

Integrating Db2 for z/OS Database Changes Into a CI/CD Pipeline

Maryela Weihrauch
Frank van der Wal
Rafael Toshio Saizaki
Kendrick Ren
Eric Radzinski
Hendrik Mynhardt
Benedict Holste
Maria Sueli Almeida



Information Management



IBM Redbooks

Integrating Db2 for z/OS Database Changes Into a CI/CD Pipeline

August 2021

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

First Edition (August 2021)

This edition applies to Db2 for z/OS Version 12, Function Level 500.

This document was created or updated on September 13, 2021.

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

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

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	x
Now you can become a published author, too!	xi
Comments welcome	xii
Stay connected to IBM Redbooks	xii
Chapter 1. Introduction	1
1.1 Stages of the software development lifecycle	2
1.2 Introduction to DevOps	3
Chapter 2. Db2 database code changes in the CI/CD pipeline	7
2.1 Requirements and pain points with the current database code change process	8
2.2 Typical database-related change use cases	9
2.3 Introducing Db2 DevOps Experience for integrating database code changes in the enterprise CI/CD pipeline	9
Chapter 3. Sample application overview	11
3.1 Introduction	12
3.2 About the GenApp application	12
3.2.1 Db2 resources for the GenApp application	12
Chapter 4. Architectural overview	15
4.1 High-level architecture	16
4.1.1 Steps 1, 2, and 3: Cloning, editing, and committing	16
4.1.2 Step 4: Automating the process with Jenkins	16
4.1.3 Step 5: Building the artifacts	17
4.1.4 Step 6: Storing the build output in a central repository	17
4.1.5 Steps 7 and 8: Moving the build output to the deployment stage	18
4.1.6 Step 9: Deploying the DDL into Db2 for z/OS by way of DOE	18
4.1.7 Step 10: Deploying the load modules in CICS and binding the DBRM	18
Chapter 5. Code repository organization	19
5.1 Code repository structure	20
5.1.1 Treat data object definitions as source code	20
5.1.2 Designing your code repository structure	21
5.1.3 Overall repository structure	21
5.1.4 Db2 schema structure	22
5.1.5 Application code structure	24
Chapter 6. Delivering database and application changes at the same speed	25
6.1 Db2 DevOps Experience architecture	27
6.2 Db2 DevOps Experience interactions with Db2 for z/OS	28
6.2.1 Using the browser-based user interface (Unified Experience for z/OS)	28
6.2.2 Scripting the REST APIs	29
6.3 DOE terms and concepts	30
6.3.1 Subsystems	30
6.3.2 Environments	30

6.3.3 Teams	30
6.3.4 Applications	31
6.3.5 Objects	31
6.3.6 Instances	31
6.4 DOE roles and responsibilities	31
6.4.1 Super administrators	32
6.4.2 Team administrators	33
6.4.3 Team members	33
6.4.4 DOE roles and RACF definitions	33
6.5 Establishing the landscape of Db2 subsystems, environments, and teams	35
6.5.1 Registering Db2 subsystems	35
6.5.2 Creating environments	38
6.5.3 Creating teams	40
6.5.4 Managing team members	40
6.6 Establishing the policy for database provisioning and schema changes	43
6.6.1 Registering applications	44
6.6.2 Provisioning an application instance	48
6.7 Defining Db2 site rules	49
6.7.1 Simple site rules	50
6.7.2 Complex site rules	51
6.8 DOE setup for DevOps engineer	52
6.8.1 Mapping DOE definitions to GitHub and UCD	52
Chapter 7. IBM UrbanCode Deploy	55
7.1 Introduction to IBM UrbanCode Deploy	56
7.2 Db2 DevOps Experience plug-in for UCD	58
7.2.1 DOE APIs used by the plug-in	59
7.2.2 Installing the DevOps Experience plug-in	62
7.2.3 Configuring UrbanCode Deploy	62
7.2.4 Creating roles and teams	63
7.2.5 Creating the component	64
7.2.6 Creating the component deployment process	68
7.2.7 Creating the resource	76
7.2.8 Creating the application	77
7.2.9 Associating the component with the application	78
7.2.10 Defining the process to install the component	79
7.2.11 Creating the environment for the application	81
7.2.12 Assigning the resources to the environment	82
7.2.13 Configuring notifications	83
Chapter 8. Jenkins automation server	85
8.1 Introduction	86
8.2 Resources for installing and configuring Jenkins	86
8.3 GenApp pipeline stages overview	87
8.3.1 GenApp pipeline stages	87
8.3.2 Extending the pipeline with Db2 for z/OS schema changes	87
8.3.3 Db2:Deployment stage	88
Chapter 9. Application development tools for z/OS	91
9.1 IBM Db2 for z/OS Developer Extension introduction	92
9.2 IBM Z Open Editor introduction	93
9.3 Installing Visual Studio Code	93
9.3.1 Installing Visual Studio Code extensions	94
9.4 Setting up a code repository in your development environment	95

9.4.1	Installing Git	95
9.4.2	Connecting to the code repository with SSH.	96
9.4.3	Cloning the code repository	96
9.5	Data modeling consideration.	97
Chapter 10. Application change execution and demonstration documentation.		99
10.1	GenApp application data structure	100
10.2	Use case	101
Chapter 11. Summary		107
11.1	Overview	108
11.2	Real example of pain points managing database changes.	109
Appendix A. Additional material		111
	Locating the GitHub material	111
	Cloning the GitHub material	111
Abbreviations and acronyms		113
Related publications		115
	Online resources	115
	Help from IBM	115

Notices

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

CICS®

Concert®

Db2®

DB2®

IBM®


IBM Z®

InfoSphere®

RACF®

Rational®

Redbooks®

Redbooks (logo) ®

UrbanCode®

z/OS®

The following terms are trademarks of other companies:

Zowe, are trademarks of the Linux Foundation.

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

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

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

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

Preface

Whether you are modernizing your existing applications or developing new ones, applying DevOps agile practices and tools makes the project run faster and more smoothly by automating key steps, improving operational efficiency, and increasing standardization.

When an organization decides to implement DevOps practices, they tend to focus primarily on implementing a CI/CD pipeline to handle application code changes. However, in practice, many application changes require database changes that also must be carefully considered in the context of that CI/CD pipeline.

Because changes to database schema are typically handled by database administrators (DBAs), application developers depend on DBAs to deploy code changes that typically require multiple hand-offs between the application developer, data modeler, and DBA. This process can result in unnecessary bottlenecks and deployment delays.

A well-engineered enterprise CI/CD pipeline supports changes to all application components, including database code changes on any platform, on-premises, or in a public cloud. It also must handle applications on IBM Z and implement any supported language (Java, COBOL, and so on) that store data within Db2® for IBM z/OS databases.

The goal of this IBM® Redbooks® publication is to demonstrate the ability to perform single click automated deployments of multi-platform applications that include IBM Db2 for z/OS database schema changes by using the capabilities of IBM Db2 DevOps Experience for z/OS.

Pushing the application and database code changes to a source control management system (SCM) triggers a single CI/CD pipeline execution for application and database changes. Therefore, it mitigates the dependency on the DBA to deploy those database changes in a separate process.

At the same time, DBAs can safeguard the integrity of their organization's data by implementing site rules in Db2 DevOps Experience. DBAs define whether a schema change can be approved automatically after all site rules are satisfied or whether it must be approved manually.

We provide an overview of the CI/CD pipeline architecture in the context of a sample application. The steps to set up a pipeline for the sample application are described in the tutorial *Build a pipeline with Jenkins, Dependency Based Build, and UrbanCode Deploy*, which is available [this web page](#). The document describes how to extend that CI/CD pipeline to support Db2 for z/OS® database schema changes.

We describe the steps that are relevant to the roles of the DevOps engineer who implements the enterprise CI/CD pipeline, the DBA who is responsible for database code changes in Db2 for z/OS and for defining site rules that ensure quality in production, and the application developer who changes the application code and communicates requirements for changes in the database schema.

Code samples for the demonstration application that is used in this Redpaper publication can be downloaded at [this web page](#).

For more information about downloading instructions, see Appendix A, “Additional material” on page 111.

Authors

This paper was produced by a team of specialists from around the world.

Maryela Weihrauch is an IBM Distinguished Engineer and worldwide IBM Z Data and AI Technical Sales and Customer Success Leader. She has extensive experience with Db2 in terms of systems, application, and database design. Maryela is engaged with enterprises worldwide, driving the adoption of new data and analytics technologies. Her most recent roles in Db2 for z/OS Development included shaping Db2 for z/OS strategy for Hybrid Transaction and Analytics Processing (HTAP), including the Db2 Analytics Accelerator strategy and implementation, and Db2's application enablement strategy. Maryela is a member of the IBM Academy of Technology and frequently shares her experience at conferences.

Frank van der Wal works in a pan-European team as Technical Leader in Data and AI on IBM Z. He also helps customers make the best of use of their data on Z in a hybrid multi-cloud world. He has contributed to IBM Redbooks publications and is a regular speaker at IBM Technical University, GSE, and customer events across Europe. He is an author and reviewer of blogs on the topic of Digital Transformation and Digital Acceleration on [the Medium.com](#).

Rafael Toshio Saizaki is an IBM Senior Client Technical Specialist for development tools on IBM Z. He has 15 years of experience in IBM on zAnalytics and zDevOps tools. He holds a bachelor's degree in System Analysis. Before joining IBM, he worked for three years at a financial company as a Db2 for z/OS system programmer.

Kendrick Ren is the Technical Lead of the DevOps solution for Db2 for z/OS. He has been leading the effort of transforming the user experience of application development and deployment and query performance tuning for Db2 for z/OS. Previously, he was an original inventor and lead architect of Optim Query Workload Tuner. He also has over a decade of experience in database and query performance tuning.

Eric Radzinski is a technical writer and editor who is based at IBM's Silicon Valley Lab. He has spent his career documenting IBM Z database technology, including Db2 for z/OS, Db2 Tools, and IMS. He is a co-author of *The IBM Style Guide and Developing Quality Technical Information: A Handbook for Writers and Editors*. He is currently working on projects that are related to transforming IBM Z applications for today's modern application development environments.

Hendrik (Hennie) Mynhardt is an Executive IT Specialist at IBM. He has led and worked on various technical projects for database customers in the US and overseas. His special interests are systems performance tuning and backup recovery for Db2 for z/OS. He currently provides technical consulting and pre- and post-sales support for Db2 and related technology for the zAnalytics space. Hennie co-authored *Securing and Auditing Data on DB2 for z/OS*, *Optimizing Restore and Recovery Solutions with DB2 Recovery Expert for z/OS*, *DB2 Recovery Expert for z/OS User Scenarios*, *DB2 9 for z/OS and Storage Management*, and *Modernize Your IBM DB2 for IBM z/OS Maintenance with Utility Autonomics and Db2 Utilities in Practice*.

Benedict Holste joined IBM® in 2011 as a cooperative student and graduated in 2014 with a Bachelor of Science in Applied Computer Science. Since then, he has worked as a Technical Specialist for Data, AI, and Automation solutions on IBM Z® with customers in Germany. Benedict is skilled in Db2 for z/OS engine and tools, data replication, and Machine Learning on IBM Z. Over the last year, he focused on Db2 for z/OS DevOps integration and contributed to the demonstration that is described in this paper.

Maria Sueli Almeida is an IBM Certified Thought Leader and member of Db2 for z/OS development at the IBM Silicon Valley Lab in San Jose, California. She has extensive experience with Db2, including database administration and systems programming. She has worked on Db2 for z/OS advanced technology and solution integration. Currently, she is engaged on Db2 for z/OS support for user experience transformation, including DevOps adoption and cloud provisioning Database-as-a-Services for DBAs, application developers, and sysprogs. Sueli also consults with Db2 customers worldwide.

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

Patrick Bossman
IBM USA

Nayer Najafi
IBM Canada

Jean-Yves Baudy
IBM France

This project was managed by:

Vasfi Gucer
IBM Redbooks®, Austin Center

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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



Introduction

Any discussion of the software development lifecycle (SDLC) begins with a definition of the process, which we provide in this chapter along with an overview of the latest evolution of SDLC: the DevOps methodology and the framework and tools it provides for rapid software development and delivery.

This chapter includes the following topics:

- ▶ 1.1, “Stages of the software development lifecycle” on page 2
- ▶ 1.2, “Introduction to DevOps” on page 3

1.1 Stages of the software development lifecycle

The SDLC was developed as a formalized methodology and framework for building information systems. It includes all aspects of planning, creating, testing, and deploying an application.

Typically, the SDLC includes the following distinct stages:

1. Investigation

During this initial stage, current business priorities and how they are implemented and managed are examined. A feasibility study is normally conducted to determine whether creating an application or improving an existing application is a viable solution to meet the needs of the business. This step is typically performed by business analysts and application architects.

2. Analysis

The objective of this stage is to identify potential areas for improvement as they relate to the application system, if applicable. This stage involves breaking down the application into different pieces to facilitate analysis of the requirement, defining project goals, identifying changes and improvements, and enlisting users to help define and validate specific requirements. Business analysts, application architects, developers, and users typically participate in this stage.

3. Design

The initial input to the design stage is a formally approved business requirements document. For each requirement in this document, one or more design elements are produced as a result of a combination of interviews, workshops, and prototyping efforts. This stage typically involves business owners, application architects, and developers. Depending on the scope of the requirements, application database administrators (DBAs) and data modelers might also be involved.

4. Environments

This stage involves defining and setting up the controlled environment that application developers use to build, distribute, install, configure, test, and run systems that move through the SDLC. Application designers, developers, and potentially system programmers, capacity planners, and application DBAs typically participate in this step.

5. Testing

During this critical phase, the application code is developed, tested, and refined. Throughout the industry, many different opinions exist as to what the stages of testing are and how much iteration, if any, should occur. However, it is widely accepted that this stage should include some level of unit testing, system testing, and user acceptance testing. Many different roles might need to participate during various phases of this stage, but the primary participants are application developers, software quality assurance engineers, and DBAs who must work together closely to achieve a common goal.

6. Training and transition

After the application is stabilized as a result of thorough testing, the SDLC ensures that proper training on the system occurs and instructions for installing, configuring, and using the system are documented before the system is transitioned to its support staff and users. This stage typically involves everyone who has participated in the previous stages. From a DBA's perspective, this stage is where performance, backup and recovery, and other DBA-related tasks are refined, agreed upon, and communicated to ensure that the Db2 constructs and data are maintained in a manner that is required to achieve the SLA of the application.

7. Operations and maintenance

Because the deployment of the application includes changes and enhancements to existing processes, the effects of these changes must be clearly understood by members of the development, database administration, operations, capacity, and performance management teams. As with the training and transition stage, this stage is critical to the DBA.

1.2 Introduction to DevOps

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) into a cohesive methodology for developing and delivering high-quality software rapidly and efficiently. It gained popularity over the last decade as the primary method for rapidly developing and releasing software.

One of the most important practices for accomplishing these goals is creating a CI/CD pipeline. A CI/CD pipeline is a collection of processes and automation tools that enable cross-functional teams to meet the growing demand for frequent releases and updates by consolidating build, test, and deployment capabilities into a single framework.

Figure 1-1 shows the CI/CD pipeline.

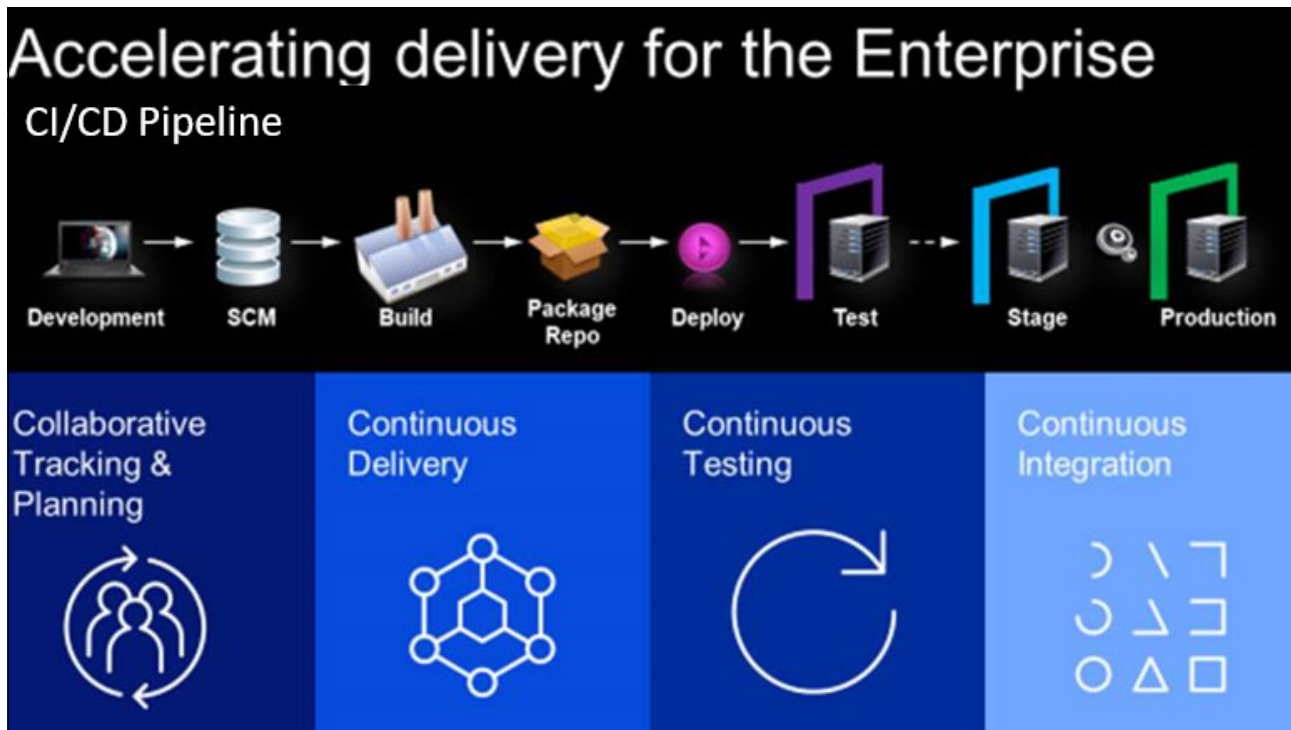


Figure 1-1 CI/CD pipeline

Pipelines are the mechanism that DevOps teams rely on to implement the practices of continuous integration/continuous delivery (CI/CD), continuous testing, and collaborative tracking and planning to improve the efficiency and effectiveness of software development.

CI/CD is a practice for developing software with the focus on delivering updates to consumers on a continuous basis.

Continuous testing is a practice that provides test cases with code changes and automatically runs them as part of the pipeline execution to maintain a high level of code quality, even with frequent code changes.

Collaborative tracking and planning are critical for providing a productive environment within an organization.

A typical CI/CD pipeline includes the following high-level steps:

1. Code, preferably developed in an integrated development environment (IDE), is committed to a source control management system (SCM), which starts the CI/CD pipeline.
2. The application is compiled and built by the CI/CD tools and deployed to the test environment.
3. Regression and unit tests are performed automatically as part of the pipeline execution.
4. If tests are passed, the application is delivered and deployed into the integration and user-acceptance environments for more tests.
5. The application is delivered and deployed into production.

Ideally, pipelines enable development teams to automate most or all of the steps from code integration to deployment. These steps can be applied to any software development project.

As shown in Figure 1-2, over many years a myriad of processes to deploy application changes on IBM Z (also called the mainframe) were available. However, in most cases, they are not fully automated and still follow the waterfall development model in which many changes are combined and rolled out in releases only a few times each year.

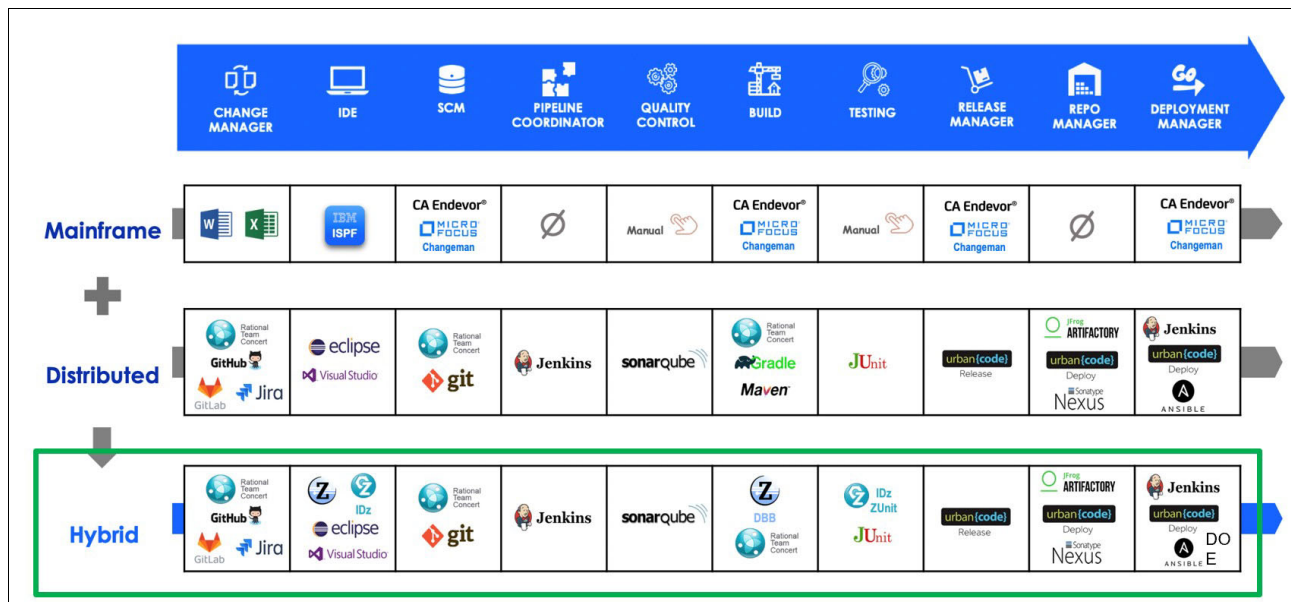


Figure 1-2 Example of tools for implementing an enterprise CI/CD pipeline

Driven by the requirement to deliver code changes to many environments, a new methodology that involved a different set of tools that evolved on the distributed platform. This methodology focused on the full integration and automation of the tools to support an automated deployment pipeline. Many of those tools are available through open source projects and are widely adopted.

The following tools are commonly used in the different stages of a CI/CD pipeline:

- ▶ Integrated Developer Environment (IDE): Eclipse-based IDEs are widely used for the development of cross-platform applications. Microsoft Visual Studio Code (VS Code) with available extensions became popular for intelligent development of applications in a wide range of languages, including established languages (such as COBOL, Java, and PLI), and newer languages (such as Python, JavaScript, and Go), that support accessing Db2 for z/OS by way of SQL or Db2 RESTful services.
- ▶ Source Control Management (SCM): Git and IBM Rational® Team Concert® (RTC) are popular tools for source control management, including database artifacts. Whereas RTC is used to manage application artifacts, many organizations also use Git as a common SCM to manage database artifacts. For more information about best practices for managing database schemas in Git, see Chapter 5, “Code repository organization” on page 19.
- ▶ Pipeline orchestration: Jenkins or GitLab are widely adopted tools to orchestrate the different steps in a pipeline, including functions to easily see the history of pipeline executions.
- ▶ Build: IBM Dependency Based Build (DBB) is a widely adopted tool for building applications that run cross-platform, including IBM Z. Built application components, such as load libraries and DBRMs, are stored in Artifactory, which are ready to be deployed to the different environments by using a deployment tool.
- ▶ Deploy: IBM UrbanCode® Deploy (UCD) is a widely adopted tool for deploying application artifacts to all platforms, including on-premises, cloud, and mobile applications. Alternatively, RedHat Ansible also became a popular tool for deployment automation of cloud-native applications. A deployment tool can substitute environment-specific definitions, such as library names or, in the context of a Db2, schema names, to fit the target environment.
- ▶ Testing and quality control: Recently, many organizations focused more on automated testing in the pipeline. SonarQube is used for the continuous inspection of code quality, and JUnit is used for component testing. Both of these tools are open source.

As customer applications become multi-platform, organizations are looking to adopt a single, multi-platform (hybrid cloud) automated CI/CD pipeline.

The steps in the CI/CD pipeline are typically associated with application code changes. In reality, many application changes also require database code changes, such as adding columns to a table, adding an index, or changing a database stored procedure.

Thus far, database changes are controlled and run by DBAs who work independently of the pipeline. DBAs control this process and enforce rules to maintain the quality of a database as a highly shared asset.



Db2 database code changes in the CI/CD pipeline

Many companies have a process in place in which database code (schema) change requirements are coordinated with the data modeler, who then designs the logical and physical data model before those changes are implemented by the database administrators (DBAs) in the different environments.

This chapter describes the requirements and pain points that are associated with the current approach of implementing the database code changes from the perspective of application developers and DBAs. The data modeler still designs the logical and physical data model and provides this input to the DBA.

It also presents some typical use cases that show how integrating those database code changes with the application changes in the enterprise CI/CD pipeline can address those requirements and pain points. Lastly, it explains the benefits that Db2 DevOps Experience can provide to application developers and DBAs in the context of this process.

This chapter includes the following topics:

- ▶ 2.1, “Requirements and pain points with the current database code change process” on page 8
- ▶ 2.2, “Typical database-related change use cases” on page 9
- ▶ 2.3, “Introducing Db2 DevOps Experience for integrating database code changes in the enterprise CI/CD pipeline” on page 9

2.1 Requirements and pain points with the current database code change process

DBAs traditionally were (and still are) the trusted gatekeepers of their companies' data. They are responsible for data availability (backup and recovery), data quality (integrity), and speed of access (performance and scalability) to enterprise data.

When a problem occurs with the database access in the production environment, they are the first ones who are called, day or night to identify the cause of the problem and to solve it. By learning from these types of events over time, companies establish a “book” of rules to minimize the chances of these problems reoccurring. When changes are proposed that violate these rules, DBAs must reject them and encourage other personas to find a better way to achieve their goals. Because of the critical nature of their role, DBAs must be meticulous and diligent when considering database changes, which to other personas is sometimes viewed as being overly cautious and bureaucratic.

Based on our cumulative years of working closely with DBAs, we know that in most organizations, database change requests are submitted to DBAs through ticket systems that DBAs must constantly monitor. These requests result in many, mostly simple, database schema change requests every day.

Although most requests are for simple changes, this repetitive process consumes most of a DBAs time, leaving them little or no time for higher-value projects, such as performance evaluation, application tuning, and security-related efforts. As with any process that involves manual changes, the possibility always exists of introducing errors, which because ever-shrinking maintenance windows, are becoming increasingly intolerable. Enforcing standardization is difficult when DBAs develop their individual way to run the requests.

Application developers are dependent on DBAs when they make any application code change that requires a corresponding database code change. They typically see the database code change as a separate activity to their application development efforts, and are frustrated by having to wait for dependent database changes (DDL) to be integrated before their changes can be approved and deployed into the next environment, which potentially jeopardizes the target dates to which they committed.

Also, many conversations between the application developer, the DBA, and the data modeler often are needed to finalize the change before it can be deployed. This back-and-forth communication is frequently repeated for each environment, which magnifies the resulting disruption of continuous flow of change integrations.

A major benefit of DevOps is that it eliminates many of these inefficiencies. However, DevOps includes a learning curve for the traditional DBA role to work through. It also presents a large culture change to the established way of doing things.

2.2 Typical database-related change use cases

From our many conversations with DBAs, we know that it is common for an application change to require an ALTER TABLE ADD COLUMN schema change, which is why we chose this use case to illustrate how to implement it in a CI/CD pipeline. You can apply the same process to other database artifacts, such as views, triggers, and stored procedures.

Other common use cases focus on automated testing of application changes that access data stored in Db2. We briefly explain them next but do not cover them in detail in this document.

The simple use case is the automated testing of an application change that destroys Db2 test data. This use case requires the test data to be reloaded as part of the pipeline execution so that the test can be run again.

An evolution of this use case is an automated test of an application change with a database code change. In this use case, the database schema change must be deployed before masked and pre-generated test data can be loaded.

Lastly, validating the performance of SQL in a test environment in preparation for deploying it to the production environment is a popular use case with DBAs. Performance validation is simpler with static SQL because the SQL can be extracted from the Db2 packages and access paths can be analyzed. Dynamic SQL requires running a test case to expose the SQL statement before access paths can be captured and evaluated following the evaluation process for static SQL.

2.3 Introducing Db2 DevOps Experience for integrating database code changes in the enterprise CI/CD pipeline

The IBM Db2 DevOps Experience (DOE) delivers Db2 for z/OS DBaaS as an integrated set of features that elevates the developer and liberates the administrator. This technology enables you to bridge the gap between the application developer and DBA personas.

