





Reading Sample

This sample chapter discusses SAP BTP, ABAP environment and its evolution. It explores continuous integration/continuous delivery (CI/CD) options for the ABAP environment and takes a close look at how to build CI/CD pipelines. Additionally, it includes coverage of how to develop an application using SAP Business Application Studio using the ABAP language and explores various development and test tools. Finally, the chapter closes with a discussion of the roles SAP Cloud ALM and SAP Cloud Transport Management play in DevOps implementation for ABAP-based solutions.

-  **“DevOps for SAP BTP, ABAP Environment”**
-  **Contents**
-  **Index**
-  **The Authors**

Raja Gupta, Sandip Jha

DevOps with SAP

387 pages | 07/2023 | \$89.95 | ISBN 978-1-4932-2419-7

 www.sap-press.com/5694

Chapter 5

DevOps for SAP BTP, ABAP Environment

In this chapter, we'll explore how to apply DevOps principles to SAP BTP, ABAP environment. We'll discuss the evolution of ABAP cloud development, the challenges involved in implementing DevOps in the ABAP environment, and various concepts related to development and DevOps in the ABAP environment. By the end of this chapter, you'll have a solid understanding of how to implement DevOps in SAP BTP, ABAP environment and how to leverage SAP BTP services to streamline your development processes.

So far, we've discussed DevOps principles, relevant tools, and implementation in on-premise and cloud platforms (i.e., SAP BTP). In Chapter 4, you learned about the DevOps portfolio in SAP BTP and how various DevOps phases are implemented for a cloud application in SAP BTP. While discussing the DevOps services, tools, and frameworks, we mainly focused on SAP BTP, Cloud Foundry environment. While Cloud Foundry is one of the most commonly used environments in SAP BTP, there is another useful environment for the ABAP world: SAP BTP, ABAP environment. Although the basic concepts and capabilities of DevOps services, tools, and frameworks remain environment-agnostic, the DevOps implementation for each environment is unique. This chapter is targeted for readers who want to leverage SAP BTP, ABAP environment and learn how to successfully implement DevOps in this environment. However, we strongly recommend reading Chapter 4 thoroughly before you start this chapter.

In this chapter, you'll first learn what SAP BTP, ABAP environment is and about its evolution. The ABAP language has seen remarkable changes and has evolved a lot from the on-premise to the cloud world. You'll learn about important concepts of the ABAP environment and how it fits into an organization's needs. We'll also explore CI/CD options for the ABAP environment and take a close look at how to build CI/CD pipelines. You'll learn how to develop an application using SAP Business Application Studio in the ABAP language and explore various development and test tools. Finally, we'll dive deep into the roles of SAP Cloud ALM and SAP Cloud Transport Management in DevOps implementation for ABAP-based solutions.

5.1 Introduction to SAP BTP, ABAP Environment

One of the main drivers for the release of the ABAP environment on SAP BTP was the need for a modern and flexible development platform that would allow SAP partners and customers to build and deploy ABAP applications in the cloud. Although SAP BTP allows you to build cloud applications using any new language, like Node.js, Java, Python, and the like, there is a huge community coming from the SAP on-premise era that is deeply in love with ABAP. There are great numbers of SAP partners and users who want to either build new applications using ABAP or reuse their code from SAP on-premise systems to move their SAP ERP extensions to the cloud. To fulfill the needs of these partners and customers, SAP has launched an ABAP environment on SAP BTP.

As a part of this introduction, let's explore SAP BTP, ABAP environment, its evolution, and its benefits in SAP BTP in the next sections.

5.1.1 What Is SAP BTP, ABAP Environment?

SAP BTP, ABAP environment provides developers with a complete development environment in SAP BTP to build and run ABAP-based applications. It includes tools for developing, testing, and deploying applications, as well as a runtime environment for running ABAP applications in the cloud. The ABAP environment is fully integrated with other services in SAP BTP such as SAP HANA Cloud, SAP Build Work Zone, and so on, allowing developers to build end-to-end business applications.

For many years, ABAP has served as the basis for SAP's on-premise solutions. However, with ABAP now available in SAP BTP, developers who are familiar with ABAP can use their existing knowledge to create and operate ABAP applications in the cloud. It's important to note that the ABAP language used in SAP BTP, ABAP environment is slightly different from traditional ABAP, mainly because we now use an evolved and advanced, optimized version of the ABAP language. In Section 5.3.1, you'll learn more about it. With SAP BTP, ABAP environment, you can not only build ABAP-based solutions but also seamlessly use all the features and capabilities of SAP BTP as well as call any service offered by SAP BTP.

Technically, SAP BTP, ABAP environment is a platform-as-a-service (PaaS) offering available on SAP BTP. Figure 5.1 shows a high-level overview of SAP BTP, ABAP environment.

To access SAP BTP, ABAP environment, you need to create an instance of this service using the SAP BTP cockpit. As discussed in Chapter 4, the SAP BTP cockpit is a web-based tool that provides a user interface for managing and configuring SAP BTP services and resources, including the ABAP environment. It enables ABAP developers to create, deploy, and manage ABAP applications in the cloud without requiring extensive knowledge of infrastructure and networking.

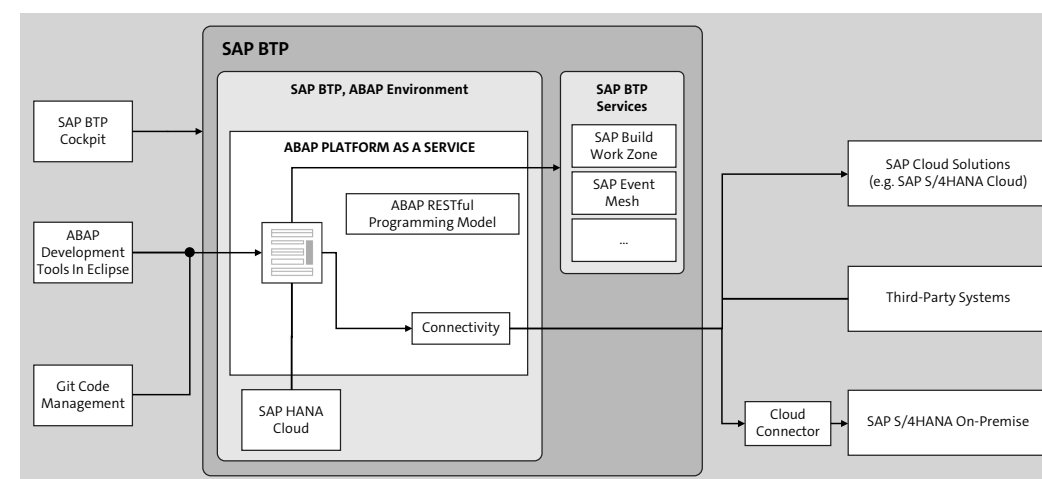


Figure 5.1 SAP BTP, ABAP Environment

ABAP development tools play an essential role in SAP BTP, ABAP environment. ABAP development tools is a set of Eclipse-based tools that allows developers to create and maintain ABAP-based applications in the cloud. Some of the key roles of ABAP development tools in the SAP BTP, ABAP environment include the following:

- **Code development**
ABAP development tools provides a code editor with syntax highlighting, code completion, and error checking to help developers write ABAP code efficiently. It also includes templates and wizards for creating new ABAP objects, such as programs, function modules, and classes.
- **Debugging**
ABAP development tools includes a powerful debugger that allows developers to debug their ABAP code in the cloud. The debugger includes features such as breakpoints, watchpoints, and variable inspection, enabling developers to quickly identify and resolve issues in their code.
- **Performance analysis**
ABAP development tools includes tools for analyzing the performance of ABAP programs, such as the SQL Monitor. These tools help developers optimize their code and improve the performance of their applications.

The ABAP environment uses Git for code exchange and code deployment. It can also use abapGit, an open-source Git client for ABAP that allows ABAP code to be version-controlled, shared, and transported between systems. In Chapter 3, we discussed abapGit in detail.

The ABAP environment is based on the latest ABAP platform cloud release that is also used for SAP S/4HANA Cloud. The ABAP RESTful application programming model,

which includes SAP Fiori and core data services (CDS), is supported by the ABAP environment.

Solutions built on the ABAP environment can easily be integrated with other SAP BTP services, such as the SAP Destination service, SAP Build Work Zone, SAP Event Mesh, and more. With the help of the SAP Connectivity service, any solution running on the ABAP environment can be integrated with any SAP or non-SAP solutions. To integrate with SAP on-premise systems, the cloud connector must establish a secure tunnel from SAP BTP to the on-premise system.

SAP BTP, ABAP environment is designed to be highly secure and compliant with industry regulations, providing a range of features and tools to ensure the security of applications and data. It also provides a range of analytics and reporting tools to help developers understand application usage and performance and to identify areas for improvement.

5.1.2 Evolution of ABAP

ABAP is a programming language that has been used for decades to develop business applications on SAP systems. Over the years, ABAP has undergone significant changes to keep up with changing business requirements and technological advancements. The evolution of ABAP has led to the introduction of new programming models, tools, and frameworks that have improved the productivity and quality of ABAP developers. In this section, we'll explore the evolution of ABAP, from its early days as a procedural language to the latest ABAP RESTful application programming model and discuss how these changes have transformed the way developers build business applications.

Figure 5.2 shows the three major phases of ABAP evolution, which are as follows:

■ Classic ABAP application programming

Classic ABAP application programming refers to the traditional programming model in ABAP, which is based on procedural programming. In this model, ABAP programs are structured as a collection of modules, where each module performs a specific task. These modules are known as *function modules* and are the basic building blocks of ABAP programs. The function modules can be called from other programs or function modules, enabling developers to build complex applications by combining and reusing existing modules.

In the classic ABAP programming model, programs are executed in a linear fashion, with each module being executed in sequence. The programs use the ABAP dictionary to define data structures and tables and the ABAP runtime environment to execute the programs. The ABAP debugger is used for debugging ABAP programs.

While the classic ABAP programming model is still supported in the ABAP language today, it has some limitations. One of the main limitations is that it is difficult to maintain and modify programs that have a large number of function modules.

Another limitation is that the programming model is not very flexible and does not support modern programming paradigms, such as object-oriented programming.

■ ABAP programming model for SAP Fiori

ABAP programming model refers to the architecture for efficient development of SAP Fiori applications optimized for SAP HANA in SAP S/4HANA. This model supports the development of various types of SAP Fiori applications, including transactional, search, analytics, and planning apps, and is built on proven customer technologies and frameworks such as CDS for defining rich data models, SAP Gateway, Business Object Programming Framework (BOPF), the OData protocol, ABAP-based application services for custom logic, and SAPUI5-based user interfaces.

■ ABAP RESTful application programming model

The *ABAP RESTful application programming model* is the successor to the ABAP programming model for SAP Fiori and is an architecture that enables efficient end-to-end development of OData services that are optimized for SAP HANA, including SAP Fiori apps. It supports the creation of various types of SAP Fiori applications and enables the publishing of web APIs. It utilizes CDS for defining data models with rich semantics, a service model infrastructure for creating OData services with bindings to an OData protocol, and ABAP-based application services for implementing custom logic. Furthermore, it allows the creation of SAPUI5-based user interfaces. This model is illustrated in detail in Section 5.3.1.

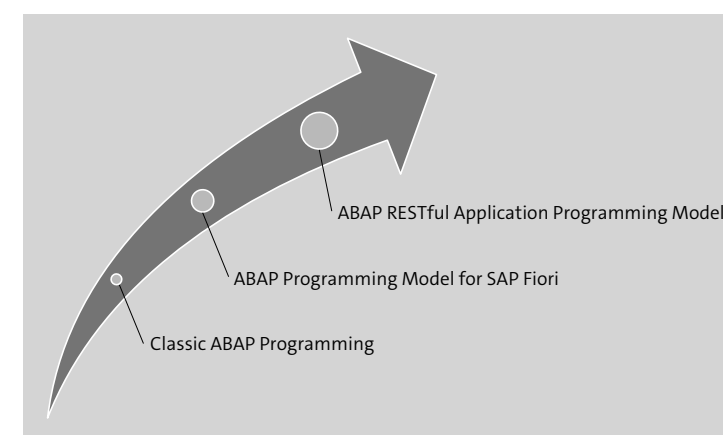


Figure 5.2 ABAP Evolution

5.1.3 Need for ABAP Environment in SAP BTP

Many SAP partners and companies running SAP have adopted SAP solutions mostly with ABAP technology and enhanced these as custom solutions as per their needs. These business applications were enterprise-grade. Many of these have also adopted SAP S/4HANA Cloud for transitioning to the cloud. These trends necessitated support for the ABAP environment in SAP BTP.

SAP has reported that more than 75% of business transactions globally involve an SAP system, most of which are likely operating on ABAP and on-premise (<http://s-prs.co/v569405>). Although the duration of the transition period may be disputed, it's highly probable that future software for business transactions will be cloud based.

SAP could have stuck with outdated concepts, but in the long run it would have become outdated. SAP innovated to align with a cloud approach. Therefore, SAP BTP, ABAP environment was developed, which is also known as project Steampunk. It has a distinct purpose: to offer an ABAP platform that is not only a benchmark for enterprise-readiness, as it currently is, but is also primed for the cloud.

5.1.4 Benefits of ABAP Environment in SAP BTP

For years, ABAP development was restricted to the on-premise stack. In SAP BTP, Cloud Foundry environment, you can create a new space for ABAP development. This space is called the ABAP environment and is where you can host ABAP developments. This enables development of ABAP applications and extensions in the cloud, commonly known as the *cloud development model*.

Salient features of the ABAP cloud development model are as follows:

- ABAP RESTful application programming model
- SAP Fiori
- CDS
- Cloud-optimized ABAP language
- Public SAP APIs and extension points

Transitioning to the cloud using existing ABAP assets helps promote reuse of existing efforts. It leverages SAP BTP by delegating infrastructure and system operations to SAP. Another benefit is decoupled cloud extensions, which helps reduced costs and promotes a clean core strategy.

5.2 Continuous Integration and Continuous Delivery Tools in SAP BTP, ABAP Environment

In the ABAP landscape deployed on-premise, the ABAP systems have the capability to regulate the lifecycle and quality processes themselves, such as administering transport routes and implementing various checks through jobs. However, this capability is no longer feasible in SAP BTP, ABAP environment, where the systems function independently and the source code is transported through Git-based solutions like gCTS and abapGit. Thus, the responsibility of driving the lifecycle and quality processes must shift from the ABAP system to an external agent, such as a traditional CI/CD server like Jenkins.

The previous section explained evolution and importance of SAP BTP, ABAP environment. Next we'll discuss the planning phase of DevOps. In the planning phase, you plan and set up the environment for CI/CD. This phase consists of setup guidance, account setup, and CI/CD enablement. These topics are well explained in Chapter 4. Regarding CI/CD enablement, Chapter 4, Section 4.2 explains the following SAP offerings:

■ SAP Continuous Integration and Delivery

This SAP BTP service, mainly suited for SAP-centric use cases, doesn't require a lot of skill to build a pipeline. There is no need to operate your own infrastructure for CI/CD.

■ Project "Piper"

This is an open-source project and associated libraries. It provides templates for pipelines, best suited for when you want to use a predefined template and have some flexibility. It provides predefined pipelines consisting of stages and steps that can be configured as per requirements. These standard pipeline templates are available in a GitHub repository and can be downloaded and adapted with various standard step libraries. We'll discuss this topic in detail with respect to the ABAP environment in this section.

■ Continuous Integration and Delivery Best Practices Guide

The best practices guide is helpful if you want to completely own your CI/CD pipeline and have full flexibility. This approach provides CI/CD guidelines for any infrastructure planned by a user.

