP for document translation is shown in Example 2-3.

Example 2-3 Method called in ABAP

```
data:
  lo_lang_translator  type ref to zcl_ibmc_lang_translator_v3,
  lo_service_exception type ref to zcx_ibmc_service_exception,
  lv_binarydata       type string,
  lv_model_id         type string,
  ls_responosedoc     type zcl_Ibmc_Lang_Translator_v3=>t_Document_Status.

* lv_binarydata has been filled prior to the method call

lv_model_id = 'en-fr'.
try.
  lo_lang_translator->translate_document(
    exporting
      i_file = lv_binarydata
      i_filename = 'en.pdf'
      i_file_content_type = zif_ibmc_service_arch=>c_mediatype-appl_pdf
      i_model_id = lv_model_id
    importing
      e_response = ls_responosedoc ).
catch zcx_ibmc_service_exception into lo_service_exception.
  message lo_service_exception type 'E'.
endtry.
```

The file name en.pdf has no direct meaning in the method execution. Instead, it is used as place holder. The document \ and the translated document are identified by using a document ID, which is part of the returned structure ls_responosedoc.



Cloud App connecting to SAP

Business processes can be implemented in multiple ways within an SAP environment. This chapter implements an application with the common programming language JavaScript. The application runs outside the SAP system and retrieves operational data that originates from an SAP system. Interfaces for data transfer must be defined and made available in the SAP system.

This chapter includes the following topics:

- ▶ 3.1, “Introduction” on page 42
- ▶ 3.2, “IBM Secure Gateway” on page 42
- ▶ 3.3, “OData and CDS Views” on page 46
- ▶ 3.4, “Building a node.js application on IBM Cloud” on page 51
- ▶ 3.5, “Using OData and IBM Watson services” on page 59

3.1 Introduction

Data from marketing campaigns that were performed by a Portuguese banking institution is used. The data set can be downloaded from the UCI Machine Learning Repository¹. It is assumed that a model was built and trained based on the data. The model is deployed by an IBM Watson Machine Learning service instance, as described in Chapter 4, “Machine Learning on IBM Power Systems” on page 63.

An application Marketing Campaign Advisor is implemented. The application reads all customer data that applies to specified filter criteria from the SAP system and predicts for each record at which probability the according customer subscribes a term deposit. The application calls an SAP OData service to retrieve the customer data and uses the machine learning model that was deployed by the IBM Watson Machine Learning service. It is run locally and as Cloud Foundry application.

3.2 IBM Secure Gateway

The SAP system often is separated from the public internet by a firewall. Therefore, a secured and controllable communication channel is essential. The IBM Secure Gateway establishes a secure connection between the IBM Cloud and the SAP NetWeaver Gateway.

To set up the IBM Secure Gateway, the SAP NetWeaver Gateway host and the port for inbound HTTP requests is needed. To get this information, run transaction SMICM (transaction code that is used for ICM Monitor in SAP) and press **shift+F1** or click **Goto → Services**, as shown in Figure 3-1.

Active Services						
No.	Protocol	Service Name/Port	Host Name	Keep Alive	Proc.Time	Active
<input type="checkbox"/> 1	HTTP	50200	myhost.domain.com	60	600	✓
<input type="checkbox"/> 2	HTTPS	50201	myhost.domain.com	60	600	✓

Figure 3-1 Gateway Services

The IBM Secure Gateway is a service that must be instantiated in the IBM Cloud. Complete the following steps:

1. Log on to the IBM Cloud. Click **Create Resource**, then, click **Integration** in the navigation window on the left. Find and click **Secure Gateway**, as shown in Figure 3-2 on page 43.

¹ UCI - Machine Learning Repository - Bank Marketing Data Set
<https://archive.ics.uci.edu/ml/datasets/Bank+Marketing>

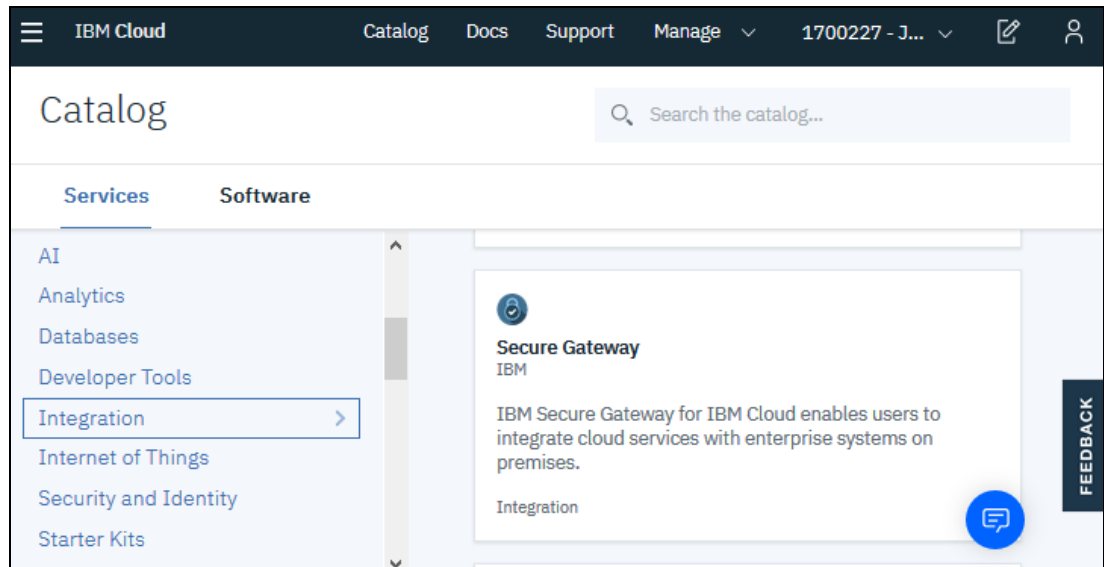


Figure 3-2 IBM Cloud Secure Gateway

2. Choose a pricing plan. You can start with a no-cost plan and update it later, if needed. Click **Create** to instantiate the Secure Gateway service.
3. Click **Add Gateway**. Enter SAPGateway for the Gateway Name and then, click **Add Gateway**, as shown in Figure 3-3.

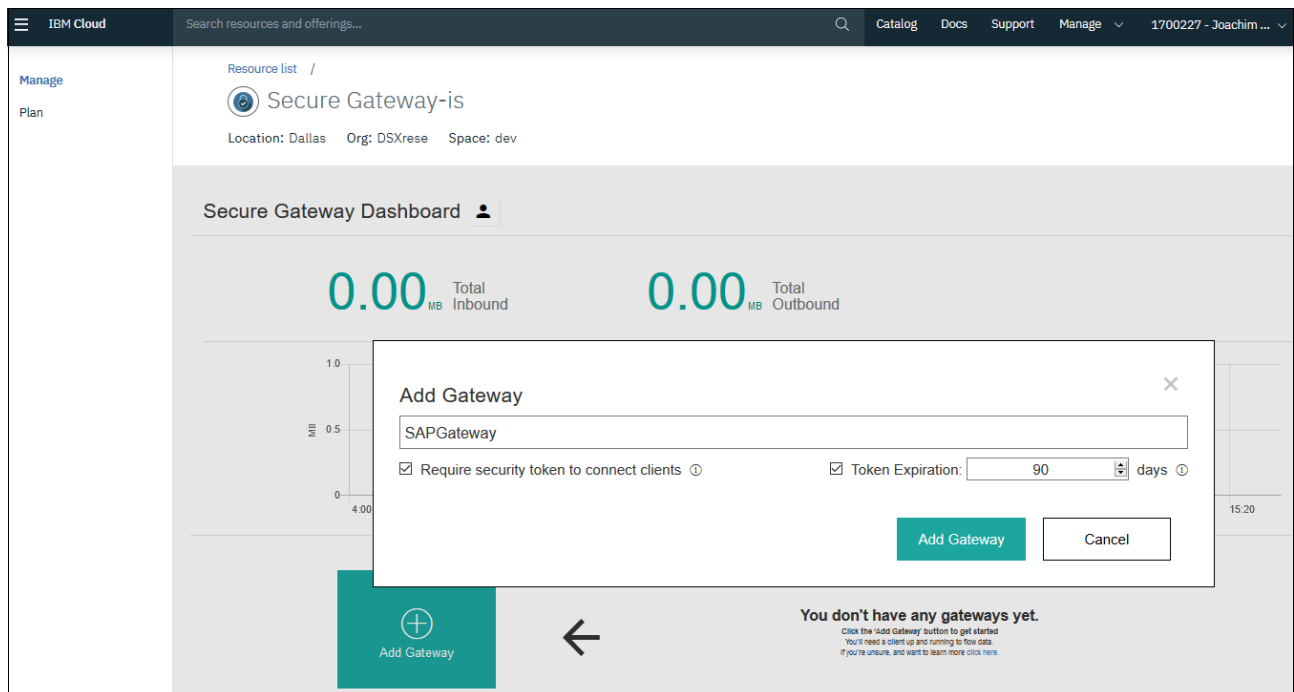


Figure 3-3 Secure Gateway add gateway

4. A destination must be defined:
 - a. On the IBM Secure Gateway Manage view, click **Add Destination** and then, select the **Advanced Setup** tab.
 - b. Select the **On-Premises Destination** option and enter a name for the destination.

- c. Specify **SAP NetWeaver Gateway** as the host name and HTTP port. Select protocol **HTTPS Server Side**. In section TLS options, select **None** for Resource Authentication. Finally, click **Add Destination**, as shown in Figure 3-4.


The screenshot shows the 'Add Destination' dialog box. It has a title bar with a close button. Below the title bar are two tabs: 'Guided Setup' and 'Advanced Setup'. Under 'On-Premises Destination', there are four input fields: 'SAPGatewayDestination' (containing 'SAPGatewayDestination'), 'myhost.domain.com', '50200', and a dropdown menu set to 'HTTPS: Server'. Below these is a 'TLS options' section with 'Resource Authentication' set to 'None'. At the bottom are 'Cancel' and 'Add Destination' buttons.

Figure 3-4 Secure gateway destination


The IBM Secure Gateway service requires a gateway client host. It is recommended that this host is in a firewall configuration for securing local area networks (DMZ) at your site.

5. Click **Add Client**. Download the client package that matches the operating system of the gateway client host. Follow the instructions to install and start the IBM Secure Gateway client on the gateway client. Guidelines are available at [this web page](#).
6. During the installation procedure, you are prompted to provide an access control list (ACL), which is a file. Create an ACL that contains at least the following line, where hostname and port must be adjusted such that it matches the HTTP port and host name of the SAP NetWeaver Gateway:


```
acl allow myhost.domain.com:50200
```
7. The destination is shown on the IBM Secure Gateway Manage view. Click the settings icon (gearwheel) to find the destination URL and port to which the connection to the SAP NetWeaver Gateway is assigned. An access to the destination URL is rerouted to the SAP NetWeaver Gateway, as shown in Figure 3-5 on page 45.


SAPGateway
✕

Destination ID
 wrnd8J6mXHm_YeFTS

Cloud Host : Port
 cap-sg-prd-3.securegateway.appdomain.cloud:17857
 

Resource Host : Port
 myhost.domain.com:50200

Created at
 12/5/2019, 3:46:54 PM

Last modified at
 12/6/2019, 2:17:20 PM

Security
 Protocol: HTTPS
[Download Authentication Files](#)

Edit

Disable

Delete

Figure 3-5 Secure Gateway parameters

- To test the connection, open a web browser and start SAP service /sap/bc/ping by way of the IBM Secure Gateway connection, for example:

`https://cap-sg-prd-3.securegateway.appdomain.cloud:17857/sap/bc/ping`

If the SAP service /sap/bc/ping is activated, the message Server reached is returned. Otherwise, the error message Service cannot be reached is displayed. In both cases, the IBM Secure Gateway connection was set up correctly. If an error message occurs that is not originated by the SAP Internet Communication Manager, check and correct the IBM Secure Gateway setup.

3.3 OData and CDS Views

The application retrieves data that is in the SAP system. Sample data must be created and an OData service to access the sample data must be implemented. Complete the following steps:

1. Start ABAP Development Tools (ADT) in Eclipse and create a project. *System connection* refers to your SAP system.
2. Log on to your system.
3. All ABAP objects must be assigned to a dedicated ABAP Package. Therefore, in ADT, select **File** → **New** → **ABAP** Package and enter ZDEM0 for Name.
4. Click **Finish**.

3.3.1 Sample data table

The customer data records that are to be stored in the SAP System must include fields for all features of the data set that were used for model training. Therefore, create table ZDEMO_MARKETING to store all features (but not the label). Add a column for record number that forms a unique key. Use field names as specified in the training data set; however, add an underscore all field names that are reserved ABAP names.

To allow lowercases for character types, it is recommended to create suitable data types on domains that feature the case-sensitive flag set. For example, create and use three domains and data types of 5, 10, and 15 characters.

The layout looks as shown in Table 3-1.

Table 3-1 Table layout

Field	Key	Data type	Length
RECORDNO		INT4	
AGE		INT4	
JOB		CHAR	15
MARITAL		CHAR	10
EDUCATION		CHAR	15
DEFAULT_		CHAR	5
BALANCE		INT4	
HOUSING		CHAR	5
LOAN		CHAR	5
CONTACT		CHAR	15
DAY_		INT4	
MONTH_		CHAR	10
DURATION		INT4	
CAMPAIGN		INT4	
PDAYS		INT4	

Field	Key	Data type	Length
PREVIOUS		INT4	
POUTCOME		CHAR	10

Populate the table with data. Throughout this document, we assume that table ZDEMO_MARKETING contains the records that are shown in Figure 3-6.

RECORDNO	AGE	JOB	MARITAL	EDUCATION	DEFAULT_	BALANCE	HOUSING	LOAN	CONTACT	DAY_	MONTH_	DURATION	CAMPAIGN	PDAYS	PREVIOUS	POUTCOME
1	18	student	single	secondary	no	40-	no	no	cellular	27	oct	269	1	91	2	success
2	40	technician	married	tertiary	no	4,396	no	no	unknown	18	jun	1,195	2	1-	0	unknown
3	40	technician	married	primary	no	284	yes	no	celluar	20	apr	1,332	2	1-	0	unknown
4	40	technician	married	secondary	no	109	no	no	unknown	20	jun	402	22	1-	0	unknown
5	64	management	married	primary	no	0	yes	no	cellular	16	feb	65	1	1-	0	unknown

Figure 3-6 Table content

3.3.2 Creating Core Data Services View

Customer data that is inserted into table ZDEMO_MARKETING must be made available for access from outside the SAP system. Therefore, an OData service must be created that operates as an interface to the data.

An OData service can be created by using several methods. Operations to read and modify data can be implemented specifically by using ABAP language. Alternatively, an OData service can be generated from an SAP BW query, a BAPI, or a Core Data Services (CDS) view. Use the latter option to generate a suitable OData service.

First, a CDS view must be defined on the data. Complete the following steps:

1. Start ADT.
2. In the project explorer windows, navigate to and right-click package **ZDEMO** and then, select **New** → **Other ABAP Repository Object**.
3. Select **Core Data Services** → **Data Definition**, as shown in Figure 3-7.

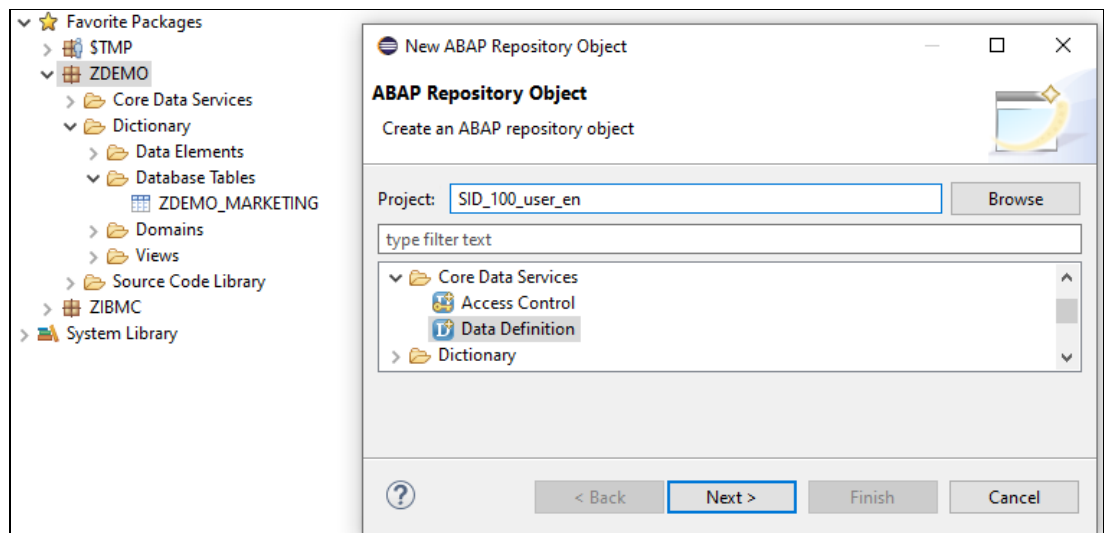


Figure 3-7 ADT new CDS

- Click **Next** and enter ZDEMO_MARKETING_DATA for a name. Enter an arbitrary description and click **Next**, as shown in Figure 3-8.

Figure 3-8 ADT new CDS data definition

- Specify a suitable transport request and then, click **Finish**. The CDS view definition with default annotations appears in the editor window.
- Modify annotations @AbapCatalog.sqlViewName to specify a database view name and add annotation @OData.publish to generate an OData service when the CDS view is activated.
- Add an asterisk (*) as a field list to the view definition. Therefore, the CDS view ZDEMO_MARKETING_DATA features the same layout as underlying table ZDEMO_MARKETING and can be accessed in the same way as the table.

CDS data definition looks as follows:

```
@AbapCatalog.sqlViewName: 'ZDEMO_MARKETINGV'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@endUserText.label: 'CDS View on table ZDEMO_MARKETING'
@OData.publish: true
define view ZDEMO_MARKETING_DATA as select from zdemo_marketing {
    *
}
```

- Activate the CDS view (press **Ctrl+F3**). A database view and an OData service for accessing view ZDEMO_MARKETING_DATA is generated.

3.3.3 Testing the OData service

Complete the following steps to add the OData service on the CDS view ZDEMO_MARKETING_DATA to the catalog of the back-end services:

- Open OData service maintenance (transaction /IWFND/MAINT_SERVICE) from ADT:
 - Select **Run** → **Run ABAP Development Object** (or press **Alt+F8**).
 - Enter search string /IWFND/MAINT_SERVICE and then, select the matching item.

- c. Click **Add Service** and enter ZDEMO* for the Technical Service Name. Press **Enter** to see the search result, as shown in Figure 3-9.

Add Selected Services

Filter

System Alias: LOCAL

Technical Service Name: ZDEMO*

External Service Name:

Co-Deployed: ☐

Version:

External Mapping ID:

Select Backend Services

Type	Technical Service Name	Version	Service Description	External Service Name
BEP	ZDEMO_CDS_CDS	1	CDS View on Bank Marketing Data	ZDEMO_CDS_CDS
BEP	ZDEMO_MARKETING_DATA_CDS	1	CDS View on table ZDEMO_MARKETING	ZDEMO_MARKETING_DATA_CDS
BEP	ZFILE_SRV	1	Upload / Download Test	ZFILE_SRV

Figure 3-9 Add service

- d. Click **ZDEMO_MARKETING_DATA_CDS** and enter ZDEMO for the Package Assignment. Press **Enter** and choose a suitable transport request for all generated objects.
- e. In the OData service maintenance tool (transaction /IWFND/MAINT_SERVICE), mark service ZDEMO_MARKETING_DATA_CDS in the Service Catalog and click **SAP Gateway Client** in the ICF Nodes section, as shown in Figure 3-10.

Activate and Maintain Services

Service Catalog

Type	Technical Service Name	Version	Service Description	External Service Name
BEP	ZDEMO_CDS_CDS	1	CDS View on Bank Marketing Data	ZDEMO_CDS_CDS
BEP	ZDEMO_MARKETING_DATA_CDS	1	CDS View on table ZDEMO_MARKETING	ZDEMO_MARKETING_DATA_CDS
BEP	ZFILE_SRV	1	Upload / Download Test	ZFILE_SRV

ICF Nodes

Status	ICF Node	Session Time-out	Soft State	Description
o	ODATA	00:00:00		Standard Mode

System Aliases

SAP System Alias	Description
LOCAL	Local System Alias

Figure 3-10 OData Service Catalog

2. The SAP Gateway Client can be used to test OData services. For example, enter /sap/opu/odata/sap/ZDEMO_MARKETING_DATA_CDS/\$metadata for the Request URI and click **Execute**. You see the metadata of the Entity Data Model of the OData service, which in this case is equal to the layout of the CDS view.

3. To retrieve all records for customers of age 40 in table ZDEMO_MARKETING_DATA as JSON format, run the following URI, as shown in Figure 3-11:

```
/sap/opu/odata/sap/ZDEMO_MARKETING_DATA_CDS/ZDEMO_MARKETING_DATA?$filter=AGE eq 40&$format=json
```

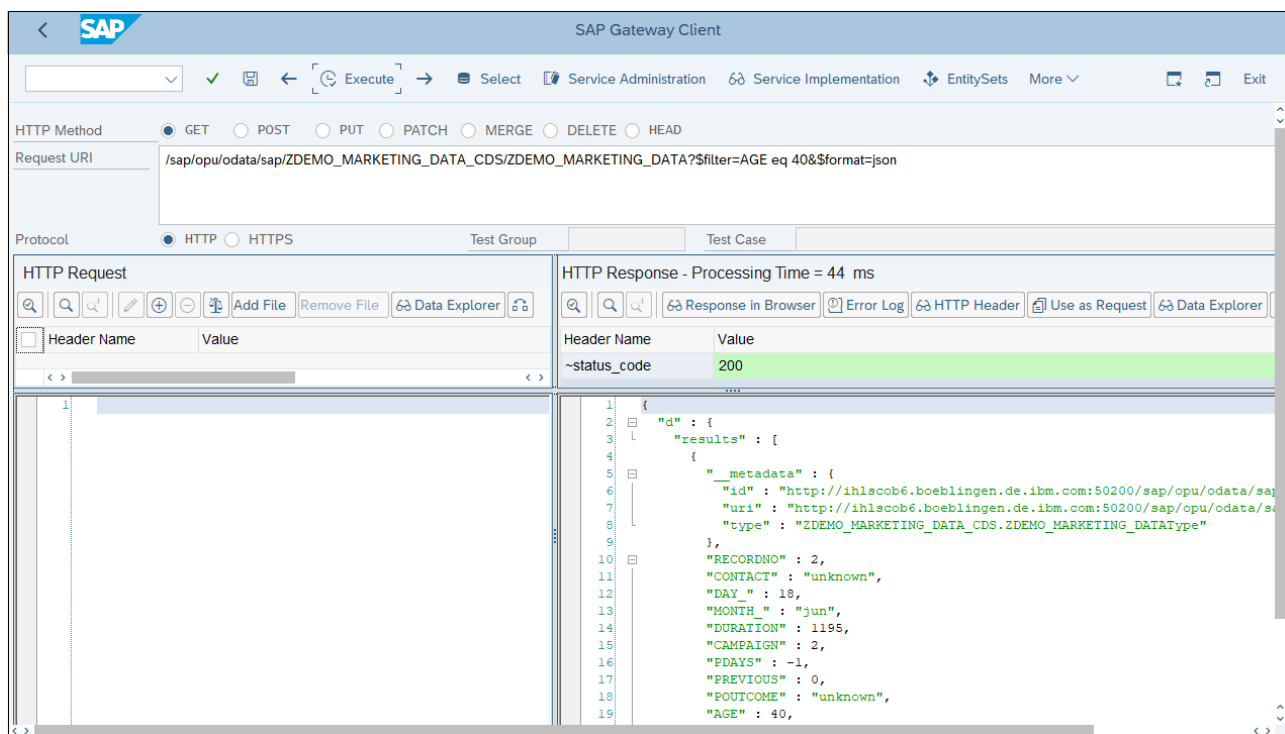


Figure 3-11 Gateway Client

To test the same query from an external system, open a shell on your local PC and connect to the OData service by using the `curl` command. Do not forget to URL-encode the query parameters. Considering that the HTTPS protocol is used, you must install the suitable TLS certificate on your local PC or use `--insecure` option:

```
curl -u "<sap user>:<sap password>" --insecure "https://cap-sg-prd-3.securegateway.appdomain.cloud:17857/sap/opu/odata/sap/ZDEMO_MARKETING_DATA_CDS/ZDEMO_MARKETING_DATA?$filter=AGE%20eq%2040&$format=json"
```

Now, the SAP system is prepared and the data can be retrieved by an external application.