How does DOE affect the following personas:

- Application developer

DOE supports self-service, on-demand database operations so that the developer's sprint can proceed without interruption and without waiting for the DBA to make the database schema changes. Although DBAs retain the ability to approve and control the changes, they can decide whether approval can be automated based on defined rules or if a manual approval is needed.

A developer can now work with the database code (DDL) as with the application code; they can create a set of test objects, make modifications that are automatically verified by coded rules (site rules) in DOE, and perform testing. When the database modifications are ready to be integrated, an approval process is started for proper review, acceptance, and merging of changes.

- DBA

The Db2 DBA controls the adoption of Db2 schema changes by creating DOE applications (a collection of Db2 objects that are provisioned together for this application), and defining site rules to which the schema changes must adhere.

The database superuser can define the environments (development, integration, test, and so on) in which provisioning operations occur and can define which teams can use these environments.

In large environments where a DBA might be responsible for dozens or even hundreds of subsystems, the use of DOE APIs can radically increase efficiency and reduce the potential for errors.

DBAs can automate the deployment of fully reviewed and approved schema changes to all of the environments that they are responsible for, whereas before, this effort was largely manual, repetitive, and time-consuming and created the potential for introducing a whole host of problems, such as columns being added in different sequences, the introduction of typographical errors, and deploying a change to the wrong environment. By using common environment definitions, the chance of introducing these types of error is significantly reduced.

- DevOps Engineer

The DevOps engineer works jointly with the database superuser to integrate the defined DOE artifacts into the pipeline. Schema (DDL) changes must be maintained in an SCM (for example, Git) that can trigger the pipeline execution in case of push changes. DOE provides over 100 REST APIs to make all administration and operational services available for integration into the pipeline orchestration and deployment tools.

Implementing an enterprise CI/CD pipeline successfully requires DBAs to fully participate in the DevOps project and to be open to reviewing processes for changing database code. The DBA must work with the enterprise DevOps team to learn how to make full use of the pipeline automation tools to streamline database code deployments through automation.

A successful DevOps implementation shifts the DBA role from deployment to development, which frees them up to provide more value-added services to the organization. It is the foundation for enabling self-service deployments as part of the standard process.



Sample application overview

This chapter describes the sample application that we use in this document and includes the following topics:

- ▶ 3.1, “Introduction” on page 12
- ▶ 3.2, “About the GenApp application” on page 12

3.1 Introduction

The examples in this document are based on a sample COBOL application that is running in an IBM CICS® Transaction Server (CICS TS). This application is called the *General Insurance application* (GenApp). We chose this application because it is provided with CICS TS, which means that it is available to all CICS users. We encourage you to use this application with the information in this document to help you explore the capabilities of IBM Db2 DevOps Experience for z/OS.

The primary use case in this document is straightforward: we need to update the GenApp application to include a country code for our customers' telephone numbers.

To do so, we must change the COBOL code that makes up the GenApp application to accept another field in the data structure, and we must add a column to the Db2 schema. We demonstrate how a developer can change the related COBOL and DDL files and how the changes are picked up by our CI/CD pipeline. We focus on the Db2 DDL changes and the process that these changes go through, up to and including deployment.

3.2 About the GenApp application

The GenApp application simulates transactions that are made by an insurance company to create and manage its customers and their insurance policies. The application provides sample data and a 3270 interface for creating and querying customer and policy information. However, the information in this document does not use the 3270 interface; instead, it demonstrates how to call the business logic by using RESTful APIs.

Because the application is designed to simulate the flow of an application, some aspects of the application architecture intentionally do not adhere to coding best practices. However, the application is extended to demonstrate other ways of accessing and transforming traditional applications that are based on best practices.

The GenApp application runs in a single CICS region. It writes to a VSAM file and to Db2 for z/OS. As you use the application to explore different features of CICS, the configuration of the application changes to include more components. This document focuses on demonstrating how the database schema change can move through the same pipeline as the related application change by using Db2 DevOps Experience capabilities.

For more information about the GenApp application and to download the application, see [this web page](#).

3.2.1 Db2 resources for the GenApp application

The database for the GenApp application contains details about customers and their insurance policies. The set of Db2 objects for this application belongs to one logical Db2 database, and all objects are under the same schema. These objects include the following components:

- ▶ A customer table that lists all the customer records, including the customer number.
- ▶ A policy table that lists all the policies, including the customer number, policy number, and policy type.
- ▶ A policy table for each type of policy: commercial, endowment, house, and motor insurance policies.

This document focuses on the customer table and the COBOL logic that are needed to create another field in this table, and progresses through the following steps:

1. The data structure of the COBOL program and the link into the Db2 for z/OS schema are introduced.
2. A field is added in the COBOL data structure that must be reflected in the Db2 schema.
3. A column is added that does not conform to site rules to demonstrate how the pipeline handles these types of errors.
4. The error is corrected and how the changes are processed and deployed is demonstrated.

For more information about the COBOL code and the Db2 tables that are involved, see Chapter 10, “Application change execution and demonstration documentation” on page 99.



Architectural overview

This chapter discusses the high-level architecture of the setup that is used in this document.

4.1 High-level architecture

Figure 4-1 shows a high-level architectural overview of the setup that we use throughout this document.

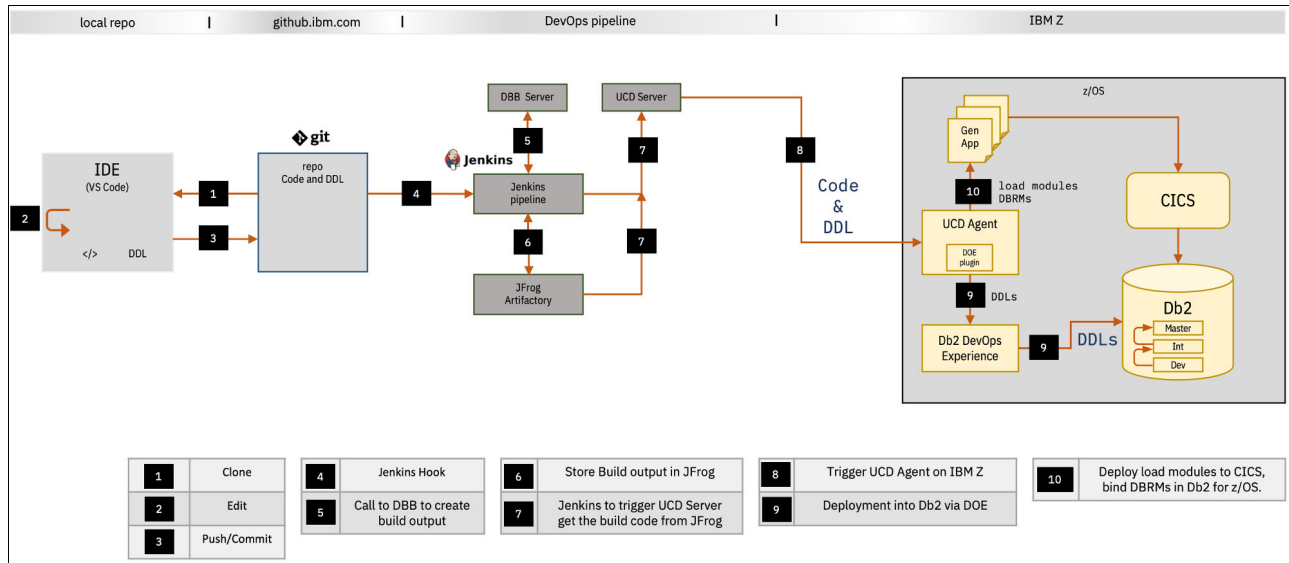


Figure 4-1 High-level architectural overview of the setup that is used in this document

4.1.1 Steps 1, 2, and 3: Cloning, editing, and committing

We start with the central repository. We use Git technology for our source control management by using GitHub Enterprise Edition. For more information about the structure of our source control management system, see Chapter 5, “Code repository organization” on page 19.

The SCM system is where all the artifacts for this project are stored.

The first time a developer must change the code or DDL, they start by cloning the repository to a local version, as shown in Step 1 in Figure 4-1. Editing is done on a local integrated development environment (IDE). We use Microsoft Visual Studio Code (step 2), which interfaces with GitHub. After the developer creates a local version of the project, they can work locally and then save and commit their changes to the GitHub Enterprise environment (step 3).

4.1.2 Step 4: Automating the process with Jenkins

The role of Jenkins is to orchestrate the pipeline, which means that it automates all of the different steps starting with picking up the changes in the artifacts all the way to deploying these changes or, in the case of an error, rejecting them.

In this step, Jenkins monitors the file system of the GitHub environment. Whenever a change is committed, Jenkins is triggered and the pipeline starts.

Figure 4-2 shows how a committed change moves through all the steps in our Jenkins pipeline.

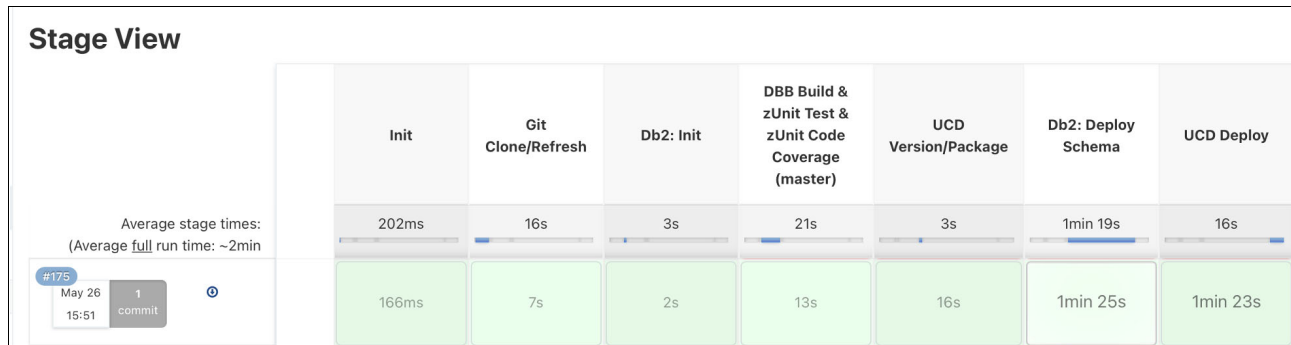


Figure 4-2 Stage View window

4.1.3 Step 5: Building the artifacts

Jenkins triggers the Dependency Based Build (DBB) server to compile and link edit the source code and to generate the Database Request Module (DBRM).

DBB is an IBM tool that allows the compiling, link-editing, and deployment of applications on IBM Z by using modern scripting languages. It is useful in the orchestration process that we use throughout this document. It can run Groovy on z/OS, which allows engineers who are not familiar with IBM Z to compile and link-edit programs and also to run MVS, TSO, and ISPF commands.

IBM DBB stores code dependencies and provides its own reporting tool. As an alternative for deeper analysis from the z/OS perspective, it is possible to write SMF records when the script calls z/OS commands.

A more detailed overview of DBB is beyond the scope of this document. For more information about setting up DBB, see [this web page](#).

Note: For our use case, we recognize that the changes that we make to the SQL also change the DBRM. However, because our changes do not cause a change in the build process, we do not cover the DBRM in relation to DBB in this document.

If the build fails, Jenkins reports the failure in the overall Stage View and notifies the suitable users.

4.1.4 Step 6: Storing the build output in a central repository

When a build completes successfully, its output is stored in the JFrog Artifactory, which is a single source of truth for all artifacts as they move across the entire CI/CD pipeline.

4.1.5 Steps 7 and 8: Moving the build output to the deployment stage

When JFrog received all the artifacts, Jenkins triggers the IBM UrbanCode Deploy (UCD) Server to pick up the build output and transfer it to the UCD Agent that is running on IBM Z. UCD standardizes and simplifies the process of deploying software components to each environment in your development cycle.

For more information about UCD, including how to set it up and how it integrates with the other components (specifically DevOps Experience for Db2 for z/OS), see Chapter 7, “IBM UrbanCode Deploy” on page 55.

4.1.6 Step 9: Deploying the DDL into Db2 for z/OS by way of DOE

In this step, the UCD Agent triggers DOE to have the DDL deployed into Db2 for z/OS. If this step completes successfully, we can continue with the final step.

4.1.7 Step 10: Deploying the load modules in CICS and binding the DBRM

Apart from triggering DOE to deploy the DDL as described in step 9, the UCD Agent also deploys the newly compiled and built load modules into CICS. A third and final task for the UCD Agent is to bind the DBRM in Db2 for z/OS.



Code repository organization

This chapter describes the code repository structure to store the artifacts that are used in the demo in this document.

5.1 Code repository structure

Most organizations implemented change management processes to ensure that changes to a product or system are introduced in a controlled and coordinated manner. As a best practice, we recommend the use of a source control management system to persist and keep track of the changes to all types of artifacts, such as application source code files, data definition language (DDL) files, Jenkins pipeline scripts, and environment properties files, in the change management process.

A source control management system (SCM) or version control system tracks changes in source code and other files during the software development process. An SCM enables you to retrieve any of the previous versions of the original source code and the changes that were stored.

Many SCM systems are available today. We chose to use GitHub Enterprise as the SCM in the solution and in the demo that is described in this document.

GitHub is based on Git, which is a free and open source distributed version control system that is designed to track changes in any set of files. It is typically used for coordinating work among programmers who collaboratively develop source code during the software development process.

GitHub is a provider of Internet hosting for software development and version control. GitHub offers the distributed SCM functions of Git plus its own features, such as access control and a robust set of collaboration features that include bug tracking, feature requests, task management, continuous integration, and wikis for every project. GitHub Enterprise is similar to GitHub but is designed for large-scale projects and is deployed behind a company's firewall.

Although we use GitHub Enterprise, the solution and the demo can be adapted to use any other SCM software. For example, you can use Rational Team Concert (RTC), Subversion, and similar products, or you can use other Git providers, such as GitLab and Bitbucket. The only requirement is that the SCM software that you use must have the interfaces available to integrate with pipeline orchestration so that a code change event triggers the pipeline automation.

5.1.1 Treat data object definitions as source code

Data Definition Language (DDL) is a subset of Structured Query Language (SQL). It is a syntax for creating and modifying database objects (tables, indexes, views, and so on). DDL statements are similar to a computer programming language for defining data structures, especially database schemas.

When you store DDL statements in the SCM, you can easily track any changes to the data object definitions and manage the changes and different versions or variations in the deployment process, just as you might do for the source code of other programming languages.

If your organization has not done so yet, we strongly suggest that you start to persist your database schemas in the SCM.

5.1.2 Designing your code repository structure

When you design your code repository structure, you typically must consider the programming languages, frameworks, deployment needs, code separation, ownership and responsibilities, and potentially many other factors that are based on the needs of your organization. You can choose to keep everything in a single repository or in multiple repositories based on your organization's situation.

For the purposes of this document, we chose to keep the database schema or DDL statements, deployment pipeline, configuration files, and the application source code in a single repository. However, the solution and demo in this document can be adapted to use a multiple-repository structure, such as storing the database schema in one repository and the application source code in another repository.

To adapt the solution to work with multiple repositories, you must customize the Jenkins pipeline to monitor and extract the artifacts from different repositories.

5.1.3 Overall repository structure

Figure 5-1 on page 22 shows the code repository structure to store the artifacts that are used in the demo in this document. As you can see, all artifacts are structured in folders: the Jenkins pipeline scripts file is stored in the `Jenkins` folder, the API artifacts in the `api` folder, and so on.

📁 .github	Update CODEOWNERS	3 months ago
📁 .vscode	Add VSCode User Build config file.	5 months ago
📁 Jenkins	Change Jenkins agents	2 months ago
📁 api	Update of API	2 years ago
📁 application-conf	Integer zUnit in "on premise" pipeline	5 months ago
📁 base	restore column	6 days ago
📁 db2-nazare-app	test db2	13 months ago
📁 db2	restore column	6 days ago
📁 doc	Move documentation to correct location.	4 years ago
📁 licence	Move licence files to common location.	4 years ago
📁 lite	Move documentation to correct location.	4 years ago
📁 policy-cloud	Project reorg (#19)	4 years ago
📁 policy-osgi	Project reorg (#19)	4 years ago
📁 projects	test 3	14 months ago
📁 tests	Add zUnit config files in nazare-demo-genapp.	2 years ago
📁 video	Move videos to correct location.	4 years ago
📁 zUnit	Integer zUnit in "on premise" pipeline	5 months ago
📄 .gitattributes	Integer zUnit in "on premise" pipeline	5 months ago
📄 .gitignore	Add VSCode User Build config file.	5 months ago
📄 .project	added logic/business rule to override expiry date	2 years ago
📄 README.md	Update README.md	2 years ago
📄 zdntinstall.md	Merge pull request #13 from IBMZSoftware/fultonm-patch-3	16 months ago

Figure 5-1 Code repository structure to store the artifacts that are used in the demo in this document

5.1.4 Db2 schema structure

The database schema is stored in the db2 folder, which contains a few subfolders for the different types of data objects, as shown in Figure 5-2. For example, TB contains the DDL statements for the tables, SP contains the DDL statements for the stored procedures, IX contains the DDL statements for the indexes, and so on.

..		
📁 DB	GENAtoGENC	4 months ago
📁 IX	Add index on CLAIM tables	4 months ago
📁 SP	GENAtoGENC	4 months ago
📁 TB	restore column	6 days ago
📁 TS	added ts09 for customer_secure	4 months ago

Figure 5-2 Db2 schema structure

Each table definition is stored in one .sql file in the TB subfolder, as shown in Figure 5-3. The same approach applies to other types of data objects.










..		
 CLAIM.sql	GENAtoGENC	4 months ago
 COMMERCIAL.sql	GENAtoGENC	4 months ago
 CUSTOMER.sql	restore column	28 minutes ago
 CUSTOMER_SECURE.sql	added ts09 for customer_secure	4 months ago
 ENDOWMENT.sql	GENAtoGENC	4 months ago
 HOUSE.sql	GENAtoGENC	4 months ago
 MOTOR.sql	GENAtoGENC	4 months ago
 PET.sql	Reset PET table to baseline	3 months ago
 POLICY.sql	GENAtoGENC	4 months ago

Figure 5-3 Organizing the data definitions at the data object level

Although you can store all of your data definitions for all data objects in one large DDL file, we recommend organizing the definitions at the data object level. This approach makes it easier to change the definition for a few data objects because you do not need to open and edit a huge DDL file. It also helps eliminate potential merge conflicts when multiple people must modify the definitions for different objects.

5.1.5 Application code structure

The application code (COBOL files in this case) is stored in the `base/src` folder, which contains a few subfolders to store files that are related to each other. For example, `cobol` contains all COBOL source code files, `copy` contains all COBOL copybook files, `testcases` contains test case files, and so on (see Figure 5-4).

..		
📁 bms	initial work (not complete) to enable build/packaging of cics-genapp ...	2 years ago
📁 cf	first work (incomplete) on coupling facility config	2 years ago
📁 cics	missed 2 files: the CICS started task proc and the corresponding SYSIN	2 years ago
📁 cobol	restore column	6 days ago
📁 copy	restore column	6 days ago
📁 input	initial work (not complete) to enable build/packaging of cics-genapp ...	2 years ago
📁 testcases/LGIPOL01	added logic/business rule to override expiry date	2 years ago
📄 .project	test v2	15 months ago

Figure 5-4 Application code structure

The samples in this chapter provide you with an example of how different types of artifacts are organized in the repository; however, you can and should design the structure based on your specific situation and plans.

After you settle on the structure, branching strategy, and continuous integration and continuous deployment process, you can start to build your pipeline. For more information about setting up a pipeline, see Chapter 8, “Jenkins automation server” on page 85.



Delivering database and application changes at the same speed

As described in Chapter 2, “Db2 database code changes in the CI/CD pipeline” on page 7, many of the leading businesses that began their digital transformation journey rely on IBM Z for running their mission-critical applications and on Db2 for z/OS for storing one of their most valuable assets: their core business data. Therefore, it is imperative that IBM continues to respond their business needs in a timely manner to help them grow and win, regardless of the industry in which they operate.

Digital transformation demands that businesses can respond quickly and efficiently to changes in their industry. Responding to an ever-evolving business environment often results in application changes that must be implemented, tested, and deployed quickly with little to no downtime.

In many cases, application code changes also require changes to the database schema, such as adding or changing indexes to support new access patterns and changing database application code, such as stored procedures, user-defined functions, and Db2 for z/OS native REST services. This dependency requires that database code changes move at the same speed as application code changes as work progresses through a CI/CD pipeline.

Db2 DevOps Experience for z/OS (DOE) provides the essential capabilities that enable an organization to meet these demands, as described in the following examples:

- ▶ Application developers can self-provision Db2 for z/OS schemas without waiting for IT or requiring mainframe knowledge, while keeping all necessary security and control in the hands of the IT department.
- ▶ Database administrators can define a policy that integrates automated deployment of database code changes into a CI/CD pipeline right alongside the corresponding application code changes.
- ▶ DevOps engineers can automate the mainframe application delivery pipeline to include database code changes, which reduces risk, cost, and complexity while improving responsiveness to changing market and customer needs.

This chapter includes the following topics:

- ▶ 6.1, “Db2 DevOps Experience architecture” on page 27
- ▶ 6.2, “Db2 DevOps Experience interactions with Db2 for z/OS” on page 28
- ▶ 6.3, “DOE terms and concepts” on page 30
- ▶ 6.4, “DOE roles and responsibilities” on page 31
- ▶ 6.5, “Establishing the landscape of Db2 subsystems, environments, and teams” on page 35
- ▶ 6.6, “Establishing the policy for database provisioning and schema changes” on page 43
- ▶ 6.7, “Defining Db2 site rules” on page 49
- ▶ 6.8, “DOE setup for DevOps engineer” on page 52

6.1 Db2 DevOps Experience architecture

Db2 DevOps Experience is built on top of an IBM Unified Management Server (UMS), which is the foundation that provides the common functions (such as base metadata and core services) that are needed for database management.

Db2 DevOps Experience uses a single interface that is called the IBM Unified Experience for z/OS. This browser-based user interface is built on top of the open source Zowe Virtual Desktop and provides you with the ability to interact with Db2 assets based on the task that you want to accomplish. This interaction with Db2 is supported by way of REST APIs, which interacts with Db2 in a modern, efficient, and flexible way (see Figure 6-1).

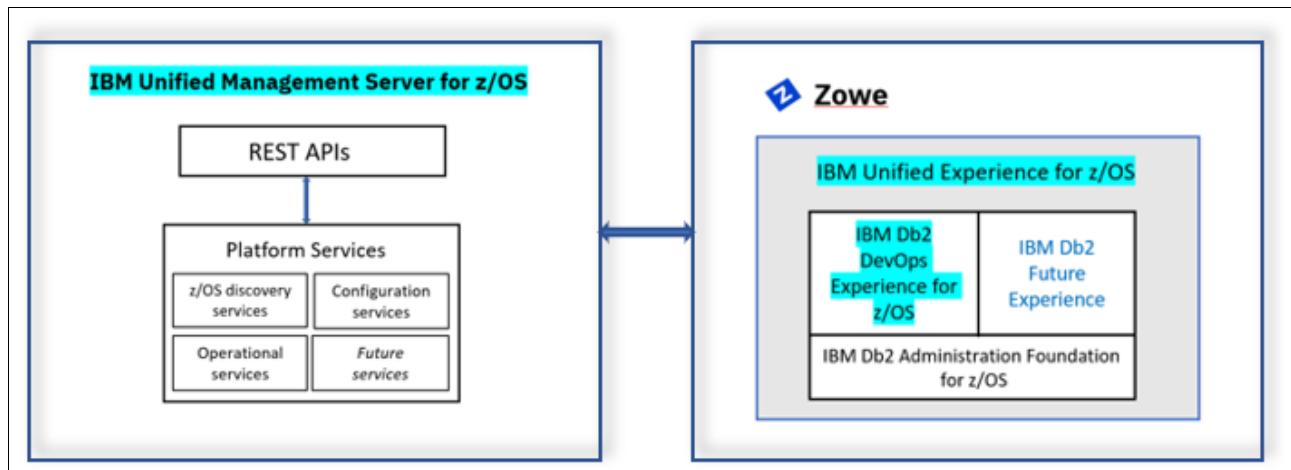


Figure 6-1 Db2 DevOps Experience architecture

One of the key benefits of Db2 DevOps Experience enables application developers to be self-sufficient with tasks, such as on-demand provisioning of Db2 modifications while DBAs retain a high level of control.

For example, Db2 DBAs can control which team members are authorized to perform which actions and how those actions must be performed (for example, only registered team members can create a table, and the table name must conform to a predefined schema naming convention).

Application developers can be self-sufficient when they need to change the database for their development or testing environment. The DBA no longer must spend time responding to and implementing database change requests and therefore, has more time for other tasks while still retaining the oversight and control that is needed to guarantee the service levels and standards for which they are responsible. The application developer no longer must wait for the DBA to approve and change the database and therefore, is more productive and efficient.

6.2 Db2 DevOps Experience interactions with Db2 for z/OS

All services and capabilities that application developers and DBAs use to interact with Db2 are handled by a series of REST APIs. These services and capabilities can be used through the browser-based interface or by scripting the REST APIs.

6.2.1 Using the browser-based user interface (Unified Experience for z/OS)

By using the **Explore** → **Instances** feature of DOE, as shown in Figure 6-2, application developers and DBAs can provision and de-provision database objects with or without data, which simplifies the process of making and testing application code changes. If a change requires a database change, the developer also can make stateful changes and evaluate them on their own instance before merging the changes into the CI/CD environment.

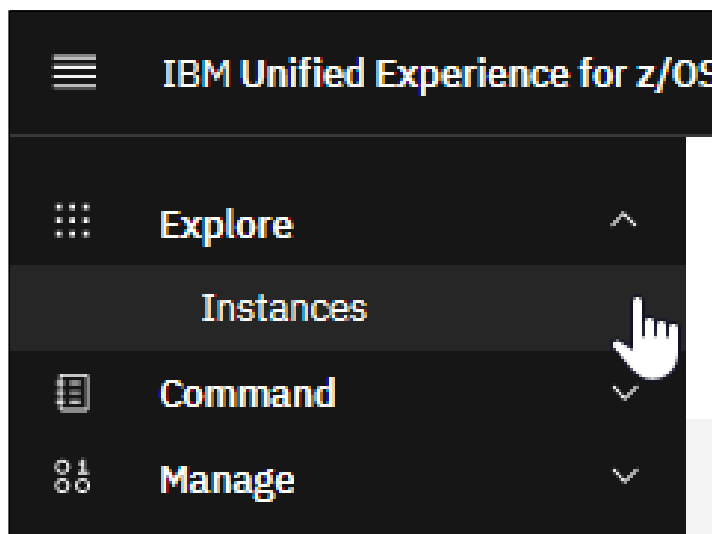


Figure 6-2 Selecting the Instances option

As shown in Figure 6-3, DBAs can use the Manage feature to define the policy and rules for each application and its artifacts. For more information, see 6.6, “Establishing the policy for database provisioning and schema changes” on page 43.

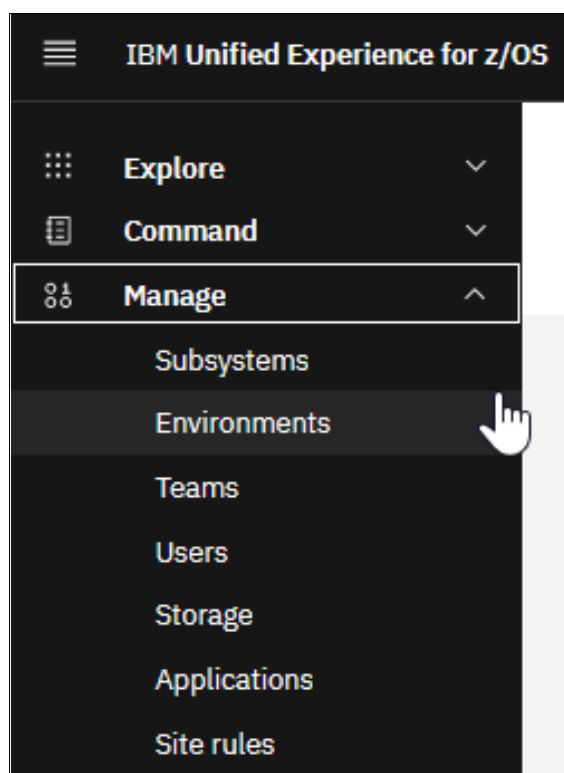


Figure 6-3 Selecting the Environments option

The following sections explain how developers’ actions can be validated immediately under the DBA’s control and rules.