Each option has its pros and cons, which we discussed in detail in Chapter 4.

Next, we'll discuss building blocks of CI/CD with reference to project "Piper", starting with Git repositories for code management, followed by APIs, the ABAP environment pipeline provided by SAP, and using Jenkins for the setup of a sample CI/CD scenario.

5.2.1 Git Code Management

In cloud infrastructure, systems like development, quality, and production systems are isolated, unlike in on-premise systems. There are benefits of this distributed architecture, but it requires additional infrastructure to connect and maintain these systems.

Git is an open-source version management tool that has been widely used for more than a decade. This tool was explained in detail in Chapter 2, Section 2.1.1.

A Git repository can be used for code management in SAP BTP, ABAP environment. This arrangement of connecting systems using Git is referred to as *Git code management*. You can connect systems like development, sandbox, test, and production to this repository. In a development system, developers create applications and extensions using tools like ABAP development tools in Eclipse. Developers can push code from the development system to a Git repository. Once this code is available in the Git repository, you can transfer the code to a quality system using a pull mechanism. On transfer

of the code to the quality system, quality colleagues can run tests and implement testing scenarios. This process is illustrated in Figure 5.3.

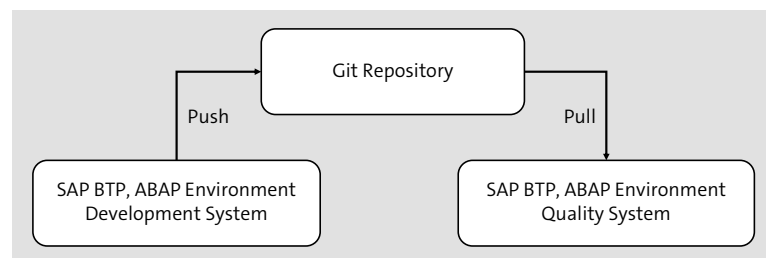


Figure 5.3 Git Code Management

These actions can be automated as part of CI/CD process. For this, SAP offers APIs that we'll discuss in the next section.

5.2.2 APIs

For automating processes in SAP BTP, ABAP environment, you need services that can control the ABAP environment remotely. These services are referred to as APIs. Several APIs are available for automating processes related to quality tasks. Some examples are as follows:

- **Cloud Foundry command line interface**

This API can be used for managing ABAP environment systems, like creating or deleting one on the fly.

- **Manage Git repository service**

This pulls development artifacts from a Git repository to a quality system.

- **ATC services**

With this API, you can execute ABAP test cockpit checks on the ABAP environment system.

SAP provides standard APIs called *steps*, which are detailed at www.project-piper.io, in the **Library steps** section. All steps with the prefix *abap* are generally applicable to SAP BTP, ABAP environment.

Now that we've discussed APIs for performing individual steps, let's move on to discussing the pipeline in next section.

5.2.3 ABAP Environment Pipeline

In Chapter 4, Section 4.2.3, you learned about project "Piper" and its offerings in terms of the standard SAP-delivered pipelines, like the general-purpose pipeline and the ABAP environment pipeline. The ABAP environment pipeline is applicable to SAP BTP, ABAP environment.

In short, the *ABAP environment pipeline* is a set of automated processes that enable the continuous delivery and deployment of ABAP code changes across different environments, such as development, testing, staging, and production. It involves the use of a version control system such as Git, which is used to manage source code changes. Developers can make changes to ABAP code and push them to the Git repository, from which automated build, test, and deployment process can be performed.

Building Blocks

Using APIs, you can create CI/CD pipelines. But this can be cumbersome, so SAP provides a predefined pipeline for the ABAP environment called the ABAP environment pipeline. Since it's standard, this pipeline covers most scenarios, as shown in Figure 5.4. The pipeline implementation is based on Jenkins.

The ABAP environment pipeline currently supports the following two scenarios:

- **Continuous testing**

This scenario helps automate ABAP tests like ABAP test cockpit and ABAP Unit tests in SAP BTP, ABAP environment. We'll use this scenario to showcase how to set up this pipeline at the end of this section.

- **Building ABAP add-ons for SAP BTP, ABAP environment**

This scenario helps when creating software-as-a-service (SaaS) solutions. Specific stages, like publish, are supported for this scenario.

The pipeline generally includes the following stage categories:

- **Code compilation and quality checks**

The ABAP code is compiled and checked for quality using tools such as ABAP test cockpit.

- **Automated testing**

The code changes are automatically tested using tools such as ABAP Unit.

- **Build and packaging**

The code changes are built into an executable package that can be deployed to various environments.

- **Deployment**

The packaged code changes are automatically deployed to different environments, such as development, testing, staging, and production, using tools such as Jenkins.

- **Monitoring**

The pipeline includes monitoring tools that track the status of the deployment and alert developers in case of any issues.

As previously mentioned, this pipeline consists of different parts, called *stages*. Stages represented in green with a checkmark are considered active. Those with empty circles are considered inactive and will be skipped during execution. In Figure 5.4, initial checks, build, and integration tests are inactive.

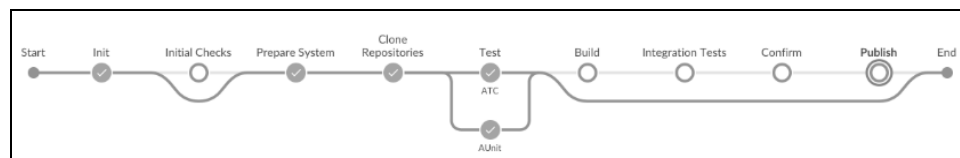


Figure 5.4 ABAP Environment Pipeline

For example, the ABAP test cockpit check is an individual step. Similarly, we can create other stages, like build stage, integration stage, or release stage, which consist of their respective individual steps. Combining these stages results in creating a pipeline, as illustrated in Figure 5.5.

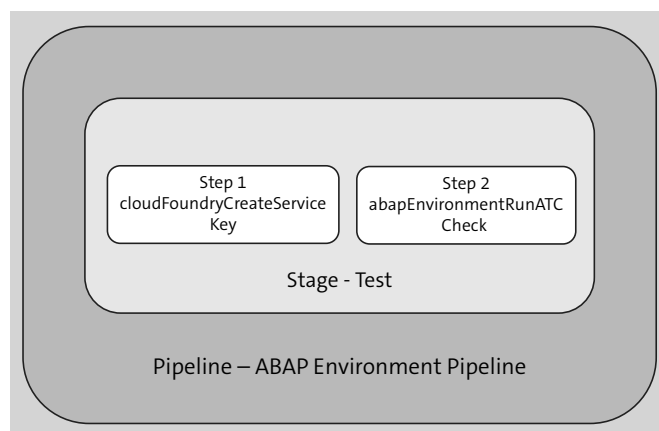


Figure 5.5 Pipeline Building Block

In the figure, `abapEnvironmentCreateSystem` and `cloudFoundryDeleteService` are individual steps (or APIs). The collection of these steps is called the test stage. Combining this stage with others would result in creating a pipeline. Here, the test stage is part of the ABAP environment pipeline.

The following stages are part of the ABAP environment pipeline:

- **Init**
This shows the start of pipeline.
- **Initial checks**
Preliminary checks required for the build stage are executed at this stage.
- **Prepare system**
The SAP BTP, ABAP environment system is created at this stage.
- **Clone repositories**
Specified software components known as repositories are pulled to SAP BTP, ABAP environment in this stage. Communication arrangement setup is required as a pre-step.

- **Test**
This stage is disabled by default; configuration is required to run tests like ABAP test cockpit and ABAP Unit tests. These tests can be executed in parallel.
- **Build**
ABAP add-on developments can be triggered for build at this step using SAP Add-On Assembly Kit as a service.
- **Integration tests**
Add-on products in the build stage are installed into a new ABAP environment system at this stage.
- **Confirm**
Manual confirmation can be configured at this stage.
- **Publish**
Add-ons are published at this stage.
- **Post**
The SAP BTP, ABAP environment system created in the prepare system stage is deleted at this stage as part of the teardown phase.

Although the ABAP pipeline is predefined, options for extending and configuring the pipeline are supported. This can be done at the stage level as part of pipeline configuration. The next section will discuss pipeline configuration.

Configuration

You can configure the standard ABAP environment pipeline. This is done in the `config.yml` file, located at `.pipeline/config.yml` in the master branch of the source code repository.

The default configuration file can be viewed at https://github.com/SAP/jenkins-library/blob/master/resources/default_pipeline_environment.yml.

Configuration files created in individual repository projects inherits from this file.

The configuration file contains multiple scopes, like general, stages, and steps. You can provide values for these scopes. Values in general stages are applicable to the entire pipeline, but values in other scopes are restricted to that level. For example, if you provide configuration values for the prepare system stage, then these configurations are valid for this stage only. A sample configuration file is shown in Listing 5.1.

```
general:
  gitSshKeyCredentialsId: GitHub_DWS

steps:
  cloudFoundryDeploy:
    deployTool: 'cf_native'
    cloudFoundry:
```

```

org: 'dwsOrg'
space: 'dwsSpace'
credentialsId: 'CF_CREDENTIALS_ID_IN_JENKINS'
newmanExecute:
  newmanCollection: 'myNewmanCollection.file'
  newmanEnvironment: 'myNewmanEnvironment'
  newmanGlobals: 'myNewmanGlobals'

```

Listing 5.1 Configurations Usage

Configurations can be accessed from custom scripts using the `setupCommonPipelineEnvironment` step. For example:

```
commonPipelineEnvironment.configuration.general.gitSshKeyCredentialsId
```

If you want to access configurations in library steps, you can use the `ConfigurationHelper` object. An example is shown in Listing 5.2.

```

Map config = ConfigurationHelper.newInstance(this)
    .loadStepDefaults([], stageName)
    .mixin(ConfigurationLoader.defaultStageConfiguration(script, stageName))
    .mixinGeneralConfig(script.commonPipelineEnvironment, GENERAL_CONFIG_
KEYS)
    .mixinStepConfig(script.commonPipelineEnvironment, STEP_CONFIG_KEYS)
    .mixinStageConfig(script.commonPipelineEnvironment, stageName, STEP_
CONFIG_KEYS)
    .mixin(parameters, PARAMETER_KEYS)
    .use()

```

Listing 5.2 ConfigurationHelper Object

Deactivating a Stage

If you don't provide configuration for a stage, you can set that stage as inactive.

Extensions

Another option to customize the pipeline is with extensions. You can extend one or multiple stages. A stage can be extended by creating a Groovy file with the stage name. If you want to extend the ABAP test cockpit stage, you should create an **Extensions** folder and place a file named `ATC.groovy` in it. You can copy the standard code (as shown in Listing 5.3) for ABAP test cockpit checks into that file and then modify the code per your requirements.

```

void call(Map params) {
  //access stage name
  echo "Start - Extension for stage: ${params.stageName}"

  //access config
  echo "Current stage config: ${params.config}"

  //execute original stage as defined in the template
  params.originalStage()

  recordIssues tools: [checkStyle(pattern: '**/
ATCResults.xml')], qualityGates: [[threshold: 1, type: 'TOTAL', unstable: true]]

  echo "End - Extension for stage: ${params.stageName}"
}
return this

```

Listing 5.3 Extension for ABAP Test Cockpit Step

Similarly, you can extend other stages as well. Source code for each stage is available in the project “Piper” documentation. In general, the ABAP environment pipeline can already support the general features required for a product lifecycle. But if you want to customize the pipeline or extend it with additional features, you can implement extensions.

5.2.4 Jenkins and Project “Piper”

Jenkins is a widely used open-source automation server. This tool plays a vital role in CI. It provides plug-ins to support CI/CD for any project. Jenkins is used for building, testing, and deploying software. It provides a large number of plug-ins that can be used to integrate various tools and processes into a pipeline. More details were explained in Chapter 2, Section 2.4.1.

Execution of a pipeline is done in the Jenkins server once the pipeline setup is done. Jenkins has a concept called the *shared library* that enables you to create reusable steps in pipelines. In Figure 5.6, the user triggers the pipeline. The pipeline then is executed on the Jenkins server. The shared library is loaded during runtime. The shared library contains the ABAP environment pipeline. During execution, several APIs of the ABAP environment are called.

The Jenkins shared library in this instance is called project “Piper”. This open-source project was initiated by SAP. It contains pipeline and steps implementations. A Jenkins server is required to execute the ABAP environment pipeline, and the shared library in Jenkins is used to create steps and pipelines. The ABAP-specific shared library is part of project “Piper”.

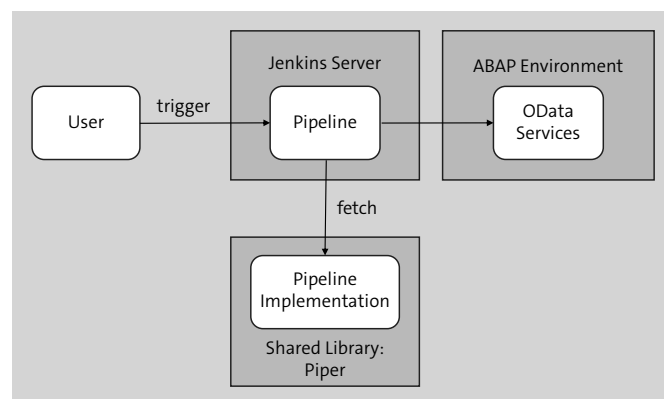


Figure 5.6 Jenkins Server

Now that we've introduced Jenkins, let's explore how to set up the Jenkins server and integrate it with SAP BTP, ABAP environment in the next sections.

Jenkins Server Setup

Docker can be used to easily create a Jenkins instance. First, however, let's go over what a Cx server is. The *Cx server* is a tool used for managing the lifecycle of a preconfigured Jenkins instance. It helps to quickly set up a Jenkins instance on a personal or virtual server by leveraging Docker images. The tool automates the process of starting the Docker image with all the necessary parameters and sidecar images, making the setup process faster and more efficient.

There are certain prerequisites required for Jenkins installation. First, Docker should be installed in the system. Second, there are a number of system requirements, as follows:

- Operating system: Ubuntu 16.04.4 LTS
- Docker: 18.06.1-ce
- Memory: 4 GB reserved for Docker
- Available disk space: 4 GB

For development purposes, the Cx server can also run on Windows or MacOS, but these are not yet supported by SAP. Further configuration for these systems is mentioned in the *Operations Guide* doc for the public repository in GitHub called *SAP/devops-docker-cx-server* (at <https://github.com/SAP/devops-docker-cx-server>).

To install Jenkins, follow these steps:

1. Create a folder called, for example, **DWS-Piper-Demo** in your local directory.
2. Navigate to this directory and initialize the Cx server by executing the following docker command:

```
docker run -it --rm -u $(id -u):$(id -g) -v "${PWD}":/cx-server/mount/ ppiper/cx-server-companion:latest init-cx-server
```

3. Executing this command creates a few files, the most notable of which are the *cx-server* shell script file and the *server.cfg* configuration file. These files help to manage the complete lifecycle of the Jenkins server.
4. Next, you need to provide permissions to the start-up script using the following script:


```
chmod +x ./cx-server
```
5. Now you can start the Cx server (Jenkins) by executing the following script:


```
./cx-server start
```

Creating a Jenkins Job with ABAP Environment Pipeline

For creating a Jenkins job, you should configure the ABAP environment pipeline for a specific scenario. Let's walk through an example of continuous testing in which we execute ABAP test cockpit checks. We'll cover multiple configurations for setting up a sample scenario:

1. Jenkins server setup

This step of Jenkins server setup was explained in the previous section.