3.4 Building a node.js application on IBM Cloud

Create a node.js script that functions as a web server application. It accepts requests from a front end and delivers the requested data in response.

IBM Cloud provides starter kits that are boilerplates for standard application schemes. When deployed, a starter kit forms a self-contained web application that can be used as a template for an individual application.

Complete the following steps:

1. Log on to the IBM Cloud.
2. Click **Create Resource** and then, select the **Developer Tools** category from the navigation window. Find and click **Node.js Express.js App**, as shown in Figure 3-12.

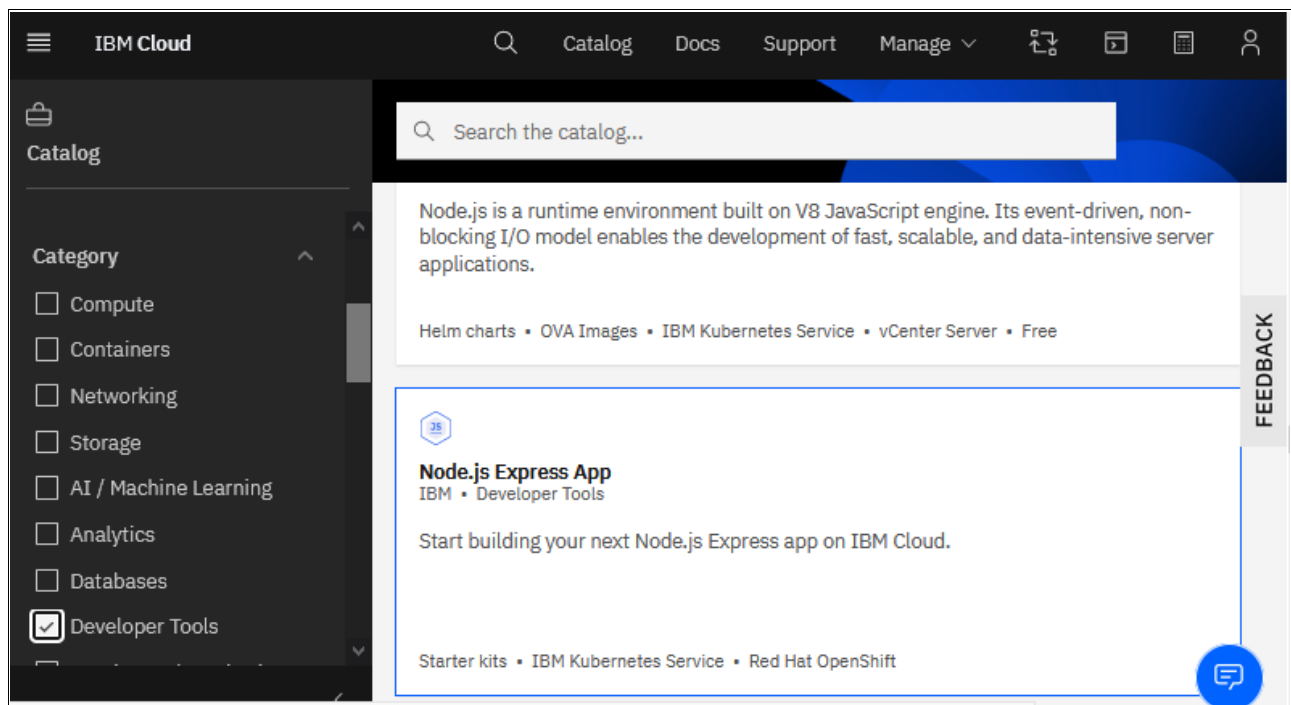


Figure 3-12 IBM Cloud Node.js Starter Kit

3. Enter Marketing Campaign Advisor for the App name and click **Create**, as shown in Figure 3-13.

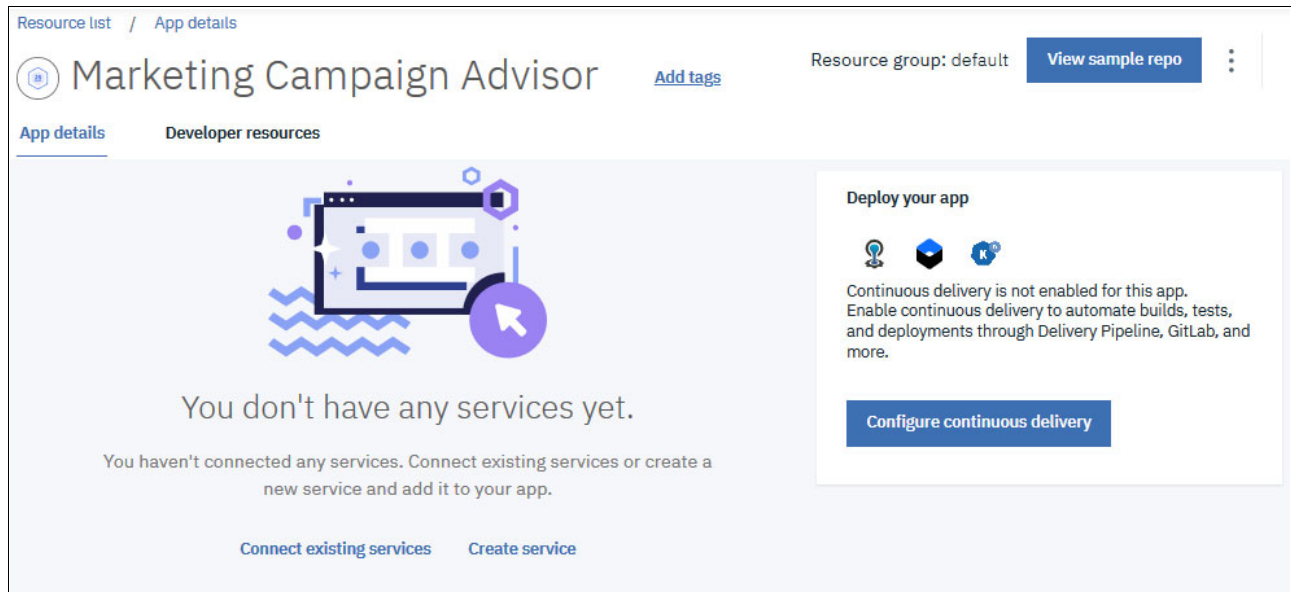


Figure 3-13 IBM Cloud Initial App

The application accesses the model that was trained and deployed by an IBM Watson Machine Learning service instance. This service instance must be connected to the application. When run in the IBM Cloud, credentials of all connected service instances are provided to the application by environment variables.

4. In the Connections section, click **Create connection**.
5. Select the IBM Watson Machine Learning service instance that you want to connect to the application and click **Connect**. The service instance is bound to the application, and new service credentials are created and displayed in the section in the App details view.
6. Click **Configure continuous delivery**.
7. In the next window, select the Deployment target. The application can be deployed in a container that is deployed on an IBM Kubernetes cluster.

Alternatively, the application can be deployed on a Red Hat OpenShift cluster.

As a third option, it can be run as a Cloud Foundry application.

When an IBM Kubernetes or a Red Hat OpenShift cluster is chosen as deployment target, a corresponding cluster service must be instantiated first.

- If you select **Cloud Foundry** as Deployment target, change the hostname if the default is no longer available (likely). Keep default values for all other parameters. Click **Create** to create a delivery pipeline, as shown in Figure 3-14.


Resource list / App details

Marketing Campaign Advisor [Cancel](#) [Create](#)


Deploy your app

Select your deployment target and configure your DevOps toolchain. After you click **Create**, the toolchain is created, and the deployment process is started automatically.


Deployment target



IBM Kubernetes Service
Provision a cluster of hosts, called worker nodes, to deploy and manage highly available application containers.



Red Hat OpenShift on IBM Cloud
Deploy your apps on highly available clusters that come installed with Red Hat OpenShift on IBM Cloud.



Cloud Foundry ✓
Deploy your app without managing underlying infrastructure.

Number of instances

1

Memory allocation per instance

64 MB 128 2000 MB

Select region to deploy in **Select an organization** **Select a space**

Dallas DSXrese dev

Host **Domain**

marketing-campaign-advisor mybluemix.net

DevOps toolchain name

Accept the default name, or enter a value up to 63 characters.

MarketingCampaignAdvisor

Select the region that your toolchain is created in, and then select the resource group that provides access to your new toolchain.

Region **Resource group**

Dallas default

Figure 3-14 IBM Cloud Continuous Delivery

- If the error message Continuous Delivery service required appears, select the link **Add the service**.

10. You are requested to choose a pricing plan. Start with a no cost Lite plan. You can upgrade to a professional plan later. Click **Create** to instantiate the Continuous Delivery service. Return to your delivery pipeline and refresh the browser page. The error message disappears, as shown in Figure 3-15.

Resource list / App details

Marketing Campaign Advisor [Visit App URL](#) [Add tags](#) Resource group: default [View repo](#)

App details Developer resources

Services (1) [Connect existing services](#) [Create service](#)

NAME	RESOURCES	ACTIONS
Machine Learning	Documentation	

Continuous delivery [Remove from toolchain](#)

[https://us-south.git.cloud.ibm.com/rese/MarketingCampaignAdvisor](#)

Toolchain

Name MarketingCampaignAdvisor

Location Dallas

Resource group default

Tool integrations

Delivery Pipelines

Name Marketing Campaign Advisor

Status ✔ Success

Last input Last commit by Joachim Rese (10 minutes ago)
[Clone from zip](#)

Credentials [Download](#) [Show](#)

```
{
  "machinelearning": [
    {
      "name": "wml-dallas",
      "credentials": {
        "apikey": ".....",
        "instance_id": "c9b85716-05c5-4278-b368-ebf3632c31af",
        "url": "https://us-south.ml.cloud.ibm.com"
      }
    }
  ]
}
```

Figure 3-15 Node.js App Toolchain

11. Wait until the status of the delivery pipeline turns to Success. The application is now deployed.

If the application runs on Cloud Foundry, it can be started by clicking **Visit App URL**. In our example, the application is available at the following website (see Figure 3-16 on page 55):

<https://marketing-campaign-advisor.mybluemix.net>

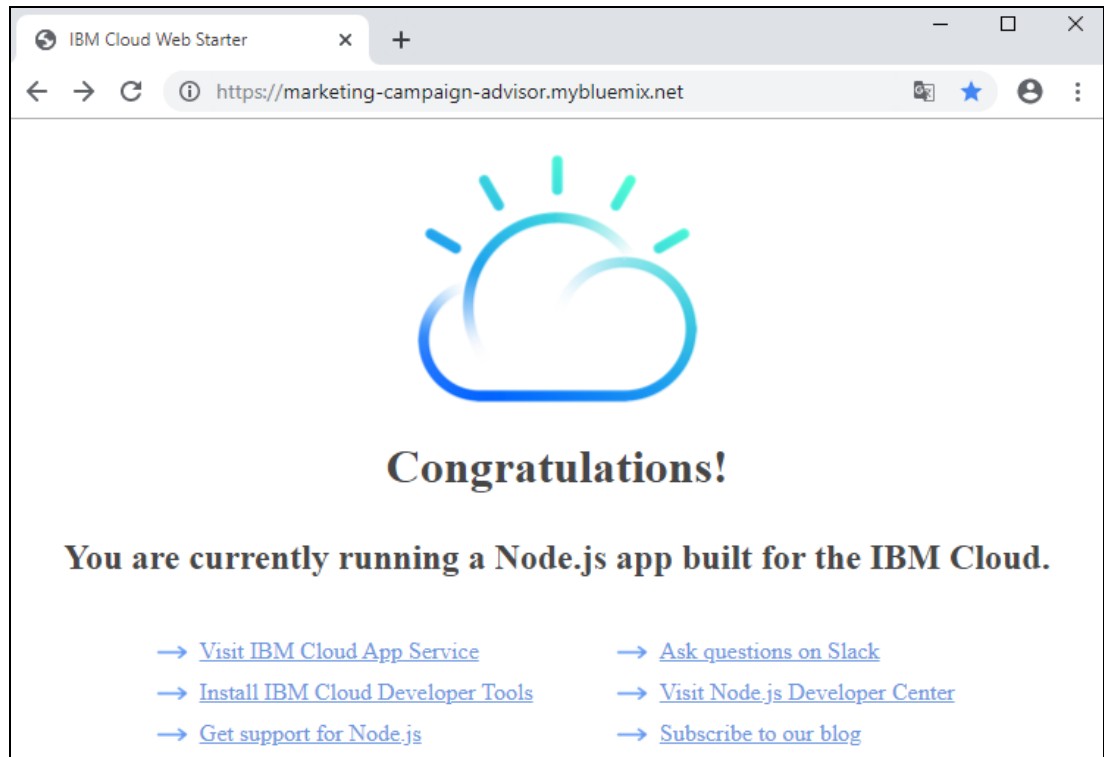


Figure 3-16 WebApp window

If the application runs as a container on an IBM Kubernetes cluster, the URL must be concatenated from the cluster worker's public IP and the application service node port. To compile this information, complete the following steps:

1. On the IBM Cloud dashboard, open the application and select **View Kubernetes cluster** on the App details view. On the cluster dashboard, select the **Worker Nodes** tab. The node information contains the Public IP, as shown in Figure 3-17.

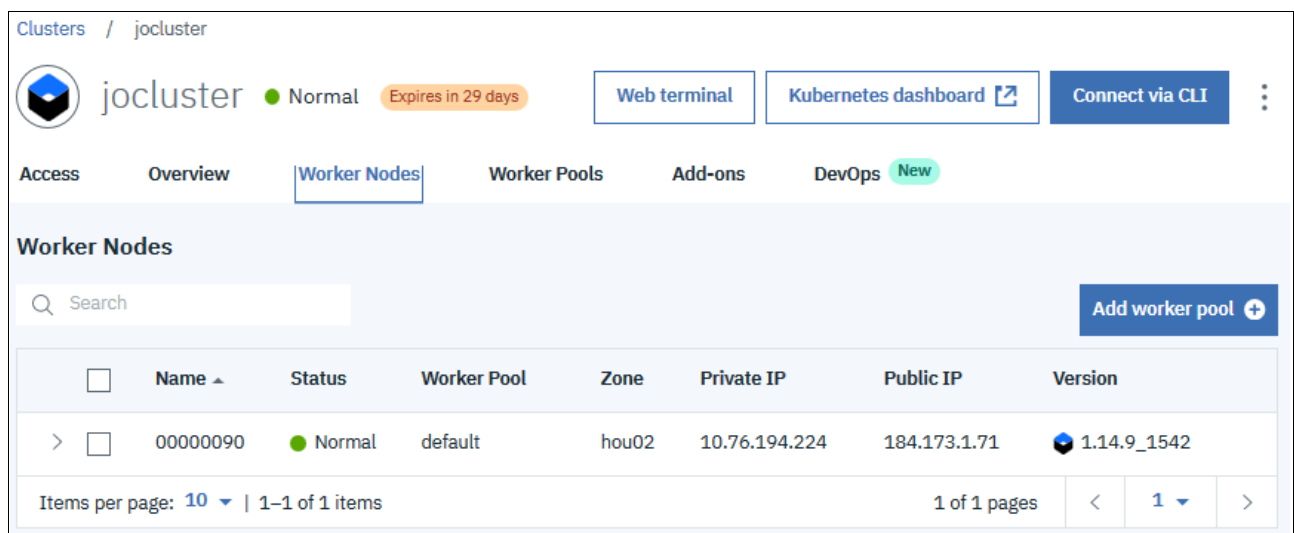


Figure 3-17 Kubernetes dashboard

2. Go to the Kubernetes dashboard. In the overview view, scroll down to the Service section and click the service that corresponds to your application. In the Details section, find the port of the last entry of the Internal endpoints, as shown in Figure 3-18.

Details

Name:	app-application-service
Namespace:	default
Labels:	chart: app-1.0.0
Annotations:	prometheus.io/scrape: true
Creation Time:	2019-12-12T12:58 UTC
Label selector:	app: app-selector
Type:	NodePort
Session Affinity:	None
Connection	
Cluster IP:	172.21.212.55
Internal endpoints:	app-application-service:3000 TCP app-application-service:32282 TCP

Figure 3-18 Kubernetes dashboard details

The application URL looks like the following example:

http://184.173.1.71:32282

3.4.1 Installing and running application locally

The generated application is stored in a Git repository. Find the URL to the repository on the application overview, section **Continuous deliver**, which is next to the Git icon, as shown in Figure 3-19.

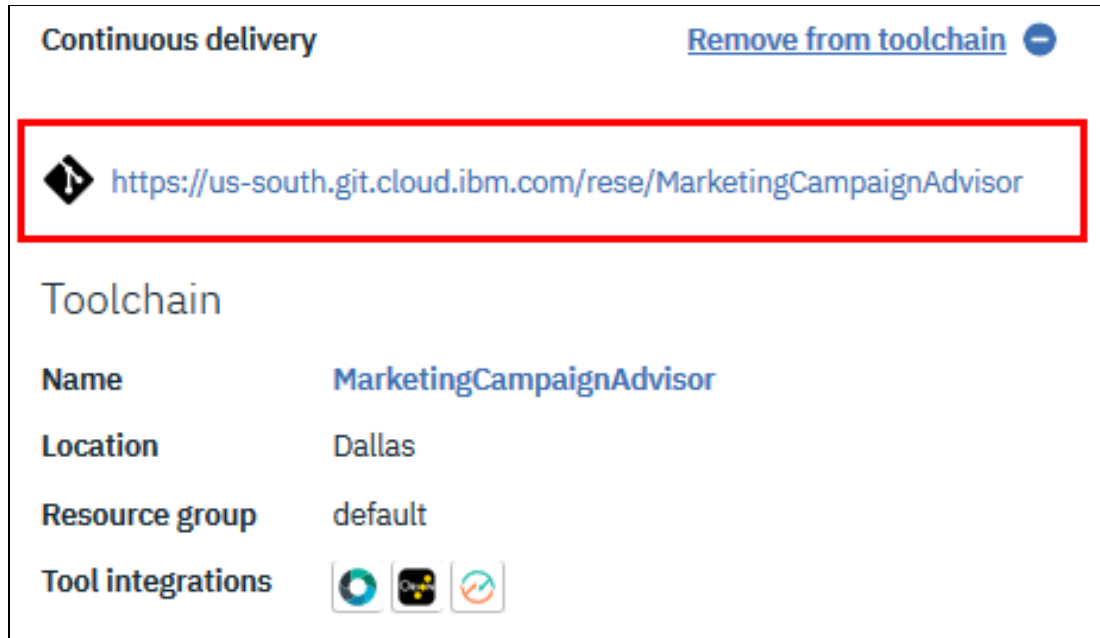


Figure 3-19 GitHub repository

Complete the following steps:

1. Open a shell on your local PC and clone the code:

```
...> git clone https://us-south.git.cloud.ibm.com/rese/MarketingCampaignAdvisor
Cloning into 'MarketingCampaignAdvisor'
Username for 'https://us-south.git.cloud.ibm.com': <your-email>@domain.com
Password for 'https://<your-email>@domain.com@us-south.git.cloud.ibm.com':
warning: redirecting to
https://us-south.git.cloud.ibm.com/rese/MarketingCampaignAdvisor.git/
remote: Enumerating objects: 113, done.
remote: Counting objects: 100% (113/113), done.
remote: Compressing objects: 100% (104/104), done.
remote: Total 113 (delta 4), reused 113 (delta 4)83.00 KiB/s
Receiving objects: 100% (113/113), 1.12 MiB | 177.00 KiB/s, done.
Resolving deltas: 100% (4/4), done.
```

2. Change directory to the location of the cloned code (cd ./MarketingCampaignAdvisor).

3. To install all depended node.js modules, run:

```
.../MarketingCampaignAdvisor> npm install
```

4. If vulnerabilities are found, check details and perform the recommended action:

```
.../MarketingCampaignAdvisor> npm audit
=== npm audit security report ===
# Run npm install --save-dev nyc@14.1.1 to resolve 7 vulnerabilities
[...]
.../MarketingCampaignAdvisor> npm install --save nyc@14.1.1
[...]
```

found 0 vulnerabilities

5. Start the application locally:

```
.../MarketingCampaignAdvisor > npm run start  
[2019-12-05T13:06:57.829] [INFO] nodejswebapp - nodejswebapp listening on  
http://localhost:3000
```

If (and only if) you are on Windows, you might experience an error that indicates that variable `$npm_package_config_entrypoint` is unknown. This variable must be replaced by the Windows scheme for environment variables in file `package.json`; for example, `%npm_package_config_entrypoint%`.

However, when the application is deployed in the IBM Cloud, variables in the file `package.json` start with the US dollar sign (\$). You can change all variable names in local file `package.json` and remove that file from Git tracking, or, you add an entry, as shown in the following example:

```
"startwin": "node %npm_package_config_entrypoint%"
```

This entry is added to the section `scripts` in file `package.json` and starts your local application by starting:

```
.../MarketingCampaignAdvisor> npm run startwin
```

Start a web browser and navigate to the URL that was printed by the `node.js` application; for example, `http://localhost:3000`. It displays the same page as seen when the application is run on the IBM Cloud.

IBM Watson Machine Learning service credentials

Credentials for the IBM Watson Machine Learning service instance that were connected to the application are provided as environment variables. For Cloud Foundry applications, service credentials are available in environment variable `VCAP_SERVICES`. On Kubernetes, an individual environment variable for each credential parameter is set.

When running locally, these environment variables are not available. Instead, create a file `server/localdev-config.json` and paste service credentials into it as provided on the App details view on the IBM Cloud dashboard. Use the copy icon that is displayed next to the credentials to ensure that the `apikey` gets copied correctly.

The various locations of the service credentials must be specified in file `server/config/mappings.json`. The following file contents can be used for the example application:

```
{  
  "version": 1,  
  "watson_machine_learning_credentials": {  
    "searchPatterns": [  
      "cloudfoundry:$.*[0].credentials",  
      "file:/server/localdev-config.json:$.*machinelearning[0].credentials"  
    ]  
  }  
}
```

The application can start function `getDictionary` of `node.js` module `ibm-cloud-env`. This function references file `mappings.json` to find the service credentials and returns those.

The chart that is shown in Figure 3-20 provides an overview of the various credentials locations and how they are consolidated.



Figure 3-20 Credentials schema

3.5 Using OData and IBM Watson services

This section describes how to use OData in IBM Watson services.

3.5.1 Creating the back-end application

The node.js sample application that was deployed is used as a skeleton for an example application.

The application uses some node.js modules that are not included in the default sample. Therefore, install the following modules:

```
.../MarketingCampaignAdvisor> npm install --save https yamljs ibm-cloud-env
```

To enable the application to access the OData EntitySet by way of the SAP OData service that were deployed in the SAP system before, create the file `sap-credentials.yml` and store the OData service credentials in this file.

Copy the following lines into the file and adapt those lines according to the credentials of your individual OData service:

```
url: "cap-sg-prd-3.securegateway.appdomain.cloud"
port: 17857
user: "<sap user>"
password: "<password for sap user>"
path: /sap/opu/odata/sap/ZDEMO_MARKETING_DATA_CDS/ZDEMO_MARKETING_DATA
```

3.5.2 Node.js server code

The Node.js server program performs the following steps:

1. Gets model metadata and access token as an initialization step.
2. Provides an API that accepts a POST request that contains a set of filter criteria from a client.
3. Calls the SAP OData service to retrieve customer data that applies to the filter criteria.
4. Calls the IBM Watson Machine Learning Endpoint to get predictions for the customer data.
5. Returns the customer data that is enriched by the predictions to the calling client.

File `server/server.js` contains comment line `// Add your code here`. Place your code immediately after that line. For more information about the example `node.js` application code, see Appendix A, “node.js server code” on page 105.

3.5.3 Creating front-end code

The front-end application must run a simple logic:

1. Compile user input to a filter object.
2. Post the filter to the back-end server application and retrieve customer data with predictions.
3. Display the data as HTML table.

Create folder `public/js`. Within that folder, implement the front-end application as JavaScript file `frontend.js`. For more information about example JavaScript code, see Appendix A, “node.js server code” on page 105.

3.5.4 Creating HTML index page

The HTML index page is the landing page for your application. This page includes the front-end JavaScript code (`js/frontend.js`) by a `<script>` tag. It also provides an input form where the user can enter various filter criteria. Also, it has an empty `<div>` section where the front-end code places the output.