6.2.2 Scripting the REST APIs

All actions that can be performed by using the user interface also can be scriptable to facilitate the automation of the following tasks:

- ▶ Creating, testing, and destroying schema changes to test applications that store data in Db2.
- ▶ Analyzing and applying database schema changes, including stored procedures, through CI/CD pipeline by scripting those services in an automation tool, such as Jenkins, UrbanCode Deploy (UCD), or Ansible. For more information about how scripting can be implemented in the context of our use case, see Chapter 7, “IBM UrbanCode Deploy” on page 55 and Chapter 8, “Jenkins automation server” on page 85.

The following sections provide detailed information about key concepts of Db2 DevOps Experience and how its features can help you overcome the challenges, as described in Chapter 2, “Db2 database code changes in the CI/CD pipeline” on page 7. They illustrate how administrators and application developers can use DOE’s features when adopting agile development practices with Db2 for z/OS:

- ▶ 6.3, “DOE terms and concepts” on page 30
- ▶ 6.4, “DOE roles and responsibilities” on page 31

- ▶ 6.5, “Establishing the landscape of Db2 subsystems, environments, and teams” on page 35
- ▶ 6.6, “Establishing the policy for database provisioning and schema changes” on page 43
- ▶ 6.7, “Defining Db2 site rules” on page 49
- ▶ 6.8, “DOE setup for DevOps engineer” on page 52

For more information about the architecture, installation, and configuration of UMS and DOE, see this [IBM Documentation web page](#).

6.3 DOE terms and concepts

To get started, it is important to have a clear understanding of some key DOE terms and concepts.

Note: The following definitions mention specific roles within a DOE environment. These roles are defined in 6.4, “DOE roles and responsibilities” on page 31.

6.3.1 Subsystems

The z/OS discovery services that are provided by UMS are started automatically when UMS is started. It discovers and stores information from all Db2 subsystems in the z/OS environment where it is running. After this initial discovery, super administrators can manually run subsystem discovery by refreshing the subsystem information.

Super administrators can view the major characteristics and register the subsystems under UMS/DOE for later use. Registered subsystems can be used by DOE to perform the following tasks:

- ▶ Create environments and associate them with the registered subsystems.
- ▶ Define a group or set of objects in a subsystem as applications.
- ▶ Provision application instances in a subsystem defined in an environment.

6.3.2 Environments

An *environment* is a collection of Db2 subsystems that is used by teams to provision application instances. Provisioning rules also are associated with each subsystem that is supported by the environment. Super administrators can create, edit, or delete environments. For more information, see 6.5.3, “Creating teams” on page 40.

6.3.3 Teams

A *team* is a group of users who work together within a defined application development or test environment. Each team must be associated with the environments that it can use. Super administrators can create teams, assign environments to teams, and add and delete team administrators and members. For more information, see 6.5.3, “Creating teams” on page 40.

6.3.4 Applications

An *application* is a set of objects, such as tablespaces, tables, and indexes that are grouped so that they can be managed and provisioned as a single unit for the use of an application program or a set of application programs. Application objects are logical, which means that they are only references to the objects.

When users provision instances of an application, the referenced objects are copied to create the instances. Super administrators and team administrators can register, change the settings of, or delete applications and assign them to a team.

An application definition is similar to an application component. The objects within an application are assigned to a team, and that team owns those application objects, which means that the team is responsible for them. They are authorized to create an instance to change the design of those application objects (at least in their sandbox), and commit code changes, just like a C, COBOL, or Python programmer who is part of a team that is associated with an application component.

For more information, see 6.6.1, “Registering applications” on page 44.

6.3.5 Objects

Objects are the Db2 resources that are required to run applications (a Db2 database, table spaces, tables, stored procedures, user-defined functions, and so on).

6.3.6 Instances

An *application instance* (which is referred to in this paper as an *instance*) is a set of application objects that were provisioned into the associated environment’s subsystem. An instance can be provisioned only by the members of the team that was assigned to an application and its environment.

This document focuses on handling instances throughout a CI/CD pipeline; however, tasks that are related to provisioning, editing, and deprovisioning an instance can be performed by using the DOE interface that is called the *IBM Unified Experience for z/OS*.

6.4 DOE roles and responsibilities

This section provides definitions of the roles and responsibilities under DOE so that you can assign your team personnel based on these definitions.

DOE provides support for a granular separation of duties by associating specific roles with specific responsibilities. The role-based, access control feature is provided through an external security manager, such as RACF®, ACF2, or Top Secret. In this document, we use IBM RACF. More information about the required RACF definitions is provided later in this section.

In preparation for defining roles for an application in DOE, you must understand the roles and responsibilities as defined by your organization for processes, such as deploying database code change to an integration environment.

Figure 6-4 shows which role is responsible for each step while deploying database schema changes to different environments such as test or production. These roles are matched to roles in DOE.

Step/task	Application Developer	DBA	Data Modeler
Create logical database design	Initiates	Consulted	Responsible for
Create/update DDL	Initiates	Responsible for	Informed
Deploy database code changes to any test environment	Responsible for		
Deploy database code changes to production	Informed	Responsible for	
Rollback database code changes to any test environment	Responsible for	Consulted	

Figure 6-4 Roles involved in developing database schema changes process

The following main roles are available when DOE is used:

- ▶ Super administrators
- ▶ Team administrators
- ▶ Team member

After you understand which tasks are associated with these roles, you can assign your personnel.

6.4.1 Super administrators

Users under the super administration profile can run the following tasks:

- ▶ Register subsystems
- ▶ Create:
 - Environments
 - Teams
 - Site rules
 - Applications
- ▶ Assign:
 - Teams to environments
 - Team administrators to teams
 - Site rule to environments
 - Applications to teams
- ▶ Specify storage limits

6.4.2 Team administrators

Users under the team administrator profile can run the following tasks:

- ▶ Update team members
- ▶ Perform the following tasks for their team's applications:
 - Create applications
 - Assign applications to teams
 - Assign site rule to team's applications
 - Approve or decline pull requests
 - Merge pull-request changes to originating applications

6.4.3 Team members

Users not under the previous two profiles can run the following tasks:

- ▶ Provision application instances
- ▶ Perform the following tasks for their instances:
 - Update instances objects
 - Issue pull requests
 - Apply updates to instances
 - Deprovision instances

Application developers and testers provision application instances into their team's environments. They can then change the definitions of the application objects that are included in the application instances in their own local environments for application program development and testing.

After they make changes and submit pull requests, team administrators and other reviewers of the team that owns the originating application can review their changes and approve or decline those requests.

6.4.4 DOE roles and RACF definitions

This section provides a high-level overview of implementing security for UMS/DOE. For more information about implementing security for UMS/DOE, see this [IBM Documentation web page](#).

As part of the UMS/DOE installation and configuration, your security administrator must define a RACF CLASS IZP and two SAF profiles: IZP.SUPER* and IZP.ADMIN*. Assign IDs that include the super administrator role to the IZP.SUPER* profile. Also, assign regular IDs that become team administrators or team members to the IZP.ADMIN* profile.

Sample JCL is provided (IZPUSRMD) that is run by your security administrator to add or remove IDs to these profiles. Also, the IDs are recorded into a SECURITY.USERLIST data set. The name of this data set is supplied to UMS/DOE by way of its PARMLIB.

Anytime that a UMS API is called when DOE tasks are run, these RACF profiles and userlist data set are checked to certify that the ID includes the suitable authorizations. The IDs that are recorded in the userlist are prompted to the super administrator and team administrator when they create or manage a team under DOE.

Notes: Consider the following points:

- ▶ At the time of this writing, UMS/DOE did not support RACF groups for super administrators or regular users; therefore, specific IDs must be added or removed individually. Nothing prevents from permitting the READ through a RACF group, but the user ID must be listed in the userlist. When IZPUSRMD is run, the script “explodes” the group into individual user IDs.
- ▶ The names of the userlist data set and RACF CLASS that are provided here are based on default values; however, your security administrator might use different names according to your site’s conventions. Also, the IZPUSRMD sample JCL helps complete these tasks, but your security administrator might decide to perform them in a different way.

For ease of reference, we use the configuration for roles and responsibilities that is shown in Figure 6-5.

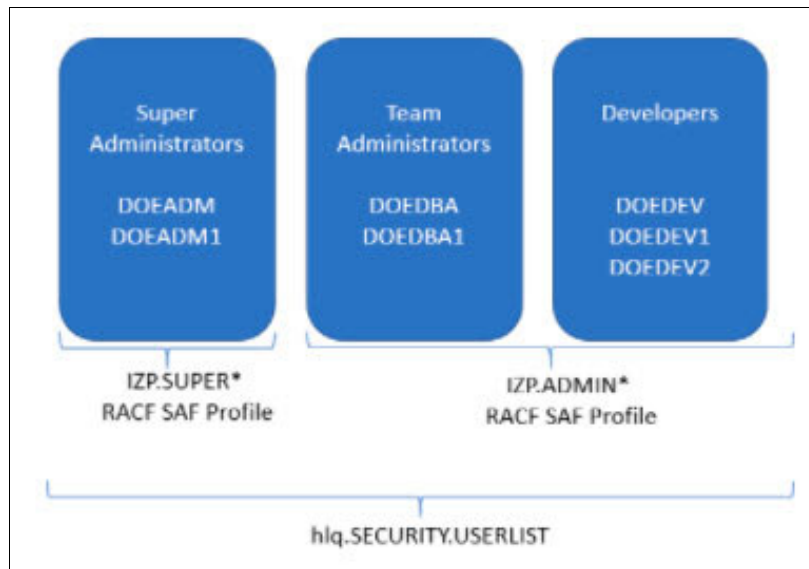


Figure 6-5 User IDs used in this document

In the following sections, you see how these IDs are used to run specific tasks to establish the landscape of Db2 systems and the policy for database provisioning and schema changes.

6.5 Establishing the landscape of Db2 subsystems, environments, and teams

Most of the steps in this section must be performed by a UMS/DOE user who has super administrator authority. Super administrators must set up subsystem environments for application development teams. To set up subsystem environments, the super administrator must perform the tasks that are described next.

6.5.1 Registering Db2 subsystems

To register a Db2 system, complete the following steps:

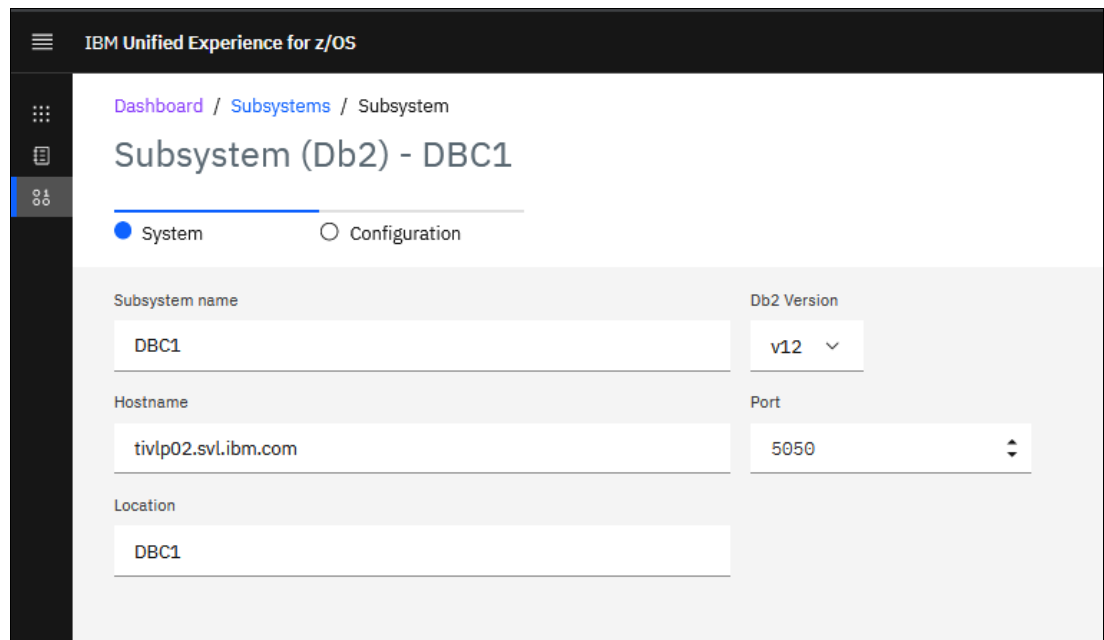
1. Log in to IBM Unified Experience for z/OS as the super administrator with the DOEADM ID.

2. Click **Manage** → **Subsystems**.

This page includes two tabs: Registered and Discovered. If subsystems were registered, they are displayed on the Registered tab (the tab is empty if no systems were registered by the super administrator). The Discovered tab contains a list of Db2 subsystems that were discovered by the UMS.

3. Click **Manage** → **Subsystems** → **Discovered**. Select the **Db2 subsystem** and then, click **Register Subsystem** to open the Register Subsystem window. The information in the System tab is populated automatically by the discovering API-based process.

Figure 6-6 shows an example of the System tab.



IBM Unified Experience for z/OS

Dashboard / Subsystems / Subsystem

Subsystem (Db2) - DBC1

☒ System ☐ Configuration

Subsystem name	Db2 Version
DBC1	v12
Hostname	Port
tivlp02.svl.ibm.com	5050
Location	
DBC1	

Figure 6-6 Registering Db2 subsystems with DOE -1

4. Click the **Configuration** tab and enter the information as shown in Figure 6-7. You can use the tooltips on each field to display information about the values that you must enter.

IBM Unified Experience for z/OS

Dashboard / Subsystems / Subsystem

Subsystem (Db2) - DBC1

☒ System ☐ Configuration

SDSNEXIT ⓘ	Schema ⓘ
DSN121.SDSNEXIT	IZPN01
SDSNLOAD ⓘ	Naming rule ⓘ
DSN.V12R1M0.SDSNLOAD	IZP1
RUNLIB ⓘ	Plan name ⓘ
DSN121.RUNLIB.LOAD	IZPPLAN1
DSNTEP2 Plan Name ⓘ	
DSNTEP2	
Job statement ⓘ	
//DBC1JOB JOB CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID, // REGION=0M,TIME=60 /*JOBPARM SYSAFF=*	

Figure 6-7 Registering Db2 subsystems with DOE -2

Figure 6-8 shows what a registered Db2 system looks like in the UMS/DOE dashboard.

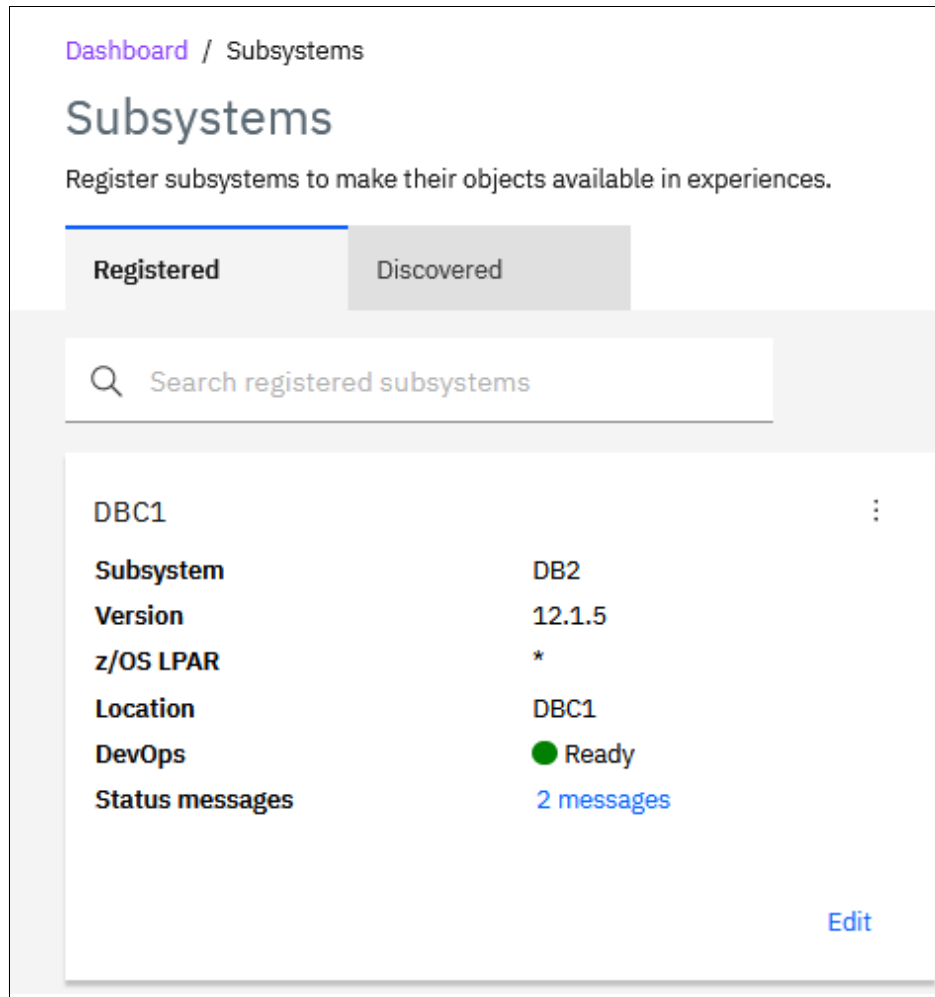


Figure 6-8 Registering Db2 subsystems with DOE -3

After the subsystem is registered, the objects from that system can be used when creating an application, as described in 6.6, “Establishing the policy for database provisioning and schema changes” on page 43.

Note: At the time of this writing, the Db2 catalog information of the registered subsystem is automatically refreshed every 4 hours by default. Therefore, if new objects are created after the subsystem is registered, they are not discovered immediately; however, the super administrator can manually refresh the registration by clicking the three dots and selecting **Refresh Discovery**, as shown in Figure 6-9.

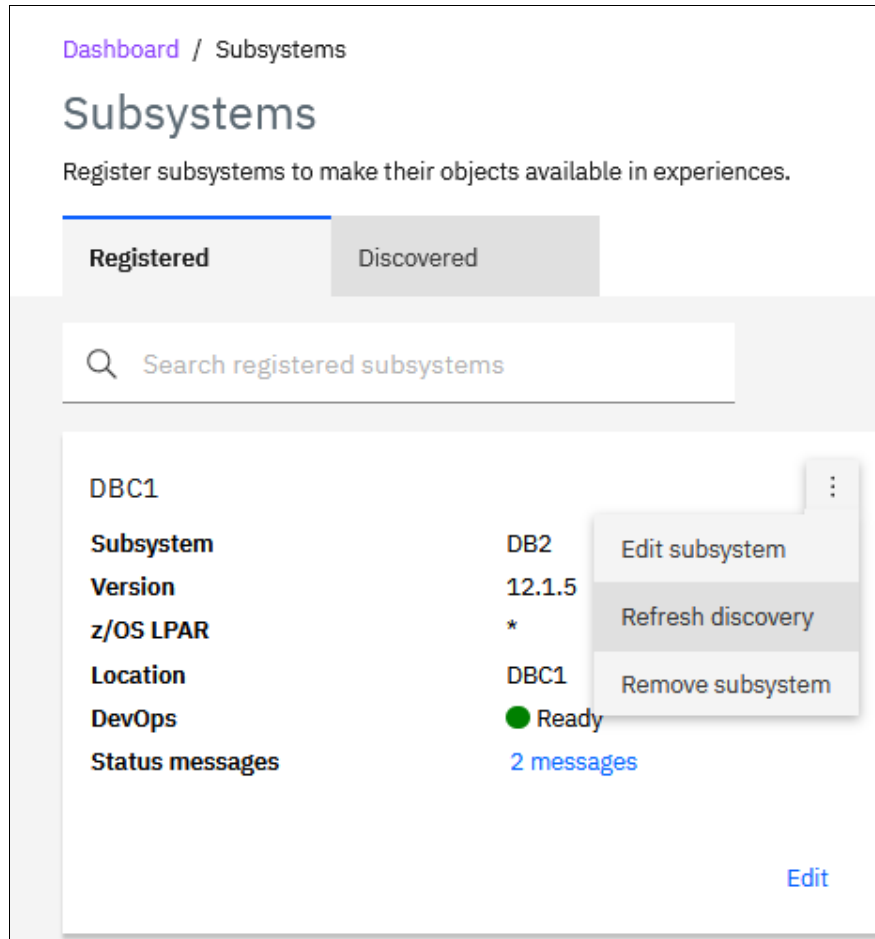


Figure 6-9 Registering Db2 subsystems with DOE -4

6.5.2 Creating environments

Now that you registered Db2 subsystems to work with, the next step is to create environments that can be assigned to a team of developers. Complete the following steps:

1. Log in to IBM Unified Experience for z/OS as the super administrator with the DOEADM ID.
2. Open the Create environment window by clicking **Manage** → **Environment** → **Create environment**.

3. Assign a name for the environment (for example, DEVENV) and assign a Db2 subsystem to that environment (for example, DBC1), as shown in Figure 6-10.

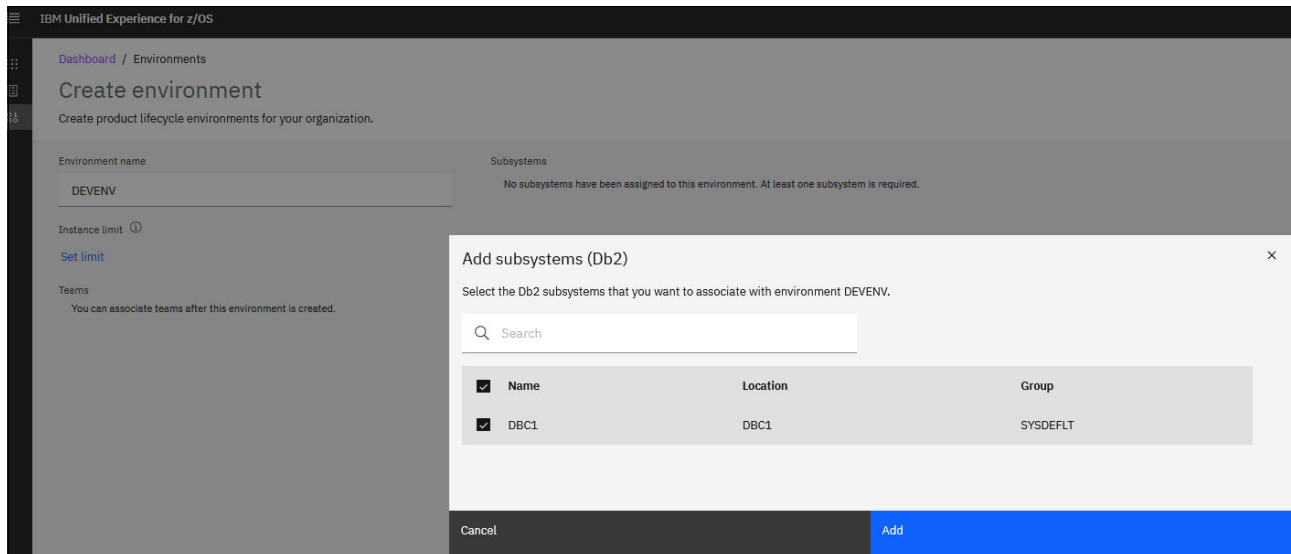


Figure 6-10 Creating environments: Window 1

You can limit the number of instances that can be provisioned under this environment. You also can establish provisioning rules that are applied when a team member provisions an instance.

In the following example, the number of instances that can be provisioned is limited to three, and a rule was established that automatically assigns the characters “GENC” to the names of schema objects that are provisioned in this instance (see Figure 6-11).

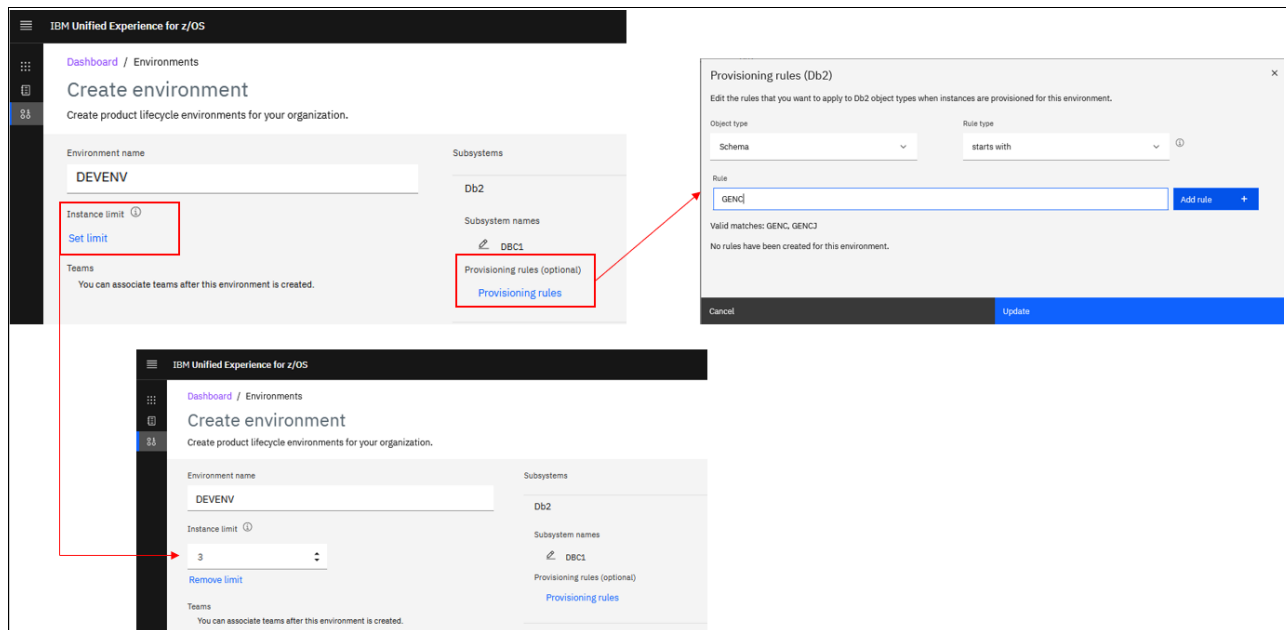


Figure 6-11 Creating environments: Window 2

6.5.3 Creating teams

Now that you registered Db2 subsystems and created an environment, your next step is to create teams of developers. Complete the following steps:

1. Open the Create team window by clicking **Manage** → **Create team**, and provide the following information:
 - A name for the team (for example, DEVTEAM)
 - (Optional) A JOB prefix to be used under this team (for example, DBC1)
2. Click **Manage Users**. Then, select the users to add to the team, including whether they are team administrators (a role that is typically assigned to a DBA or lead developer).

Note: The IDs that are prompted here are the contents of the hlq.SECURITY.USERLIST, as described in 6.4.4, “DOE roles and RACF definitions” on page 33.

3. Assign the team an environment (for example, DEVENV) and optionally assign a limit to the number of instances that the team and each team member can establish.

Figure 6-12 shows the result.

The screenshot displays the 'Team - DEVTEAM' configuration interface. On the left, the 'Team name' is set to 'DEVTEAM' and the 'Job prefix (optional)' is 'DBC1'. Below these, a list of 'Team members and administrators (optional)' is shown, including users like DOEDBA, DOEDBA1, DOEDEV, etc., with checkboxes for 'Team administrator'. A 'Manage users' button is located next to this list. On the right, the 'Environments' section shows a table with columns: Name, Type, Subsystems, Environment, and Instance limit. The table contains one entry: 'DEVENV' (Type: Db2, Subsystems: DBC1, Environment: No limit, Instance limit: 0). There are 'Remove limit' links for both the team and user instance limits.

Name	Type	Subsystems	Environment	Instance limit
DEVENV	Db2	DBC1	No limit	0

Figure 6-12 Creating teams

6.5.4 Managing team members

You can always add users to a team, remove users from a specific team, or remove users from all teams to which they are assigned. However, only a super administrator (DOEADM) or a team administrator (DOEDBA or DOEDBA1) can perform these operations.

Assigning a new user to the team

Complete the following steps to assign a new user to the team:

1. Open the Users window by clicking **Manage** → **Users**.
2. From the ID column, select the user that you want to assign to a team. For example, user DOEDEV3 can be assigned to our new DEVTEAM team.
3. From the Teams column, click the **blue pencil** to display a list of teams. Select the team from the Team name column or enter the team name in the search field.
4. Indicate user role (Member or Team administrator).