2. Git repository creation

You should create a Git repository for pipeline configuration. This repository should be reachable by the Jenkins server. Open and log into <https://github.com>. Create a private repository in GitHub as shown in Figure 5.7 (for more detailed instructions, see Chapter 3, Section 3.2.3).

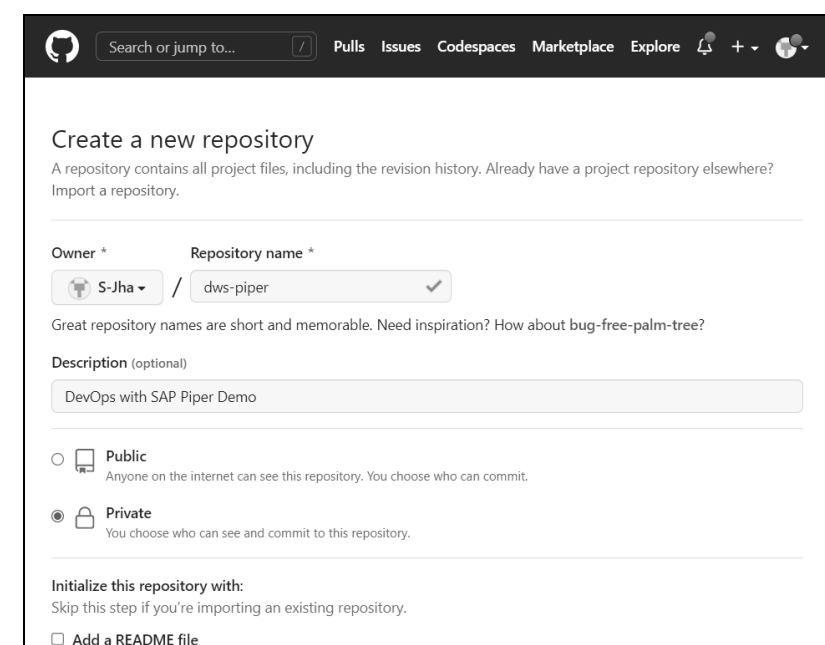


Figure 5.7 Create Repository in GitHub

3. Create GitHub access token

You can create a developer token in GitHub as shown in Figure 5.8. Navigate to **User profile • Settings • Developer Settings • Personal Access Tokens**. Then select **Generate new token**.

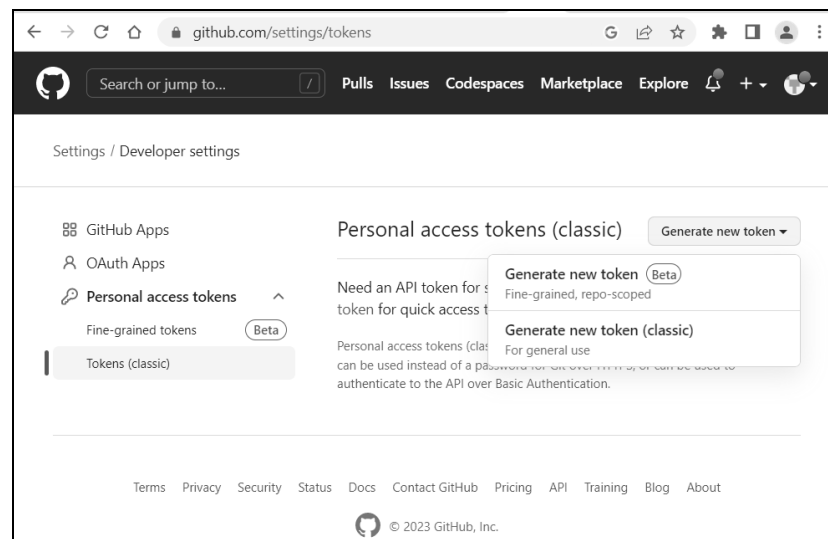


Figure 5.8 GitHub Developer Token

Copy the generated token and save it in the Jenkins credentials store in the **Manage Jenkins** section of the Jenkins UI. Select **Global** and add your credentials by providing your user name, password, and ID. By doing this, you've connected Jenkins with the GitHub repository.

4. Communication arrangement setup in ABAP system

If you don't want to create a system on the fly using the prepare system step and want to connect Jenkins with an existing system, you can configure a communication arrangement in your ABAP system.

Access the SAP Fiori launchpad of the ABAP system and launch the Communication Arrangements app. Create a new communication arrangement by selecting **New**. Enter "SAP_COM_0510" in the **Scenario** field and select **Create**.

Next, you should create a communication system by entering the SID of this system as an example. Select **Inbound Only**. Select the **Add** button to create users for inbound communication. Enter a user name and password. You can save this user to the Jenkins credential store.

5. Jenkinsfile creation

Navigate to your repository in GitHub. Create a file named *Jenkinsfile*, as shown in Figure 5.9. Paste the following content into the Jenkins file:

```
@Library('piper-lib-os')_
abapEnvironmentPipeline script: this
```

The @Library('piper-lib-os') annotation refers to Jenkins configuration. We can use a specific release of the project "Piper" library by adding a version with the following annotation:

```
@Library('piper-lib-os@v1.53.0')_
```

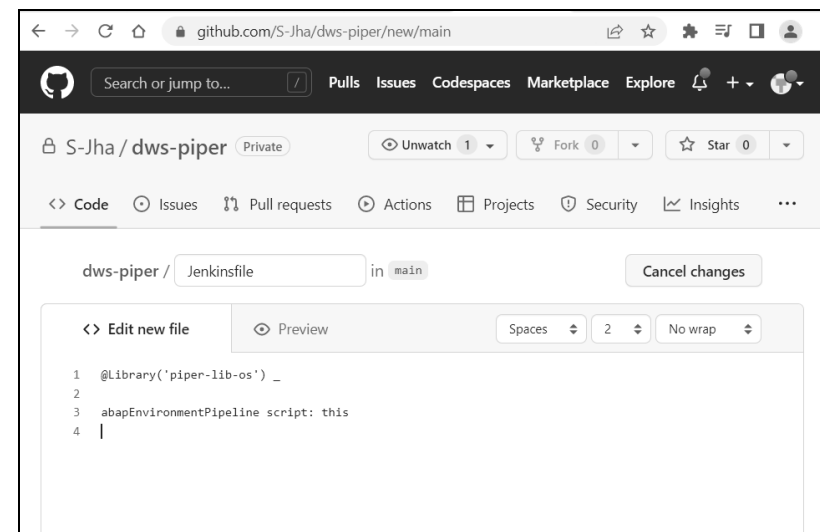


Figure 5.9 Jenkinsfile Creation in Repository

Scroll down to the bottom of the page and select **Commit New File**. The Jenkinsfile is created and shown in the repository, as shown in Figure 5.10.

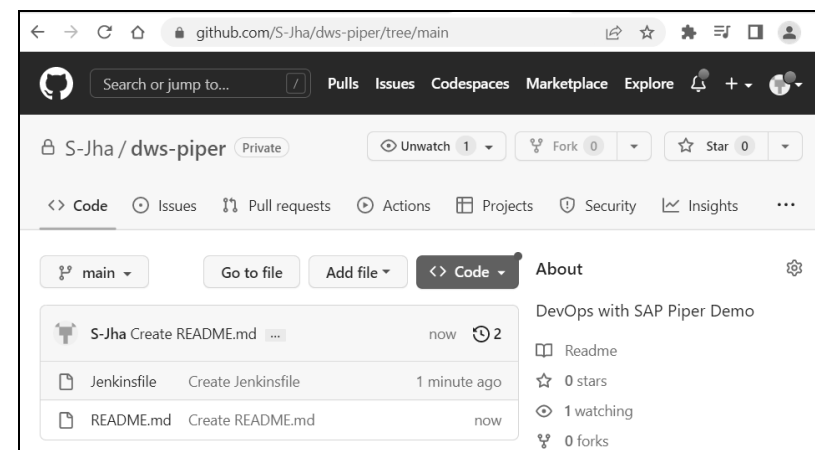


Figure 5.10 Jenkinsfile Created in Repository.

6. Configuration for cloning repository

The clone repositories stage requires a configuration file. Create a *repositories.yml* file using the same procedure as for the Jenkinsfile creation, with the following content:

```
repositories:
  - name: '/DMO/SWC'
    branch: 'master'
```

7. Configurations for ABAP test cockpit checks

Create a config.yml file. Sample code is shown in Listing 5.4. The code shows configurations categorized as *general*, which is applicable across stages and stage-specific configurations as well.

```
general:
  cfApiEndpoint: 'https://api.cf.sap.hana.ondemand.com'
  cfOrg: 'myOrg'
  cfSpace: 'mySpace'
  cfCredentialsId: 'cfAuthentication'
  cfServiceInstance: 'abap_system'
stages:
  Clone Repositories:
    repositories: 'repositories.yml'
    strategy: 'Clone'
```

Listing 5.4 Configuration file for System Clone

In Listing 5.4, parameters starting with *cf* are used for creating systems on the fly. We've included the clone repositories stage here. You can include other stages as well for building a pipeline, like ABAP test cockpit checks, for example. You can combine general stage parameters in the *config.yml* file, as shown in Listing 5.5.

```
general:
  cfApiEndpoint: 'https://api.cf.sap.hana.ondemand.com'
  cfOrg: 'myOrg'
  cfSpace: 'mySpace'
  cfCredentialsId: 'cfAuthentication'
  cfServiceInstance: 'abap_system'
stages:
  ATC:
    atcConfig: 'atcConfig.yml'
```

Listing 5.5 Configuration File for ABAP Test Cockpit Checks

Because the ABAP test cockpit configuration refers to *atcConfig.yml*, you should create that file with the following content:

```
objectSet:
  softwarecomponents:
    - name: "/DMO/SWC"
```

8. Jenkins pipeline creation

Navigate to the Jenkins server UI. Select **New Item** to create a new pipeline. Enter a pipeline name (“dws-pipeline”) in the **Enter an Item Name** field. Select **Pipeline** for your **Project Types** and then select the **Pipeline** tab in the **Pipeline** section. For the **Definition** field, select **Pipeline Script from SCM** and provide the URL path for the Jenkinsfile of your repository in the **Repository URL** field. Select **Save**.

9. Pipeline execution in Jenkins

Once you've configured all of the previous steps, you can go to the Jenkins UI and select **Pipeline**. Next, we can select **Build Now** icon in the left menu. Execution takes some time, and the results are displayed in the Jenkins UI. You can analyze the results further in the Jenkins console.

5.3 Developing Applications with SAP Business Application Studio

Once the continuous integration setup is done in the planning phase, you can explore the implementation phase, consisting of the development and quality phases. We'll discuss development aspects of SAP BTP, ABAP environment in this section, primarily build phase. We'll focus on concepts like the ABAP RESTful application programming model, and tools like ABAP development tools for Eclipse and SAP Business Application Studio, which can be implemented and used in this environment. SAP BTP as a platform supports application development. You can build side-by-side extensions as well. These applications and extensions are loosely coupled.

5.3.1 ABAP RESTful Application Programming Model

The ABAP RESTful application programming model is an architecture that enables efficient end-to-end development of OData services that are optimized for SAP HANA, including SAP Fiori apps. It supports the creation of various types of SAP Fiori applications and enables the publishing of web APIs. It utilizes CDS for defining data models with rich semantics, a service model infrastructure for creating OData services with bindings to an OData protocol, and ABAP-based application services for implementing custom logic. Furthermore, it allows the creation of SAPUI5-based user interfaces.

Prominent features of the ABAP RESTful application programming model are as follows:

- **Service-driven architecture**

This model is built on a service-driven architecture where each RESTful service is responsible for providing data access and business logic. The services are defined using OData and implemented using ABAP.

■ Data modeling

This model uses CDS to define data models. CDS provides a simplified way of defining complex data structures and relationships.

■ SAP Fiori user interface

This model provides an SAP Fiori-based user interface that is responsive and provides a consistent user experience across different devices. SAP Fiori is based on SAPUI5 and provides a set of prebuilt UI components that can be easily configured and customized.

■ Authorization and security

This model provides a built-in authorization and security framework based on the ABAP authorization concept. This framework allows you to control access to data and operations based on user roles and permissions.

■ Business logic

This model provides a way to implement complex business logic using ABAP classes and methods. The business logic can be defined as annotations on the CDS data model.

■ Extensibility

This model allows you to easily extend the data model and user interface without modifying the existing code.

Now let's discuss development components, layers, and supported development scenarios of the ABAP RESTful application programming model.

Development Components

There are three layers of development components for the ABAP RESTful application programming model architecture (see Figure 5.11):

■ Data modeling and behavior

This comprises domain-specific business objects, which are defined using CDS and transactional behavior.

■ Business services provisioning

This includes projection views that focus on specific aspects of the data model, with their respective projection behaviors. These projection views are exposed as business services through the OData protocol.

■ Service consumption

This enables the consumption of various types of OData services, as well as OData web APIs.

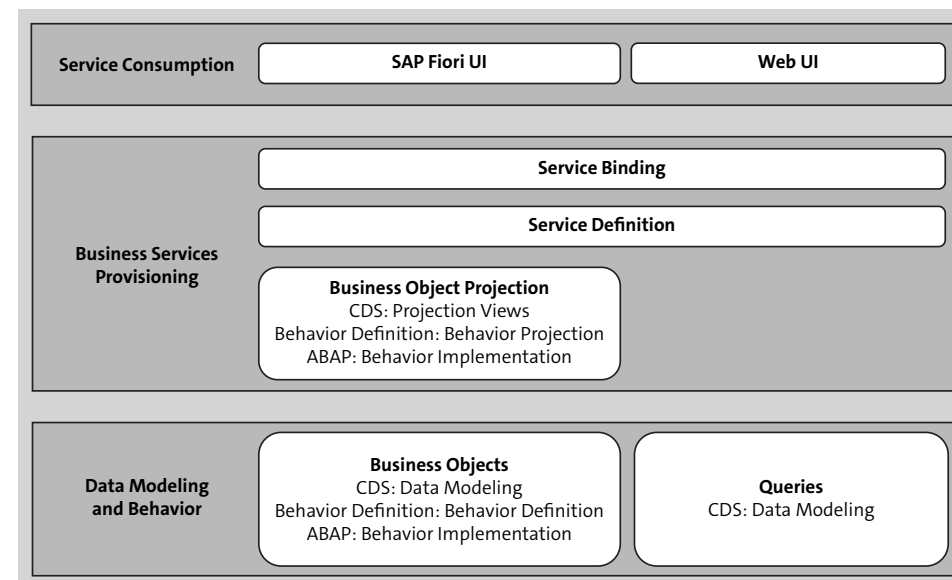


Figure 5.11 ABAP RESTful Application Programming Model Components

Layers

There are different layers involved in the ABAP RESTful application programming model, as follows:

■ Database

The definition of dictionary tables defines the database layer.

■ CDS-based data model

The semantic data model is defined using CDS, which creates views on top of the dictionary tables. By using the CDS layer, you can access and manipulate data that has been persisted in the database.

A *projection* is a subset of fields from the underlying data model that are relevant for the application. This could include UI annotations, for example. Through the service definition, you can specify which data will be exposed as a business service. Service bindings enable you to link service definitions with client-server communication protocols, such as OData. The service binding is used to initiate the SAP Fiori elements app preview, which makes the application visible on the user interface.

■ Transactional behavior

The functionality for creating, updating, and deleting is determined by the behavior definition. The *behavior implementation* is the actual implementation of the behavior. In the managed approach, the creation, update, and delete functionalities are automatically implemented.

Development Scenarios

The ABAP RESTful application programming model uses CDS to build data and service models. ABAP-based application services and user interfaces are based on SAPUI5 components. This model provides a modern, scalable, and extensible way to build RESTful web services and applications on top of the ABAP platform. It provides a consistent way to develop, test, and deploy applications with minimal coding effort.

