Redbooks
ibm.com/redbooks

CICS and DevOps: What You Need To Know

Hernan Cunico
Rod Ainge
Chris Carlin
Ben Cox
Tsahi Duek
Ezriel Gross
Lydia Hao Yang Li
Subhajit Maitra

z Systems

IBM

Redbooks

International Technical Support Organization

CICS and DevOps: What You Need to Know

January 2016

SG24-8339-00

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

First Edition (January 2016)

This edition applies to Version 5, Release 3, Modification 0 of CICS Transaction Server.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

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, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: 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 states 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 supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

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 the names and addresses used by an actual business enterprise 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.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®
CICS Explorer®
CICSPlex®
Concert™
DB2®
IBM®
IBM UrbanCode™
IBM z™
IBM z Systems™
IMS™
Jazz™
Rational®
Rational Team Concert™
Redbooks®
Redbooks (logo) ®
WebSphere®
z Systems™
z/OS®

The following terms are trademarks of other companies:

Inc., and Inc. device are trademarks or registered trademarks of Kenexa, an IBM Company.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

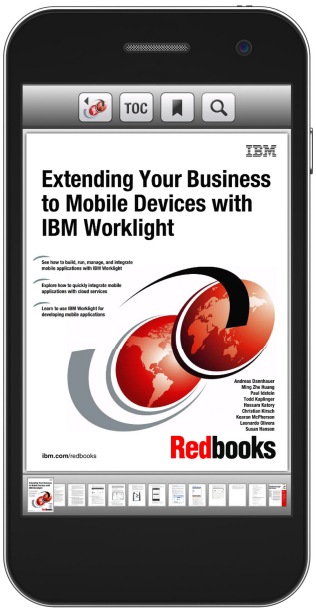
Other company, product, or service names may be trademarks or service marks of others.

Find and read thousands of IBM Redbooks publications

- Search, bookmark, save and organize favorites
- Get up-to-the-minute Redbooks news and announcements
- Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App

iOS **Download Now** Android



Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!

ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication provides an example approach of an agile IT team that implements development and operations (DevOps) capabilities into an IBM CICS® application. Several tools are used to show how teams can achieve transparency, traceability, and automation in their application lifecycle with the assistance of all the stakeholders to deliver high-quality application changes that meet the requirements.

The application changes that are built highlight the composable and dynamic nature of using CICS, the Liberty JVM runtime server, and IBM UrbanCode™ Deploy, which allows developers to get their applications running quickly by using only the programming model features that are required for their applications.

The target audience for this publication is IT developers, managers, and architects, and project managers, test managers and developers, and operations managers and developers.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Hernan Cunico is a Senior Information and IT Architect at the ITSO, Raleigh Center. He has over 18 years of experience in the consulting and development fields. His areas of expertise include user experience design, information and application development, middleware, portals, open source, cloud, mobile, and agile development. He has written extensively on commerce, portals, migrations, human resources, and learning solutions.

Rod Ainge is a Software Engineer at the Bank of Western Australia, in Australia. He has over 30 years of experience in development and software management. He holds a Bachelor of Science degree in Computer Science from the Royal Melbourne Institute of Technology (RMIT). His areas of expertise include application development, Software Release Management, and Development Infrastructure. He is a Mainframe Evangelist and keen advocate of new technologies, such as Java, on IBM z Systems™.

Chris Carlin is a Staff Software Engineer in the United States. He has 11 years of experience as a CICS Level II Technical Support representative at Research Triangle Park, North Carolina. He has worked at IBM for 20 years, spending nine years as an Application Developer at various IBM clients. He has a Bachelor of Arts degree in Computer Science and a minor in Management from New Jersey Institute of Technology (NJIT).

Ben Cox is a Software Engineer and Designer at IBM Hursley, UK. He has six years of experience working in the development team for CICS Transaction Server and IBM CICS Explorer®, including bringing OSGi support to CICS, creating the CICS cloud, and working on the build and deployment tools that are described in this book.

Tsahi Duek is a CICS system programmer in Israel. He has 10 years of experience in the IT field. He holds a B.S.C degree in Mathematics and Computer Science from the Open University in Israel, and a M.S.C degree in Technology and IT Management from Holon Institute of Technology in Israel. His areas of expertise include application management and development, team management, business impact analysis, and workflow process design.

Ezriel Gross is the CEO of Circle Software Inc, formerly Circle Computer Group LLC, an IBM Business Partner that specializes in hands-on classes in CICS, IBM DB2®, and IBM WebSphere® MQ. Ezriel has been an IBM Gold Consultant for many years and attends the annual Gold briefing to keep current. Besides consulting, he teaches and develops CICS courses for both IBM and Circle. His specialties include CICS Web Services, CICS Web Support, CICS Performance / Tuning, and CICS Internals.

Lydia Hao Yang Li is a Client Technical Specialist with IBM ASEAN in Singapore. She has eight years of experience in the z Systems family. For the past five years, she has been in presales supporting customers across Southeast Asia with z Systems solutions. Her areas of expertise include CICS, problem determination Tools, connectivity, and mobile solutions. She holds a Bachelor of Engineering degree in Electrical and Electronic Engineering from Nanyang Technological University in Singapore.

Subhajit Maitra is a Senior IT Specialist and member of the IBM North America Systems Middleware Technical Sales team. In addition to CICS, his expertise includes IBM Operational Decision Manager, IBM Integration Bus, and WebSphere MQ on z Systems products. Subhajit, who has worked in IT for 22 years, is also an IBM Global Technical Ambassador for Central and Eastern Europe, where he helps IBM clients implement business-critical solutions on z Systems servers. He holds a master's degree in computer science from Jadavpur University, in Kolkata, India, and is an IBM zChampion.

Also contributing to the development of this book was **Shawn Tooley**, a technical writer with the ITSO at the Raleigh Center.

In addition, the authors want to thank the following individuals for their contributions to this project:

Matthew Comer, Mark Cooker, David Knibb, Phil Wakelin, Matthew Wilson

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 ITSO 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 books to be as helpful as possible. Send us your comments about this book 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, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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

1

Introduction

This chapter introduces the new development and operations (DevOps) features that are available in IBM Customer Information Control System (CICS) Transaction Server V5.3. These new capabilities, including the IBM CICS Transaction Server build toolkit (CICS build toolkit) and a new CICS plug-in for IBM UrbanCode Deploy, enable IT departments to release new applications faster than before by breaking down the traditional walls between the development and operations teams.

Of course, with CICS handling transactions for some of the world's biggest banks and other enterprises, some IT managers are reluctant to adopt the new DevOps features for fear of impacting vital applications. However, the new features can be introduced slowly. You can even adopt a bi-modal approach so that you can use DevOps capabilities to release new Java business or interface logic in rapid cycles while retaining your existing development methods for less frequent updates to mission-critical, back-end applications.

This chapter covers the following topics:

1.1 The need for a new approach

CICS Transaction Server is the transactional workhorse that keeps many of the world's biggest banks and other large enterprises running smoothly. Its flexibility, adaptability, and underlying components, such as IBM z Systems products, have kept it the dominant force in the industry for years.

But the IT world is changing.

Traditionally, there was a clear line between development and operations. The development team wrote code and created applications. The operations team then built, packaged, and deployed those applications for use. Unfortunately, this approach is linear. Each stage of the process depended on the previous stage. This led to several problems:

- Time-consuming manual software deployments
- Lack of traceability from requirements to release
- Long delays in incorporating user and customer feedback

Worse, with the teams operating in their own silos, communication was often ineffective and business goals were sometimes not achieved. New software releases were frequently delayed and the overall process was subject to sudden disruption.

Such was the state of application development for years. It was acceptable (at least somewhat) because customer expectations were still low. If your software did not work perfectly today, a fix would arrive in the next release. So, you waited six months to a year and everything would be corrected.

But, that situation was before the advent of today's *app economy*. With the emergence of smartphones and other devices, there is an explosion in apps that can do such varied tasks as ordering pizza to managing a warehouse. Through the *Internet of Things*, companies can better monitor remote equipment and gain new insight into customer behavior (even at the most minute level), so the pressure to make full and immediate use of this data grows by the day.

So, waiting six months for a software update is unrealistic. Your customers will ask you why their smartphone manufacturer can push out new operating system updates every week, but they must wait months for the improvements they requested.

1.2 The DevOps solution

To speed the release of new applications and updates, the IT industry works to apply agile and lean principles to development and deployment. At the core of these principles is eliminating wasted effort, breaking down artificial barriers between related functional teams, and adopting continuous release cycles that push improvements out to users faster than ever before.

One such artificial barrier is the line (thicker in some companies than in others) between development and operations teams. However, this line is being eliminated and a new, more aggressive, and *business-driven* approach that is called DevOps has emerged. Applications are built, deployed, rebuilt, and redeployed in rapid cycles aimed at delivering continuous, incremental improvements.

So can CICS continue to thrive in a DevOps world? The clear answer is yes.

In this publication, you see that tools such as the CICS build toolkit and the DFHDPLOY utility are removing the walls between CICS developers and operations staff. Apps and updates can move more rapidly from raw code to deployable artifacts. You can even enable automatic deployment of approved apps, configurations, and other artifacts by integrating CICS with IBM UrbanCode Deploy.

1.2.1 Benefits of DevOps

In a DevOps shop, the business owner, the developers, and the operations team work collaboratively. The plan-develop-build-deploy-feedback process is continuous (see Figure 1-1). When possible, human actions are minimized in favor of autonomous tools, including software builds that start on their own when eligible code updates become available. Applications reach their full potential faster, often by using fewer resources than in the past.

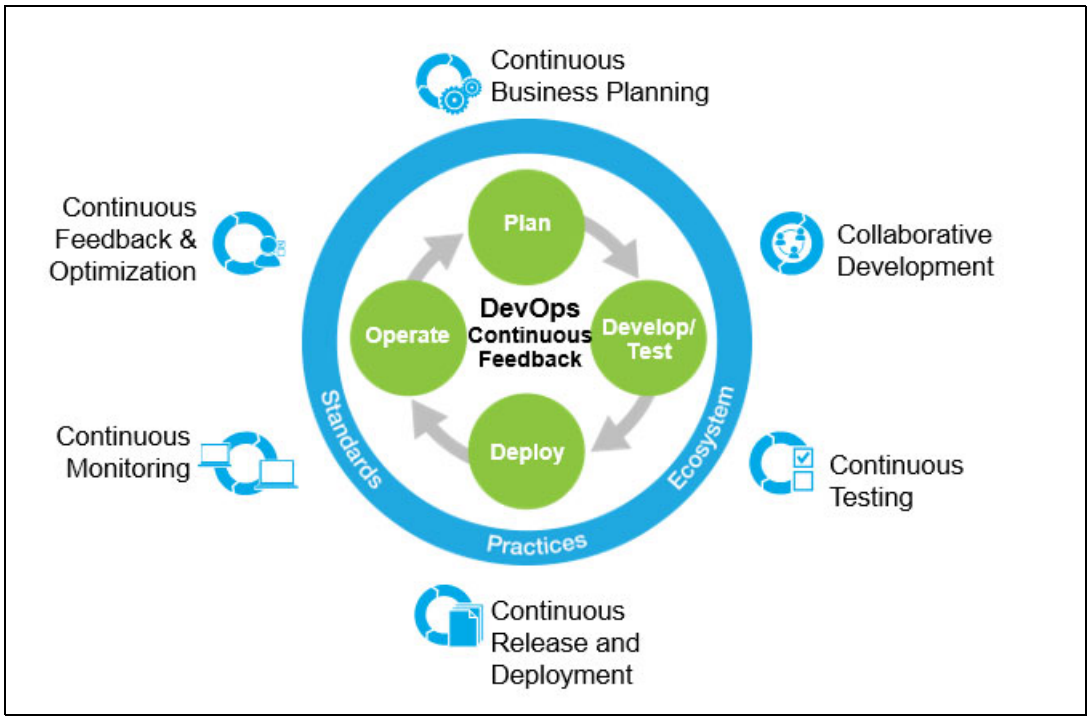


Figure 1-1 DevOps overview

There is a strong focus on continuous integration, continuous collaboration, and continuous improvement. These are the core principles of the DevOps approach. Together, they provide numerous advantages:

- Enhanced customer experience

  New features and functions are pushed out to users faster than before. As each new feature is released, you get user feedback that can be used to improve the next release.

- Increased ability to innovate

  Teams can work on small, incremental updates rather than holding everything for a master release, which means more time for innovation, which in turn becomes an ongoing, repeatable process.

- Faster time to value

  By releasing more frequent but smaller updates and generating new customer feedback each time, applications are improved faster and the end state of the application is achieved more rapidly.

The DevOps approach is multi-platform. The principles are the same whether the application runs on a mainframe, in a distributed environment, or in the cloud. It can be applied to any development effort, from enterprise applications such as back-end banking systems to more customer-facing products such as smartphone tools. It is customer-neutral. Whether you are working with an external client or an internal line of business, DevOps can help you meet your commitments faster and with better results.

1.3 Bi-modal IT: The best of both worlds

So, the benefits of DevOps are clear. But, adopting a DevOps approach can be daunting. Breaking down the walls is the easy part. It is more difficult to determine where within your IT operation to apply DevOps principles. Where is the rapid release, rebuild, and rerelease aspect of DevOps most beneficial? Where might such frequent changes create problems? How can you be both *fast* and *stable* at the same time?

One answer is to adopt a *bi-modal* approach to IT. Use the rapid development-release methodology for user interface updates, for example, which today's users expect, but retain your typical, longer development calendars for core business applications. Or, use DevOps teams to build quickly new applications that are aimed at winning new business, but keep traditional methods in place to assure the quality of existing business applications.

1.3.1 CICS and bi-modal IT

CICS Transaction Server V5.3 is well-suited to bi-modal IT because it contains the best of both worlds: traditional application development and application development through the DevOps approach.

Table 1-1 describes how CICS work can be split up in a bi-modal fashion.

Table 1-1 Bi-modal division of CICS work

Component	DevOps mode	Traditional mode
Suitable applications	New CICS applications, often Java business or interface logic	Existing, well-tested CICS applications (often COBOL, Assembly and PL/I-based) running core business applications
Methodology	Agile and iterative processes	Strict development calendars, with long and thorough regression testing
Primary objective	Rapid delivery of new features	Accurate, assured service delivery
Business purpose	Win new business	Support and maintain existing business

1.3.2 Old and new can coexist

Another way to look at bi-modal DevOps is to consider where your updates are targeted. Where rapid changes are beneficial because of user expectations or a changing business environment, use the DevOps approach that is possible in CICS Transaction Server V5.3. Where a stable base is your priority, use more traditional methods with typical regression testing.

Table 1-2 provides examples of where the different modes are preferably applied.

Table 1-2 Ideal applications of different development modes

DevOps mode (CICS Transaction Server V5.3)	Traditional mode (CICS Transaction Server V5.1 and V5.2)
► Java owning regions ► Liberty Profile owning regions ► Web owning regions ► Mobile owning regions ► Rapid development and testing regions ► Developer/Sandbox regions	► Traditional application owning regions ► Traditional data-owning / file-owning regions ► Traditional terminal-owning regions ► Traditional development/test regions

1.3.3 Continued evolution towards the DevOps mode

Many IT experts predict that as new technologies emerge that require more interfaces to more channels, the DevOps approach will gain popularity for the speed with which it can generate releases and updates. Over time, as DevOps methods mature and more companies adopt them, the quality of the involved processes will improve and the rapid updates that are released will grow more stable and reliable.

Even if companies continue to maintain their critical applications by using old methods, it is anticipated that as their DevOps teams gain more experience and produce good results, these companies will be more willing to use DevOps even for their most vital applications. In this sense, as the industry gets better at DevOps, it is expected that the role of DevOps within the typical IT shop will expand even more.

1.4 DevOps roles: Old versus new

In the traditional software delivery pipeline, developers write code in an integrated development environment (IDE), such as Eclipse. When they are ready, they deliver the new code into the company's source code management (SCM) system. Then, the operations team builds the applications so that they are machine-ready; the resulting packages are put into a package repository. The application is deployed to the test environment, then a pre-production or staging environment, and finally the production environment.

The DevOps delivery pipeline has all the same phases. But, it adds automation and continuous feedback loops, so more things happen automatically, and the deliverable artifact can improve with each develop-build-deploy cycle.

Some traditional roles (and where they fit in the progression) are revised in the DevOps pipeline:

- Developers: Developers are responsible for coding program changes and performing tests and quality checks on applications. Developers work across multiple platforms and work with other developers to ensure that code changes are integrated across all involved platforms and environments.
- Build engineers: These individuals build and manage tools to automate, manage, and monitor the build processes. The tools monitor the SCM system to learn when new code arrives there and then start autonomous processes that build and package the artifacts (bundles, load modules, and so on) and place them in common repository for deployment. The new CICS build toolkit is one such tool.
- Release engineers: These individuals create automated processes to take the builds from the common repository and deploy them to the target environment, such as a CICS region. To help these processes, CICS Transaction Server V5.3 includes utilities such as DFHDPLOY and a CICS plug-in for IBM UrbanCode Deploy.

An integrated deployment solution is vital in a DevOps environment to bring together the deployment of all artifacts. The artifacts can be developed by different teams, written in different languages, originate from different platforms, and have different deployment solutions within a platform and across multiple platforms. The aim is to manage the artifacts within an integrated process. So, deployment of a Java bundle in a CICS Liberty environment, deployment of a COBOL load module into a CICS application environment, and deployment of a CICS cloud application can occur together.

1.5 DevOps tools in CICS Transaction Server V5.3

CICS Transaction Server V5.3 includes a number of solutions to manage artifacts and assist organizations in adopting the DevOps approach.

1.5.1 CICS build toolkit for automated builds

The CICS build toolkit provides a command-line interface (CLI) to build your CICS projects that are created in CICS Explorer. This method is superior to the previous manual method of using a wizard to export the CICS projects to zFS because you can use a script to build projects in a reliable, repeatable manner. Using the build toolkit is one step toward a fully automated, continuous integration process (see Figure 1-2).

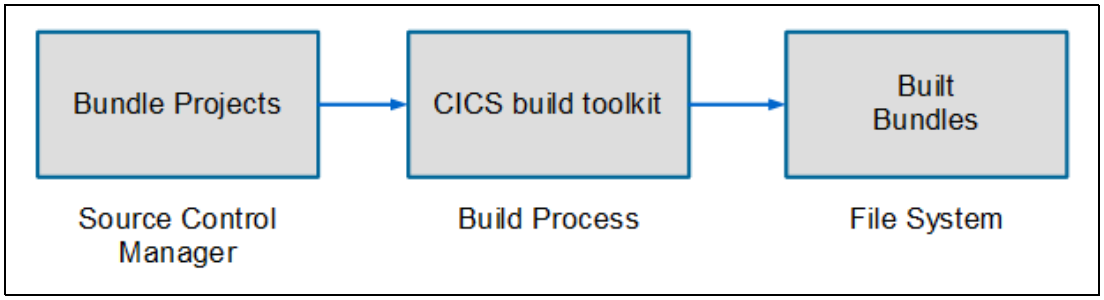


Figure 1-2 CICS Built Toolkit pipeline

The CICS build toolkit can build these projects:

- CICS bundles
- CICS applications
- CICS application bindings
- CICS reference projects, such as OSGi bundles, OSGi applications (EBAs), enterprise applications (EARs), and dynamic web projects (WARs)

A significant feature of the CICS build toolkit is its ability to resolve variables within attributes by using values from a properties file, which means that CICS bundles can be redeployed to different target environments by using different variable settings, without any changes to the underlying source code. For example, a debug variable can be set in the development environment but not in the test or production environment.

The automation features of the CICS build toolkit apply within two specific processes. First, the build automation process uses the toolkit to build the project. Then, the deployment automation process uses the toolkit to resolve the variables into specific values for the target environment.

The CICS build toolkit is supported on IBM z/OS®, Linux, and Microsoft Windows, which makes it easy to integrate it with various SCM system tools. It can be used with CICS Transaction Server V4.1 and later versions, enabling organizations that are not yet using CICS Transaction Server V5.3 to still make the move toward DevOps.

1.5.2 DFHDPLOY for batch deployments

Also introduced in CICS Transaction Server V5.3 is a new utility called DFHDPLOY. DFHDPLOY provides a set of commands that can be used in a script to deploy, undeploy, and set the state of CICS bundles and cloud-enabled CICS applications. The utility also can connect to a CICSplex.

When you use the utility's **`Deploy`** command, the CICS bundle or CICS application can be defined and its state changed to Disabled, Enabled, or Available within a CICS system or group of CICS Systems.

For a cold start, the **`Deploy`** command can be used to add a bundle definition to a CSD group that is part of a startup list, or add it to a RESGROUP that is added to a RESDESC.

When you use the **`Undeploy`** command, the CICS bundle or CICS applications state can be changed to Unavailable, Disabled, or Discarded. The Unavailable state waits for the application workload to complete before performing any action.

When you use the **`SET`** command, a higher version of an OSGi bundle can be phased in without disrupting active tasks.

The DFHDPLOY utility is called by using JCL and can be easily integrated into existing deployment tools and automation procedures. DFHDPLOY can be used with the CICS build toolkit to automate the building and deployment of CICS bundles and cloud-enabled CICS applications. It can also be used to enhance a continuous delivery environment and help the move to DevOps.

1.5.3 Deploying through IBM UrbanCode Deploy

The IBM UrbanCode Deploy tool automates the deployment of artifacts ranging from applications to middleware configurations to database changes. CICS Transaction Server V5.3 includes a plug-in that enables IBM UrbanCode Deploy to work with CICS resources.

For more information about IBM UrbanCode Deploy, including typical use cases and features or to download an evaluation version, go to the following website:

https://developer.ibm.com/urbancode/products/urbancode-deploy

Organizations can use the new IBM UrbanCode Deploy plug-in in CICS Transaction Server V5.3 to improve workflow efficiency and wrap CICS deployments into their overall continuous delivery processes.

The plug-in can perform these actions:

- Define and install bundle resources in CICS System Definitions (CSD).
- Define and install bundle resources in business applications services (BAS).
- Define groups and add them to startup lists.
- Enable, disable, and discard resources.
- Open and close resources.
- Perform NEWCOPY and PHASEIN actions on resources.
- Scan pipelines.
- Check the status of resources (Enable or Open).
- Deploy and undeploy bundles and applications (by using DFHDPLOY).
- Set resources as Available and Unavailable.

The IBM UrbanCode Deploy plug-in and the new DevOps tools that are built into CICS (the CICS build toolkit and the DFHDPLOY utility) are described in greater detail in Chapter 4, “Release environment” on page 43.

1.6 Putting it all together

When the new tools and utilities that are described in this chapter are considered together, it creates a new DevOps-style delivery pipeline.

Figure 1-3 on page 9 illustrates how IBM Rational® Team Concert™ can provide a framework for development and build activities. Developers can use the Rational Team Concert client and use IDE extensions with plug-ins such as CICS Explorer. Concurrently, they can use source control management practices for accurate versioning of the source code repository.

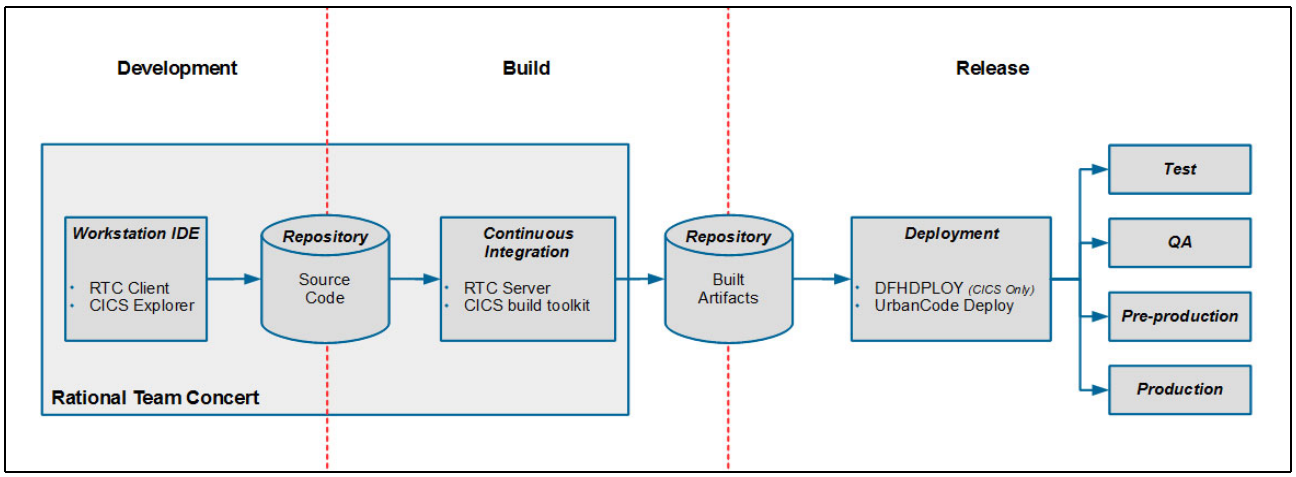


Figure 1-3 Tools for DevOps in CICS Transaction Server V5.3

As you move into the build, continuous integration, and testing functions, Rational Team Concert can be extended with the CICS build toolkit for process integration and build automation. Built artifacts can then be stored in a separate repository where deployment tools such as DFHDPLOY (for CICS projects) and IBM UrbanCode Deploy inject environment-specific variables and definitions before finally deploying the artifacts to the target environments.

2

Development environment

This chapter describes the development processes and tools that are commonly used when applying the development and operations (DevOps) methodology. It includes a description of the new features in CICS Transaction Server V5.3 that support DevOps-style continuous delivery.

Here are some of the tools that are covered in the chapter:

- Integrated development environments (IDEs), such as Rational Developer for z Systems, CICS Explorer, and the CICS Configuration Manager plug-in
- Source control management (SCM) systems
- Testing systems, including personal build processes and unit testing from the IDE

The goal is to create an integrated DevOps pipeline in which developers can develop in parallel, get rapid feedback on their work, deliver updates continuously, and maintain stable deployment environments. You can use such a pipeline to achieve quickly benefits from the DevOps methodology.

This chapter covers the following topics:

2.1 Integrated development environments

An IDE is software program that is used for developing applications. IDEs support different types of coding languages and build engines and usually allow the addition of plug-ins to support additional features:

- Source code management (SCM)
- Build and deploy
- Database connectivity

CICS application development is supported on various IDEs, for example:

- Rational Developer for z System
- CICS Explorer (the CICS systems management tool)
- CICS Configuration Manager (CM) (the CICS resource definition change tool)

This book uses CICS Explorer to demonstrate how the development process can be linked to the new DevOps deployment features in CICS Transaction Server V5.3. Learning about CICS Explorer can be beneficial even if you still develop your z/OS applications under TSO/E.

2.1.1 Rational Developer for z Systems

Rational Developer for z Systems is an Eclipse-based IDE that provides different toolsets for developing IBM z/OS applications. Rational Developer for z Systems is unique in that it offers development tools for PL/I, COBOL, Java, C++, and assembly language, along with real-time syntax validations.

Rational Developer for z Systems also includes connectivity to z/OS systems (through IBM z/OS Explorer), CICS Transaction Server (through CICS Explorer), and to DB2 and IBM IMS™ products. Because Rational Developer for z Systems is an Eclipse-based IDE, you can add features and plug-ins to Rational Developer for z Systems either manually or with the provided installation manager.

Rational Developer for z Systems is typically used to develop z/OS applications with the supported tools for PL/I and COBOL development. The user can compile, debug, and run programs whether online or batch.

Rational Developer for z Systems is installed by using IBM Installation Manager. You can use IBM Installation Manager to install and manage different IBM software from a single program. Software can be installed as a stand-alone instance, which is known as a product, or can be installed into existing products, where compatible. This installation into the same product permits greater integration between products, such as adding team collaboration features into text editors.

2.1.2 CICS Explorer

CICS Explorer is an Eclipse-based tool for managing CICS systems. In CICS Explorer, the user can manage all CICS resources, from regions to resource definitions, in one centralized place, with various wizards and resource editors that can help. You can also manage CICS bundles, cloud applications, and platforms, and integrate your CICS environment with an SCM system by using an Eclipse plug-in.

You can also develop Java EE, web, and JCICS applications by using the CICS Explorer SDK plug-in. The CICS Explorer SDK plug-in also supports WebSphere Application Server Liberty profile, which you can use as your target platform for application development.

There are many options for installing IBM Explorer for z/OS and CICS Explorer. The preferred method is to use IBM Installation Manager, which you can use to create a product or extend an existing product that was installed by IBM Installation Manager. Alternatively, you can extend an existing Eclipse installation by using Eclipse's built-in p2 installation process, or you can simply download CICS Explorer as a compressed file and extract it.

Note: Mixing installation methods can result in installation problems. If you first use IBM Installation Manager to install a product, use only IBM Installation Manager to extend it. Likewise, if you first download a product as a compressed file and extract it, use only Eclipse's built-in p2 installation process to extend it.

For more information about each of these installation options, see the z/OS Explorer and CICS Explorer downloads documentation, found at:

http://www.ibm.com/support/docview.wss?uid=swg24033579

Another major ability of CICS Explorer is to use variables for attributes that are defined in many kinds of definitions (CICS Transaction Server resource definition, XML files, and other files). You can create a general resource definition that is applied to all of your different environments. The value of the attribute is replaced based on the properties files within the target directory. Using the IBM CICS Transaction Server build toolkit (CICS build toolkit), you can replace all the attribute variables with appropriate values, depending on the environment to which this resource is deployed. Here are the different ways to create and use variables substitution:

- Creating variable for CICS Transaction Server resource definition with the embedded wizard

  Complete the following steps:

  a. Within the tabular view of any resource definition, you can right-click any of the fields and select **Export value to variable** (see Figure 2-1).

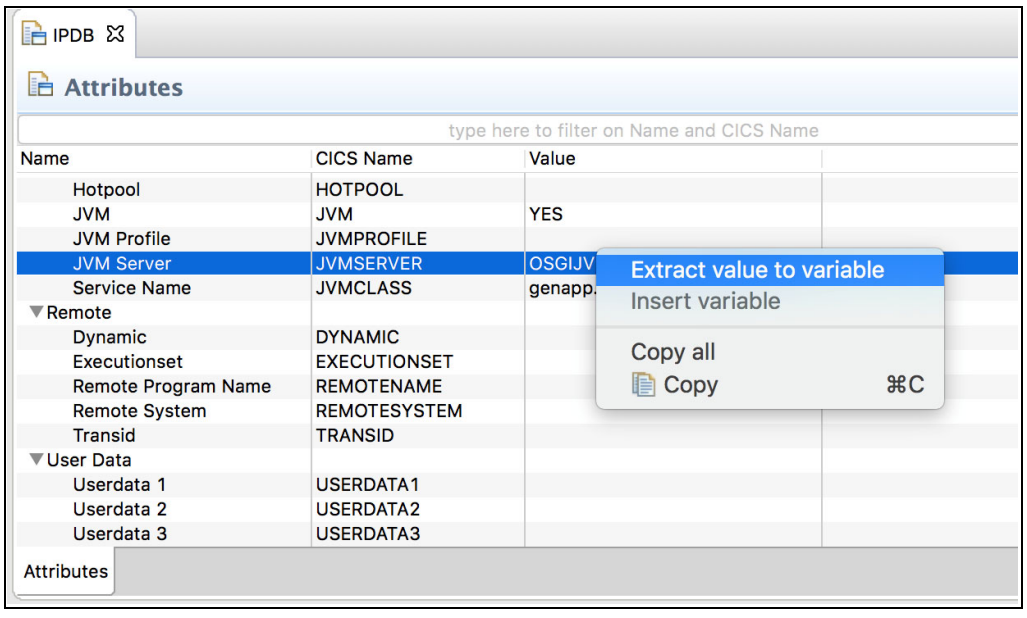


Figure 2-1 Export value to variable example

b. In the next window, enter the wanted variable name and value and click **Finish**. If there are related projects (for example, application binding project), you can specify different values for different projects. In this example, we change the name of the JVM server to OSGIJVMS for the current bundle, PLATJVMS for the project with the multi-region platform, and PLATJVMS for the project with the single region PLATFORM (see Figure 2-2).

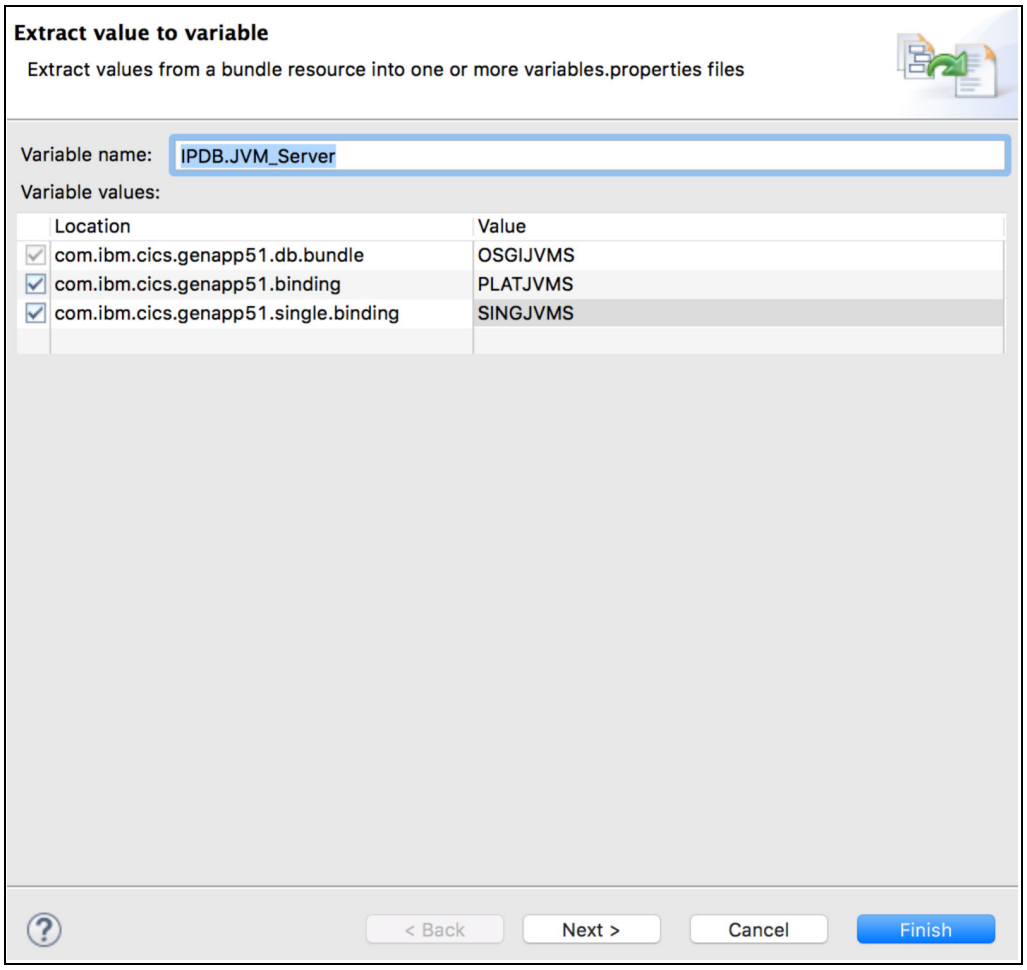


Figure 2-2 Extract value to variable

c. The `variables.properties` files should be created in each project that was selected in the previous step. The file should look like what is shown in Figure 2-3.

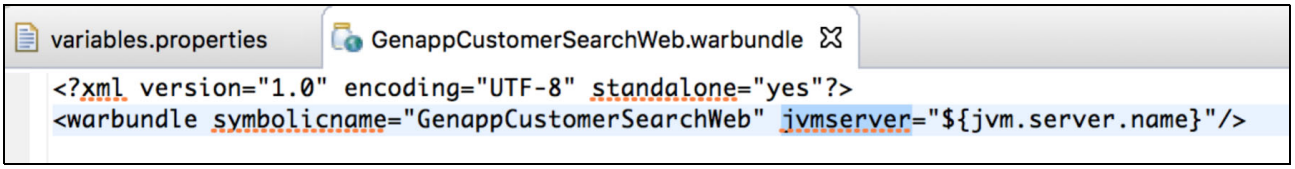


Figure 2-3 variables.properties file

d. Finally, instead of a hardcoded value in the field that you choose, the new variable appears with a dollar sign and brackets surrounding it (`${IPDB.JVM_Server}`) (see Figure 2-4 on page 15).

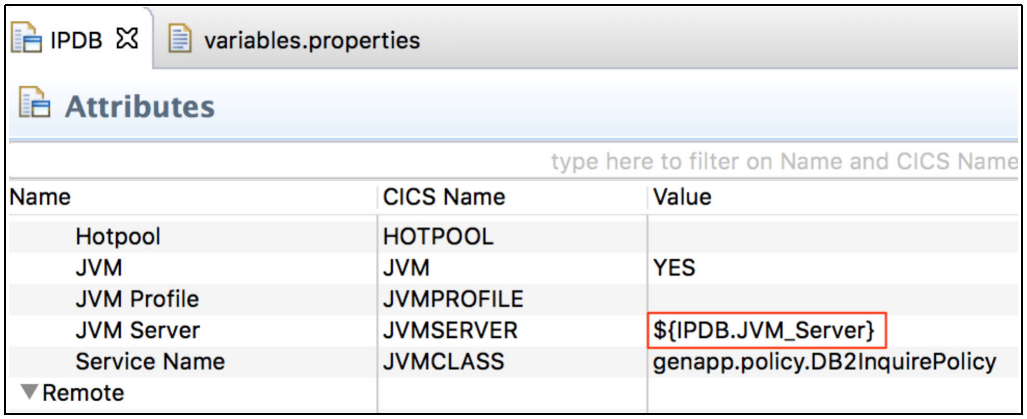


Figure 2-4 Variable instead of a hardcoded value in the definition

- Creating the `variables.properties` file manually

  If your file does not have a tabular view like a CICS Transaction Server resource definition (for example, XML files), or you do not want to use the wizard, you can manually create a text file that is called `variables.properties` in the containing project, and use the variable in any file you like.