5. Click **Assign** (see Figure 6-13).

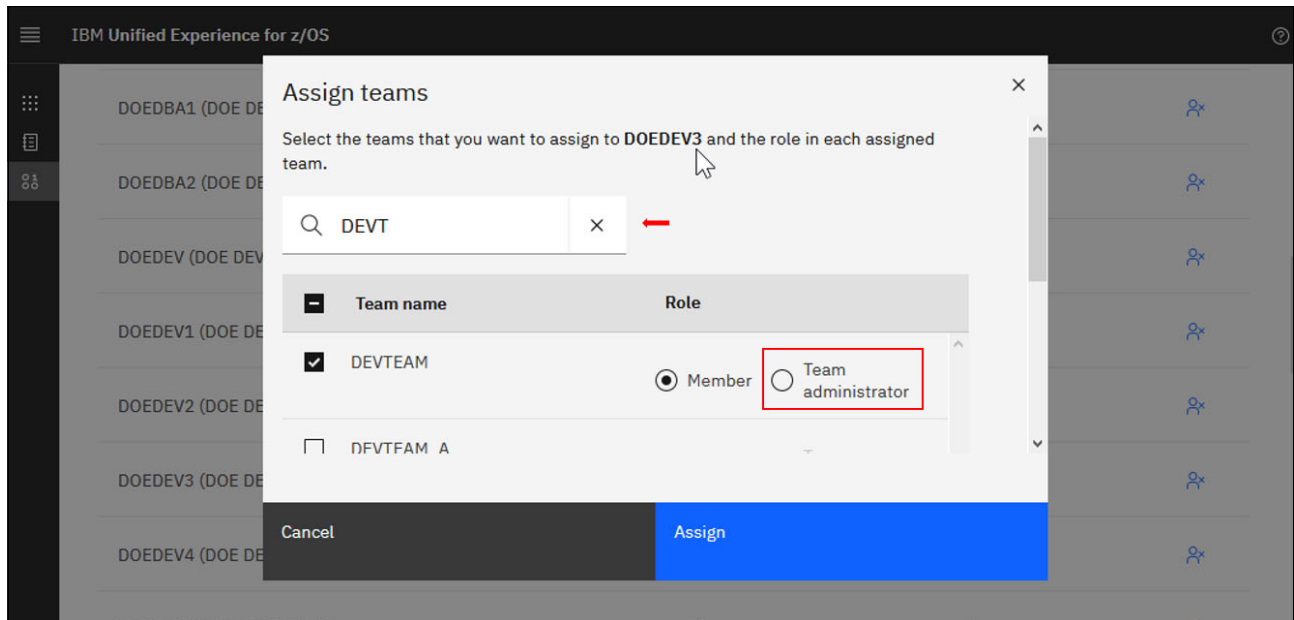


Figure 6-13 Assigning a new user to the team

Removing users from teams

Complete the following steps to remove users from teams:

1. Click **Manage** → **Users**.
2. From the ID column, select the user that you want to remove from a team. For example, user DOEDEV2 must be removed from the ANSTEAM team.
3. From the Teams column, click the **blue pencil** to display a list of teams. Select the team from the Team name column or enter the team name in the search field.

4. Click **Assign** (see Figure 6-14).

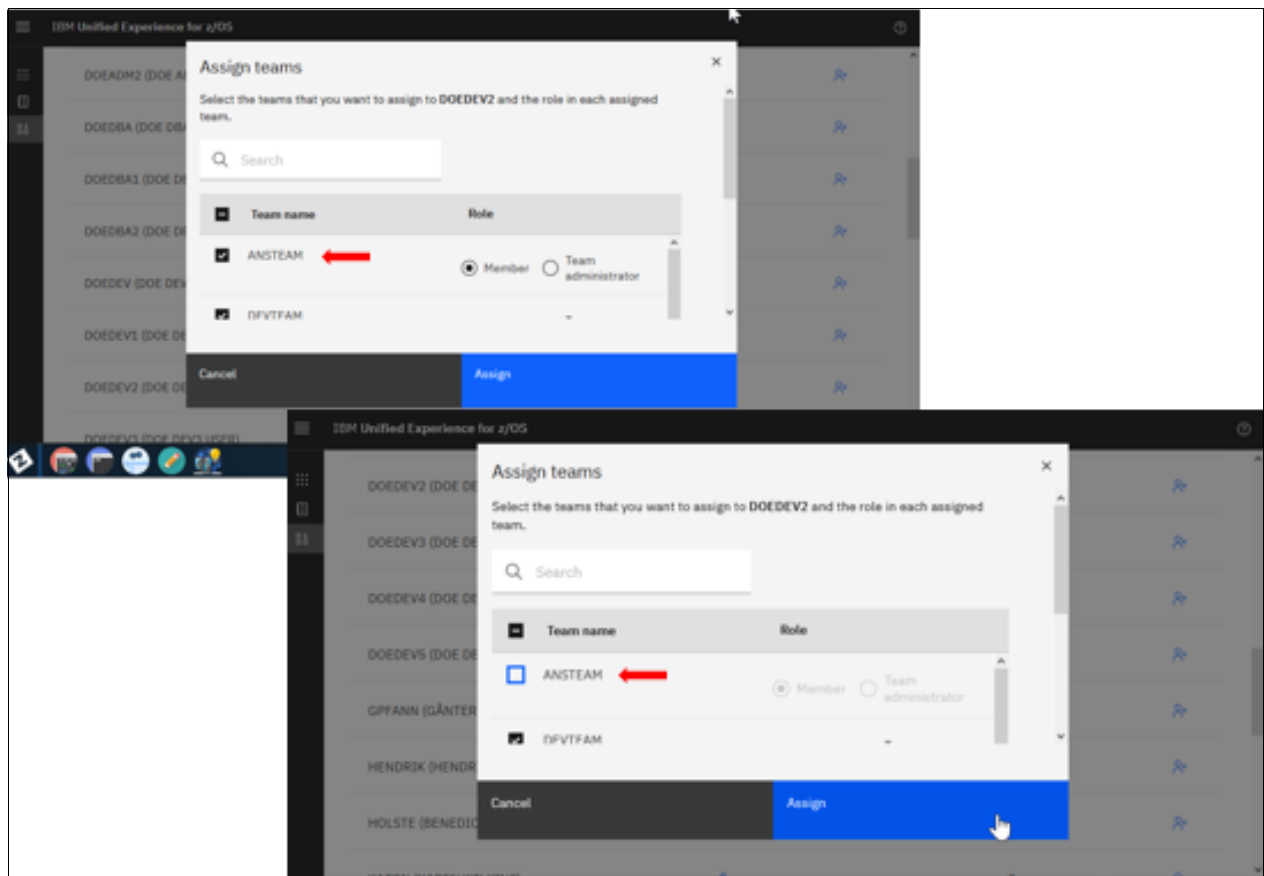


Figure 6-14 Removing users from teams -1

Complete the following steps to remove user DOEDEV2 from all teams:

1. Click **Manage** → **Users**.
2. From the ID column, select the user that you want to remove from all teams. For example, user DOEDEV2 must be removed from all teams.
3. Click the **Remove from all teams** icon in the far-right column (see Figure 6-15 on page 43).

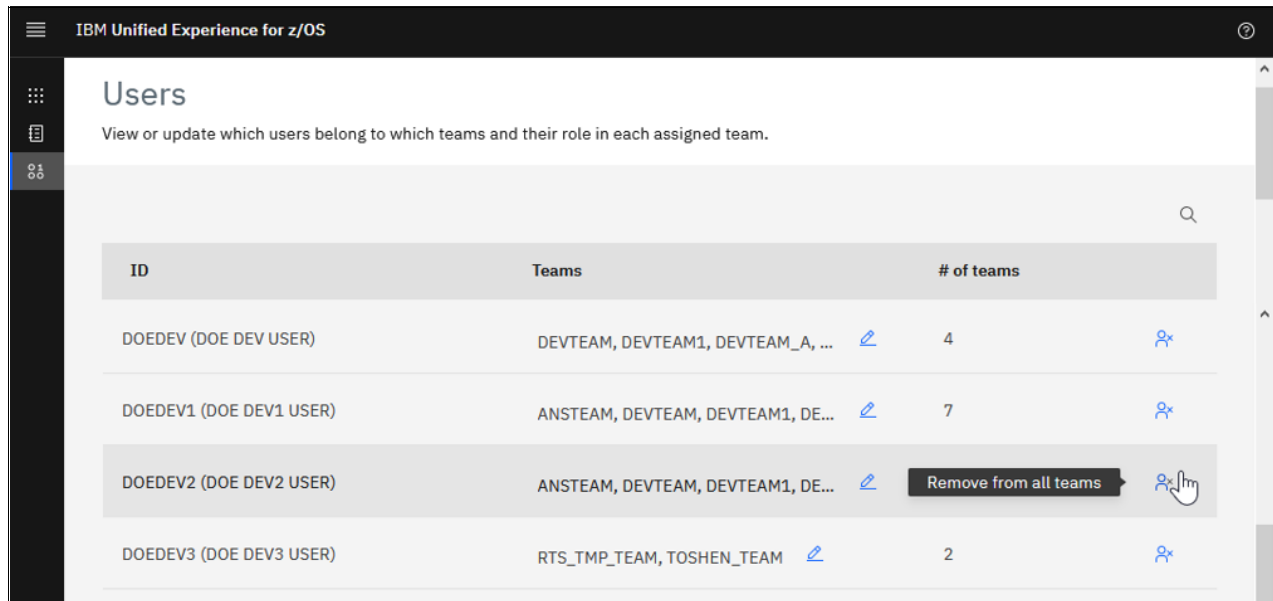


Figure 6-15 Removing users from teams -2

4. You are prompted to confirm or cancel the operation, as shown in Figure 6-16.

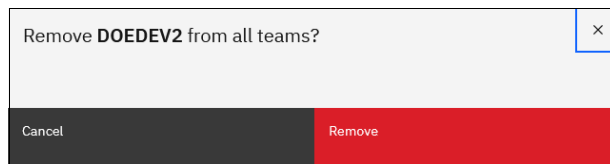


Figure 6-16 Removing users from teams -3

6.6 Establishing the policy for database provisioning and schema changes

UMS/DOE team administrators are responsible for setting up the policy that defines an application.

The following sections provide step-by-step instructions for setting up an application. These steps must be performed by a super administrator or a team administrator.

As discussed in 6.3, “DOE terms and concepts” on page 30, in DOE the term *application* refers to a set of Db2 objects that are grouped and used in application programs.

Registering an application under DOE can be done by a super administrator (in our example, DOEADM) or by the team administrator (in our example, DOEDBA or DOEDBA1) of the team that works on the application programs that use that set of Db2 objects.

When an application is registered, it is assigned to a team, which enables developers, testers, and other members of that team to provision application instances in their own environment and to work on the objects.

Note: The use case in this publication uses an instance for the integration branch.

6.6.1 Registering applications

Complete the following steps to register applications:

1. Click **Manage** → **Application** → **Register Application** (see Figure 6-17).

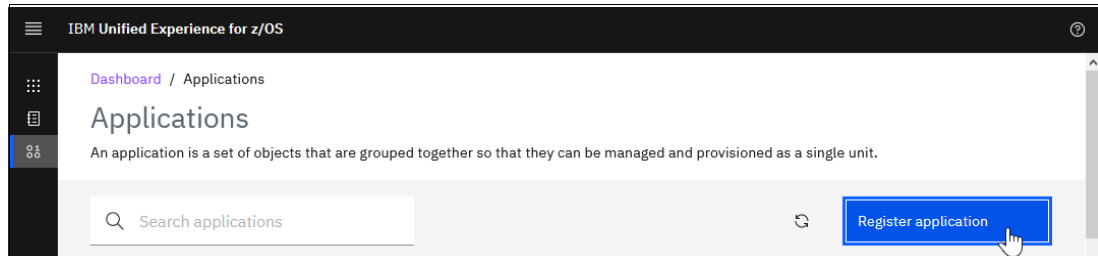


Figure 6-17 Selecting Registering application option

2. Select the registered Db2 subsystem that stores the objects for this application, and click **Next** (see Figure 6-18).

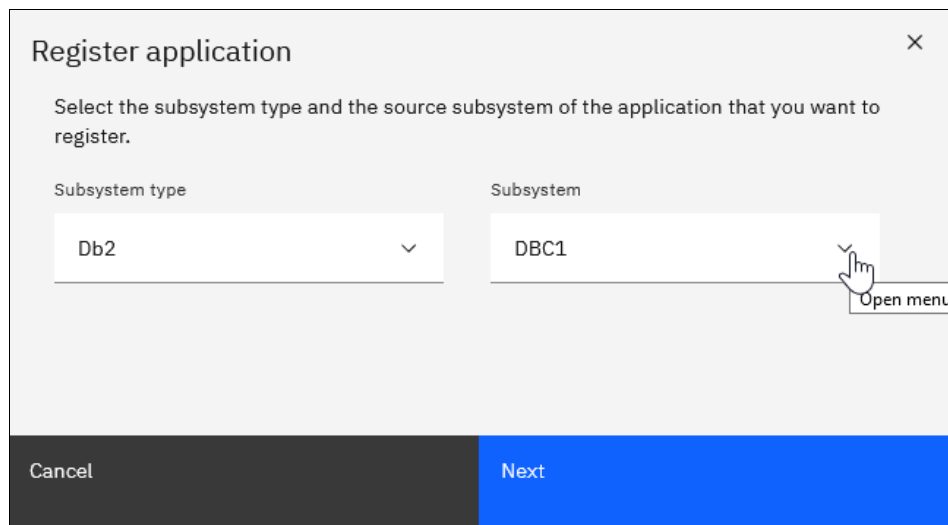


Figure 6-18 Selecting subsystem type

3. Use the next window (see Figure 6-19) to search for and select the objects that can form the set for the application that you are registering.

Note: Values that are entered in the search argument are case-sensitive.

IBM Unified Experience for z/OS

Dashboard / Applications / Register application

Register application

☒ Search and select ☐ Review and register

Source subsystem: DBC1

Search and select objects

Search by object type ^

Select an object type to begin a query. ⓘ

Database	Name
TableSpace	Name
Table ⓘ	Schema
View	Schema
Synonym	Database
Alias	Database
Stored Procedure	Tablespace
Function	Tablespace
Trigger	Tablespace

Close Search

Selected objects (0)

Search object

Dependent objects ⓘ

☒ Include dependent objects

Associated applications (optional) ⓘ

Cancel Back Next

Figure 6-19 Searching for and selecting objects

4. After you submit the search criteria, you can select the required objects from the list that is returned (see Figure 6-20).

Note: You can select all objects or you can select individual objects. You can select objects from multiple schemas to be part of the application.

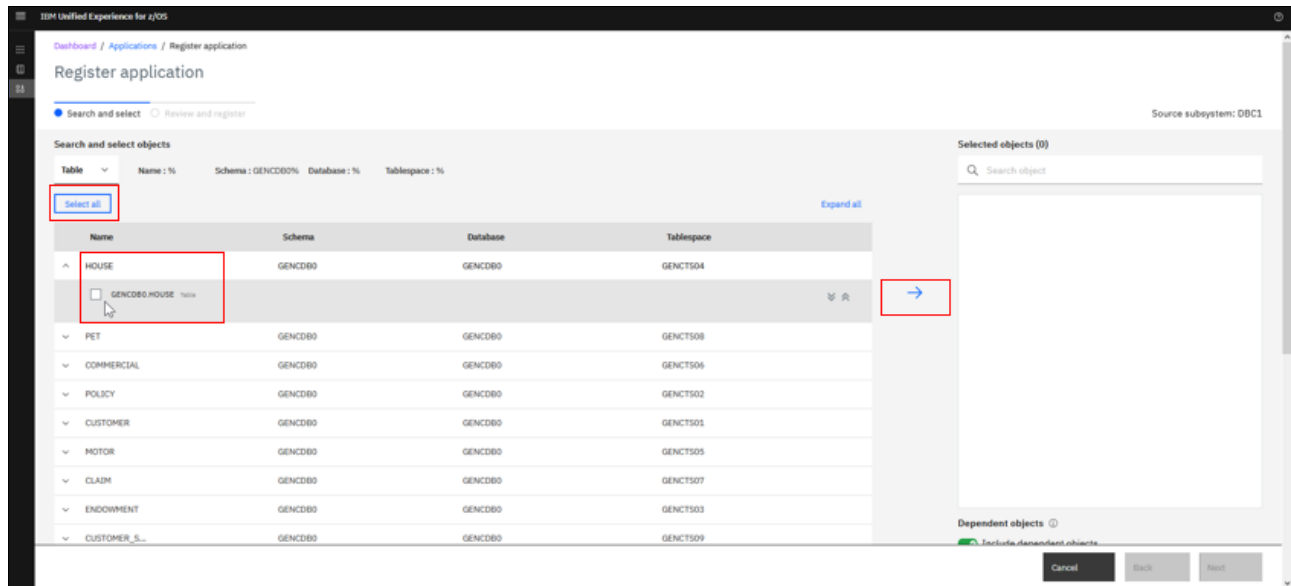


Figure 6-20 Selecting required objects

5. After you select the objects you want, click **Next** and complete the registration by providing the following information:
 - A unique name for your application (for example, GenApp)
 - A color for the tile of the application in the dashboard
 - (Optional) A description for the application
 - The team that owns that application (for example, DEVTEAM)

6. Click **Register** to complete the registration (see Figure 6-21).

The screenshot shows the 'Register application' page in the IBM Unified Experience for z/OS. The breadcrumb trail is 'Dashboard / Applications / Register application'. The page has two tabs: 'Search and select' (active) and 'Review and register'. The 'Source subsystem' is 'DBC1'. On the left, under 'Selected objects (29)', a list of tables is shown, including 'GENCDB0.CUSTOMER_SEC...', 'GENCDB0.CLAIM', 'GENCDB0.CUSTOMER', 'GENCDB0.PET', 'GENCDB0.MOTOR', 'GENCDB0.COMMERCIAL', 'GENCDB0.POLICY', 'GENCDB0.HOUSE', and 'GENCDB0.ENDOWMENT'. The main form has fields for 'Name' (GenApp), 'Color' (blue), 'Description (optional)' (with a placeholder 'Enter a description'), and 'Owner team' (DEVTEAM). At the bottom right are 'Cancel', 'Back', and 'Register' buttons.

Figure 6-21 Selecting Register

When the registration is complete, a new tile for the application is added to Dashboard/Applications. The color that you selected appears as a vertical bar at the left of the tile (see Figure 6-22).

The screenshot shows a tile for the application 'GenApp'. It has a blue vertical bar on the left. The tile displays the following information: 'Db2' (with 'DBC1' to its right), 'Team' (with 'DEVTEAM' to its right), 'Status' (with a green checkmark and 'Ready' to its right), and 'Status messages' (with '4 messages' to its right). A 'Details' link is at the bottom right.

Figure 6-22 Updated application tile

6.6.2 Provisioning an application instance

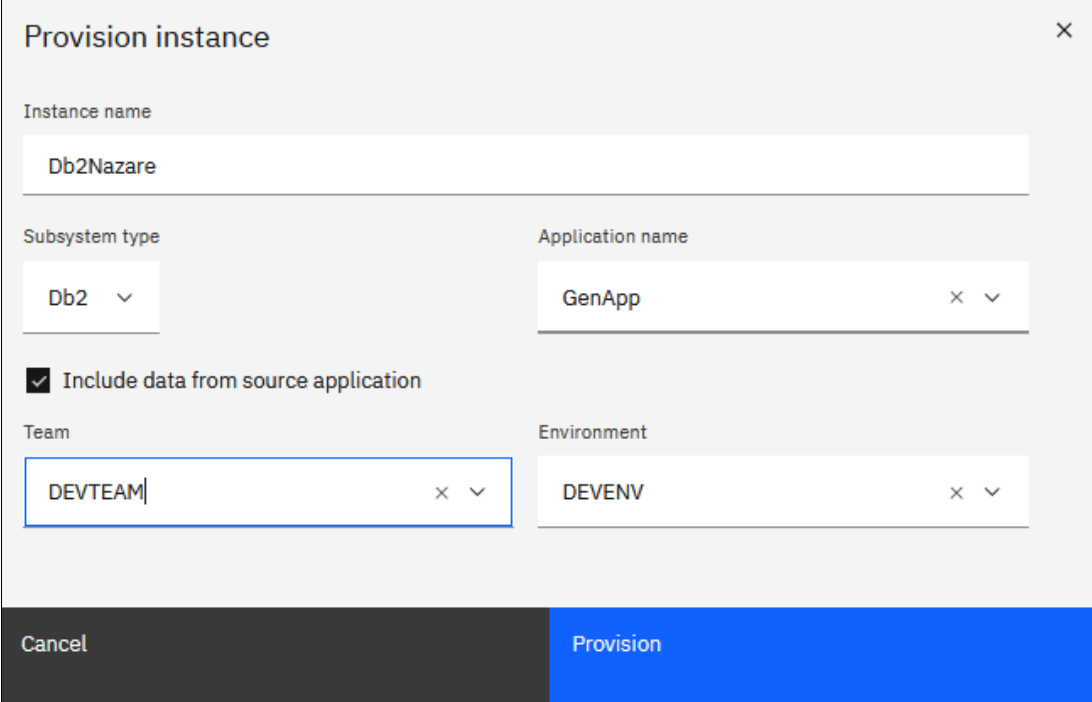
When an application is assigned to a team, the developers and testers on that team can provision their own application instances. For the purpose of this document, we show how a database administrator can provision an instance for their continuous integration environment.

In later sections of this document, we explain how information that is registered in DOE is correlated to the pipeline orchestrator (for example, Jenkins) and to the deployment manager (for example, UrbanCode Deploy – UCD).

The following steps must be performed by a team administrator of the team:

1. Click **Explore** → **Instances** → **Provision instance**.

Figure 6-23 shows an example of how to enter the information to provision an instance based on our use case.



The screenshot shows a 'Provision instance' dialog box. It includes the following fields and controls:

- Instance name:** A text input field containing 'Db2Nazare'.
- Subsystem type:** A dropdown menu with 'Db2' selected.
- Application name:** A dropdown menu with 'GenApp' selected.
- Include data from source application:** A checked checkbox.
- Team:** A dropdown menu with 'DEVTEAM' selected.
- Environment:** A dropdown menu with 'DEVENV' selected.
- Buttons:** 'Cancel' and 'Provision' buttons at the bottom.

Figure 6-23 Provisioning instance window

2. Click **Provision**. A new tile for that instance is created in the **Dashboard/Instances** (see Figure 6-24).

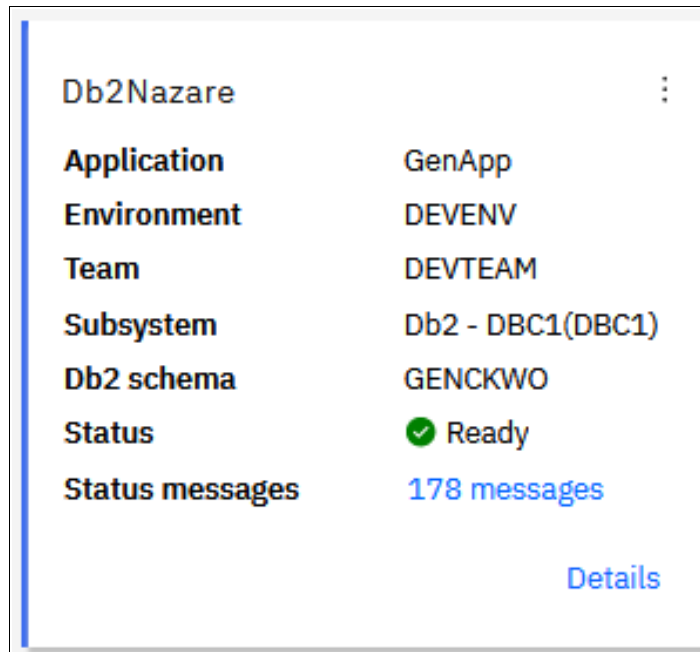


Figure 6-24 Provisioning an application instance

The provisioned instance includes a schema as GENCKWO, which adheres to the provisioning rules that were established when the DEVENV environment was created. For more information, see 6.5.2, “Creating environments” on page 38.

As a super administrator, you can also manage the storage that is used by teams, environments, users, and applications. The limits are soft, which means that when they are exceeded, Unified Experience for z/OS displays alerts but does not prevent continued activity.

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

6.7 Defining Db2 site rules

When you adopt a development model that provides the flexibility and agility to application developers to make application and DDL changes on their own development environment, it is imperative that administrators can use mechanisms to enforce Db2 guidelines and rules. Although the application developer can independently make database changes to test new features or to accommodate application code changes, those changes must maintain the installation standards, such as naming convention rules and attribute values.

Although the creation of site rules is optional, they are important to constrain how application developers can change object definitions in provisioned application instances. To define site rules, DBAs must be granted a super administrator role under DOE.

After a rule is defined, it can be associated to one or more applications, environments, or both. The rule is honored for every provisioned application instance of that application and for every provisioned application instance to that environment.

Site rules can be classified as:

- ▶ Simple
- ▶ Complex

6.7.1 Simple site rules

A simple rule is characterized by involving object, attribute, and verification. For example, you can create a site rule that specifies that column (object) names (attribute) must have a maximum length of 20 characters (verification).

Complete the following steps to create a simple site rule:

1. Click **Manage** → **Site rules** → **Create rule**.

You can use the drop-down arrows to define the elements of the rule (see Figure 6-25).

IBM Unified Experience for z/OS

Dashboard / Site rules /

Edit site rule (Db2)

Review and edit the existing site rule.

Rule name ⓘ

COL_Name_1

Object: Column

Attribute: NAME

Constraint: max length

Value: 20

Figure 6-25 Edit site rule (Db2) window

2. After you create the site rule, assign it to applications and environments. Operations such as editing, duplicating, and deleting also are available (see Figure 6-26).

Dashboard / Site rules

Site rules

Create site rules and apply them to applications and environments.

Search

Create rule

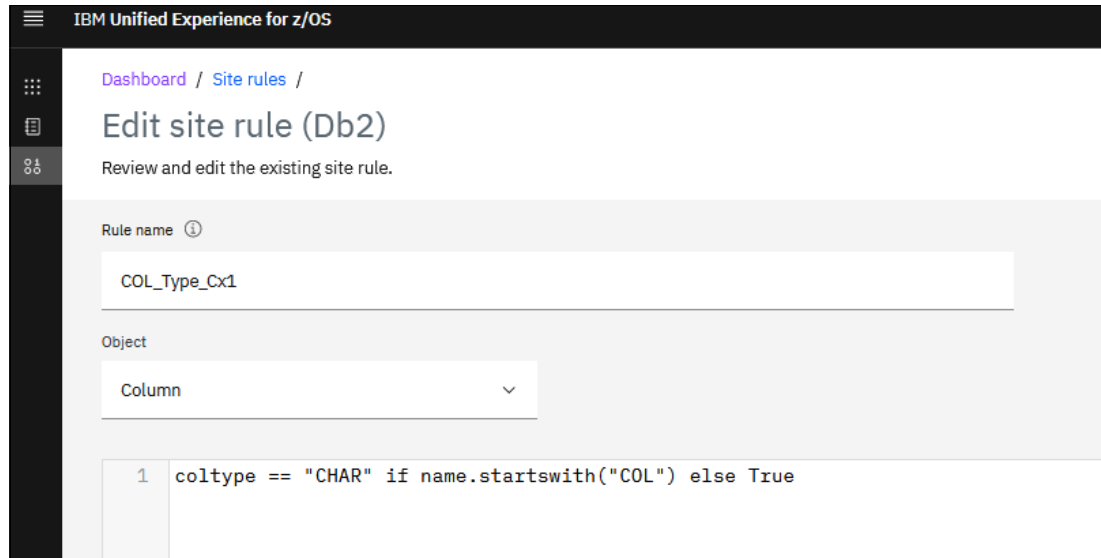
Name	Subsystem type	Rule	Applications	Environments
COL_Name_1	Db2	Column NAME must have maximum length of 20	GenApp, GenApp_A, SUELI_APP	DEVENV, DEVENV_A, SUELI_ENV
COL_Type_1	Db2	Column COLTYPE must not be one of the following: DECFLOAT	-	SUELI_ENV
COL_Type_Col	Db2	Column coltype == "CHAR" if name.startwith("COL") else True	SUELI_APP	DEVENV, SUELI_ENV
IDK_CK1	Db2	Index name.startwith("I" + tname) and \ name[-2:] >= "00" and name[-2:] <= "99" and \ copy == "NO" and \ compress == "NO" and \ bpool != "BP0" and \ cluster == "YES" and \ clustering == "NOT CLUSTER"	SUELI_APP	SUELI_ENV

Edit rule
Duplicate rule
Assign to applications
Assign to environments
Delete rule

Figure 6-26 Assigning applications and environments

6.7.2 Complex site rules

You also can create complex site rules, which give you a more flexible way to define a rule by adding logic to it. A subset of the Python syntax can be used to compose a site rule. As shown in Figure 6-27, a rule is defined by using the **startswith** Python function.



IBM Unified Experience for z/OS

Dashboard / Site rules /

Edit site rule (Db2)

Review and edit the existing site rule.