Replace the file `public/index.html` with the example index page that is available in Appendix A, “node.js server code” on page 105.

The implementation of the example application is now complete. To run the application locally, enter:

```
.../MarketingCampaignAdvisor> npm run start
[...]  
[2019-12-05T13:06:57.829] [INFO] nodejswebapp - nodejswebapp listening on  
http://localhost:3000  
[2019-12-05T13:06:59.696] [INFO] nodejswebapp - Metadata loaded, access token  
expires in 3600 seconds.
```

The `node.js` program prints the URL (host and port) to which it is listening.

Open that URL and enter some filter criteria (or keep the defaults) and click **Predict**.

The application connects to the SAP system to get the list of customers that apply to the filter criteria, calls the IBM Watson Machine Learning service to enrich that data with a probability that the customer subscribes a term deposit, and displays the result in the browser, as shown in Figure 3-21 on page 61.

action	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	prediction	probability
call_customer	40	technician	married	tertiary	no	4396	no	no	unknown	18	jun	1195	2	-1	0	unknown	yes	0.9124521
call_customer	40	technician	married	primary	no	284	yes	no	celluar	20	apr	1332	2	-1	0	unknown	yes	0.8949225
call_customer	40	technician	married	secondary	no	109	no	no	unknown	20	jun	402	22	-1	0	unknown	no	0.1013633

Figure 3-21 Market Campaign Advisor

After the application runs locally, commit and push the changes to the Git repository:

```
git add sap-credentials.yml
git add server/config/mappings.json
git commit -a -m "<description>"
git push
```

Immediately after the changes are pushed to the repository, the delivery pipeline starts to rebuild the application and deploys the updated application on IBM Cloud, as shown in Figure 3-22.

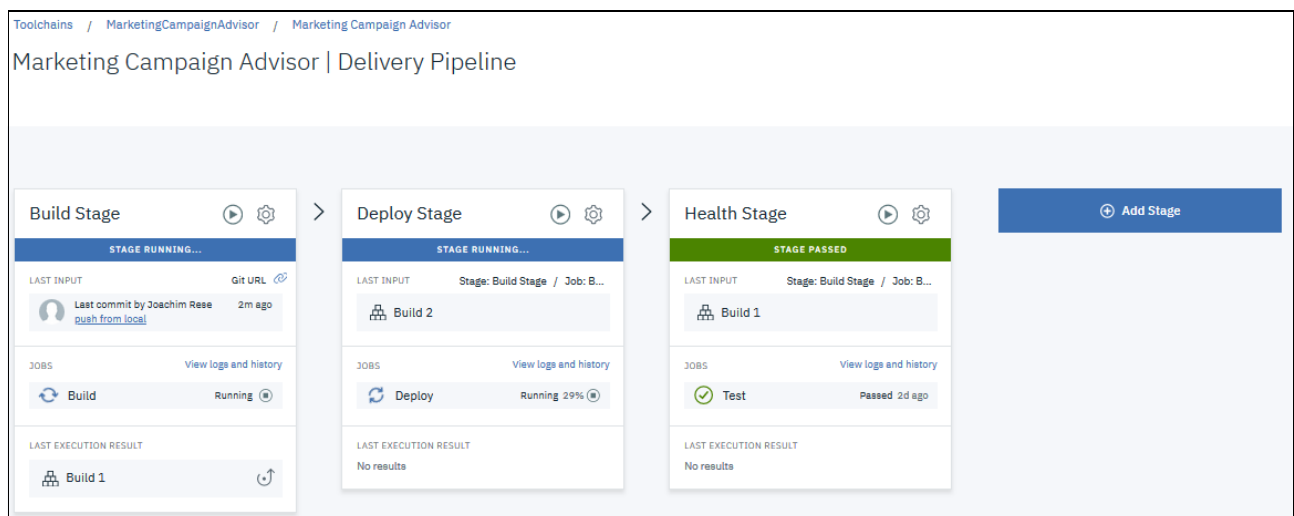


Figure 3-22 Market Campaign Advisor | Delivery Pipeline

After all stages of the delivery pipeline are performed successfully, the updated application is available by way of the public URL.

Start the application in a web browser.



Machine Learning on IBM Power Systems

This chapter focuses on an approach of connecting Artificial Intelligence (AI) models that are hosted on a TensorFlow Model Server (TMS) to SAP HANA. Such a scenario allows you to use powerful AI capabilities, which are not supported by SAP HANA immediately.

The data contained in your SAP HANA in the form of relational tables that can serve as a direct input to the AI model of your choice, and deployed on an external TMS instance.

Achieving the same functions without direct connectivity between SAP HANA and a model requires exporting database tables to an outside location and then, calling the inferencing methods of an externally deployed model. This configuration provides the exported data as an input.

In real-world scenarios that involve big data, choosing the latter approach might introduce scalability issues.

Starting with SAP HANA 2.0 SPS 02, SAP introduced a component under the name of External Machine Learning (EML), which is meant to serve as a direct connection module between SAP HANA and TMS.

EML is delivered in form of an Application Function Library (AFL) for SAP HANA.

Such AFLs contain application logic in the form of functions that are implemented with C++ and are callable by way of plain SQL, much like any stored procedure.

This chapter includes the following topics:

- ▶ 4.1, “IBM Watson Machine Learning Accelerator” on page 64
- ▶ 4.2, “Use case scenario and implementation” on page 69
- ▶ 4.3, “Preparing AI model for deployment” on page 77
- ▶ 4.4, “Conclusion” on page 85

4.1 IBM Watson Machine Learning Accelerator

The Community edition of IBM Watson Machine Learning Accelerator provides open source, deep learning frameworks, such as TensorFlow, for running on IBM POWER® hardware. It is an end-to-end, deep learning platform for data scientists.

IBM Watson Machine Learning Accelerator can be installed by using one of the following methods:

- ▶ Installing on Red Hat Enterprise Linux.

This installation is the standard method that can be found in the product installation guide.

- ▶ Installing on Red Hat OpenShift 3.11.

Deploy IBM Watson Machine Learning Accelerator as a Docker container in Red Hat OpenShift 3.11. This is a modern way to provide services; for example, TensorFlow.

This section describes this installation method and verifies the installation by calling a `hello tensorflow` application.

4.1.1 Installing on Red Hat Enterprise Linux

The IBM Watson Machine Learning Accelerator 1.2.1 installation process features the following overall tasks:

- ▶ Planning
- ▶ Completing prerequisites
- ▶ Installing the operating system
- ▶ Opening necessary ports
- ▶ Ensuring user access of client machines to cluster hosts
- ▶ Setting the suitable heap size
- ▶ Logging in by using a root permission
- ▶ Mounting a shared file system
- ▶ Completing further setup steps and starting the installation

For more information, see this [IBM Documentation web page](#).

4.1.2 Installing on Red Hat OpenShift 3.11

The official Docker images for IBM PowerAI are available at [this web page](#).

Complete the following steps:

1. Use the following commands to pull and install the Docker images:

```
docker pull ibmcom/powerai
Using default tag: latest
Trying to pull repository registry.redhat.io/ibmcom/powerai ...
Trying to pull repository docker.io/ibmcom/powerai ...
latest: Pulling from docker.io/ibmcom/powerai
Pull complete
Status: Downloaded newer image for docker.io/ibmcom/powerai:latest

docker run -ti --env LICENSE=yes ibmcom/powerai:latest bash
(wmlce) pwrai@501ce9522cd7:/$ uname -a
Linux 501ce9522cd7 3.10.0-1062.el7.ppc64le #1 SMP Thu Jul 18 20:29:07 UTC
2019 ppc64le ppc64le ppc64le GNU/Linux
```

```
docker ps | grep 501ce9522cd7
```

```
501ce9522cd7    ibmcom/powerai:latest
"bash"         2 minutes ago    Up 2 minutes
clever_knuth
```

```
docker save --output powerai.tar ibmcom/powerai:latest
```

```
ls -al powerai.tar
```

```
-rw-----. 1 root root 15411359232 Nov 22 16:37 powerai.tar
```

2. Load it in to the local Docker repository on the Red Hat OpenShift cluster:

```
docker load -i powerai.tar
```

```
Loaded image: ibmcom/powerai:latest
```

3. Verify that the Docker image was loaded:

```
docker images
```

4. Copy the log-in command from the Red Hat OpenShift console, as shown in Figure 4-1.

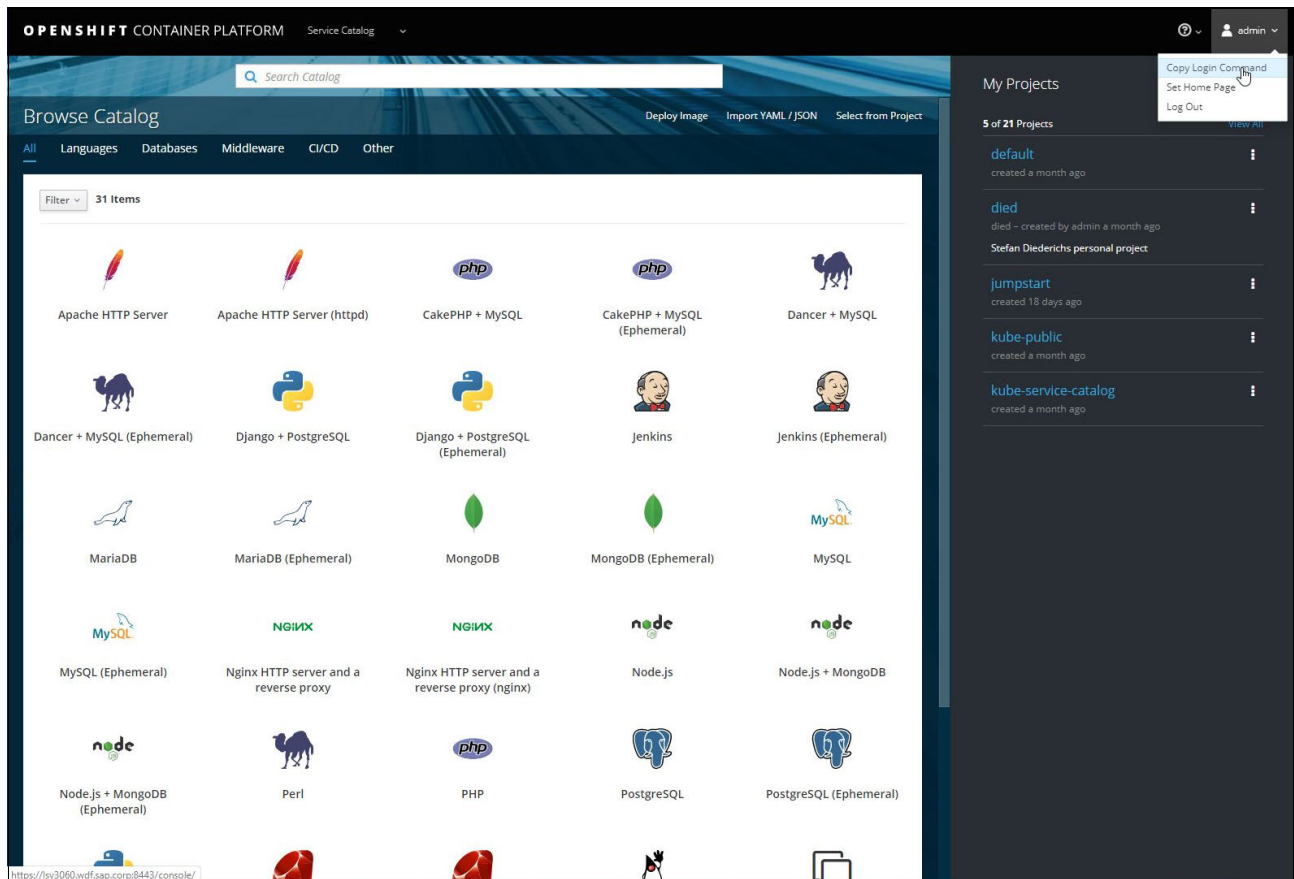


Figure 4-1 Copy the login command from Red Hat OpenShift Console

5. Log in to Red Hat OpenShift:

```
oc login https://1sv3060.ibm.com:8443 --token=<token>
```

6. Log in to `https://1sv3060.ibm.com:8443` as `admin` by using the provided token.

For more information about deploying applications from images in Red Hat OpenShift by using the web console, see [this web page](#).

7. Deploy the image (see Figure 4-2).

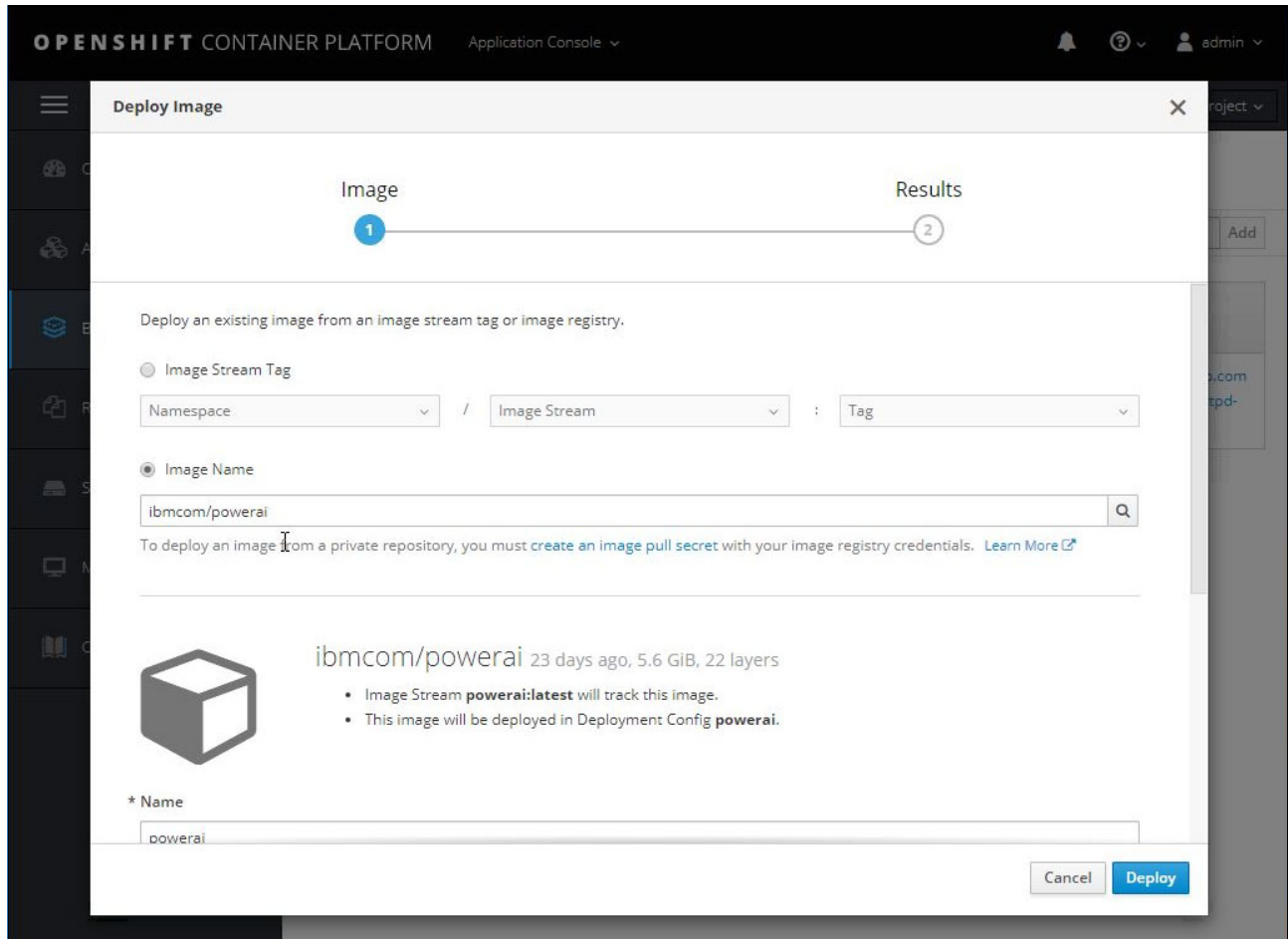


Figure 4-2 Deploy image

Figure 4-3 shows that the deployed image of powerai was created.

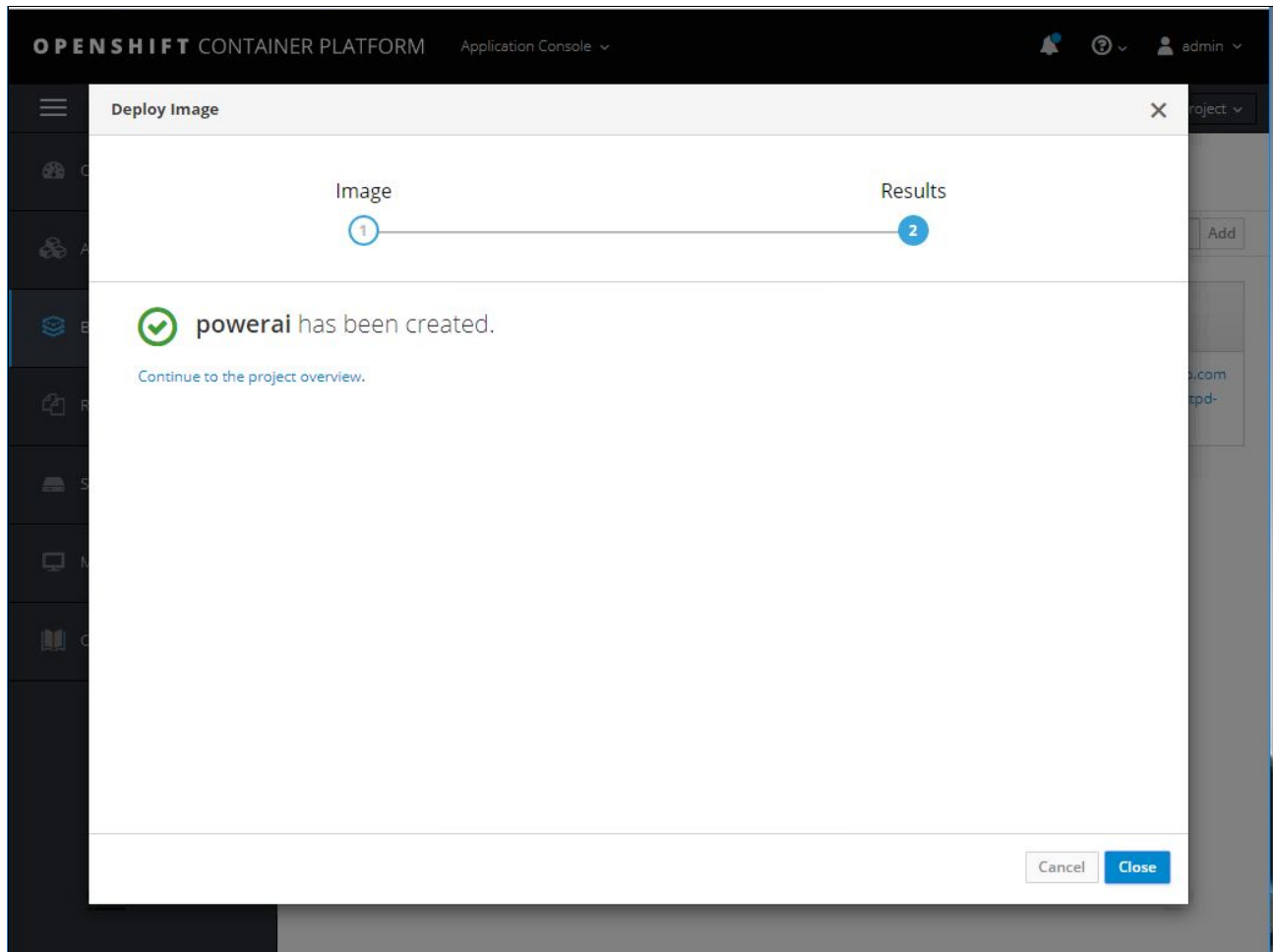


Figure 4-3 PowerAI has been created

Figure 4-4 shows powerai is available on the application console.

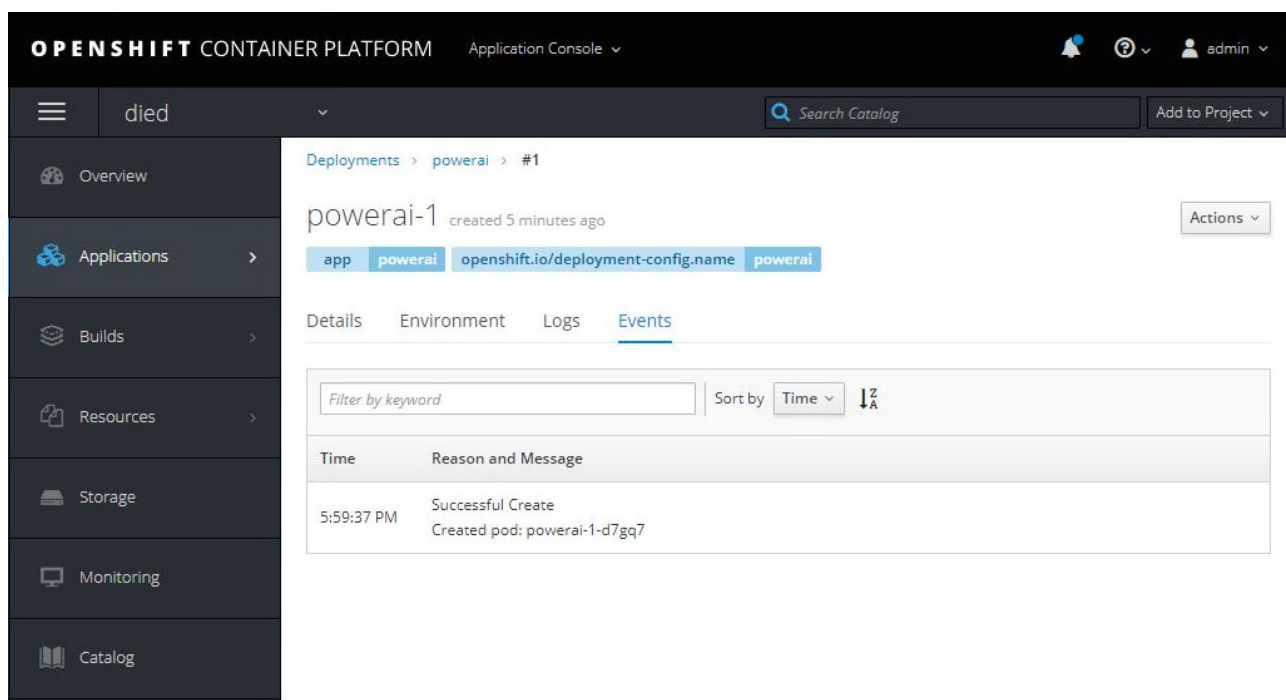


Figure 4-4 PowerAi Application Console

8. Log in to the powerai container and verify that TensorFlow is working:

```
docker run -ti --env LICENSE=yes ibmcom/powerai:latest bash
```

```
(wmlce) pwrai@51ce1cbd72af:/$uname -a
Linux 51ce1cbd72af 3.10.0-1062.el7.ppc64le #1 SMP Thu Jul 18 20:29:07 UTC
2019 ppc64le ppc64le ppc64le GNU/Linux
(wmlce) pwrai@51ce1cbd72af:/$python
Python 3.7.5 (default, Oct 25 2019, 16:29:01)
[GCC 7.3.0] :: Anaconda, Inc. on linux
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
b'Hello, TensorFlow!'
```

4.2 Use case scenario and implementation

To easily understand all steps of the deployment, connection, and use of the AI model by way of EML and SAP HANA, this section focuses on a popular use case of an object detection network that is similar to YOLO¹. This use case implements the architecture that is shown in Figure 4-5.