The ABAP RESTful application programming model supports different types of SAP Fiori applications and publishing web APIs. Some of these scenarios are as follows:

- Developing read-only list reporting apps
- Developing managed transactional apps
- Developing unmanaged transactional apps
- Developing transactional apps with draft capabilities
- Developing a web API
- Developing a UI service with access to a remote service

Primarily, the following two tools are used for development in SAP BTP, ABAP environment: ABAP development tools for Eclipse is used for code involved with the ABAP language, and SAP Business Application Studio is used for SAP Fiori UI development. These two tools are detailed in the next sections.

5.3.2 ABAP Development Tools for ABAP Environment

ABAP development tools is built on the open-source Eclipse platform and is recommended for ABAP development. It helps build ABAP applications for deployment to SAP BTP and helps consume standard SAP BTP services.

For installing ABAP development tools in Eclipse, refer to <https://tools.hana.ondemand.com/#abap>. Follow the steps given there to download Eclipse and add the ABAP development tools plug-in. Figure 5.12 shows ABAP development tools in Eclipse after installation of the plug-in.

You can develop ABAP applications using ABAP development tools for Eclipse via the following steps:

- **ADT local setup**
This includes the Eclipse download and installing the plug-in.
- **Connection to ABAP system**
You should create an ABAP cloud project using the tool and connect to your ABAP system. Developer role assignment and service key creation are prerequisites for this configuration. Select **File**, then **New**, then **ABAP Cloud Project**, as shown in Figure 5.12. Follow the *New ABAP Cloud Project* wizard and select **SAP BTP, ABAP Environment** for **System Connection**. Enter a service key, logon info, and a project name, then select **Finish**.

■ Building ABAP applications

Use the toolset to create different artifacts and build applications. Application development using ABAP development tools has prerequisites like having an ABAP developer role assigned to your user, transport creation roles, transport release roles, software component creation roles, and so on.

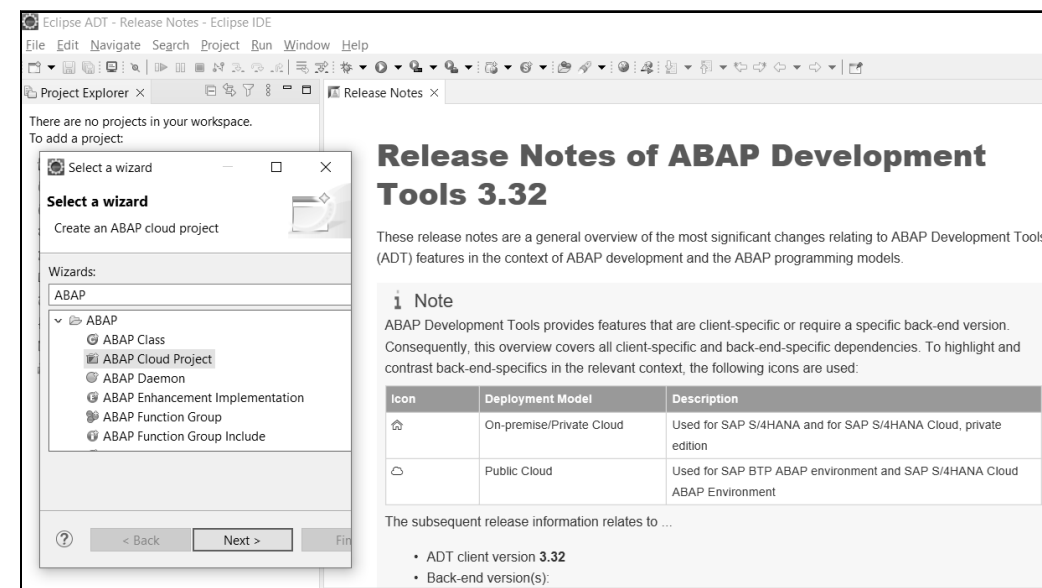


Figure 5.12 ABAP Development Tools in Eclipse

5.3.3 SAP Business Application Studio

SAP Business Application Studio is a development environment hosted on SAP BTP. It's based on an open-source project named Code OSS. Optimized editors and an integrated command line are its key capabilities. It supports pro code application development, facilitating a high productivity approach in your local environment and in a cloud environment. Application development includes coding, testing, and running the application. Faster development helps reduce total costs. The SAP Business Application Studio home page is displayed in Figure 5.13.

Some sample use cases addressed by SAP Business Application Studio are as follows:

- Web full stack application development using SAP Fiori and SAPUI5
- SAP BTP extensions for low-code development
- SAP HANA native application development

Now let's discuss the dev spaces concept associated with this tool, its prominent features, setup steps, and a sample application development using the tool in the next sections.

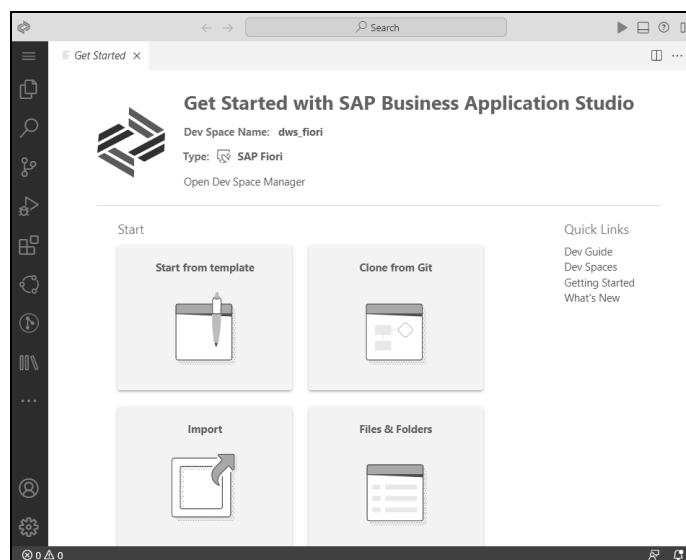


Figure 5.13 SAP Business Application Studio Home Page

Dev Spaces

A *dev space* is a preconfigured private development environment with tools, extensions, and resources required for developing an application of a specific type. SAP Business Application Studio supports the dev space types listed in Table 5.1.

Dev Space Type	Recommended Usage
SAP Fiori	SAP Fiori application development
Low-code-based full-stack cloud application	Low-code application platform-based development
Full-stack cloud application	Application, service, or SAP S/4HANA extension development using SAP Cloud Application Programming Model, SAP Fiori, and Java or Node.js
SAP HANA native application	Native SAP HANA application or analytical model development
SAP mobile application	Mobile application development using SAP Mobile Development Kit
Basic	Basic tools are available by default

Table 5.1 Dev Space Types

These dev spaces support different types of applications. You can select the most suitable dev space and can add additional SAP extensions during space creation based on the type of application and tools required. Sample dev spaces are shown in Figure 5.14.

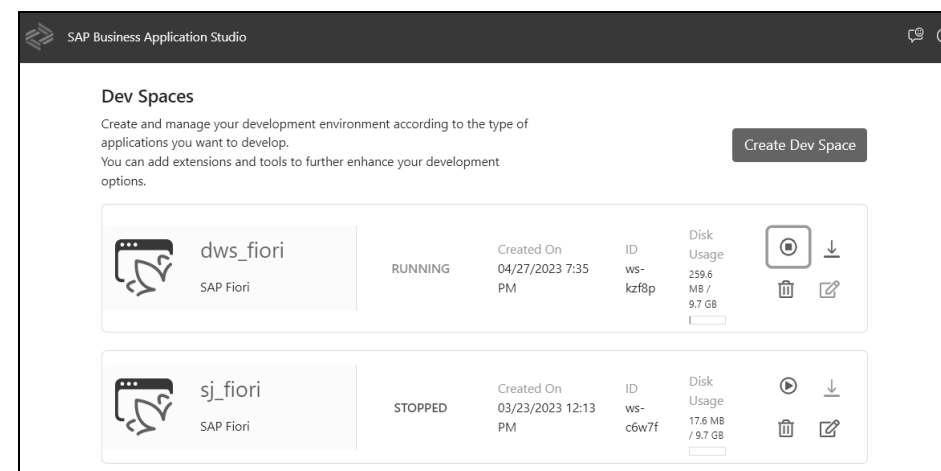


Figure 5.14 Dev Spaces

Features

SAP Business Application Studio opens on a **Get Started** tab. The tab contains quick links, development guides, and help documents, plus application enablement options like start template, Git clone, and so on.

In the left panel, you can see multiple icons corresponding to different features of this tool:

- **Dev space manager**
Dev spaces can be managed (created, stopped, deleted, etc.) here.
- **IDE tools**
Features like search capabilities, integrated command line interface, command palette, outline view, problems view, and Git integration are available here.
- **Productivity tools**
Here you'll find tools to increase development productivity by creating an application from a template and creating run configurations. You'll also find the SAP Cloud Application Programming Model project explorer and MTA tools.
- **Build and deploy applications**
SAP Business Application Studio supports both on-premise and cloud deployment.
- **SAP services consumption**
SAP and non-SAP services can be consumed by applications built using SAP Business Application Studio.
- **Code editor**
Technologies like Java, Node.js, XML, and SAPUI5 are supported for creating an application using the code editor.
- **Graphical editor**
An application's user interface can be created using drag-and-drop controls.

SAP Business Application Studio can be used for creating SAP Fiori applications. You can use Visual Studio Code for application development also. SAP Business Application Studio was explained in detail in Chapter 4, Section 4.3.2. Here, we'll discuss where this tool fits into SAP BTP, ABAP environment.

Setup

The following steps should be configured for setting up SAP Business Application Studio:

1. Subscribe to SAP Business Application Studio in the SAP BTP cockpit for a sub-account.
2. Assign the `Business_Application_Studio_Developer` role collection to developers in the SAP BTP cockpit.
3. Create a service key for the ABAP system in the SAP BTP cockpit in a service instance of the ABAP environment.
4. Create a service destination for the ABAP system in the **Destination** tab of the SAP BTP cockpit for connecting the ABAP system with SAP Business Application Studio.

You can run the **Prepare an Account for ABAP Development** booster for performing the first two steps. A booster is a bundle of steps designed for a particular set of technical configurations. On execution, it opens a wizard. Boosters are available for global accounts and can be accessed in the SAP BTP cockpit navigation menu. Creating service keys and a destination is optional for this tool.

SAP Fiori Application Development

SAP Fiori development is part of UI development. You can create SAP Fiori applications using SAP Business Application Studio or Visual Studio Code. Then you can deploy these applications in SAP BTP, ABAP environment. Figure 5.15 illustrates SAP Fiori UI development in SAP BTP, ABAP environment using SAP Business Application Studio.

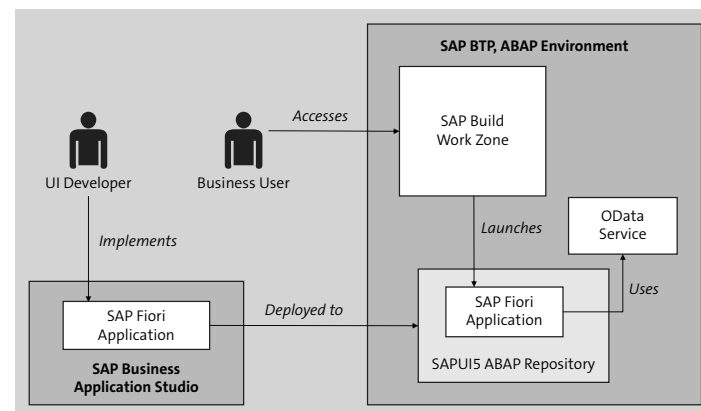


Figure 5.15 SAP Fiori UI Development in SAP BTP, ABAP Environment

Broadly, you can categorize application development into implementation and deployment areas, which are discussed in the following sections.

SAP Fiori Application Implementation

For creating an SAP Fiori application, the SAP Fiori tools application generator available in either SAP Business Application Studio or Visual Studio Code can be used.

Figure 5.16 shows how to open the command palette in SAP Business Application Studio by selecting **View • Command Palette...**. From the command palette, you can open the Application Generator wizard as shown in Figure 5.17 by typing “fiori Open App” into the search field.

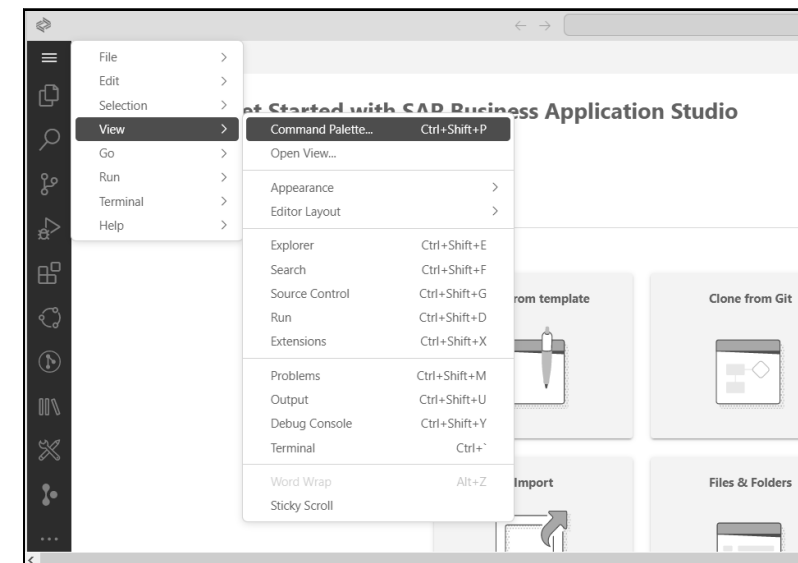


Figure 5.16 SAP Business Application Studio: Command Palette

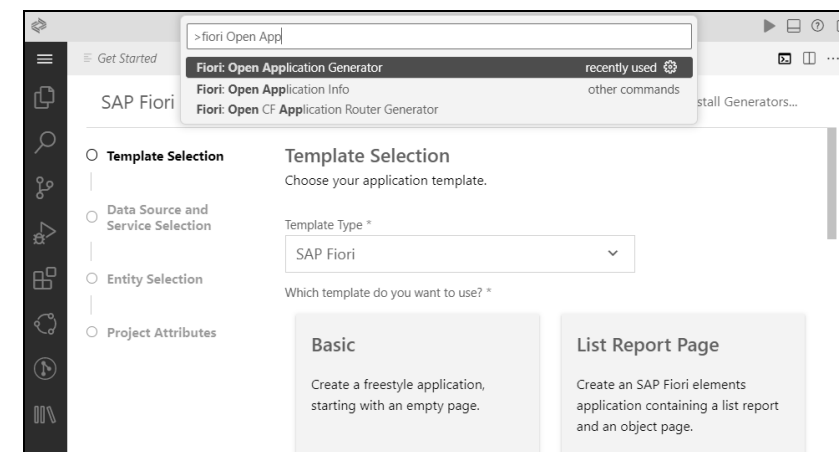


Figure 5.17 SAP Fiori Open Application Generator

You can select any of the available templates and enter other parameters like data source, entity, and project attributes using this wizard to create an SAP Fiori application, as shown in Figure 5.18.