Rule name ⓘ

COL_Type_Cx1

Object

Column

```
1 coltype == "CHAR" if name.startswith("COL") else True
```

Figure 6-27 Using the *startswith* Python function

Although it is possible to create a rule that involves several attributes of an object, the best practice is to create one rule for each attribute of an object so that it is easy to identify which rules an object definition might violate. Otherwise, troubleshooting an object that violates multiple rules can become a tedious chore that requires multiple rounds of debugging. For example, do not create a rule that looks like the example that is shown in Figure 6-28.



IBM Unified Experience for z/OS

Dashboard / Site rules /

Edit site rule (Db2)

Review and edit the existing site rule.

Rule name ⓘ

IDX_Seveal_Attributes

Object

Index

```
1 name.startswith("I" + tbnam) and \
2 name[-2:] >= "00" and name[-2:] <= "99" and \
3 copy == "NO" and \
4 compress == "NO" and \
5 bpool != "BP0" and \
6 closerule == "YES" and \
7 clustering == "NOT CLUSTER"
8
```

Figure 6-28 Example of object that violates multiple rules

Instead, break down the rule into individual rules to be assigned, as shown in Figure 6-29.

<input type="checkbox"/> Name	Subsystem type	Rule
<input type="checkbox"/> IDX_Name_Cx1	Db2	Index name.startswith("I" + tbnname) and \ name[-2:] >= "00" and name[-2:] <= "99"
<input type="checkbox"/> IX_BPOOL	Db2	Index bpool and bpool == "BP11"
<input type="checkbox"/> IX_CLOSE	Db2	Index CLOSERULE must equal YES
<input type="checkbox"/> IX_CLUSTER	Db2	Index CLUSTERING must equal NOT CLUSTER
<input type="checkbox"/> IX_COMPRESS	Db2	Index COMPRESS must equal NO
<input type="checkbox"/> IX_COPY	Db2	Index COPY must equal NO
<input type="checkbox"/> IX_PRQTY	Db2	Index priqty == -1
<input type="checkbox"/> IX_SEQQTY	Db2	Index seqqty == -1

Figure 6-29 Breaking down a rule into individual rules

You can define as many site rules as needed. For more information about a complete reference of objects, attributes, and Python functions that you can use to create your site rules, see this [IBM Documentation web page](#).

6.8 DOE setup for DevOps engineer

When you establish the policy for database provisioning and schema changes in your environment, you must provide your DevOps engineer with critical information that they use when setting up the pipeline and the automated deployment.

6.8.1 Mapping DOE definitions to GitHub and UCD

In this section, you see how the definitions that we created in the previous sections are mapped in the pipeline orchestrator (Jenkins) and UCD configuration.

DOE and GitHub

Chapter 8, “Jenkins automation server” on page 85 provides more information about setting up the Jenkins pipeline. The name of the application (GenApp) that is defined under DOE must be used by your DevOps engineer when they configure the pipeline.

In our use case, we use GitHub as our source control management (SCM) system. The json file (app_def.json) in GitHub contains the application definition. To illustrate this mapping, we use the db2dev_integration branch in GitHub and the Integration UCD environment for deployment.

Figure 6-30 shows how the name of the application that is defined under DOE correlates, how it is stored in GitHub, and how it is used in UCD by the DOE-UCD plug-in during the deployment process.

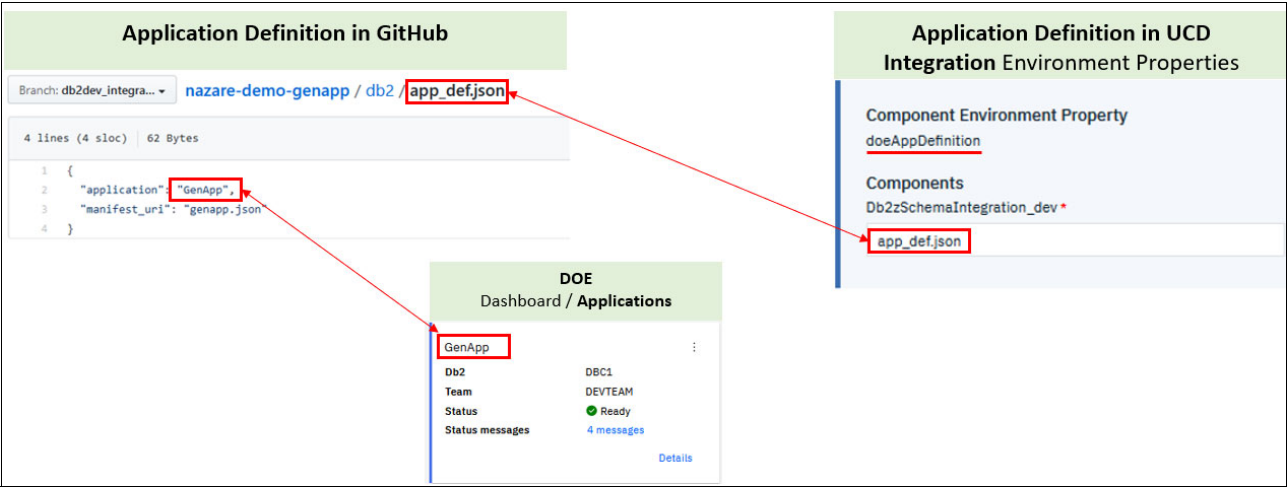


Figure 6-30 DOE and GitHub

DOE and UCD

Chapter 7, “IBM UrbanCode Deploy” on page 55 provides more information about configuring the UCD artifacts for automated deployment of Db2 schema changes. The name of the instance (Db2Nazare) that is defined under DOE must be used by your DevOps engineer when they configure UCD.

Note: The name of the application that is defined under DOE (GenApp) also is passed to UCD.

In UCD, an application also is defined. For each application, environments exist into which that application can be deployed. When you configure the environment properties, the application instance name that was defined under DOE must match the environment configuration. In our use case, we use the Integration environment to pass the Db2Nazare instance of the GenApp application (see Figure 6-31).

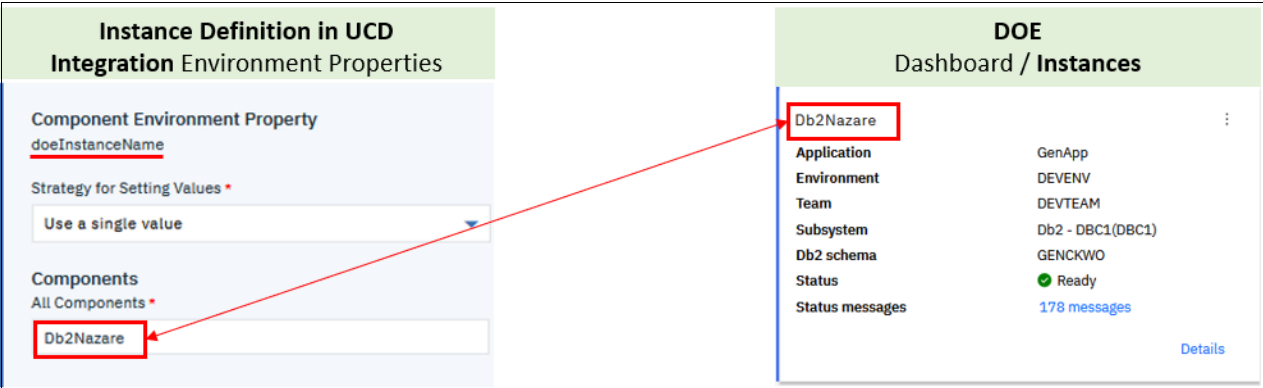


Figure 6-31 DOE and UCD



IBM UrbanCode Deploy

This chapter introduces you to the basic concepts of IBM UrbanCode Deploy (UCD) and describes how you integrate IBM Db2 DevOps Experience for z/OS (DOE) into a deployment process to drive Db2 for z/OS schema changes.

Instructions for installing UCD are beyond the scope of this document. For more information, see this [IBM Documentation web page](#).

This chapter includes the following topics:

- ▶ 7.1, “Introduction to IBM UrbanCode Deploy” on page 56
- ▶ 7.2, “Db2 DevOps Experience plug-in for UCD” on page 58

7.1 Introduction to IBM UrbanCode Deploy

IBM UrbanCode Deploy (UCD) helps you meet the challenge of deploying software throughout your enterprise by providing tools that improve deployment speed and reliability. UCD standardizes and simplifies the process of deploying software components to each environment in your development cycle. It also provides the tools to model processes that orchestrate complex deployments across every environment and approval gate.

The IBM UCD elements that you use to model software deployments include applications, environments, components, and related processes. Figure 7-1 shows the relationships between different elements.

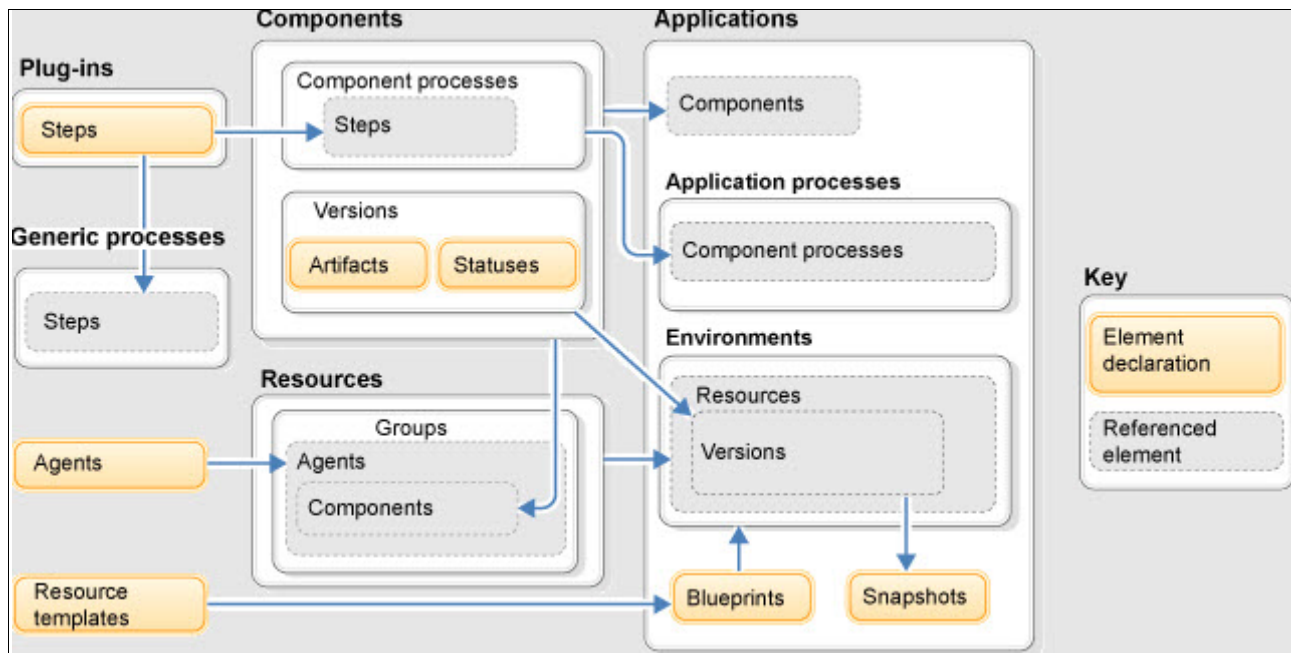


Figure 7-1 UCD elements that are defined in an environment to implement Db2 for z/OS schema changes

To implement Db2 for z/OS schema changes, you must define the following elements in your environment:

- **Components**

Components represent deployable items, also called *artifacts*, which can be files, images, database schema definitions (DDL), configuration materials, or anything else that is associated with a software project. Artifacts can come from a number of sources, such as file systems, build servers, and source version control systems.

Whenever you import new files that are associated with a component, a component version is created. The server keeps copies of these component version artifacts, and you specify which component version to use each time you deploy.

Components include one or more component processes that are associated with them. A component process contains all the steps that are necessary for the server to deploy the component to an environment. These steps describe all actions, such as starting an application server and creating or updating tables in a database. Many process steps are available by default, and you can obtain more steps from plug-ins.

- Applications

An application is a logical group of components that are deployed together. Applications include processes, but their steps often start processes or modify resources.

An application contains environments that describe all the deployment locations that are needed in your deployment process. Application processes are run to deploy components to environments.

- Resources

Resources represent the relationships between components, the agents that deploy the components, and the target deployment environments. You create groups to represent target servers and environments, attach the agents that deploy to the environments, and attach components to the agents that run their processes.

- Agents

An agent is a lightweight process that runs on a deployment target host and communicates with the UCD server. It does the work of deploying components.

- Environments

An environment is a user-defined collection of resources that hosts an application.

Environments often are modeled on some stage of the software project lifecycle, such as development, quality assurance (QA), or production. An environment can consist of a single server or it can be spread over several servers or over clusters of servers.

Environments are assigned to specific applications.

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

7.2 Db2 DevOps Experience plug-in for UCD

DOE provides a rich set of REST APIs that provide all of the functions that you need to drive Db2 for z/OS schema changes.

You can access the Swagger API documentation by browsing `https://<host>:<port>/ws/swagger-ui.html` where `<host>` is the DNS name or IP address of the z/OS system where you installed DOE and `<port>` is the port number that you specified for Swagger (12023 by default).

Figure 7-2 shows some examples of the API documentation in Swagger.

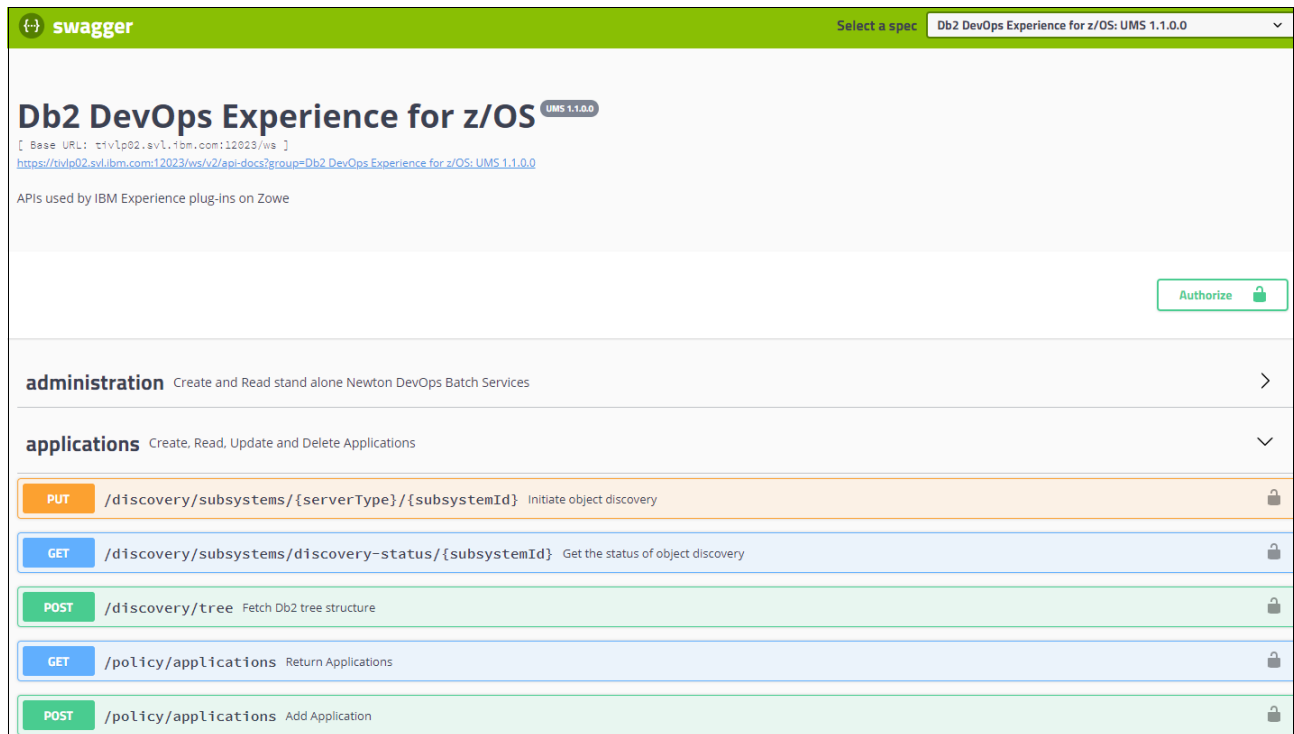


Figure 7-2 Examples of the API documentation in Swagger

To make it easier for you to drive Db2 for z/OS schema changes by using DOE from a UCD deployment process, IBM developed a plug-in that implements the following tasks (see Table 7-1) by calling the REST APIs that are provided by DOE.

Table 7-1 Db2 DevOps Experience plug-in tasks

Task	Description
Provision	Provision a Db2 schema instance of an application.
Deprovision	Deprovision a Db2 schema instance of an application.
Analyze	Analyze schema changes on a Db2 schema instance.
Apply	Apply schema changes on a Db2 schema instance of an application.
Promote	Promote schema changes from a Db2 schema instance to the golden master copy of an application.
Sync up	Sync up schema changes from the golden master copy to a Db2 schema instance of an application.

You can download the Db2 DevOps Experience plug-in from this [UCD plug-ins web page](#).

7.2.1 DOE APIs used by the plug-in

As described in Chapter 6, “Delivering database and application changes at the same speed” on page 25, the DOE plug-in for UCD calls multiple DOE APIs to implement a specific task. This section lists the required parameters and APIs that are called by each task.

All tasks require the following input parameters:

- ▶ Host name or IP address of the DOE server.
- ▶ Port of the DOE server.
- ▶ User ID and password that are used to execute the API calls.

The following parameters are optional:

- ▶ If the plug-in accepts untrusted certificates; for example, self-signed (default is NO).
- ▶ If the full REST API request and response is printed in the log (default is NO).

Provision

The following input naming parameters are required:

- ▶ Application
- ▶ Environment where the new instance should be provisioned
- ▶ Team that is to own the newly provisioned instance
- ▶ Instance to be created

The API calls and flow includes the following steps:

1. Get the unique identifiers (UUIDs) for the application, environment, and team:

```
GET /policy/application/{appId}
GET /policy/environments/{envId}
GET /policy/teams/{teamId}
```

2. Provision the new instance:

```
POST /policy/instances
```

The following output parameters are required:

- ▶ `doe.schemaName`: The schema of the newly created instance
- ▶ `doe.dbName`: The database name of the newly created instance
- ▶ `doe.provisionStatus`: The completion status of the Provision task (success or failure)

Deprovision

This input parameter is the name of the instance to be deprovisioned.

To deprovision the instance, run:

```
DELETE /policy/instances/{instanceId}
```

The following output parameter results: `doe.deprovisionStatus`, which is the completion status of the Deprovision task (success or failure).

Analyze

The following input parameters are required:

- ▶ DOE instance name
- ▶ Application definition path (JSON file), as described in 6.8, “DOE setup for DevOps engineer” on page 52

API calls and flow includes the following steps:

1. Get the UUID for the instance:

```
GET /policy/instance/{instID}
```

2. Validate syntax and site rules for each object in the application definition (JSON file). Exit in case of syntax errors or site rule violations:

```
POST /site-rules/validate/db2
```

3. Save the object definition:

- a. If the object exists, update the definition:

```
PUT /policy/instances/{instanceId}/objects/def
```

- b. If the object does not exist, add an object to the instance:

```
PUT /policy/instances/{instanceId}/objects/def?objectName={objectId}
```

- c. If saving fails for any object, restore all object to the last committed state and exit:

```
PUT /policy/instances/{instanceId}/objects/restore
```

4. Request the instance changes report:

```
POST /policy/instances/{instanceId}/objects/reports
```

5. Receive the report:

```
GET /policy/instances/{instanceId}/reports/{jobId}
```

The following output parameters are required:

- ▶ `doe.analysisResult`: A report that lists the registered changes for each object
- ▶ `doe.analysisStatus`: The completion status of the Analyze task (success or failure)

Apply

Input parameter is the DOE instance name.

API calls and flow includes the following steps:

1. Get the UUID for the instance:

```
GET /policy/instance/{instID}
```

2. Apply the pending changes to the instance:

```
POST /policy/instances/{instanceId}/objects/apply
```

3. Get the apply report:

```
GET /policy/instances/{instanceId}/apply/{jobId}
```

4. If the apply failed for any object, discard the changes and restore the object definitions to the last committed state:

```
PUT /policy/instances/{instanceId}/objects/discard-apply
```

The following output parameters are required:

- ▶ `doe.applyResult`: A report that lists the changes that were applied to each object
- ▶ `doe.applyStatus`: The completion status of the Apply task (success or failure)

Promote

The following input parameters are required:

- ▶ Instance name
- ▶ Approver user ID
- ▶ Approver password

API calls and flow includes the following steps:

1. Get the instance UUID:
`GET /policy/instance/{instID}`
2. Create a pull request:
`POST /ws/policy/pull-requests/`
3. Approve the pull request:
`POST /policy/pull-requests/{pullRequestId}/approve`
4. If the pull request status is opened, request to merge it:
`POST /policy/pull-requests/{pullRequestId}/merge`
5. Check the pull request status until status is merged:
`GET /policy/pull-requests/{pullRequestId}`

The following output parameter is required:

`doe.promoteResult`:

This parameter is the completion status of the Promote task (success or failure).

Sync-up

The input parameter is the instance name.

API calls and flow includes the following steps:

1. Get the UUID for the instance:
`GET /policy/instance/{instID}`
2. Pull the object definitions from the application upstream:
`POST /policy/instances/{instanceId}/objects/pull`
3. Apply the changes as described in steps 2 to 4 of the **Apply** task (“Apply” on page 60):

The following output parameters are required:

- ▶ `doe.syncResult`: A report that lists the objects that were pulled from the master branch
- ▶ `doe.syncStatus`: The completion status of the Sync-up task (success or failure)

7.2.2 Installing the DevOps Experience plug-in

After you download the plug-in to your local system, complete the following steps to install it on the UCD server.

1. Log in to the UCD web interface by using an administrative user ID.
2. Navigate to **Settings** in the upper menu bar.
3. In the **Automation** pillar, click **Automation Plugins**.
4. Click **Load Plugin** on the right side.
5. In the pop-up window, click **Browse** and select the downloaded plug-in on your local system. Click **Submit** to upload and install the plug-in on the UCD server.

The DOE plug-in is displayed in the list of installed plug-ins (see Figure 7-3).

Plugin	Description
DOE	
DOE	The Db2 for z/OS DevOps Experience plug-in includes deployment activities using IBM® Db2 DevOps Experience
Items per page: 10 1-1 of 1 item	

Figure 7-3 Installing the DOE plug-in

7.2.3 Configuring UrbanCode Deploy

The next step is to define and configure in UCD elements that are required to deploy Db2 schema changes by using the DOE plug-in.

For the UCD configuration that is related to the COBOL application, follow the instructions in [Build a pipeline with Jenkins, Dependency Based Build, and UrbanCode Deploy](#).

For more information about configuring UCD elements, see the *Modeling software deployment* chapter in the [UCD documentation](#).

The examples and figures in this section might not represent a complete configuration of all UCD elements. For more information about downloading a complete snapshot of the UCD elements, see Appendix A, “Additional material” on page 111.

7.2.4 Creating roles and teams

Before you configure the deployment, roles and teams must be defined that control access to applications and environments in UCD.

A *role* is a set of granted permissions and does not impart its granted permissions to any specific user. Roles and their associated permissions are applied to users or groups by adding them to teams.

You can configure roles and teams under **Settings** → **Security**. For our demonstration environment, we configured a set of DBA users called DOEDBA, DOEDBA1, and DOEDBA2. We also created a group that is called DOEDBA and added the users to the group.

Next, we added a role that is called DOEDBA. Finally, we created a team that is called DOEDBA and added the users and group to the team, as shown in Figure 7-4. We use the DOEDBA team in the manual approval step and when configuring notifications as described in 7.2.13, “Configuring notifications” on page 83.

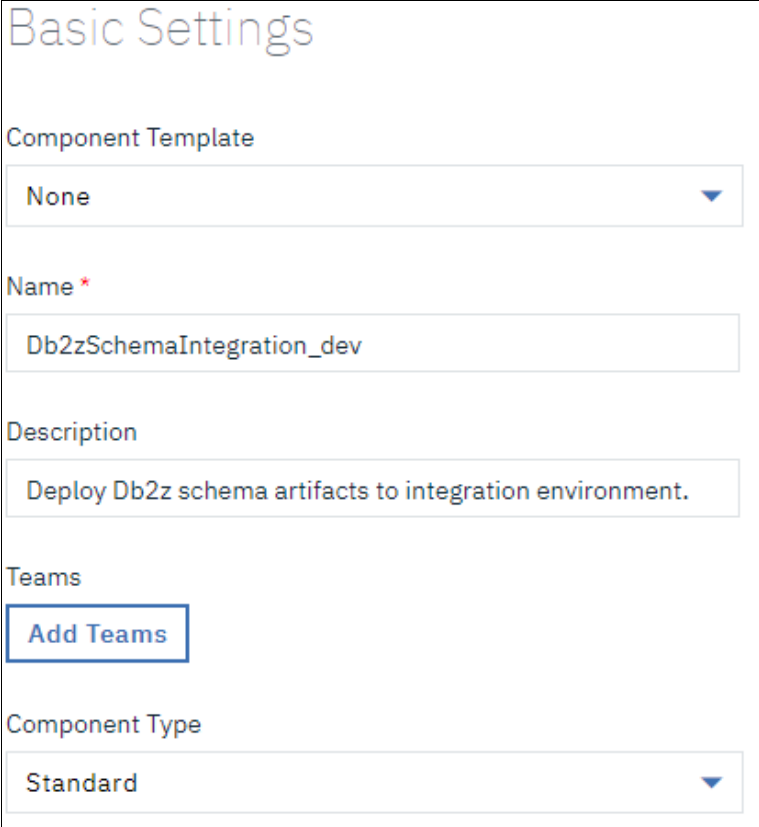
The screenshot shows the configuration page for a team named 'DOEDBA'. At the top, there is a 'Name' field containing 'DOEDBA' and a 'Save' button. Below this is the 'Role Members' section. It contains two expandable panels. The first panel, 'Administrator', has an 'Add' button and an 'Edit' button. Below it, the 'Users' list contains 'admin' and the 'Groups' list is empty. The second panel, 'DOEDBA', also has 'Add' and 'Edit' buttons. Below it, the 'Users' list contains 'DOEDBA', 'DOEDBA1', and 'DOEDBA2', and the 'Groups' list contains 'DOEDBA'.

Figure 7-4 Creating users, groups, and teams

7.2.5 Creating the component

Complete the following steps to create a \ component:

1. Navigate to the **Components** tab of the upper menu bar.
2. Click **Create Component**.
3. In the pop-up window, complete the required fields. Figure 7-5 shows the sample configuration that was used for the demonstration.



The screenshot shows a 'Basic Settings' form with the following fields and controls:

- Component Template:** A dropdown menu with 'None' selected.
- Name *:** A text input field containing 'Db2zSchemaIntegration_dev'.
- Description:** A text input field containing 'Deploy Db2z schema artifacts to integration environment.'
- Teams:** A section containing an 'Add Teams' button.
- Component Type:** A dropdown menu with 'Standard' selected.

Figure 7-5 Component basic settings

4. In the Version Source Configuration section, select **Git** as the source configuration type and enter the repository URL, branch, username, and password. Whenever a change is committed to the specified repository, a component version is automatically created in UCD (see Figure 7-6).

Version Source Configuration

Source Configuration Type

Git

Repository URL *

https://github.ibm.com/IBMZSoftware/nazare-demo-genapp

Branch

db2dev_integration

Username ⓘ

sueli@us.ibm.com

Password ⓘ

....