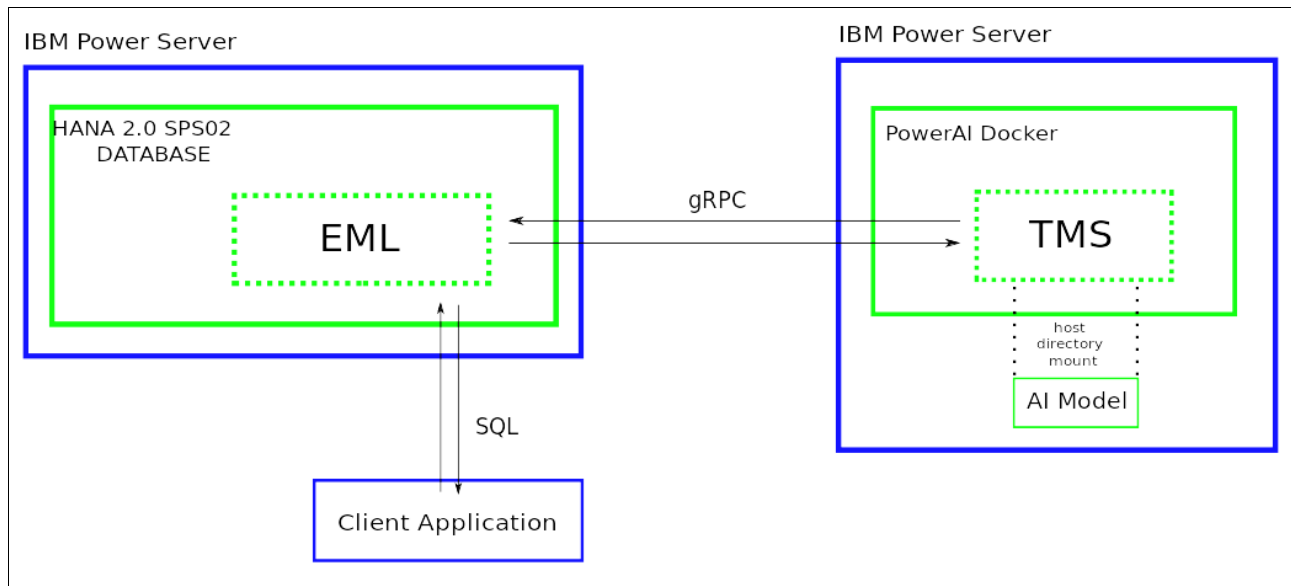


Figure 4-5 Connection and use of an external AI model by way of EML and SAP HANA

4.2.1 Naming scheme

We differentiate between a server, which hosts SAP HANA with EML denoted as [A] and a server, where a TMS model is stored, denoted as [B].

For practical reasons, test this scenario uses only one IBM Power Systems machine, in which case [A] denotes the same server as [B], but such a scenario is not always feasible in a real-world application. Therefore, we recommend using the set up that is described in Figure 4-5, which features distinct [A] and [B] servers.

4.2.2 Installing SAP HANA

Start by getting the SAP HANA, Express Edition Download Manager (DM). This application helps to download the installers for SAP HANA and the EML component.

Now, you can use your own workstation to run DM and later move the SAP HANA and EML installation artifacts that are fetched by DM to server [A].

For more information about registering your account and to getting the download, see [the SAP website](#).

After running the DM executable file in your workstation, an installation wizard starts, as shown in Figure 4-6 on page 70.

¹ Joseph Chet Redmon: YOLO: Real-Time Object Detection

Note: The Platform field specifies the platform on which SAP HANA and its components are intended to be installed. This platform is *not* the same platform on which the Download Manager is running.

Complete the following steps:

1. Set the target platform to Linux/PowerPC (little endian) and specify the download directory, which is used to store the real installation artifacts, as shown in Figure 4-6.

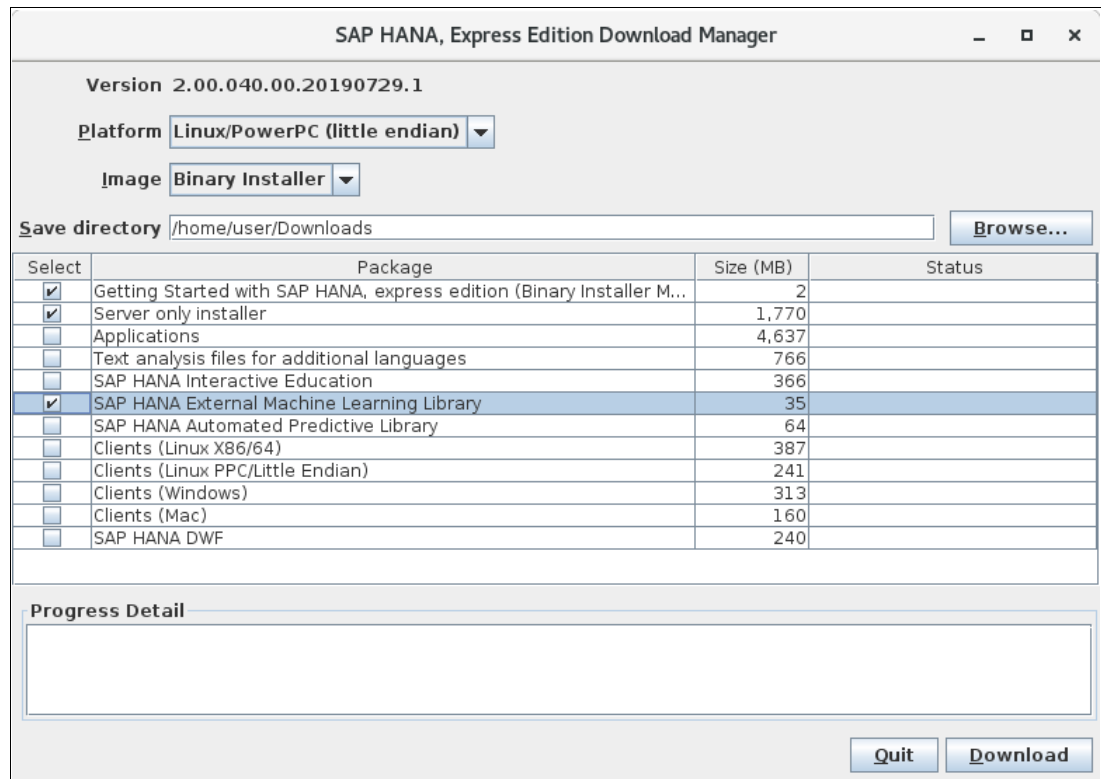


Figure 4-6 SAP HANA, Express Edition Download Manager - Package selection

2. Select the box that corresponds to SAP HANA External Machine Learning Library. Then, click **Download**.

The DM tool fetches the installation artifacts and places them into the specified save directory. The result of this operation looks similar to the example that is shown in Figure 4-7 on page 71.

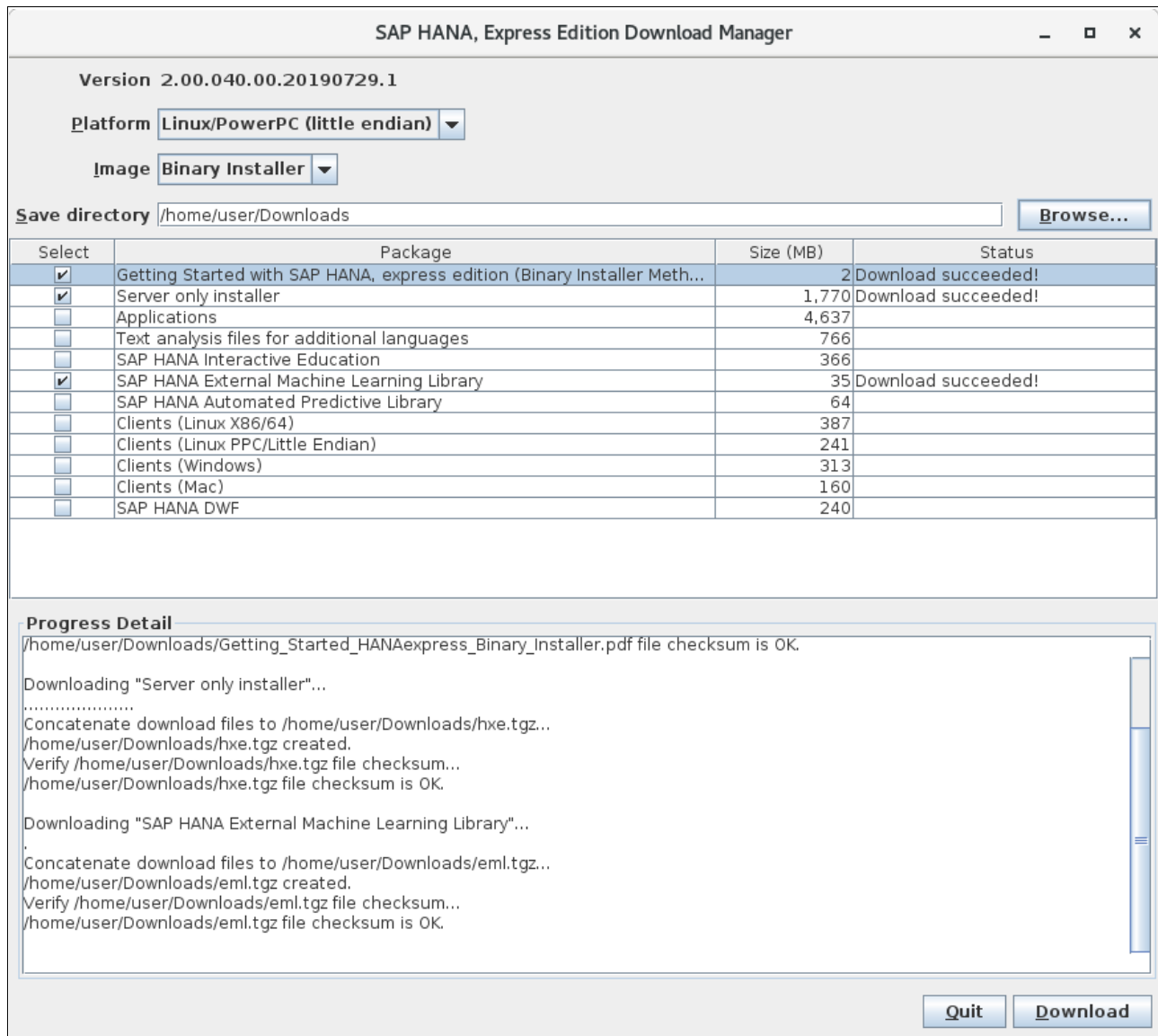


Figure 4-7 SAP HANA, Express Edition Download Manager - Progress detail

3. Transfer `hxe.tgz` and `eml.tgz` from your Save directory (`/home/user/Downloads` in our example) to `/sapmnt` on [A] server, on which SAP HANA with EML is to be installed.

4.2.3 Target [A] machine

Complete the following steps:

1. Extract the archive hxe.gz SAP HANA installation artifacts and run the installation script. The installation script requires you to run it as a root user:

```
$ cd /sapmnt && tar -xf hxe.tgz && ./setup_hxe.sh
```

2. During the installation process, you can select preset defaults (except for the master password) by leaving input fields empty and confirming this input by pressing **Enter**. The resulting window before confirming the installation is shown in Figure 4-8.

```
#####
# Summary before execution                                     #
#####
HANA, express edition installer : /sapmnt/hanainst/HANA_EXPRESS_20
Component(s) to install          : HANA server, Application Function Library, SAP HANA EPM-MDS
Host name                        : myhostA
HANA system ID                   : HDL
HANA instance number             : 35
Master password                  : *****
Log file                         : /var/tmp/setup_hdl_2019-11-29_16.28.50.log
```

Figure 4-8 Summary before execution

After confirming, the installation starts and can take some time to complete, depending on your hardware and system load.

4.2.4 Adding EML to the installed SAP HANA

Complete the following steps:

1. Extract the EML component installer and run the installation.
2. Assuming from here on that your SAP HANA administrator in the underlying operating system is hdladm, switch to that user by using the following command:

Note: If required, grant corresponding read/write access rights to the /sapmnt directory (and subdirectories) to hdladm.

```
$ su - hdladm
```

Followed by:

```
$ cd /sapmnt && tar -xf eml.tgz && cd HANA_EXPRESS_20 && ./install_eml.sh
```

3. After confirming the installation, wait for the process to complete, at which point the output looks similar to the example that is shown in Figure 4-9.

```
HANA, express edition installer : /sapmnt/HANA EXPRESS 20
Component(s) to install      : SAP HANA External Machine Learning Library
Proceed with installation? (Y/N) : Y

Install EML...
SAP EML AFL FOR SAP HANA installation kit detected.

SAP HANA Lifecycle Management - SAP EML AFL Installation 2.00.040.0000.1553698248
*****

Checking installation...
Preparing package 'EML'...
Installing SAP EML AFL FOR SAP HANA to /hana/shared/HDL/exe/linuxppc64le/plugins/eml_2.00.040.0000.1553698248_79315c8b599c1e9bb429ce
Installing package 'EML'...
Stopping system...
  Stopping 5 processes on host 'myhostA' (worker):
  Stopping on 'myhostA' (worker): hdbdaemon, hdbcompileservers, hdbnameserver, hdbwebdispatcher, hdbindexserver (HDL)
  All server processes stopped on host 'myhostA' (worker)
Activating plugin...
Starting system...
  Starting 1 process on host 'myhostA' (worker):
  Starting on 'myhostA' (worker): hdbdaemon
  Starting 6 processes on host 'myhostA' (worker):
  Starting on 'myhostA' (worker): hdbdaemon, hdbcompileservers, hdbdisserver, hdbindexserver (HDL), hdbnameserver, hdbwebdispatcher
  Starting on 'myhostA' (worker): hdbdaemon, hdbcompileservers, hdbdisserver, hdbindexserver (HDL), hdbwebdispatcher
  Starting on 'myhostA' (worker): hdbdaemon, hdbdisserver, hdbindexserver (HDL), hdbwebdispatcher
  Starting on 'myhostA' (worker): hdbdaemon, hdbdisserver, hdbwebdispatcher
  Starting on 'myhostA' (worker): hdbdaemon, hdbdisserver
  All server processes started on host 'myhostA' (worker):
Installation done
Log file written to '/var/tmp/hdb_eml_2019-12-05_17.08.59_64188/hdbinst_eml.log' on host 'myhostA'.
hdbadm@lsv3060:/sapmnt/HANA EXPRESS 20>
```

Figure 4-9 Summary after installation

4.2.5 Installing SAP HANA Studio

In the following sections, you run SQL queries in the SAP HANA database that is hosted on [A]. The easiest way to do so involves the use of SAP HANA Studio (HS), which represents a convenient IDE for viewing, modifying, and administering your databases.

For more information about SAP HANA Studio instance, see [the SAP website](#).

Note: If you decide not to use SAP HANA Studio, you can use the command-line application `hdbsql` as an alternative for performing SQL queries. This application is part of the SAP HANA Express installation and available through the command line if you log in as the SAP HANA user administrator.

4.2.6 Connecting SAP HANA Studio to your SAP HANA database

Complete the following steps:

1. After opening SAP HANA Studio, click **Add System** to add a new connection.
2. Click **Next** to proceed to the authorization. Enter SYSTEM as a username and your installation password to establish and save this connection in your Systems view, as shown in Figure 4-10.

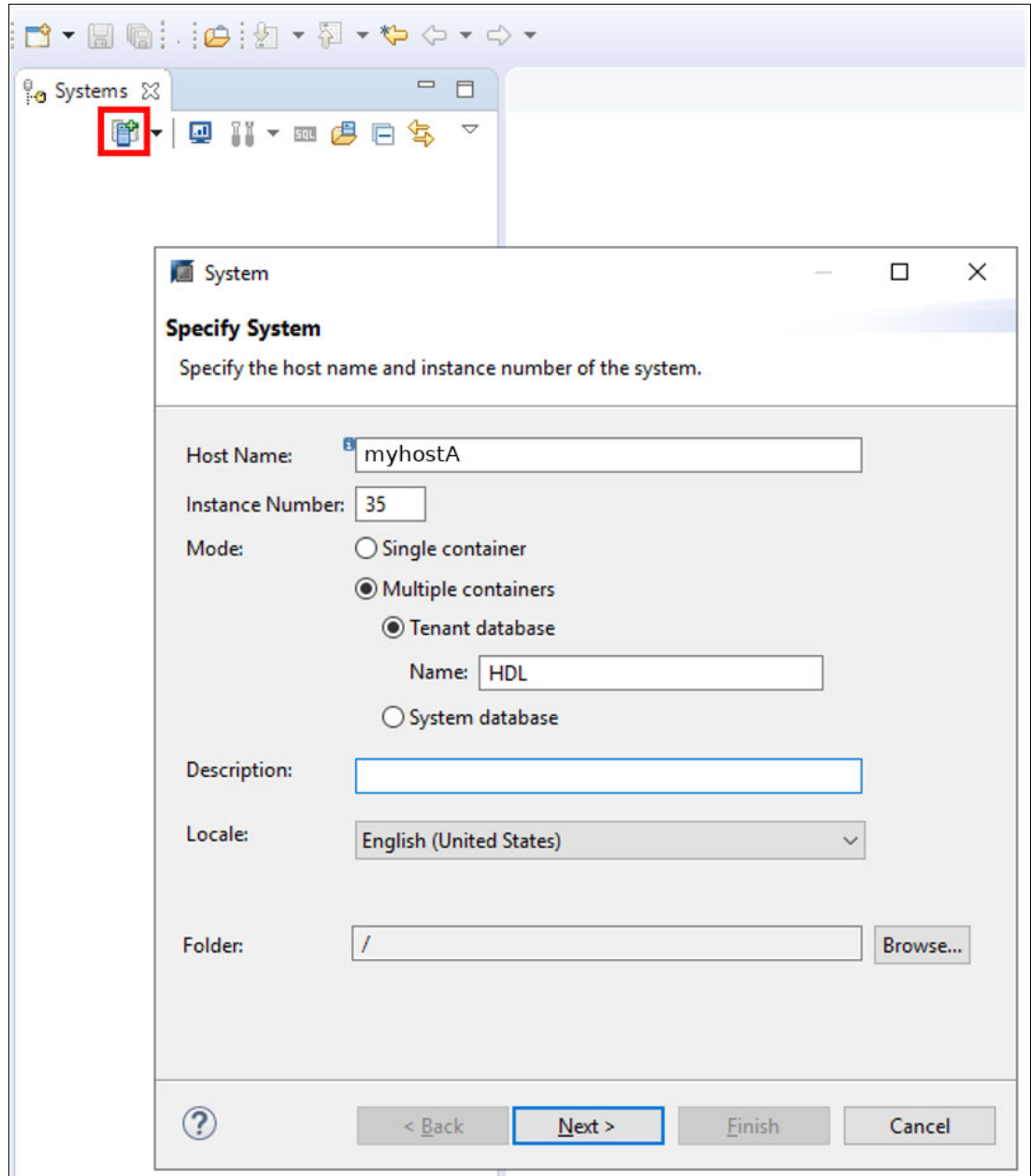


Figure 4-10 Connections in your system view

4.2.7 Checking the EML installation

Now that you a working connection was created to HANA on server [A], the EML installation must be checked.

SAP HANA places meta-information of installed AFLs into the following views:

- ▶ sys.afl_areas
- ▶ sys.afl_packages
- ▶ sys.afl_functions

Therefore, it is recommended to check those views by running the following SQL commands:

```
SELECT * FROM SYS.AFL_AREAS      WHERE AREA_NAME = 'EML';  
SELECT * FROM SYS.AFL_PACKAGES  WHERE AREA_NAME = 'EML';  
SELECT * FROM SYS.AFL_FUNCTIONS WHERE AREA_NAME = 'EML';
```

Complete the following steps:

1. In the menu of the newly added system, open an SQL console, as shown in Figure 4-11.

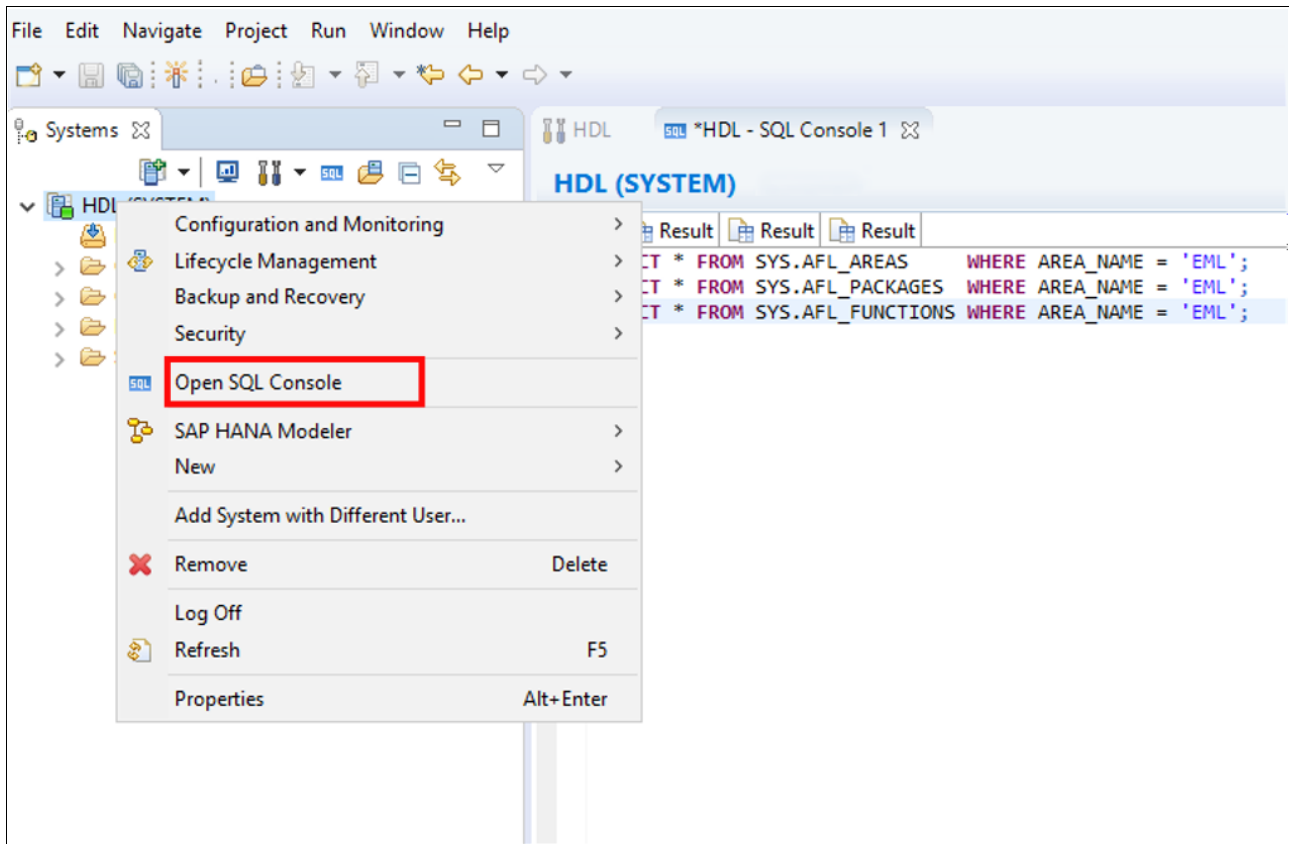


Figure 4-11 Open SQL Console

2. Enter the three SQL queries and run them by pressing **F8**. Each result set of the SELECT queries must contain EML-related information and not be empty if the EML installation was performed correctly.

Also, check if the SAP HANA ScriptServer is running for HDL and enable it if it is not. The ScriptServer is required for the correct functioning of AFL procedures.

One way to check whether it is running is by using the Linux shell of server [A] by running the following line as a hdladm user:

```
$ ps ax -u hdladm | grep hdbscriptserver
```

The resulting list contains the operating system process of the ScriptServer that is running under the hdladm user, as shown in Figure 4-12.

```
root:# ps ax -u hdladm | grep hdbscriptserver
21965 ?          Sl      0:20 hdbscriptserver -port 33540
73090 pts/0      S+      0:00 grep --color=auto hdbscriptserver
```

Figure 4-12 ScriptServer running under the hdladm user

If the ScriptServer is inactive, connect to the SYSTEMDB by using SYSTEM as a username and the SAP HANA administrator password that was chosen during the installation (see Figure 4-9).

The screenshot shows a 'Specify System' dialog box with the following fields and options:

- Host Name: myhostA
- Instance Number: 35
- Mode: ☒ Multiple containers, ☐ Single container, ☐ Tenant database
- Name: (empty field)
- ☒ System database (highlighted with a red box)
- Description: (empty field)
- Locale: English (United States)
- Folder: / (with a 'Browse...' button)

At the bottom, there are navigation buttons: '< Back', 'Next >' (highlighted with a blue box), 'Finish', and 'Cancel'.

Figure 4-13 System database

3. Open the SQL console in SYSTEMDB as a SYSTEM user and run the following SQL query:

```
ALTER DATABASE HDL ADD 'scriptserver';
```

After a successful execution, the corresponding ScriptServer operating system process on server [A] appears.

For more information about the activation procedure, see [SAP Note 1650957](#) (SAP login required).