  For example, say that you have a CICS bundle project with a `warbundle` file inside it. The `warbundle` file is an XML file in which one of its attributes is `jvmserver`. Complete the following steps:

  a. Create a file that is called `variables.properties` within the projects and enter this line:

     `"jvm.server.name=GENALIBS"`

  b. Open the `warbundle` file and change the value of the `jvmserver` attribute to `${jvm.server.name}`. The file should look like what is shown in Figure 2-5.

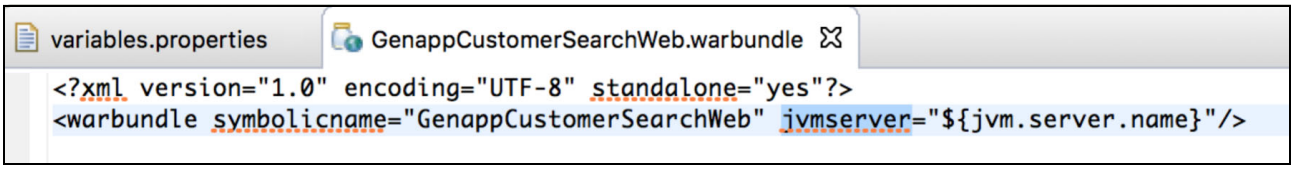


Figure 2-5 Replace a value with a variable manually

You can also combine variables with hardcoded values, or concatenate as many variables as you want. For example, if you create one variable with the name `application.initials` and a value of `LG`, and another variable with the name `sub.application` and the value `CUS` (meaning customer), then the following examples are possible uses of these variables:

- `${application.initials}ICUS01` is resolved as LGICUS01.
- `${application.initials}I${sub.application}01` is resolved as LGICUS01.
- `${application.initials}${sub.application}` is resolved as LGCUS.

2.1.3 IBM CICS Configuration Manager

IBM CICS Configuration Manager for z/OS (CICS CM for z/OS) provides mechanisms to deploy and manipulate easily CICS resource definitions in a controlled and auditable manner. In particular, managers of CICS systems can prepare source and target region combinations, which are known as *migration paths*, to define the transition of resource definitions through various environments. Managers can establish transformation rules for CICS resource definitions, which enable automatic changes to resource definition attributes based on the requirements of the target environment. CICS CM includes features for standards enforcement and security controls, which makes it possible to delegate CICS resource changes directly to application programmers while still maintaining centralized control. This allows rapid deployment of the resource definitions, within the identified systems security constraints.

Using the CICS CM plug-in for CICS Explorer, developers build and start what are known as *change packages*. With these packages, the developer defines CICS resource definition changes that are required by the current application release, including new, altered, and deleted resource definitions. With the package defined, it can progress through a continuous delivery pipeline along the dependent code changes.

The CICS CM plug-in for CICS Explorer can be installed through IBM Installation Manager or the Eclipse's p2 installation process in the same way as CICS Explorer.

2.2 Source control management systems

At a minimum, SCM systems provide backup and version management of source code and related assets. Different SCMs work differently, with some requiring files to be locked, edited, and only unlocked after a change is made, and others permit editing by many users and then resolve any conflicts when the changes are integrated (the latter method allowing for faster, more concurrent development). Some SCMs also provide their own build infrastructure or integrate tightly with build tools, creating a complete application development solution.

The benefits of SCM systems vary depending on the system that is used:

- Share source code changes with other team members
- Notify team members when changes are made to certain source code components
- Create snapshots for application builds or releases
- Work on the same codebase independently of other developers and merge the changes together at the end

For the examples that are provided in this book, we use IBM Rational Team Concert as our SCM tool to manage source code changes.

Rational Team Concert uses a centralized repository model. It offers command-line, Eclipse, and Microsoft Visual Studio IDE clients for developing code. It permits concurrent editing and encourages backups of undelivered changes in personal on-server repositories known as *repository workspaces*. Rational Team Concert has tight integration between its own SCM system, continuous integration (CI) server, and build engines. Alternatively, it can integrate with external CI servers, such as Jenkins CI. Rational Team Concert also provides project planning and defect tracking capabilities that integrate with its SCM system, providing a direct audit trail between code changes and the related work item.

To provide direct SCM support in CICS Explorer, the Rational Team Concert client for Eclipse can either be installed into the same instance as CICS Explorer through IBM Installation Manager or Eclipse's p2 installation process in the same way as CICS Explorer.

2.3 Testing systems

One of the primary principles of DevOps is to amplify feedback loops. This activity can occur at the start of development by using testing techniques that allow the developer to understand quickly the impact of changes and verify the behavior of code with test frameworks and systems.

Eclipse and products that are based on Eclipse (such as Rational Developer for z Systems and CICS Explorer) contain the toolset known as Java Development Tools. These tools provide integration for running JUnit tests to unit test Java code (Figure 2-6) and promote test-driven development (TDD). Additionally, many libraries support test methodologies, such as behavior-driven development (BDD), and techniques such as mocking and stubbing for Java code. In CICS, this is applicable for code that is run on Liberty or OSGi JVM servers.

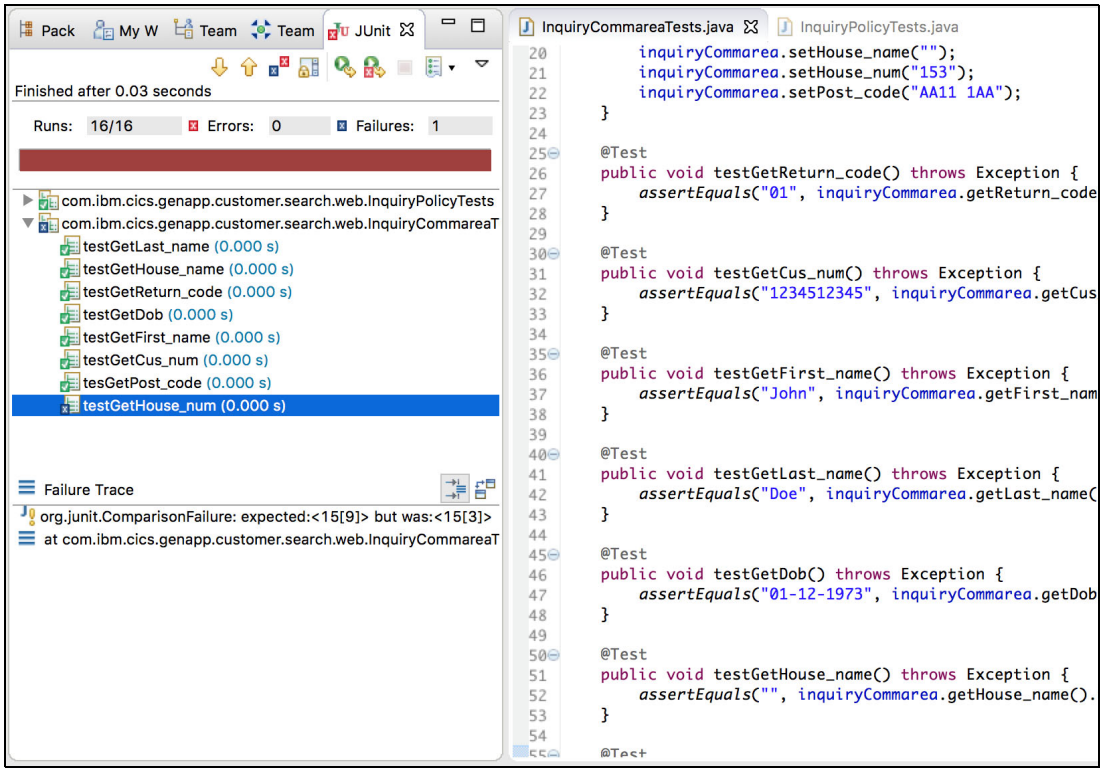


Figure 2-6 Testing with the integrated JUnit support in Eclipse-based IDEs

When working with COBOL code, unit testing can be achieved through the Rational Developer for z Systems Unit Test feature, or the Rational Development and Test Environment for z Systems feature, both of which emulate z/OS to replicate a system and allow testing of code before it gets to a real z/OS system.

Some tests might rely on a server-side infrastructure setup or must be run on the server. Ideally, this testing can be achieved through automation for quick and consistent setups, after which personal builds (such as can be run through Rational Team Concert) allow for in-progress code to be tested before integration, by producing personal copies of the application.

When code is delivered to the main codebase, it is important that the same tests are run on each build, so that the developer and team are immediately alerted about issues, so that they can be fixed immediately. Then, further tests such as load tests or production data tests (which might be prohibitively expensive or too slow to be of value in personal builds) can also be run. Continuous integration servers such as Rational Team Concert can run all these tests at every build or when required.

3

Build environment

This chapter describes the tools and process to automate reliably and repeatably the building of CICS Transaction Server V5.3 applications and bundles. This chapter describes these items from the perspective of a build engineer, with a focus on build automation and continuous integration.

This chapter covers the following topics:

3.1 Automated build

The agile approach brings dramatic changes in the software development process. Software development is now done in smaller pieces as a way to deliver new functions faster. There is a new continuous integration approach in which developers deliver code in chunks and software updates are released more frequently.

Automated build is a key part of continuous integration. A build is a process in which software artifacts, including source code and resources, are compiled, packaged, and made ready for deployment. Automated build is a repeatable build process that can be performed at any time that requires no human intervention. The build utility tools that are described later in this chapter make implementing automated builds easy.

A typical automated build scenario is shown in Figure 3-1 and described in the paragraphs that follow it. This chapter focuses on the build automation part of the diagram.

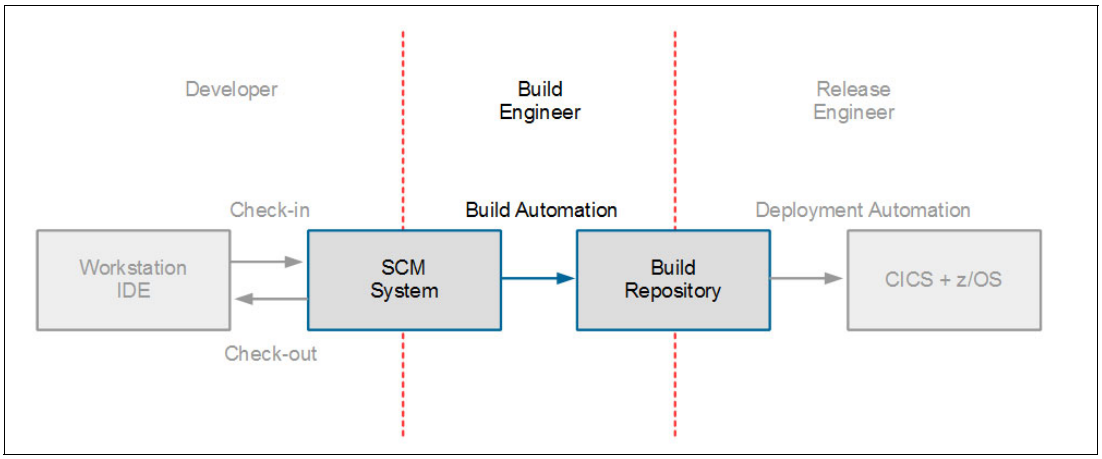


Figure 3-1 DevOps build automation

As shown in Figure 3-1, a development and operations (DevOps) build scenario follows these steps:

1. Developers check in their code to a source code management (SCM) system
2. A Build Engineer uses various tools and scripts to create a build automation process. The output of the build automation process is a set of deployable artifacts that are stored in the build repository
3. The Build Engineer also creates the post-deployment process that publishes the artifacts to the deployment repository.

3.2 Automated build in CICS Transaction Server

This section describes the tools and process that are required for build automation and how an automated build process is created.

The tools that are used for this book are listed in Table 3-1 on page 21. Different tools can be used, but we choose these products because there is good integration between them. Better integration between tools leads to better DevOps processes.

Table 3-1 Functions and tools for automated builds

Function	Tool
Continuous build	Rational Team Concert)
Build engine	Rational Team Concert
Build utility	IBM CICS Transaction Server build toolkit (CICS build toolkit), IBM UrbanCode Deploy z/OS Deploy Toolkit
Build repository	IBM UrbanCode Deploy CodeStation

Figure 3-2 shows the tools that we describe in this chapter for build automation.

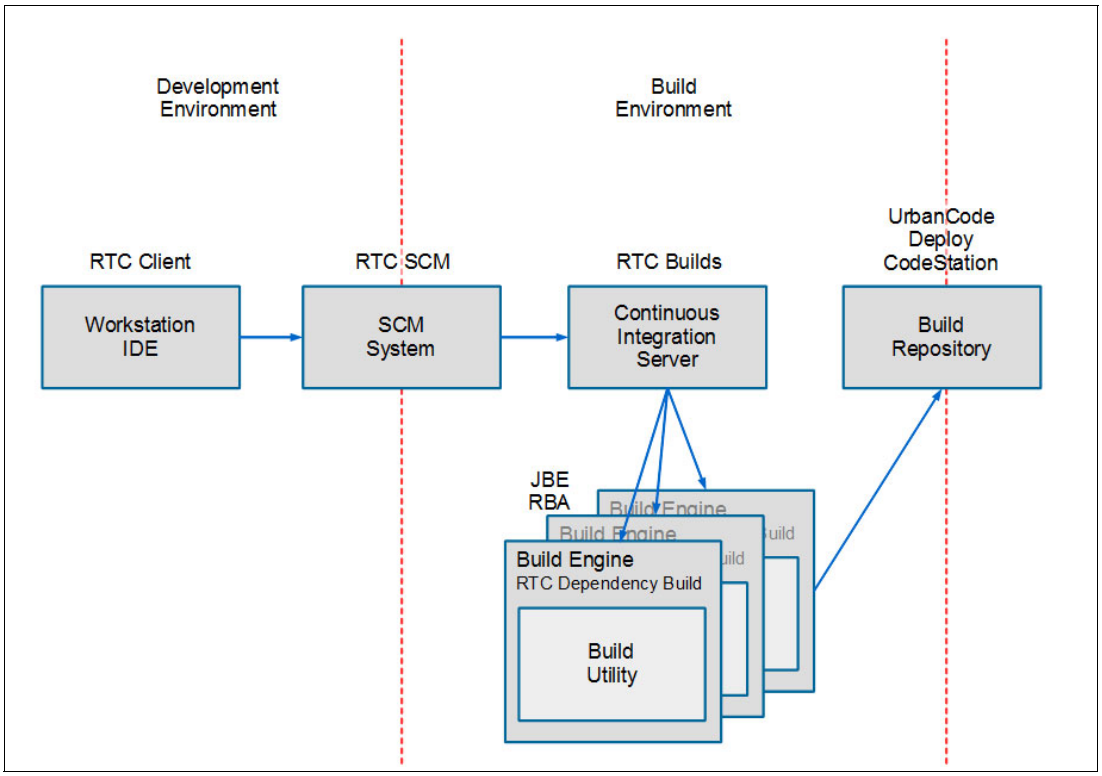


Figure 3-2 Build environment

3.2.1 Build objects in Rational Team Concert

The following types of basic build objects are used for configuring and running Rational Team Concert builds:

- Build engine
- Build definition
- Build request
- Build result

We describe the build objects in detail in the following section.

Build objects

The build component of Rational Team Concert supports the automation, monitoring, and build awareness of team builds. The team build component integrates the team's build system into Rational Team Concert, providing build awareness, control, and traceability to the team. Team members can track build progress, view build alerts and results, request builds, and trace builds to other artifacts, such as change sets and work items. Team members can see what builds there are, inspect build results, monitor builds in progress, and request builds at any time, all within the Rational Team Concert Eclipse-based IDE. Rational Team Concert Build provides a model for representing a team's build definitions, build engines, and build results. Figure 3-3 introduces the build objects that abstract, organize, and store build settings. We describe the build objects in Rational Team Concert in detail.

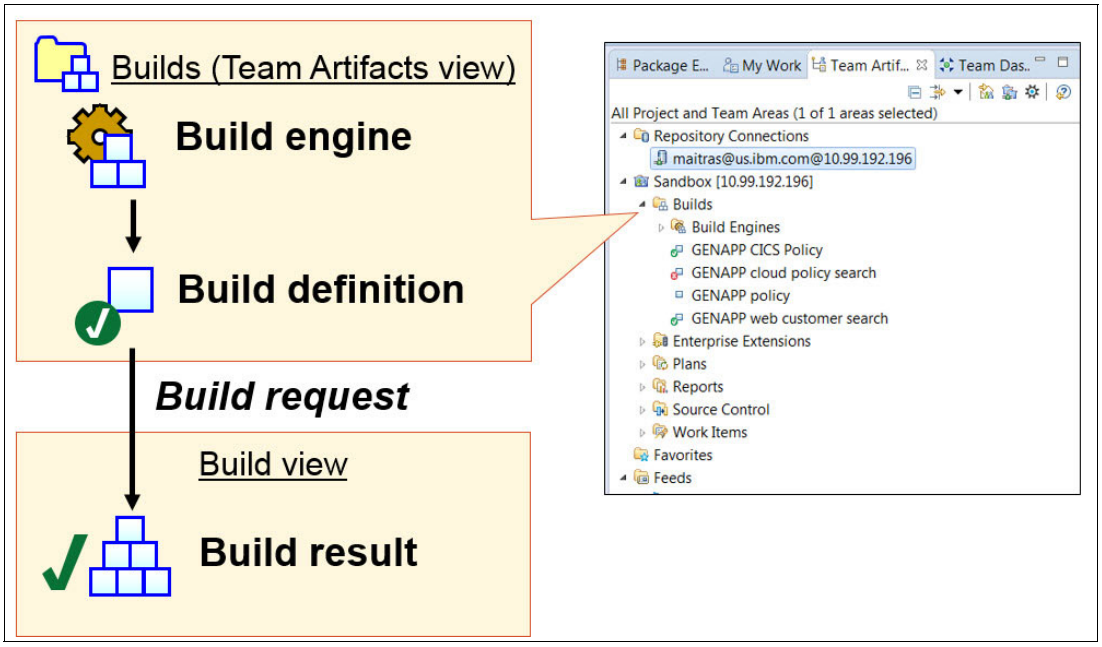


Figure 3-3 Build objects

These basic build objects are used for configuring and running builds by using Rational Team Concert Build:

- Build engine

  A build engine processes schedule and manual build requests, starts and completes builds, and publishes build logs. Build engines provide a standard way to support and interact with various build technologies. In Rational Team Concert, a build engine is an abstraction of the build technology and is how Rational Team Concert interacts with the build technology in an operational sense. The build engine runs external to the target system. It can even run on other systems, which allows load to be shared and also provides isolation for different types of builds.

- Build definition

  A build definition specifies a collection of settings and properties for a build, such as the events that must occur before the build, what is to be built and where, what happens after the build, and how the build logs its tasks.

A build definition is based on a build template, which provides base settings and capabilities to create builds for your platform, compiler, and environment. The template that is most often used for CICS/COBOL applications is Rational Build Agent - Dependency Build. The Ant IBM Jazz™ Build Engine template is used to build non-COBOL applications, such as CICS bundles.

A build definition uses a repository workspace. It fetches the artifacts from the repository to build it. This must be a dedicated repository workspace, which is owned by a designated build user.

You can configure pre- and post-build settings to specify what files to fetch from the repository workspace, and where they should be built. There are also optional settings, including automatic scheduling, email notification, and pre- and post-build command options.

- Build request

  A build request instructs the build server to run a build. Build requests can happen automatically (when triggered by schedules or by changes being delivered) or manually by human intervention.

- Build result

  The build result contains the output from a build, such as execution summaries, logs, and properties. A build result is available when the build request begins processing. When you open a build result in progress, you can view build progress from the Activities tab of the build result editor. You can also refresh a build result that is in progress.

3.2.2 Using Rational Team Concert Enterprise Extension to build CICS COBOL applications

Rational Team Concert includes support for IBM z Systems and other enterprise-level platforms. This function is unlocked through the Developer for IBM Enterprise Platforms Client Access License and includes the z/OS Dependency Build template. The z/OS Dependency Build is used to compile and link-edit COBOL and PL/I applications.

Figure 3-4 shows how the user selects the z/OS Dependency Build during the build definition process.

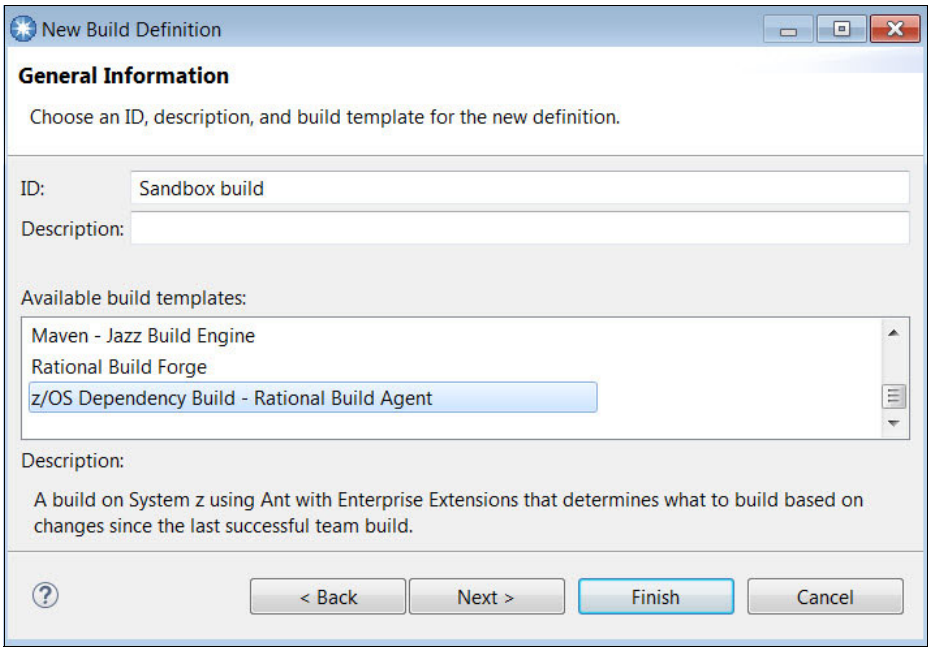


Figure 3-4 Select the z/OS Dependency Build during the build definition process

The build process for COBOL applications is shown in Figure 3-5 and described in the paragraphs that follow it.

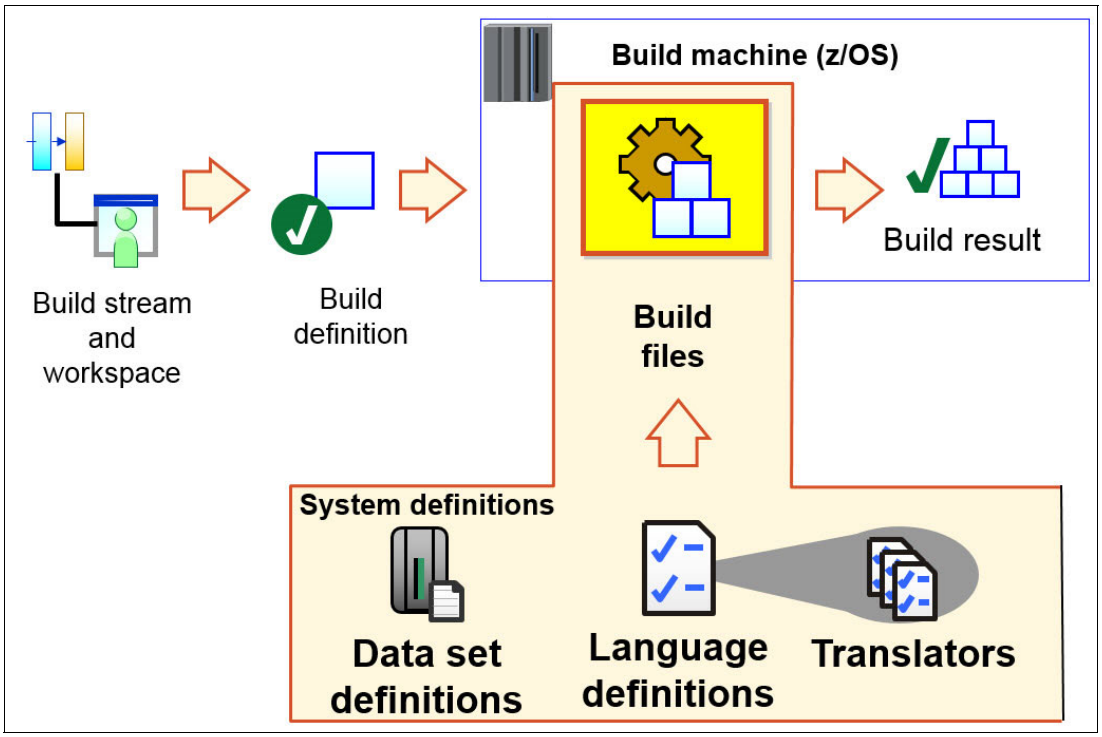


Figure 3-5 Build process for COBOL applications

Here are the steps in the COBOL application build process that is shown in Figure 3-5 on page 24, particularly regarding the Rational Team Concert server and repository:

1. The developer delivers project artifacts to a stream and then schedules an automatic build request based on the build definition. A stream is a collection of one or more components. A component is a collection of related artifacts, such as the code for a plug-in, or the group of documents that are used on a website. Artifacts under source control are grouped into components, which can be included in zero or more streams. Any group of files and folders that share a common root can be a component. Streams allow an organization to work on different versions of the same components in parallel. For example, one stream can be configured for the most recent version of a release's components, and another stream can be configured for the maintenance of an earlier version of that same software.
2. The Rational Build Agent build engine runs on the build machine, producing the appropriate build files and output. The build files are generated based on the build definition and z/OS system definitions (data set definitions, language definitions, and translators) in the Enterprise Extensions folder of the project area.

When you begin a build, the build process accepts all of the latest changes from the stream into a build-owned repository workspace and creates a snapshot of the files. The change sets and work items that are included in the build are linked to the build.

You can create build definitions to call existing host build facilities or JCL. The z/OS build agent is integrated with the Job Monitor capability to provide proper job submission and monitoring.

The Ant with Enterprise Extensions build capability is based on Apache Ant extensions to build any z/OS artifacts. The Rational Build Agent engine relies on the following artifacts:

- Data set definitions
- Language definitions
- Translators

Data set definitions

Data set definitions are containers of information about a data set or library on a remote z/OS system. All data sets that are referenced by a build process must have a corresponding data set definition. Among the data set definitions that you must create are the COBOL listing output and CICS load modules.

In this example, we create a data set definition that is named CICS STAGING LOADLIB for the staging load library that we use with IBM UrbanCode Deploy. This data set is not allocated because it is already created on our z/OS system. To create the data set definition, complete the following steps:

1. In the Team Artifacts view of Rational Team Concert, expand your project area and then click **Enterprise Extensions folder** → **System Definitions** → **z/OS**. Right-click the Data Set Definitions node and click **New Data Set Definition**.
2. Give the definition a name, in this case, CICS STAGING LOADLIB.
3. Give the data set a name, in this case, CICSDPP.SCEN1.LOAD.

Figure 3-6 shows the Data Set Definition window.

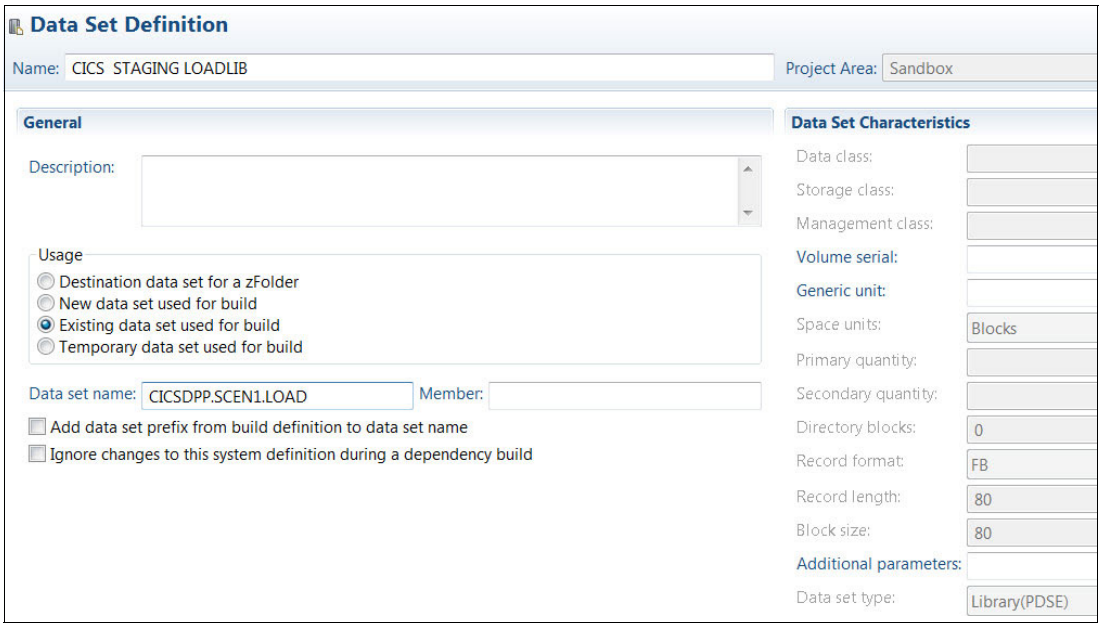


Figure 3-6 Data Set Definition window

Language definitions

Language definitions establish how to build a program. Each z/OS program that you build with Rational Team Concert has an associated language definition that contains multiple steps, represented by translations that are performed on a file during a build. Translators defines the steps that are required to build program for a specific language. One or more translators are associated with language definitions. A language definition specifies the steps, or translators, that must be performed during the build. Each program that is built in z/OS must be associated with a language definition. The tasks are defined by using a translator, such as the IBM DB2 preprocessor, CICS translator, or COBOL compile.

Figure 3-7 shows the Language Definition window with appropriate selections to create a language definition.

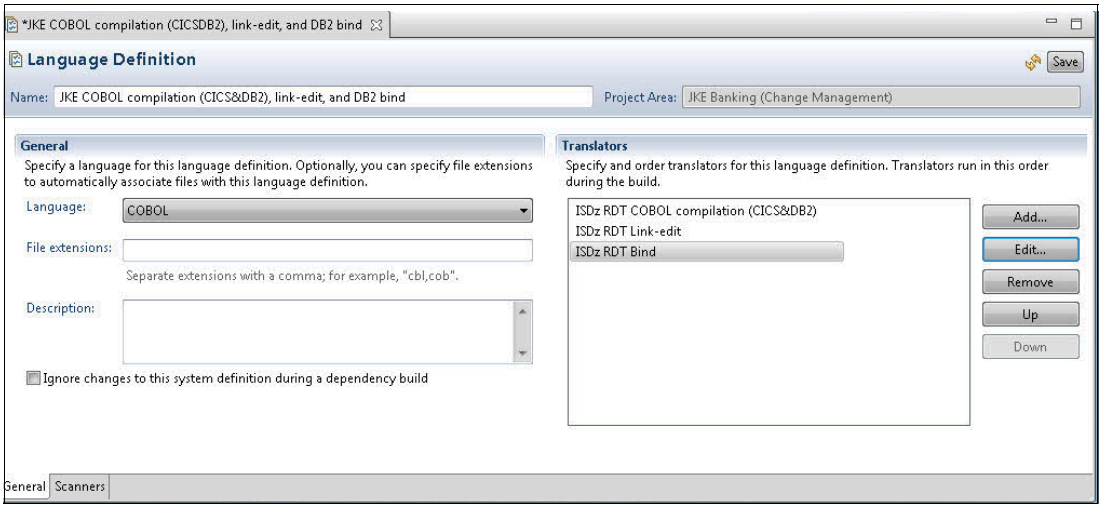


Figure 3-7 Language Definition window

Translators

As part of a language definition, translators describe the operations that are performed on a program during a build. Language definitions that contain no translators are used for simpler operations, such as collecting source code data.

A translator is similar to a JCL job step and specifies the compiler or translator to be used, the data set definition allocations that are needed by the compiler, and the maximum return codes. Depending on the COBOL compiler version that you use, the steps can also include DB2 precompile, CICS precompile, COBOL compile, link-edit, or DB2 bind.

Figure 3-8 shows a translator for the COBOL program.

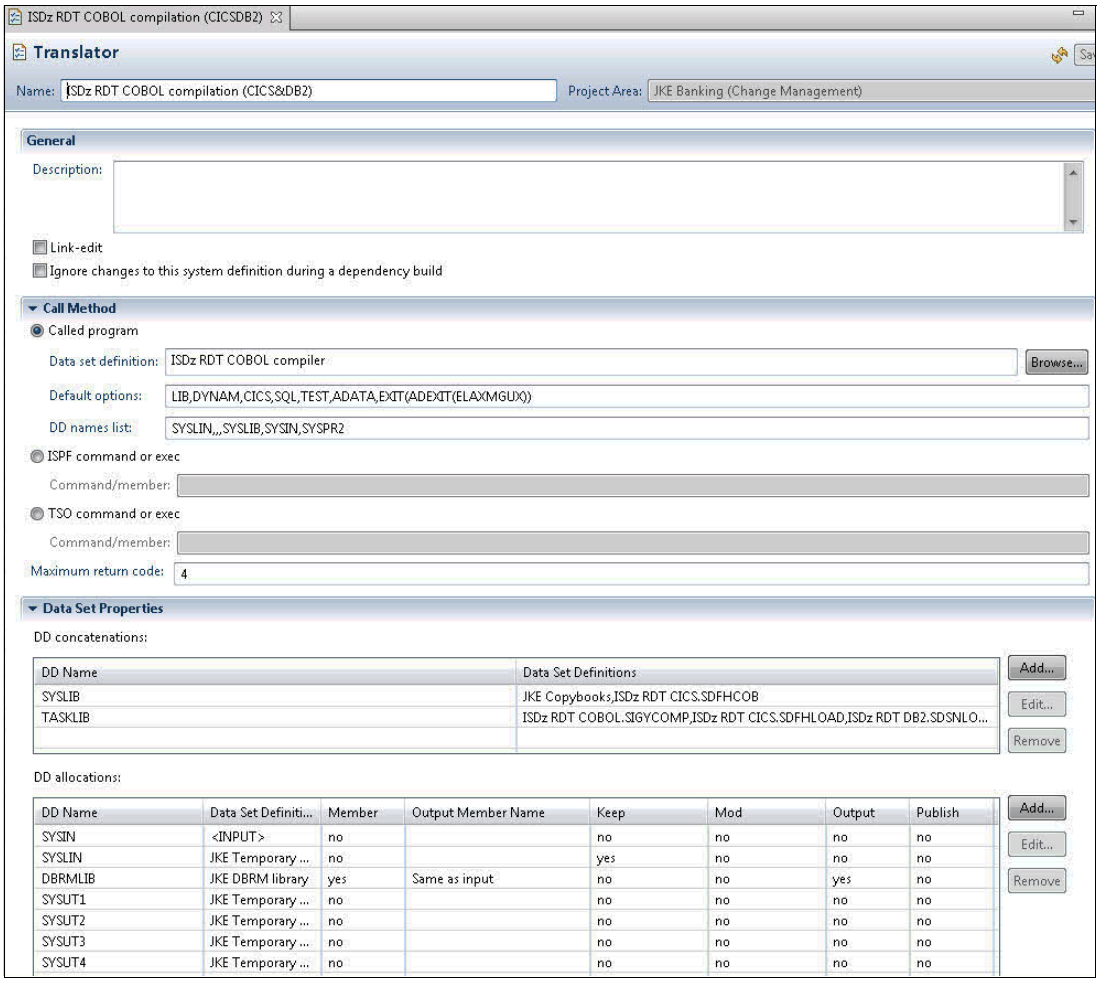


Figure 3-8 Translator for the COBOL program

Creating build definitions for z/OS

Rational Team Concert helps you define builds and manage them. Rational Team Concert Build provides a template for representing the definitions, engines, and results for team builds, including support for different build technologies. The following build definition templates are available for z/OS:

- The Ant with Enterprise Extensions build template is a build definition for z/OS that uses the Rational Build Agent build engine and a version of the Ant build toolkit that is extended to work on z/OS. The Ant build toolkit is suited for Ant builds, but you can use any scripting technology that can start Ant.
- The z/OS Dependency Build template is used to build only those items that changed or depend on changes since the last successful team build.

Figure 3-9 shows the z/OS Dependency Build tab within Rational Team Concert.