☐ Watch for Tags

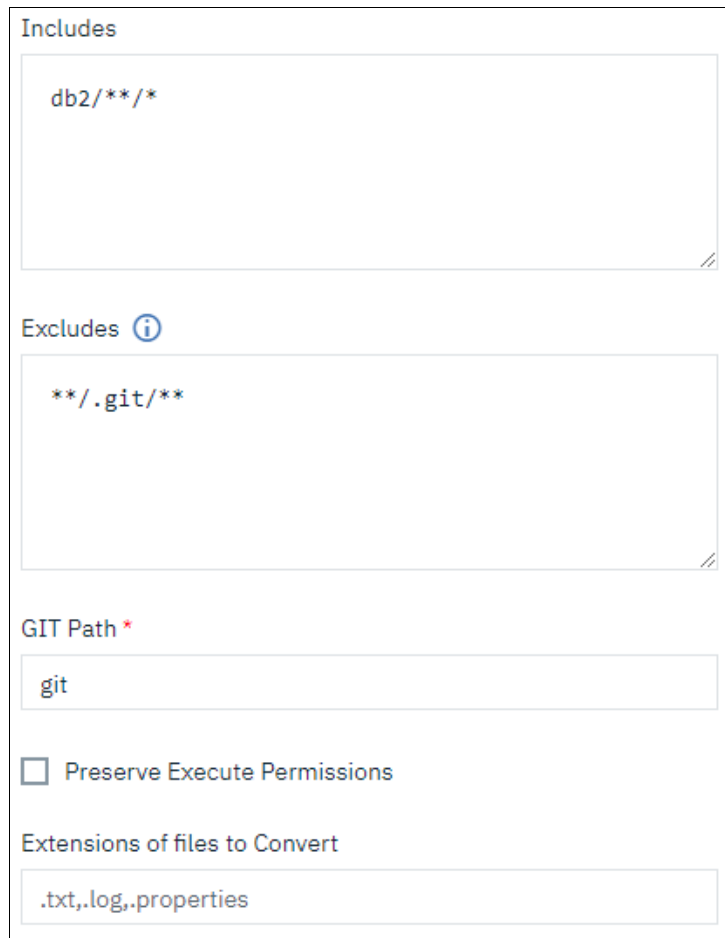
☐ Recursive

☐ Disable SSL Verification

Figure 7-6 Version Source Configuration window

5. In the Includes field, specify `db2/**/*` so that only the content of the `db2` directory of the repository is part of the component.

In the GIT Path field, specify the path to the Git executable on the UCD server. If you added the Git executable to the system PATH, you can specify the executable name as `git` (see Figure 7-7).



The screenshot shows a configuration form with the following sections:

- Includes**: A text area containing the pattern `db2/**/*`.
- Excludes**: A text area containing the pattern `**/.git/**`, with an information icon to its left.
- GIT Path ***: A text field containing the value `git`.
- Preserve Execute Permissions**: An unchecked checkbox.
- Extensions of files to Convert**: A text field containing the list `.txt,.log,.properties`.

Figure 7-7 Specifying the executable name

6. Select **Copy to CodeStation** to copy artifacts from the specified source to the UCD server's repository from which they can be retrieved during deployments (see Figure 7-8).

☐ Import Versions Automatically

☒ Copy to CodeStation

Default Version Type *

Full

Import Configuration

☒ Use the system's default version import agent/tag.

☐ Import new component versions using a single agent.

☐ Import new component versions using any agent with the specified tag.

Cleanup Configuration

☒ Inherit Cleanup Settings

Figure 7-8 Copying artifacts from the specified course to the UCD

7. Click **Save**.
8. After you create the component, select it from the list of all components and click the **Configuration** tab. From the menu on the left side, select **Environment Property Definitions**. From here, you can define component properties for each environment where the component is to be deployed. Figure 7-9 shows the properties that we defined in our demonstration.

Environment Property Definitions ⓘ

Version 1 of 1

[Add Property](#)

Index	Name	Label	Pattern	Required	Default Value	Description	Is Inherited
	<input type="text" value="Filter"/>	<input type="text" value="Filter"/>			<input type="text" value="Filter"/>	<input type="text" value="Filter"/>	
0	doeAppDefinition	doeAppDefinition		true			false
1	doeInstanceName	doeInstanceName		true			false
2	srcDirectory	srcDirectory		false			false
3	doeServerHost	doeServerHost		true			false
4	doeServerPort	doeServerPort		true			false
5	doeUsername	doeUsername		true			false
6	doePassword	doePassword		true	****		false

Items per page: 10 | 1-7 of 7 items

1 of 1 pages < Previous 1 Next >

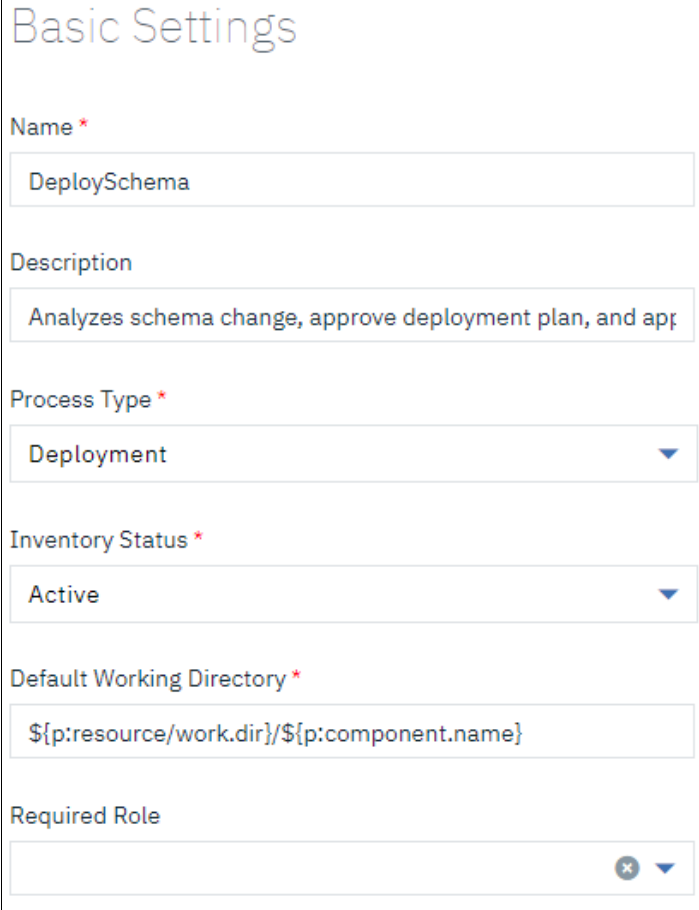
Figure 7-9 Properties defined in the demonstration

7.2.6 Creating the component deployment process

After you create the component for the database schema definition, complete the following steps to create the deployment process:

1. Navigate to the **Components** tab in the top menu bar and select the previously created component from the list of all components.
2. Click the **Processes** tab and then, click **Create Process**.

3. Complete the required fields in the pop-up window; then, click **Save**. The Process Designer opens in which you can begin to model the deployment process (see Figure 7-10).



The image shows a 'Basic Settings' window with several input fields. The 'Name' field is required and contains 'DeploySchema'. The 'Description' field contains 'Analyzes schema change, approve deployment plan, and ap'. The 'Process Type' dropdown is set to 'Deployment'. The 'Inventory Status' dropdown is set to 'Active'. The 'Default Working Directory' field contains the placeholder text '\${p:resource/work.dir}/\${p:component.name}'. The 'Required Role' field is empty and has a dropdown arrow.

Field	Value
Name *	DeploySchema
Description	Analyzes schema change, approve deployment plan, and ap
Process Type *	Deployment
Inventory Status *	Active
Default Working Directory *	\${p:resource/work.dir}/\${p:component.name}
Required Role	

Figure 7-10 Basic Settings window

A list of predefined actions is displayed on the left side. If you installed the UCD DOE plug-in as described in 7.2.2, “Installing the DevOps Experience plug-in” on page 62, the tasks in the Db2zOS / DevOpsExperience entry in the Database section is available, as shown in Figure 7-11.

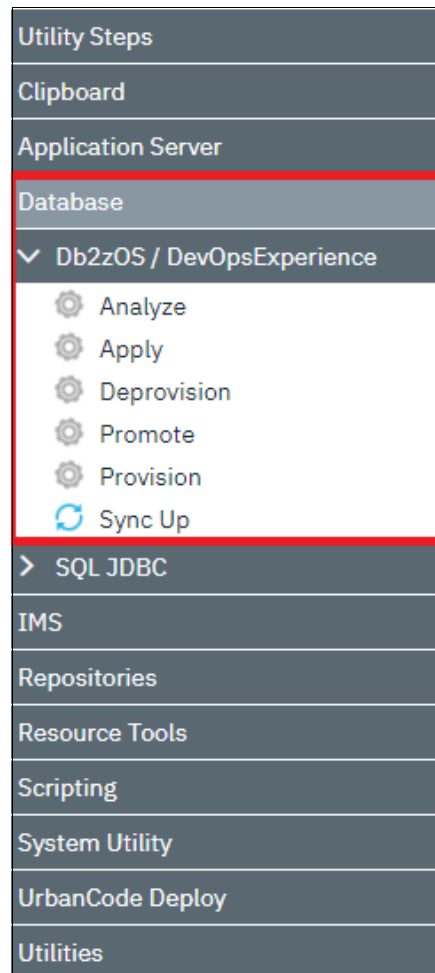


Figure 7-11 Db2zOS / DevOpsExperience option available

You can use these tasks to model the deployment process according to the sample that is shown in Figure 7-12. The steps of the process are described next.

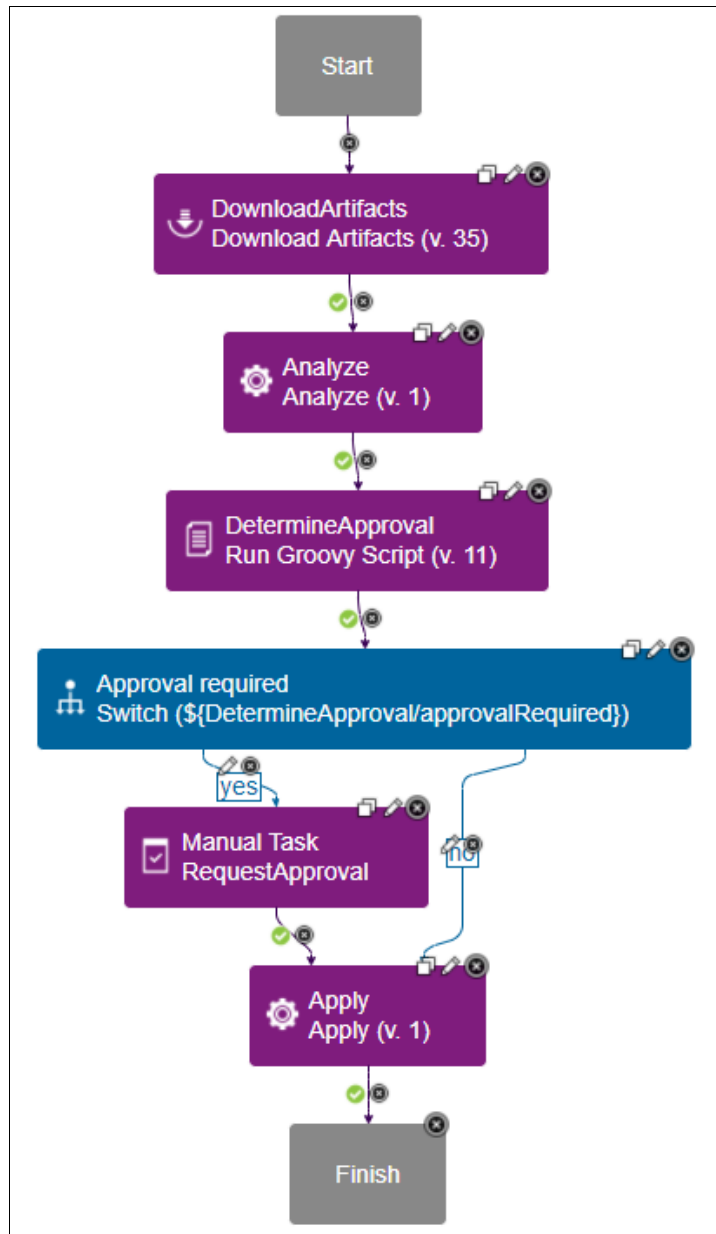


Figure 7-12 Model deployment tasks

The process includes the following overall tasks (as shown in Figure 7-12):

1. DownloadArtifacts

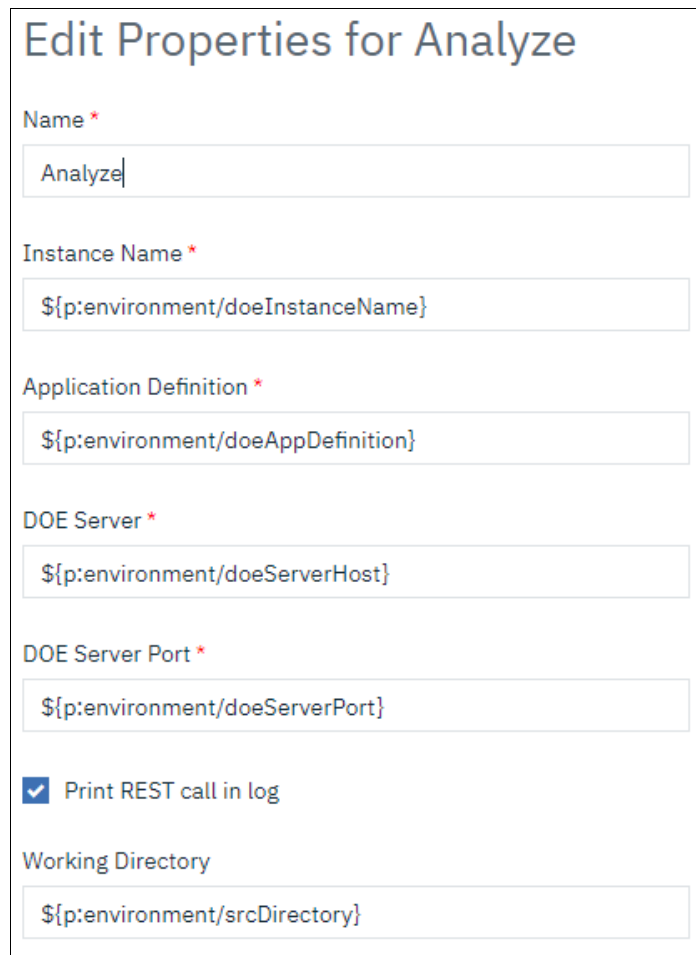
The first step in the deployment process is to download the specified version of the component artifacts to the agent's working directory. When you run the process, you specify which version of the component artifacts to use. The corresponding task Urban Code Deploy / Download Artifacts is in the Artifacts section under Repositories. No customization is required for this task.

2. Analyze

This task that is provided by the UCD DOE plug-in is used to read the DDL from the working directory, validate syntax and site rules, and register the new object definitions in DOE.

The task requires the DOE instance name where the DDL changes are to be deployed, the path to the application definition (JSON file), and the DOE server name and port. You can specify concrete values for these parameters or you can use environment properties.

In our demonstration, we use the `${p:environment/<property name>}` notation to point to environment properties, as shown in Figure 7-13. The environment properties are mapped to their specific values later.



Edit Properties for Analyze

Name *
Analyze

Instance Name *
\${p:environment/instanceName}

Application Definition *
\${p:environment/appDefinition}

DOE Server *
\${p:environment/serverHost}

DOE Server Port *
\${p:environment/serverPort}

☒ Print REST call in log

Working Directory
\${p:environment/srcDirectory}

Figure 7-13 Edit Properties for Analyze window

3. DetermineApproval

The output the Analyze task is a JSON report that contains information about the schema changes (for example, ALTER) that is to be run. This step consists of a Groovy script that parses the JSON report and determines if disruptive changes exist that require manual approval.

You can model the step by using the Run Groovy Script task that is listed in the Groovy section under Scripting in the task palette.

The Groovy script code is shown in Example 7-1.

Example 7-1 Run Groovy Script task

```
import groovy.json.JsonSlurper

// recursive loop through the summary report JSON
def loop(array) {
    def count = 0
    array.each { entry ->
        println(entry["Object type"] + " " + entry["Source Object"] + ": " +
entry["Result"])
        if(entry["Result"] == "Dropped/created")
            count++
        if(entry["Children"].size() > 0)
            count += loop(entry["Children"])
        }
    }
    return count
}

// get the summary report JSON from the previous step
def summaryReport = '${Analyze/does.analysisSummaryReport}'

// parse the report
def parsedJson = new JsonSlurper().parseText(summaryReport)

// loop through the report
def c = loop(parsedJson["Results"])

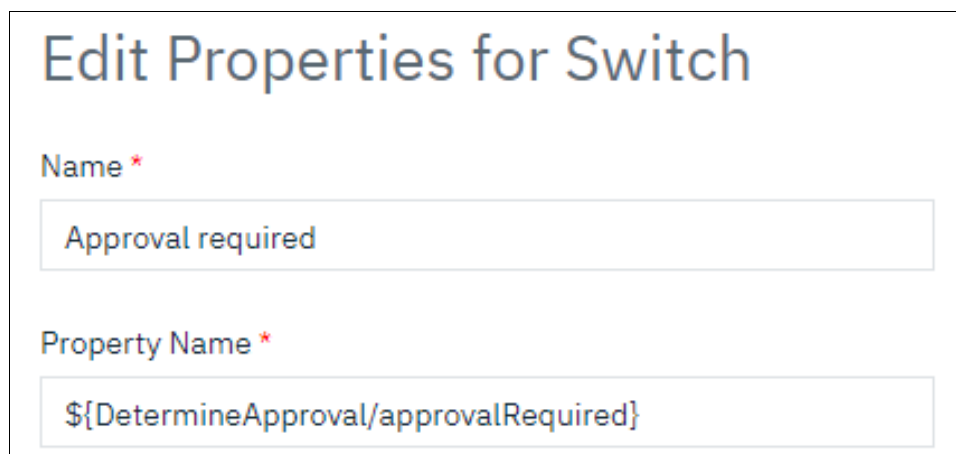
// determine, if approval is required
def approvalRequired
if(c > 0)
    approvalRequired = "yes"
else
    approvalRequired = "no"
println("Approval required: ${approvalRequired}")

// set the output properties of the step
outProps.put("approvalRequired", approvalRequired)
outProps.put("Status", "Success")
outProps.put("exitCode", "0")
```

The script loops through the hierarchy of objects and checks the `Result` field. In our demonstration, we set the `approvalRequired` output variable to `yes` if the value of `Result` is `Dropped/Created` for any of the objects. For all other values (for example, `Altered` or `No change`), we set `approvalRequired` to `no`. This handling easily can be changed in the script according to your requirements.

4. Approval required

In this step, the approvalRequired output variable of the previous step is evaluated. You can implement it by using the Switch step that is listed under Utility Steps. You must edit the step properties and complete the Property Name that must be evaluated (see Figure 7-14).



Edit Properties for Switch

Name *

Approval required

Property Name *

`${DetermineApproval/approvalRequired}`

Figure 7-14 Edit Properties for Switch window

You also must draw the links from this step to the following steps and edit the condition for each link. If approvalRequired is yes, the manual approval step is called next. If approvalRequired is no, the Apply step is called next (see Figure 7-15).

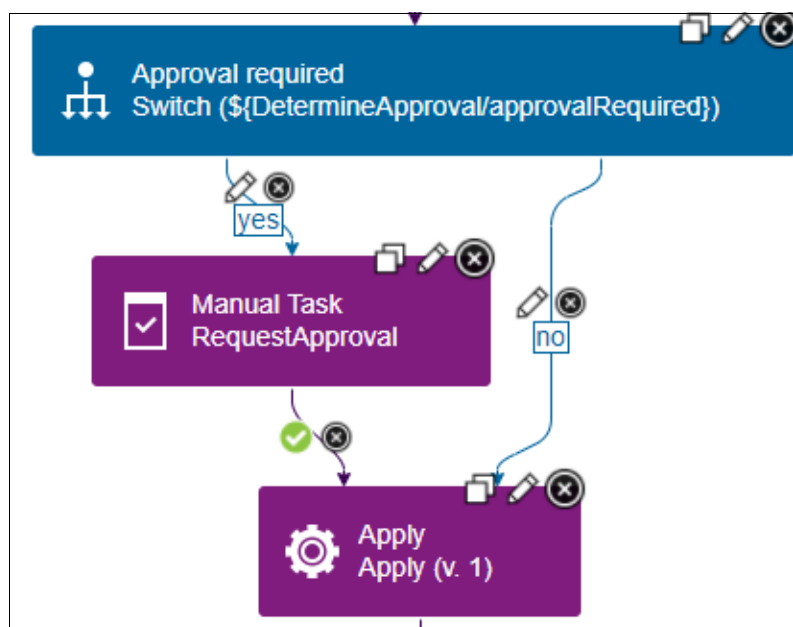


Figure 7-15 Creating the component deployment process

5. RequestApproval

For the manual approval, UCD provides the Manual Task step under Utility Steps. For the demonstration, we customized the manual approval step to create a task in UCD that must be approved by a member of the DOEDBA team.

We also added the summary report of changes that is generated by the Analyze step as a property to the approval task. The demonstration configuration is shown in Figure 7-16.

Edit Properties for Manual Task

Name *

RequestApproval

Notification Template

ApprovalCreated

Properties

[Add Property](#)

Name	Label	Pattern	Required	Default Value	Description	Is Inherited
Filter	Filter			Filter	Filter	
doeAnalysisReport	DOE Analysis Report		false	\${Analyze/doe.analysis Result}		false

Items per page: 10 | 1-1 of 1 item 1 of 1 pages < Previous 1 Next >

[Refresh](#) [Print](#)

☐ Only Allow Deploying User To Complete

Who can approve this task

Role Member By Application

DOEDBA

for

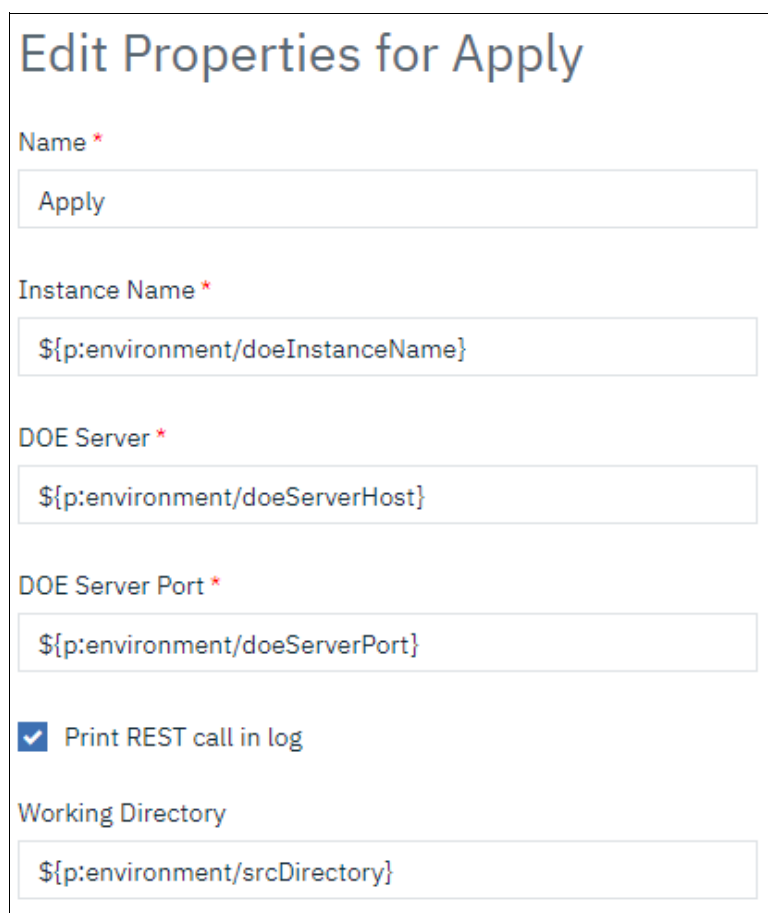
Standard Application

Figure 7-16 Editing Properties for Manual Task window

6. Apply

The final step in the deployment process is the Apply that is provided by the UCD DOE plug-in. This step materializes the registered changes to the Db2 for z/OS schema.

You must customize the DOE instance name server and port. You can specify those parameters directly in the step or you can define them as environment properties (see Figure 7-17).



Edit Properties for Apply

Name *

Apply

Instance Name *

`${p:environment/instanceName}`

DOE Server *

`${p:environment/instanceHost}`

DOE Server Port *

`${p:environment/instancePort}`

☒ Print REST call in log

Working Directory

`${p:environment/srcDirectory}`

Figure 7-17 Edit Properties for Apply window

Finally, save the process.

7.2.7 Creating the resource

To deploy the created component on a target environment, you must create a resource that associates UCD agents with components. For this step, you need a UCD agent on z/OS that was configured and started. These instructions are beyond the scope for this document.

For more information, see the following web pages:

- ▶ [IBM Developer](#)
- ▶ [UCD documentation](#)

Figure 7-18 shows the resource configuration of our demonstration.



Figure 7-18 Resource configuration of our demonstration

7.2.8 Creating the application

Complete the following steps:

1. Continue creating the application by clicking the **Applications** tab in the top menu.
2. Click **Create Application**, and select **New Application**. Complete the required fields. Figure 7-19 on page 78 shows the application definition in our demonstration. Although the figure contains a notification scheme, that scheme can be ignored for now and is left it blank. For more information about configuring notifications, see 7.2.13, “Configuring notifications” on page 83.



Basic Settings

Name *

Db2zDeploy_dev

Description

Teams

 DOEDBA	×
 IMS	×

Add Teams

Notification Scheme

Db2NazareDBA ▼

☐ Enforce Complete Snapshots

Figure 7-19 Creating the application

7.2.9 Associating the component with the application

Complete the following steps:

1. Select the previously created application from the list of all applications and click the **Components** tab.
2. Click **Add Component**, select the previously created component, and click **Save**.
Figure 7-20 shows the components that we added to the application in our demonstration.

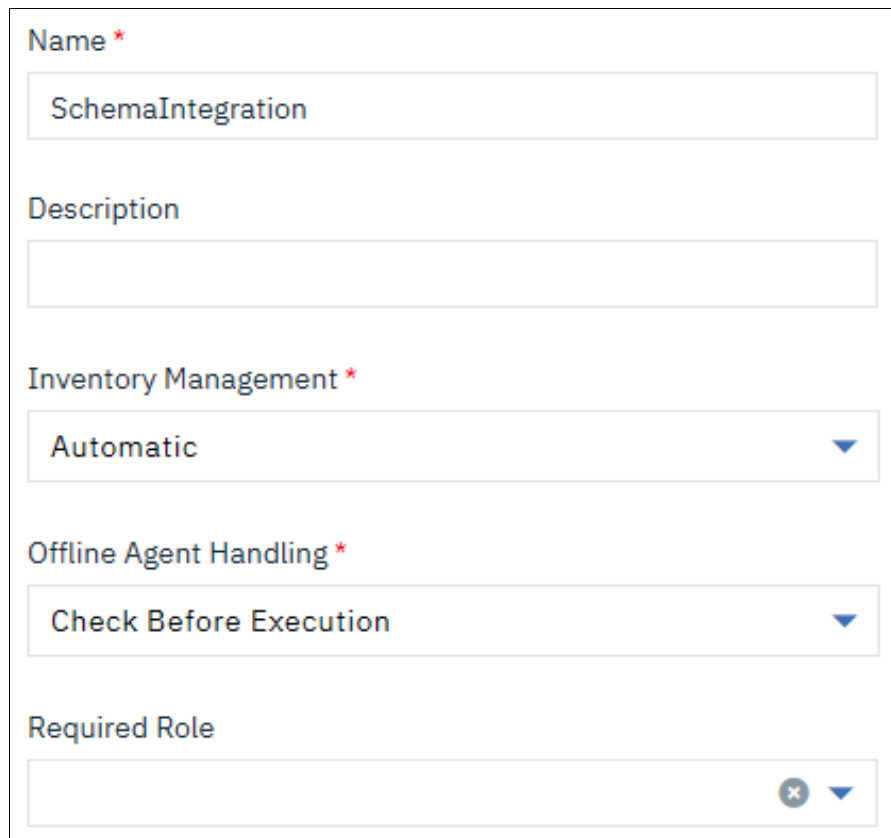
Name	Last Import	Last Version	Description
▽ Name			
Db2zSchemaGoldenMaster_dev	Successful	2020-12-10T06:51:48.051	Deploy Db2z schema artifacts to gold
Db2zSchemaIntegration_dev	Successful	2021-06-16T04:53:39.053	Deploy Db2z schema artifacts to integ

Figure 7-20 Associating the component with the application

7.2.10 Defining the process to install the component

Complete the following steps:

1. Select the application and browse to the **Processes** tab.
2. Click **Create process**, complete the required fields in the pop-up window and then, click **Save**. Figure 7-21 shows the process configuration of our demonstration.



The screenshot shows a configuration form for a process. It contains the following fields and options:

- Name ***: A text input field containing "SchemaIntegration".
- Description**: An empty text input field.
- Inventory Management ***: A dropdown menu with "Automatic" selected.
- Offline Agent Handling ***: A dropdown menu with "Check Before Execution" selected.
- Required Role**: An empty text input field with a clear button (X) and a dropdown arrow.

Figure 7-21 Demonstration process configuration

3. After you save the process configuration, the Process Designer window opens. Use the Install Component step from the Application Steps section to model the process, as shown in Figure 7-22.