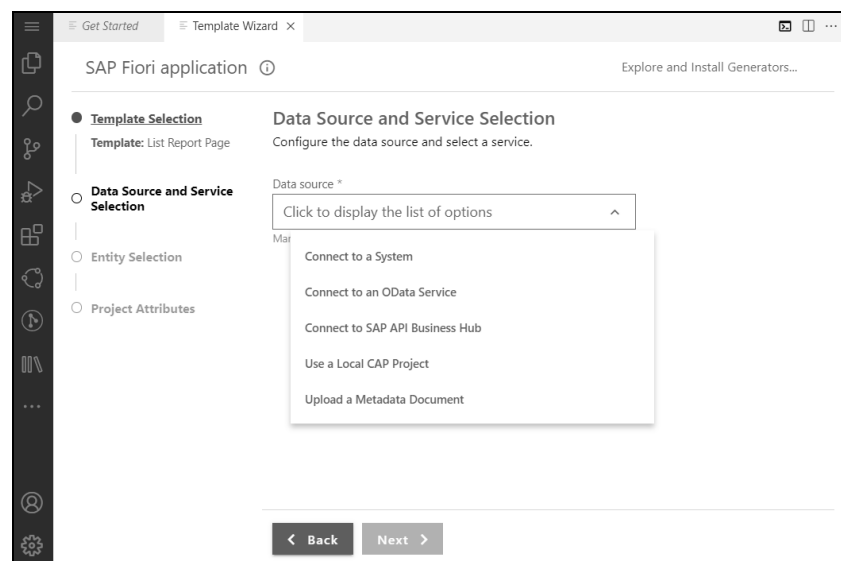


Figure 5.18 SAP Fiori Application Template Wizard

SAP Fiori Application Deployment

Before deployment, you can define a tile for the application and run the application in preview mode. This can be done using the **Run Configurations** icon in the left menu of SAP Business Application Studio and executing the `npm start` command.

For deploying an SAP Fiori application, you should create a deployment configuration and enter a target package and transport request. Make sure the application name has the `z_` prefix, and follow the wizard for other parameters, like deployment, for example. Execute `npm run deploy`. The SAP Fiori application is deployed into the SAPUI5 ABAP repository in SAP BTP, ABAP environment. In the terminal view, logs are displayed.

Note

To summarize, follow these steps for SAP Fiori application development using SAP Business Application Studio:

1. Assign role collections to users.
2. Create a dev space.
3. Set up an organization and space.
4. Create an application.
5. Run the SAP Fiori application for data preview.
6. Deploy your application.

5.4 SAP Cloud ALM for Test Management

In a continuation of the implementation phase, consisting of development and testing, let's explore how the quality phase can be implemented as part of DevOps in SAP BTP, ABAP environment. Quality management plays a prominent role in software development and the same applies here as well. Bringing the quality phase in early in the development cycle helps you receive feedback early, which helps resolve bugs in the product release and contributes to a robust product.

Because the landscape consists of only cloud infrastructure, you can use SAP Cloud ALM for test management. We'll introduce the SAP Cloud ALM product followed by its test management features in the next sections.

5.4.1 Product Overview

SAP Cloud ALM is a cloud-based application lifecycle management (ALM) tool that provides a comprehensive set of features for managing the entire lifecycles of SAP applications. It's designed to help organizations manage the complexities of their SAP system landscapes and to streamline the processes of planning, building, testing, deploying, and operating SAP applications.

It provides a single platform for managing SAP projects, including ALM, project management, quality assurance, and compliance management, as shown in Figure 5.19.

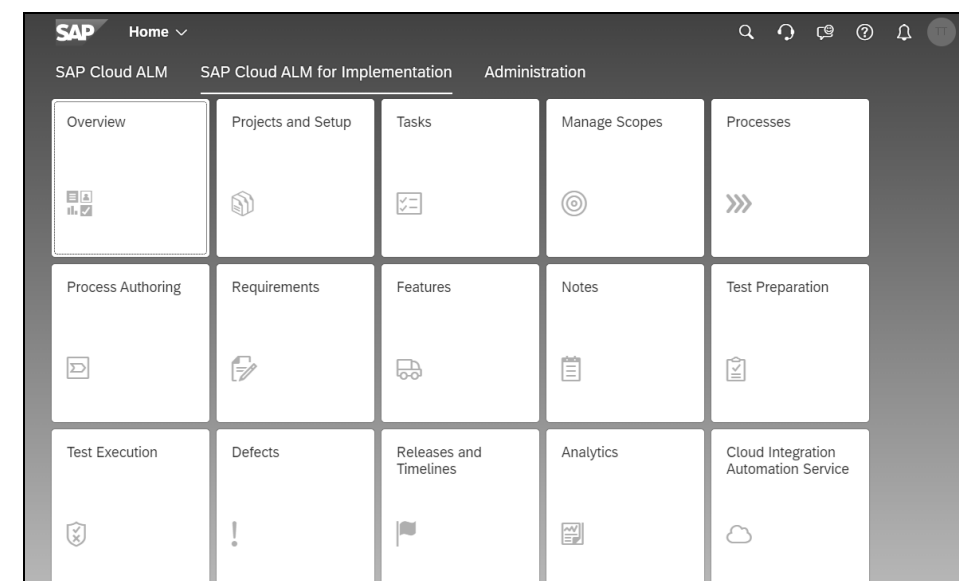


Figure 5.19 SAP Cloud ALM

The benefit of SAP Cloud ALM is a harmonized implementation experience across SAP's cloud solutions. The content-driven implementation based on SAP Activate and

SAP Best Practices processes helps to achieve fast time to value, allowing you to start reaping the benefits of the solution as quickly as possible. The requirement-driven implementation approach enables company-specific innovation, ensuring that the solution meets your specific needs. The solution is instantaneously available, eliminating any deployment or configuration activities, which helps to reduce implementation time and costs. The fast onboarding of project teams is supported by a comprehensive workspace for fit-to-standard workshops, which helps to ensure that the implementation process is efficient and effective. The built-in transparency of implementation progress helps users to stay informed about the status of their implementation, while the seamless integration process covers all relevant implementation capabilities, ensuring a smooth and hassle-free implementation experience.

Notable features of SAP Cloud ALM are as follows:

- **Project management**
SAP Cloud ALM provides project planning and tracking tools to help manage project timelines, resources, and deliverables.
- **Requirements management**
The tool allows users to define, manage, and track requirements across the entire lifecycle of an application.
- **Test management**
SAP Cloud ALM supports test planning, execution, and reporting, and it allows users to manage test cases and defects.
- **Change management**
The tool helps to manage changes to SAP systems, including tracking changes, approvals, and rollbacks.
- **Compliance management**
SAP Cloud ALM provides features to manage regulatory compliance requirements and to ensure that applications meet compliance standards.
- **Collaboration**
The tool includes collaboration features to facilitate communication and teamwork between project stakeholders.

5.4.2 Test Management Feature of SAP Cloud ALM

Quality management is supported by SAP Cloud ALM. It's built on SAP BTP and primarily used for cloud development. It provides management of processes, tasks, testing, and deployment scenarios as core capabilities. We'll explore the capabilities of test management in this section.

Test management supports test scoping, manual testing, automated testing, and monitoring of test execution. Test cases which are marked as *prepared* can be executed. Test planning, test preparation, and test execution applications are integrated with SAP

Cloud ALM. In the test planning phase, you can create a manual test case, add activities to the test case, and add actions to the activities. Management of these test artifacts is possible, like deleting test cases, activities, and so on, in addition to downloading test cases.

SAP Cloud ALM contains a test preparation application showing process flows categorized as ready or not ready and test cases in prepared or not prepared status. The test preparation application lists test process flows analogous to business use case scenarios, which contain individual test cases. Test execution can be performed only when the **Test Readiness** flag is enabled for the individual process flow.

SAP Cloud ALM also contains a test execution overview application. Test execution can be performed from this application. These capabilities are part of the test automation framework. This framework supports automation tools from SAP and non-SAP entities. One such tool is the test automation tool for SAP S/4HANA Cloud. Another automation tool integrated with this framework is from an SAP partner organization named Tricentis, called SAP Test Automation by Tricentis.

Test management in SAP Cloud ALM consists of test preparation, test execution, and defect management as part of the implementation portfolio, as shown in Figure 5.20.

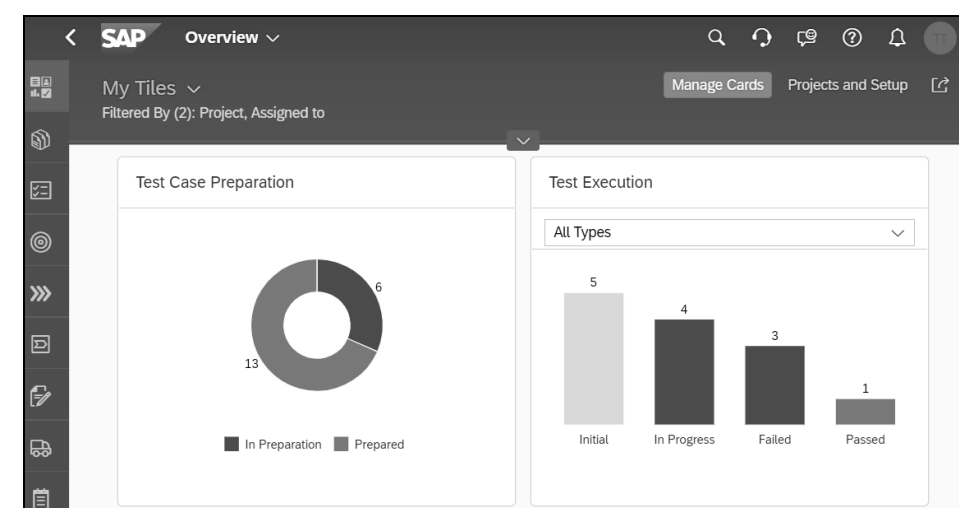


Figure 5.20 Test Management in SAP Cloud ALM

The test management functionality of the software supports lean, Agile, and process-oriented testing approaches. It allows for the reuse of solution process activities as test steps, facilitating a more streamlined and efficient testing process. The solution also offers the integration of automated test tools and supports manual functional testing. The test execution history is captured and can be analyzed, providing insights into the testing process. The analytics also include requirements traceability, which helps to ensure that all requirements have been properly tested and validated.

We'll explore each of these features in the following sections.

Test Preparation

The Test Preparation app enables you to handle and prepare test scenarios, ensuring that the functionalities you've executed function as intended. The Test Preparation app is shown in Figure 5.21.

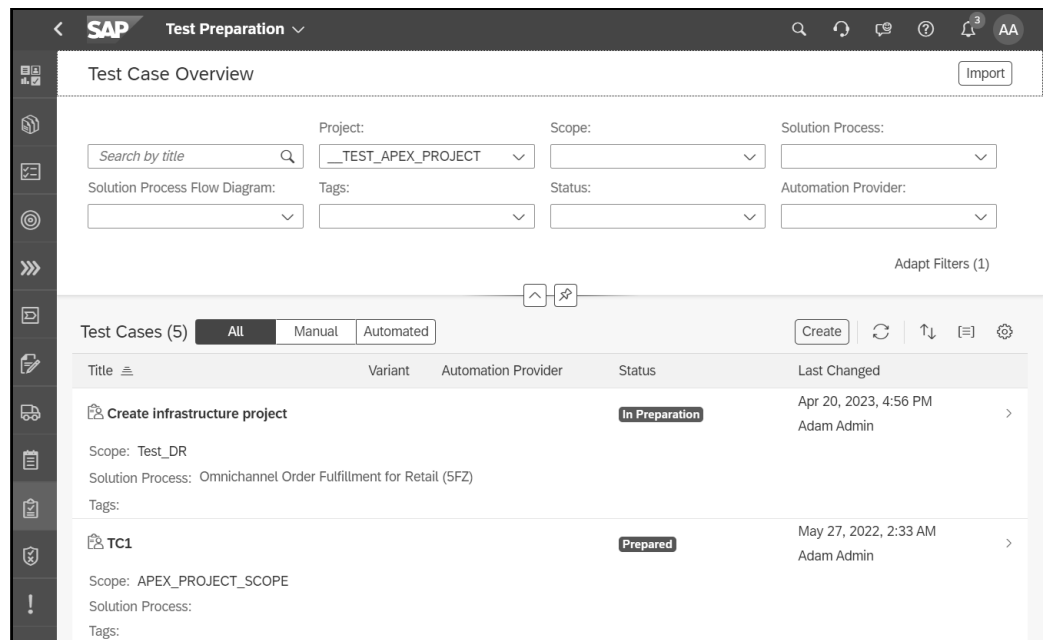


Figure 5.21 Test Preparation App in SAP Cloud ALM

The application provides the following features:

- Obtain a comprehensive view of all your manual and automated test cases in a single application.
- Construct manual test cases and plan their framework and content by specifying activities and actions.
- Integrate automated test cases from the connected test automation tool.
- Establish and handle automated test cases from the connected test automation tool.
- Allocate requirements and user stories to test cases.
- Monitor the advancement of test preparation.

You can create test cases manually in the Test Preparation app. Select **Create** (see Figure 5.21). The **New Test Case** screen is displayed as shown in Figure 5.22.

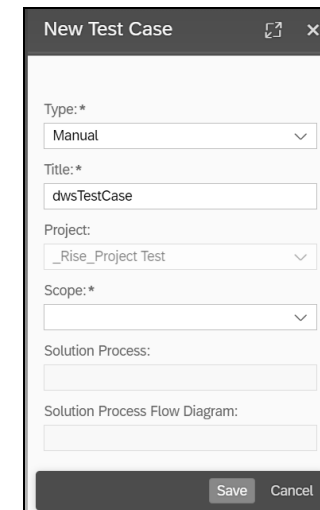


Figure 5.22 Create Manual Test Case

Enter **Title**, **Scope**, and other parameters as per the test case structure and select the **Save** button. The test case is created successfully and displayed as shown in Figure 5.23.

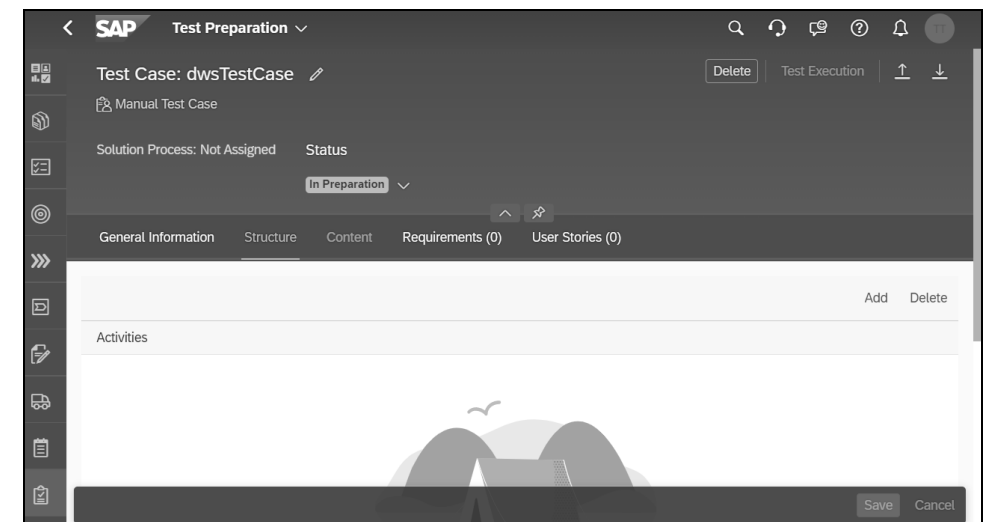


Figure 5.23 Test Case Creation

Test Execution

The Test Execution app allows you to execute test cases that have been prepared in the Test Preparation app and monitor the progress of the test execution. The Test Execution app is shown in Figure 5.24.