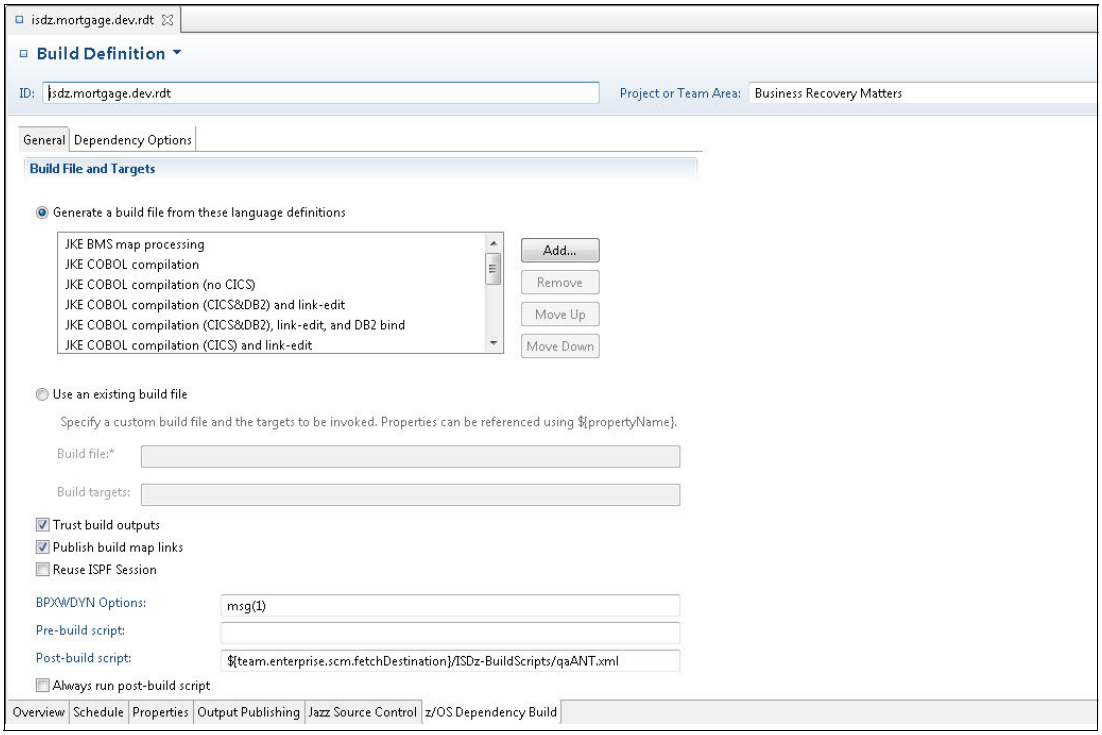


Figure 3-9 z/OS Dependency Build tab

3.2.3 Using Rational Team Concert to build CICS cloud applications and bundles

Section 3.2.2, “Using Rational Team Concert Enterprise Extension to build CICS COBOL applications” on page 23 described the process of building a CICS/COBOL application by using Rational Team Concert Enterprise Extension. This section describes how to build a CICS bundle or CICS cloud application. You create the build definition by using the Ant – Jazz Build Engine as the template, as shown in Figure 3-10 on page 29, although a number of different build templates are available for this purpose, including some that work from the command line.

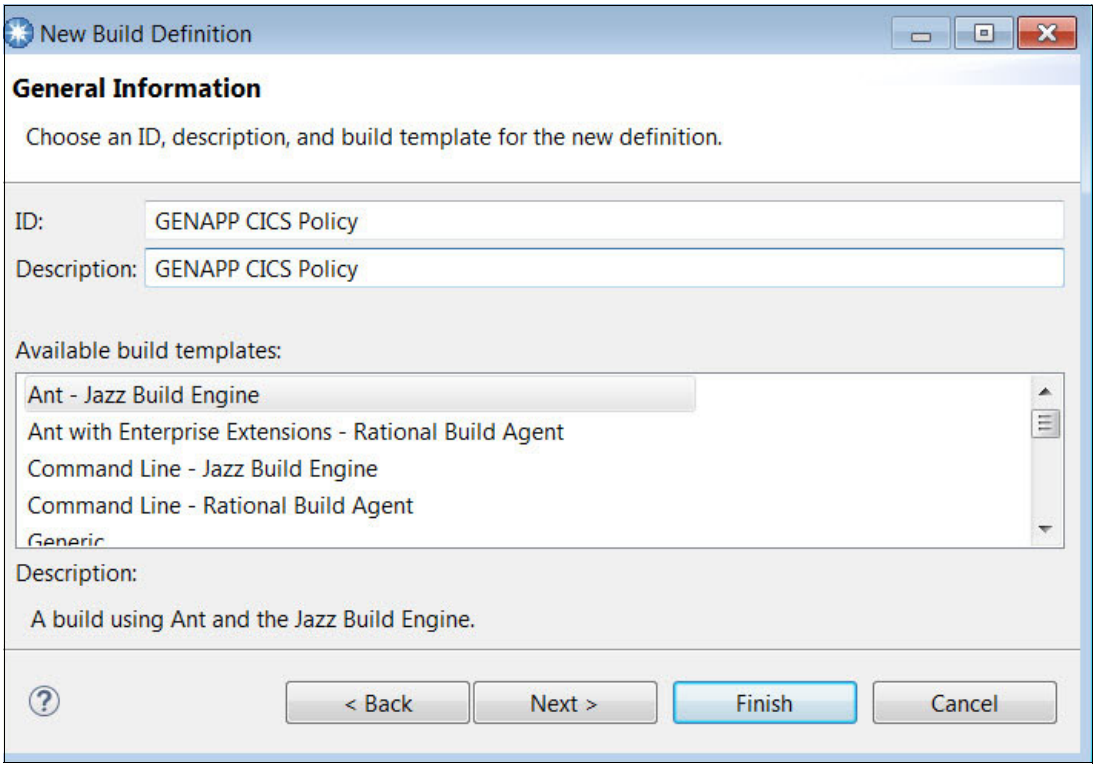


Figure 3-10 New Build Definition window showing the selection of the Ant – Jazz Build Engine

The following tabs are used to establish the key components of build definitions for CICS cloud applications and bundles:

- Properties tab
- Jazz Source Control tab
- Ant tab

In this example, we show how to create a definition for a CICS bundle by completing the following steps:

1. In the Properties tab (see Figure 3-11), specify the source directory of the bundle definition, the name of the bundle definition, and the output directory of the build. The specific entries that you make on this tab depend on the particular bundle you are creating.

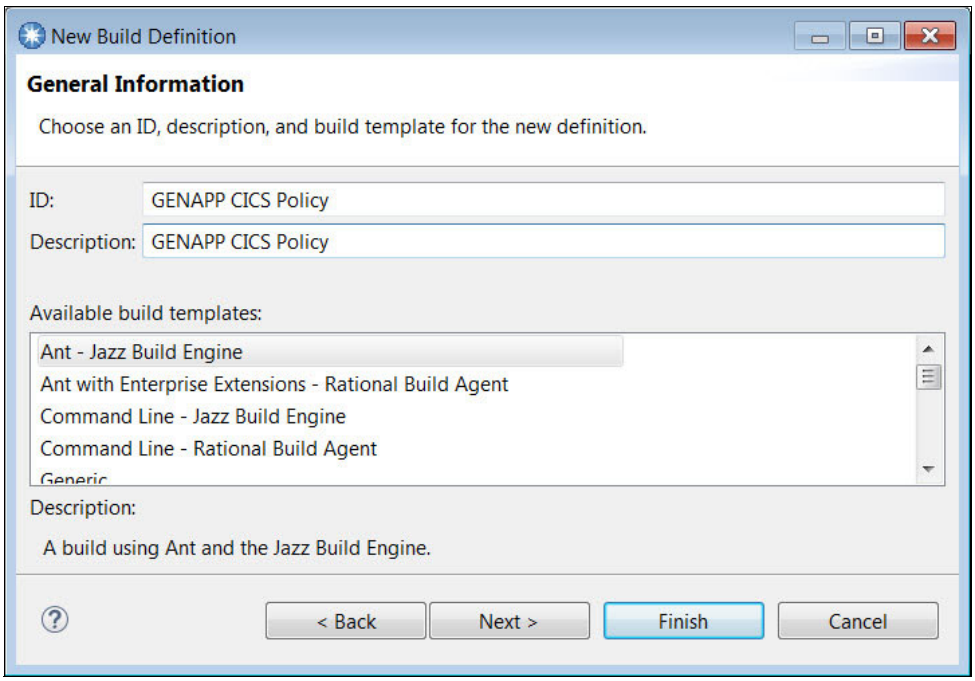


Figure 3-11 Properties tab with selections for a CICS bundle

2. In the Jazz Source Control tab (see Figure 3-12), create a repository workspace for the build. Specify the load options for the build, including the source directory and the components to exclude from the build. Your entries are specific to the bundle you are creating.

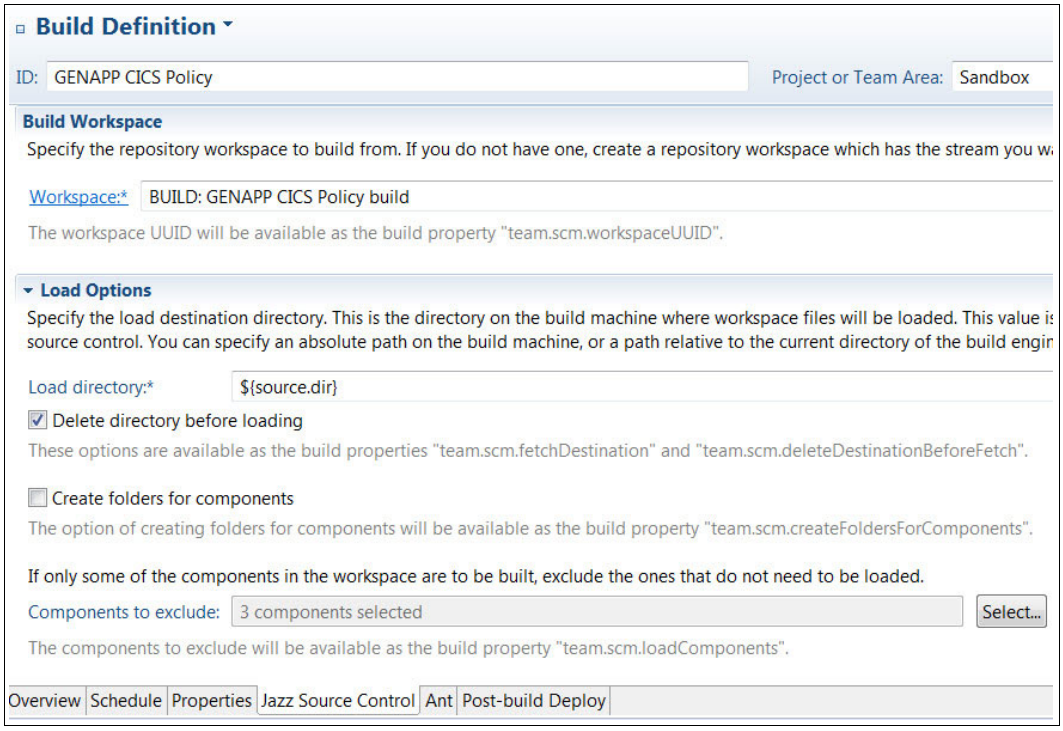


Figure 3-12 Jazz Source Control tab with selections for a CICS bundle

3. In the Ant tab (see Figure 3-13), specify the build script that runs the Ant tasks that communicate with the Jazz Team Server. The script fetches source files to compile from a stream or workspace, reports progress, and creates the build output. In the example that is shown in Figure 3-13, we designate the `build.xml` Ant script that runs the CICS build toolkit to create CICS bundles.

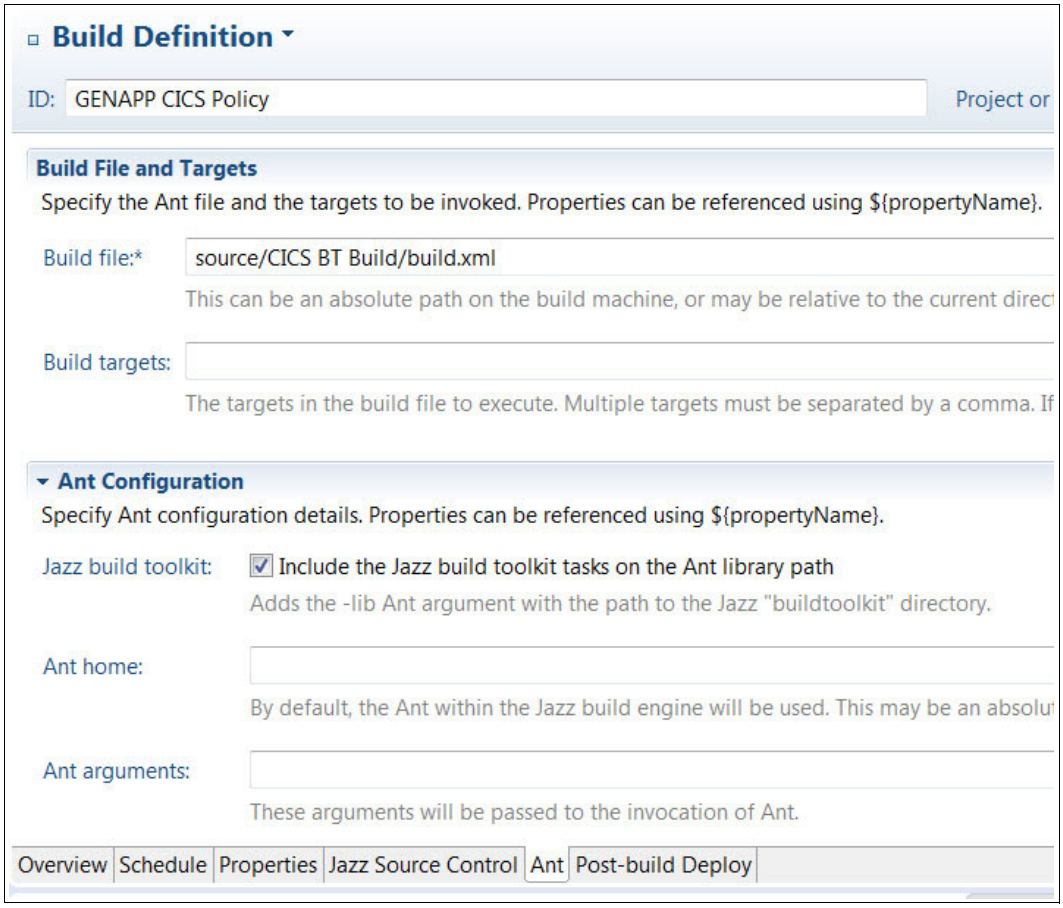


Figure 3-13 Ant Tab with selections for a CICS bundle

The Ant script `build.xml` that runs the CICS build toolkit is shown in Example 3-1. We provide details about the CICS build toolkit command in 3.3.4, “CICS build toolkit” on page 36.

Example 3-1 Build.xml Ant script that runs CICS build toolkit

```
<exec executable="${cicsbt.install.dir}/${cicsbt.command}"
      failonerror="false" resultproperty="return.code" dir="${user.dir}">
<env key="IBM_JAVA_OPTIONS" value="-Xshareclasses:name=cicsbt,groupAccess" />
<arg line="--input ${source.dir}/*
           --build ${bundle.name}
           --target ${target.platform}
           --workspace ${cicsbt_workspace}
           --verbose" />
</exec>
```

3.2.4 Using Rational Team Concert for continuous builds

One of the keys to the DevOps approach is the ability to do continuous builds without any human intervention. The continuous build concept means that developers need only to check in their code to an SCM system for a build to be triggered. Builds are triggered only if the system determines that the checked-in code contains changes from what was previously in place.

Rational Team Concert supports the continuous build process while creating the build definition. The build schedule can be configured to run at a specific interval, such as every 5 minutes.

So, to achieve continuous build, you start by setting the build definition schedule in the Schedule tab (see Figure 3-14). Within the tab, make these selections:

- Select **Enabled** to enable automatic builds.
- In the Continuous interval in minutes field, type your preferred interval (for example, 5 minutes).
- In the Build days pane, confirm that you identified the days when you want builds to run.

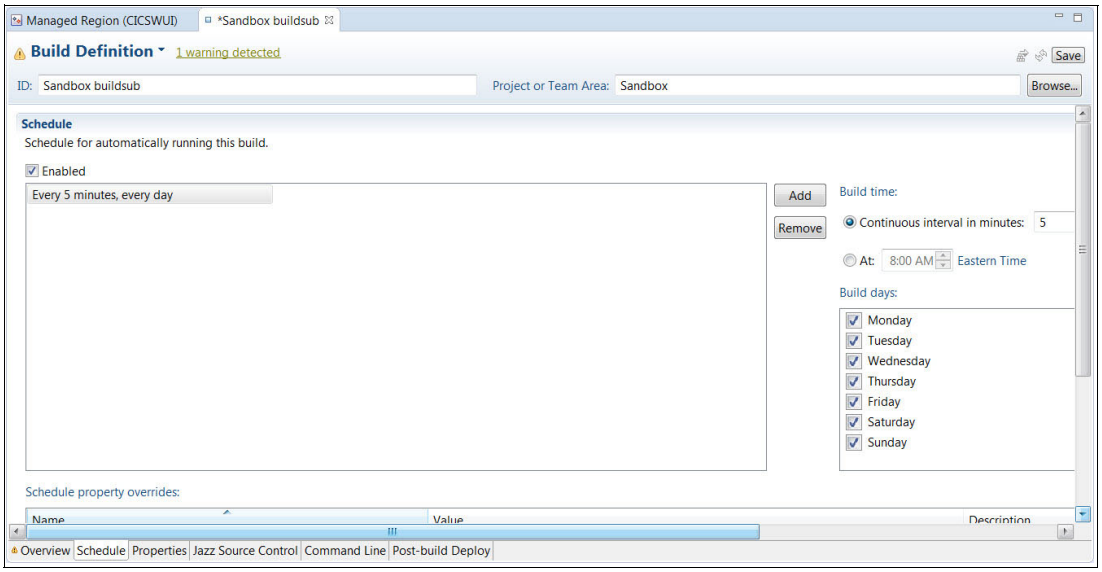


Figure 3-14 Enable and set the build definition schedule

To establish that builds be performed only if new changes are delivered to the stream, select the **Build only if there are changes accepted** option on the Jazz Source Control tab, as shown in Figure 3-15.

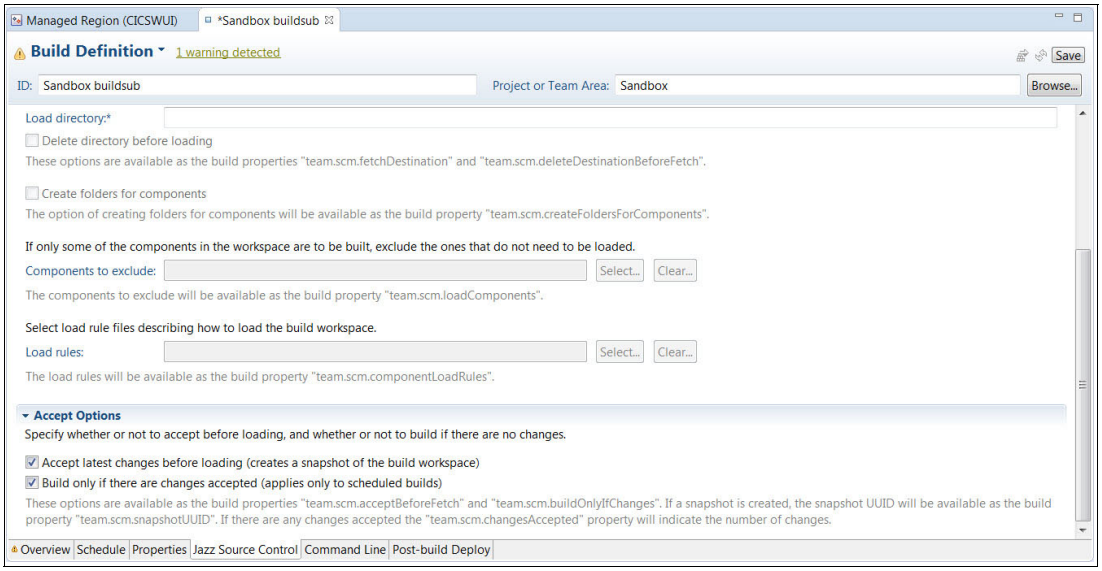


Figure 3-15 Jazz Source Control tab with a setting to perform builds only when changes are accepted

3.3 Build automation utilities

The build engine runs one or more build automation utilities, where the bulk of build activity is completed. These utilities take the source items and perform operations on them, such as compiling, link-editing, and packaging.

Particular build automation utilities often work best with particular application types or styles. For example, the make utility is often associated with C and C++ builds, Ant and Maven are typically used for Java builds, and JCL is the preferred utility for z/OS compatible languages, such as COBOL and PL/I.

When building CICS applications, you can use many different build automation utilities, depending on the application being deployed.

3.3.1 Utilities for traditional CICS applications

When a CICS application is composed primarily of COBOL load modules, use JCL to run compile and link-edit jobs. Optionally, you can use more advanced structures such as Rational Team Concert Enterprise Extension build definitions and CA Endeavor generate processors, as shown in Figure 3-16 on page 35.

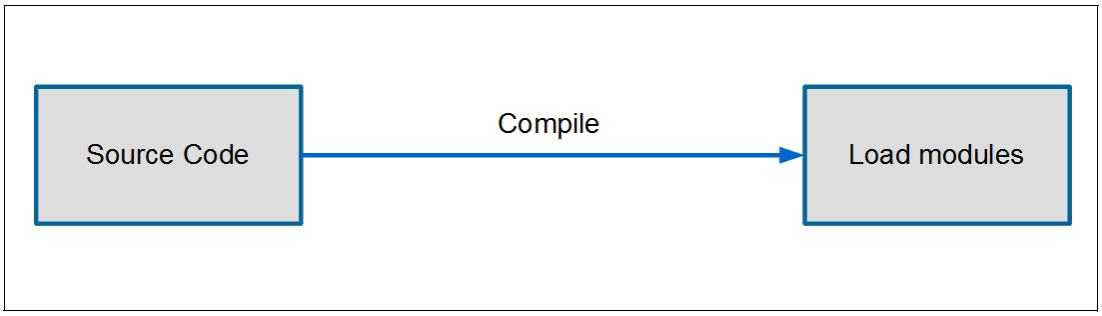


Figure 3-16 The process of building traditional CICS applications

3.3.2 Utilities for CICS bundles and cloud applications

When a CICS application is composed of CICS bundles or cloud applications, use the CICS build toolkit to build the artifacts into their installable form.

You can run the CICS build toolkit from a shell or batch script running on Windows, Linux, or z/OS by using the BPXBATCH program from JCL. You can use other tools too. For example, if you use Ant, you can run the CICS build toolkit with the **<exec>** task. The process of using the CICS build toolkit to build CICS bundles is shown in Figure 3-17.

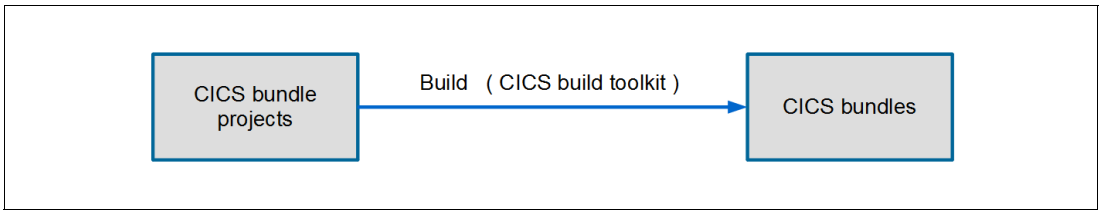


Figure 3-17 The process of building CICS bundles and CICS cloud applications

3.3.3 Utilities for CICS Java applications

When a CICS application is composed of CICS bundles or CICS cloud applications that also include Java code, you can build the Java code by using the CICS build toolkit's Java build feature. This process is shown in Figure 3-18.

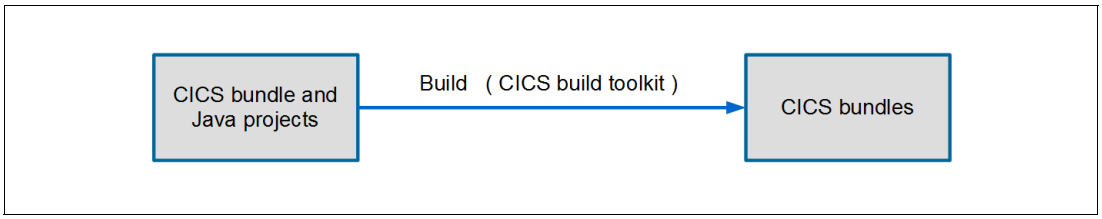


Figure 3-18 Process to build CICS Java applications by using the CICS build toolkit's built-in Java build

Alternatively, you can build the Java build separately by using other Java build methods, such as Ant, Maven, or Gradle, before you use the CICS build toolkit to finish the process. To do this task, run the external Java build as required and copy the necessary JAR, WAR, EAR, or EBA files into the relevant directory in the CICS bundle. This process is shown in Figure 3-19. WAR, EAR, and EBA files must be named in this manner:

```
symbolicname_version.extension
```

For example, name the file `org.example.website_1.2.3.war`.

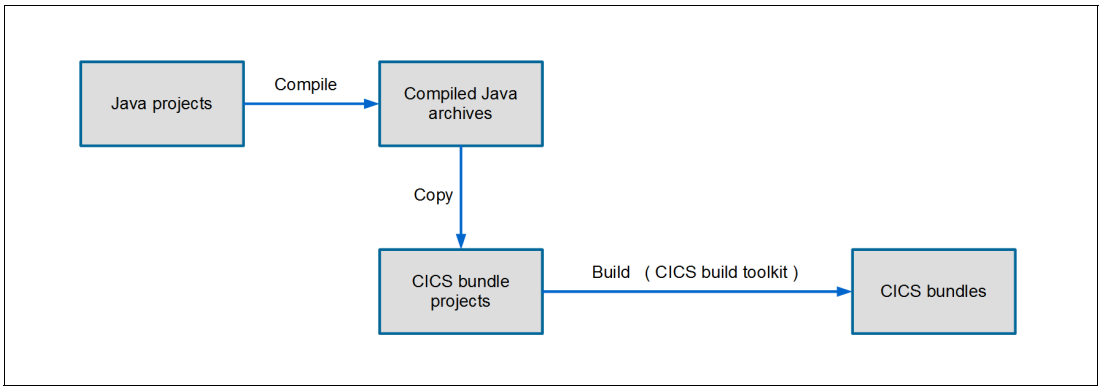


Figure 3-19 Process of building CICS Java applications, by using external Java build methods

3.3.4 CICS build toolkit

The CICS build toolkit provides a command-line interface (CLI) for automating builds of CICS projects, including bundles, applications, and application bindings. CICS build toolkit also supports projects that are referenced by CICS bundles, including OSGi applications, OSGi bundles, enterprise applications, and dynamic web projects, and can take pre-built OSGi bundles and Liberty applications as input.

Build automation

In a continuous integration environment, a build script runs automatically when a developer checks in an updated version of an application. The build script calls CICS build toolkit to build the projects that form the application. The script checks the result of the build for errors and, if appropriate, copies the built projects to a suitable location, such as an artifact repository (UrbanCode CodeStation, for example) or a staging area on zFS.

Resolving build variables

To facilitate the deployment of the same application to different environments, such as development test and production, you can replace hardcoded values in bundle part attributes with variables. A deployment script can then use the CICS build toolkit to resolve the variables with specific values, which are defined in a properties file, that are appropriate for the target environment. This process is described in Chapter 4, “Release environment” on page 43.

Installing the CICS build toolkit

The CICS build toolkit is supported on the z/OS, Linux, and Microsoft Windows operating systems and comes with CICS Transaction Server in the form of a downloadable `cicsbt_v#######.zip` file.

Software requirements

The CICS build toolkit requires a Java 7 compatible Java Runtime Environment (JRE) or Java Development Kit (JDK), either 32-bit (31-bit on z/OS) or 64-bit.

Installing the CICS build toolkit

Regardless of the underlying platform, to install CICS build toolkit, complete the following steps:

1. Download the most recent version of the `cicsbt_v#######.zip` file from the following website:

   `http://www.ibm.com/support/docview.wss?uid=swg24041185`

2. Transfer the `cicsbt_v#######.zip` file in binary format to the system on which you run the CICS build toolkit.

3. Extract the `cicsbt_v#######.zip` file to create the `cicsbt` directory. For example, on z/OS, run the following command:

   ```
   jar -xf cicsbt_v#######.zip
   ```

Building a CICS bundle, application, or application binding

Automated builds of CICS projects, including CICS bundles, applications, and application bindings, can be done with build scripts that call the CICS build toolkit. The process of creating the build automation is done after development and before deployment.

You start the CICS build toolkit from the CLI. Here are the commands for each platform:

- Linux: **`cicsbt`**
- Windows: **`cicsbt.bat`**
- z//OS: **`cicsbt_zos`**

Example 3-2 shows a sample CICS build toolkit command in Linux. It builds the latest version of the bundle by using a symbolic name and version number from an input directory indicating the location of the CICS projects and any referenced projects to be built, and uses the CICS Transaction Server V5.3 target to build the referenced Java project. The source and output files must be on the local file system where the CICS build toolkit runs.

Example 3-2 Sample CICS build toolkit command

```
cicsbt --input /top/leve/source/dir/*                        \
       --build my.cics.bundle(1.0.0)                         \
       --target com.ibm.cics.explorer.sdk.runtime53.target \
       --output /output/dir                                  \
       --workspace /cicsbt_workspace                         \
       --verbose
```

Example 3-2 includes several parameters for running the CICS build toolkit:

- Input parameters

  The input parameters (specified by **`–input`**) indicate the location of the CICS projects to be built and any referenced projects. The asterisk at the end of a path tells the CICS build toolkit to build all projects in that directory in a single call.

- Build parameters

  The build parameters (specified by **`--build`**) indicate which CICS bundles, applications, and application bindings to build from the source location.

  A project can be specified with either the symbolic name and version (for example, `my.cics.bundle(1.0.0)`), or path to the project (for example, `/u/maitra/applications/myApplication`). If you specify a symbolic name but do not specify a version, the CICS build toolkit builds the highest version of the project that is available. Therefore, use symbolic names.

- Output parameter

  The output parameter (specified by **`--output`**) indicates the location of the output directory. The CICS build toolkit creates subdirectories for applications and bundles by using names based on their symbolic name and version. If only applications or bundles are built, they are placed in the `/applications` and `/bundles` subdirectories. If one or more application bindings are built, then all associated artifacts are placed in a directory structure that includes the platform name. Any applications or bundles not associated with a binding are placed in subdirectories, as is usually the case.

- Target parameters (optional)

  The target parameters (specified by **`--target`**) indicate the target platform that is used. The target platform defines the Java libraries (APIs) that are required to build referenced OSGi bundles and OSGi applications. The target platform can be predefined, such as a CICS release, or it can be an Eclipse `.target` file that is created by the user.

- Encoding parameter

  The encoding parameter (specified by **`–-encoding`**) indicate the Internet Assigned Numbers Authority (IANA) character set name that represents the default CCSID for the CICS region into which the bundle is installed. This parameter is optional. If it is ignored, the default of `cp037` is used.

- Workspace parameter

  The workspace parameter (specified by **`--workspace`** along with a path name) provides the path of the Eclipse workspace that the CICS build toolkit uses. If this parameter is not included in the CICS build toolkit command, a temporary Eclipse workspace is created and then deleted after the command completes. It is important to specify a unique workspace directory if you are running multiple builds concurrently.

- Verbose parameter

  The verbose parameter (specified by **`--verbose`**) is ideal for debugging, as it provides extra details to the console about the progress of the CICS build toolkit run.

3.4 Publishing artifacts to IBM UrbanCode Deploy

After the application is built and packaged, it must be available in the deployment repository before the automated deployment process can proceed. In our example, the deployment repository is the IBM UrbanCode Deploy CodeStation, so a new component version must be created.

Based on the type of build to be performed, different processes are used to create the component versions. This section shows the steps for creating a z/OS component version and a CICS bundle component version.

3.4.1 Creating a z/OS component version

In IBM UrbanCode Deploy, deployable items are combined into logical groupings called *components*. Components have versions. Component versions in UrbanCode CodeStation are created after the build is done.

The IBM UrbanCode Deploy z/OS Deploy Toolkit includes utilities that help deploy the build artifacts to component versions. The utility that we use in our example is called Buztool. Buztool is a UNIX shell script that can be run from JCL, TSO, or z/OS UNIX System Services.

Buztool needs the load libraries, database resource management (DBRM), if any, and ship list file as input. It creates the component version, as shown in Figure 3-20.

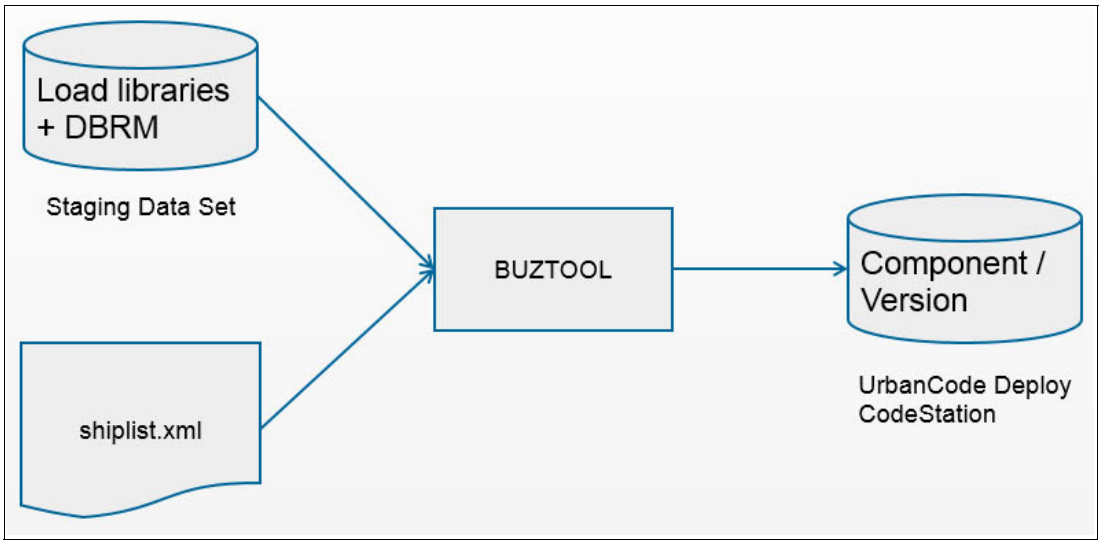


Figure 3-20 Buztool flow for creating the component version

Example 3-3 provides an example of the JCL that runs Buztool to create a component version in IBM UrbanCode Deploy CodeStation.

Example 3-3 Sample JCL to start Buztool

```
//*
//ST1      EXEC PGM=IKJEFT01,DYNAMNBR=30,PARM='%ISPFCL',REGION=0M
//SYSEXEC  DD DSN=REDS01.UCDTLKT.SBUZEXEC,DISP=SHR
//         DD DSN=SYS1.SISPCLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//PROFILE  DD DSN=REDS01.UCDTLKT.SBUZSAMP(MYPROF),DISP=SHR
//ISPPROF  DD RECFM=FB,LRECL=80,SPACE=(TRK,(2,2,2))
//ISPLLIB  DD DSN=SYS1.SISPLOAD,DISP=SHR
//ISPMLIB  DD DSN=SYS1.SISPMENU,DISP=SHR
//ISPTLIB  DD DSN=SYS1.SISPTENU,DISP=SHR
//ISPPLIB  DD DSN=SYS1.SISPPENU,DISP=SHR
//ISPSLIB  DD DSN=SYS1.SISPSLIB,DISP=SHR
//SYSTSIN  DD *
 ISPSTART CMD(BUZTOOL "createzosversion"                    -
    "-c" "GENAPP Base"                                  -
    "-v" "version2"                                             -
    "-s" "/var/cicsts/devops/scenario1/build/shiplist.xml")
/*
```

3.4.2 Creating a CICS bundle component version

After the CICS bundle is created, you must set some options on the Post-Build Deploy tab of the build definition (see Figure 3-21).