4.3 Preparing AI model for deployment

In this section, we assume that your model expects the input as a single BASE64 encoded string of bytes that contain the image to be analyzed.

As of the current version, EML supports the output tensor of a form (-1, N) or a subset of it, (K, N), where K and N are fixed values. If you have an AI model, the output of which does not correspond to the shapes, you must modify the output (-s) of your network to match the shape supported by EML.

Note: -1 denotes a tensor dimension of variable length.

Model output modification example

In our testing, we used a pre-trained Keras model that was serialized as an H5 file. The pre-trained model has two distinct output tensors of the following shapes:

- Output Tensor 1. Shape: (-1,), denotes the class of identified object, as shown in Table 4-1.

Table 4-1 Object ID list

Object iD
7.0
8.0
4.0
...

- Output Tensor 2. Shape: (-1, 4), denotes bounding boxes for the corresponding objects that are identified in the Output Tensor 1, as shown in Table 4-2.

Table 4-2 Output Tensor 1

Top	Left	Width	Height
259.45	238.27	76.12	23.93
320.34	331.12	95.31	51.43
119.12	213.90	30.45	46.56
...

EML is expecting an output of shape compatible with (-1, N). One way to achieve compatibility in this case is to concatenate the output layers into one tensor of five fields per detected object, as shown in Example 4-1.

Example 4-1 Concatenate the output layers into one tensor of five fields per detected object

```
x0 = Reshape(target_shape = (1,), input_shape =
model.output[0].shape)(model.output[0])
x1 = model.output[1]
merged = Concatenate(axis = 1)([x0, x1])

tf_input = tf.saved_model.utils.build_tensor_info(model.input)
tf_output = tf.saved_model.utils.build_tensor_info(merged)

model_signature =
(
    tf.saved_model.signature_def_utils.build_signature_def
    (
        inputs = {"input": tf_input},
        outputs = {"output": tf_output},
        method_name = signature_constants.PREDICT_METHOD_NAME
    )
)
```

This process transformed two output tensors with the same number of rows into a single output tensor, as shown in Table 4-3.

Table 4-3 Object ID, top, left, width, and height

Object ID	Top	Left	Width	Height
7.0	259.45	238.27	76.12	23.93
8.0	320.34	331.12	95.31	51.43
4.0	119.12	213.90	30.45	46.56
...

This output tensor is compatible with the EML supported output shape (-1, N).

4.3.1 Preparing a TMS instance

Recent versions of IBM Watson Machine Learning Accelerator include a PowerAI Docker image, which contains TMS for IBM POWER® architecture. We deploy it to our object-recognition neural network.

Note: For more information about installing and the use of the PowerAI Docker image, see 4.1.2, “Installing on Red Hat OpenShift 3.11” on page 64, which includes details about the IBM Watson Machine Learning Accelerator product.

Complete the following steps:

1. Transfer the object-detection model `imagerec` to server [B] and place it into a chosen directory; for example, `/models`.

The next line starts the PowerAI Docker container that is included in the IBM Watson Machine Learning Accelerator product. It mounts the host directory `/models` on [server B] into the container under a similar location `/models`. We also make available port 8500 to the host system. This configuration is required to connect to TMS' gRPC interface:

```
$ docker run -d --name tms -ti --env LICENSE=yes -v /models:/models -p 8500:8500 ibmcom/powerai:latest bash
```

Now, the container starts and detaches from the terminal, running permanently in the background.

2. Open a bash shell inside the container:

```
$ docker exec -it tms bash
```

Note: The subsequent commands in this section must be run inside the bash shell of the TMS container.

3. Using the conda package manager, install and activate the TMS application that is provided by PowerAI:

```
$ conda create -n tf_serving_env tensorflow-serving==*gpu*
$ conda activate tf_serving_env
```

4. After the installation and activation finishes, start TMS with the object-recognition model that was deployed and fork the process into the background:

```
$ tensorflow_model_server --port=8500 --model_name=imagerec --
model_base_path=/models/imagerec &
```

5. Create the EML user in SAP HANA and grant permissions.

6. As a SYSTEM user on HDL, create the EML user for the experiments. Specify a password in double quotes instead of password placeholder:

```
CREATE USER EML PASSWORD "password" NO FORCE_FIRST_PASSWORD_CHANGE;
```

The next commands let the EML user register, then unregister and run EML-related procedures:

```
GRANT CREATE REMOTE SOURCE TO EML;
GRANT AFLPM_CREATOR_ERASER_EXECUTE TO EML;
GRANT SELECT, UPDATE, INSERT, DELETE ON _SYS_AFL.EML_MODEL_CONFIGURATION TO EML;
GRANT AFL__SYS_AFL_EML_EXECUTE TO EML;
```

After the user is created and permissions are granted, use the same process as described in 4.2.6, “Connecting SAP HANA Studio to your SAP HANA database” on page 74 to log in as EML user on HDL and open the SQL console in that user’s context.

From this point on, we assume that all subsequent SQL queries are performed in the EML user’s context.

4.3.2 Communicating with TMS

Now that TMS is deployed model on server [B], we can connect to it from SAP HANA hosted on [A].

The communication between EML and TMS occurs by way of Google RPC protocol (gRPC). The EML component is responsible for retrieving data from its input tables and propagating it by way of gRPC to the TMS instance. It waits for the execution to complete and returns the result in form of a table.

To bind SAP HANA to a TMS instance, we add a Remote Source in SAP HANA that can be done in the following way (remember to replace myhostB with your own host name that is specified by the fully qualified domain name or IP address of server [B]):

```
DROP REMOTE SOURCE "TMS";
CREATE REMOTE SOURCE "TMS" ADAPTER "grpc" CONFIGURATION
'server=myhostB;port=8500';
```

Note: We named the remote connection “TMS”, specifying that the communication is carried out by way of gRPC and pointed to a network location and port of our TMS instance.

4.3.3 Updating model configuration

Update the SAP HANA configuration as described in SAP HANA External Machine Learning Library² and shown in Example 4-2.

Example 4-2 Updating SAP HANA configuration

```
DROP TABLE UPDATE_CONFIGURATION_PARAMS;
DROP TABLE UPDATE_CONFIGURATION_RESULT;
DROP PROCEDURE UPDATE_CONFIGURATION;

CREATE TABLE UPDATE_CONFIGURATION_PARAMS ("Parameter" VARCHAR(100), "Value"
VARCHAR(100));
CREATE TABLE UPDATE_CONFIGURATION_RESULT ("Key" VARCHAR(100), "Value" INTEGER,
"Text" VARCHAR(100));
CREATE PROCEDURE UPDATE_CONFIGURATION() AS
BEGIN
    DECLARE CURSOR CUR FOR
        SELECT VOLUME_ID FROM SYS.M_VOLUMES WHERE SERVICE_NAME = 'indexserver';
    FOR CUR_ROW AS CUR DO
        EXEC 'CALL _SYS_AFL.EML_CTL_PROC(''UpdateModelConfiguration'',
UPDATE_CONFIGURATION_PARAMS, UPDATE_CONFIGURATION_RESULT)'
        || ' WITH OVERVIEW WITH HINT(ROUTE_TO('' || :CUR_ROW.VOLUME_ID || '))';
    END FOR;
```

² [SAP HANA External Machine Learning Library, SAP HANA Platform 2.0 SPS 03, Document Version: 1.1 – 2018-10-31](#)

```
END;
TRUNCATE TABLE UPDATE_CONFIGURATION_RESULT;
CALL UPDATE_CONFIGURATION();
```

Restart SAP HANA to check that the changes are correctly applied by running the following command as a root user:

```
$ su - hdladm -c 'HDB stop && HDB start'
```

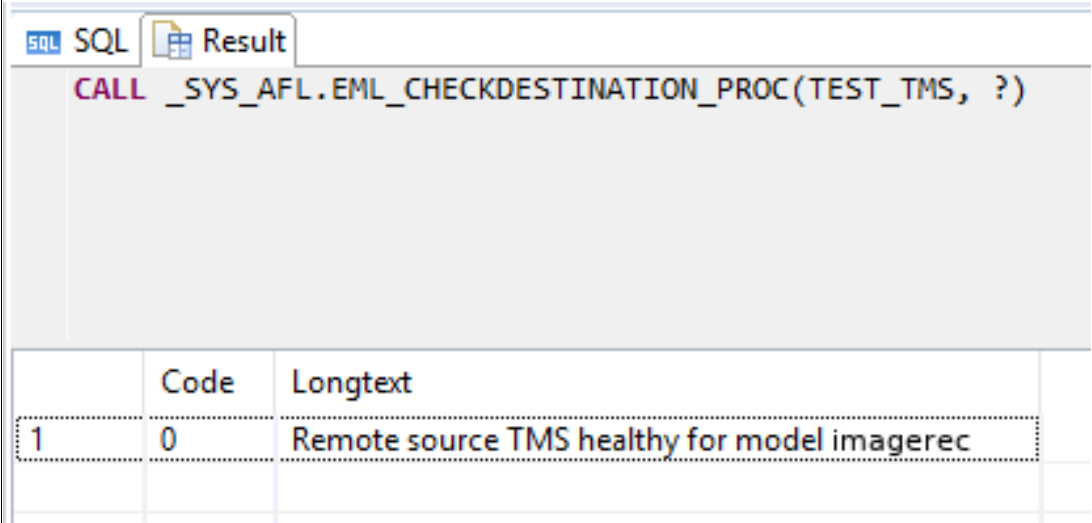
4.3.4 Checking connectivity from SAP HANA

To check the connectivity to your TMS model after all the performed steps, run the SQL statements (as described in: SAP HANA External Machine Learning Library) as shown in Example 4-3.

Example 4-3 Check the connectivity to your TMS model

```
DROP TABLE TEST_TMS;
CREATE TABLE TEST_TMS ("Parameter" VARCHAR(100), "Value" VARCHAR(100));
INSERT INTO TEST_TMS VALUES ('RemoteSource', 'TMS');
INSERT INTO TEST_TMS VALUES ('Model', 'imagerec');
CALL _SYS_AFL.EML_CHECKDESTINATION_PROC(TEST_TMS, ?);
```

The SQL statements produce the output as shown in Figure 4-14.



The screenshot shows a SQL console window with two tabs: 'SQL' and 'Result'. The 'SQL' tab contains the following text: `CALL _SYS_AFL.EML_CHECKDESTINATION_PROC(TEST_TMS, ?)`. The 'Result' tab displays a table with the following data:

	Code	Longtext
1	0	Remote source TMS healthy for model imagerec

Figure 4-14 Remote source TMS healthy for model imagerec

4.3.5 Preparing input images

The following SQL commands create a table that contains the input images in BASE64³ format:

```
DROP TABLE DATA_IMAGES;
CREATE TABLE DATA_IMAGES ("ID" INT, "NAME" CHAR(20), "DATA" CLOB);
```

The DATA column contains the encoded images.

³ RFC 4648: <https://tools.ietf.org/html/rfc4648>

4.3.6 Encoding input images as BASE64

Standard GNU/Linux command-line tools can be used to encode input images in BASE64 format. Because an image is on server [A] under /images/image.jpg, the command in “Web-safe encoding” on page 82 produces a BASE64 representation of this image. Then, it builds an INSERT statement that contains it in the third field, as described in “Standard encoding” on page 82.

Note: if you use standard TensorFlow capabilities to decode the BASE64 string of the input, you must replace ‘+’ with ‘-’ and ‘/’ with ‘_’, which is considered a web-safe representation. It is highly probable that you must use the web-safe encoding. For more information, see the [TensorFlow documentation](#).

Web-safe encoding

The following command produces a BASE64 representation of an input image:

```
$ base64 -w0 /images/image.jpg | tr '+/' '-_' | sed "s/.*/INSERT INTO DATA_IMAGES VALUES (1, 'image.jpg', '&');/" > /images/image.sql
```

Standard encoding

The following command builds an INSERT statement of an input image:

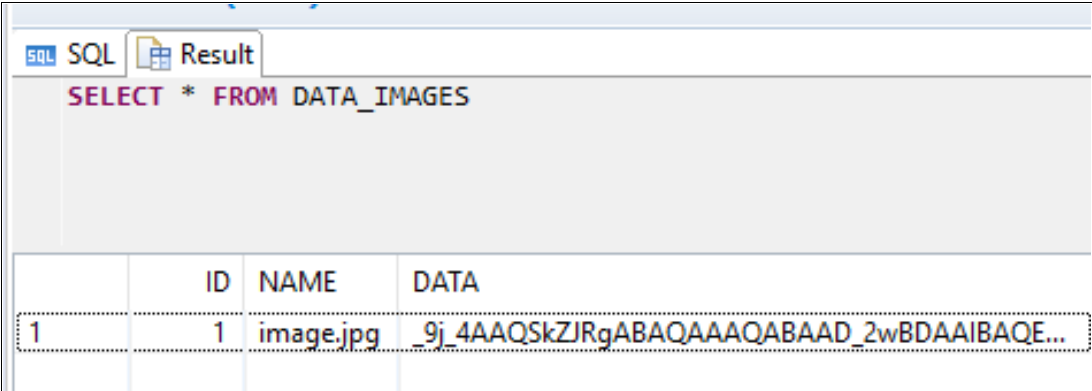
```
$ base64 -w0 /images/image.jpg | sed "s/.*/INSERT INTO DATA_IMAGES VALUES (1, 'image.jpg', '&');/" > /images/image.sql
```

It is not convenient to paste a long INSERT statement into SAP HANA Studio. Therefore, use a command-line client application, such as hdbsql, to move the image into SAP HANA by using the image.sql file. Use the following command to run the INSERT statement from the hd1adm user:

Note: Set permissions on the /images/image.sql file for the hd1adm user to access its content.

```
$ su - hd1adm -c 'hdbsql -u EML -p [mypassword] -i 35 -I /images/image.sql'
SELECT * FROM EML.DATA_IMAGES;
```

Then, check if SAP HANA contains the newly inserted record. The output is only one row, assuming that you inserted only one test image (see Figure 4-15).



The screenshot shows the SAP HANA Studio SQL Results window. The SQL editor contains the query `SELECT * FROM DATA_IMAGES`. The results pane displays a single row with the following data:

	ID	NAME	DATA
1	1	image.jpg	_9j_4AAQSkZIRgABAQAAQABAAD_2wBDAAIBAQE...

Figure 4-15 Test image output

4.3.7 Creating input and output types and a parameters table

The following SQL statements create SQL types that correspond to the input and output of the imagerec model. The third type corresponds to the parameters that are needed to start the model, such as a target remote-source, port, and others:

```
DROP TYPE IMAGEREC_INPUT;
CREATE TYPE IMAGEREC_INPUT AS TABLE ("DATA" CLOB);

DROP TYPE IMAGEREC_OUTPUT;
CREATE TYPE IMAGEREC_OUTPUT AS TABLE ("OBJECTID" FLOAT, "TOP" FLOAT, "LEFT"
FLOAT, "WIDTH" FLOAT, "HEIGHT" FLOAT);

DROP TYPE IMAGEREC_PARAMS;
CREATE TYPE IMAGEREC_PARAMS AS TABLE ("Parameter" VARCHAR(100), "Value"
VARCHAR(100));
```

4.3.8 Creating EML procedure wrapper

After defining the parameters of the procedure, we can register an AFL wrapper that contains the model's input and output signatures. The wrapper-creating API requires placing the signature and model location-related parameters into a special "configuration table":

```
DROP TABLE SIGNATURE;
CREATE COLUMN TABLE SIGNATURE (
    POSITION          INTEGER,
    SCHEMA_NAME      NVARCHAR(256),
    TYPE_NAME        NVARCHAR(256),
    PARAMETER_TYPE   VARCHAR(7)
);
```

We now populate the wrapper configuration table:

```
INSERT INTO SIGNATURE VALUES (1, CURRENT_SCHEMA, 'IMAGEREC_PARAMS' , 'in');
INSERT INTO SIGNATURE VALUES (2, CURRENT_SCHEMA, 'IMAGEREC_INPUT' , 'in');
INSERT INTO SIGNATURE VALUES (3, CURRENT_SCHEMA, 'IMAGEREC_OUTPUT' , 'out');
```

It is only task left is to create the AFL wrapper by using the configuration table and other parameters.

Here, the first two parameters define the AFL functions for which you are creating a wrapper. Next, the schema name is determined, in which the wrapper procedure is to be created. Then, the name of the wrapper procedure is determined, followed by the signature of the wrapper defined inside SIGNATURE:

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP(CURRENT_SCHEMA, 'IMAGEREC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('EML', 'PREDICT', CURRENT_SCHEMA,
'IMAGEREC', SIGNATURE);
```

At this stage, the EML.IMAGEREC procedure is called from SQL and expects an input table of type IMAGEREC_INPUT, an output table of type IMAGEREC_OUTPUT, and a configuration table IMAGEREC_PARAMS, which contains TMS-related information.

Completing the input and calling the remote inferencing by way of TMS

The last step is to populate the input and configuration tables and call the EML.IMAGEREC procedure awaiting for the response from TMS.

Populate input

Prepare the input by creating a corresponding table and inserting the BASE64 value of a single row of data from the table DATA_IMAGES into it:

```
DROP TABLE IMAGEREC_IN_TAB;  
CREATE TABLE IMAGEREC_IN_TAB LIKE IMAGEREC_INPUT;  
INSERT INTO IMAGEREC_IN_TAB (SELECT "DATA" FROM DATA_IMAGES WHERE  
ID = 1);
```

Note: We chose only one row of data as an input to the imagerec model, which is identified by the ID = 1. Although the corresponding usage is not covered in this chapter, EML also supports the execution of inferencing on a batch of input data. This process can be done by registering a PREDICTM AFL wrapper. For more information about PREDICT and PREDICTM functions and their differences^a, see the EML documentation

a. SAP HANA External Machine Learning Library, SAP HANA Platform 2.0 SPS 03, Document Version: 1.1 – 2018-10-31

Preparing output table

This section prepares an empty table that holds the output from the TMS instance:

```
DROP TABLE IMAGEREC_OUT_TAB;  
CREATE TABLE IMAGEREC_OUT_TAB LIKE IMAGEREC_OUTPUT;
```

Specifying remote-connection details

As described in “Preparing output table” on page 84, the important parameters of the configuration table specify the created remote connection and the model of interest that are served by the TMS instance of that remote connection:

```
DROP TABLE IMAGEREC_PARAMS_TAB;  
CREATE TABLE IMAGEREC_PARAMS_TAB LIKE IMAGEREC_PARAMS;  
INSERT INTO IMAGEREC_PARAMS_TAB VALUES ('Model', 'imagerec');  
INSERT INTO IMAGEREC_PARAMS_TAB VALUES ('RemoteSource', 'TMS');  
INSERT INTO IMAGEREC_PARAMS_TAB VALUES ('Deadline', '10000');
```

Calling the imagerec procedure

Remote processing of the call on TMS is performed by the next line:

```
CALL IMAGEREC(IMAGEREC_PARAMS_TAB, IMAGEREC_IN_TAB, IMAGEREC_OUT_TAB) WITH  
OVERVIEW;
```

Upon successful execution, you see the output as shown in Figure 4-16, which is similar to Figure 4-14 on page 81.

CALL IMAGEREC(IMAGEREC_PARAMS_TAB, IMAGEREC_IN_TAB, IMAGEREC_OUT_TAB) WITH OVERVIEW			
	variable	table	
1	P3	"EML"."IMAGEREC_OUT_TAB"	

Figure 4-16 Calling the imagerec procedure

The last step is to read the result from IMAGEREC_OUT_TAB by running:

```
SELECT * FROM IMAGEREC_OUT_TAB;
```

4.4 Conclusion

We used SAP HANA Studio to run the SQL code, which triggered input processing by using a model that was deployed in TMS. This process us a good starting point for understanding how EML works. However, the usage scenarios are not limited to this example only.

The next step is to implement an SAP application that calls the EML `imagerec` procedure from ABAP code through OpenSQL's `CALL` statement. This process enables the ABAP → TMS scenario and allows for the implementation of client-centric AI applications.



Benefits of modernizing SAP applications with IBM AI

This chapter describes the benefits of modernizing SAP applications and includes the following topics:

- ▶ 5.1, “Enterprise agenda” on page 88
- ▶ 5.2, “IBM Watson Studio” on page 88
- ▶ 5.3, “IBM Watson OpenScale” on page 89
- ▶ 5.4, “AutoAI” on page 91

5.1 Enterprise agenda

Nearly all enterprises are in the process of adapting or already adapted Artificial Intelligence (AI) into their business processes. With technology maturing and more implementations across business units, new challenges for companies developed.

It is not enough to develop a working machine learning (ML) model and attach it to a business application. ML models are moving to the core of the computing landscape. With the increased usage come problems monitoring, and checking and governing the results of predictions all across the lifecycle of current deployments. Many enterprise companies are deeply invested in the following areas:

- ▶ Advancing (core) AI: Driven by the pace of the industry, they must learn more, master language, and develop reasoning.
- ▶ Trusting AI: Develop trust through fairness, robustness, explainability, and transparency.
- ▶ Scaling AI: Manage, operate, and automate AI throughout its complete lifecycle.

The following section describes a few examples of IBM and IBM Watson offerings that address these areas and help enterprises build applications quicker and with more confidence, maintain and monitor their operation, and check that they provide the correct business outcomes.

5.2 IBM Watson Studio

IBM Watson Studio is an integrated environment to efficiently use data, AI, ML, and deep learning in any business. It simplifies and accelerates the preparation of data, exploration of data, and modeling and training of algorithms. IBM Watson Studio realizes the following benefits:

- ▶ Brings AI algorithms to where the data is stored
- ▶ increases the productivity of the team of data scientists, analysts, developers, and other experts
- ▶ implements the data science lifecycle from insight to prediction and optimization

IBM Watson Studio is well suited for hybrid clouds. It provides mission critical performance, security, and governance in public or private clouds, on-premises, and on the desktop, including IBM Cloud Pak® for Data.

Together with IBM Watson Machine Learning, IBM Watson Studio makes it easy to create, develop, and deploy AI models. IBM Watson Machine Learning enables businesses to quickly and simply harness the power of AI.

The strength of IBM Watson Machine Learning includes:

- ▶ Deploy AI models that are built with IBM Watson Studio and open source tools.
- ▶ Retrain models dynamically.
- ▶ Generate APIs automatically to enable AI powered applications.
- ▶ Integrate with IBM Watson OpenScale™ to manage models.
- ▶ Easy end-to-end management and deployment of models.

IBM Watson Studio is the interface to bring a vast amount of tools (many open source), such as TensorFlow and PyTorch, into one environment to design and develop ML applications as individuals, or small or large groups.

Many tools and features are included in IBM Watson Studio, for example:

- ▶ **AutoAI.**
Generate a working ML model and test hyperparameters with a few clicks and no code development.
- ▶ **Collaborative data science.**
Use Jupyter Notebooks, collaboration tools, project features, and version control to enable teams to find the ideal model for production.
- ▶ **Enhanced visual modeling.**
Use a visual tool with drag functions to optimize data and learning capabilities.
- ▶ **Automated deep learning.**
Use an intuitive interface to automatically enhance the neural network in the modeler without writing any code.

For more information, see the [IBM Watson Studio web page](#).

5.3 IBM Watson OpenScale

AI is often seen as a “black box”; that is, some complicated code that provides mysterious insights and recommendations. People do not trust what they cannot fully debug and trace, and rightfully so.

IBM Watson OpenScale™ provides a set of monitoring and management tools that help build trust and implement control and governance structures around AI investments.

ML models are constantly evolving and the quality of data and results change over time, often AI teams cannot keep up with the pace of these changes. IBM Watson OpenScale is designed to give a clear and accurate view of running AI systems that are helping to manage, optimize, and monitor AI applications across their lifecycle.

IBM Watson OpenScale tracks and measures outcomes from AI models and helps ensure they remain fair, explainable, and compliant wherever they are built or are running. IBM Watson OpenScale is designed as an open platform that operates with various model development environments and various open source tools, including TensorFlow, Keras, SparkML, Seldon, AWS SageMaker, and AzureML.

5.3.1 Monitors

Model monitors allow IBM Watson OpenScale to capture information about the deployed model, evaluate transaction information, and calculate metrics. It checks for compliance and puts safeguards in place. The following monitors can be enabled:

- ▶ *Fairness monitor* scans your deployments for biases to ensure fair outcomes across the population. IBM Watson OpenScale helps organizations maintain regulatory compliance by tracing and explaining AI decisions across workflows and intelligently detecting and correcting bias to improve outcomes.
- ▶ *Quality monitor* (previously known as *accuracy monitor*) establishes how well a model predicts relevant outcomes. The quality monitor scans the requests that are sent to a model deployment to establish how well the specific model predicts outcomes. Quality metrics are calculated hourly, when IBM Watson OpenScale sends manually labeled feedback data set to the deployed model.

- *Drift monitor* detects changes in accuracy when it starts receiving data that is inconsistent with how it was trained. The drift monitor scans the requests that are sent to the model deployment to ensure that the model is robust and does not drift over time. Drift in model predictions can occur because the requests that are sent to the model are requests that are similar to samples in the training data where the model struggled, or because the requests are becoming inconsistent with the training data the model originally used.

Explainability provides transparency in models by showing what lead the model to make specific predictions.

5.3.2 Data sets

For many of the monitors that are described in 5.3.1, “Monitors” on page 89, IBM Watson OpenScale considers the following factors:

- Training data that was used to train the model (optional).
- Transaction data inputs and requests, outputs, and responses that are going to and from the deployed models that are stored in the payload tables of the data mart.
- Feedback data with labeled predictions to measure the effectiveness of predictions and when retraining is needed, which is stored in the feedback table of the data mart.

5.3.3 Checking model drift

Over time, the data that is coming into the model can vary from the initial training data, which affects the accuracy of our model and the business processes that use the model. IBM Watson OpenScale analyzes transactions to detect drift in model accuracy and drift in data.

Drift in model accuracy occurs if there an increase in transactions that are similar to those that the model did not evaluate correctly in the training data.

Drift in data estimates the drop in consistency of the data at runtime as compared to the characteristics of the data at training time.

5.3.4 Trust in AI and IBM Watson OpenScale

IBM Watson OpenScale is designed to de-mystify the process that AI models use and make them understandable by business users. It was built with the four pillars of trust in mind:

- **Explainability:** Provides tools and methods to understand why a model features a specific prediction.
- **Fairness:** Detect, show, and mitigate bias.
- **Accuracy:** Identify and combat model drift. Ensure that models are resilient to changing situations.
- **Openness:** Build trust through open source community contributions and reviews.

Fundamentally, IBM Watson OpenScale is aligning model performance with required business outcomes.

5.3.5 Open Source Initiatives

IBM Watson OpenScale is sponsoring several open source projects around fairness, explainability, and management of AI models. These projects can be downloaded and used at no cost. Enterprise customers find these offerings on the IBM Cloud readily available for quick development and deployment.

Open-source projects include the following examples:

- ▶ AI Fairness 360 open source Toolkit includes over 70 fairness metrics and checkers, 10 bias mitigators, and industry tutorials.
- ▶ AI Explainability 360 includes eight explainability algorithms, two metrics, and industry tutorials.
- ▶ Adversarial Robustness Toolkit includes a toolkit to attack and defend AI models.

5.4 AutoAI

AutoAI automates data preparation, model development, feature engineering, and hyperparameter tuning when implementing an AI application. AutoAI can help with implementing good ML models but also discover questions to ask of the data. It supports experimentation, model modification deployment, and governance steps, all integrated into one user experience in the IBM Cloud or on-premises using the IBM Cloud Pak® for Data through IBM Watson Studio.

In a sense, it is AI for AI. The AutoAI tool automatically analyzes the data and generates several candidate model pipelines that are optimized for the prediction problem. These model pipelines are created over time as AutoAI algorithms learn more about the data set and discover data transformations, estimator algorithms, and parameter settings that work best for the specific problem.