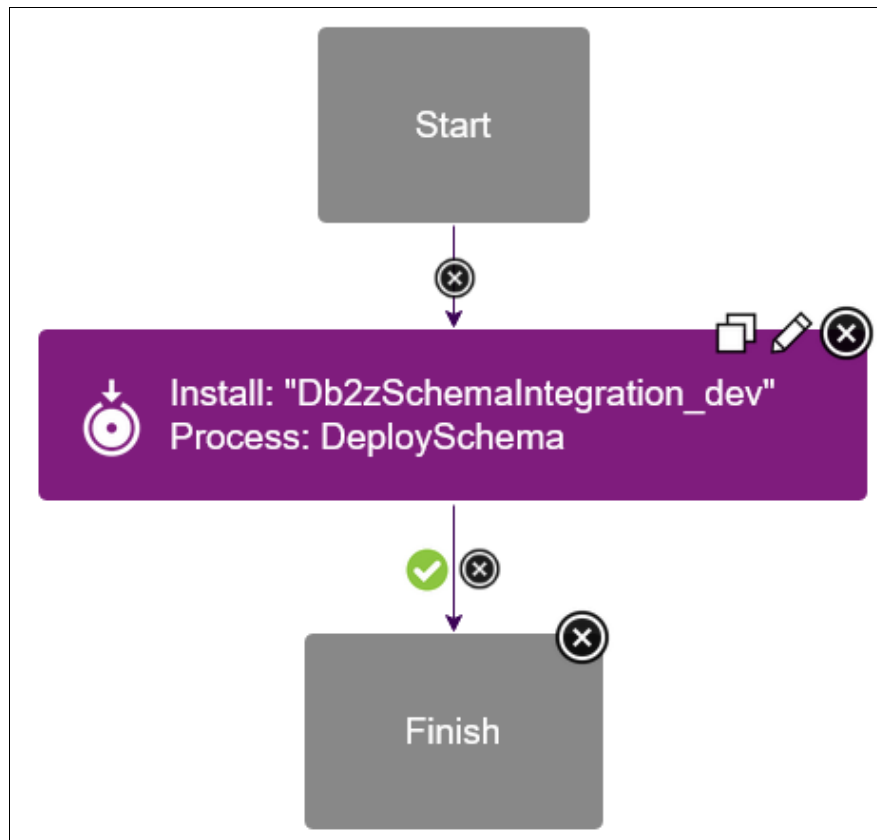


Figure 7-22 Install Component step

4. Customize the step by specifying the Component Name and Component Process values. Figure 7-23 shows the configuration in our demonstration environment.

Edit Properties for Install Component

Name *
Install: "Db2zSchemaIntegration_dev"

Component *
Db2zSchemaIntegration_dev

Use Versions Without Status *
Active

Component Process *
DeploySchema

Limit to Resource Tag
x

Maximum number of concurrent processes *
-1

☐ Fail Fast

☐ Ignore Child Warnings

Precondition
1

Figure 7-23 Edit Properties for Install Component window

7.2.11 Creating the environment for the application

Complete the following steps:

1. Select the application and navigate to the **Environments** tab.
2. Click **Create Environment** and complete the required fields in the pop-up window.

For our demonstration, we created two environments: Integration and GoldenMaster as shown in Figure 7-24.

> Integration	Snapshot: None	● Inventory: 1 / 1
> GoldenMaster	Snapshot: None	● Inventory: 0 / 0

Figure 7-24 Creating the environment for the application

Each environment is mapped to an individual DOE instance by setting environment properties.

- To set the properties, click the environment; then, go to the **Configuration** tab and select **Environment Properties** from the menu on the left side. A list of properties that you can set for the environment is displayed in the Component Environment Properties section. Figure 7-25 shows the configuration in our demonstration environment.

Figure 7-25 Component Environment Property window

7.2.12 Assigning the resources to the environment

Finally, complete the following steps to assign resources to the environments where the application can be deployed:

- Click the environment to display a list of assigned resources. The list should be empty.
- Click **Add Base Resource** and select the previously created resource. Figure 7-28 on page 84 shows the configuration in our demonstration environment.

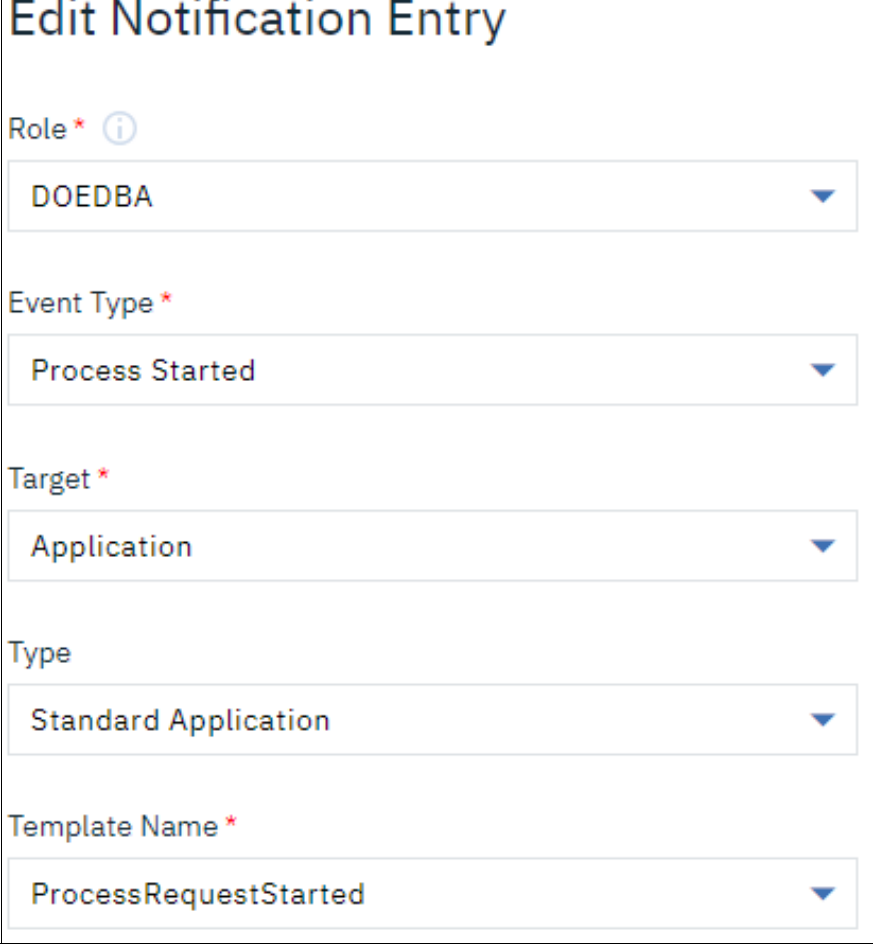
<div> <div>Show</div> <div>Bulk Actions</div> <div>Select All...</div> <div>Add Base Resources</div> <div>Customize Display</div> </div>				
Name	Tags	Inventory	Status	Description
<div> <div>Name</div> <div>TIVLP02_db2dev / tivlp02.svl.ibm.com (View Agent) / Db2zSchemaIntegration_dev</div> </div>	<div> <div>Tags</div> <div></div> </div>	<div> <div>Filter</div> <div>2021-06-17T05:15:48.015</div> </div>		

Figure 7-26 Assigning the resources to the environment

7.2.13 Configuring notifications

If you want to be informed about events that are related to the deployment, complete the following steps to such configure notifications:

1. In the **System** pillar, click **Settings** → **Notification Schemes**.
2. Click **Create Notification Scheme** and complete the required fields in the pop-up window.
3. Click **Add Notification Entry** to add notifications to the scheme.
4. Complete the required fields in the pop-up window. Figure 7-27 shows an example for a notification when a deployment process is started.



The screenshot shows a web form titled "Edit Notification Entry". It contains five dropdown menus, each with a red asterisk indicating it is a required field. The first dropdown is labeled "Role" and has an information icon; it is set to "DOEDBA". The second is labeled "Event Type" and is set to "Process Started". The third is labeled "Target" and is set to "Application". The fourth is labeled "Type" and is set to "Standard Application". The fifth is labeled "Template Name" and is set to "ProcessRequestStarted".

Figure 7-27 Configuring notifications

For Role, we selected the DBA role that we created in 7.2.4, “Creating roles and teams” on page 63. For the templates, we used the default templates that are provided by UCD.

For more information about customizing notification templates, see [this web page](#).

Figure 7-28 shows all notifications that we added to the scheme in our demonstration, including notifications on process start, start error, failure and success, and approval failure.

Notification Scheme: Db2NazareDBA

Description
b2 Nazare DBA Notifications

main Edit

Notification Entries Add Notification Entry

Type	Role	Template	
Process Failure	Application / DOEDBA	ApplicationDeploymentFailure	⋮
Process Started	Application / DOEDBA	ProcessRequestStarted	⋮
Approval Failed	Application / DOEDBA	ApprovalFailed	⋮
Process Success	Application / DOEDBA	ApplicationDeploymentSuccess	⋮
Process Not Started	Application / DOEDBA	ProcessNotStarted	⋮

Figure 7-28 Notifications added to demonstration's scheme

- To enable notifications for an application, go to the application and click the **Configuration** tab. Under Notification Scheme, select the created notification scheme from the drop-down list, as shown in Figure 7-29.

Basic Settings

Name *

Db2zDeploy_dev

Description

Teams

DOEDBA x

IMS x

Add Teams

Notification Scheme

Db2NazareDBA ▼

☐ Enforce Complete Snapshots

Figure 7-29 Selecting the notification scheme



Jenkins automation server

This chapter introduces you to the basic concepts of the Jenkins automation server and describes how you integrate the previously developed UCD process into the pipeline.

This chapter includes the following topics:

- ▶ 8.1, “Introduction” on page 86
- ▶ 8.2, “Resources for installing and configuring Jenkins” on page 86
- ▶ 8.3, “GenApp pipeline stages overview” on page 87

8.1 Introduction

Jenkins is an open source automation server that can be used to automate many tasks that are related to building, testing, and delivering or deploying software. For more information about Jenkins downloads and documentation, see [this website](#).

A core component of Jenkins is Pipeline, which supports implementing and integrating continuous delivery pipelines into Jenkins. A CD pipeline is an automated expression of your process for getting software from version control directly to your users and customers.

Every change to your software that is committed in source control goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, and progressing the built software (called a *build*) through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple to complex delivery “pipelines as code” by way of the Pipeline domain-specific language (DSL) syntax.

The definition of a Jenkins Pipeline is written into a text file (called a *Jenkinsfile*) which in turn can be committed to a project’s source control repository. This capability is the foundation of “Pipeline-as-code” because it enables the CD pipeline to be treated as a part of the application to be versioned and reviewed like any other code.

8.2 Resources for installing and configuring Jenkins

Instructions for installing and configuring Jenkins are beyond the scope of this document. For more information about installing instructions, see the following web pages:

- ▶ [Jenkins documentation](#)
- ▶ [hIBM Developer](#)

For our demonstration, the Jenkins pipeline is configured to run automatically when a change is pushed to the repository.

8.3 GenApp pipeline stages overview

The GenApp pipeline that we use for our demonstration implements the stages for the application build and deployment that are shown in Figure 8-1.

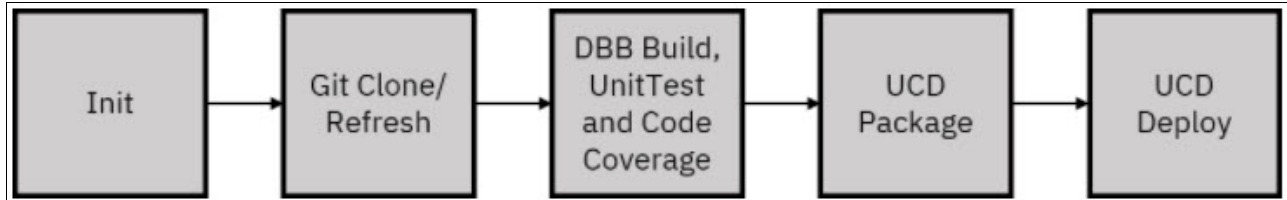


Figure 8-1 GenApp pipeline

8.3.1 GenApp pipeline stages

The GenApp pipeline includes the following stages:

1. Init
Initialize values that are used throughout the pipeline; for example, build type (commit, merge, or pull request), the Jenkins agent or environment properties.
2. Git Clone/Refresh
Get a fresh local copy of the content of the source code repository.
3. DBB Build, UnitTest, and Code Coverage
Run DBB and zUnit to compile the COBOL source code into executable load modules and test them.
4. UCD Package
Push a new component version of the COBOL load modules and DBRMs to UCD.
5. UCD Deploy
Run the UCD deployment process for the new COBOL component.

8.3.2 Extending the pipeline with Db2 for z/OS schema changes

Including Db2 for z/OS schema changes in the pipeline requires the addition of two stages, Db2:Init and Db2:Deploy, as shown in Figure 8-2. These stages are functionally similar to stages 4 and 5. In the sections that describe these two new stages, only parts of the pipeline code are listed. For more information about instructions for downloading the complete pipeline code, see Appendix A, “Additional material” on page 111.

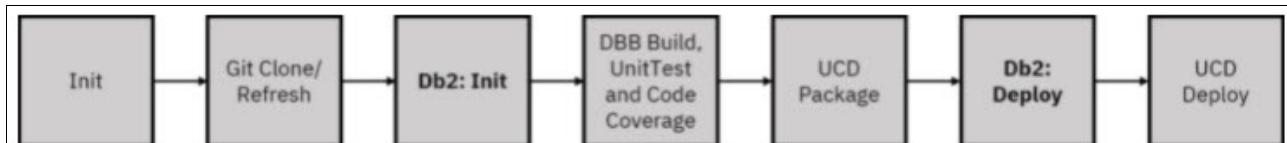


Figure 8-2 Extending the pipeline with Db2 for z/OS schema changes

Db2:Init stage

The first stage that you must add to the pipeline initializes and sets parameters for the Db2 schema deployment. The parameters all relate to the definitions that you created in UCD. The values depend on the type of build that triggered the pipeline.

In the case of a commit, the pipeline deploys to the integration environment. In the case of a merge or pull request, the pipeline deploys to the golden master environment. Example 8-1 shows how to set these parameters for both cases.

Example 8-1 Db2 init stage sample code

```
// Set deployment properties based on build type.
if (!isPullOrMergeRequest()) {
    db2DeployProps = [
        ucdApplication: 'Db2zDeploy_dev',
        ucdEnvironment: 'Integration',
        ucdComponentSchema: 'Db2zSchemaIntegration_dev',
        ucdApplicationProcessSchema: 'SchemaIntegration',
    ]
}
// Deploy to Golden Master, if Pull or Merge Request
else {
    db2DeployProps = [
        ucdApplication: 'Db2zDeploy_dev',
        ucdEnvironment: 'GoldenMaster',
        ucdComponentSchema: 'Db2zSchemaGoldenMaster_dev',
        ucdApplicationProcessSchema: 'SchemaGM',
    ]
}
}
```

8.3.3 Db2:Deployment stage

The second stage that you must add to the pipeline pushes a new component version to UCD and runs the deployment process. You can use the UCD plug-in to run both tasks.

Example 8-2 shows the code to push a new component version to UCD. It uses the DDL in the agent's copy of the repository as input.

Example 8-2 Db2 deployment stage 1

```
// Push artifacts to UCD.
step([
    $class: 'UCDeployPublisher',
    siteName: 'THINKDEMO',
    component: [
        $class:
'com.urbancode.jenkins.plugins.ucdeploy.VersionHelper$VersionBlock',
        componentName: db2DeployProps.ucdComponentSchema,
        delivery: [
            $class: 'com.urbancode.jenkins.plugins.ucdeploy.DeliveryHelper$Push',
            pushVersion: currentTime.format("yyyy-MM-dd'T'hh:mm:ss.mmm"),
            baseDir: "${WORKSPACE}/nazare-demo-genapp/db2",
            fileIncludePatterns: '',
            fileExcludePatterns: '',
            pushProperties: "JenkinsBuildURL=${BUILD_URL}",
            pushDescription: 'Pushed from Jenkins'
        ]
    ]
])
```

Example 8-3 shows the code to run the deployment process in UCD. It uses the values that you set in the db2DeployProps array in the Db2:Init stage.

Example 8-3 Db2 deployment stage 2

```
// Deploy UCD application process.
step([
    $class: 'UCDeployPublisher',
    siteName: 'THINKDEMO',
    deploy: [
        $class: 'com.urbancode.jenkins.plugins.ucdeploy.DeployHelper$DeployBlock',
        deployApp: db2DeployProps.ucdApplication,
        deployEnv: db2DeployProps.ucdEnvironment,
        deployProc: db2DeployProps.ucdApplicationProcessSchema,
        deployVersions: db2DeployProps.ucdComponentSchema + ':latest',
        deployOnlyChanged: false
    ]
])
```

After you add the two Db2-related stages to the pipeline, you are ready to run your first integrated COBOL and Db2 for z/OS change.



Application development tools for z/OS

The solution that is described in this document can be applied to any integrated development environment (IDE) that works with a source code management (SCM) system.

One of today's most commonly used IDEs for developing Db2 for z/OS applications is IBM Developer for z/OS (IDz). IDz is a robust toolset for developing and maintaining IBM z/OS applications and offers COBOL, PL/I, High Level Assembler, C/C++, JCL, and Java development tools on an Eclipse base. With the Enterprise Edition of IDz, developers can choose Microsoft VS Code or Red Hat CodeReady Workspaces for their daily z/OS development work.

Another popular IDE for developing Db2 for z/OS applications is Microsoft Visual Studio Code (VS Code). With a large and growing number of extensions available on the VS Code marketplace that support various languages and tasks, such as COBOL, C/C++, Java, JavaScript, Python, Rust, RESTful APIs, Docker, and Jenkins, VS Code's popularity continues to increase. Developers can easily work on different tasks without leaving the VS Code environment.

IBM publishes a pair of VS Code extensions to simplify the development of applications for z/OS and Db2 for z/OS: IBM Db2 for z/OS Developer Extension and IBM Z Open Editor. IBM Db2 for z/OS Developer Extension provides language support for the SQL syntax that is used to define, manipulate, and control data in IBM Db2 for z/OS databases. IBM Z Open Editor provides language support for the IBM Enterprise programming languages for z/OS.

This chapter discusses the Application development tools for z/OS and includes the following topics:

- ▶ 9.1, "IBM Db2 for z/OS Developer Extension introduction" on page 92
- ▶ 9.2, "IBM Z Open Editor introduction" on page 93
- ▶ 9.3, "Installing Visual Studio Code" on page 93
- ▶ 9.4, "Setting up a code repository in your development environment" on page 95
- ▶ 9.5, "Data modeling consideration" on page 97

9.1 IBM Db2 for z/OS Developer Extension introduction

IBM Db2 for z/OS Developer Extension provides language support for the SQL syntax that is used to define, manipulate, and control data in IBM Db2 for z/OS databases.

Db2 Developer Extension simplifies the task of developing applications that interact with data in Db2 for z/OS databases by providing basic productivity features, such as the following examples:

- ▶ SQL formatting
- ▶ Code completion and signature help
- ▶ Syntax checking and highlighting
- ▶ Customizable code snippets

It includes a robust set of capabilities for running SQL, including the ability to perform the following tasks:

- ▶ Display consolidated results from running multiple SQL statements
- ▶ Run SQL that includes parameters and variables from within a native stored procedure (.spsql file).
- ▶ Sort query history by the timestamp of the execution.
- ▶ Select multiple SQL elements on different lines and run those elements as a complete statement.
- ▶ Restrict the number of rows in SQL result sets.
- ▶ Quickly identify and display failing SQL statements.
- ▶ Use null values and retain input values.
- ▶ Validate XML for host variable parameters input.
- ▶ Run selected SQL statements from any type of file.
- ▶ Run SQL with or without parameter markers and host variables and to save SQL execution results.
- ▶ Commit or rollback the results of SQL executions that are based on customizable success or failure settings.

It also includes the following abilities for working with stored procedures:

- ▶ Deploy, run, and debug native stored procedures
- ▶ Set conditional breakpoints when debugging stored procedures

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

9.2 IBM Z Open Editor introduction

IBM Z Open Editor provides language support for the IBM Enterprise programming languages for z/OS. It supports COBOL 6.3, PL/I 5.3, and High Level Assembler for z/OS 2.4. This support also includes capabilities for embedded statements for CICS 5.6, IMS 15.1.0, and IBM SQL DB2® for z/OS 12.1. Earlier versions of any of these components also work.

IBM Z Open Editor enables IBM Z developers to use many features, including the following examples:

- ▶ Real-time syntax checking and highlighting while you type.
- ▶ Problems view with all syntax errors and (in COBOL) unreachable code.
- ▶ Outline view and outline search.
- ▶ For variables and paragraphs:
 - Declaration hovers
 - Peek definition
 - Go to definition
 - Find all references
- ▶ Completing code and variables.
- ▶ Finding and navigating references.
- ▶ Previewing of included copybooks and include files and assembler macros.
- ▶ Navigating to copybook and including file source.
- ▶ Refactoring, such as *rename symbol*.
- ▶ Supporting custom code snippet and more than 200 high value code snippets for COBOL, PL/I, and JCL.
- ▶ Searching and replacing refactoring across multiple program files.

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

Both extensions can also run on Eclipse Theia, which is an IDE that runs in a web browser.

For this demonstration, we chose VS Code and the two previously mentioned IBM extensions to show how these tools can simplify and modernize the experience of developing Db2 applications for the mainframe.

9.3 Installing Visual Studio Code

To install VS Code, visit the [VS Code Download web page](#) and click the suitable operating system to download the installation file.

After you download the file, follow the VS Code installation instructions for the operating system that you are using:

- ▶ [Windows](#)
- ▶ [Linux](#)
- ▶ [macOS](#)

When VS Code is installed successfully, continue to install the extensions (see 9.3.1, “Installing Visual Studio Code extensions” on page 94).

9.3.1 Installing Visual Studio Code extensions

Complete the following steps to install both extensions:

1. Open VS Code.
2. Switch to the Extensions view in the activity bar on the left (see Figure 9-1).

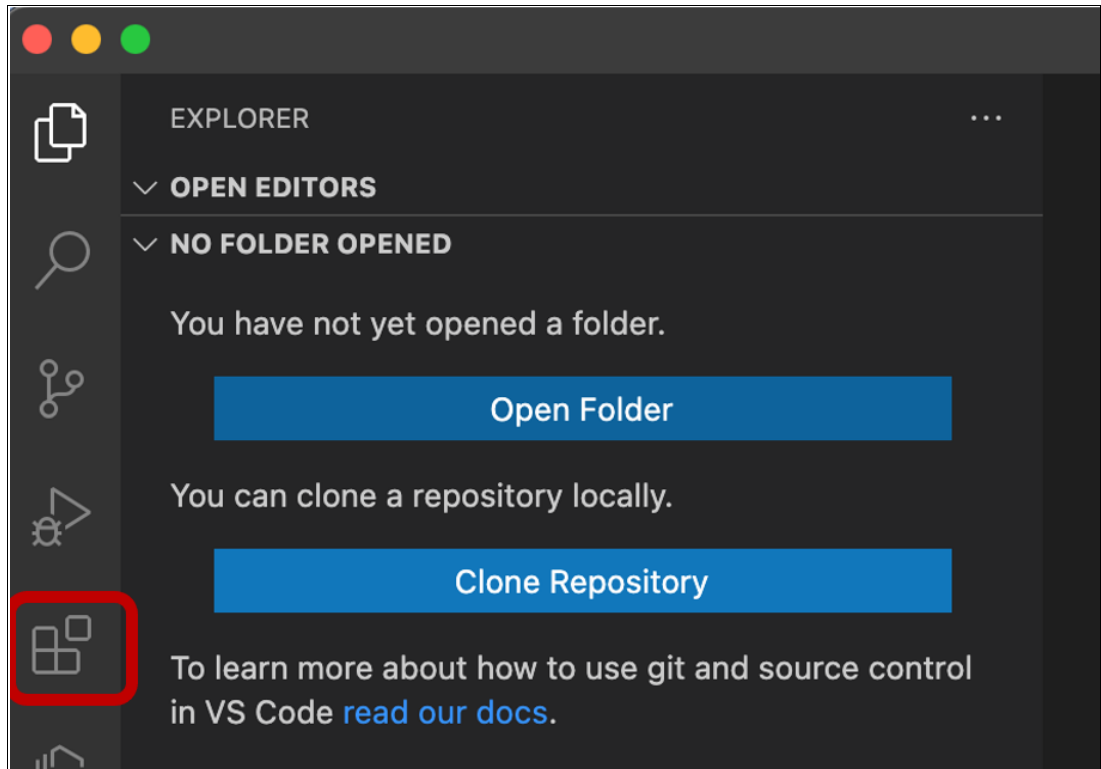


Figure 9-1 Extensions view

3. Enter `ibm z` in the search field. Both extensions are displayed at the top of the search results.
4. Click **Install**.

Figure 9-2 shows the result.

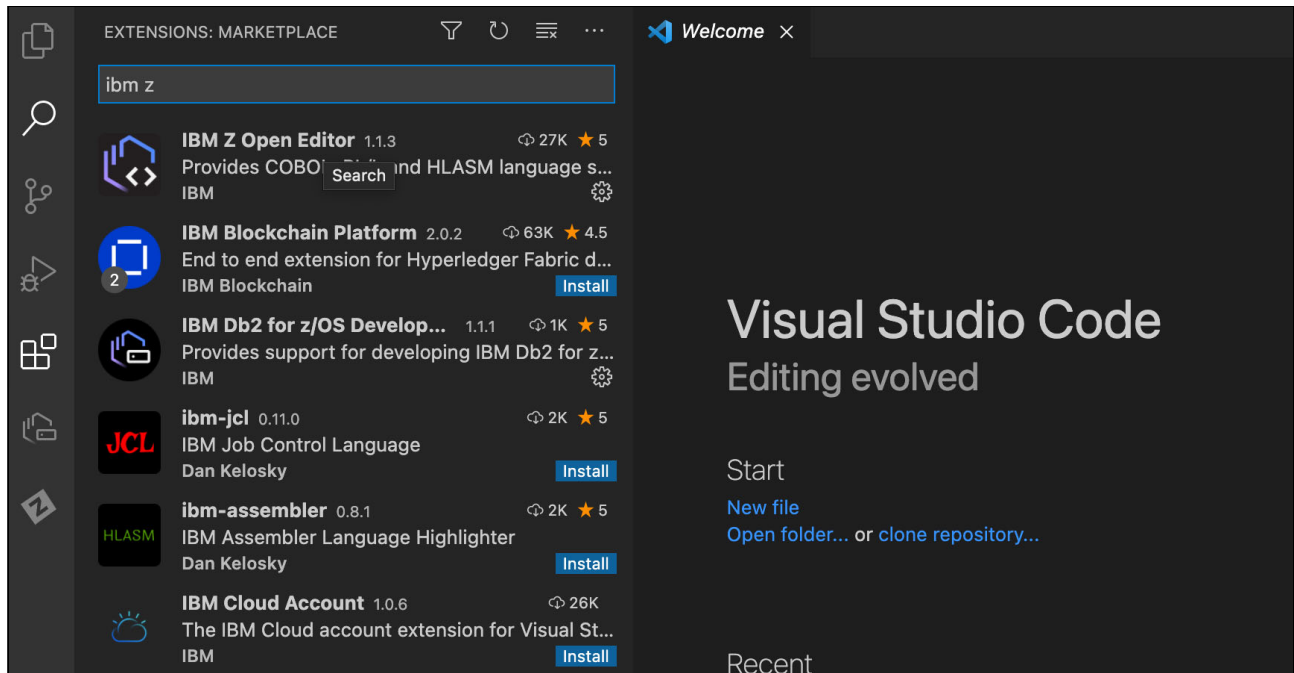


Figure 9-2 Installing Visual Studio Code extensions

The development tools that you use with the sample application in this demonstration are now installed.

9.4 Setting up a code repository in your development environment

The next step is to set up a connection to the source control management system (SCM) and set up the SCM workspace. For our demonstration, we use GitHub. If you use a different SCM, such as GitLab, Bitbucket, Rational Team Concert, or Subversion, the set-up process might differ. Refer to the documentation for the SCM tool that you use.

9.4.1 Installing Git

Before you start using Git, it must be made available on your computer. Also, it is a good idea to always install or upgrade to the latest version.

You can download the latest version at [this web page](#) and follow the [instructions for installing Git](#) for the operating system that you use.

After the installation is complete, several tools are available, including the Git command-line interface (CLI), Git GUI, and Git Bash shell. All of these features are free, and we use many of them along with VS Code in this document.

Alternatively, you can install [one of the Git clients](#) in which Git is included. (Some of these Git clients are free.)

9.4.2 Connecting to the code repository with SSH

By using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to GitHub without entering your username and personal access token at each visit.

For more information about setting up the SSH key, see [this web page](#).