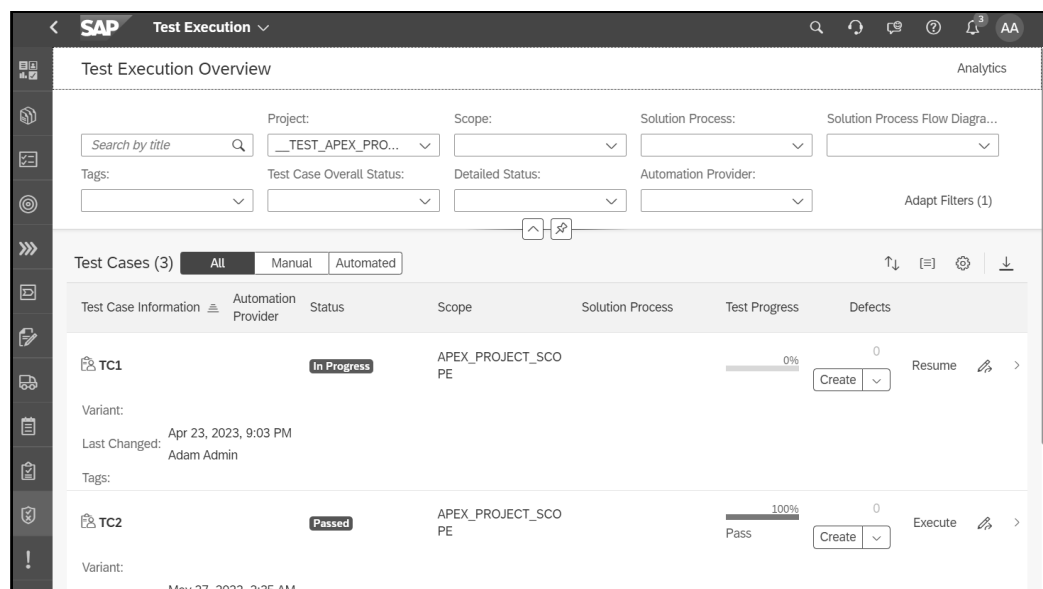


Figure 5.24 Test Execution Application in SAP Cloud ALM

You can carry out both manual and automated test cases, as follows:

- **Manual test cases**

You can assign statuses and enter comments for individual test actions based on the observed application behavior in comparison to the expected outcome. You can also assign defects.

- **Automated test cases**

You can trigger the execution of automated test cases in the connected test automation tool and assign defects.

You can oversee the progress and results of manual test runs executed in SAP Cloud ALM and automated test runs executed in the connected test automation tool. The application provides the following options:

- View a history of all executed manual and automated test runs.
- Gain insight into all the actions that are performed during manual test runs and their statuses.
- Observe the overall status of a test run and the statuses of individual manual test actions.
- Access information about identified errors.

In the Test Execution app, you can execute the test cases created in the Test Preparation app. A test case is saved with the **In Preparation** status on creation. Change the status to **Prepared** as shown in Figure 5.25.

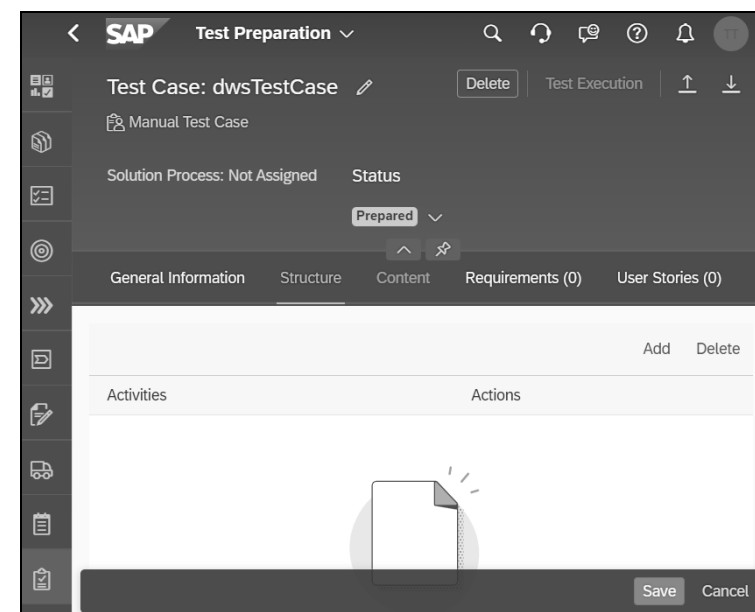


Figure 5.25 Test Case Status Change

Test cases with the **Prepared** status are displayed in the Test Execution app with the **Initial** status, as shown in Figure 5.26. Select the **Execute** button to execute the test case.

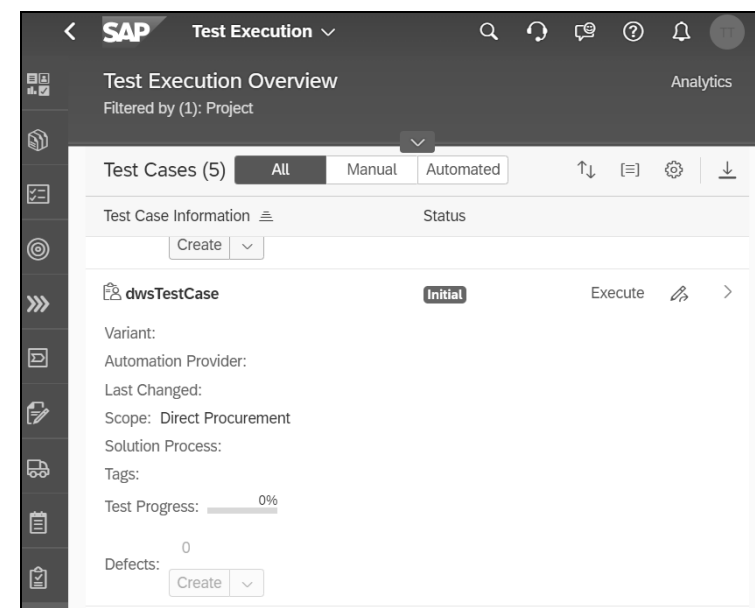


Figure 5.26 Test Case Execution

Defects Management

The Defects Management app provides the ability to create and manage defects for issues that are found during testing or in the application. The Defects Management app is shown in Figure 5.27.

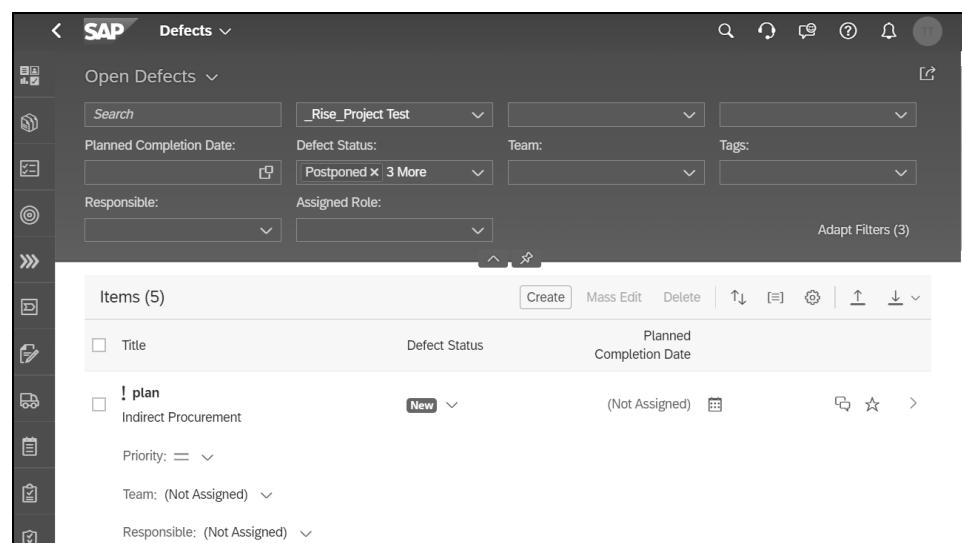


Figure 5.27 Defects Management Application in SAP Cloud ALM

If a test run fails due to an application issue, or a deficiency is discovered during regular application usage, you can utilize the Defects Management app to ensure that the defect is handled and resolved.

The following is a typical lifecycle of defect management:

1. When you create a defect, you need to provide information like a description of the problem, priority, due date, and tags for categorization purposes.
2. The defect is then picked up and assigned to a team, role, or processor.
3. The processor receives a notification about the defect assignment and analyzes and corrects the defect.
4. Once the defect has been resolved, the processor changes the defect status to **Retest Required** and assigns it to the original reporter.
5. The reporter is notified, who retests the test case in Test Execution app.
6. If the retest is successful, you can set the defect status to **Closed**.

5.5 SAP Cloud Transport Management for SAP BTP, ABAP Environment

After the implementation phase (i.e., the development and quality phases) come the release and deploy phases. The SAP Cloud Transport Management service allows you to manage transports in a cloud landscape, as explained in detail in Chapter 4, Section 4.4.2. In this section, we'll explore how to use SAP Cloud Transport Management in SAP BTP, ABAP environment.

In SAP BTP, ABAP environment, a developer uses an Eclipse-based IDE called ABAP development tools to create and modify ABAP code. The toolset includes a debugger and troubleshooting and testing tools. Any changes made are recorded in an ABAP transport request and then released, committing the changes and pushing them to the corresponding Git repository's development branch. Once testing is completed and a release decision is made, a release branch is created from the main branch, and the resulting combination of the software component and the commit ID, Git tag, or branch name is used as a reference for the following export call to SAP Cloud Transport Management. Using a commit ID is recommended to ensure deployability.

In SAP Cloud Transport Management, a transport request containing the reference is created and added to the import queue of the next SAP Cloud Transport Management service node on the configured service transport route. Finally, a transport administrator uses the SAP Cloud Transport Management service UI to generate the import to the target SAP BTP, ABAP environment system configured on the SAP Cloud Transport Management service transport route. This is done by calling an import API and passing the reference. The import is performed asynchronously, and the service monitors its progress status. Once completed, the transport logs from the target SAP BTP, ABAP environment instance are sent back to the service.

SAP Cloud Transport Management can be integrated with SAP BTP, ABAP environment. Enablement, configuration, and integration with a CI/CD pipeline were explained in Chapter 4, Section 4.2. The next sections will cover configurations specific to this environment in SAP BTP.

5.5.1 Configuration for Transport Export

Transport export requires configuration of the Communication Arrangement and Manage Software Components applications, which we'll discuss in the following sections.

Creation of Communication Arrangement

To release transport requests that generate commits, it's necessary to create a new communication arrangement to use an endpoint. First, create a service instance and service key for SAP Cloud Transport Management. Navigate to a subaccount and under

the **Cloud Foundry Environment** tab, create a space. Then create an instance of the SAP Cloud Transport Management service by navigating to **Instances and Subscriptions** under the **Services** navigation menu of the SAP BTP cockpit. Select **Standard** as your plan and provide a unique instance name field (“ctms_instance1”). Selecting **Create** will result in your service instance being created.

Next, to create a service key, we can select the created instance then select **Create Service Key** from the **Actions** menu (represented by triple dots). We should provide a unique service key name. The generated key then can be viewed and downloaded. Next, open the Communication Arrangements app and click **New** to search for the SAP_COM_0599 communication scenario in the list. Enter a name and click **Create**.

On the next screen, create a new communication system by clicking **New** and entering your **System ID**, which is the URI from your service key. Click **Create**. On the communication system creation page, define the **Host Name** parameter under **General Properties** according to the URI property in the service key created for the SAP Cloud Transport Management service. For example, enter “https://transport-service-app-backend.ts.cfapps.eu10.hana.ondemand.com”.

Next, under **OAuth 2.0 Settings**, enter a **Token Endpoint**, which is the URL from your service key—for example, enter “tmststest.authentication.sap.hana.ondemand.com/oauth/token?grant_type=client_credentials”.

To create a user for outbound communication, click **+** and select **OAuth 2.0** from the **Authentication Method** dropdown. Enter your OAuth 2.0 client ID and client secret, which can be obtained from the service key for basic authentication. Click **Create**.

Finally, define a SAP Cloud Transport Management service node name that will be required for the export. Make sure the selected transport node allows uploads, and use the same name when creating the communication arrangement and creating the same in SAP Cloud Transport Management.

Export to SAP Cloud Transport Management Using the Manage Software Components App

To export a commit for a software component and a branch to SAP Cloud Transport Management, first open the Manage Software Components app and select the desired software component and branch. Then choose a commit for the export and click the **Export to cTMS** button. A new dialog box will appear, prompting you to select the SAP Cloud Transport Management service node to use for the export.

Once you've made your selection, click **Export** and a status message in the center of the screen will inform you if the process was completed successfully. If an error occurs, you'll receive an HTTP error code and a description.

5.5.2 Configuration for Transport Import

You can use SAP Cloud Transport Management to transport references to ABAP artifacts created in Git repositories connected to SAP BTP, ABAP environment. All configuration steps for SAP Cloud Transport Management remain the same, except destination creation and communication arrangements creation.

Using the Communication Arrangements app, create an instance of the SAP_COM_0510 communication scenario in the target SAP BTP, ABAP environment instance. To achieve this, create a communication system that employs an inbound communication user, with user ID and password as the authentication method.

Next, select a subaccount subscribed to the SAP Cloud Transport Management service. Create a new destination in **Destinations** under the **Connectivity** tab of the SAP BTP cockpit, as shown in Figure 5.28.

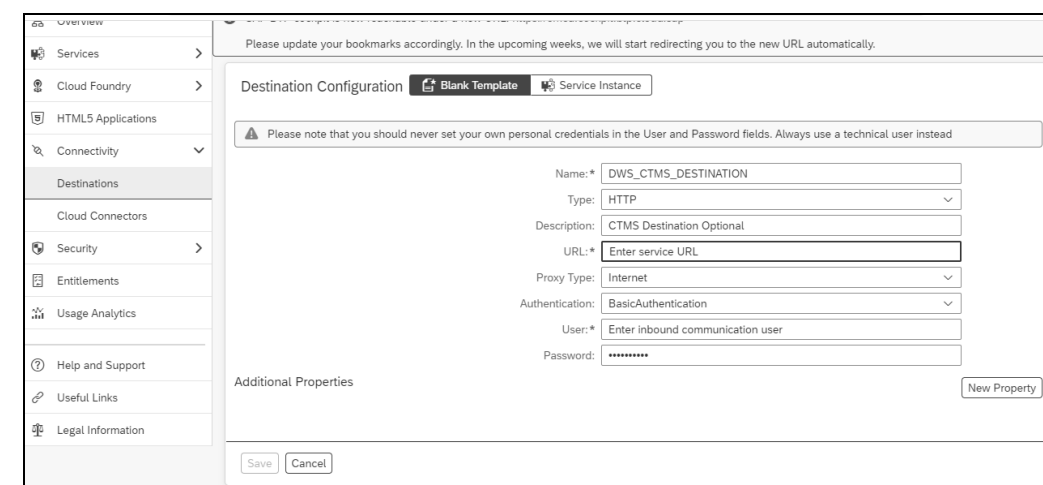


Figure 5.28 Destination Creation

Enter the destination **Name**, select **Type** (HTTP), set **URL** as the already-generated service URL (created for the SAP_COM_0510 communication scenario in the Communication Arrangement app), set **Proxy Type** to **Internet**, and click **Save**. You can check the created destination using the **Check Connection** button after saving.

5.6 Summary

This chapter explained DevOps implementation for SAP BTP, ABAP environment. Starting with the evolution of ABAP-based development in the introduction section, we then explored the importance of DevOps in today's world and how to set up development infrastructure with a lean and Agile approach for CI/CD. This was followed by

development topics like ABAP application development including automation and test management using SAP Cloud ALM.