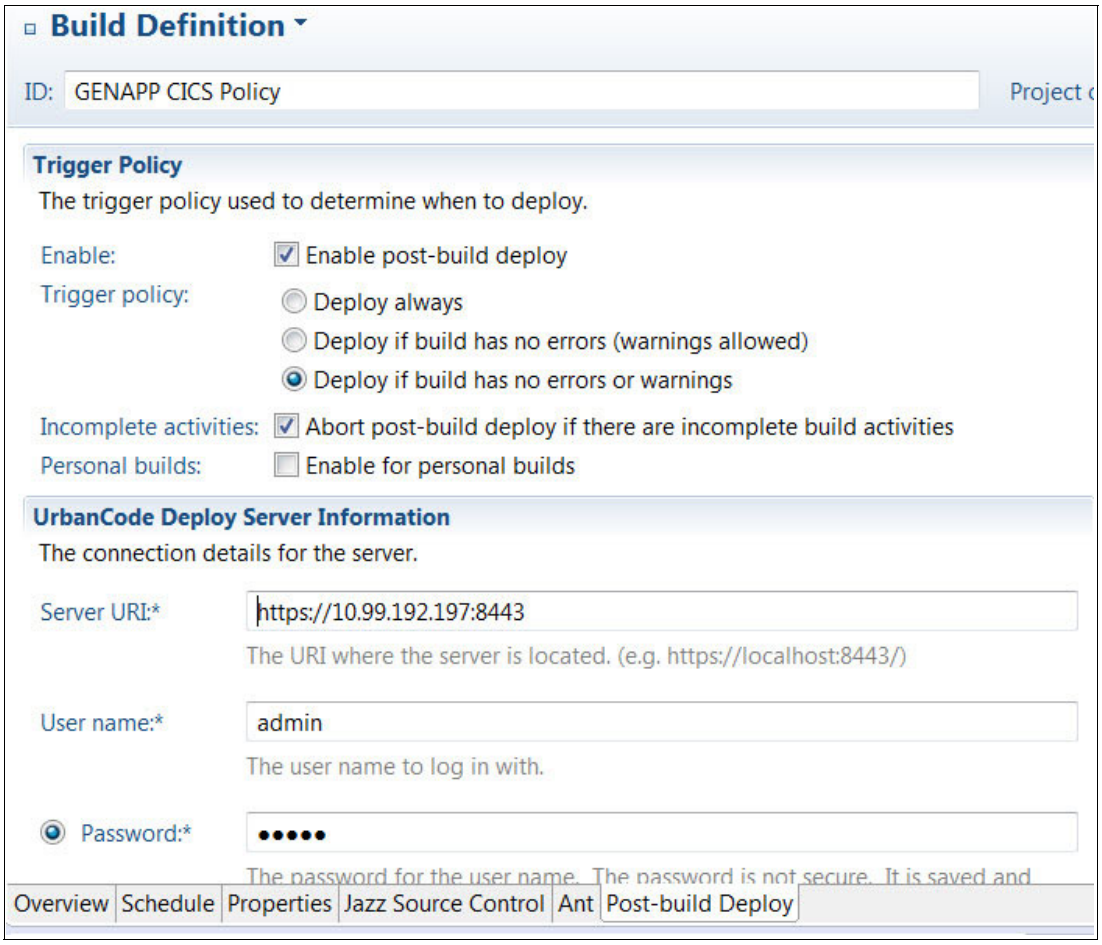


Figure 3-21 Post Build Deploy options

The Build Definition has a post-build section that you can specify what to set after the build is completed. You can automatically set up deployment to IBM UrbanCode Deploy CodeStation by specifying options in the trigger policy. On this tab, you establish the trigger policy for when to deploy the built artifacts and the IBM UrbanCode Deploy server information. Here is what we established for this example:

- Enabled the post-build deployment
- Set it to deploy only if the build has no errors or warnings

You also must provide the IBM UrbanCode Deploy server information that shown in Figure 3-21.

You must provide the component name and version, and the files that are to be included in the component version. As shown in Figure 3-22, in this example, we use the existing component. We use the version name to be built dynamically with the build label. The build label is a built-in property of the build engine. The build label is a unique label generated by the build engine, for example, 20151107-1234.

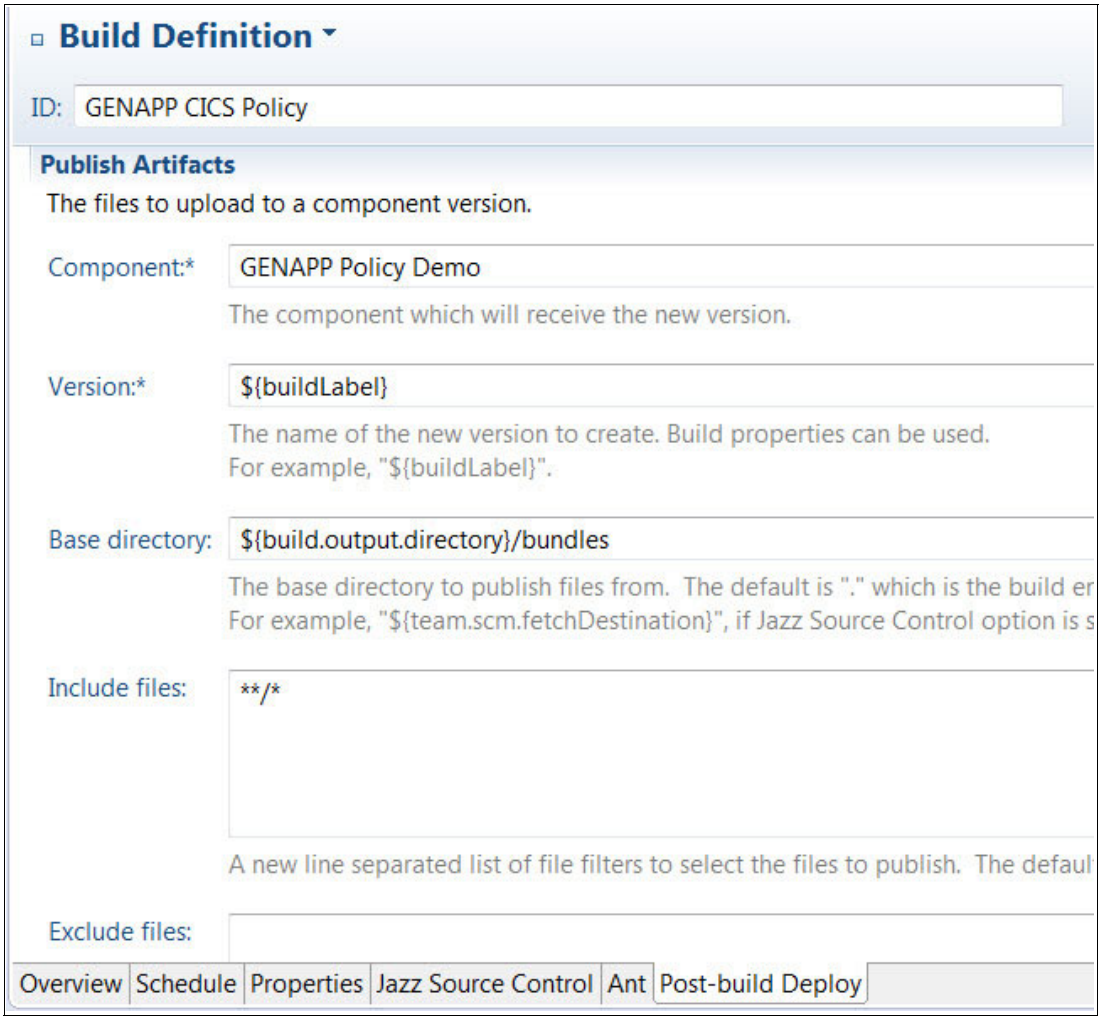


Figure 3-22 Publish new component version

4

Release environment

Software deployment refers to the process of moving applications through various stages, such as development, test, and production. At each stage, changes must be made to enable the application to work in the new environment. Managing these changes puts significant pressure on the release engineers who are responsible for application deployments.

To cope with the increasing complexity of release management within ever-shrinking deployment cycles, organizations need a more automated, streamlined release management process that is flexible, reliable, and reusable. Ultimately, to achieve true continuous delivery, you must be able to deploy an application to any environment at any time.

A wide range of tools is available to help with release management. This chapter introduces three of them. IBM UrbanCode Deploy is a multi-platform deployment tool that helps to automate and streamline application deployment across the organization. DFHDPLOY and the IBM CICS Transaction Server build toolkit (CICS build toolkit) focus specifically on application deployments within CICS.

This chapter covers the following topics:

4.1 IBM UrbanCode Deploy

IBM UrbanCode Deploy automates and speeds up software deployment through different environments. It is designed to support the DevOps approach, enabling incremental application changes to be rolled out quickly and in a reliable and repeatable manner. It includes build and test tools that can automate deployment of applications all the way to production.

IBM UrbanCode Deploy primarily fits into the Deploy stage of the whole DevOps cycle, as shown in Figure 4-1. It takes the packages from the Build process and manages the deployment of these packages through Test, Stage, and Prod environments.

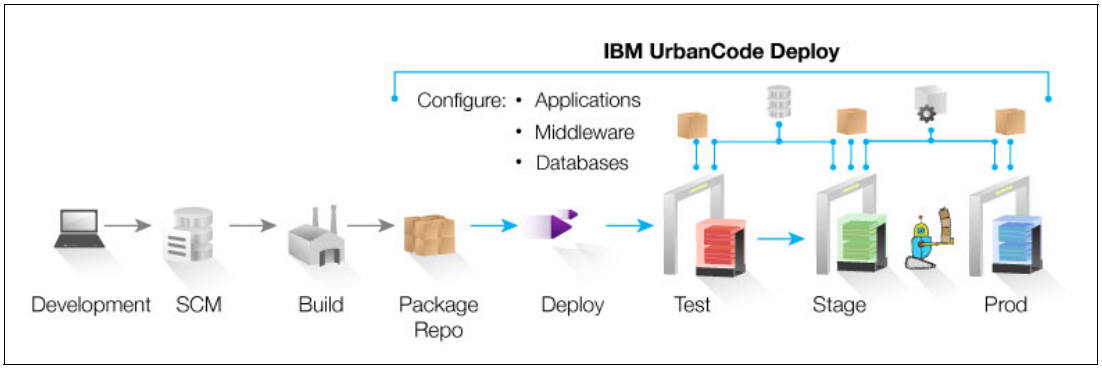


Figure 4-1 IBM UrbanCode Deploy in DevOps

IBM UrbanCode Deploy includes a browser-based Process Designer interface that is used to define various deployment items and perform deployment operations. It provides drag capabilities for defining process flows where commonly used operation steps can be dragged from a list and dropped as nodes onto the process flow diagram to represent the various operations in a process. Nodes are then connected to provide a visualization of the entire deployment flow and any interdependencies between the steps.

IBM UrbanCode Deploy also features a stand-alone server that provides most of its core services, including the user interface, workflow engine, configuration tools, access control, and so on. An agent on each target deployment system communicates with the IBM UrbanCode Deploy server and performs specific deployment that is needed in that system, as shown in Figure 4-2 on page 45.

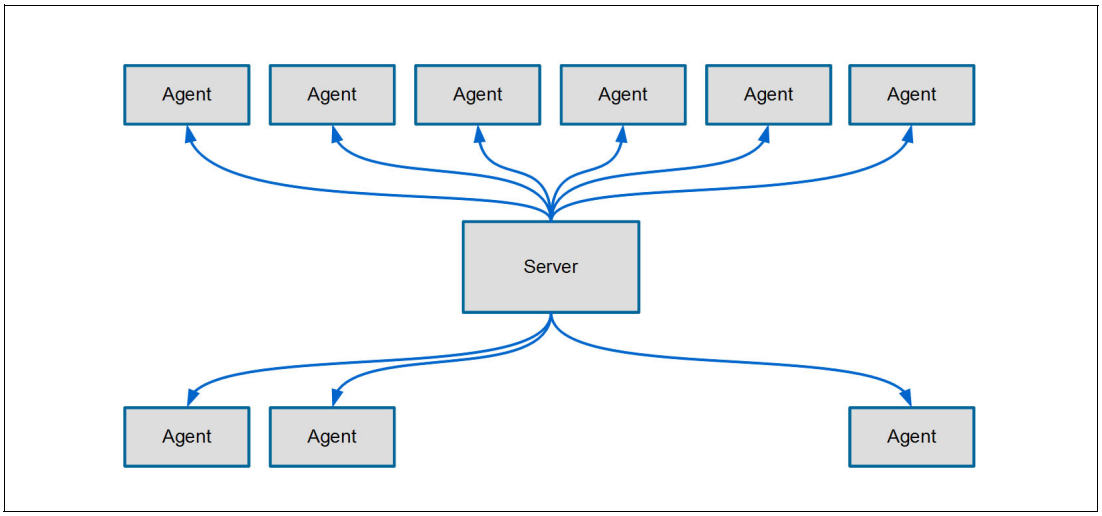


Figure 4-2 A central server communicates with a number of agents on different systems

To integrate with an evolving collection of systems and environments, IBM UrbanCode Deploy uses a series of plug-ins (more than 140 of them are already available at no cost from IBM and third parties). These plug-ins, which can be added or removed dynamically, help keep the core IBM UrbanCode Deploy installation lean while allowing users to configure the tool as needed. The agent on each target system runs the steps that are provided by the plug-in to perform whatever deployment tasks are necessary for a specific system.

Section 4.1.3, “CICS Transaction Server plug-in for IBM UrbanCode Deploy” on page 57 describes some components of the CICS Transaction Server plug-in.

4.1.1 Concepts and terms in IBM UrbanCode Deploy

To get started with IBM UrbanCode Deploy, you must learn about the basic concepts and terms that are involved, such as processes, resources, components, and applications. It is important to know how each item can help in release management.

Processes

Processes are automated tasks that run on agents and are used to model commonly used deployment flows. IBM UrbanCode Deploy provides a great degree of granularity in process definitions, and you can define a large collection of steps by using plug-ins for various operations. Steps in a process can be defined to run sequentially, in parallel, or through a combination of both.

The Process Designer tool provides an intuitive interface with which users can create processes and visualize the involved flows. To access generic processes (that is, those processes that do not apply to a particular application or component) in IBM UrbanCode Deploy, click **Processes** from the top-level menu, where you can see the list of defined processes and create processes. To access processes that apply to a particular application or component, navigate to that application or component and click the **Processes** subtab. When a particular process is selected, you can use the Dashboard, Design, Configuration, and Changelog menu to view specific information regarding that process or make changes to it.

Figure 4-3 shows a Process Designer view of a typical flow in a z/OS environment, with the CICS, IBM DB, and VSAM steps already defined.

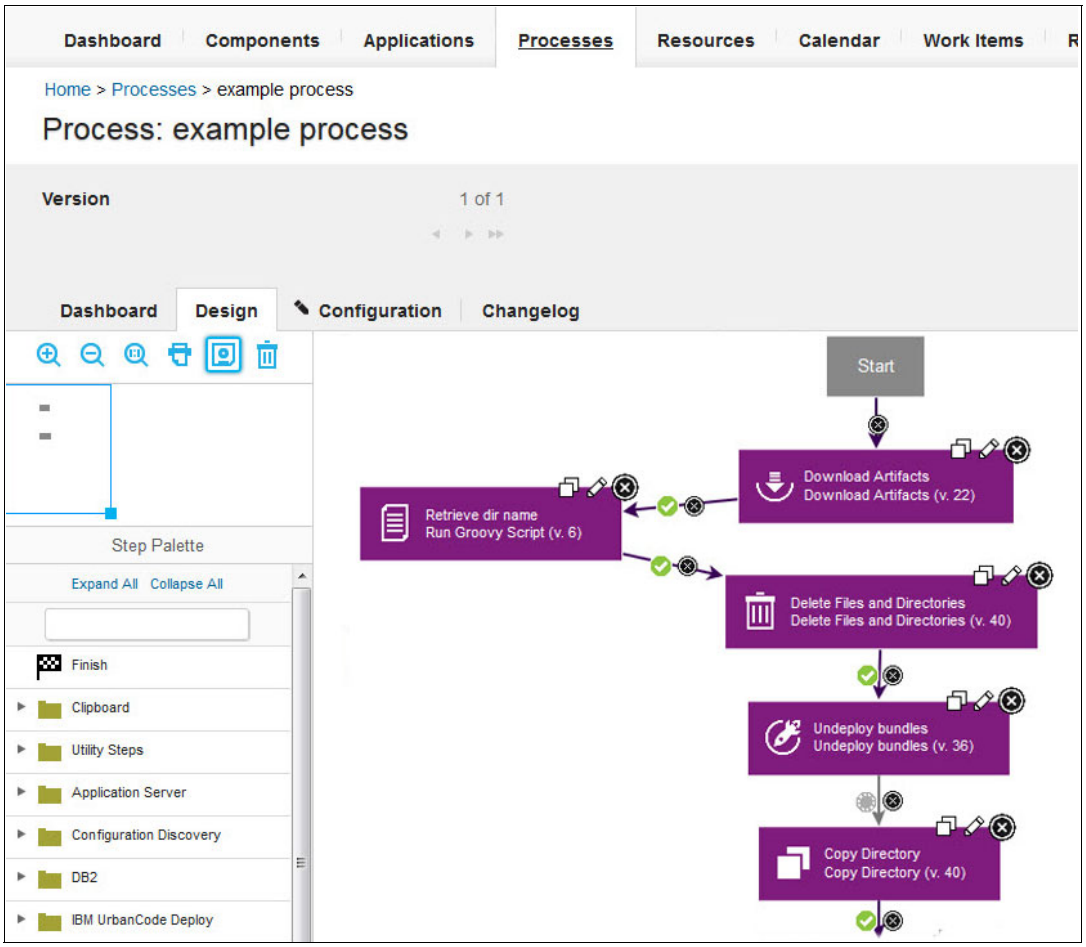


Figure 4-3 Process Designer view with CICS, DB2, and VSAM steps running in parallel

In Figure 4-3, each box represents a step. The Step Palette on the left side of the Process Designer window provides a list of available deployment steps that are grouped by category (the list is searchable so you can quickly find the step that you need). This list can be extended by installing additional IBM UrbanCode Deploy plug-ins.

The arrows that connect each box represent the sequence of the steps, and the label on each arrow indicates the dependency between the connected steps. To define a connection, hover your cursor over a step and click the blue circle that appears. By default, each step runs upon successful completion of the previous step (indicated by the green circle with a check mark). Click the circle to toggle between options for Run on success, Run on failure, and Always run. The Run on failure option is most useful when you need a recovery or back-out step when something fails.

Each step defines a specific operation, and depending on the operation, different properties and parameters might be required. For example, in a DB2 binding step, the DB2 plan, DBRM data set, and member names are required among binding parameters. You can use the pencil icon at the upper right corner of each step to view and edit the properties of that step. Click the icon to open an Edit Properties window with list of properties for that step.

Figure 4-4 shows an example of the Edit Properties window for a CICS New Copy step, which can be accessed by clicking the pencil icon on the step. The fields that are marked with a red asterisk must be completed; the others are optional and can be left blank. To learn more about each field, hover your cursor over the ? icon beside it.

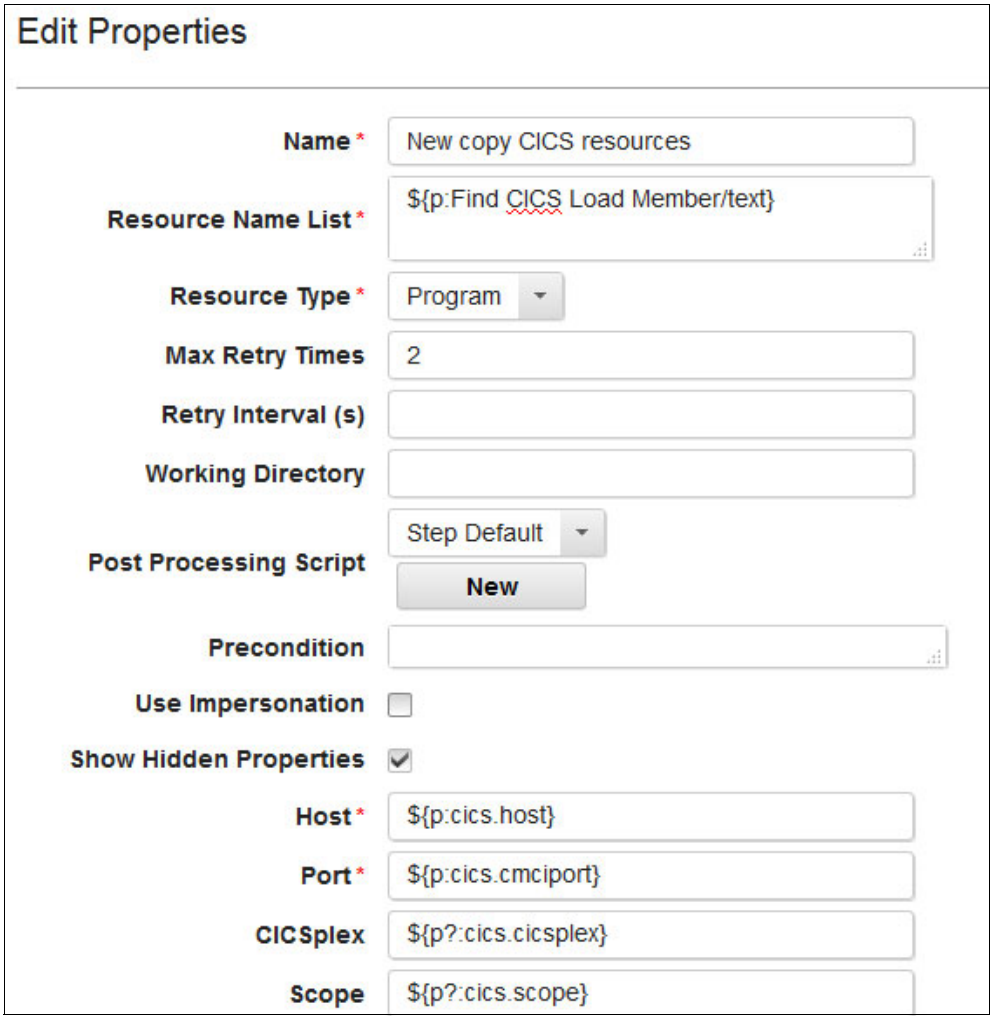


Figure 4-4 The Edit Properties window for a CICS New Copy step

The properties of a step can be hardcoded or specified by using variables. Several factors can make using variables preferable:

- Variables are reusable and can be applied to different objects without being defined every time.
- Repeated hardcoding is more prone to human error than creating a variable once and reusing it.
- Variable values can be defined in different places, such as the whole system, a particular CICS region, or a particular application, as is appropriate for the particular variable. For example, CICS related information can be defined as CICSplex or CICS region resources, and DB2 binding properties can be defined as application properties (because you are controlling DB2 application behavior).
- When defined as part of the resource tree, variable values are inherited from their ancestors. Variable values can be defined once at a high level, rather than being repetitively defined in many resources.

Variables can be used in these formats:

- For values that must be resolved: `${p:variable-name}`
- For optional values: `${p?:variable-name}`

In the data that is entered into the fields in Figure 4-4 on page 47, `${p:Find CICS Load Member/text}` refers to the text value from a step that is called Find CICS Load Member. This method enables values to be passed from one step to another.

The properties of each step are classified into standard and hidden properties. Standard properties are visible, and consist of properties that are specific to this step. Hidden properties are revealed by selecting **Show Hidden Properties**, and are properties that are expected to be common to many steps. Hidden properties should be defined elsewhere (often in resources), and referred to by variables. These properties usually do not need changing in the step, but should be considered to ensure that the variables they refer to are defined.

Resources

Users can use resources to represent topologies and systems. Users create resources, assign them relevant properties, and structure them in a tree format that defines the various interrelationships. The resources are then used as targets for deployments.

To find all resources in the web interface, click **Resources** in the top-level menu of IBM UrbanCode Deploy.

Resources form a tree-like structure, in which each resource can have its own subresources. Several types of resources can be added to the resource tree:

- Group resources are the simplest resource. They act as units of organization, allowing many subresources to be attached.
- Agent resources represent individual agents running on different machines. When deploying components to a particular resource, the agent that is an ancestor of that resource in the resource tree runs the automation.
- Component resources define the points in the topology where components are installable. For example, a Java EE web application component appropriates as a child of a resource representing a Java EE web server.

Within the tree, each type resource can be identified based on its unique icon (the tree that is shown in Figure 4-5 shows some of these icons).

Resources can be assigned properties that are used when deploying components to that resource. Each resource inherits the properties of its ancestors in the tree, which means that information about resources can be entered once, avoiding duplication and reducing the chance of errors and inconsistency when reconfiguration occurs. For example, in Figure 4-5, the Production group resource contains an agent resource. In turn, the agent resource contains a component resource.

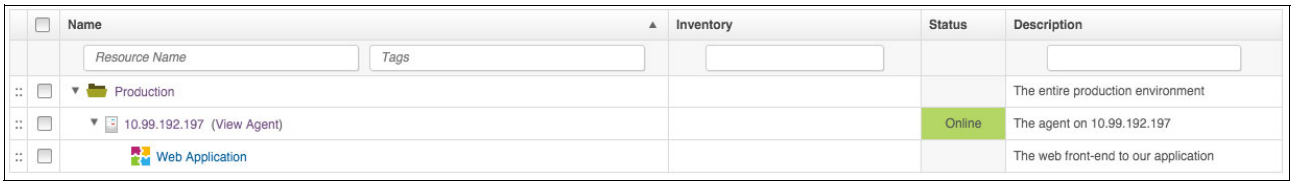


Figure 4-5 A sample resource tree

When deploying applications that include CICS components, a resource tree should be created with CICS specific concepts and components, including CICSplexes, CICS system groups (CSYSGRPs), and CICS regions. The exact granularity of the tree (for example, whether all regions are included or just CICS system groups) depends on the granularity of the applications that are deployed to it. At each level in the tree, you add properties that are used by the CICS deployment steps (provided by the CICS Transaction Server plug-in for IBM UrbanCode Deploy) to run without extensive configuration.

Any tree that involves CICS resources must include these properties:

- `cics.host` specifies the host name to which to connect. Given that the agent runs on the same LPAR as the CICS regions, this is likely to be localhost.
- `cics.cmciport` specifies the port to use to communicate with CICS. This connection uses the CMCI protocol, so CMCI must be turned on. For CPSM WUI servers, one port is required for many regions, but with stand-alone regions, each region needs a separate port.
- `cics.cicsplex` specifies the name of the CICSplex to control.
- `cics.scope` specifies a CPSM scope within which CICS commands operate, such as a particular CICS system group or a particular CICS region.
- `cics.username`, `cics.password`, `cics.ssl`, `cics.kslocation`, `cics.ksttype`, `cics.kspassword`, `cics.tslocation`, `cics.tstype`, and `cics.tspassword` are used for providing credentials, through either basic authentication or client certificates, and securing communication with SSL.

Assuming a situation where credentials for LPARs are shared across a sysplex, the base of the resource tree appears as shown in Figure 4-6. A group resource represents Sysplex H, where the credential properties are set. An agent resource represents the LPAR WINMVSH1, where the `cics.host` property is set.

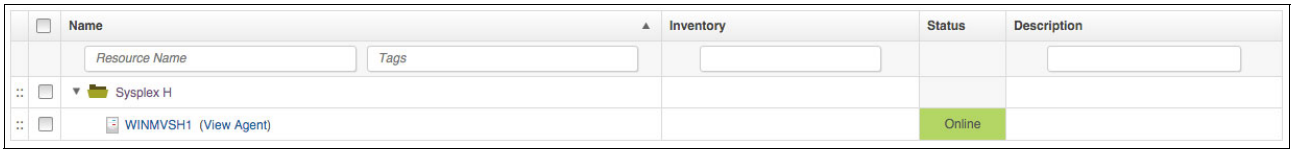


Figure 4-6 Base of resource tree for z/OS systems (sysplex has one or more LPARs with agents)

When creating the resource tree for a CICSplex, create a resource for the CICSplex and then set the `cics.cicsplex` property on it. Within the CICSplex, create the CICS system groups and set the `cics.scope` property on them. Individual CICS regions can also be created in the CICSplex in case the individual regions are ever intended as deployment targets.

If the WUI server controls more than one CICSplex, you must create a WUI server resource as the parent of the CICSplex and set the `cics.cmciport` property there, resulting in a resource tree similar to that shown in Figure 4-7, in which two CICSplexes, CICSPLX1 and CICSPLX2, are both controlled by the same WUI server. If the WUI server controls a single CICSplex, then the WUI server and CICSplex resources can be merged and the `cics.cmciport` property set on the CICSplex, as shown in Figure 4-8.

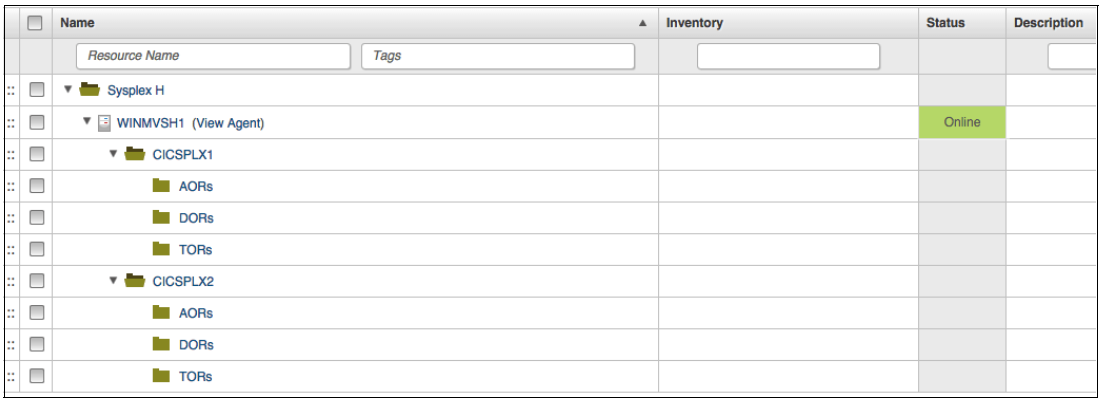


Figure 4-7 CICS resource tree in which a WUI server manages multiple CICSplexes

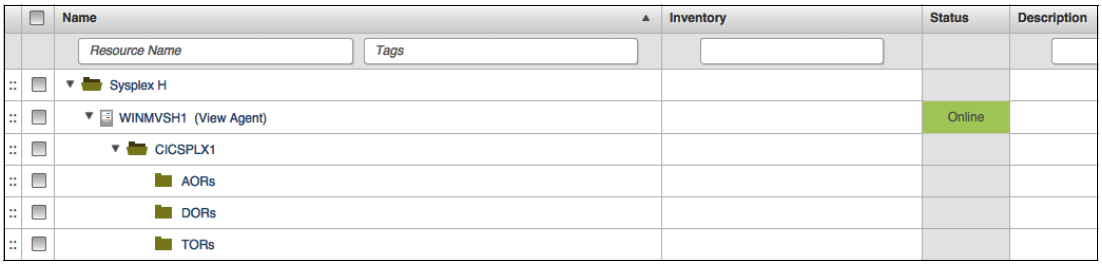


Figure 4-8 Optional CICS resource tree in which a WUI server manages a single CICSplex

When using stand-alone regions, the resource tree is different because there are no CICSplexes or WUI servers. In this case, set the `cics.cmciport` property on each region to the CMCI port of the region, set the `cics.cicsplex` name to the region name, and omit the `cics.scope` property.

Components

Components are individually deployable artifacts that in IBM UrbanCode Deploy represent logical groupings of deployable items and the processes that operate on them (referred to as *component processes*). The deployable items can come from various sources, including file systems, build servers, for example, IBM anthill Pro, source version control systems, for example, Rational Team Concert, and so on.

Components are grouped into applications that map to business-level functions. For example, you might have three components: a front-end web application running in WebSphere Application Server on UNIX, business logic COBOL programs running in CICS, and an IBM DB2 running on z/OS, as shown in Figure 4-9 on page 51.

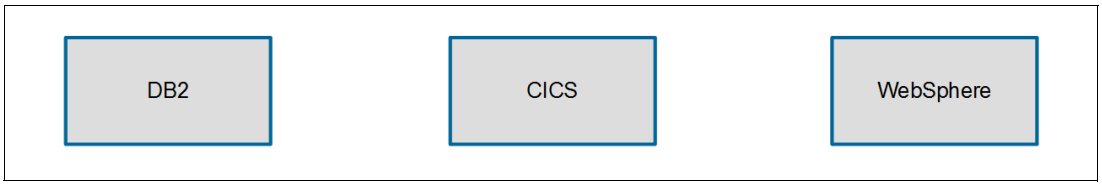


Figure 4-9 Simple application featuring web application, business logic, and database components

Component processes are a series of user-defined steps that operate on the component's artifacts, often simply to deploy them. They are created for specific components by using Process Designer, either as simple single-step processes or more complex ones involving relationships and dependencies between multiple steps. Figure 4-10 shows some typical processes of a CICS component.

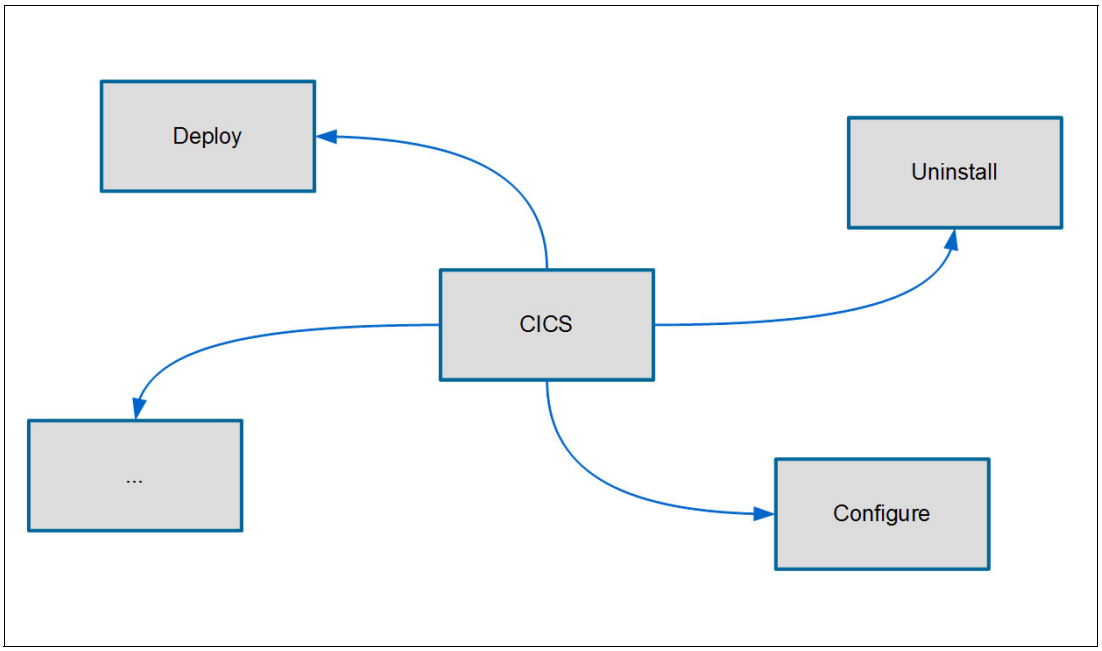


Figure 4-10 Processes of a CICS component

Versions are assigned to instances of a component to differentiate between them. Whenever a copy of a component is built, a new version number is assigned to it, and these version numbers provide the basis for the release management features of IBM UrbanCode Deploy. Versions can be full (contain all component artifacts) or incremental (contain only the artifacts that are modified from the previous version).

There are two types of component: standard and z/OS. All artifacts on distributed systems fall into the standard component category, as do artifacts on z/OS UNIX file systems. z/OS and CICS artifacts that are stored in data sets are part of the z/OS component category.

For standard components, source configuration can be used to define the source of the artifacts. All artifacts in a component must have the same source type. IBM UrbanCode Deploy can use a pull mechanism to retrieve artifacts from supported sources, such as file systems or Rational Team Concert. Alternatively, artifacts can be pushed to IBM UrbanCode Deploy by application build tools, such as the BUZTOOL utility or by using REST APIs.

Figure 4-11 shows the window for creating a component. In this example, we create a component that takes artifacts from a file system. When **Standard** is selected as the Component Type, the Version Source Configuration fields appear. You can use the drop-down list to choose from a range of supported sources as the Source Configuration Type. Most other fields can be left as the default.

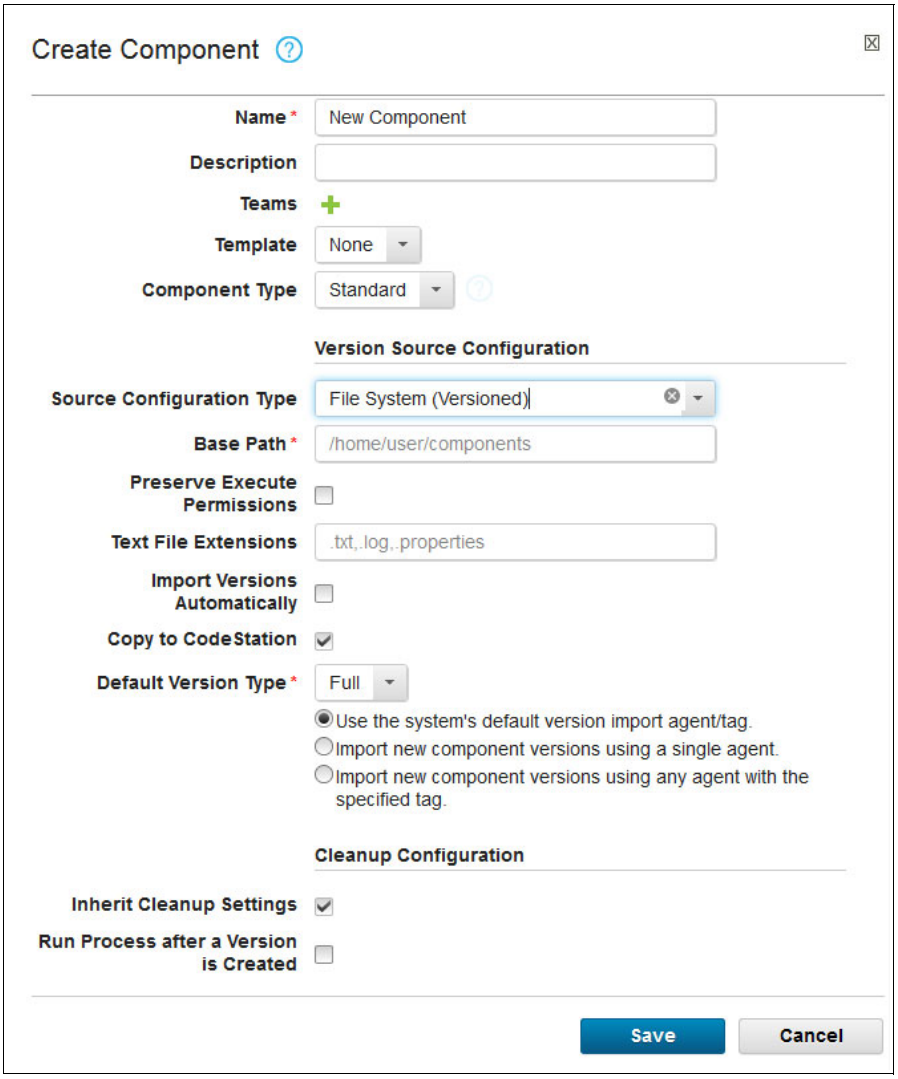


Figure 4-11 Options when creating a component

Additional details about IBM UrbanCode Deploy components are available in the product's IBM Knowledge Center the following website (search for "components"):

`http://www.ibm.com/support/knowledgecenter/SS4GSP/ucd_welcome.html`

Applications

Applications are composed of components and are defined at a business-function level. An application can be used to group one or more components on different systems to create a multi-tiered application. Applications enable the deployment of logically related components, as a whole, in an automated and managed manner. For example, Figure 4-12 on page 53 shows how the DB2, CICS, and WebSphere components are grouped into the Banking Application.