9.4.3 Cloning the code repository

The GitHub code repository can be cloned by using one of the following methods: by using Git commands or by using VS Code.

Complete the following steps to clone the code repository to your workspace by issuing commands from the command line or terminal:

1. Choose or create the directory that you want to persist the local code repository as the parent folder, and change to that directory from the command line or terminal.

2. Enter the following command:

```
# git clone git@github.com:YourOrganization/YourRepository.git
```

where:

- YourOrganization is the organization or user name under which your code repository is created.
- YourRepository is the name of the code repository.

3. Press **Enter** to create your local clone (see Example 9-1).

Example 9-1 git clone command

```
# git clone git@github.com:YourOrganization/YourRepository.git
> Cloning into `Cool-Project`...
> remote: Counting objects: 10, done.
> remote: Compressing objects: 100% (8/8), done.
> remove: Total 10 (delta 1), reused 10 (delta 1)
> Unpacking objects: 100% (10/10), done.
```

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

For more information about cloning a repository to your local workspace directly through VS Code, see [this web page](#).

You are now ready to start making code changes to the application in VS Code.

9.5 Data modeling consideration

Data modeling is the process of defining and analyzing the data requirements that are needed to support business processes within the scope of corresponding information systems in organizations. The process of data modeling typically involves professional data modelers working closely with business stakeholders and potential users of the information system.

In some organizations, data modeling tools are used in database schema design as a key part of application design, where a data modeler team uses the tools to design and create the data models. The data models are eventually converted to DDL and related physical objects.

The data model standardizes and defines the data elements, structure, and the relationship between the elements. Its main purpose is to represent the types of data within a system, the relationships between objects, and their attributes. Figure 9-3 shows what a data model might look like.

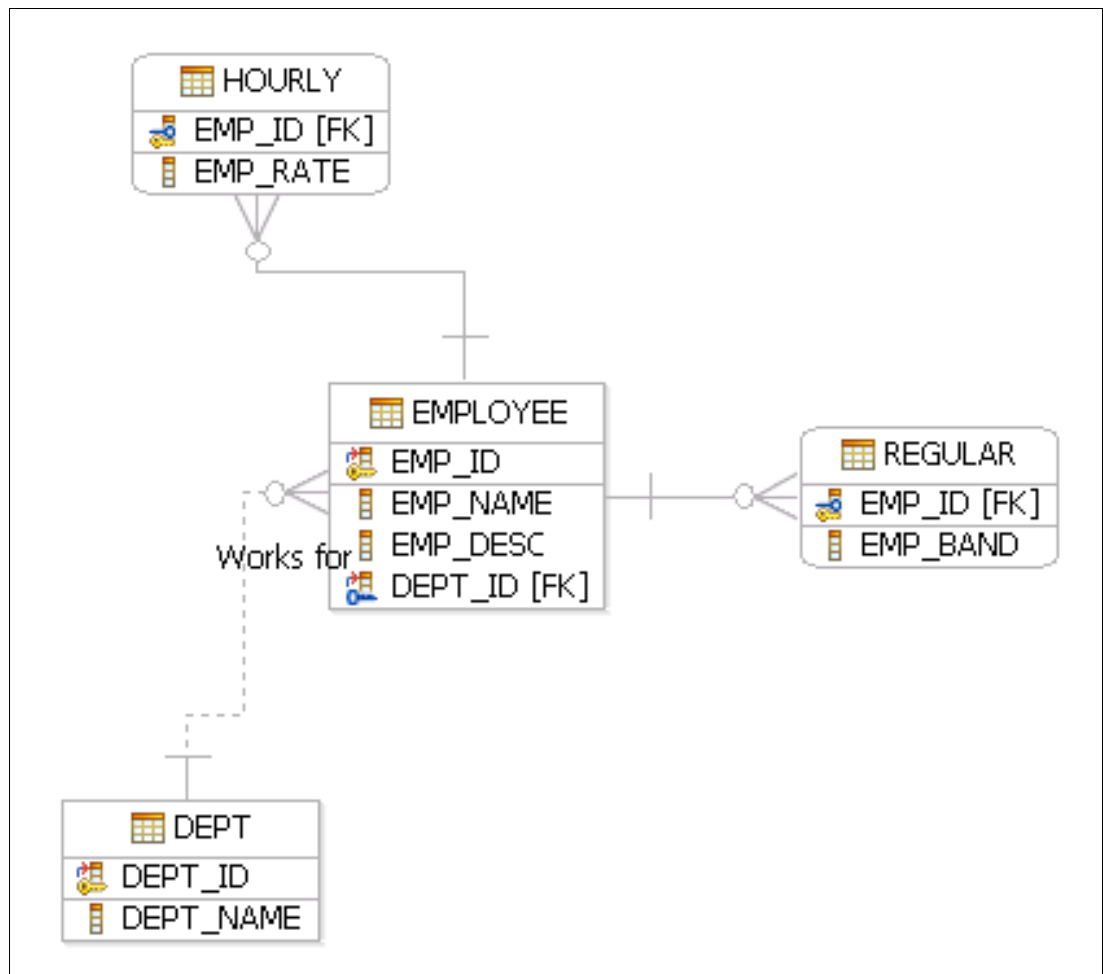


Figure 9-3 The data model

Many data modeling products are available in the industry today, such as IBM InfoSphere® Data Architect. These products help you simplify and accelerate integration design for business intelligence, master data management, and service-oriented architecture initiatives. They also help you create or update logical and physical data model diagrams to describe various applications and systems.

When an organization must make a business logic change that requires database schema changes in the application, the data modeler team often makes the data model change first. Then, the data model change is converted to the DDL changes to be deployed to the systems.

The solution that is described in this paper does not cover how the data modeler takes the requirement change and creates or updates the data model. Most organizations have an established data modeling process already in place. Our solution picks up after the data model change was translated to the DDL changes for the deployment.

Different methods are available to transfer the data model or data model change into the DDL. In most data modeling products, you export the data model into the DDL for deployment when you are ready. The exported DDL can be checked into the SCM as is any other type of source code.

After the DDL is checked in, the schema change in the DDL is handled by the CI/CD pipeline by using the same process that is explained in the solution and sample. In this case, data modelers or DBAs become the developers of the DDL.



Application change execution and demonstration documentation

As described in Chapter 3, “Sample application overview” on page 11, the GenApp application must be adapted to add a country code for the telephone number.

This chapter describes the data structure that is used in GenApp and how that data structure relates to the Db2 for z/OS schemas. We also review what must be changed in the COBOL code and the DDL.

This chapter includes the following topics:

- ▶ 10.1, “GenApp application data structure” on page 100
- ▶ 10.2, “Use case” on page 101

10.1 GenApp application data structure

Many COBOL applications that are deployed in CICS use a COMMUNICATION AREA (COMMAREA) to transfer data into the business logic. When GenApp was originally developed, data was entered by an employee of an insurance company by using a 3270 screen (SSMAPC1 in Figure 10-1). When the employee processed the input, the data from the screen (the presentation layer) was stored in the COMMAREA, and the LGTESTC1 COBOL module was called. Depending on the employee's choice during their screen session, one of the following three tasks were started:

- Update a customer record: LGUCUS01
- Inquire customer details: LGICUS01
- Add a customer: LGACUS01

For the scenario that we use in this document, we focus on the COBOL program LGACUS01 because we are adding only a new customer and a new field for the country code.

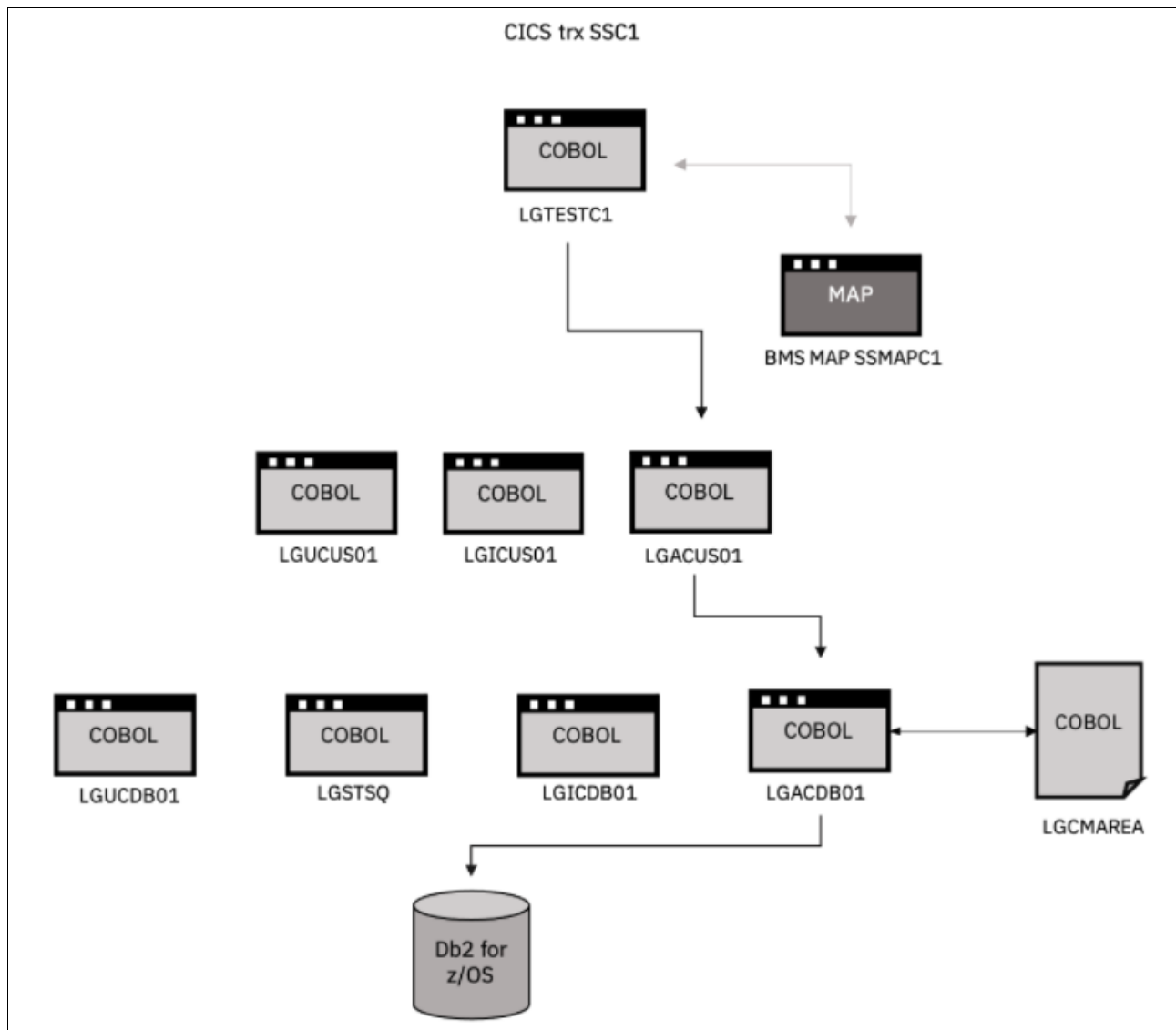


Figure 10-1 Program flow

As shown in Figure 10-1 on page 100, the LGACUS01 program starts the LGACDB01 program, which in turn links to a separate COMMAREA file (LGCMAREA). In this file, we find the original COMMAREA that we use.

To simplify our explanation, we included only the lines that are related to our use case (see Example 10-1).

Example 10-1 Lines that are related to our use case

```
03 CA-CUSTOMER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
    05 CA-FIRST-NAME          PIC X(10) .
    05 CA-LAST-NAME           PIC X(20) .
    05 CA-DOB                 PIC X(10) .
    05 CA-HOUSE-NAME          PIC X(20) .
    05 CA-HOUSE-NUM           PIC X(4) .
    05 CA-POSTCODE            PIC X(8) .
    05 CA-NUM-POLICIES        PIC 9(3) .
    05 CA-PHONE-MOBILE        PIC X(20) .
    05 CA-PHONE-HOME          PIC X(20) .
    05 CA-EMAIL-ADDRESS       PIC X(100) .
    05 CA-POLICY-DATA         PIC X(32267) .
    ...
```

10.2 Use case

To demonstrate the use of Db2 DevOps Experience for z/OS, we change the COBOL code and the Db2 for z/OS Schema (DDL) and show that the two changes are picked up by the CI/CD pipeline. We also introduce a violation of a site rule to demonstrate how DOE ends a deployment and how it can generate an automated message when a site rule is not met.

In this use case, we add the country code field (CA-PHONE-CC) to the COMMAREA (see Example 10-2).

Example 10-2 Adding country code field

```
...
    05 CA-PHONE-MOBILE        PIC X(20) .
    05 CA-PHONE-CC            PIC(5) .
    05 CA-PHONE-HOME          PIC X(20) .
    ...
```

Because a COMMAREA is defined as a specific number of bytes, the COBOL program expects that specific byte length. When we add a new field, we must keep the total length of the COMMAREA constant (one character in a COBOL program fits into one byte). In our use case, we picked the CA-POLICY-DATA field, which was 32267 bytes long, and reduced it by 5 bytes to 32262 (see Example 10-3).

Example 10-3 Reducing field by 5 bytes

```
...
    05 CA-PHONE-MOBILE        PIC X(20) .
    05 CA-PHONE-CC            PIC(5) .
    05 CA-PHONE-HOME          PIC X(20) .
    05 CA-EMAIL-ADDRESS       PIC X(100) .
    05 CA-POLICY-DATA         PIC X(32262) .
```

...

Because we added the new country code field to COMMAREA, we also must change the schema that holds the customer data.

In our setup, the DDL files are stored in the Git repository in the db2/TB folder. For our use case, changed the CUSTOMER.sql file. The original schema looks like Example 10-4.

Example 10-4 Original schema

```
SET CURRENT SQLID='GENCDBO';
CREATE TABLE GENCDBO.CUSTOMER
  (CUSTOMERNUMBER      INTEGER NOT NULL GENERATED BY DEFAULT
                        AS IDENTITY
                        (START WITH 1000001, INCREMENT BY 1, CACHE 20, NO
CYCLE,
                        NO ORDER, MAXVALUE 2147483647, MINVALUE 1000001),
  FIRSTNAME            CHAR(10) FOR SBCS DATA WITH DEFAULT NULL,
  LASTNAME             CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
  DATEOFBIRTH          DATE WITH DEFAULT NULL,
  HOUSENAME            CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
  HOUSENUMBER          CHAR(4) FOR SBCS DATA WITH DEFAULT NULL,
  POSTCODE             CHAR(8) FOR SBCS DATA WITH DEFAULT NULL,
  PHONEHOME            CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
  PHONEMOBILE          CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
  EMAILADDRESS         CHAR(100) FOR SBCS DATA WITH DEFAULT NULL,
  CONSTRAINT CUSTOMERNUMBER
  PRIMARY KEY (CUSTOMERNUMBER))
IN GENCDBO.GENCTS01
AUDIT NONE
DATA CAPTURE NONE
CCSID      EBCDIC
NOT VOLATILE
APPEND NO  ;
COMMIT;
```

Figure 10-2 Use case - 3

We must add a column to the end of the column definitions (in this case, immediately after the EMAILADDRESS column) to hold the data of the country code (see Example 10-5).

Example 10-5 Adding a column

```
...
  PHONEHOME      CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
  PHONEMOBILE    CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
  EMAILADDRESS   CHAR(100) FOR SBCS DATA WITH DEFAULT NULL,
  PHONEMOBILECOUNTRYCODE CHAR(5) FOR SBCS DATA WITH DEFAULT NULL,
  CONSTRAINT CUSTOMERNUMBER
  PRIMARY KEY (CUSTOMERNUMBER))
IN GENCDBO.GENCTS01
...
```

Unfortunately, the new column name that we added (PHONEMOBILECOUNTRYCODE) is too long and violates a site rule.

When we save our changes in the COBOL files, the pipeline mechanism is triggered. The COBOL code is compiled by using the DBB process and succeeds.

However, when the DDL schema is processed, the Db2 DevOps Experience for z/OS plug-in in the UrbanCode Deploy Agent notices that the site-rule violation of the new column name is too long (see Figure 10-3).

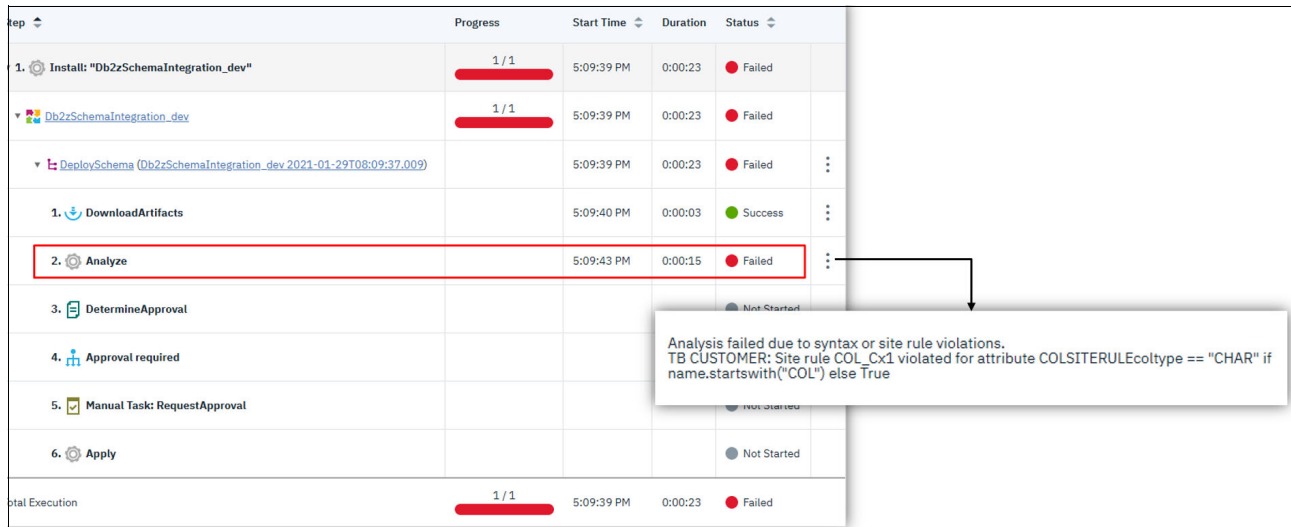


Figure 10-3 Execution Log: Site-rule violation

As shown in Figure 10-3, the Jenkins pipeline failed and you can see that Analysis failed due to syntax or site rule violation message. The operator can zoom in on the failed step, identify the error, edit the CUSTOMER.sql file, and change the column name PHONEMOBILECOUNTRYCODE to PHONEMOBILECC (see Example 10-6).

Example 10-6 Changing the column name

```
...
PHONEHOME      CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
PHONEMOBILE    CHAR(20) FOR SBCS DATA WITH DEFAULT NULL,
EMAILADDRESS   CHAR(100) FOR SBCS DATA WITH DEFAULT NULL,
PHONEMOBILECC  CHAR(5) FOR SBCS DATA WITH DEFAULT NULL,
CONSTRAINT CUSTOMERNUMBER
PRIMARY KEY (CUSTOMERNUMBER))
IN GENCDB0.GENCTS01
...
```

Now that the DDL is corrected and complies with the site rules, we must update the COBOL SQL statement in the 1gacdb01.cb1 file (see Example 10-7).

Example 10-7 Updating the COBOL SQL statement

```
INSERT-CUSTOMER.
*=====
      MOVE ' INSERT CUSTOMER' TO EM-SQLREQ
*=====
      IF LGAC-NCS = 'ON'
      EXEC SQL
          INSERT INTO CUSTOMER
            ( CUSTOMERNUMBER,
```

```

FIRSTNAME,
LASTNAME,
DATEOFBIRTH,
HOUSENAME,
HOUSENUMBER,
POSTCODE,
*      PHONEMOBILE,
        PHONEMOBILECC,
        PHONEHOME,
        EMAILADDRESS )
VALUES ( :DB2-CUSTOMER-INT,
        :CA-FIRST-NAME,
        :CA-LAST-NAME,
        :CA-DOB,
        :CA-HOUSE-NAME,
        :CA-HOUSE-NUM,
        :CA-POSTCODE,
        :CA-PHONE-MOBILE,
*      :CA-PHONE-CC,
        :CA-PHONE-HOME,
        :CA-EMAIL-ADDRESS )
END-EXEC
IF SQLCODE NOT EQUAL 0
  MOVE '90' TO CA-RETURN-CODE
  PERFORM WRITE-ERROR-MESSAGE
  EXEC CICS RETURN END-EXEC
END-IF
ELSE
  EXEC SQL
  INSERT INTO CUSTOMER
    ( CUSTOMERNUMBER,
      FIRSTNAME,
      LASTNAME,
      DATEOFBIRTH,
      HOUSENAME,
      HOUSENUMBER,
      POSTCODE,
      PHONEMOBILE,
*     PHONEMOBILECC,
      PHONEHOME,
      EMAILADDRESS )
VALUES ( DEFAULT,
        :CA-FIRST-NAME,
        :CA-LAST-NAME,
        :CA-DOB,
        :CA-HOUSE-NAME,
        :CA-HOUSE-NUM,
        :CA-POSTCODE,
        :CA-PHONE-MOBILE,
*     :CA-PHONE-CC,
        :CA-PHONE-HOME,
        :CA-EMAIL-ADDRESS )
END-EXEC

```

Now that the COBOL program and the DDL are changed, both files can be saved. With the git-hooks in place, Jenkins starts running the steps again. This time, the changes run error-free. The pipeline also is run, the new compiled code is deployed, and the new DDL is activated.

When a new customer is added, a Country Code can be implemented and reflected in the Db2 for z/OS table.



Summary

In this chapter, we summarize the benefits of the use case that is discussed in this document.

This chapter includes the following topics:

- ▶ 11.1, “Overview” on page 108
- ▶ 11.2, “Real example of pain points managing database changes” on page 109

11.1 Overview

As we showed in this document, organizations today face numerous challenges as they transform their business processes to adapt to the digital economy.

Many organizations must modernize their IBM Z applications to add the flexibility that is required to better serve their customers while simultaneously attaining ever-increasing levels of speed to market, all while maintaining the quality standards that are critical to enterprise applications that are accessing data that is stored in Db2 for z/OS.

It is at this point where DevOps practices comes into play.

By integrating Db2 DevOps Experience for z/OS capabilities into the enterprise CI/CD pipeline, we showed that the deployment and integration of database code changes in tandem with application code changes can be effectively modernized.

We also showed that Db2 DevOps Experience is a role-based solution that provides significant benefits to multiple personas within your organization. As always, the Db2 system programmer is still responsible for the installation of Db2 subsystems and maintenance updates. The automated provisioning of Db2 subsystems for test environments as part of the CI/CD pipeline reduces repetitive work for them.

The DBA is still responsible for maintaining control and security over the Db2 databases; however, with applying DevOps practices, they can simplify aspects of these tasks by creating and linking teams, environments, permissions, and site rules that automatically are enforced during the pipeline execution when Db2 objects are changed.

Db2 DevOps Experience addresses many pain points, including the following examples:

- ▶ Application developers are dependent on DBAs for code deployment. They cannot afford to wait for the DBA to change database structure.
- ▶ Typically, more communication is needed between the DBA, application developer, and data modeler before a change can be implemented. This added communication slows down the change deployment.
- ▶ Application developers open tickets mostly for simple database code changes that add up to many changes over the year. Also, each request must be repeated for multiple environments (unit test, integration test, user acceptance test, and finally, production).

A real case study is shown in Figure 11-1.

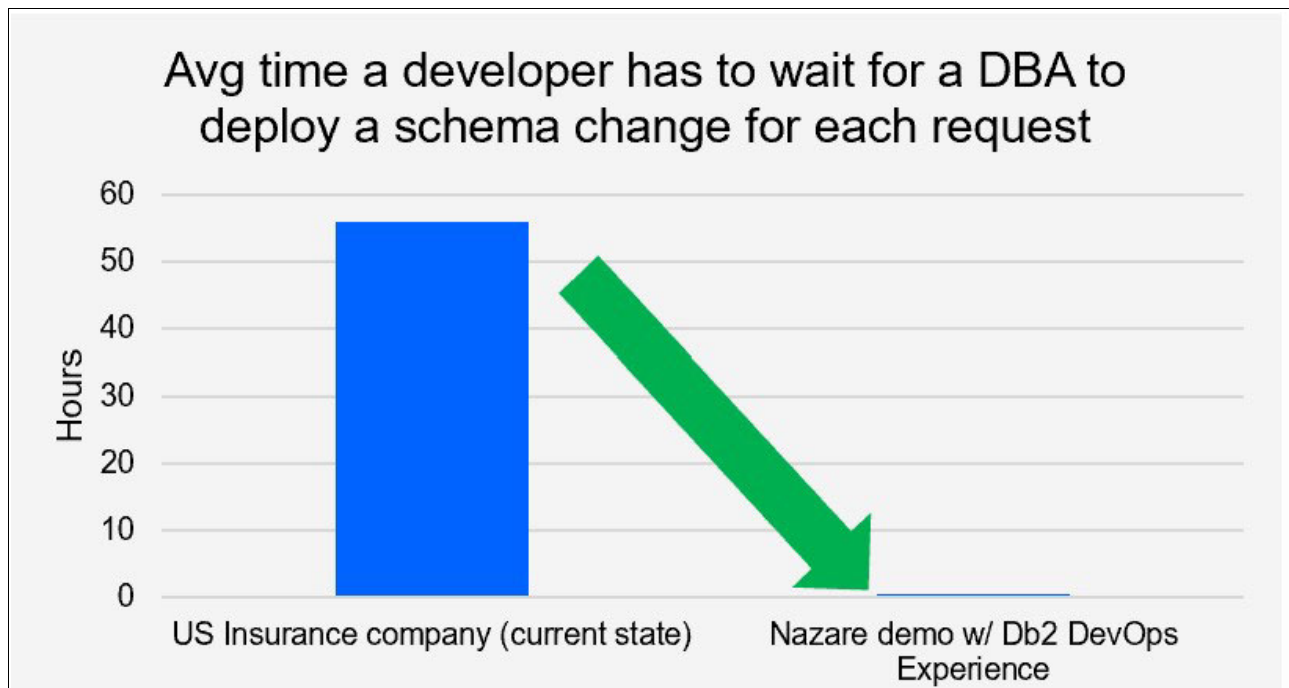


Figure 11-1 Real case study

11.2 Real example of pain points managing database changes

A US insurance company calculated that the average amount of time that application development must wait for a database change request to be completed is 56 hours.

The pipeline execution demonstration illustrates that checking database code changes against DBA-defined site rules and deploying those changes to the target database together with deploying the application change completes in only a few minutes.

Considering the number of database change requests that are fulfilled annually, these capabilities represent a massive time saver. Also, the developer gets immediate feedback about any violations of DBA-defined site rules and can correct them on their own without disruptive communication.

The benefits of integrating database code changes (in this case, Db2 for z/OS) into a CI/CD pipeline can bolster and improve any DBA's efficiency. As a DBA, you can keep pace with the rate of change and not get bogged down in time-consuming activities so that you can provide greater benefits for your IT organization.



A

Additional material

This paper refers to additional material that can be downloaded from the internet as described in the following sections.

Locating the GitHub material

The web material that is associated with this paper is available in softcopy on the internet from the IBM Redbooks [GitHub location](#).

Cloning the GitHub material

Complete the following steps to clone the GitHub repository for this book:

1. Download and install `Git` client (if not installed) from [this web page](#).
2. Run the `git clone` command to clone the GitHub repository.

Abbreviations and acronyms

CI/CD	Continuous Integration / Continuous Delivery
DBaaS	Database-as-a-Service
DBA	Database Administrator
DDL	Data Definition Language
DevOps	Set of practices that combines software development (Dev) and IT operations (Ops)
DOE	IBM Db2 DevOps Experience for z/OS
IDz	IBM Developer for z/OS
JCL	Job Control Language
RDBMS	Relational Database Management System
RTC	Rational Team Concert
SCM	Source Code Management
SDLC	Software Development Life Cycle
SQL	Structured Query Language
UCD	IBM UrbanCode Deploy
UMS	IBM Unified Management Server for z/OS
VS Code	Visual Studio Code

Related publications

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

Online resources

The following websites are also relevant as further information sources:

- ▶ GenApp:
<https://www.ibm.com/support/pages/cb12-general-insurance-application-genapp-ibm-cics-ts>
- ▶ IBM UrbanCode Deploy 7.1.2 IBM Documentation:
<https://www.ibm.com/docs/en/urbancode-deploy/7.1.2>
- ▶ UCD plug-ins:
<https://www.urbancode.com/plugin/ibm-db2-devops-experience-for-zos/>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



REDP-5646-00

ISBN 0738459941

Printed in U.S.A.

Get connected