Next, we explored SAP Cloud Transportation Management configuration in SAP BTP, ABAP environment. (Operation and monitoring features within SAP BTP were discussed in the previous chapter.) Effective feedback mechanisms are critical for any process and are handled in the operations phase. Adoption of DevOps practices in SAP BTP, ABAP environment can help to improve the speed, quality, and reliability of application development and deployment and can enable organizations to rapidly innovate and respond to changing business requirements.

The next chapter explores DevOps for SAP systems in a hybrid landscape. This scenario will be the most common for companies running SAP that are transitioning to the cloud or evaluating some areas on cloud platforms.

Contents

Acknowledgments	15
Preface	17
1 Introduction to DevOps	27
<hr/>	
1.1 What Is DevOps?	27
1.1.1 Solving Problems Using DevOps	28
1.1.2 Definition of DevOps	30
1.1.3 Four Principles of DevOps	30
1.1.4 Key Benefits of DevOps	32
1.2 Continuous Integration and Delivery	33
1.3 How Does DevOps Work?	35
1.3.1 Plan Phase	36
1.3.2 Develop Phase	36
1.3.3 Build Phase	37
1.3.4 Test Phase	37
1.3.5 Release Phase	38
1.3.6 Deploy Phase	38
1.3.7 Operate Phase	38
1.3.8 Monitor Phase	39
1.3.9 Guidelines for DevOps	40
1.4 History: The Evolution of DevOps	41
1.5 Myths and Misconceptions about DevOps	41
1.6 Who Needs DevOps?	43
1.7 DevOps Culture	45
1.8 Challenges in Implementing DevOps	47
1.8.1 Tools and Services	48
1.8.2 Culture Shift	49
1.8.3 Lack of Vision and Support from Top Management	50
1.8.4 Too Much Focus on Technologies	52
1.9 Summary	53

2	DevOps Tools	55
2.1	Code and Version Control Tools	55
2.1.1	Git	56
2.1.2	GitHub	59
2.2	Build Tools	60
2.2.1	Maven	61
2.2.2	Chef	63
2.2.3	Jenkins	64
2.3	Test Automation Tools	67
2.3.1	Selenium	68
2.3.2	JUnit	71
2.3.3	SonarQube	73
2.3.4	Jenkins	75
2.4	Deployment Tools	76
2.4.1	Docker	77
2.4.2	Kubernetes	83
2.4.3	Jenkins	85
2.4.4	GitLab	85
2.4.5	Chef	87
2.5	Monitoring Tools	88
2.5.1	Dynatrace	88
2.5.2	Grafana	89
2.5.3	Splunk	91
2.6	SAP's DevOps Portfolio	92
2.6.1	DevOps Tools Offered in SAP S/4HANA	92
2.6.2	DevOps Tools Offered in SAP Business Technology Platform	94
2.7	Summary	96
3	DevOps for On-Premise SAP Systems	97
3.1	Introduction to DevOps in SAP S/4HANA	97
3.1.1	DevOps Features in SAP S/4HANA	98
3.1.2	Evolution of Change and Transport System	100
3.2	abapGit	104
3.2.1	Introduction	104
3.2.2	Installation	105

3.2.3	Configuration	109
3.2.4	Implementation of a Sample Project	115
3.2.5	Development Guidelines and Best Practices	116
3.3	Change and Transport System	121
3.3.1	Introduction	121
3.3.2	Configuration	125
3.3.3	Implementation of a Sample Project	130
3.4	Enhanced Change and Transport System	130
3.4.1	Introduction	131
3.4.2	Configuration	131
3.4.3	Implementation of a Sample Project	133
3.4.4	Integration with SAP Solution Manager	134
3.5	Central Change and Transport System	134
3.5.1	Introduction	134
3.5.2	Configuration	136
3.5.3	Implementation of a Sample Project	137
3.5.4	Integration with SAP Solution Manager	139
3.6	Continuous Integration for ABAP On-Premise with Git-Enabled CTS	139
3.6.1	Introduction	139
3.6.2	Configuration	140
3.6.3	Setup and Implementation	141
3.6.4	GitHub Repository Creation	141
3.6.5	Authentication in gCTS	141
3.6.6	Integration with ABAP Workbench	142
3.6.7	Integration with Change Request Management	142
3.7	SAP HANA Transport for ABAP	143
3.8	Continuous Testing in ABAP	145
3.8.1	Introduction	145
3.8.2	Unit Testing with GitHub Actions	149
3.8.3	SAP Solution Manager Test Suite and Focused Build for Testing	151
3.9	ABAP Continuous Integration and Delivery Pipeline	153
3.9.1	SAP Continuous Integration and Delivery	153
3.9.2	SAP Solution Manager	154
3.9.3	Project "Piper"	154
3.9.4	ABAP Development Tools for Eclipse	158
3.9.5	Third-Party Tools	158
3.10	End-to-End DevOps Scenario for SAP S/4HANA	158
3.10.1	Planning Phase	159
3.10.2	Develop, Build, and Test Phases	160

3.10.3	Release and Deploy Phases	161
3.10.4	Operate and Monitor Phases	161
3.11	Summary	162
4	DevOps with SAP Business Technology Platform	163
4.1	SAP BTP DevOps Portfolio	163
4.2	Services for the Planning Phase	166
4.2.1	SAP BTP Landscape Setup	167
4.2.2	Best Practices and Guidelines for Planning Phase	169
4.2.3	Continuous Integration and Delivery in SAP BTP	170
4.2.4	Configure and Run a CI/CD Pipeline for an SAP Fiori Project	175
4.2.5	Configure and Run a CI/CD Pipeline for an SAP Cloud Application Programming Model Project	189
4.2.6	Configure and Run a CI/CD Pipeline Using Project “Piper”	193
4.2.7	Choosing the Right CI/CD Solution from SAP	195
4.3	Services for the Develop, Build, and Test Phases	196
4.3.1	SAP BTP Development Environment and Programming Models	197
4.3.2	SAP Business Application Studio	198
4.3.3	SAP Cloud Application Programming Model	204
4.3.4	Multitarget Application	206
4.3.5	SAP Cloud SDK	209
4.4	Services for the Release and Deploy Phases	210
4.4.1	Continuous Delivery	210
4.4.2	SAP Cloud Transport Management	211
4.5	Services for the Operate and Monitor Phases	217
4.5.1	SAP Alert Notification Service for SAP BTP	218
4.5.2	SAP Automation Pilot	219
4.6	An End-to-End DevOps Scenario for Cloud Landscapes with SAP BTP	221
4.6.1	Planning Phase	221
4.6.2	Develop, Build, and Test Phases	223
4.6.3	Release and Deploy Phases	224
4.6.4	Operate and Monitor Phases	224
4.7	Summary	224

5	DevOps for SAP BTP, ABAP Environment	225
5.1	Introduction to SAP BTP, ABAP Environment	226
5.1.1	What Is SAP BTP, ABAP Environment?	226
5.1.2	Evolution of ABAP	228
5.1.3	Need for ABAP Environment in SAP BTP	229
5.1.4	Benefits of ABAP Environment in SAP BTP	230
5.2	Continuous Integration and Continuous Delivery Tools in SAP BTP, ABAP Environment	230
5.2.1	Git Code Management	231
5.2.2	APIs	232
5.2.3	ABAP Environment Pipeline	232
5.2.4	Jenkins and Project “Piper”	237
5.3	Developing Applications with SAP Business Application Studio	243
5.3.1	ABAP RESTful Application Programming Model	243
5.3.2	ABAP Development Tools for ABAP Environment	246
5.3.3	SAP Business Application Studio	247
5.4	SAP Cloud ALM for Test Management	253
5.4.1	Product Overview	253
5.4.2	Test Management Feature of SAP Cloud ALM	254
5.5	SAP Cloud Transport Management for SAP BTP, ABAP Environment	261
5.5.1	Configuration for Transport Export	261
5.5.2	Configuration for Transport Import	263
5.6	Summary	263
6	DevOps for Hybrid SAP Systems	265
6.1	Introduction to Hybrid Change Management	265
6.1.1	Intelligent Enterprise with SAP	266
6.1.2	Hybrid SAP Systems	267
6.1.3	Hybrid Change Management	270
6.2	Available Tools and Their Purposes	272
6.2.1	Continuous Integration and Delivery	273
6.2.2	Project “Piper”	275
6.2.3	SAP Cloud Transport Management	277
6.2.4	Enhanced Change and Transport System	278
6.2.5	Application Lifecycle Management	278

6.3	Tool Integration with Change Management in Hybrid Scenarios	284
6.3.1	SAP Cloud Transport Management with CTS+ and SAP Solution Manager	284
6.3.2	SAP Cloud ALM with External Incident Systems	290
6.4	Continuous Development	292
6.4.1	Development	293
6.4.2	Test Management	293
6.4.3	Release Management	295
6.5	Operation Monitoring with SAP Solution Manager	296
6.5.1	Integration Monitoring	296
6.5.2	Job Management	297
6.5.3	Exception Management	298
6.5.4	Data Consistency Management	299
6.5.5	Business Process Monitoring	300
6.5.6	User Experience Monitoring	302
6.6	DevOps for SAP Data Intelligence	302
6.6.1	Solution Overview	303
6.6.2	Continuous Integration and Deployment	303
6.6.3	Internal Transportation	307
6.6.4	Monitoring	308
6.7	An End-to-End DevOps Scenario for a Hybrid Landscape	308
6.7.1	Planning Phase	309
6.7.2	Develop, Build, and Test Phases	309
6.7.3	Release and Deploy Phases	310
6.7.4	Operate and Monitor Phases	310
6.8	Summary	311
7	Best Practices	313
7.1	Continuous Integration and Continuous Delivery	313
7.1.1	Continuous Integration	314
7.1.2	Continuous Delivery	319
7.2	Microservices	324
7.3	Infrastructure as Code	326
7.4	Monitoring and Logging	327

7.5	Communication and Collaboration	328
7.6	Fail Fast	329
7.7	Summary	330
8	Security	333
8.1	Common Security Tools and Concepts	333
8.2	Secure Code Analytics	335
8.2.1	Manual Secure Code Analytic Techniques	336
8.2.2	ABAP-Related Scans	339
8.2.3	Java-Related Scans	344
8.2.4	SAP Fiori-Related Scans	353
8.3	Automated Security Testing	360
8.4	Security Vulnerability Monitoring	361
8.4.1	Vulnerability Scanning	362
8.4.2	Continuous Monitoring	363
8.4.3	Best Practices	364
8.5	Implementation	365
8.6	Summary	366
9	Outlook	367
9.1	Emerging DevOps Technologies	367
9.1.1	SAP Build	368
9.1.2	Code Inspector	369
9.1.3	SAP Focused Run	369
9.1.4	Cloud Native Technologies	369
9.2	DevOps for SAP BTP, Kyma Runtime	371
9.3	Future Directions of DevOps in SAP	372
9.3.1	Artificial Intelligence for IT Operations	372
9.3.2	No Operations	372
9.3.3	GitOps	373
9.3.4	DevSecOps	373
9.3.5	Low-Code/No-Code	374
9.3.6	Role of Cloud Adoption in DevOps	374

9.4	The First Steps in Your DevOps Journey	374
9.5	How to Stay Up to Date	376
9.6	Summary	378
	The Authors	379
	Index	381

Index

.pipeline/config.yml 155

A

ABAP development tools 102, 158, 246
ABAP environment pipeline 174, 233
 configure 235
 extensions 236
 scenarios 233
 stages 233, 234
ABAP evolution 228
ABAP programming model for SAP Fiori ... 229
ABAP RESTful application programming
 model 229, 243
 development components 244
 development scenarios 246
 features 243
 layers 245
ABAP scans 339
ABAP test cockpit 118, 148, 232, 242, 343
ABAP Unit 148
ABAP workbench 141
abapGit 93, 103, 104
 best practices 116
 configuration 109
 connect to Git server 112
 installation 105
 license 118
 overview 104
 sample project 115
Agile methodology 53, 99, 330
Alerts 328
Amazon 43
API gateways 325
Application lifecycle management (ALM) ... 278
Application security 335
Application vulnerability scanning 360
Artificial intelligence 372
Authenticated scanning 362
Automate release process 319
Automated security testing 360
Automated tests 70, 315, 334
Automation 31, 98

B

Behavior implementation 245
Best practices 313
 planning phase 169
Black Duck 347
 create report 348
 process flow 348
 scenarios 347
Black Duck Binary Analysis 350
Blue-green deployment 321
Branch 56
Branching strategy 117
Build automation 315
Build phase 37, 160, 164, 196, 223, 309
Business process monitoring 300
 benefits 301

C

Central Change and Transport System
 (cCTS) 97, 101, 134
 configuration 136
 key features 134
 sample project 137
 SAP Solution Manager 139
Change and Transport System (CTS) 93, 97,
 101, 121
 activate 125
 configuration 125
 DevOps 124
 evolution 100
 overview 121
 sample project 130
Change management 283
 hybrid scenarios 284
Change Request Management (ChaRM) 284
Checkmarx 353
 process flow 355
 scenarios 354
Chef 63, 64
 deployment 87
CI/CD pipeline 34, 42, 172, 274
 ABAP system 153
 project 193
 *SAP Cloud Application Programming
 Model* 189

- CI/CD pipeline (Cont.)
 - SAP Cloud Transport Management* 217
 - SAP Fiori app* 175
 - trigger* 184
 - Citizen developer 368
 - Cloud adoption 374
 - Cloud connector 189
 - Cloud Foundry 232
 - Cloud MTA Build Tool 207
 - Code analysis 74, 348
 - Code development 227
 - Code Inspector 339, 369
 - development objects* 340
 - features* 342
 - process flow* 342
 - results* 340, 341
 - Code reviews 120
 - Code transfer 117
 - Collaboration 32, 46, 99, 283, 328
 - Commit messages 120
 - Communication 328
 - Communication arrangement 240
 - create* 261
 - Communication Arrangements app 263
 - Component analysis 350
 - Component tests 147
 - config.yml 186
 - Containerization 78, 370
 - Continuous availability 322
 - Continuous delivery (CD) 33, 210
 - best practices* 319
 - Continuous deployment 99
 - Continuous development (CD) 292
 - Continuous improvement 50
 - Continuous integration (CI) 33
 - best practices* 314
 - Continuous integration and delivery
 - (CI/CD) 33, 94, 99, 273
 - best practices* 313
 - choosing an approach* 195
 - Continuous Integration and Delivery
 - Best Practices Guide 95, 170, 174, 231, 273
 - Continuous monitoring 363
 - Continuous testing 145, 372
 - Cookbooks 63
 - Culture 31, 45
 - shift* 49
 - Culture, automation, measurement,
 - and sharing (CAMS) 30
 - Customer centricity 32
 - Cx server 238
- ## D
- Data consistency management 299
 - benefits* 300
 - Debugging 227
 - Defects Management app 260
 - Definition 30
 - Delivery units 143
 - Deploy phase 38, 161, 164, 210, 224, 310
 - Deployment 76
 - Deployment pipeline 321
 - Dev space 199, 248
 - Develop phase 36, 160, 164, 196, 223, 309
 - Development and operations teams 28
 - Development management 293
 - Development history 41
 - DevOps Research and Assessment (DORA) 41
 - DevSecOps 373
 - Distributed version control system 57
 - Docker 77, 173
 - benefits* 81
 - client* 80
 - daemon* 80
 - host* 80
 - operation* 80
 - overview* 78
 - with SAP BTP* 81
 - Docker Hub 80
 - Docker image 79, 82
 - Documentation 318
 - Dynamic application security testing
 - (DAST) 335
 - Dynatrace 88
 - ActiveGate extensions* 89
 - monitoring extension for ABAP* 89
- ## E
- Eclipse 246
 - Emerging technologies 367
 - Enhanced Change and Transport
 - System (CTS+) 93, 97, 101, 130, 278, 284, 286, 295
 - configuration* 131
 - export/import systems* 132
 - integration benefits* 288
 - overview* 131
 - register application* 132
 - sample project* 133
 - with CharM* 287
 - Ethical hacking 338
 - Etsy 43