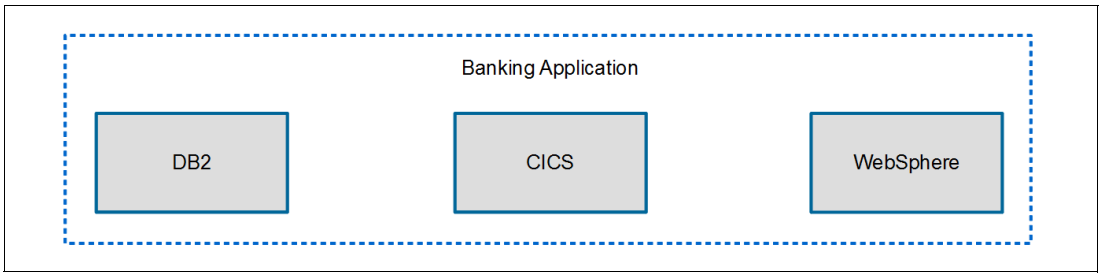


Figure 4-12 Applications group several components together

Figure 4-13 shows adding components to an application called CICS Application. Under the Application, click the **Components** tab and click **Add Component**. A window opens where you can select components from a drop-down list of existing components. Components must be defined in IBM UrbanCode Deploy before they can show up in the list to be selected.

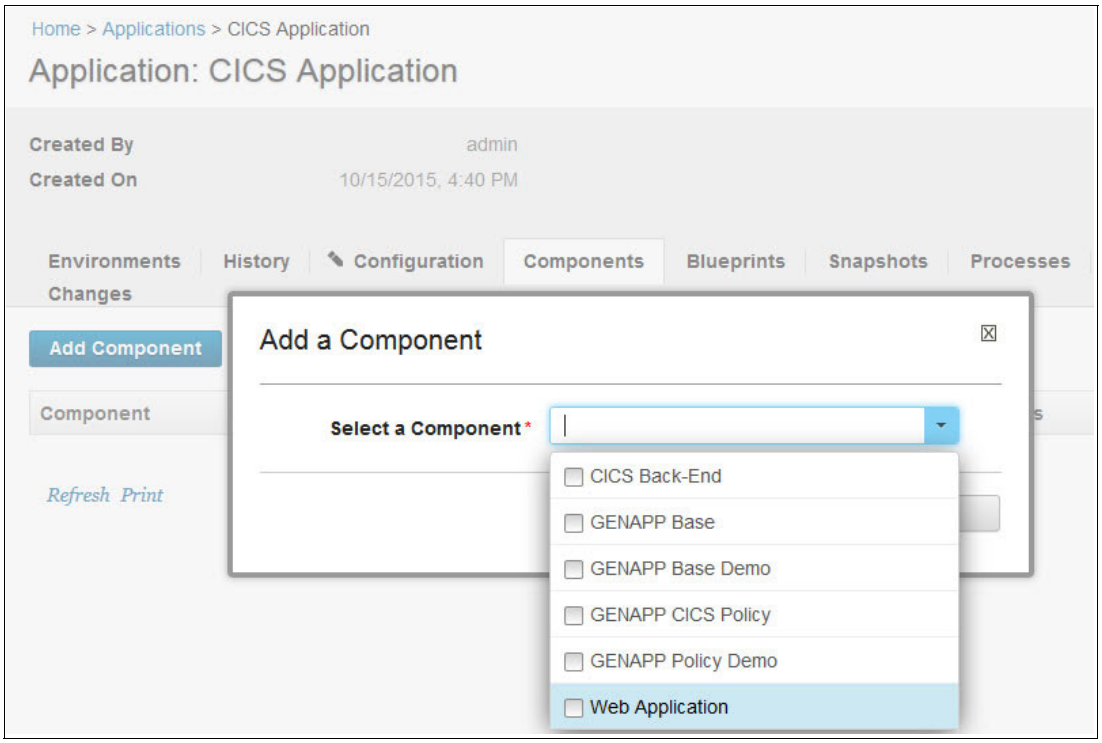


Figure 4-13 Add components to an application

Like a component, an application has user-defined processes. These processes control deployment and undeployment of the application components. At the application level, processes are concerned only with the components, not the individual deployment steps (such as copying artifacts) that make up the underlying component processes.

A typical application process consists of one or more component deployment steps to deploy the components that make up the application. IBM UrbanCode Deploy tracks the inventory (that is, whether it is installed and whether the installation succeeded) of each component version.

Application environments

Environments are another user-defined construct in IBM UrbanCode Deploy. In typical IT terms, an environment is a collection of resources that host an application. Examples include Development, Test, Production, and so on, as shown by the Banking Application in Figure 4-14. When an application process runs, it runs in a specific environment.

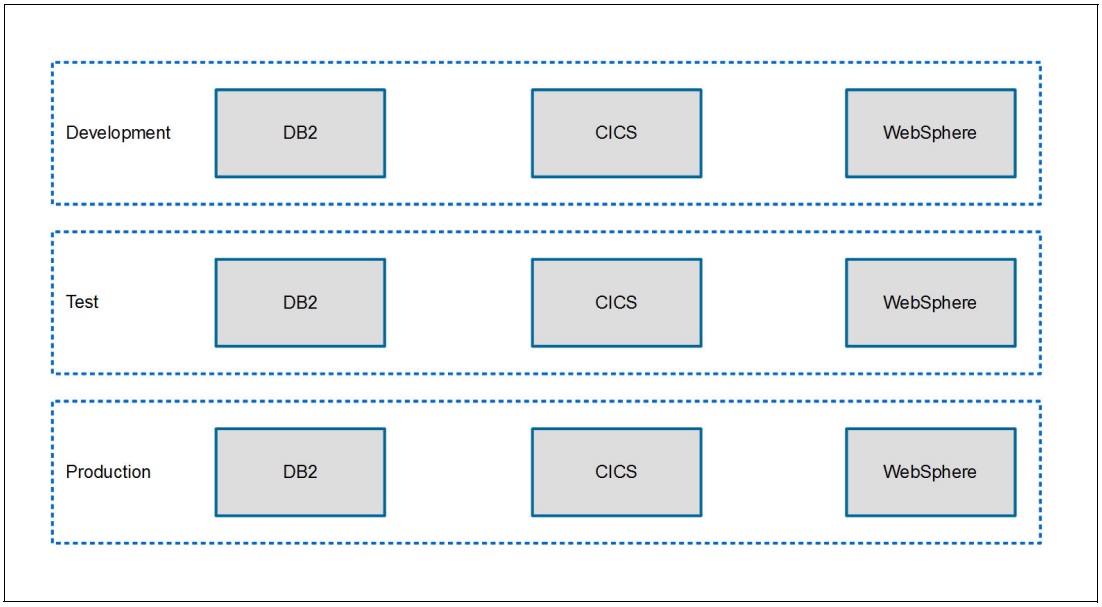


Figure 4-14 Applications can be deployed to different environments

Application environments are assigned a set of resources to which they can deploy components. For example, a development environment might have one server, a test environment might have several servers, or a production environment might have a cluster of servers. Depending on the components that are deployed to the application, the assigned resources are different, such as CICS regions for CICS application components or DB2 subsystems for DB2 schema components. Figure 4-15 shows a payroll application with three environments: development, test, and production. Each environment can be assigned a color in the web interface for better visualization.

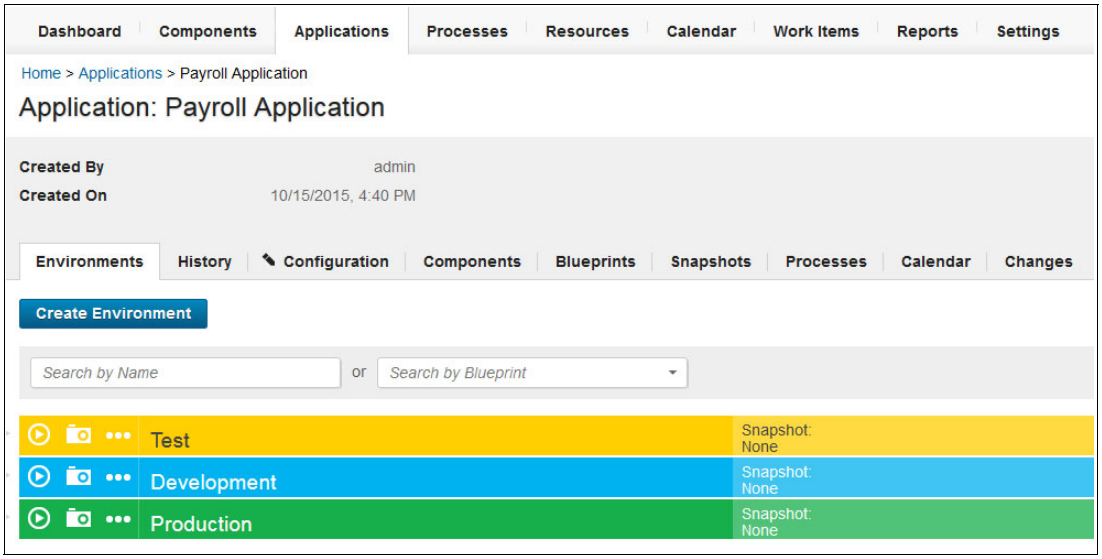


Figure 4-15 Payroll application with three environments

Environments can be assigned properties that can be used by processes to customize each deployment to the target environment.

Tags, statuses, environment gates, and snapshots

IBM UrbanCode Deploy provides many other concepts that improve the workflow and aid auditability through increased automation. For brevity, they are not covered detail here, but they include the following items:

- *Tags* are short labels that can be used to classify objects such as applications, components, and resources. They can then be viewed, and operations performed on them, as a group.
- *Statuses* are used to track component versions and the inventory states of components in environments and resources. Version statuses are used with environment gates to ensure that only component versions that meet certain requirements are deployed.
- *Environment gates* are used to set requirements that must be met before component versions can be deployed to an environment, such as passing a system integration test or receiving specific approval for deployment. Each environment gate requires component versions to have certain statuses before they can be deployed to the environment.
- *Snapshots* are a collection of specific versions of components and processes. A snapshot is taken to represent a set of component versions that are known to work together. For example, you can take a snapshot of an application with its associated component versions when the application passes User Acceptance Testing (UAT).

  As an application and its components move from one environment to another, snapshots help you ensure that specific versions and processes are used in each environment. This capability helps to manage complex deployments where there are multiple environments and development teams.

4.1.2 Defining properties in appropriate contexts

Properties can be defined in many places in the IBM UrbanCode Deploy web interface, which you can use to customize processes for the conditions in which they are run. Depending on the property, it must be defined in particular places. For example, the behavior of a component should be set in its component properties, so they are maintained as the component moves into different environments. Each environment has different configurations and such properties should be set as environment properties.

Properties can then be referred to in the steps of processes by using the following syntax:

```
${p:scope/propertyName}
```

The scope refers to the place in the IBM UrbanCode Deploy web interface where the property is defined, such as component or environment. Alternatively, the scope in which the property can be found may be omitted so that properties are referred to by using the following syntax:

```
${p:propertyName}
```

In this case, the property is searched for in a number of different places by using the following order of precedence:

- Process
- Component version
- Resource
- Agent
- Environment
- Component

- Application
- System

To illustrate this concept, Figure 4-16 shows the list of hidden properties for a CICS New Copy step. There are many properties that control how to connect to CICS and how to secure the connection. Each property is set to refer to another property, such as the CICSplex property, which refers to `${p:cics.cicsplex}`.

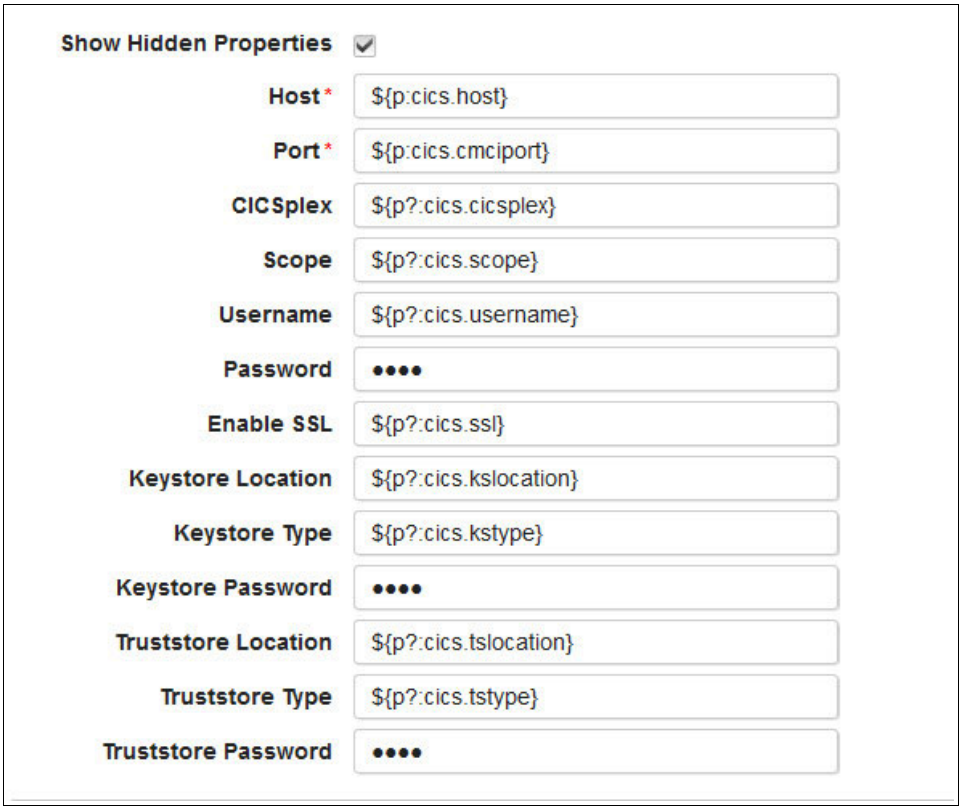


Figure 4-16 Hidden properties of a CICS New Copy step when editing it in the process editor

Many of these properties apply to the CICSplex rather than just this New Copy step, so set the values as properties of a resource in the resource tree that represents a CICSplex. They can be reused by any components, processes, or applications that are deployed to this CICSplex. Figure 4-17 on page 57 shows the property `cics.cicsplex`, along with others, set as properties of a resource named CICSPLEX.

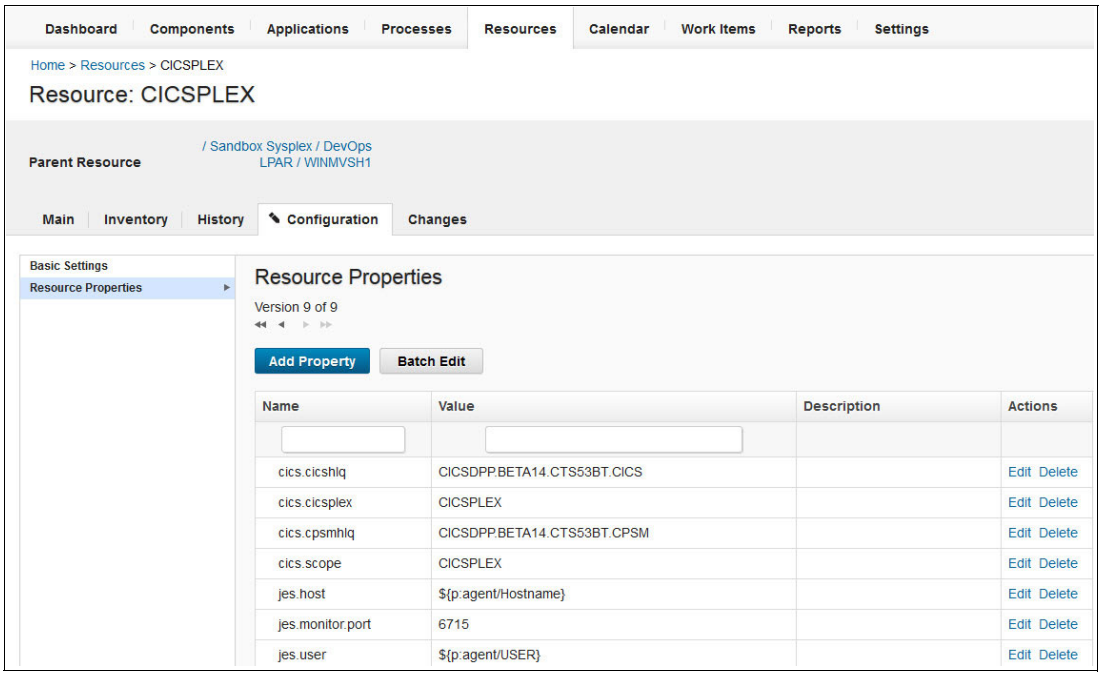


Figure 4-17 Resource properties for a CICSplex

4.1.3 CICS Transaction Server plug-in for IBM UrbanCode Deploy

The CICS Transaction Server plug-in for IBM UrbanCode Deploy provides deployment and undeployment operations for CICS environments. It enables the deployment of CICS applications in IBM UrbanCode Deploy and allows CICS applications to participate in the DevOps adoption along with distributed applications. By installing the plug-in, many additional steps are available to be dragged from the palette in the process editor. These steps include the following abilities:

- Install CSD resources, groups, and lists
- Install BAS resources, resource descriptions, and groups
- Discard resources
- Perform NEWCOPY and PHASEIN of programs
- Deploy and undeploy CICS bundles
- Scan pipelines

For more information about the CICS Transaction Server plug-in for IBM UrbanCode Deploy, see the UrbanCode plug-in website, found at:

`https://www.developer.ibm.com/urbancode/plugindoc/ibmucd/cics-ts-plug/0-1/`

4.2 DFHDPLOY utility

Applications and CICS bundles are a convenient way to package and manage components and resources (and their dependencies) in CICS. The DFHDPLOY utility provides a reliable and repeatable method of automating the deployment of CICS bundles, OSGi bundles and Liberty web application within CICS bundles, and cloud-enabled CICS applications. This utility helps organizations streamline their application development pipeline and fits well within the continuous delivery development and operations (DevOps) approach.

You can start the DFHDPLOY utility by using JCL, which provides a set of simple commands that Release Engineers can use in a script to deploy, undeploy, and set the state of CICS bundles and applications. Scripted deployments allow a CICS project in z/OS File System (zFS) to be programmatically deployed across CICS systems.

The following DFHDPLOY utility commands are provided:

- **SET CICPLEX**
- **DEPLOY BUNDLE**
- **UNDEPLOY BUNDLE**
- **SET BUNDLE**
- **DEPLOY APPLICATION**
- **UNDEPLOY APPLICATION**
- **SET APPLICATION**

You can read more about these utility commands and their functions in the CICS IBM Knowledge Center at the following website:

https://ibm.biz/BdHa8w

The DFHDPLOY utility is run after the Build Engineer completes the automated build process that is described in Chapter 3, “Build environment” on page 19. After the Build Engineer resolves all artifacts by using automation to run the CICS build toolkit and moves the output contents to zFS, then are you ready to deploy the CICS bundle or CICS cloud application to your target environment with DFHDPLOY.

The DFHDPLOY utility can be started by using JCL and provides a set of simple commands that are used in a script to deploy, undeploy, and set the state of CICS bundles and applications. Scripted deployments allow a CICS project in z/OS File System (zFS) to be programmatically deployed across CICS systems.

4.3 DFH$DPLY sample program

The DFH$DPLY sample program can be found in the CICS sample library `CICSTS53.CICS.SDFHSAMP`.

DFHDPLOY utility commands are read from the JCL SYSIN data definition and messages are written to the SYSTSPRT data definition. You can specify a command across multiple lines. A semicolon (;) indicates the end of a command. Comments starting with an asterisk (*) as the first non-blank character are ignored. Leading and trailing spaces are ignored.

When an application is changed from AVAILABLE or UNAVAILABLE to any other state, DFHDPLOY waits for tasks that are associated with the application operation to complete during the UNAVAILABLE stage. If a task fails to complete within the specified amount of time, the job fails and you see warnings and a return code of 4 or 8. For more information about the return codes, see the CICS IBM Knowledge Center for DFHDPLOY utility return codes at the following website:

https://ibm.biz/BdHaur

DFHDPLOY requires read access to the zFS directories that are involved in the job, and CPSM UPDATE access to the bundle and application resource classes that are involved. For more information about security authorization in CPSM, see the “Security for platforms and applications” topic in the CICS IBM Knowledge Center at the following website:

https://ibm.biz/BdHauK

Example 4-1 shows how to connect to MYPLEX, remove the existing bundle WEBSITE if it exists, and deploy a new bundle WEBSITE to an enabled state.

Example 4-1 Connect to MYPLEX, remove the existing bundle, and deploy a new one

```
//DFHDPLOY JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//DFHDPLOY  EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB   DD   DISP=SHR,DSN=CICSTS53.CICS.SDFHLOAD
//          DD   DISP=SHR,DSN=CICSTS53.CPSM.SEYUAUTH
//SYSTSPRT  DD   SYSOUT=*
//SYSIN     DD   *
SET CICSPLEX(MYPLEX);
*
UNDEPLOY BUNDLE(WEBSITE) CSDGROUP(BANKING) SCOPE(SYS1)
STATE(DISCARDED);
*
DEPLOY BUNDLE(WEBSITE) BUNDLEDIR(/var/cicsts/bundles/Website_1.0.0/)
CSDGROUP(BANKING) SCOPE(SYS1) STATE(ENABLED) TIMEOUT(60);
/*
```

Example 4-2 shows the DFHPLOY output when the job from Example 4-1 completes.

Example 4-2 DFHDPLOY output from Example 4-1

```
DFHRL2132I Analyzing CICS regions and CSD attributes.
DFHRL2093I BUNDLE(WEBSITE) found in SCOPE(SYS1).
DFHRL2129I BUNDLE(WEBSITE) state is INSTALLED on 2 CICS regions in SCOPE(SYS1).
DFHRL2129I BUNDLE(WEBSITE) state is ENABLED on 2 CICS regions in SCOPE(SYS1).
DFHRL2054I Setting BUNDLE state to DISABLED. DFHRL2042I Discarding
BUNDLE(WEBSITE).
DFHRL2077I BUNDLE(WEBSITE) has been discarded from SCOPE(SYS1).
DFHRL2037I UNDEPLOY command successful.

DFHRL2132I Analyzing CICS regions and CSD attributes.
DFHRL2051I Creating BUNDLE definition on the CSD in system(SYS1).
DFHRL2052I Installing BUNDLE definition.
DFHRL2131I Waiting for BUNDLE(WEBSITE) to be installed in active regions in
SCOPE(SYS1).
DFHRL2130I BUNDLE(WEBSITE)installed in 2 of 2 regions in SCOPE(SYS1).
DFHRL2054I Setting BUNDLE state to ENABLED.
DFHRL2012I DEPLOY command completed successfully.
```

4.4 Resolving CICS bundles and cloud applications with the CICS build toolkit

As shown in Chapter 2, “Development environment” on page 11, CICS Explorer V5.3 introduces the concept of variables in CICS bundles. Variables allow a CICS bundle from a single source to be deployed into a number of different environments, despite the bundle containing resources that have affinities to particular environments (file DSNAME high-level qualifiers, Java parts with JVM server names, programs with debugging techniques such as CEDF turned On or Off, and so on).

In addition to building CICS bundles and CICS cloud applications, the CICS build toolkit can resolve the CICS bundles and CICS cloud application by replacing these variables with specific values.

During development, after hardcoded values are replaced with variables, you then store the CICS bundles in a source code management (SCM) system, build them with the CICS build toolkit (see Chapter 3, “Build environment” on page 19), and place them in a build repository. Throughout this process, the CICS bundle remains non-specific to any single deployment environment. It is only when being deploying the bundle into a target environment that it is resolved by using the CICS build toolkit to make it appropriate to that environment. This whole pipeline is shown in Figure 4-18.

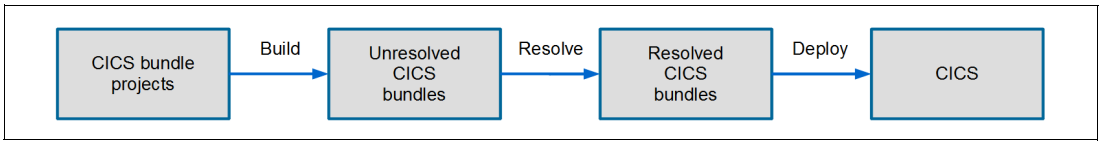


Figure 4-18 Pipeline for CICS bundles (also applies to CICS cloud applications)

CICS bundles can be resolved by using the properties with which they were developed. However, those properties are unlikely to be appropriate for every target environment, so the bundles can also be resolved by using properties files that are customized for the environment. Distinct properties files for different environments such as development, test, and production are typically expected.

At a higher level, CICS cloud applications are themselves composed of CICS bundles, and CICS cloud applications can also be resolved by using the CICS build toolkit. Resolving a CICS cloud application resolves all of the CICS bundles that it contains. CICS cloud applications are resolved with a particular application binding that contains the properties appropriate for that application.

As with the CICS build toolkit’s build command, the resolve command can run on the z/OS, Linux, or Windows operating systems.

4.4.1 Preparing to resolve CICS bundles

To resolve CICS bundles, first make sure that the CICS build toolkit is correctly installed (see to “Installing the CICS build toolkit” on page 36).

Start by retrieving the input to be resolved and placing it in a suitable place in the file system. The CICS build toolkit takes input files and folders, transforms them, and outputs them to a different place. The input files and folders are expected to be in the same structure as the output folder of the CICS build toolkit’s **`build`** command. When only bundles and applications are built, the input folder takes the format that is shown in Figure 4-19 on page 61, which shows applications and bundles stored under separate, top-level directories.

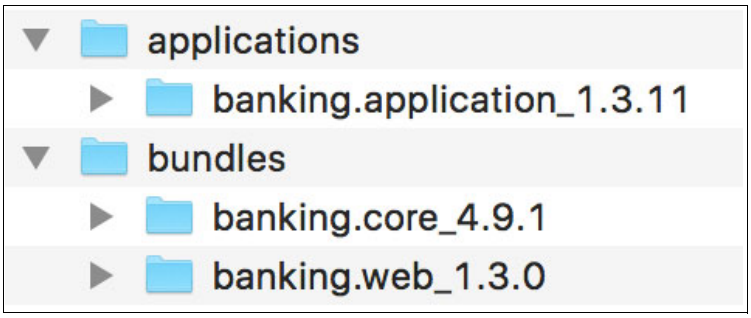


Figure 4-19 Input format when resolving only applications and bundles

When application bindings are built, the output folder has a deeper format, as shown in Figure 4-20. The name of the platform that the application binding binds to is used as the top-level directory name. Within that, the applications, bundles, and application bindings all have their own directories. This mirrors the structure of the platform home directory on zFS.

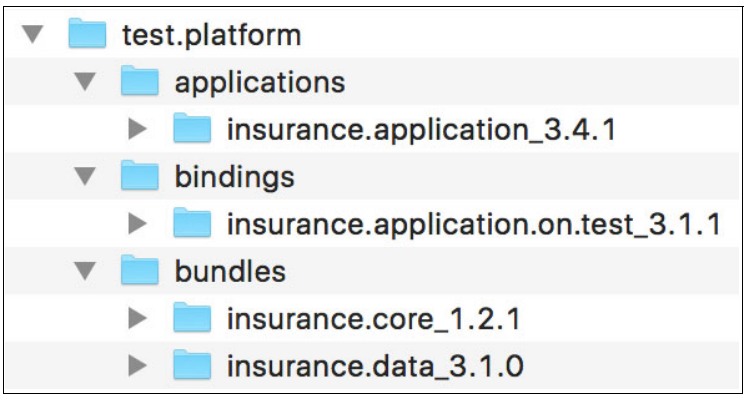


Figure 4-20 Input format when resolving application bindings, applications, and bundles

This directory structure can be re-created after pulling resources out of a build repository, or the structure itself can be stored inside the build repository.

The output of the resolve command is in the same format as the input, which makes it ready to be copied into a platform home directory or some other directory.

4.4.2 Resolving CICS bundles

With the input file structure and the output directory created, the CICS build toolkit is ready to resolve variables. In the example that is shown in Example 4-3, the input file structure is entirely contained within the folder `/home/user/unresolved`. The output is placed in the folder `/home/user/resolved`.

Run the CICS build toolkit's resolve command by using the **`--resolve`** option, as shown in Example 4-3. By default, this action takes the properties files from inside the CICS bundles and resolves with the properties that they contain, which are likely those that are used in the development environment.

Example 4-3 Resolve CICS bundles by using the CICS build toolkit and default properties

```
cicsbt --resolve /home/user/unresolved --output /home/user/resolved --verbose
Successfully resolved /home/user/unresolved to /home/user/resolved
```

When resolving to other environments, stand-alone properties files can be used. These files take precedence over properties in the CICS bundle to make the CICS bundle relevant to the target environment, as shown in Example 4-4.

Example 4-4 Resolve CICS bundles by using the toolkit and stand-alone properties file

```
cicsbt --resolve /home/user/unresolved --output /home/user/resolved --properties
/var/cicsts/envs/production.properties --verbose
```

Note: The CICS build toolkit command differs between operating systems:

- On Linux, use **cicsbt**.
- On Windows, use **cicsbt.bat**.
- On z/OS, use **cicsbt_zos**.

4.4.3 Resolving CICS cloud applications

When resolving CICS cloud applications, the CICS bundles within the applications have their variables replaced with properties from the file inside each CICS bundle. However, if properties are also contained inside the application binding, those properties take precedence over the ones in the CICS bundles, which allows property changes to affect consistently a whole application. Resolving a CICS cloud application is shown in Example 4-5.

Example 4-5 Command to resolve CICS cloud applications by using the CICS build toolkit

```
cicsbt --resolve /home/user/unresolved --output /home/user/resolved --verbose
```

4.4.4 Resolving bundles and apps with the toolkit in an automated process

Resolving CICS bundles and CICS cloud applications typically takes place automatically as part of an automated deployment process.

When using DFHDPLOY to deploy CICS bundles and CICS cloud applications, the automated process includes these steps:

1. Extract the unresolved input from a build repository and place it in a suitable place on the file system.
2. Extract the stand-alone properties file for the specific environment (where appropriate) from the SCM system.
3. Run the CICS build toolkit resolve command by using the BPXBATCH utility program.
4. Move the resolved artifacts to the wanted location.
5. Run DFHDPLOY to deploy the CICS bundle or CICS cloud application to the target environment.

A job to run these steps is shown in Example 4-6.

Example 4-6 JCL to resolve CICS bundles by using the toolkit and deploy them by using DFHDPLOY

```
//RSLVDPLY JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//     SET CPSMHLQ=CICSDPP.BETA14.CTS53BT.CPSM
//     SET CICSHLQ=CICSDPP.BETA14.CTS53BT.CICS
//     SET CICSPLEX=CICSPLEX
```

```
//     SET CIBUNDNM=GENACWSB
//     SET CISCOPE=CICSDA01
//*
//     SET JAVAHOME='/java/J8.0_64/'
//*       Bundle Work Dir
//     SET BUNWKDIR='/var/cicsts/workdir'
//*       Bundle Build Dir
//     SET BUNINDIR='/var/cicsts/devops/builds/bundles'
//*       Target Dir
//     SET RESLVDIR='/var/cicsts/devops'
//*       Bundle Name
//     SET BUNDNAME='bundle.name_1.1.0'
//*       CICS build toolkit full path and command
//     SET CICSBTCM='/usr/lpp/cicsts/cicsbt/cicsbt_zos'
//*       Environment we deploy to
//     SET ENVMT='production'
//*
//* First step is to copy only the bundle you want to resolve
//*
//COPY  EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
//STEPLIB DD  DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV  DD  *
JAVA_HOME=&JAVAHOME
/*
//STDPARM DD  *
SH
rm -r &BUNWKDIR/bundles/*;
cp –r &BUNINDIR/&BUNDNAME &BUNWKDIR/bundles
//*
//* Second step is to resolve the bundle using cicsbt_zos
//*
//RESOLVE EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
//STEPLIB DD  DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV  DD  *
JAVA_HOME=&JAVAHOME
/*
//STDPARM DD  *
PGM
&CICSBTCM
  --resolve &BUNWKDIR --output &RESLVDIR/
  --properties /var/cicsts/envs/&ENVMT.properties
//*
//* Third step is DEPLOY and set available the new bundle
//*
//DFHDPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB  DD  DISP=SHR,DSN=&CPSMHLQ..SEYUAUTH
//     DD  DISP=SHR,DSN=&CICSHLQ..SDFHLOAD
//*
//SYSTSPRT DD  SYSOUT=*
//*
```

```
//SYSIN DD  *
*
  SET CICSPLEX(&CICSPLEX);
*
*  Define a CICS bundle resource and transition it towards an
*  ENABLED state within the current CICSplex.
*
 UNDEPLOY BUNDLE(&CIBUNDNM)
     SCOPE(&CISCOPE)
     STATE(DISCARDED);
*
 DEPLOY BUNDLE(&CIBUNDNM)
    SCOPE(&CISCOPE)
 BUNDLEDIR(&RESLVDIR/bundles/&BUNDNAME/)
    STATE(ENABLED);
/*
```

When using IBM UrbanCode Deploy, the process of resolving CICS bundles and CICS cloud applications by using the CICS build toolkit uses the IBM UrbanCode Deploy built-in Shell multi-purpose step. The automated process includes the following steps:

1. Download the component version from IBM UrbanCode Deploy CodeStation to a place in your working directory.
2. Run the CICS build toolkit resolve command by using the Shell multi-purpose step. Resolve the CICS bundle or CICS cloud application to another location in the working directory.
3. Copy the resolved artifacts to the wanted location.
4. Deploy the artifact by using the Deploy bundle or Deploy application steps.

Figure 4-21 shows this whole process in the IBM UrbanCode Deploy process editor. To increase the speed of execution, many steps are run in parallel.

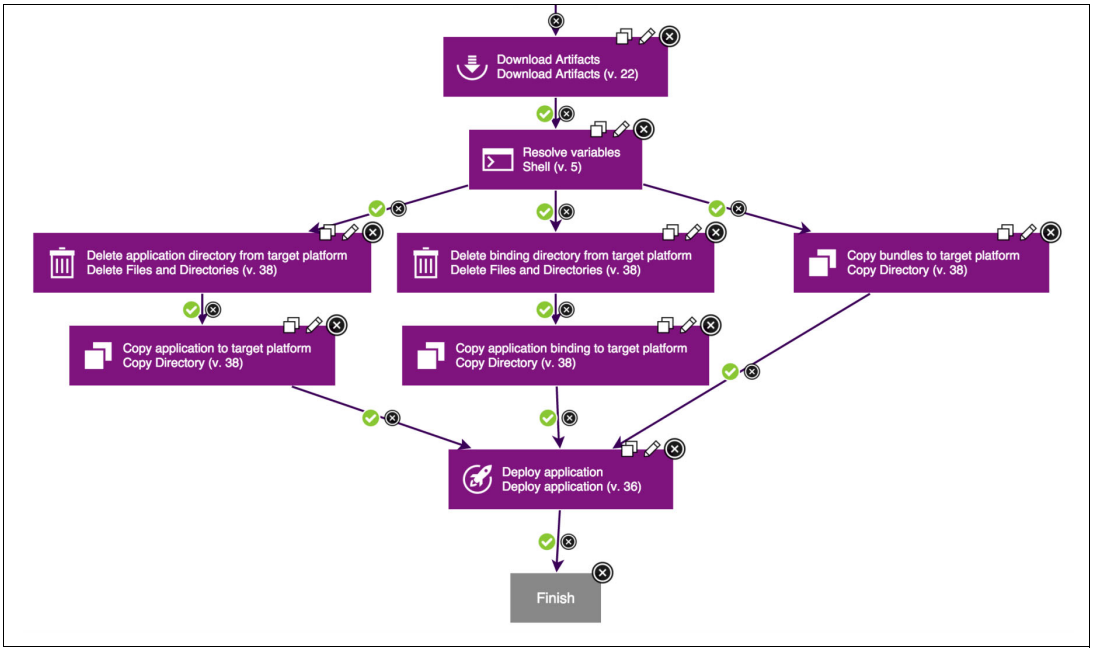


Figure 4-21 Deployment process for a CICS cloud application, - resolve it with the CICS build toolkit

Multiple deployments can occur concurrently, so be sure to set the workspace that the CICS build toolkit uses while running to something that is unique, such as the process's working directory. The settings for calling the Resolve variables Shell step that are shown in Figure 4-21 on page 64 are provided in Figure 4-22, with the 'Shell Script' property set to the CICS build toolkit command.