Results are displayed on a leaderboard that shows the automatically generated model pipelines, which are ranked according to the optimization objective and encourage further experiments.

With AutoAI ML, models can be created and deployed in minutes without writing a single line of code. However, it is possible for skilled data scientists to interact with AutoAI at any stage to apply their knowledge and optimize the model to fit their needs.

5.4.1 AutoAI in data science process

AutoAI is helping in the following parts of the data science process:

- ▶ Data cleansing: Detect, identify, and eliminate incorrect or inconsistent data.
- ▶ Feature engineering: Finding properties and characteristics that are independent and informative is a difficult part of ML. AutoAI helps identify the most important features.
- ▶ Model building and hyperparameter tuning: Create and choose many different ML models, apply different parameters, and test them all to provide a comprehensive list of well-suited ML models without human interaction.

5.4.2 AutoAI benefits

AutoAI realizes the following benefits:

- ▶ **Building models faster:** Because AutoAI prepares data, identifies features, performs optimizations, and generates models much faster than humans can do by themselves, the best model can be chosen quickly.
- ▶ **Overcoming the skills gap:** ML is not a simple topic. To effectively deploy models, skilled data scientists are required throughout the entire process. AutoAI is making it possible for industry specialists to develop insight into the business without relying on experts in ML.
- ▶ **Uncovering more use cases:** Because exploring different models is quicker, more time can be spent to explore other opportunities in the data.
- ▶ **Identifying key predictors:** The auto-feature engineering option makes it simpler to derive more insight from existing data.
- ▶ **Ranking and exploring models:** By building and comparing multiple possible model pipelines, the best and most efficient model can be chosen quickly.
- ▶ **Deploying models easily through AutoAI-generated pipelines:** The deployed models can be accessed later and used through calls to a REST API.

5.4.3 Creating a machine learning model by using AutoAI

This section shows an example of how to create an ML model with IBM Watson AutoAI without the need to write a single line of code.

Creating a machine learning service

Complete the following steps to create an ML service:

1. Log in to a cloud account. Then, select **Catalog** from the top menu, select the **AI** category on the left side, and then, click **Machine Learning**, as shown in Figure 5-1.

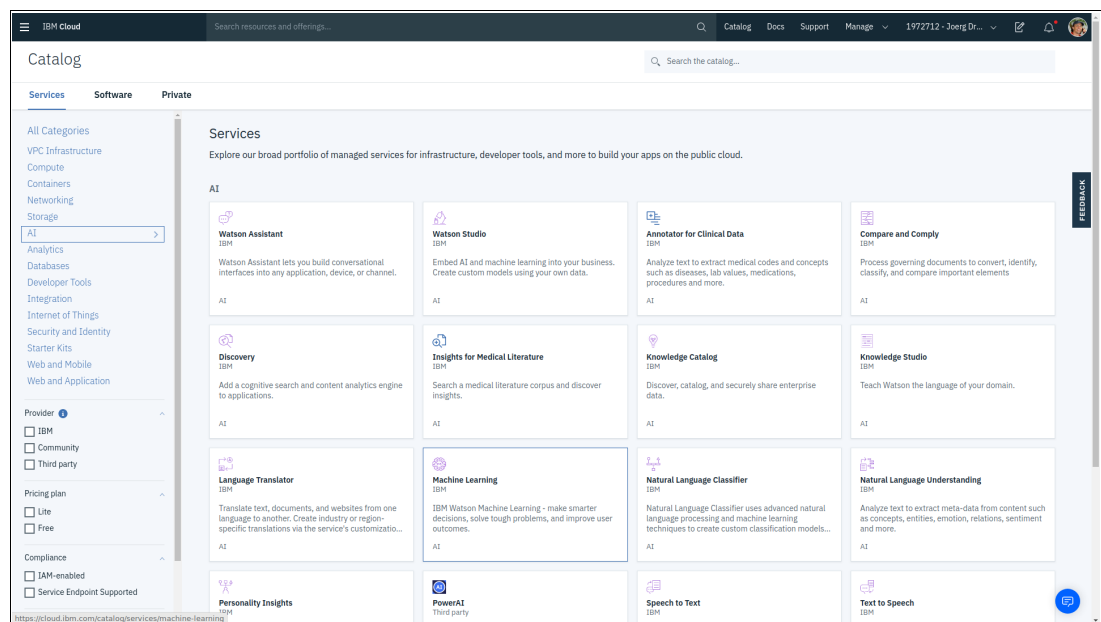


Figure 5-1 Catalog select

2. Select the location and your plan. Then, enter a name for the Machine Learning Service. Click **Create** on the right of the window, as shown in Figure 5-2.

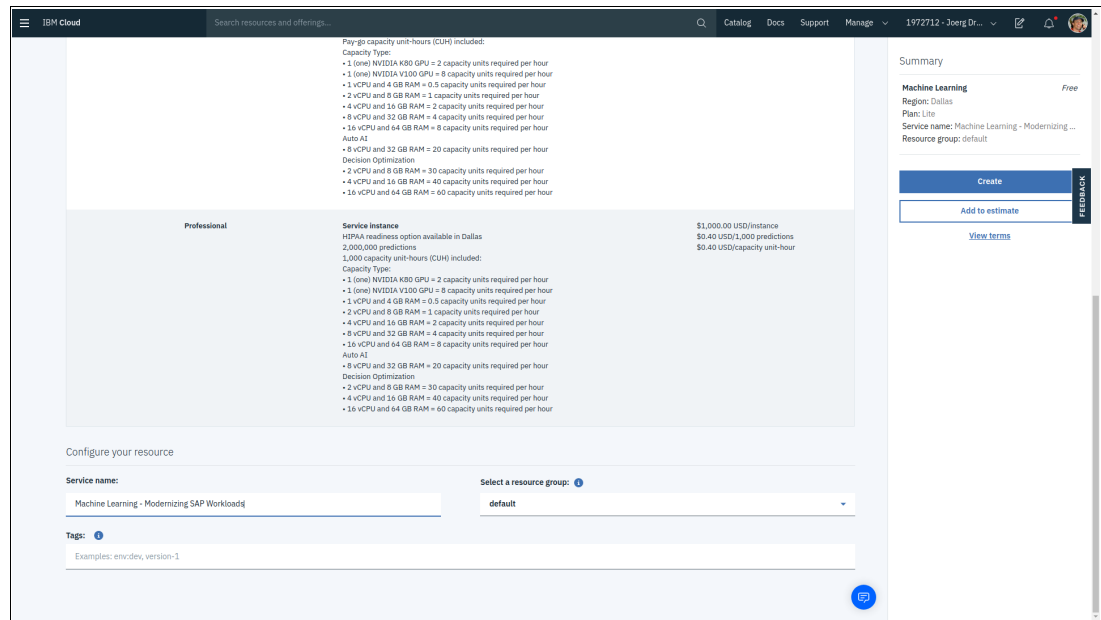


Figure 5-2 Create ML service

To access the new Machine Learning Service, a set of credentials must be created. These credentials are used to identify and access the service from an outside source. Complete the following steps:

1. In the Machine Learning Service window, select **Credentials** from the left menu; then, click **Create Credentials**, as shown in Figure 5-3.

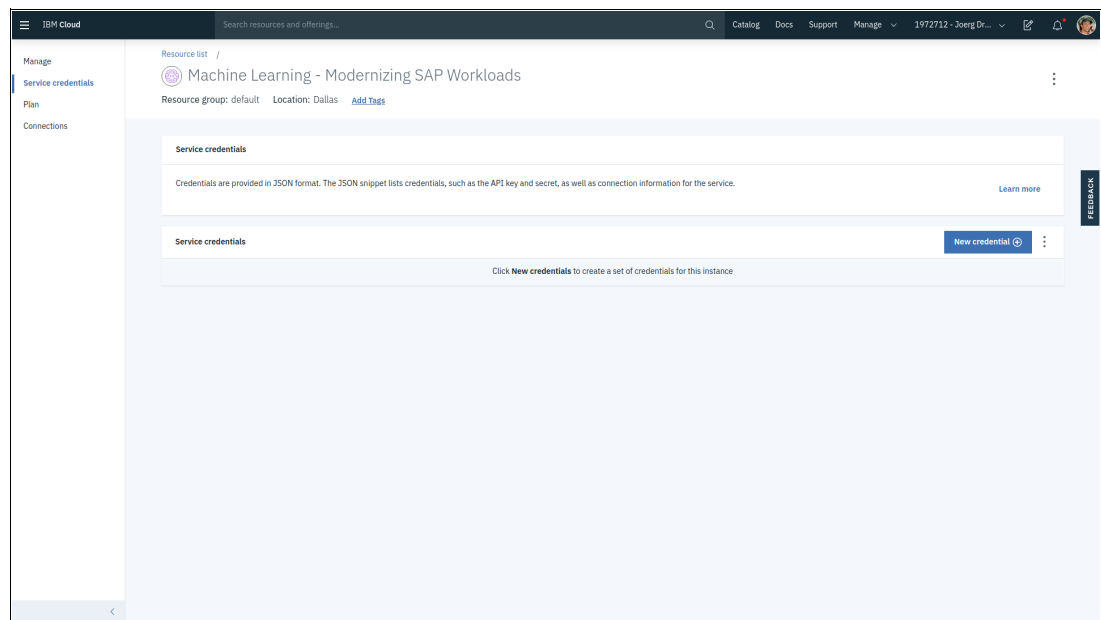


Figure 5-3 New credentials

2. Enter a name to identify these credentials and click **Add**, as shown in Figure 5-4.

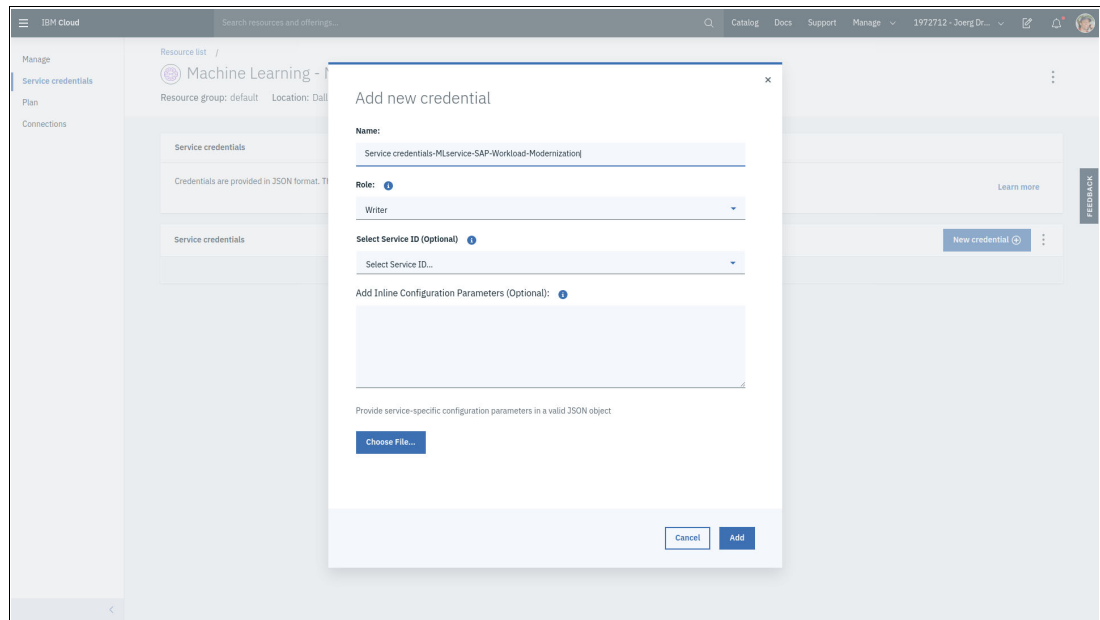


Figure 5-4 Add credentials

After the credentials are created, they can be viewed by clicking **View Credentials**, as shown in Figure 5-5. Make a note of these credentials because these are needed later by the AutoAI Experiment Service to use this Machine Learning Service.

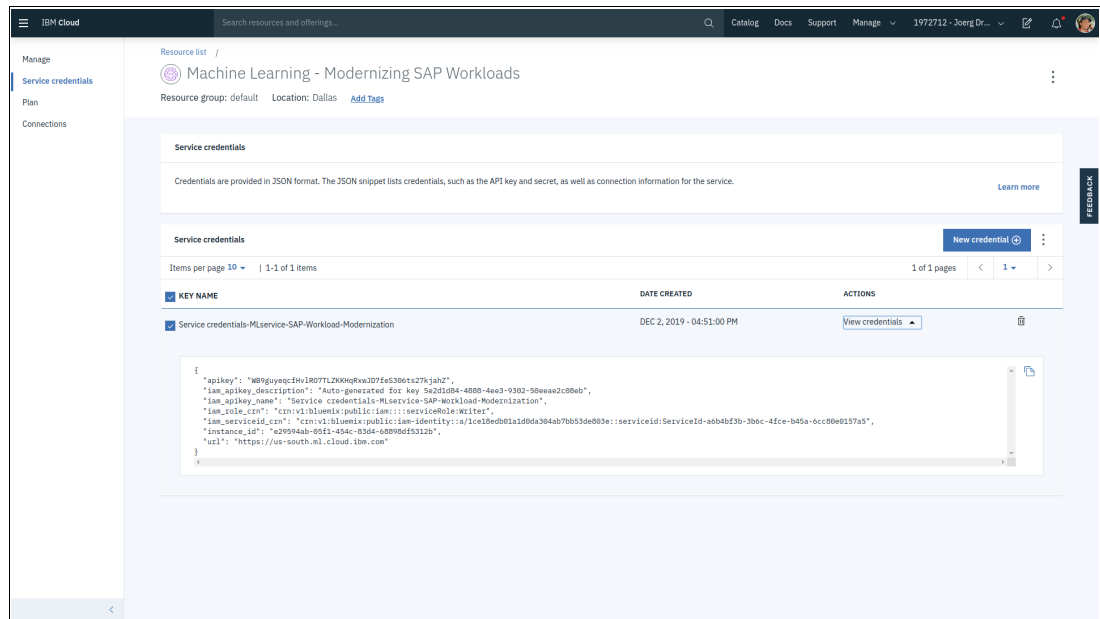


Figure 5-5 View credentials

Creating an IBM Watson Studio Service and Project

The AutoAI Service is part of IBM Watson Studio. To use AutoAI, an IBM Watson Studio Service must be created. Complete the following steps:

1. In the initial window, select **Catalog** from the top menu. Then, select the **AI** category on the left side, and click **IBM Watson Studio**, as shown in Figure 5-6.

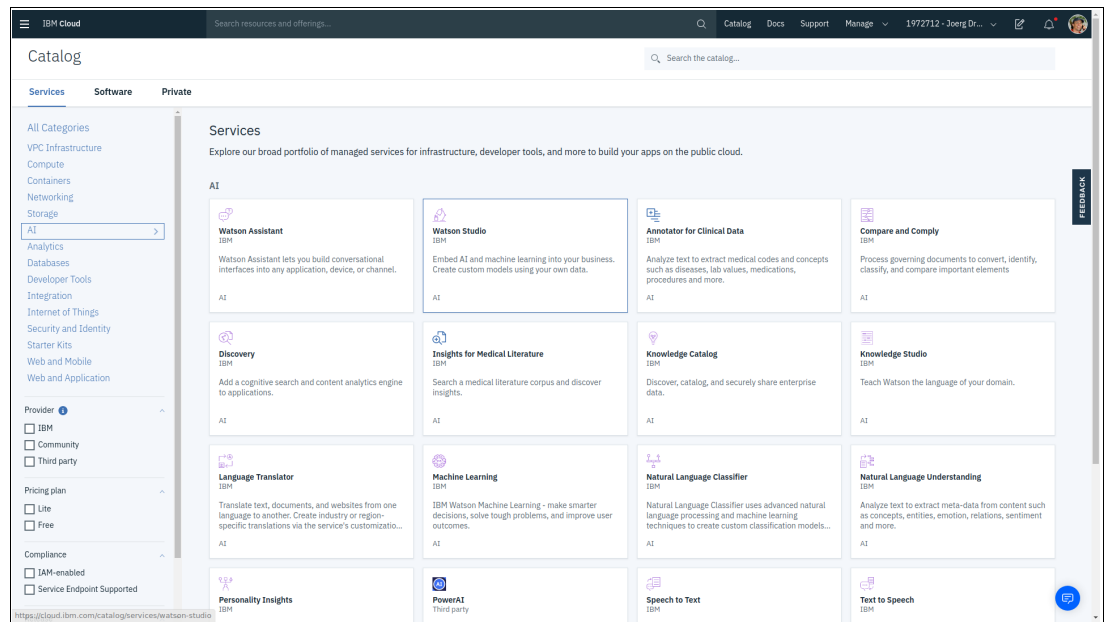


Figure 5-6 Create an IBM Watson Studio

2. Select your region and plan; then, enter a name for the IBM Watson Studio Service. Click **Create** to create the service, as shown in Figure 5-7.

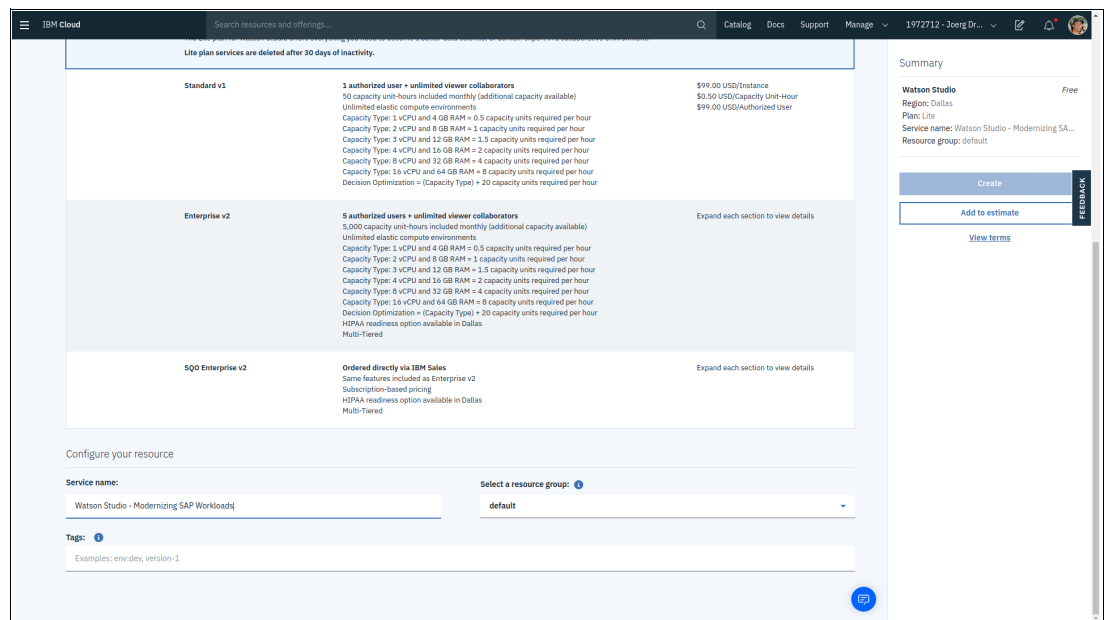


Figure 5-7 Add IBM Watson Studio

After the IBM Watson Studio Service is ready, it can be started and a new project can be created. Complete the following steps:

1. From the Home window, select the pull-down menu in the upper left and select **Resource List**.
2. In the next window, click the created IBM Watson Studio Service to start IBM Watson Studio, as shown in Figure 5-8.

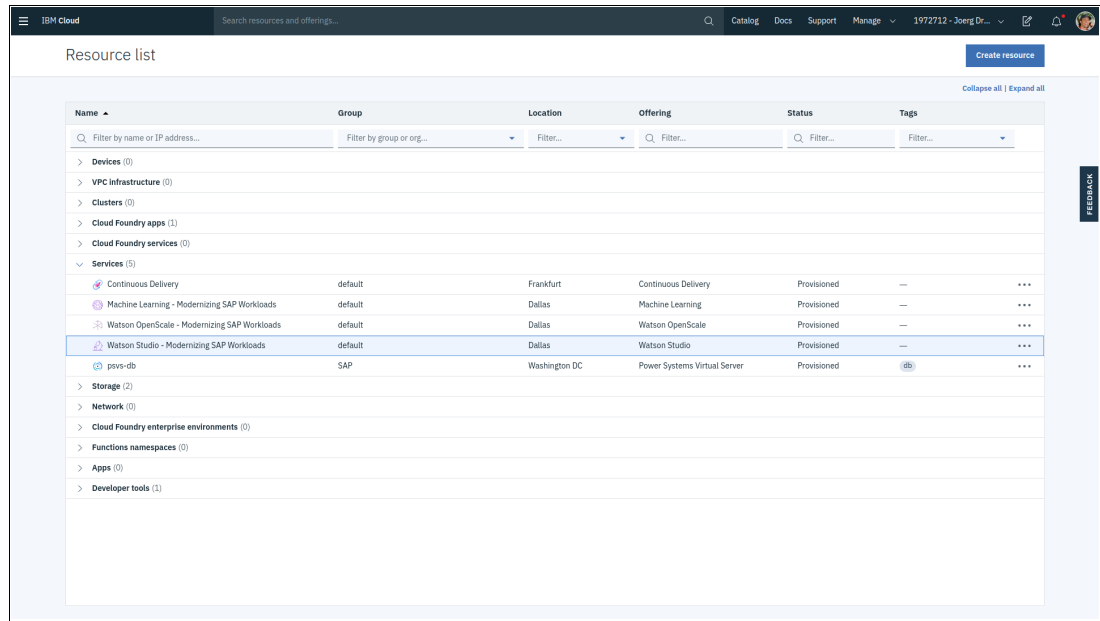


Figure 5-8 Start IBM Watson Studio

3. When IBM Watson Studio start page is displayed, click **Get Started** to start IBM Watson Studio, as shown in Figure 5-9.

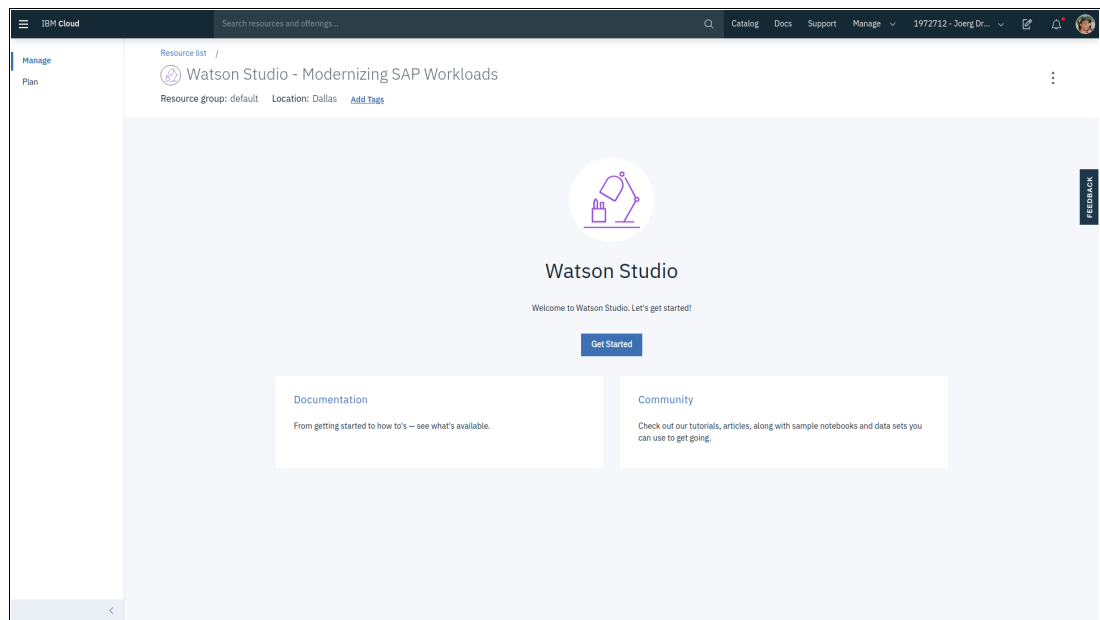


Figure 5-9 IBM Watson Studio: Get started window

After IBM Watson Studio is started, a project must be created to hold all the components of the AutoAI experiment. To create a project, click **Create New Project**. Then, enter the project name and click **Create**, as shown in Figure 5-10.

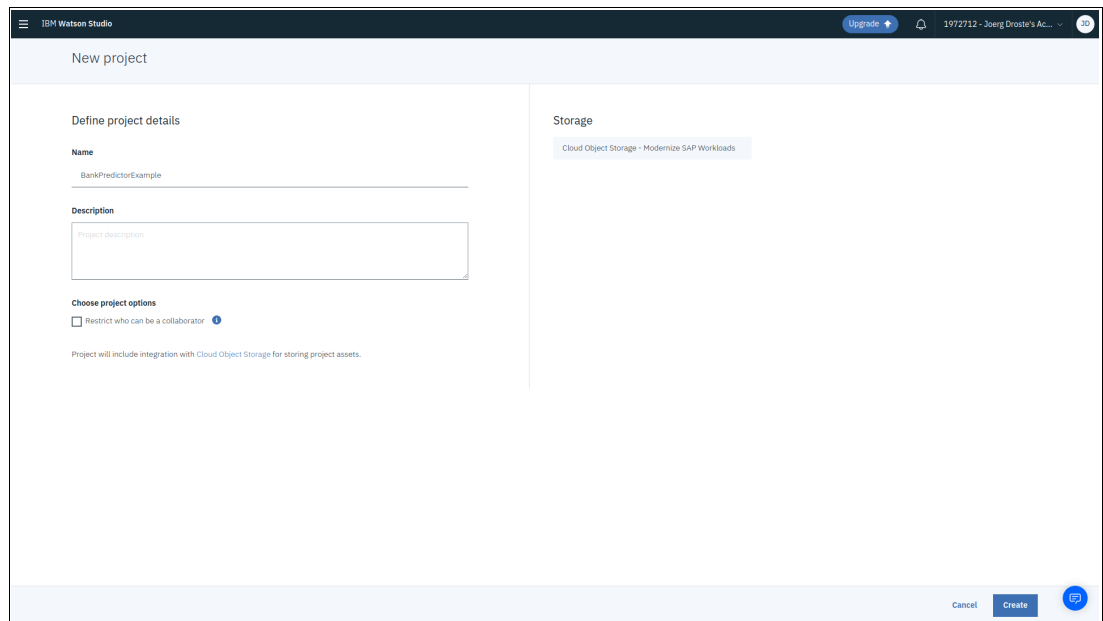


Figure 5-10 New project

For the sake of simplicity, this scenario uses example data that is supplied with IBM Watson Studio, although any data set can be imported into IBM Watson Object Storage. Complete the following steps to add data to the project:

4. Select **Explore Gallery** at the upper right of the window. Click the filter icon and select **Data Set**, as shown in Figure 5-11.

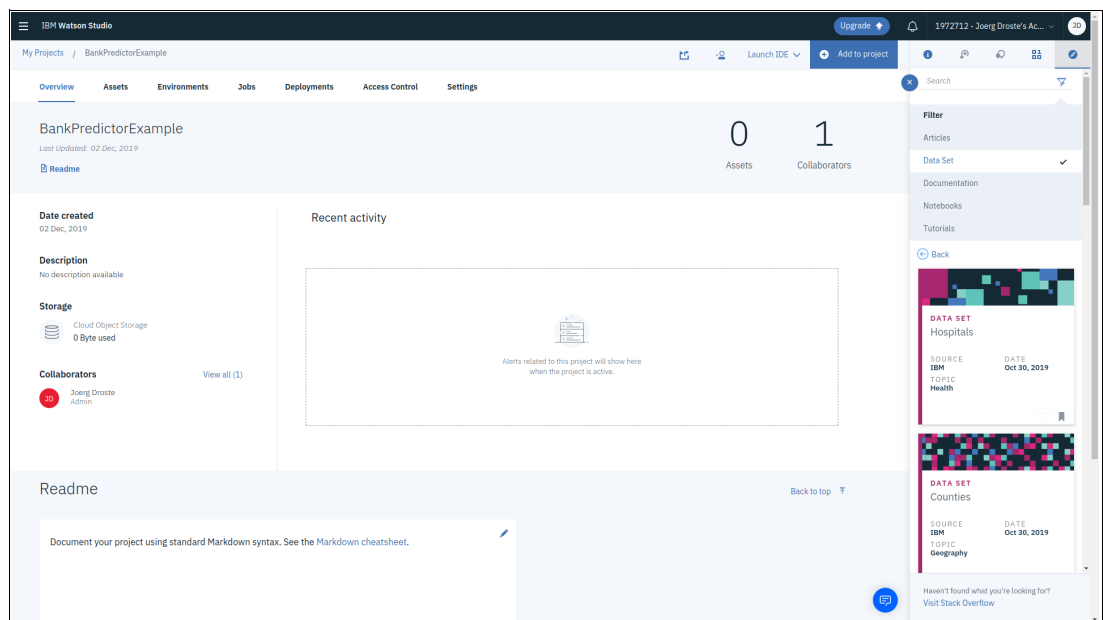


Figure 5-11 Filter data set

5. Scroll down in the list and click **UCI: Bank Marketing Data Set - full data set**. Click the data set title and in the next window, select **Add to Project**. Select the previously created project and choose **Add**, as shown in Figure 5-12.

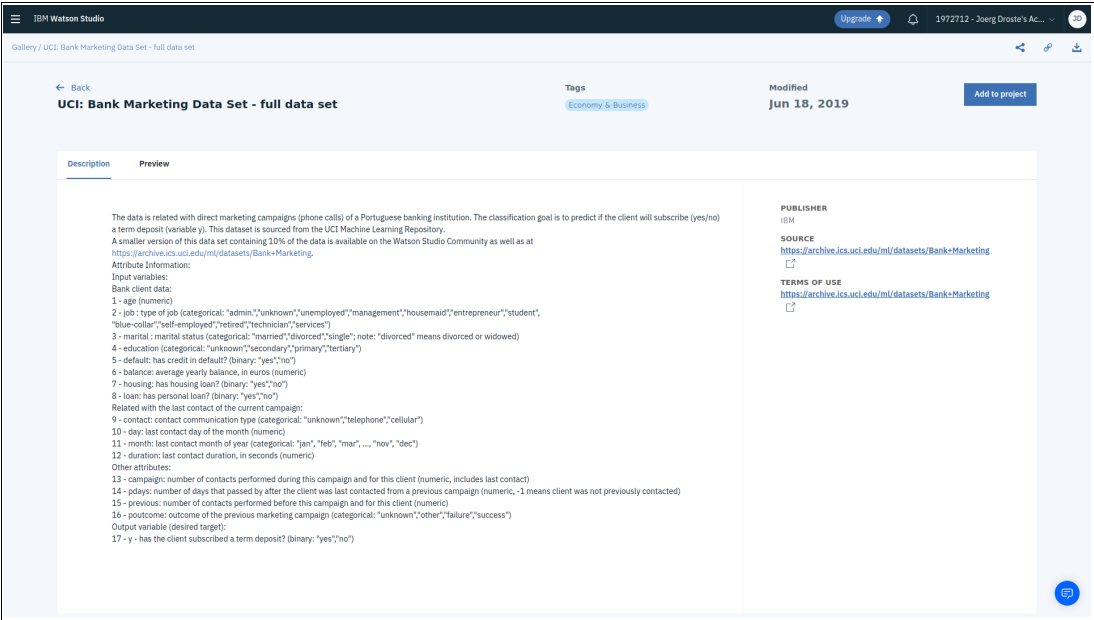


Figure 5-12 Add data to project

6. Return to the project and select the **Asset** tab. The newly imported data is listed under Data Assets, as shown in Figure 5-13.

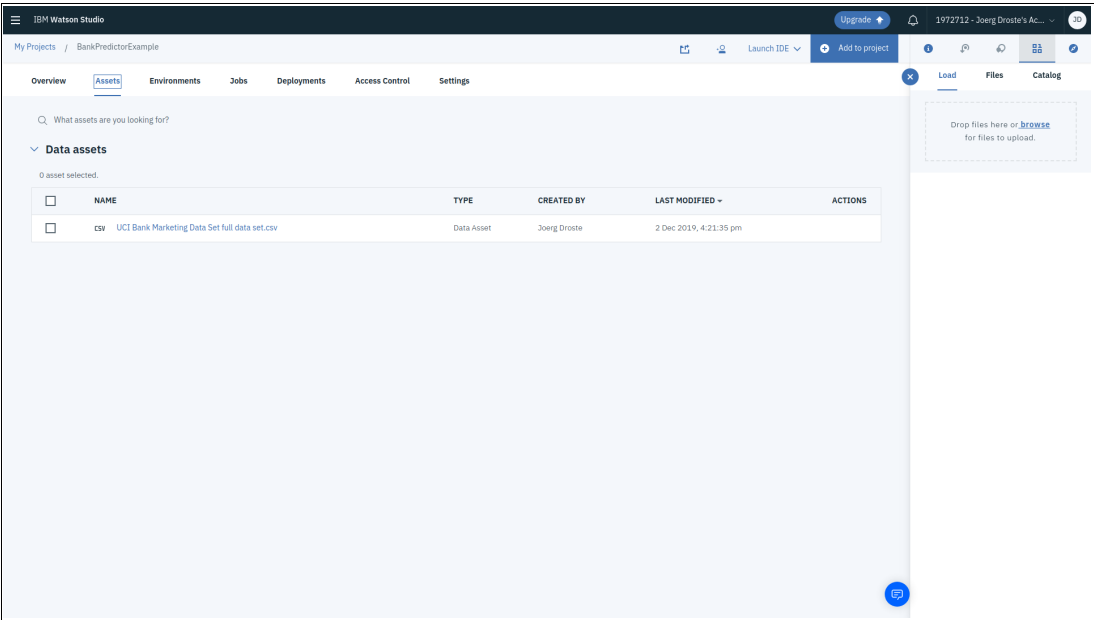


Figure 5-13 Project with data asset

Creating an AutoAI experiment

To use AutoAI to create an effective ML model, complete the following steps:

1. Click **Add to Project** in the upper right of the window, as shown in Figure 5-14. In the next window, select the AutoAI tile.

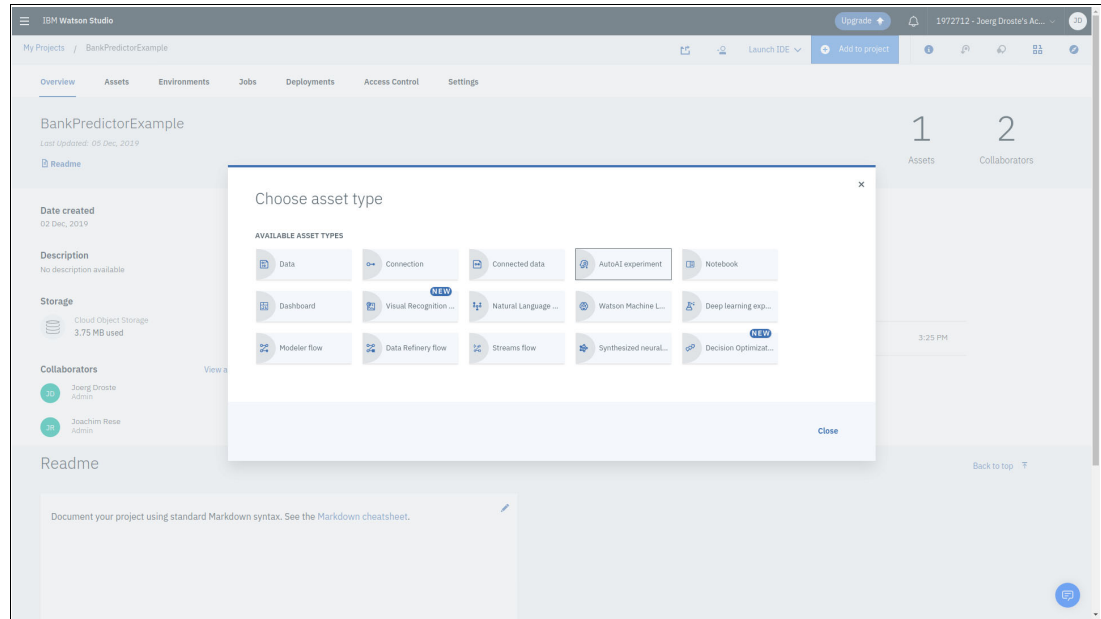


Figure 5-14 AutoAI asset type

2. In the next window, enter a name and description for the experiment or accept the provided text by selecting **From Sample**, as shown in Figure 5-15.

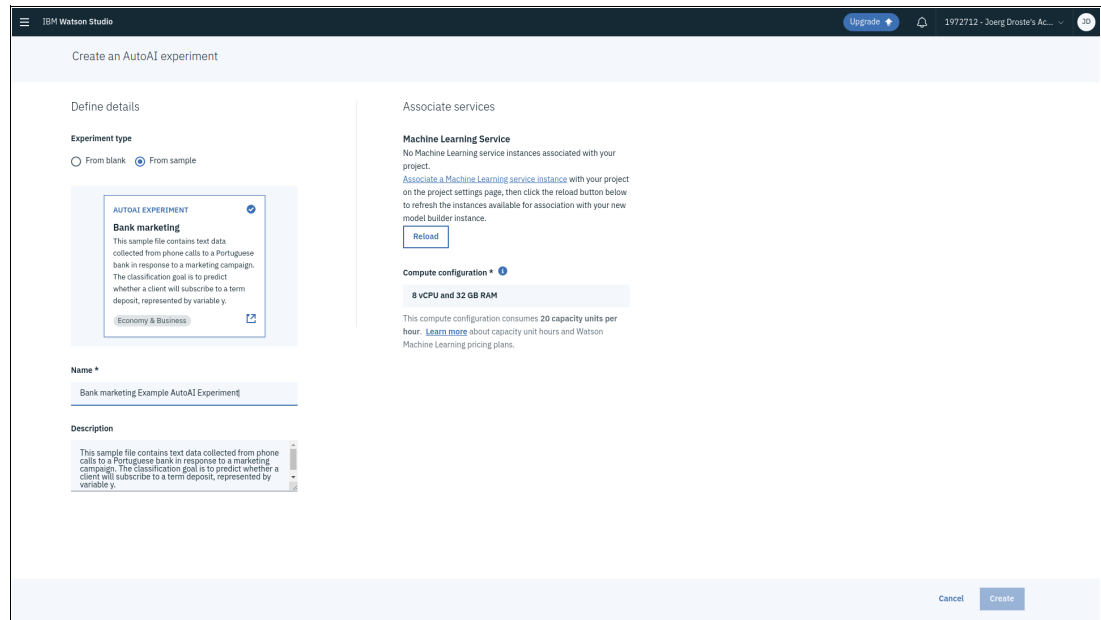


Figure 5-15 AutoAI experiment definition

Next, the ML instance must be added to the AutoAI experiment. Complete the following steps:

1. Select the link to Associate a Machine Learning Service instance.
2. Use the drop-down list and choose the ML instance that was created during the previous steps and click **Select**, as shown in Figure 5-16.

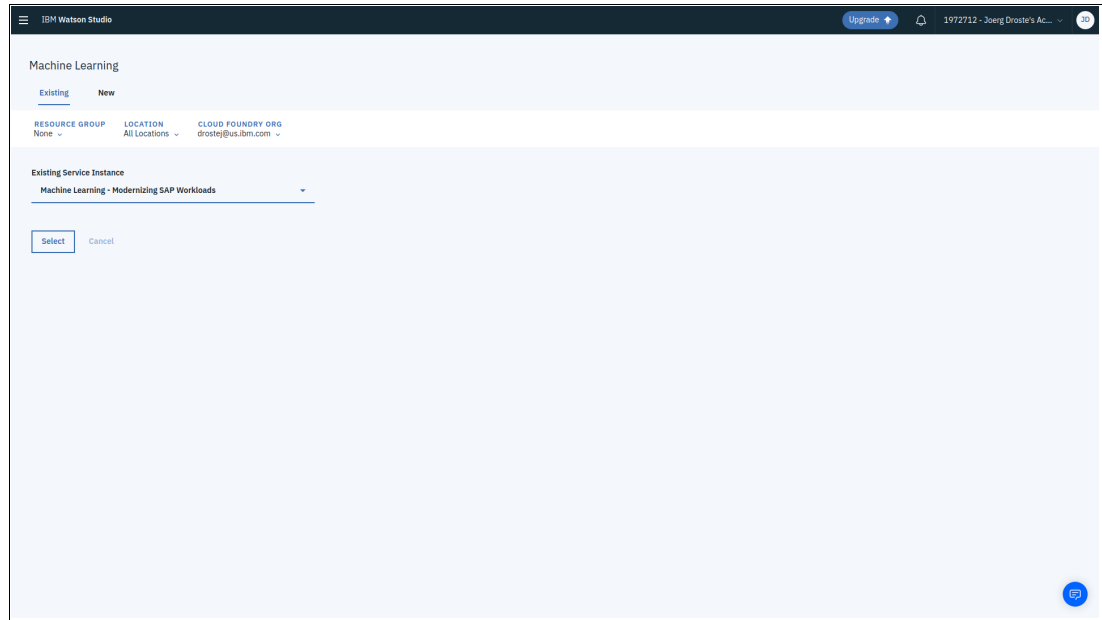


Figure 5-16 Adding Machine Learning Service

3. Click **Reload** (as shown in Figure 5-15) to import your Machine Learning Service definition. Then, click **Create**, as shown in Figure 5-17.

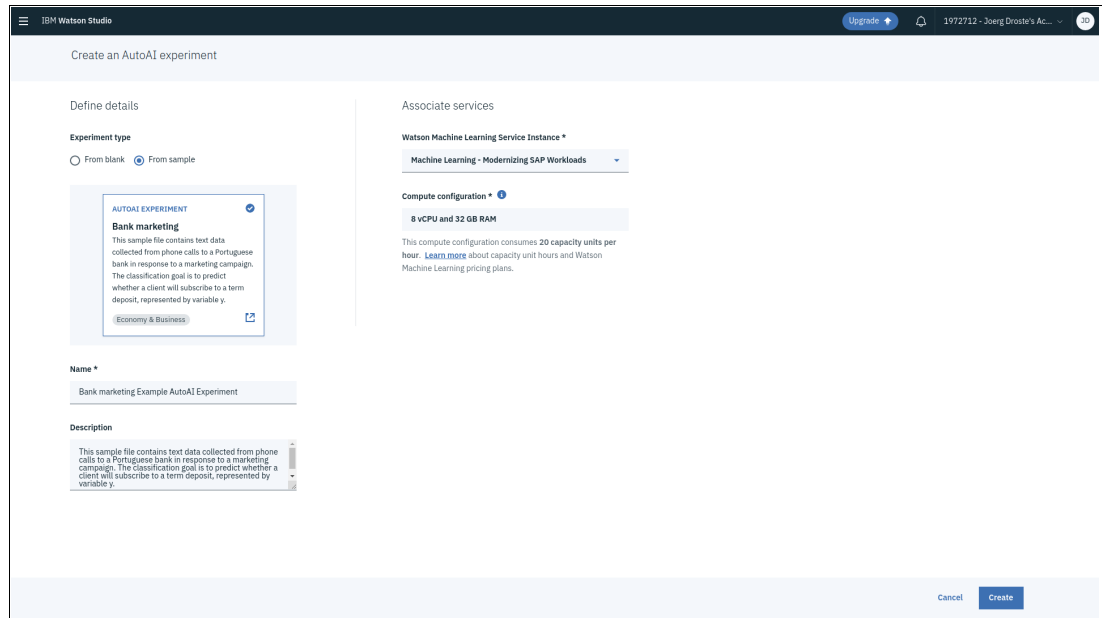


Figure 5-17 Create AutoAI experiment

4. As shown in Figure 5-18, select the experiment specifics:

- Select the data set that is available at the lower right.
- Select the column to hold the prediction.
- Verify the prediction type.

Configure the experiment by using the controls at the lower right. When ready, click **Run experiment**.

The screenshot shows the IBM Watson Studio interface for configuring an AutoAI experiment. The main area is titled 'Configure AutoAI experiment' and 'Bank marketing Example AutoAI Experiment'. A 'Select a source file' dialog is open, prompting the user to select or upload a data set. Below this, the 'Data source' table lists available data sets. The 'Bank marketing' data set is selected. To the right, the 'Select prediction column' table lists columns and their types. The 'y' column is selected as the prediction column. At the bottom, the 'Prediction type' is set to 'Binary Classification', the 'Positive class' is 'Yes', and the 'Optimized metric' is 'ROC AUC'. A 'Run experiment' button is visible at the bottom right.

Column name	Type
y	String
age	Integer
job	String
marital	String
education	String
default	String
balance	Integer
housing	String
loan	String
contact	String
day	Integer
month	String

Prediction column: y
Prediction type: Binary Classification
Positive class: Yes
Optimized metric: ROC AUC

Figure 5-18 AutoAI source file

After a few minutes, the results of the AutoAI experiment are displayed. Review the result and background information, as shown in Figure 5-18. The experiment can be repeated if necessary.

- After a model provides reasonable quality, hover over its row and choose **Save Model**. In the next window (see Figure 5-19), enter a name and description for the new ML model.

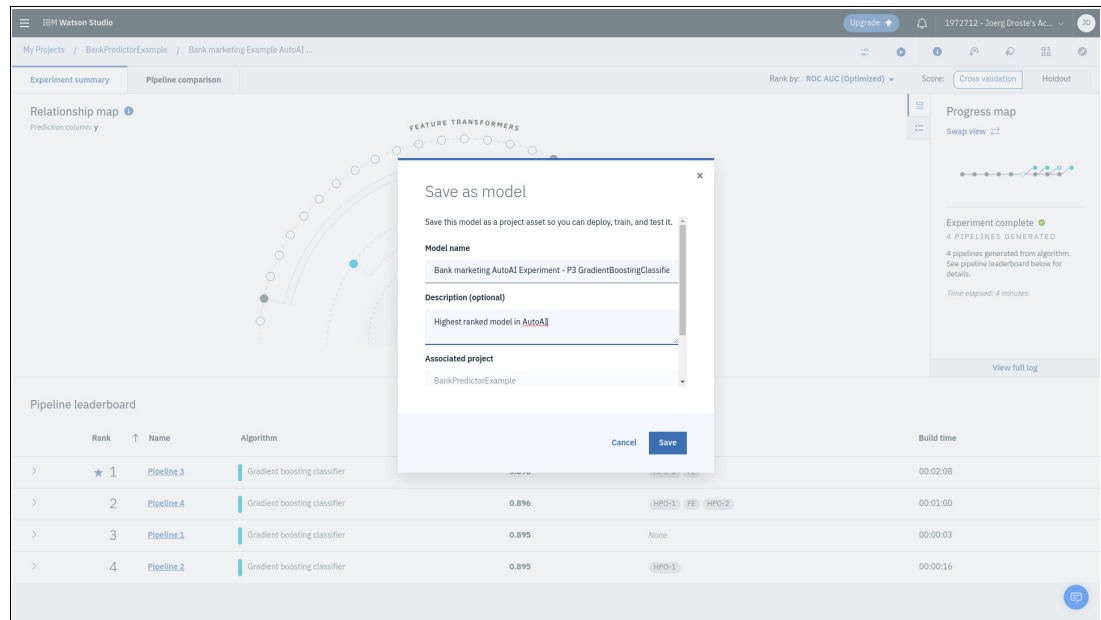


Figure 5-19 Save model

The model now appears in the project overview as an IBM Watson Machine Learning Model.

- The model must be deployed so that it can be used. To deploy this model, select the model menu and click **Deploy**, as shown in Figure 5-20.

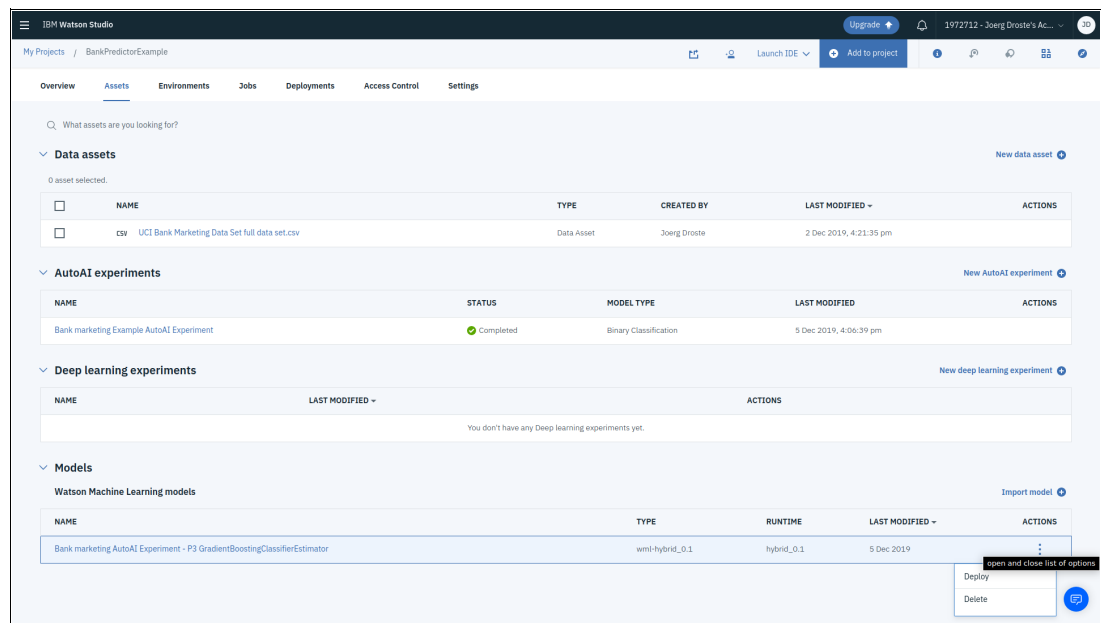


Figure 5-20 Deploy model

7. Click **Add Deployment**, as shown in Figure 5-21. Enter a name and description for the deployment and then, click **Save**.

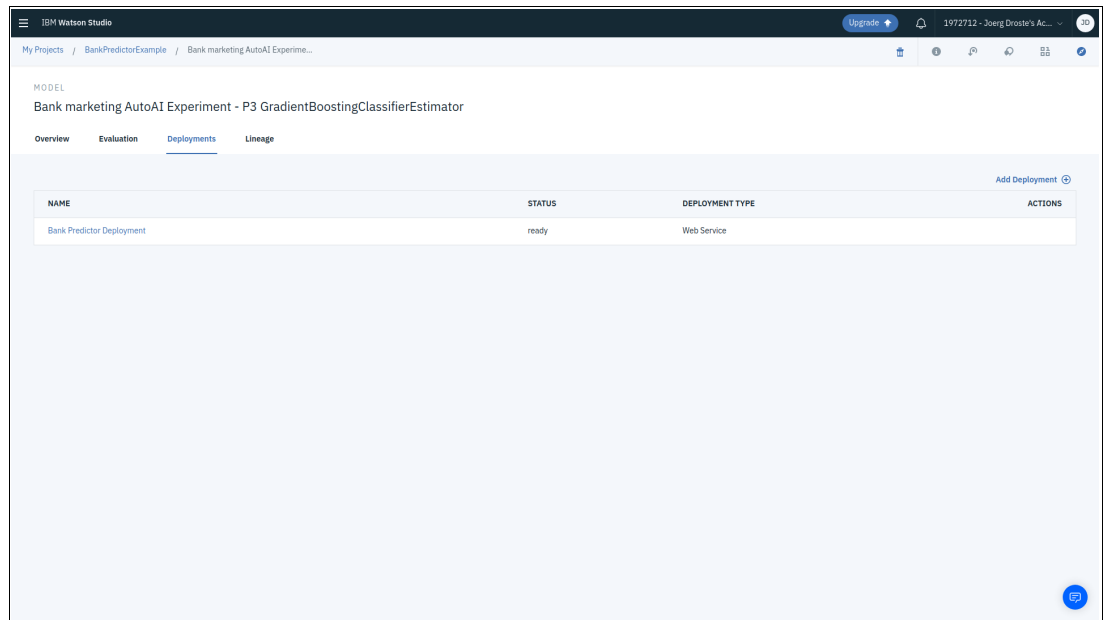


Figure 5-21 Deployment ready

The model is now deployed and can be used by other applications.

For more information about these IBM technologies, see the following resources:

- ▶ [Manage production AI with trust and confidence in outcomes](#)
- ▶ [IBM Cloud Doc](#)
- ▶ [IBM Watson OpenScale Demos](#)
- ▶ [Installing Trust](#)
- ▶ [Explainable AI](#)
- ▶ [Trust in AI from IBM Research](#)



A

node.js server code

This appendix provides the node.js server code.

node.js server code

Example A-1 shows the node.js server code.

Example A-1 File server/server.js