- Event situation APIs 291
 - Exception management 298
 - benefits* 298
 - Exception monitoring 281
 - External API subscription 290
- ## F
- Fail fast 329
 - Failover plan 168
 - Feature flag 323
 - Feedback 47
 - File formats 117
 - Focused Build for SAP Solution Manager ... 152
 - Fortify 344
 - admin* 345
 - dashboard* 345
 - process flow* 346
 - reports* 345
 - Function modules 228
 - Functional testing 67
- ## G
- Git 56
 - benefits* 58
 - process flow* 57
 - server* 142
 - Git code management 231
 - Git flow 119
 - Git repository 239
 - create* 109
 - private* 111
 - Git-enabled change and transport
 - system (gCTS) 58, 94, 98, 103, 139
 - ABAP workbench* 142
 - authentication* 141
 - CharM* 142
 - configuration* 140
 - overview* 139
 - process flow* 58
 - registry* 140
 - Git-enabled CTS app 140, 142
 - GitHub 59, 179–181, 223
 - access token* 240
 - benefits* 59
 - configure credentials* 188
 - login* 109
 - repository creation* 141
 - secrets* 150
 - with SAP BTP* 60
 - GitHub Actions 149, 173
 - GitLab 85
 - benefits* 86
 - GitOps 373
 - Grafana 89
 - features* 90
 - Grafana (Cont.)
 - with SAP BTP* 90
 - Groovy 275
- ## H
- Health monitoring 282
 - Hybrid change management 265, 270
 - best practices* 271
 - challenges* 271
 - Hybrid landscape 265
 - end-to-end scenario* 308
 - SAP systems* 267
 - Hyperscalers 166
- ## I
- Identity and access management (IAM) 334
 - Immutable infrastructure 320
 - Implementation challenges 47
 - Infrastructure as code (IaC) 63, 313, 326, 370
 - Integrate early and often 316
 - Integration monitoring 296
 - features* 297
 - Integration tests 34, 147
 - Intelligent automation 372
 - Intelligent enterprise 266
 - Intelligent event processing 290
- ## J
- Java scans 344
 - Jenkins 64, 237, 275
 - benefits* 66
 - create job* 239
 - deployment* 85
 - install* 238
 - pipeline* 243
 - process flow* 64
 - server setup* 238
 - test automation* 75
 - Jenkinsfile 155, 240, 275
 - Job management 297
 - JUnit 71
 - benefits* 72
 - process* 71
 - tests* 71

K

- Keep builds clean 316
- Kubernetes 83, 371
 - features* 83
 - manifest file* 84

L

- LCNC development 374
- License compliance management 348, 350
- Linux command line tool 173
- Logging 327

M

- Manage Software Components app 262
- Master-slave architecture 84
- Maven 61
 - benefits* 62
 - process flow* 61
- Measurement 31
- Mend 356
 - home screen* 357
 - process flow* 359
 - scan* 358
 - scenarios* 357
- Microservices 324, 370
- Microservices architecture 313, 324
- Monitor build status 317
- Monitor phase 39, 161, 165, 217, 224, 310
- Monitoring 88, 327
- Monolithic application 324
- Multitarget application 206
 - benefits* 208
 - example* 208
 - MTA descriptor* 207
- Multitarget application archive builder 276
- Myths and misconceptions 41

N

- National Vulnerability Database (NVD) 358
- Netflix 44
- No operations (NoOps) 372

O

- Object-oriented programming (OOP) 118
- OData services 274
- On-premise landscape 97

- openSAP 378
- Operate phase 38, 161, 165, 217, 224, 310

P

- Paradox of choice 166
- Penetration testing 337
- Performance analysis 227
- Performance testing 67
- Planning phase 36, 159, 164, 221, 309
 - services* 166
- Predictive analytics 372
- Principles 30
- Program ZDWS_ABAPGIT 107
- Project 95, 154, 170, 172, 173, 194, 237, 273, 275
 - Project “Piper” 231
 - Project management 282
 - Project object model (POM) 61
 - Projection 245
 - Protecode 350
 - process flow* 351
 - sample results* 352
 - scan* 351
 - scenarios* 350

Q

- Quality assurance 123
 - configure* 128
- Quality gate 74
- Quality Gate Management (QGM) 285

R

- Regression testing 67
- Release management 295
- Release phase 38, 161, 164, 210, 224, 310
- Release size 319
- Reliability 32
- Remote Function Call (RFC) 131
- Repository 56
- Requirements management 283
- Risk management 348, 350
- Risk mitigation 323
- Rollback 322

S

- SAP Add-On Assembly Kit 235
- SAP Alert Notification 95, 184, 218
 - integration* 219

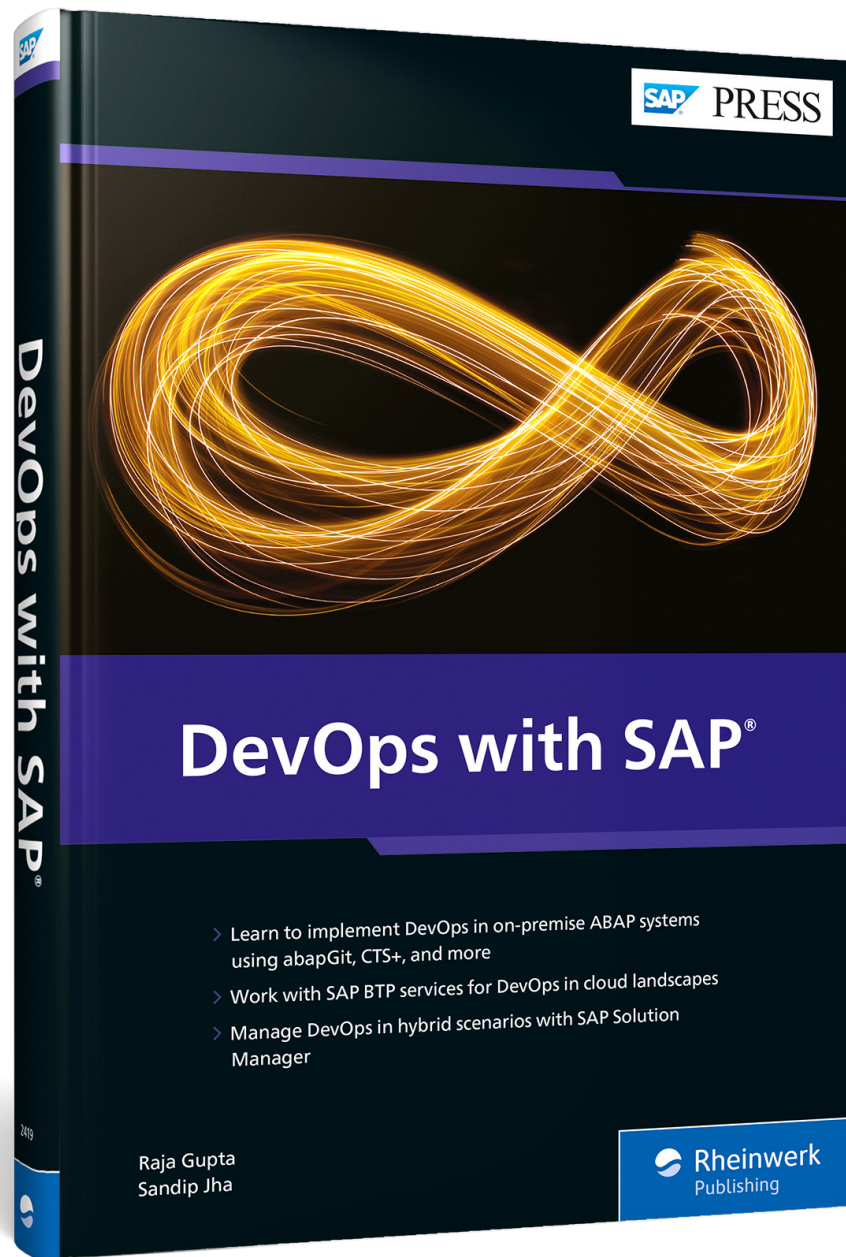
- SAP Ariba 270
- SAP Automation Pilot 96, 219, 220
- SAP BTP cockpit 167, 177, 201
- SAP BTP, ABAP environment 225
 - APIs* 232
 - benefits* 230
 - development tools* 227
 - overview* 226
- SAP BTP, Cloud Foundry environment 81, 176, 197
 - configure credentials* 189
- SAP BTP, Kyma runtime 82, 371
- SAP Build 368
- SAP Business Application Studio 95, 198, 243, 247, 293
 - access* 202
 - assign role collection* 202
 - command palette* 251
 - enable* 201
 - features* 249
 - how it works* 199
 - layout editor* 200
 - setup* 250
- SAP Business Technology Platform (SAP BTP) 34, 94, 163, 266
 - accounts* 167
 - command line interface* 168, 193
 - DevOps portfolio* 163
 - end-to-end scenario* 221
 - landscape setup* 167
- SAP Business Technology Platform Newsletter 376
- SAP Cloud ALM 93, 253, 281
 - benefits* 283
 - external incident systems* 290
 - features* 254, 282, 290
 - integration benefits* 291
 - product overview* 253
 - test features* 294
 - test management* 254
- SAP Cloud ALM for security 363
- SAP Cloud Application Programming Model 95, 172, 189, 199, 204, 215
 - benefits* 205
 - configure project* 190
 - landscape* 204
- SAP Cloud SDK 209
- SAP Cloud Transport Management 95, 211, 261, 277, 284, 286
 - access* 214
 - CI/CD* 217
 - CI/CD pipeline* 289

- SAP Cloud Transport Management (Cont.)
 - CTS+* 286
 - enable* 212
 - how it works* 215
 - with CharM* 288
- SAP Code Vulnerability Analyzer 343, 360, 361, 363
 - features* 344
- SAP Community 377
- SAP Continuous Integration and Delivery 94, 153, 170, 171, 231, 273
 - enable* 177
 - job* 182, 191
 - job fields* 182
 - predefined pipelines* 172
 - roles* 177
 - source code repository* 178
 - source repository* 185
 - stages* 182
- SAP Customer Experience 267
- SAP Data Intelligence 302
 - CI/CD* 303
 - development tenant* 305
 - features* 303
 - Git and CI/CD integration* 306
 - internal transportation* 307
 - modeler* 304
 - monitoring* 308
 - system management command-line interface client* 304
 - tenant solutions* 305
 - test tenant* 305
 - user workspace* 304
- SAP Discovery Center 169
- SAP Enterprise Threat Detection 363
- SAP Fiori 172, 199, 274
 - application deployment* 252
 - application development* 250
 - application implementation* 251
 - scans* 353
- SAP Focused Run 280, 369
 - features* 280
- SAP GRC solutions 334
- SAP HANA 199
- SAP HANA transport for ABAP 104, 143
 - transport request* 143
- SAP Integration Suite 172
- SAP Road Map Explorer 377
- SAP S/4HANA 92, 97
 - DevOps features* 98
 - end-to-end scenario* 158
 - side-by-side extensions* 268

- SAP S/4HANA Cloud 267, 269
 - SAP Solution Manager ... 94, 154, 275, 279, 295, 334, 361, 363
 - cCTS* 139
 - CTS+ integration* 134
 - operation monitoring* 296
 - SAP Solution Manager Test Suite 151, 294
 - SAP SuccessFactors 269
 - extend* 270
 - SAP Test Automation by Tricentis 284
 - SAP_UI 7.53 component 274
 - Scalability 324
 - Scenario test 147
 - Secure code analytics 335
 - manual* 336
 - Security 333
 - best practices* 364
 - implementation* 365
 - tools and concepts* 333
 - vulnerability monitoring* 361
 - Security optimization self-service 360
 - Security testing 67
 - Selenium 68, 70
 - components* 68
 - Selenium Grid 69
 - Selenium IDE 69
 - Selenium Remote Control 69
 - Selenium WebDriver 69
 - Service mesh 370
 - Service registry 325
 - Services 48
 - Shared library 237
 - Shared responsibilities 47
 - Sharing 32
 - Shift left 329
 - SonarQube 73
 - benefits* 74
 - call flow* 73
 - steps* 75
 - Source code analysis 361
 - Specific, measurable, achievable, relevant, and time-bound (SMART) 159
 - Speed 32
 - Splunk 91
 - features* 91
 - SQL Monitor 227
 - Stages 34
 - Standardization 165
 - Static application security testing (SAST) ... 335
 - Subaccount 212
 - Synthetic user monitoring 281
 - System cluster 135, 138
 - create* 136
 - System tests 147
- ## T
- Test automation 67
 - Test Execution app 257
 - Test management 283, 293
 - Test phase 37, 160, 164, 196, 223, 309
 - Test Preparation app 256
 - Time series analytics 90
 - Tomcat 79
 - Tools 48, 55
 - build* 60
 - code and version control* 55
 - deployment* 76
 - monitoring* 88
 - SAP's DevOps portfolio* 92
 - test automation* 67
 - Transaction
 - SCI* 339
 - SCTS_HTA* 143
 - SEO9* 130
 - SE38* 106
 - SLIN* 343
 - SM59* 137
 - STMS* 127, 128
 - STMS_IMPORT* 144
 - SZENCONFIG* 136
 - Transformation mapping 291
 - Transport collection 135, 138
 - Transport domain 125
 - Transport export 261
 - Transport import 263
 - Transport layers 123
 - configure* 127
 - Transport Management System (TMS) 127
 - Transport nodes 215
 - Transport Organizer 101, 123, 130, 141
 - key features* 102
 - Transport request 117, 122
 - Transport roles 129
 - Transport route 215
 - create* 133
 - define* 126
- ## U
- Unit test 34, 119, 146
 - User acceptance test 34, 184
 - User experience monitoring 302

- ## V
- Version control 55, 314, 326
 - Virus scan interface 360
 - Visual Studio Code 304
 - Vulnerability scanning 347, 350, 362
- ## W
- Web services 131
 - Webhook 179, 190
 - configure* 180, 181
 - Workbench request 113

- ## Y
- YAML format 276
- ## Z
- Zero-downtime 322



Raja Gupta, Sandip Jha

DevOps with SAP

387 pages | 07/2023 | \$89.95 | ISBN 978-1-4932-2419-7

 www.sap-press.com/5694



Raja Gupta is a technology enthusiast working for SAP and is passionate about providing training and writing blog posts and books about SAP technologies. He has more than 14 years of experience in the IT industry and has led various implementation projects of SAP solutions for customers. In his current role as a solution architect, he is focused on working with SAP partners to co-innovate and build innovative solutions.

Raja is an expert in cloud technologies, SAP Business Technology Platform (SAP BTP), SAP Fiori, DevOps, and SAP HANA. He has conducted many trainings and workshops for SAP partners and customers. He is also an active member of the SAP community and a frequent blogger. You can find his blog posts at bit.ly/raja-gupta. You can connect with Raja on LinkedIn at <https://www.linkedin.com/in/raja-gupta>.



Sandip Jha is a development architect at SAP with more than a decade of SAP product development experience. Currently, his focus is on enabling partners to build innovative products in co-innovation mode. Previously, he contributed to and led product development for SAP MII and SAP Digital Manufacturing Cloud. He has also created technical training content and conducted workshops for customers and partners across the globe. You can connect with him at <https://www.linkedin.com/in/sandipjha/>.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.