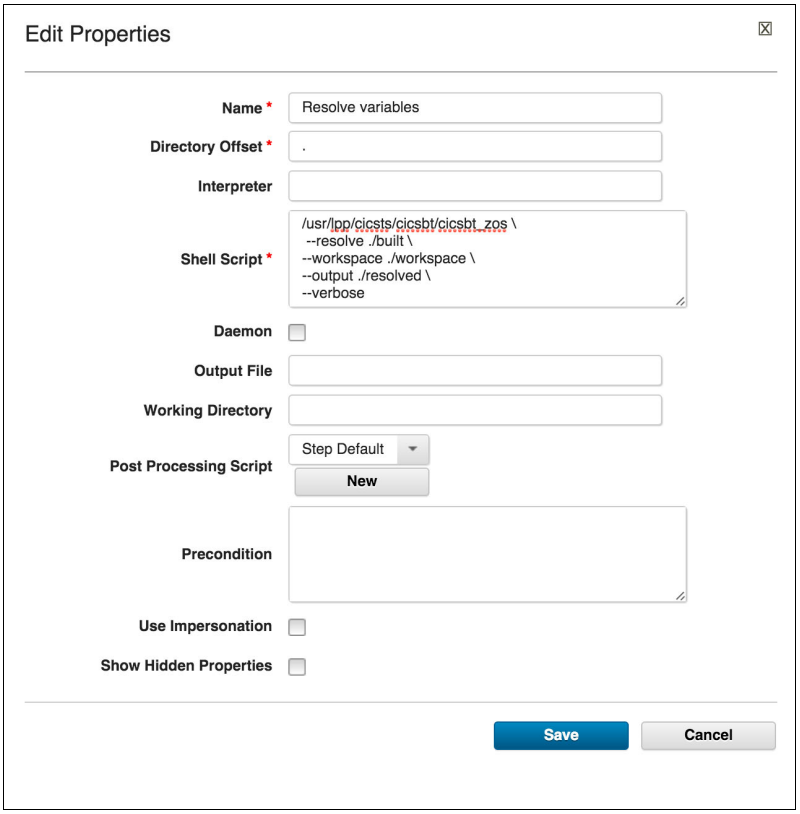


Figure 4-22 Properties of the Shell step that is used to run the CICS build toolkit's resolve command

5

Applying DevOps to COBOL applications and CICS policies

This chapter looks at implementing a development and operations (DevOps) process for CICS COBOL applications and a CICS policy. It reviews how CICS applications and CICS policies are built in Rational Team Concert and published to an IBM UrbanCode Deploy repository. It then reviews the process of orchestrating and automating deployment of the applications to the target platform.

In this example, we use the GENAPP sample application and focus on a traditional CICS/COBOL DB2 application implementation and a CICS policy.

This chapter covers the following topics:

5.1 Current challenges

Many mainframe customers have CICS based core applications that require new functions and policy changes from time to time. Competitive business factors also push companies to adapt their applications at a faster rate. But, the application delivery pipeline is often limited by factors, such as the following ones:

- Disparate development and production environments.
- Complex, manual release processes that lack repeatability and speed.
- Costly and inefficient release processes can consume days of work effort.

Fortunately, there are tools that can help mitigate some of these challenges. Rational Team Concert, various CICS Tools and toolkits, and IBM UrbanCode Deploy provide the capabilities to support continuous delivery of enterprise applications. These tools help accelerate delivery by reducing cycle time to develop and test applications across heterogeneous environments. These tools help you reduce costs and minimize delays around delivering mainframe applications.

Here are some examples of how these tools address specific needs of continuous delivery and DevOps in general:

- Rational Team Concert provides tools for z/OS builds and CICS Application and Bundle projects to support continuous builds.
- The IBM CICS Transaction Server build toolkit (CICS build toolkit) can be started from Ant Scripts to deploy CICS applications and Bundles. IBM UrbanCode Deploy has a component that is called Buztool to deploy z/OS artifacts. With these tools, you can deploy CICS projects to build repositories.
- z/OS Plug-ins for IBM UrbanCode Deploy can be used to FTP, copy, and deploy Partitioned data sets.
- You can use the CICS Plug-in for IBM UrbanCode Deploy use NEWCOPY to copy programs and deploy CICS applications and bundles.

The following sections describe specific steps to implementing and deploying the different components of the GENAPP application.

5.2 Implementing the GENAPP base component

This section introduces the build and deployment of the GENAPP base COBOL application by using the DevOps approach.

5.2.1 Building the CICS application

In this scenario, the CICS/COBOL source code is pre-compiled bind and link-edited. The load modules and DBRM are made available in staging data sets that are named CICSDPP.SCEN1.LOAD and CICSDPP.SCEN1.DBRMLIB. The ship list file is created.

The ship list file specifies which files from the build must be included in the new component version to deploy. You must create a ship list file before you run the deployment tools.

Ship list files are XML files that contain a list of file specifiers. The container type that is used to identify partitioned data set (PDS) resources is PDS, and the resource type is PDSmember.

Figure 5-1 shows the output of the build process for COBOL applications.

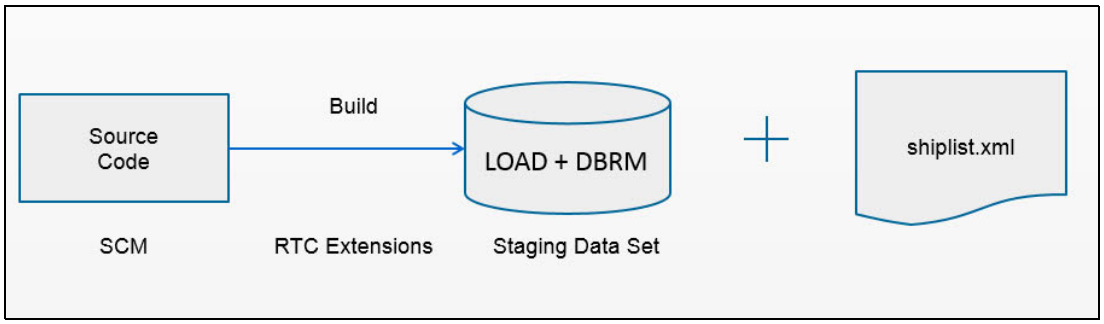


Figure 5-1 COBOL build

Example 5-1 shows the ship list file that is already created for the example component.

Example 5-1 shiplist.xml

```
<?xml version="1.0" encoding="CP037"?>
<manifest type="MANIFEST_SHIPLIST">
    <container name="CICSDPP.SCEN1.LOAD" type="PDS" deployType="CICS_LOAD">
      <resource name="*" type="PDSMember"/>
    </container>
    <container name="CICSDPP.SCEN1.DBRMLIB" type="PDS" deployType="DBRM">
      <resource name="*" type="PDSMember"/>
    </container>
</manifest>
```

5.2.2 Defining a component in IBM UrbanCode Deploy

IBM UrbanCode Deploy z/OS Deploy Toolkit provides a command-line utility that is called Buztool that is used to deploy z/OS artifacts.

Buztool creates the z/OS component version in IBM UrbanCode Deploy CodeStation. CodeStation is not a separate product. It is simply a name that is given to the versioned storage area that is provided in IBM UrbanCode Deploy.

Before you can build a version of a component by using Buztool, first you must define the component by completing the following steps:

1. Log in to IBM UrbanCode Deploy and click **Components** → **Create Component**. In this example, we name our component GENAPP Base for the COBOL components. Specify the rest of the parameters, as shown in Figure 5-2.

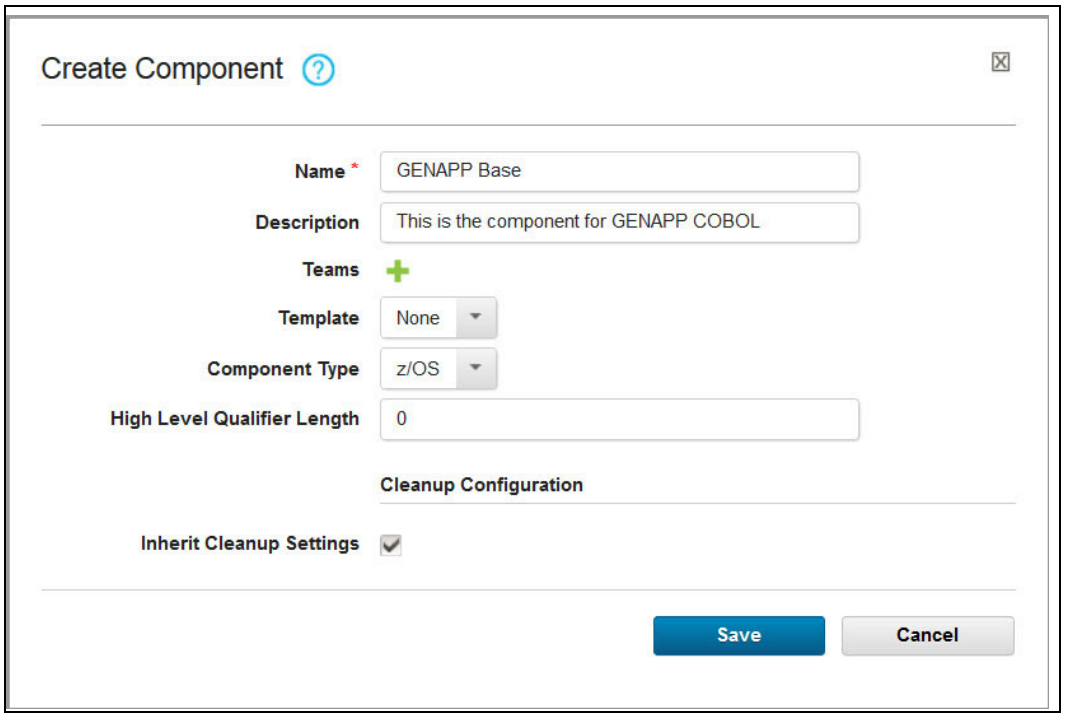


Figure 5-2 Create the GENAPP component

2. Click **Save**, which creates the component that is called GENAPP Base. The window that opens is the component dashboard window, which shows that the component was created, as shown in Figure 5-3.

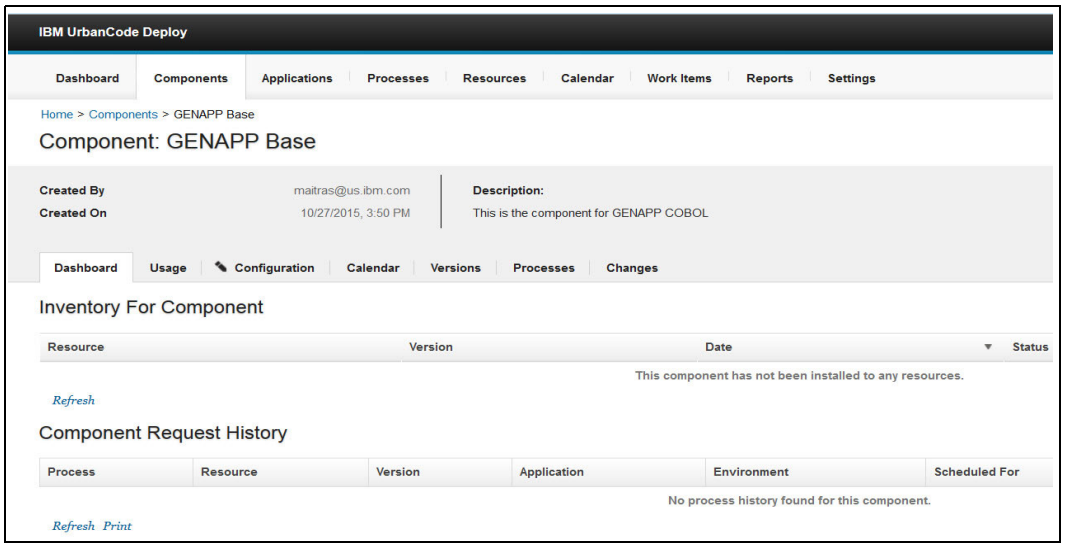


Figure 5-3 Component dashboard showing that the component was created

5.2.3 Running Buztool

You must run Buztool to generate a new version of the component and put the artifacts into UCD CodeStation.

Figure 5-4 illustrates how the component version is created by Buztool.

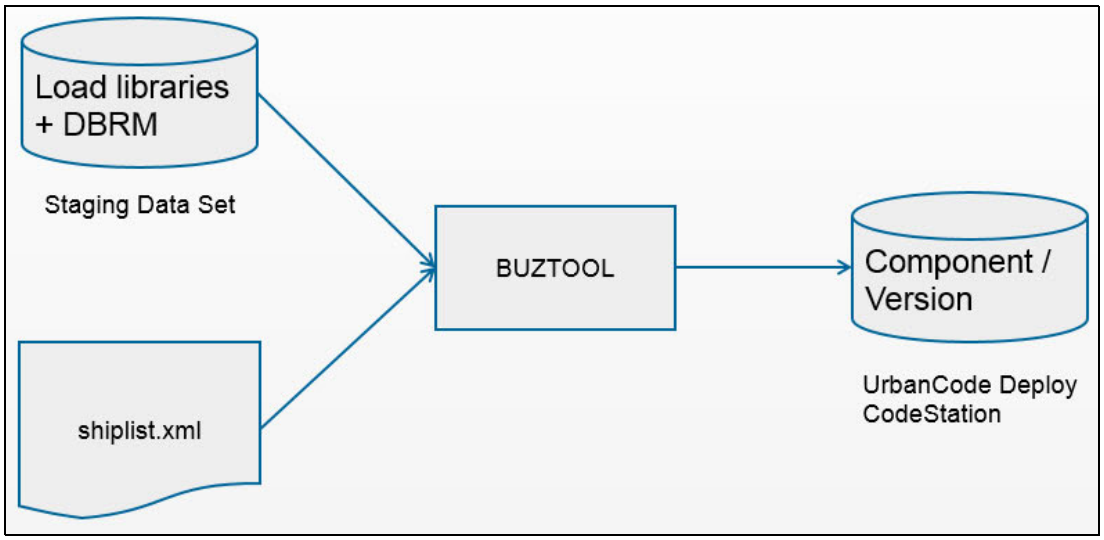


Figure 5-4 Buztool deploys the component version

You can run Buztool by using JCL in a batch job. A sample JCL is provided as part of the SMP/E installation and is found in a data set that is named `<HLQ>.UCDTLKT.SBUZSAMP`, where `<HLQ>` is the high-level qualifier that is used during the installation. In this example, the sample data set is `REDS01.UCD.UCDTLKT.SBUZSAMP`.

You can access the data set from the CICS Explorer z/OS perspective. Use Figure 5-5 (taken from CICS Explorer) as a reference, and filter for `REDS01.UCD` as high-level qualifier to find the sample data set.

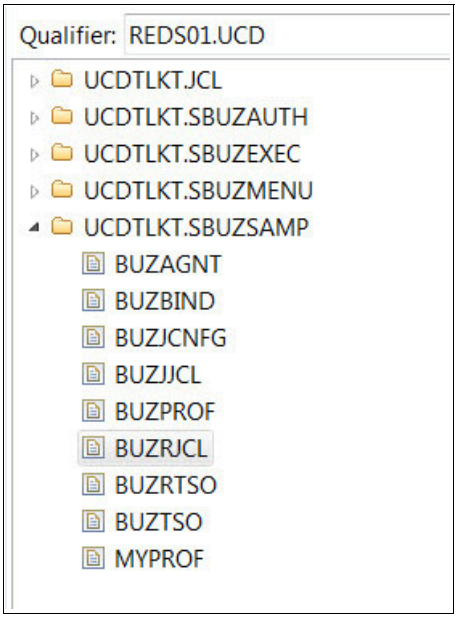


Figure 5-5 Location of sample Buztool JCL in the CICS Explorer z/OS perspective

The sample JCL `BUZRJCL` was copied to the local data set, was renamed `BUZTOOL`, and customized to our environment, as shown in Figure 5-6.

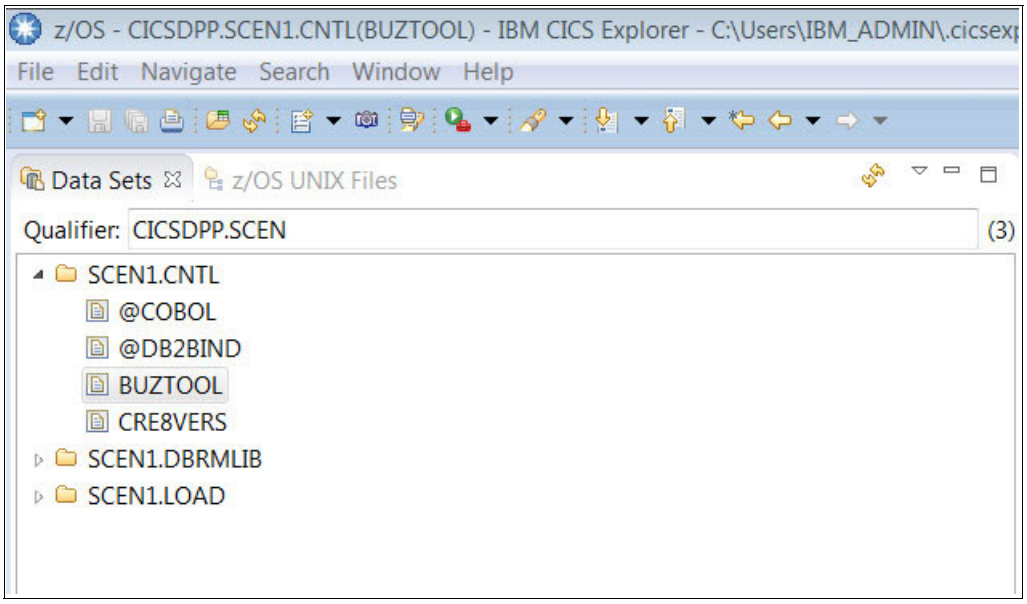


Figure 5-6 Buztool JCL

Example 5-2 shows a sample of the JCL. For this example, we customized the JCL according to our environment. We were careful about using the following options:

-c Component name
-v Sets the version
-s Path of the `shiplist.xml` file

Example 5-2 Sample JCL

```
//ST1      EXEC PGM=IKJEFT01,DYNAMNBR=30,PARM='%ISPFCL',REGION=0M
//SYSEXEC  DD DSN=REDS01.UCDTLKT.SBUZEXEC,DISP=SHR
//         DD DSN=SYS1.SISPCLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//PROFILE  DD DSN=REDS01.UCDTLKT.SBUZSAMP(MYPROF),DISP=SHR
//ISPPROF  DD RECFM=FB,LRECL=80,SPACE=(TRK,(2,2,2))
//ISPLLIB  DD DSN=SYS1.SISPLOAD,DISP=SHR
//ISPMLIB  DD DSN=SYS1.SISPMENU,DISP=SHR
//ISPTLIB  DD DSN=SYS1.SISPTENU,DISP=SHR
//ISPPLIB  DD DSN=SYS1.SISPPENU,DISP=SHR
//ISPSLIB  DD DSN=SYS1.SISPSLIB,DISP=SHR
//SYSTSIN  DD *
 ISPSTART CMD(BUZTOOL "createzosversion"                    -
    "-c" "GENAPP Base"                                  -
    "-v" "version2"                                             -
    "-s" "/var/cicsts/devops/scenario1/build/shiplist.xml")
/*
```

After you customize the JCL, right-click and submit the z/OS job, as shown in Figure 5-7 on page 73.

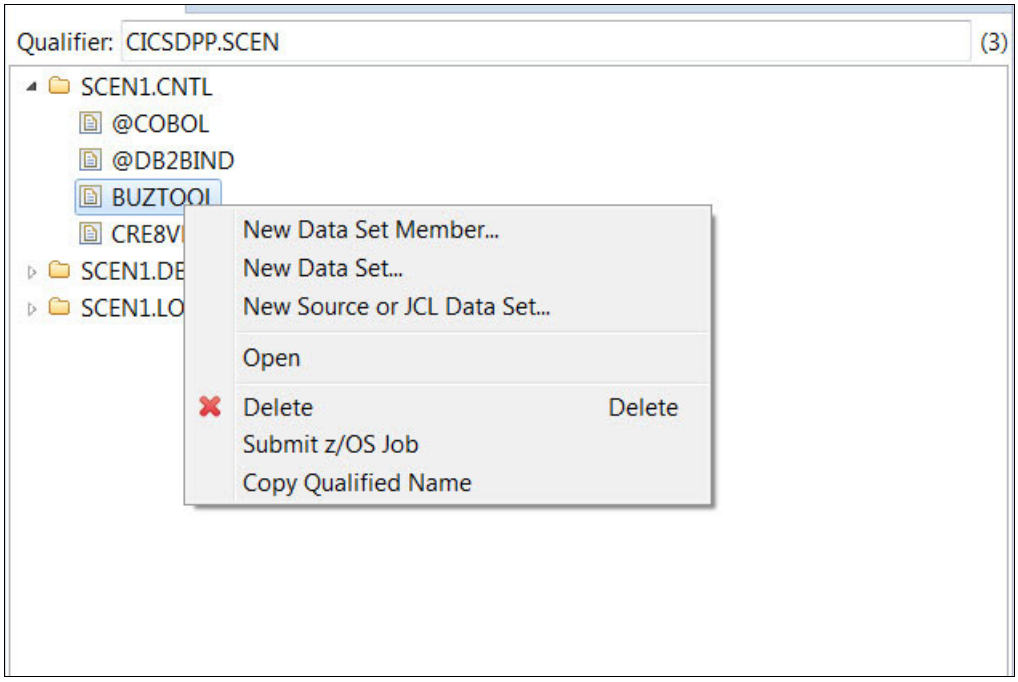


Figure 5-7 Run the Buztool JCL

After the job finishes, you can see the results, as shown in Figure 5-8.

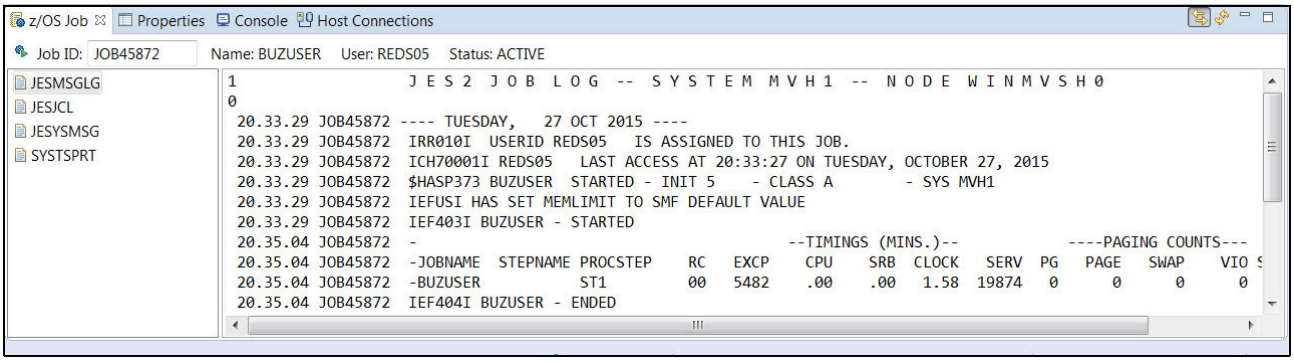


Figure 5-8 Result of the Buztool JCL job

You should have a component version that is named version2 of the GENAPP Base component in IBM UrbanCode Deploy CodeStation, as shown in Figure 5-9.

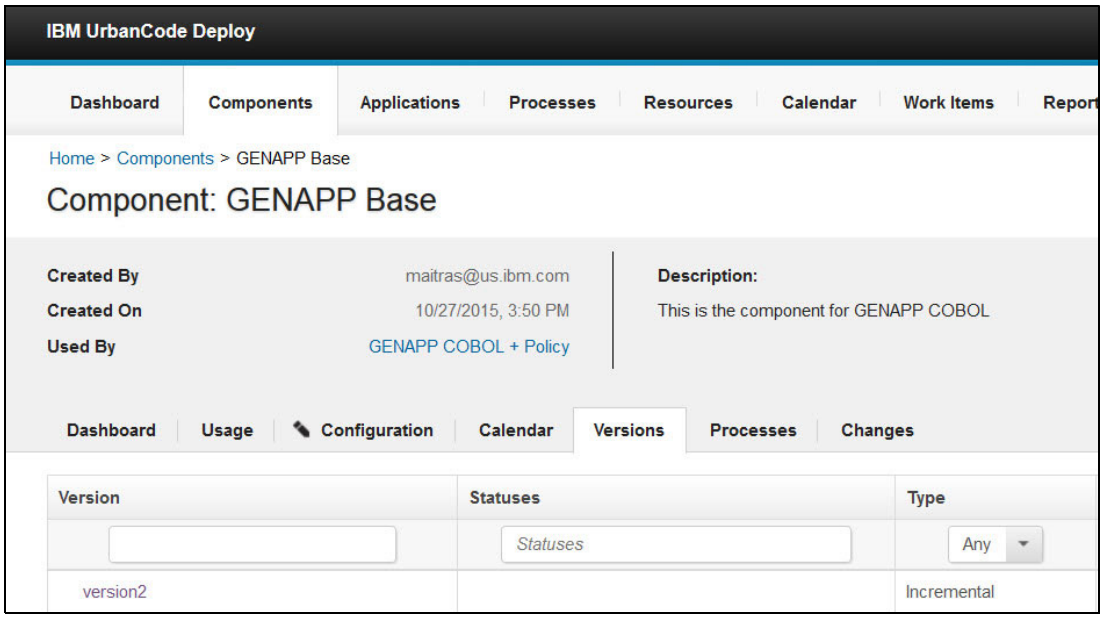


Figure 5-9 Component version2 created

To look at the version of the component that is created, click the **Versions** tab in IBM UrbanCode Deploy and refresh. You should see version2 there. Click **version2**, and you see the list of artifacts that was specified in the `shiplist.xml` file, as shown in Figure 5-10.

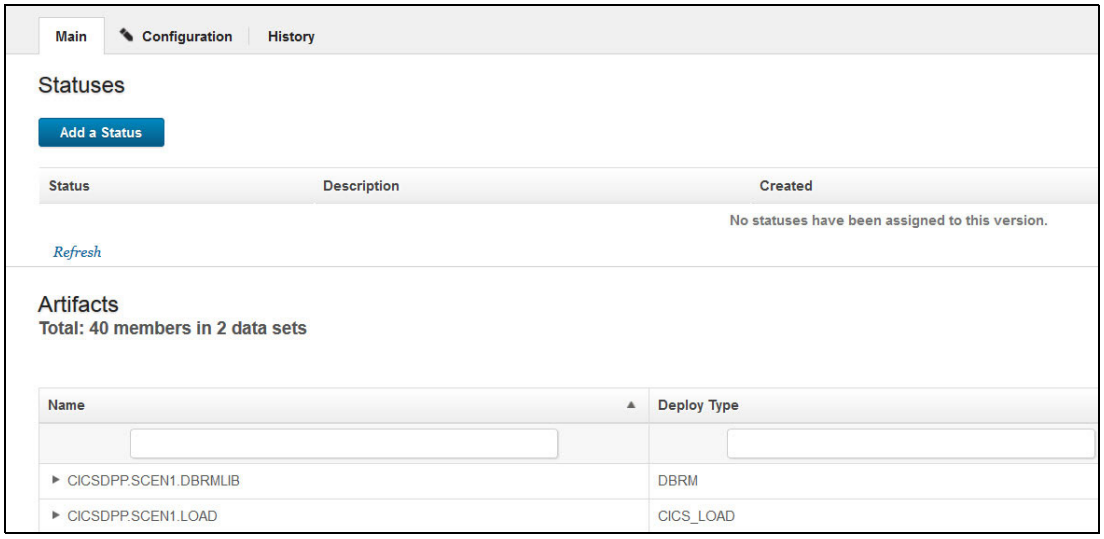


Figure 5-10 Component version artifacts

5.2.4 Creating and defining a component process

In this section, you create a component process to deploy the GENAPP Base component. Figure 5-11 shows the complete deployment process that you create. Each box in the figure represents a different step.

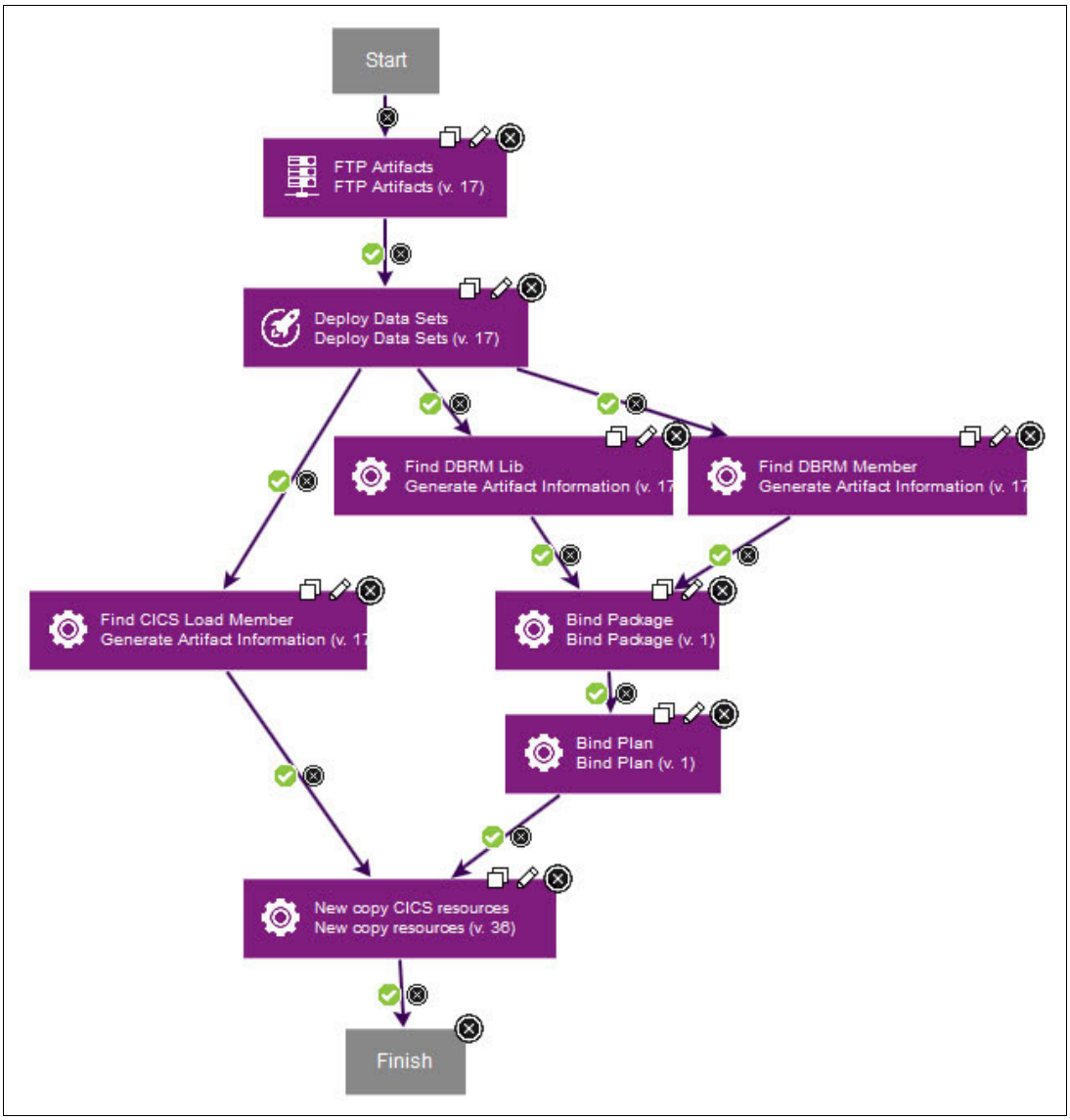


Figure 5-11 Component process for GENAPP base

1. The FTP Artifacts step retrieves the artifacts (CICS Load modules and DRMB) from the IBM UrbanCode Deploy repository and puts them into a working directory.
2. The Deploy Data Sets steps copy these artifacts from the working directory into the data sets. The process branches off to two parallel subprocesses: a CICS Load module step and DBRM0-related steps.
3. The Find CICS Load Member step generates the list of load module member names to prepare for the New Copy step.
4. The Find DBRM Lib and Find DBRM Member steps are the preparation steps to generate a DBRM library data set name and a DBRM member list.

5. The Bind Package and Bind Plan steps perform the binding with DB2.
6. Only when both branches run successfully does the process proceed to the New Copy step, which installs the artifacts.

Here are the details of defining this process and the individual steps:

1. Go to the Processes tab under component GENAPP Base and click **Create Process**, as shown in Figure 5-12.

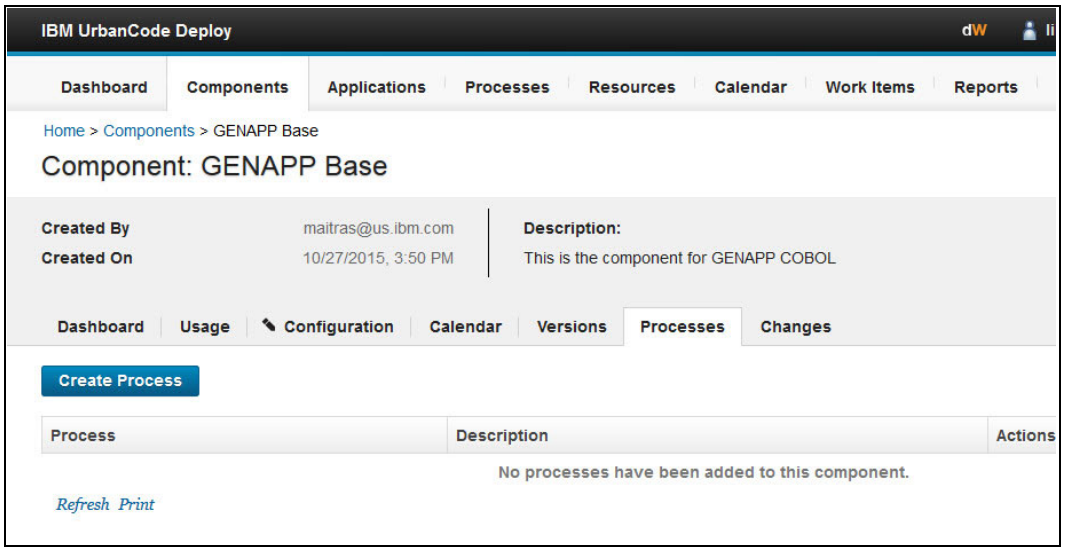


Figure 5-12 Create process for the GENAPP Base component

2. Define the Deploy process, as shown in Figure 5-13. Name it Deploy, add a description, and configure the rest of the fields. Click **Save**.

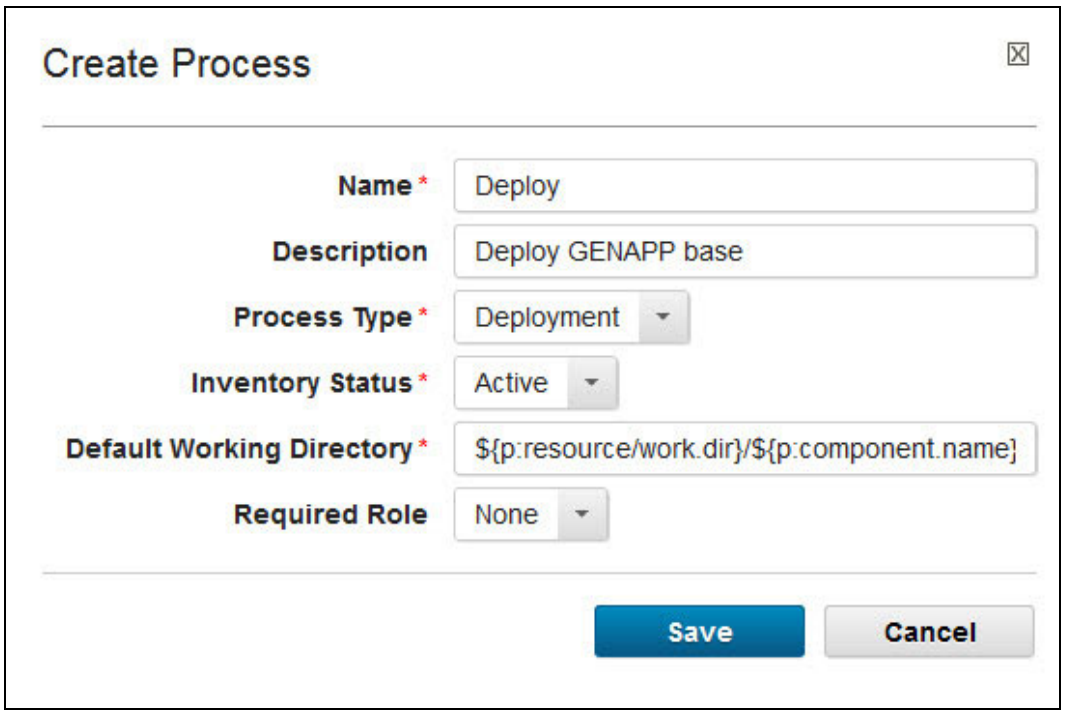


Figure 5-13 Create the Deploy process

3. You should now see a process called Deploy created. Click the process or **Edit**. The design window opens, where you define the steps. Figure 5-14 shows the layout of the Design tab in process designer window. On the left side, below the Design tab label, there are six buttons, which are zoom in, zoom out, actual size, print preview, save, and discard changes.