```
const appName = require(' ../../package').name;
const http = require('http');
const express = require('express');
const log4js = require('log4js');
const localConfig = require('./config/local.json');
const path = require('path');

const logger = log4js.getLogger(appName);
logger.level = process.env.LOG_LEVEL || 'info'
const app = express();
const server = http.createServer(app);

app.use(log4js.connectLogger(logger, { level: logger.level }));
const serviceManager = require('./services/service-manager');
require('./services/index')(app);
require('./routers/index')(app, server);

// Add your code here

/*****      START: Custom Implementation
*****/

// Import modules
const bodyParser = require('body-parser');
const https = require('https');
const yamljs = require('yamljs');
const IBMCloudEnv = require('ibm-cloud-env');

// Initialize cloud environment for credentials
IBMCloudEnv.init();

// Global variables
var Credentials_SAP = {};
var Credentials_Watson = {};
var WatsonScoringEndpoint = '';
var Token = '';
var Fields = {};

// Load configuration, get access token and model metadata
init();

async function init() {
  // Load configuration file
  Credentials_SAP = yamljs.load('sap-credentials.yml');

  // Get Watson service credentials
  Credentials_Watson = IBMCloudEnv
    .getDictionary('watson_machine_learning_credentials');
```



```

// Get IAM access token
// CAUTION: The token is aquired only once. When expires, the server program
must
//          be restarted.
var token = await rest_request(
  { method: 'POST',
    hostname: 'iam.cloud.ibm.com',
    path: '/identity/token',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/x-www-form-urlencoded' },
    body: 'grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey='
      + Credentials_Watson.apikey
  } ).catch(function(err) {logger.error(err); process.exit(8)});

if (token['access_token'] == null) {
  logger.error('IAM: Cannot retrieve access token');
  process.exit(8);
}
Token = token.access_token;

// Get all deployments of the Watson Machine Learning servie
var deployments = await rest_request(
  { method: 'GET',
    hostname: Credentials_Watson.url.replace(/^(\https?:\/\//), ''),
    path: '/v4/deployments',
    //url: Credentials_Watson.url + '/v4/deployments',
    headers: {
      'Accept': 'application/json',
      'Authorization': 'Bearer ' + Token,
      'ML-Instance-ID': Credentials_Watson.instance_id }
  } ).catch(function(err) {logger.error(err); process.exit(8)});

// Sort deployments by date descending (i.e. most recent first, non-ready last)
deployments.resources.sort((deployment1, deployment2) =>
  { var date1 = (deployment1.entity.status.state === 'ready') ? (
    deployment1.metadata.modified_at !== '' ?
    deployment1.metadata.modified_at :
    deployment1.metadata.created_at ) :
    '1000-01-01T00:00:00.000Z';
    var date2 = (deployment2.entity.status.state === 'ready') ? (
    deployment2.metadata.modified_at !== '' ?
    deployment2.metadata.modified_at :
    deployment2.metadata.created_at ) :
    '1000-01-01T00:00:00.000Z';
    return (date1 < date2) ? 1 : -1 });
WatsonScoringEndpoint = deployments.resources[0].entity.status.online_url.url
  .replace(Credentials_Watson.url, '');

logger.info(`Using Model ` +
  `${deployments.resources[0].entity.asset.href.replace(/(\?.*)$/, '')}`);
logger.info(`Using WML Scoring Endpoint ${WatsonScoringEndpoint}`);

// Get metadata of Watson Machine Learning model

```

```

var model = await rest_request(
  { method: 'GET',
    hostname: Credentials_Watson.url.replace(/^(\https?:\/\//), ''),
    path: deployments.resources[0].entity.asset.href,
    headers: {
      'Accept': 'application/json',
      'Authorization': 'Bearer ' + Token,
      'ML-Instance-ID': Credentials_Watson.instance_id }
  } ).catch(function(err) {logger.error(err); process.exit(8)});

// Watson Machine Learning service returns model metadata like
// {... "entity": {...
//       "schemas": {
//         "input": [
//           { "fields": [
//             { "name": "age", "type": "int64" },
//             { "name": "job", "type": "object" },
//             ... ] } ] } } }
Fields = model.entity.schemas.input[0].fields;

logger.info(`Metadata loaded, access token expires in ${token.expires_in}
secs.`);
}

/* This function performs a REST call. It is being used for calling IBM Watson
endpoints and the SAP OData service.
All information needed for the REST call is provided in the options parameter.
*/
function rest_request(options) {
  return new Promise(function(resolve, reject) {

    var result = "";
    var request = https.request(options, function(response) {
      response.setEncoding('utf8');

      response.on('data', function (data) {
        // Called multiple times -> combine chunks to a single result
        result += data;
      });

      response.on('end', function() {
        // End of communication
        if ((response.statusCode !== 200) && (response.statusCode !== 404)) {
          logger.error(`Request to ${options.hostname}${options.path} ` +
            `failed with status ${response.statusCode}`);
          // Result contains error message (html or json)
          reject(result);
        } else {
          // Return response to caller
          resolve(JSON.parse(result));
        }
      });
    });
  });
}

```

```

    request.on('error', function(err) {
        reject(err);
    });

    // Send body (if available)
    if ((typeof(options.body) === 'string') && (options['body'] !== '')) {
        request.write(options.body);
    }

    // Finish request
    request.end();
});
}

/* This function implements the core functionality of the application:
1. Call OData service to retrieve customer data according to filter
   criteria that are passed as parameter
2. Build array of feature vectors from customer data
3. Post array of feature vectors to Watson Machine Learning endpoint;
   get predictions in response
4. Send merged customer data and predictions to callback function
*/
async function predict(filter, callback) {

    // Build OData filter parameter
    // Field names in OData service are assumed to be upper-case.
    // For example: filter = {age: 45, job: 'management'} is converted to
    // url-encode(AGE eq 45 and JOB eq 'manager')
    var filterstring = '';
    for (var filter_field in filter) {
        field = Fields.find(_field => _field.name == filter_field);
        if (typeof(field) !== 'undefined') {
            filterstring += (filterstring !== '' ? '%20and%20' : '') +
                filter_field.toUpperCase() + '%20eq%20' +
                (!field.type.startsWith('int') ?
                    '%27' + filter[filter_field] + '%27' :
                    filter[filter_field]);
        }
    }

    // Call OData service to retrieve data from the SAP system
    var odata_result = await rest_request(
        { method: 'GET',
          hostname: Credentials_SAP.url,
          port: Credentials_SAP.port,
          path: Credentials_SAP.path + '?$format=json&$filter=' + filterstring,
          headers: {
              'Accept': 'application/json',
              'Authorization': 'Basic ' +
                  new Buffer.from(Credentials_SAP.user + ':' +
                      Credentials_SAP.password).toString('base64') },
          rejectUnauthorized: false,

```

```

        agent: false
    } ).catch(function(err) {logger.error(err)});

    if (odata_result.d.results == 0) {
        // No matching record found in the SAP system -> Return empty result set
        return callback({fields: [], values: []});
    }

    /* Build WML request
    For each customer data record that has been received from SAP an array is
    build
    that has an entry for every field of the model.
    Values from the OData result are mapped to model fields by name, where
    upper-case and leading underscores are ignored.
    Default values are used for model fields that do not have an corresponding
    value in the OData result.
    Example:
    Fields: ["age","marital","balance","default"]
    OData: {"d":{
        "results":[
            {"recordno":31, "age":45, "marital":"single",
"default_":"no"},
            {"recordno":67, "age":56, "marital":"married"}}]}
    -> values = [[45, "single", 0, "no"],[56,"married",0,""]]
    */
    fields = [];
    for (field of Fields) fields.push(field.name);
    values = [];
    for (var entity of odata_result.d.results) {
        entity_values = [];
        for (var field of Fields) {
            var _value = null;
            for (var _fieldname of [field.name, field.name.toUpperCase(),
                field.name+'_', field.name.toUpperCase()+'_']) {
                if (entity.hasOwnProperty(_fieldname)) {
                    _value = entity[_fieldname];
                    break;
                }
            }
            if (_value != null) {
                entity_values.push(_value);
            } else {
                entity_values.push(field.type.startsWith('int') ? 0 : '');
            }
        }
        values.push(entity_values);
    }

    // Call IBM Watson Machine Learning endpoint
    var body = JSON.stringify({'input_data': [{'fields': fields, 'values':
values}]});
    var wml_result = await rest_request(
        { method: 'POST',
          hostname: Credentials_Watson.url.replace(/^(https?:\/\/)/, ''),
          path: WatsonScoringEndpoint,

```

```

        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
            'Authorization': 'Bearer ' + Token,
            'ML-Instance-ID': Credentials_Watson.instance_id },
        body: JSON.stringify({'input_data': [{'fields': fields, 'values':
values}]})
    } ).catch(function(err) {logger.error(err)});

    var prediction = wml_result.predictions[0];

    // Merge customer data and predictions
    // Watson Machine Learning returns prediction and confidence of the two classes
    // "no" and "yes" like ["no", [0.68, 0.32]]
    // Confidence of class "yes" is used
    for (field of prediction.fields) fields.push(field)
    for (var i=0; i<Math.min(values.length, prediction.values.length); i++) {
        for (var j=0; j<prediction.fields.length; j++) {
            if (typeof(prediction.values[i][j]) === 'object') { // type is array
values[i].push(prediction.values[i][j][prediction.values[i][j].length-1]);
            } else {
                values[i].push(prediction.values[i][j]);
            }
        }
    }

    // Sort values descending by probability (last value in arrays)
    values.sort((entity1, entity2) =>
        - ( entity1[entity1.length - 1] - entity2[entity2.length - 1]) );

    // Send merged data to callback function
    callback({'fields': fields, 'values': values});
}

//Exploit express.js framework to server POST requests against api endpoint
/api/predict
//Use body-parser module to convert request body to javascript objects
app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());

app.post('/api/predict', function(request, response) {
    predict(request.body, (result) => {const resp = response; resp.json(result); }
    );
});

/*****      END: Custom Implementation
*****/

const port = process.env.PORT || localConfig.port;
server.listen(port, function(){
    logger.info(`nodejswebapp listening on http://localhost:${port}`);

```

```

});

app.use(function (req, res, next) {
  res.sendFile(path.join(__dirname, '../public', '404.html'));
});

app.use(function (err, req, res, next) {
  res.sendFile(path.join(__dirname, '../public', '500.html'));
});

module.exports = server;

Frontend (Web Browser) javascript code

File public/js/frontend.js

/* This function is invokes the node.js application via POST request.
   It renders the response into a table that is added to the DOM.      */
function predict() {

  // build filter from input elements
  var filter = {};
  for (var elem of document.querySelectorAll('input')) {
    if (elem.value !== '') filter[elem.name] = elem.value;
  }

  // clear output section
  div_out = document.getElementById('div_out');
  div_out.innerHTML = 'Processing ...';

  // call server to get customer list with preditions
  call_server(JSON.stringify(filter)).then(
    function(response) {
      const div = div_out;

      if (response.values.length == 0) {
        // empty result set
        div_out.innerHTML = 'No matching record found.';
      } else {
        div_out.innerHTML = '';

        // create output table
        var table = document.createElement('table');
        table.setAttribute('class', 'out');

        // create table header
        var thead = table.createTHead();
        thead.setAttribute('class', 'out');
        var row = thead.insertRow();
        row.setAttribute('class', 'out');
        var cell = row.insertCell();
        cell.setAttribute('class', 'out');
        cell.innerText = 'action';
        for(var property of response.fields) {

```

```

        cell = row.insertCell();
        cell.setAttribute('class', 'out');
        cell.innerHTML = property;
    }

    // create table body
    var tbody = table.createTBody();
    for (var values of response.values) {
        var row = tbody.insertRow();
        row.setAttribute('class', 'out');
        var cell = row.insertCell();
        cell.setAttribute('class', 'out');
        // action: just display alert
        cell.innerHTML =
            '<a href="javascript:alert(\'Calling customer ...\')">'
            + 'call customer</a>';
        for (var value of values) {
            cell = row.insertCell();
            cell.setAttribute('class', 'out');
            cell.innerHTML = value;
        }
    }
}

div.appendChild(table);
});
}

```

```

// This function executes a POST request against application endpoint
"/api/predict".
function call_server(body) {
    return new Promise(function(resolve, reject) {

        request = new XMLHttpRequest();
        request.open('POST', '/api/predict');
        request.setRequestHeader('Accept', 'application/json');
        request.setRequestHeader('Content-type', 'application/json');
        request.onreadystatechange = function() {
            if (this.readyState == 4) {
                if (this.status == 200) {
                    resolve(JSON.parse(this.response));
                }
            }
        };
        request.onerror = function() {
            var status = this.status;
            reject(status);
        };

        request.send(body);
    });
}

```

Index HTML page

File public/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>IBM Demo</title>
  <style>
    table.out { font-family: arial, sans-serif;
                border-collapse: collapse;
                width: 100%; }
    thead.out { background-color: #555555;
                color: #eeeeee; }
    td.out, th.out { border: 1px solid #555555;
                     text-align: center;
                     padding: 8px; }
    tr.out:nth-child(even) { background-color: #dddddd; }
    .section { border: 1px solid #000000;
               margin-bottom: 10px; }
    .block { width: 100%; }
  </style>
  <script src="js/frontend.js"></script>
</head>
<body>
<header>
  <h1>Marketing Campaign Advisor</h1>
</header>
<main>

  <div id="div_filter" class="section" padding="4px">
    <table cellpadding="8px">
      <tr><th colspan="2" align="left">Filter Criteria</th></tr>
      <tr><td>Age</td><td><input id="input_age" type="number"
                                name="age" value="40"
                                min="18" max="99" step="1"></td></tr>
      <tr><td>Job</td><td><input id="input_job"
                                name="job" value="technician"></td></tr>
      <tr><td>Marital</td><td><input id="input_marital"
                                name="marital"
                                value="married"></td></tr>
      <tr><td>Education</td><td><input id="input_education"
                                name="education"></td></tr>
      <tr><th colspan="2" align="left"><button id="button_predict"
                                type="button"
                                class="block"
                                onclick="predict();">
                                Predict</button></th></tr>
    </table>
  </div>
  <div id="div_out">
    <!-- result table comes here -->
  </div>
</main>
</body>
```


</html>

Abbreviations and acronyms

Convention	All ABAP Objects that are create for demos and examples must be assigned to ABAP Package ZDEMO.	SDK	Software Development Kit
Convention	All ABAP Object names must have prefix ZDEMO_ or ZCL_DEMO_, respectively.	SQL	Structured Query Language
ABAP	Advanced Business Application Programming	SSF	Secure Store & Forward
ADT	ABAP Development Tools (in Eclipse)	SSL	Secure Sockets Layer
AFL	Application Function Library	TF	TensorFlow
AI	Artificial Intelligence	TLS	Transport Layer Security
API	Application Programming Interface	TMS	TensorFlow Model Server
API	Application Program Interface	URI	Uniform Resource Identifier
DB	Database	URL	Uniform Resource Locator
DM	HANA Express Edition Download Manager	YOLO	You Only Look Only (CNN Neural Network)
EML	External Machine Learning		
ERP	Enterprise Resource Planning		
FQDN	Fully Qualified Domain Name		
gRPC	Google Remote Procedure Call		
GUI	Graphical User Interface		
HANA	High-Performance Analytic Appliance		
HS	SAP HANA Studio		
HTTP	Hypertext Transfer Protocol		
HTTPS	Hypertext Transfer Protocol Secure		
IAM	Identity and Access Management		
IBM	International Business Machines Corporation		
ICM	Internet Communication Manager		
IP	Internet Protocol		
JSON	JavaScript Object Notation		
ML	Machine Learning		
PDF	Portable Document Format		
PSE	Personal security environment		
REST	Representational State Transfer		
RTTI	Runtime Type Information		
SAP	Systems, Applications and Products		
SAP System	SAP Business Suite or S/4HANA		
SCP	SAP Cloud Platform		

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Some publications that are referenced in this list might be available in softcopy only:

- ▶ *SAP HANA on IBM Power Systems: High Availability and Disaster Recovery Implementation Updates*, SG24-8432
- ▶ *SAP HANA Data Management and Performance on IBM Power Systems*, REDP-5570
- ▶ *SAP HANA on IBM Power Systems Architectural Summary*, REDP-5569
- ▶ *SAP HANA Platform Migration*, REDP-5571

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft, and additional materials, at the following website:

ibm.com/redbooks

Other publications

The following publications are also relevant as further information sources:

- ▶ *IBM Power Systems - SAP Software Deployed in Red Hat OpenShift*, REDP-5619
- ▶ *Software Defined Data Center with Red Hat Cloud and Open Source IT Operations Management*, SG24-8473
- ▶ *Red Hat OpenShift V4.X and IBM Cloud Paks on IBM Power Systems Volume 2*, SG24-8486
- ▶ *Red Hat OpenShift V4.3 on IBM Power Systems Reference Guide*, REDP-5599

Online resources

The following websites also are relevant as further information sources:

- ▶ IBM Public Cloud:
<http://www.cloud.ibm.com>
- ▶ TensorFlow Core documentation:
https://www.tensorflow.org/api_docs/python/tf/io/decode_base64
- ▶ IBM Watson Studio:
<https://www.ibm.com/cloud/watson-studio>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



REDP-5577-00

ISBN 0738459712

Printed in U.S.A.

Get connected