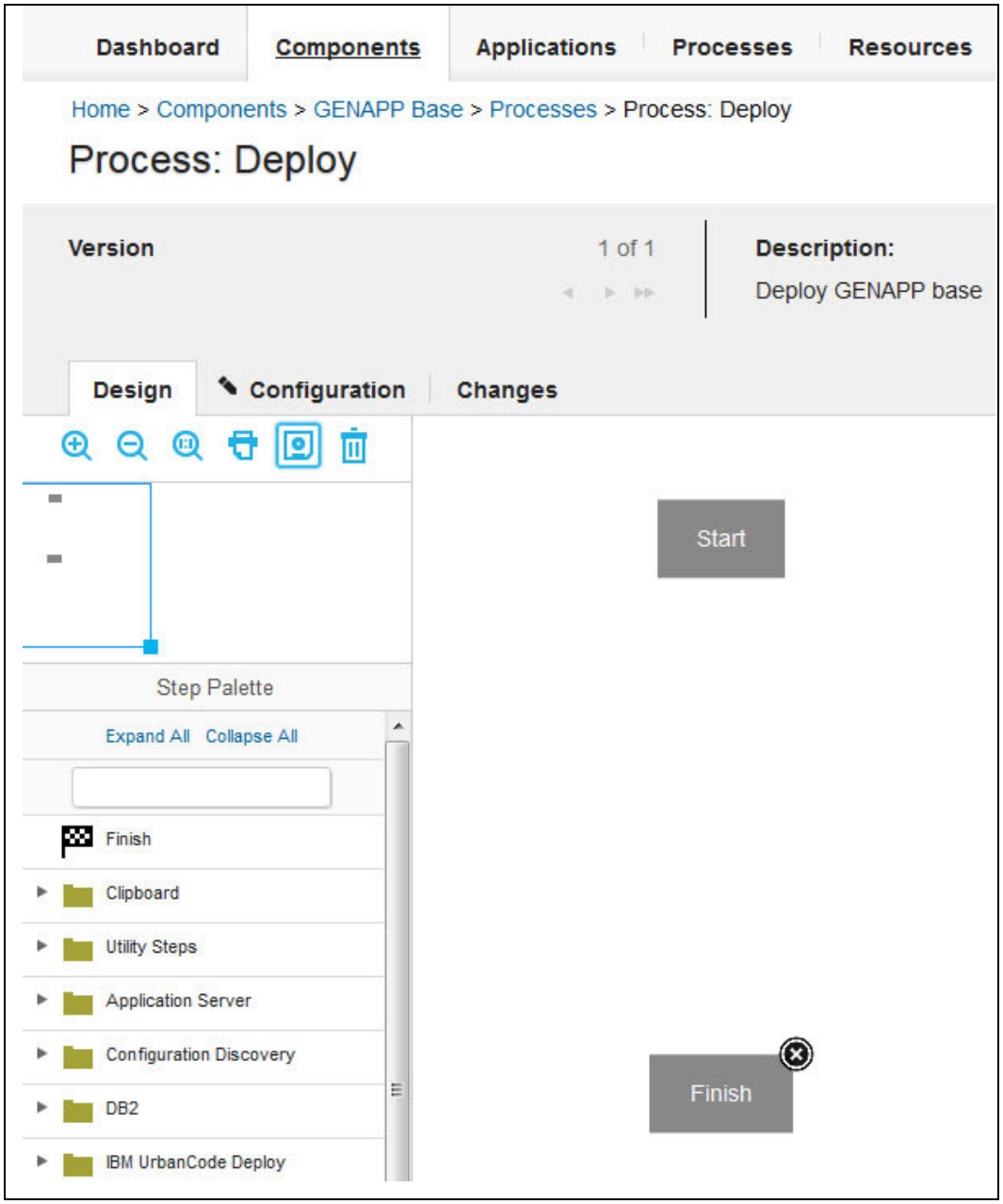


Figure 5-14 Process designer window

It is a preferred practice to save changes as you create the process and before you move on to other tasks, especially if you are accessing an IBM UrbanCode Deploy Server through a remote connection or VPN.

Moving down from the buttons and the zoom window, there is the Step Palette with lists of available steps that are grouped in categories. Additional steps can be added by installing additional plug-ins. There is a search box that you can use search by using a key word to find specific steps quickly. On the right side is the process design canvas where you create your process flow.

FTP Artifacts

To retrieve the artifacts (CICS Load modules and DRMB) from the IBM UrbanCode Deploy repository and put them into a working directory, complete the following steps:

1. Enter FTP into the search box to find the FTP Artifact step (Figure 5-15).

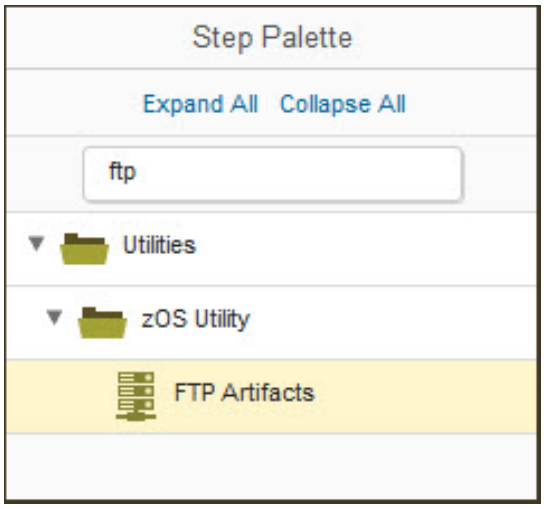


Figure 5-15 FInd the FTP Artifact step by using the search box

2. Drag the FTP Artifacts step and drop it into the design canvas on the right. An Edit Properties window opens, where you specify the parameters (Figure 5-16 on page 79). Give it a name and leave most of the other values as the defaults.

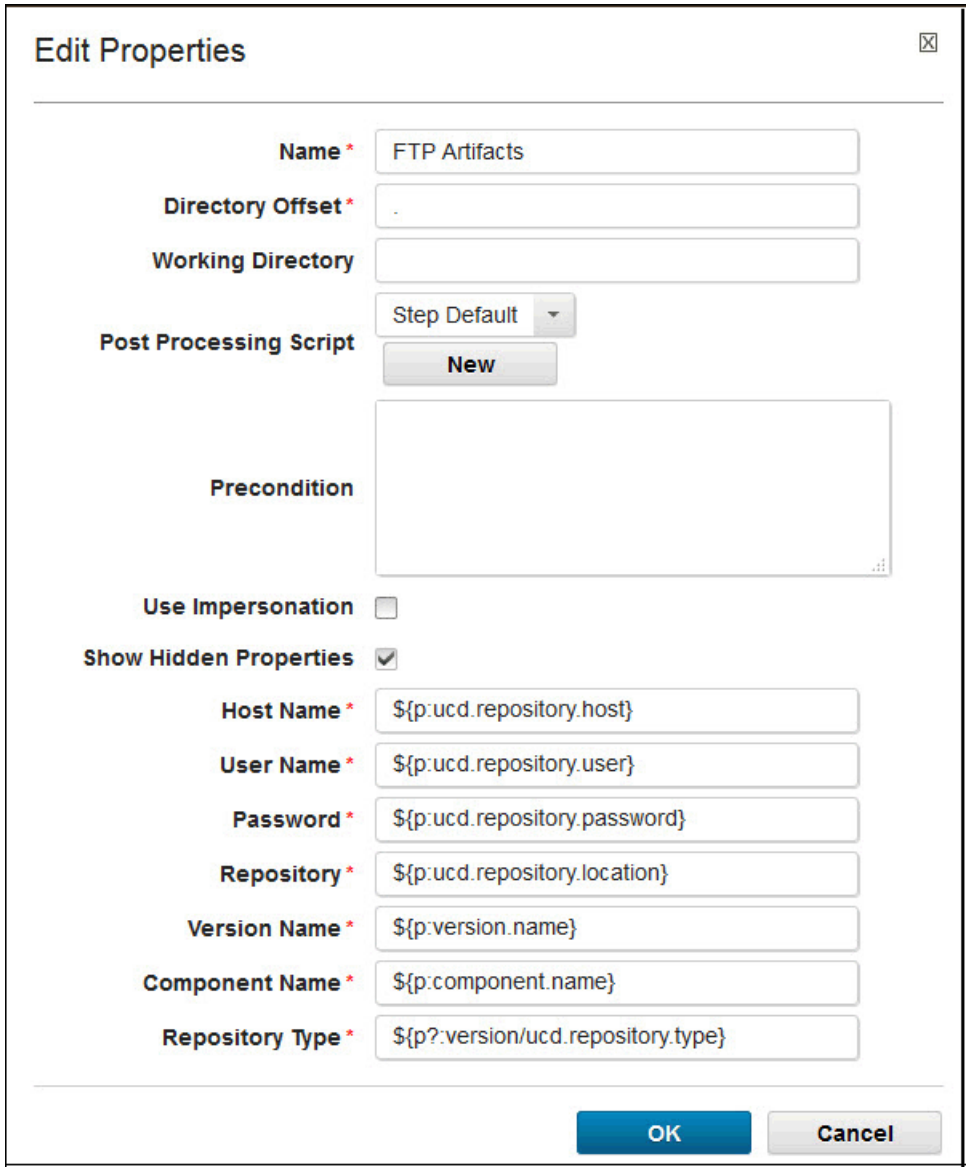


Figure 5-16 Specify properties for the FTP Artifacts step

3. Pay attention to the following items:
   - Directory Offset field: Make sure that there is a dot, which indicates that the current working directory is used.
   - Show Hidden Properties: Select this box to see the list of hidden properties.
   - All the fields that are marked with an '*' are mandatory and cannot be left blank.
   - Many of the properties apply in a larger context beyond this specific component process. Therefore, keep the variable format (`$p{…}`) rather than hardcoding them to make the process more flexible and reusable without the need for modification. It is important to make sure all the variable properties here are defined in components, applications, agents, or other contexts.

– `${p:variable name}` is the syntax for pointing to the value of the variable. The value of the variable that is specified must be successfully resolved during the running of the process or the process fails.

– `${p?:optional variable name}` indicates that this variable is optional. If it is not set to a value, then an empty string is used and the process can still run.

Each of the fields has an explanation, which is available to the right of the field when you hover your cursor over the field (marked by a circle with a question mark).

4. Click **OK** and you see the first step that you created on the canvas. Hover your cursor over the Start box until a blue circle with an arrow appears, as shown in Figure 5-17.

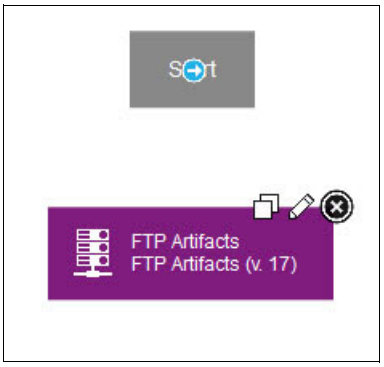


Figure 5-17 Hover your cursor over Start until you see a circle with arrow

5. Click the circle and drag it to the FTP Artifacts step to connect the two boxes. You see a connecting arrow after you complete the connection, as shown in Figure 5-18.

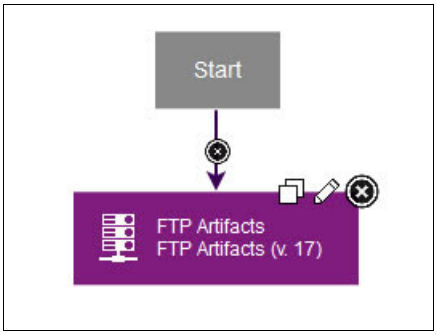


Figure 5-18 Connect the Start and FTP Artifacts steps

Create the other steps of the deployment process in the same manner. The next section shows the properties that we use in this example.

Deploying data sets

This step copies the artifacts to the CICS load library and DBRM library. The PDS Mapping that is shown below means copy anything originally in a library ending with `.LOAD` to `CICSDPP.CB12V51.LOAD`, which is the CICS load library that is used in our example. Similarly, the second line means copy anything originally from a library that ends with `.DBRMLIB` to `CICSDPP.CB12V51.DBRMLIB`, which is the DBRM library that is used in our example.

```
*.LOAD,CICSDPP.CB12V51.LOAD
*.DBRMLIB,CICSDPP.CB12V51.DBRMLIB
```

For more information, see Figure 5-19 on page 81.

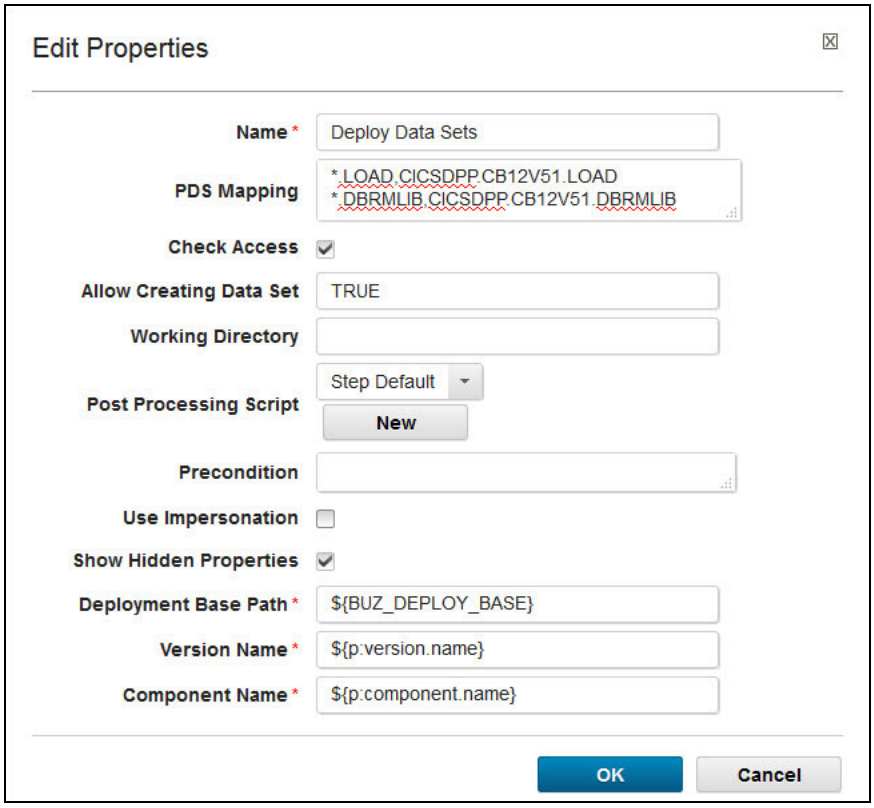


Figure 5-19 Deploy data sets step properties

Finding CICS load members

This step generates the list of load module member names to prepare for the New Copy step:

1. Search for “generate art”, and then click **Generate Artifact Information** under the z/OS utility, as shown in Figure 5-20.

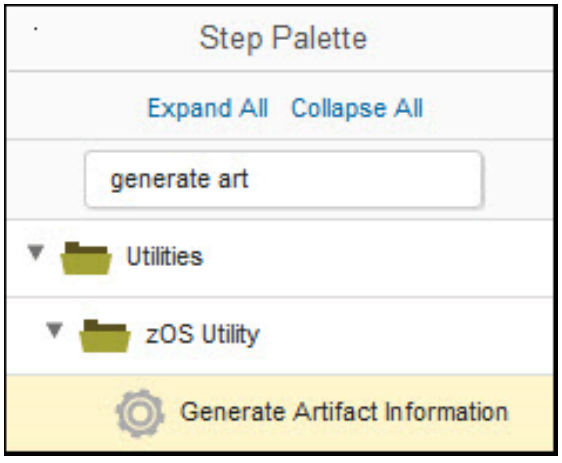


Figure 5-20 FInd the Generate Artifact Information step from the Step Palette

2. Specify the properties. Figure 5-21 shows the values that we use in our scenario.

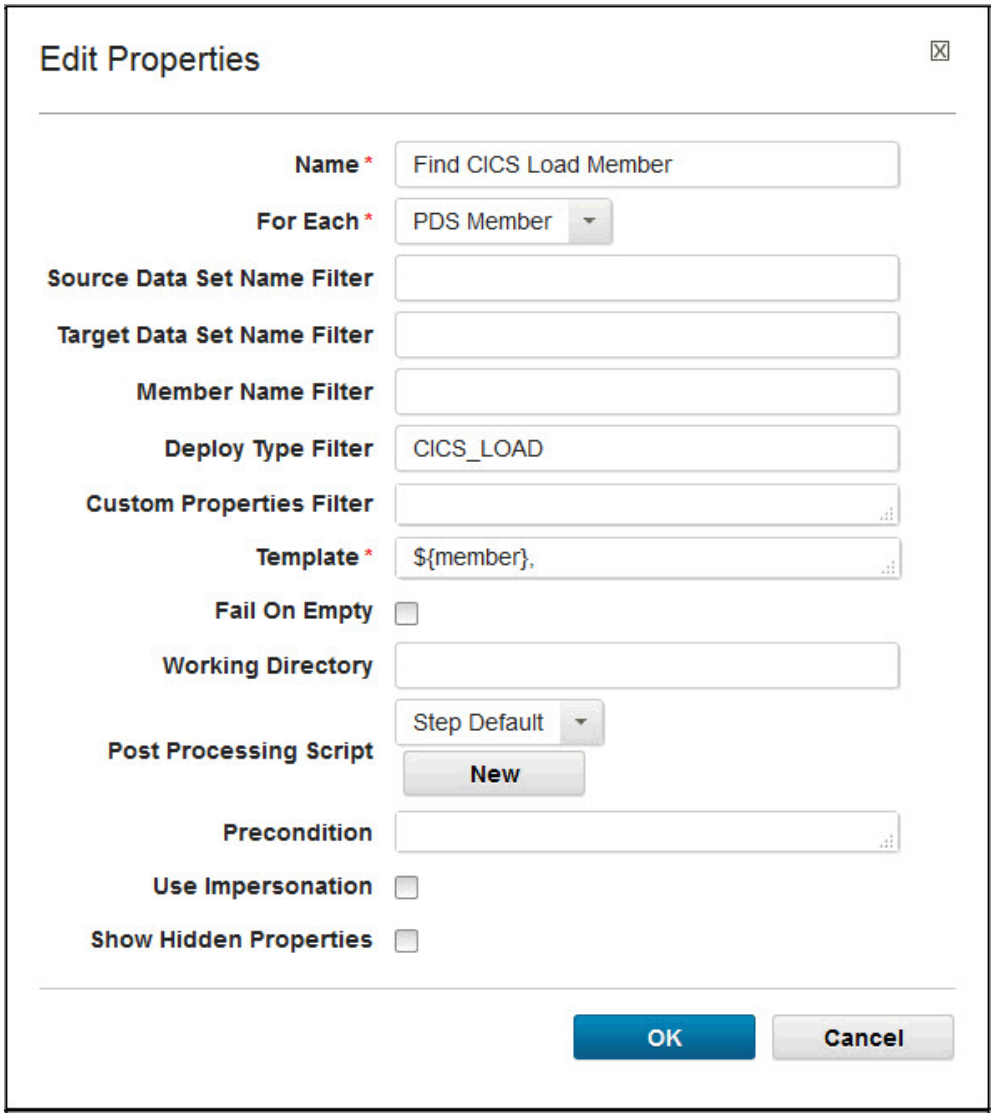


Figure 5-21 Find the CICS Load Member step properties

Table 5-1 lists several properties of particular note.

Table 5-1 Properties to note for the Find CICS Load Members step

Property name	Value	Explanation
For Each	PDS Member	For each member in the PDS data set
Deploy Type Filter	CICS_LOAD	Artifacts with a type=CICS_LOAD
Template	${member},	List member names separated by comma

Finding the DBRM Lib

This is one of the two preparation steps for DB2 bind. It generates the DBRM library data set name that is used in the DB2 Bind steps. Figure 5-22 shows the values that we specified in our example.

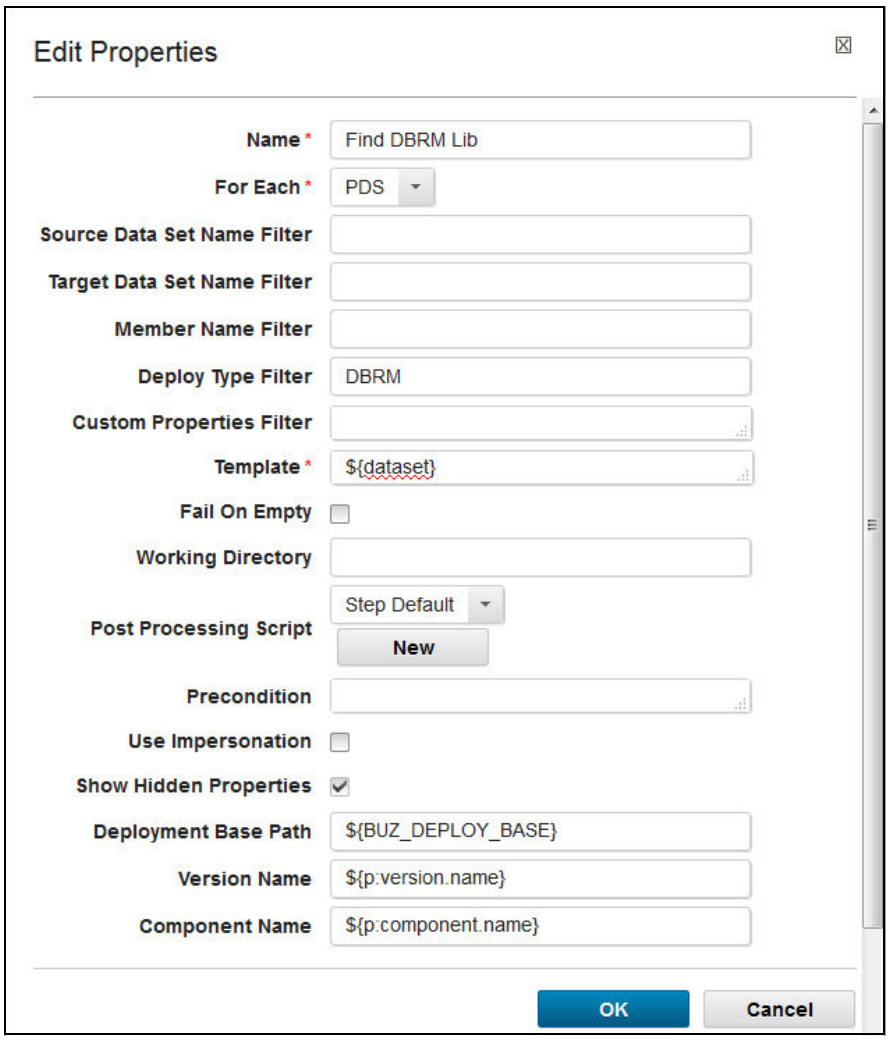


Figure 5-22 Find the DBRM Lib step properties

Table 5-2 lists several properties of particular note.

Table 5-2 Properties to note for Find CICS Load Members step

Property name	Value	Explanation
For Each	PDS	For each PDS data set.
Deploy Type Filter	DBRM	Artifacts with a type=CICS_LOAD.
Template	${dataset}	Generate the data set name for the DBRM library. We are expecting only one here.

Finding the DBRM Member

This is second of the two preparation steps for DB2 bind. It generates the DBRM member list that is used in the DB2 Bind steps. Figure 5-23 shows the values that we specified in our example.

Pay special attention to the For Each, Deploy Filter Type and Template fields.

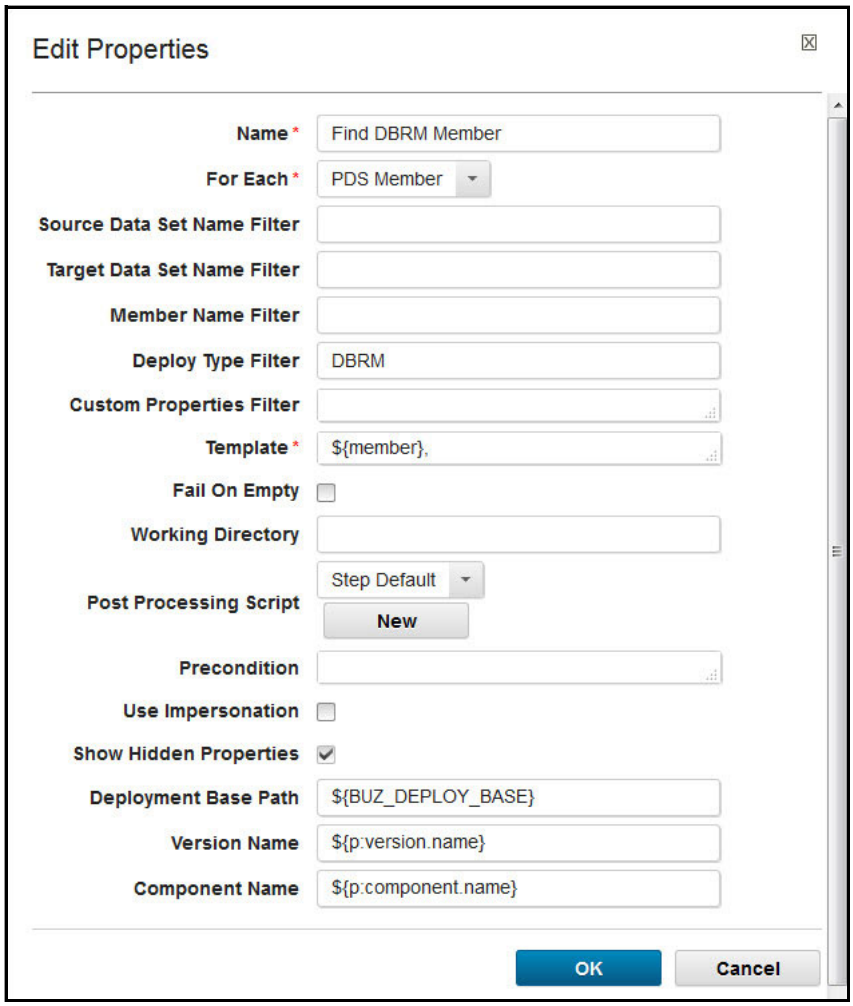


Figure 5-23 Find the DBRM Member step properties

Bind Package step

Now that you have the DBRM library name and DBRM members list from the previous steps, create a DB2 Bind Package. Find the bind steps by using the Step Palette, as shown in Figure 5-24 on page 85.

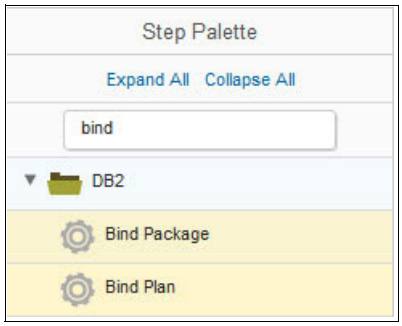


Figure 5-24 Find the bind steps

Figure 5-25 shows how the properties are configured in our scenario.

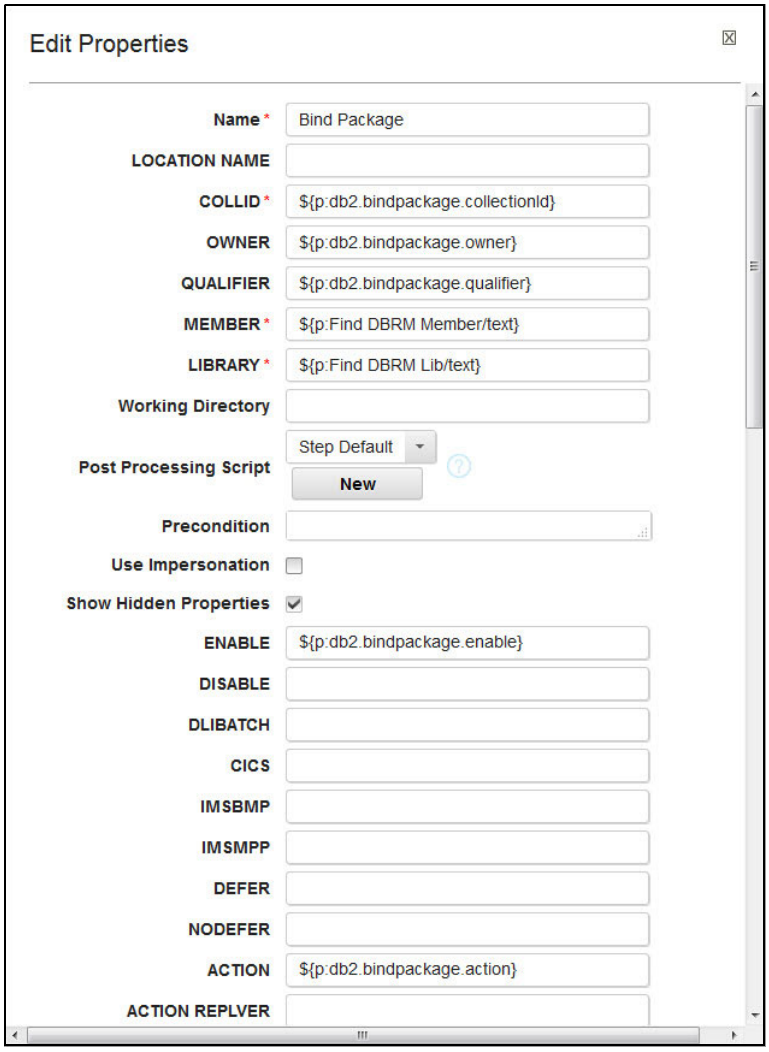


Figure 5-25 Bind Package step properties

In particular, the MEMBER and LIBRARY properties take information that is generated from previous steps, as shown in Table 5-3.

Table 5-3 Specify the DBRM information in bind by using information from previous steps

Property name	Value	Explanation
MEMBER*	${p:Find DBRM Member/text}	Take the DBRM data set name that is generated from the Find DBRM Lib step.
LIBRARY*	${p:Find DBRM Lib/text}	Take the DBRM member list that is generated from the Find DBRM Member step.

Note: Selecting the **Show Hidden Properties** check box shows the list of hidden properties for this step. However, you might not need to specify all the properties. Which properties are required for the binding depends on how the system is set up in your environment. Typically, the DB2 administrator in the organization has this information. All the properties that are specified *must be resolved* during process execution. Unresolved properties result in the step and process failing. So, blank out the properties that are not needed.

Most of the binding properties describe the behavior of the application and how it interacts with DB2. Therefore, it is a preferred practice to specify the value of these properties at the application level. Figure 5-26 shows where you can define application properties. Click the **Applications** tab, select your application (for example, GENAPP COBOL+Policy), and click **Configuration** → **Application Properties**.

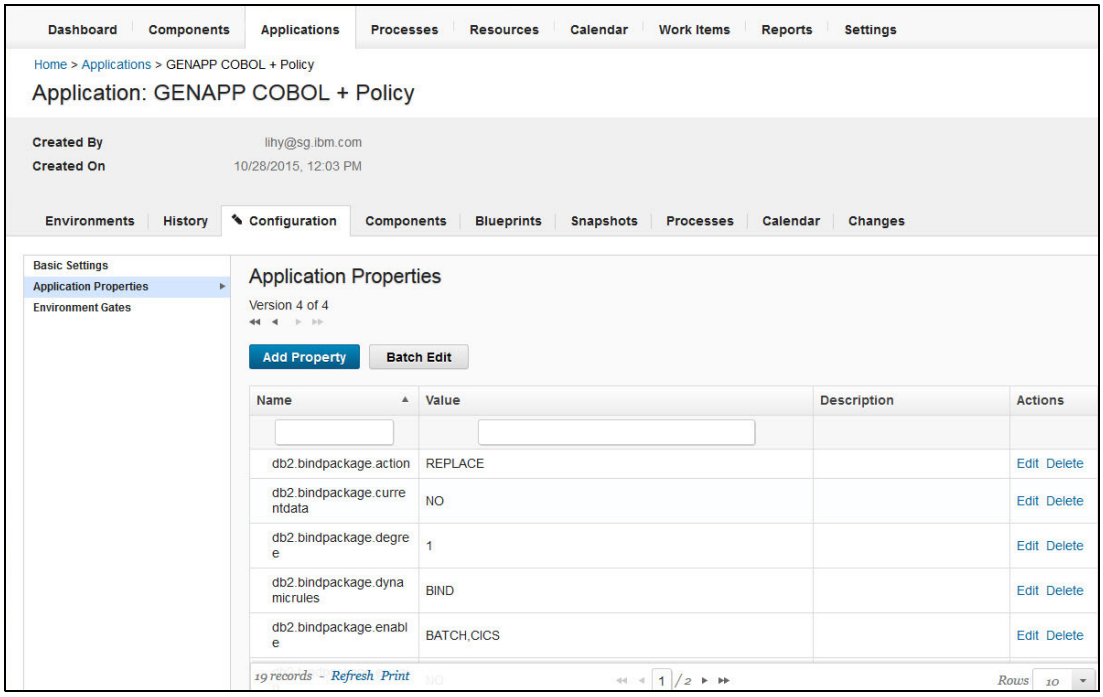


Figure 5-26 Specify the application properties

Bind Plan step

Similar to Bind Package, define a Bind Plan step, as shown in Figure 5-27.

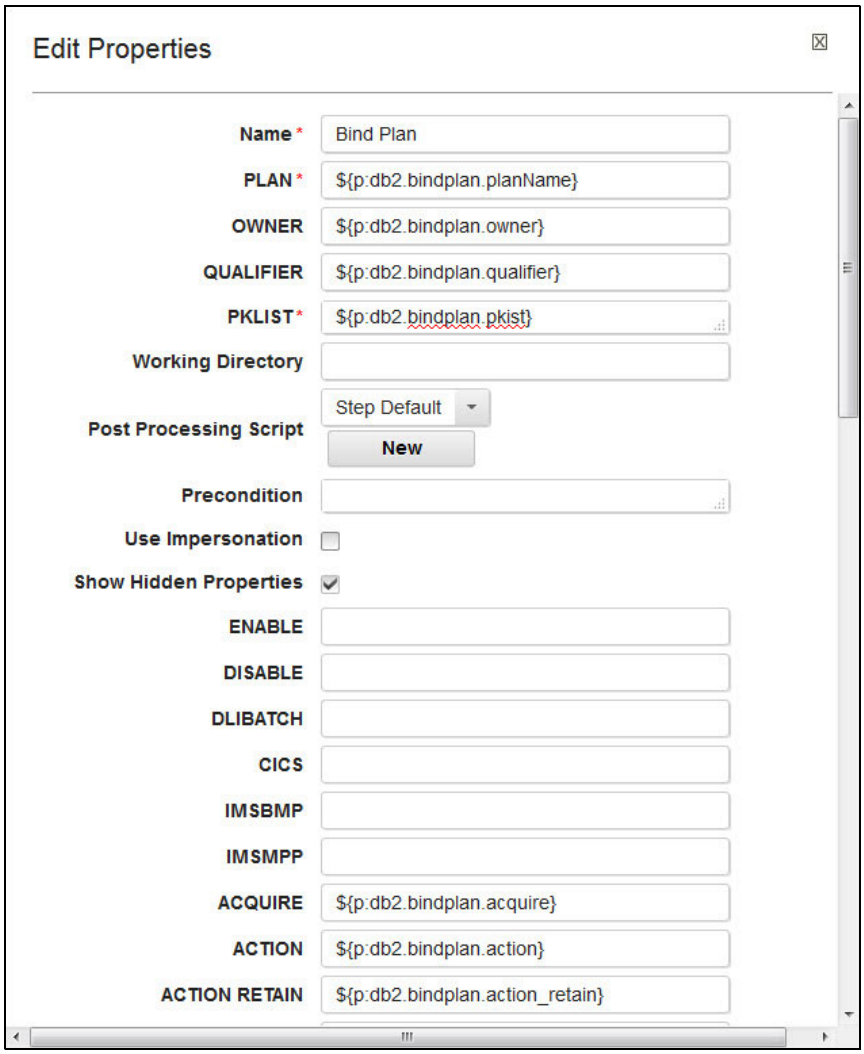


Figure 5-27 Bind Plan step properties

New Copy step

Now we are ready to install the artifacts by completing the New Copy step. Define the properties that are shown in Figure 5-28. For Resource Name List, use the following syntax to get the list of programs from the Find CICS Load Member step:

```
${p:Find CICS Load Member/text}
```

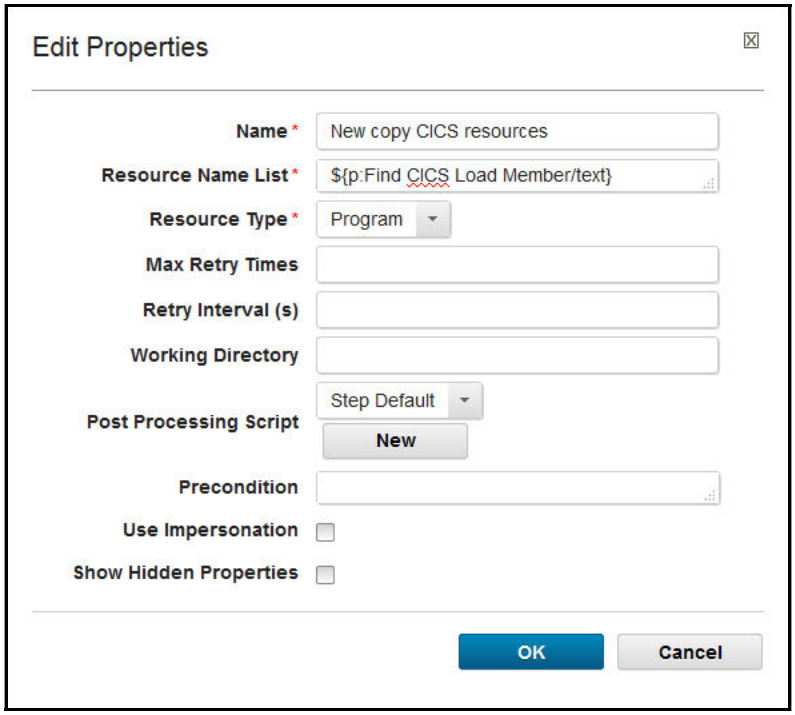


Figure 5-28 New Copy step properties

Now, you have all the steps that are needed to deploy the component GENAPP Base. Make sure to connect all the steps from start to finish to complete the process.

5.2.5 Defining resources and adding a component to them

IBM UrbanCode Deploy resources represent system topologies, including logical partitions, databases, application servers, and so on. Before you can deploy any application, you must define the resources to which the artifacts are to be deployed.

Figure 5-29 on page 89 shows the resource tree that we created for this scenario. We use “Sandbox Sysplex”. Within the Sandbox Sysplex, we have an LPAR named DevOps LPAR, and an agent for this LPAR called WINMVSH1, under which we have an IBM CICSPlex® with region CICSDA01 and a DB2 subsystem called DJH1.

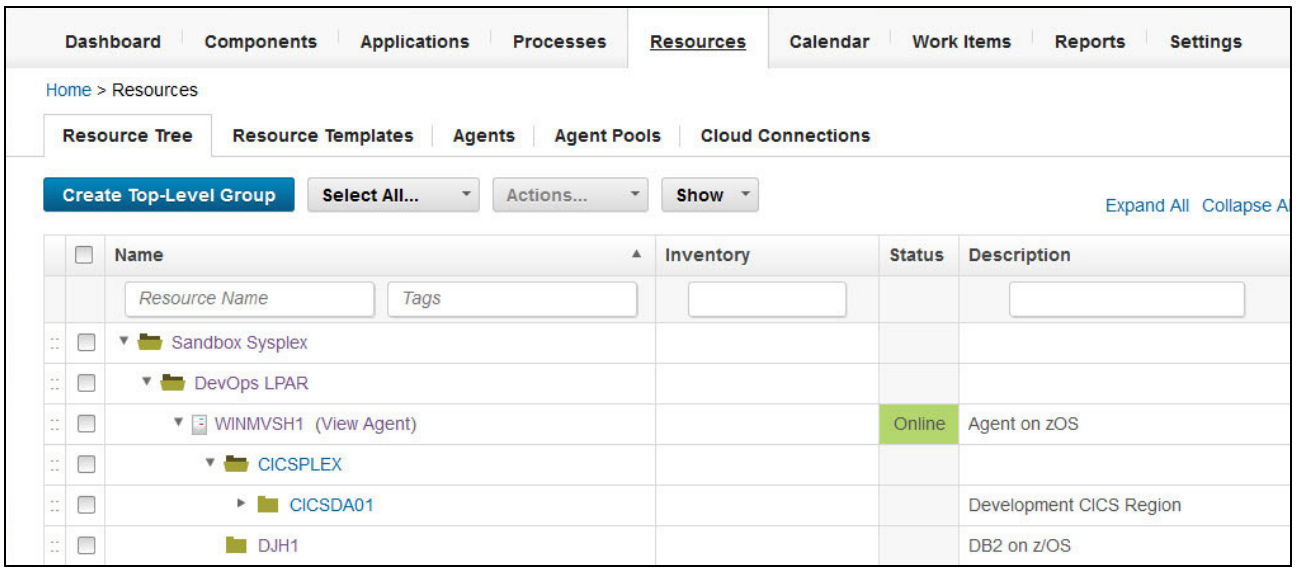


Figure 5-29 Resource tree

Complete the following steps:

1. Create the resource tree that is shown in Figure 5-30 by creating the top-level resource group Sandbox Sysplex. To do so, click **Resources** → **Resource Tree** → **Create Top-Level Group**.

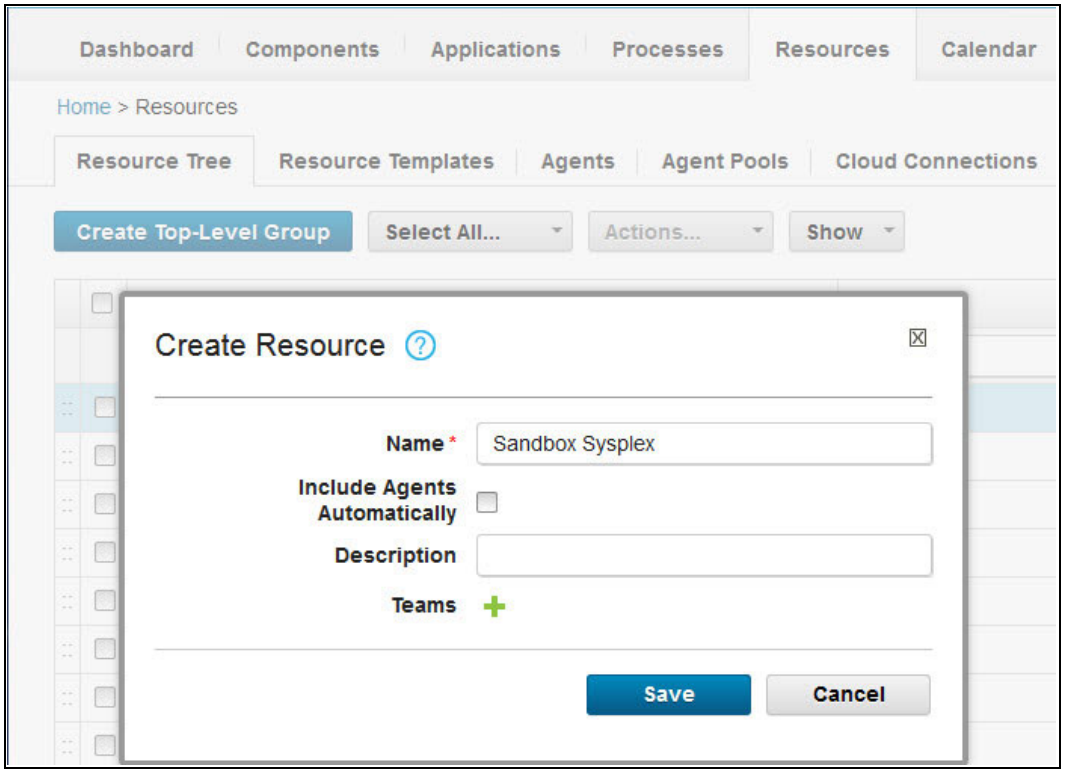


Figure 5-30 Create top-level resource group

2. Click **Save**. Your top-level resource group Sandbox Sysplex is created.

3. Hover your cursor over the Sandbox Sysplex and an Actions menu appears. Click the drop box to see the list of actions that are available, as shown in Figure 5-31.

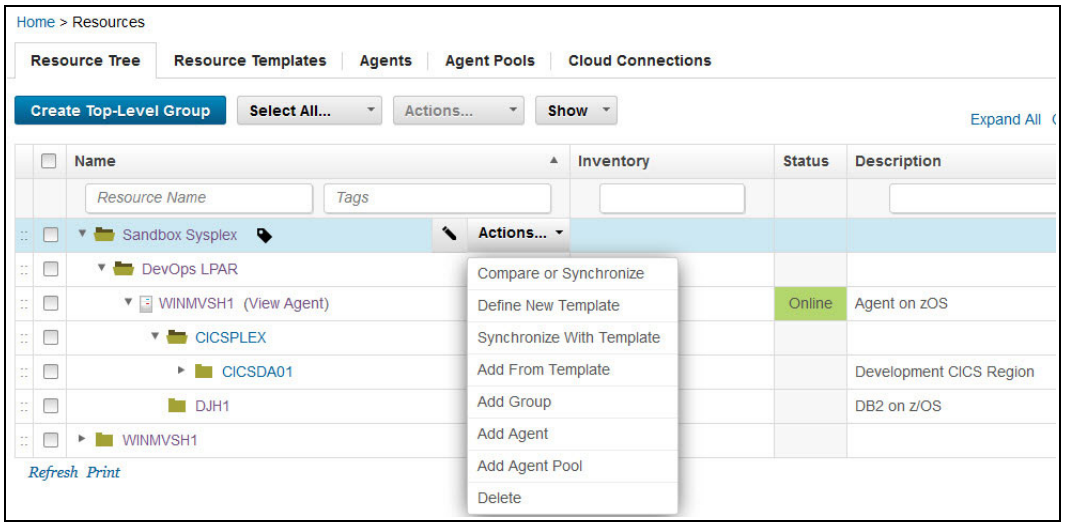


Figure 5-31 Add resources

4. Select **Add Group** to add other resources under Sandbox Sysplex to complete the resource tree according to the figure.
5. Click **Add Agent** to select the agent for this LPAR from the list of installed agents (assuming the agent is installed).

Next, you must add a component to resources. This can either be done from Resource Tree or from Application Environment Resource. In this example, we add the component from Resource Tree. As the GENAPP Base component is a CICS component, we add it to the CICS region CICSDA01 where it will be deployed.

Complete the following steps:

1. Hover your cursor over CICSDA01 and click **Actions**. Select **Add Component**, as shown in Figure 5-32 on page 91.

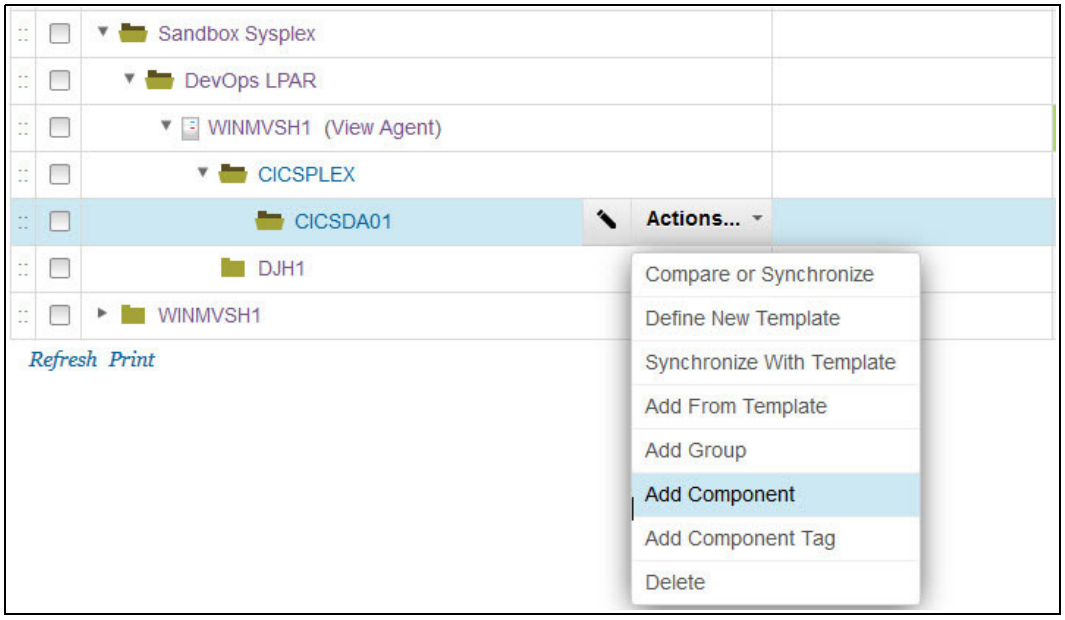


Figure 5-32 Add a component to resources

2. Select GENAPP Base from the list of components and click **Save**. You see that the component is associated to the CICS Region CICSDA01, as shown in Figure 5-33.

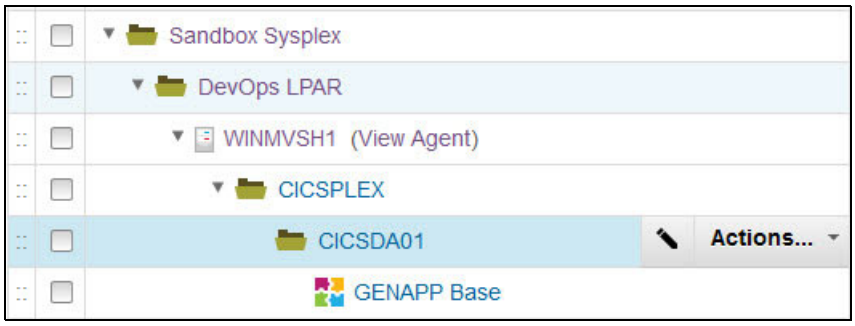


Figure 5-33 Component GENAPP Base added to resource CICSDA01

5.2.6 Creating an application and environment for GENAPP

To run a deployment process of one or more components, an application containing these components must be defined together with the application environments and application processes.

The application environment must have resources that are associated with them. So, this section shows how to create these definitions.

Creating the GENAPP application

To create the GENAPP application, complete the following steps:

1. In the Applications tab, click **Create Application**. In this example, name it “GENAPP Application” (see Figure 5-34).

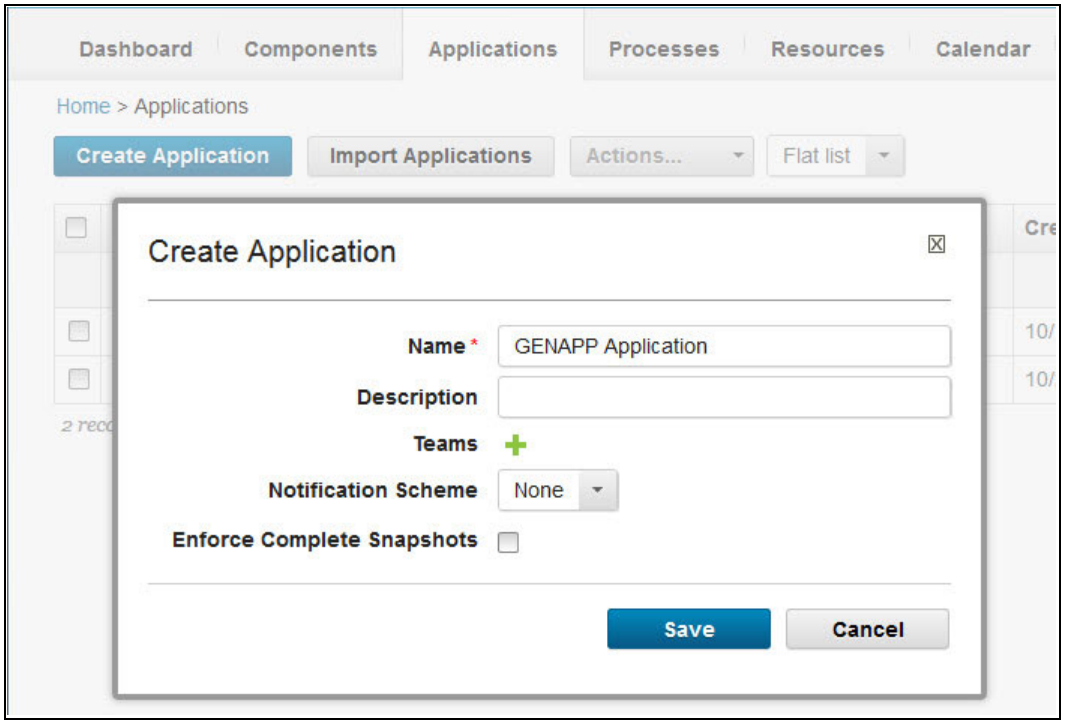


Figure 5-34 Create Application - GENAPP Application

2. Click **Save**. The application is created and you automatically are taken to the application under the Environment tab.

Adding a component to the application

Now, you must add the GENAPP Base component to the application. Click the **Components** tab under Application. Make sure that it is the Components tab within the application and not the primary components table on the top. Then, click **Add Component** and select **GENAPP Base**, as shown in Figure 5-36 on page 94.

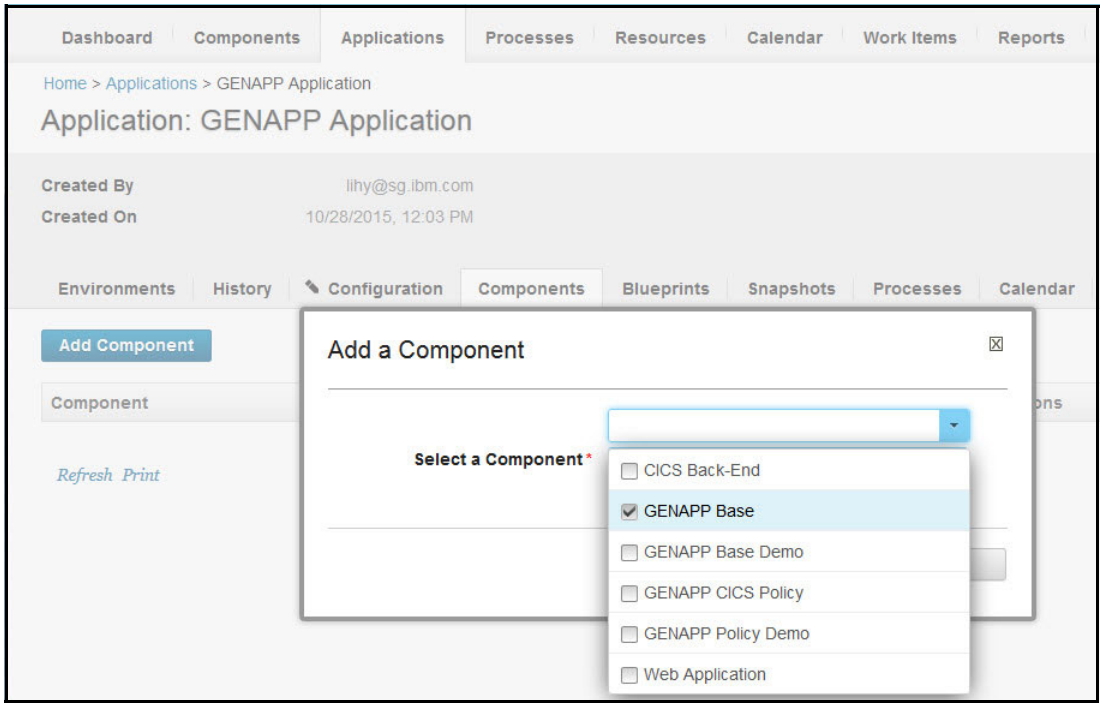


Figure 5-35 Add the GENAPP Base component to the application

Multiple components can be selected to be added to an application. However, in this example select only **GENAPP Base**.

Creating a development environment

Application environments are a collection of resources to host the application. Applications must have at least one environment as the target for deployment.

Complete the following steps:

1. To create a Development environment, click the **Environment** tab under Application and then click **Create Environment**, as shown in Figure 5-36.

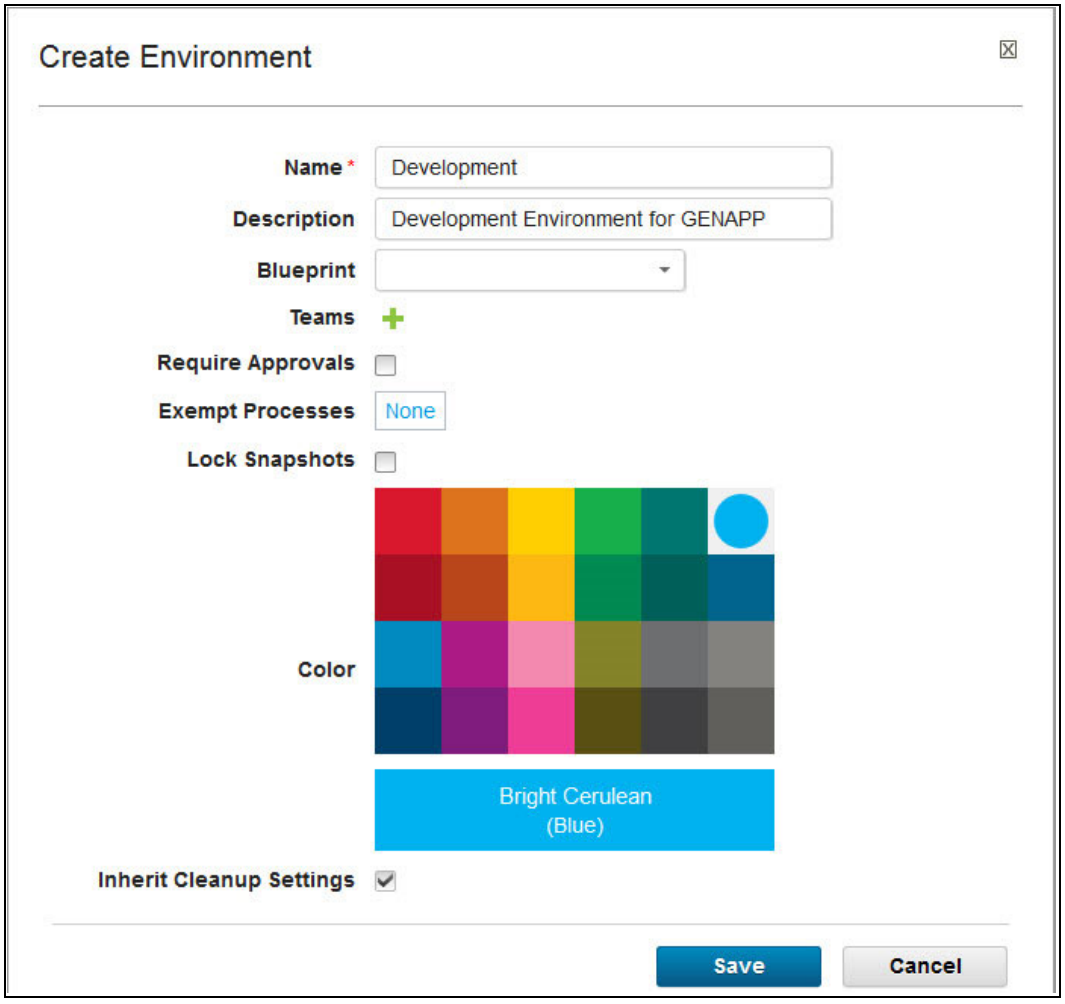


Figure 5-36 Create a development environment

2. Click **Save** to create the environment.
3. Click the development environment.
4. Under the Resources tab, click **Add Base Resources** to select the resources for this environment.
5. Select the Sandbox Sysplex, as shown in Figure 5-37 on page 95. This step associates the whole sysplex and resources under it to this environment, including the GENAPP Base component that was added to CICSDA01.
6. Click **OK** to save.

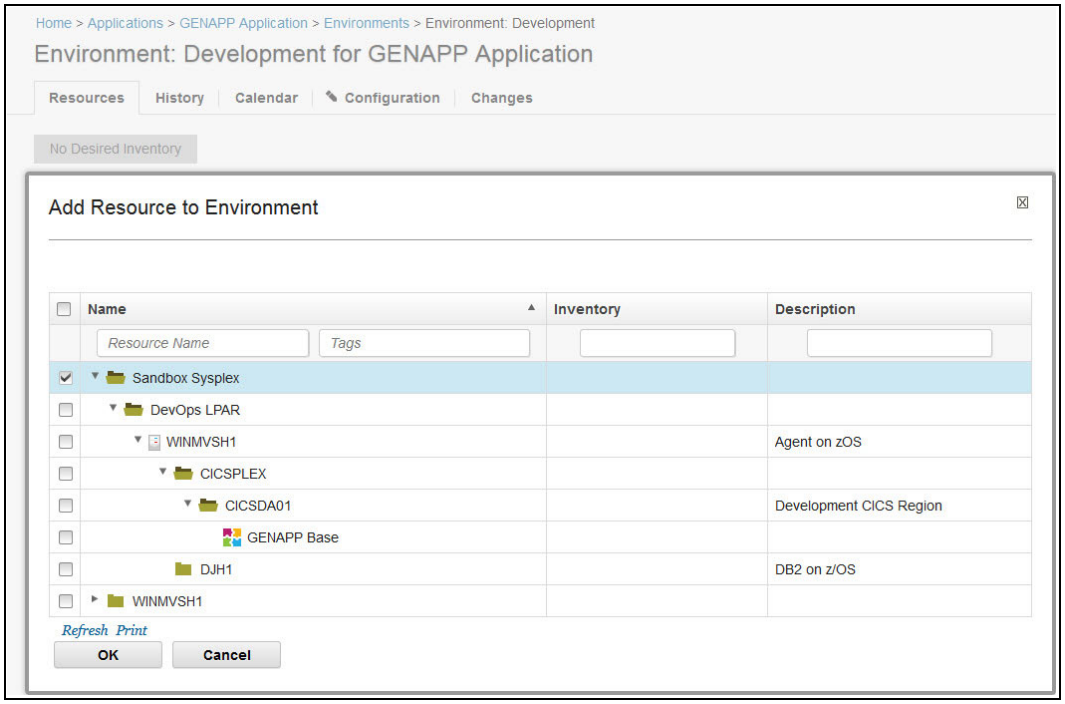


Figure 5-37 Add a resource to an environment

5.2.7 Defining the application deployment process

You have the elements defined (component, application, resources, and environment). Now, you must define the application process to deploy this application.

Complete the following steps:

1. Click the **Processes** tab under Application.
2. Click **Create Process** and complete the fields, as shown in Figure 5-38.

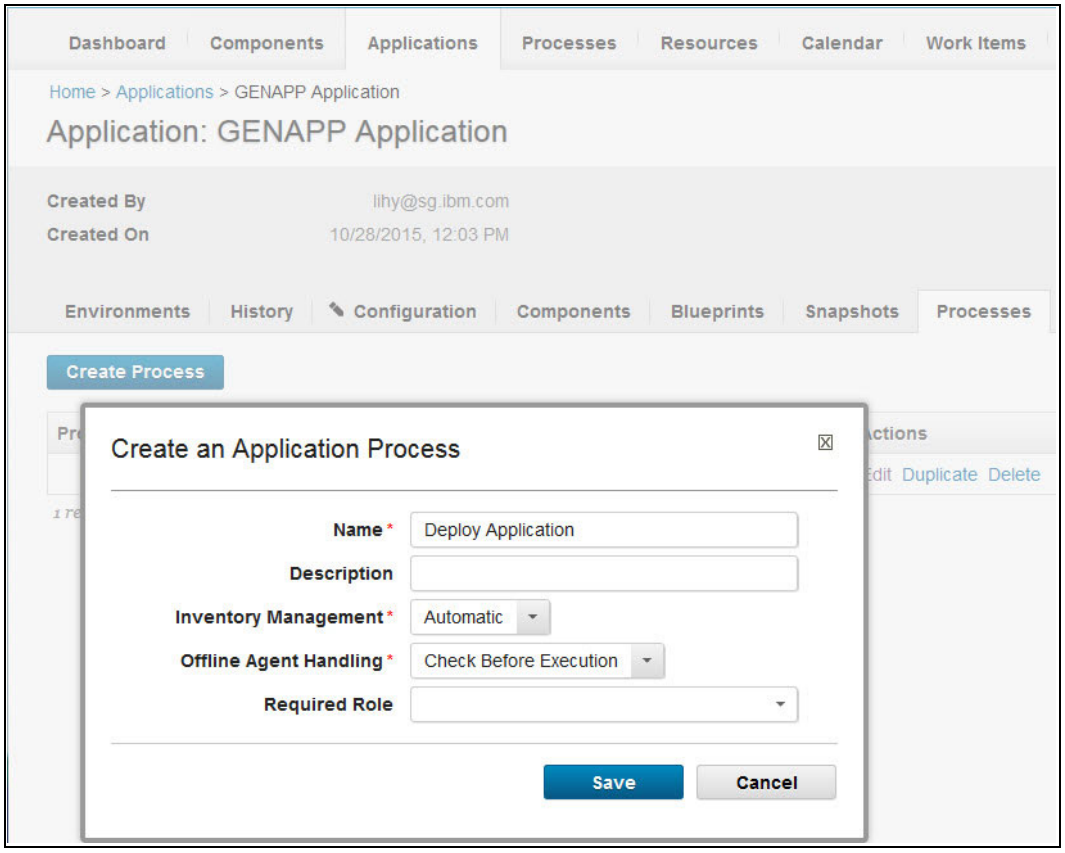


Figure 5-38 Create an application process for GENAPP Application

3. Click **Save** to create the process.

Similar to defining the GENAPP base component process, define the steps for this application process by completing the following steps:

1. Click the deployment application process that you created to open the process design canvas, as shown in Figure 5-39 on page 97.

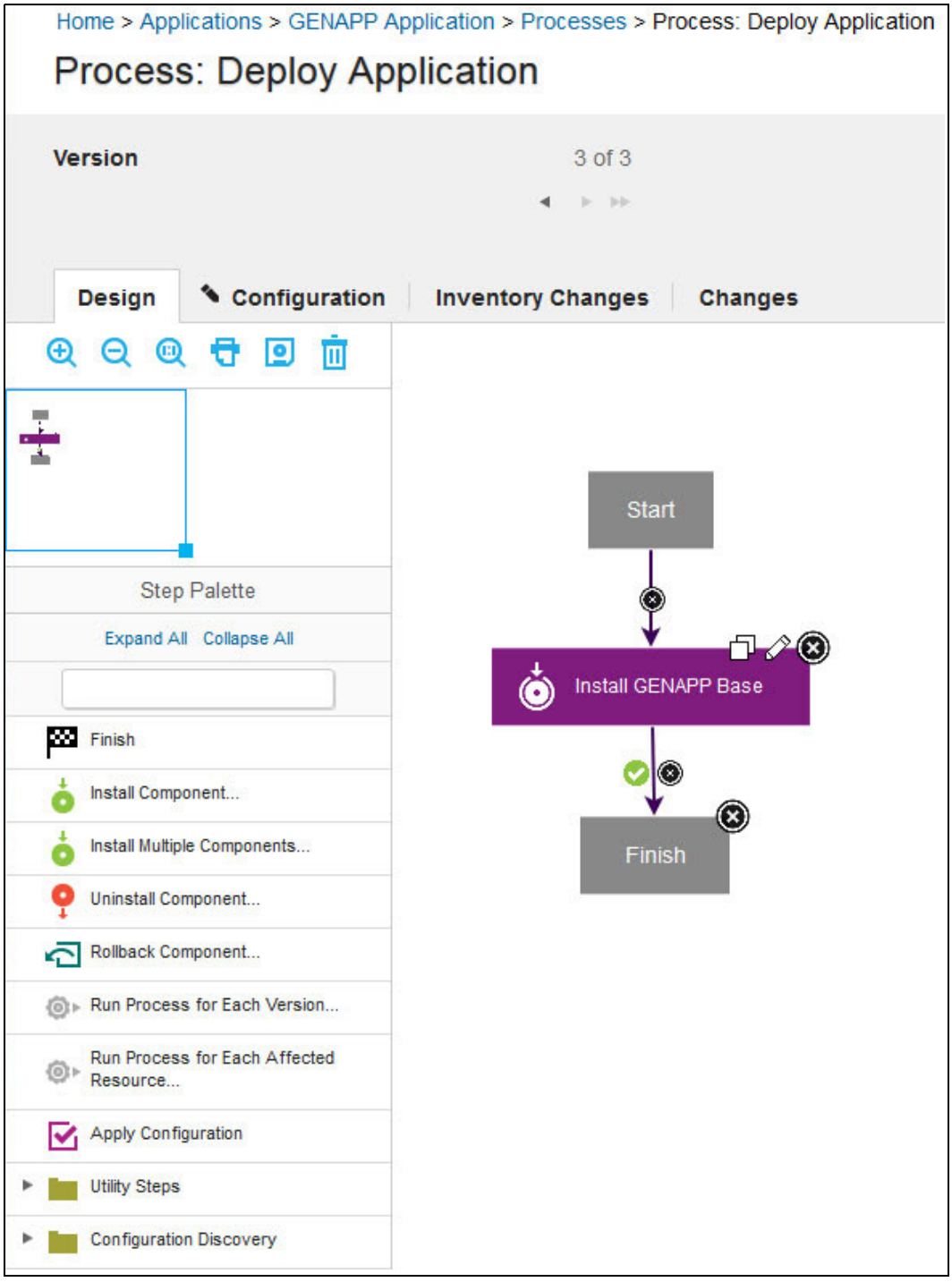


Figure 5-39 Application process to install GENAPP Base

As application processes are concerned only with the component level, it is typical for an application process to be an assembly of its component processes.

The Step Palette for application process has a much shorter list of steps that represent operations on components such installation, uninstallation, rollback, and so on.

As there is only one GENAPP Base component in the example application at the moment, create a rather simple process with only one step to deploy the GENAPP Base component. This step starts the component process “Deploy” that was defined for the GENAPP Base component.

2. Select **Install Component** from the Step Palette and configure the properties, as shown in Figure 5-40.

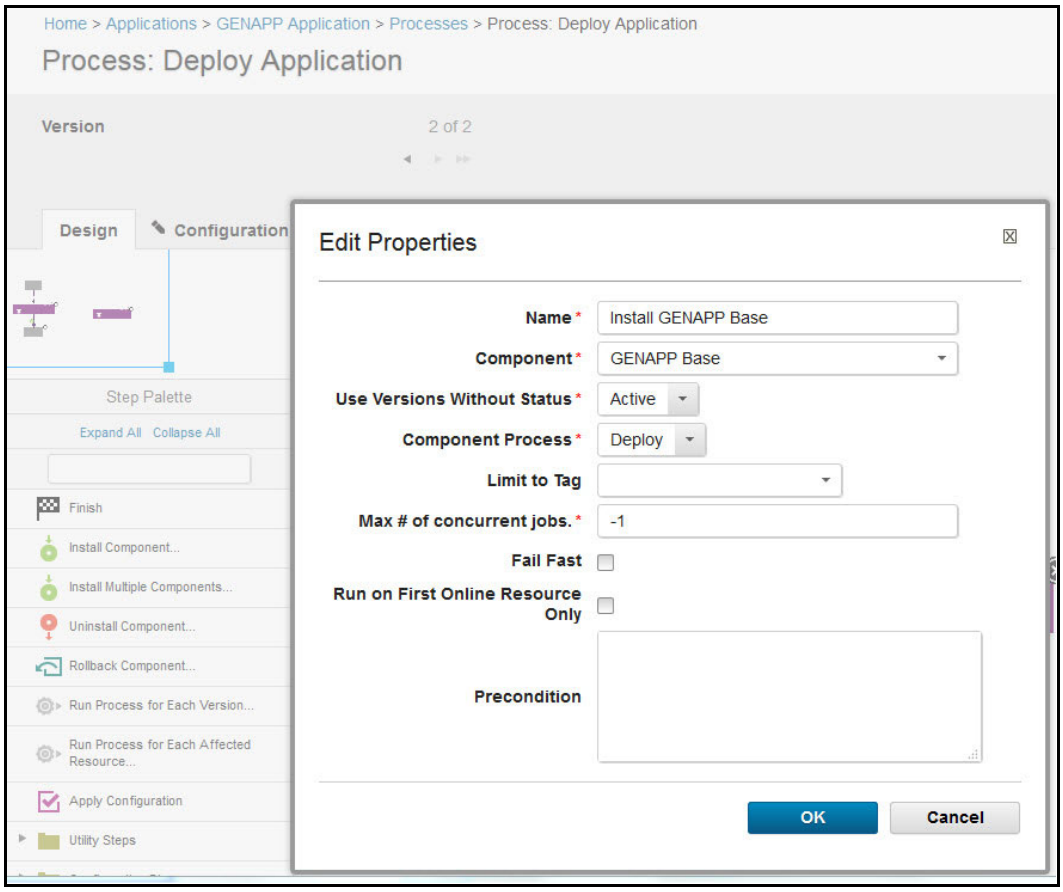


Figure 5-40 Configure the installation component step to deploy the GENAPP Base component

3. Click **OK** to save.
4. Connect the start and finish and save the process.

5.2.8 Deploying the GENAPP Base component in the Application

Now, you can run the application process, which starts the component deployment process to install the GENAPP Base component into the environment. Complete the following steps:

1. Click **Applications** → **GENAPP Application** → **Environments**.
2. Click **Play** to request the process, which is marked by a circle with an arrow inside it, as shown in Figure 5-41 on page 99.

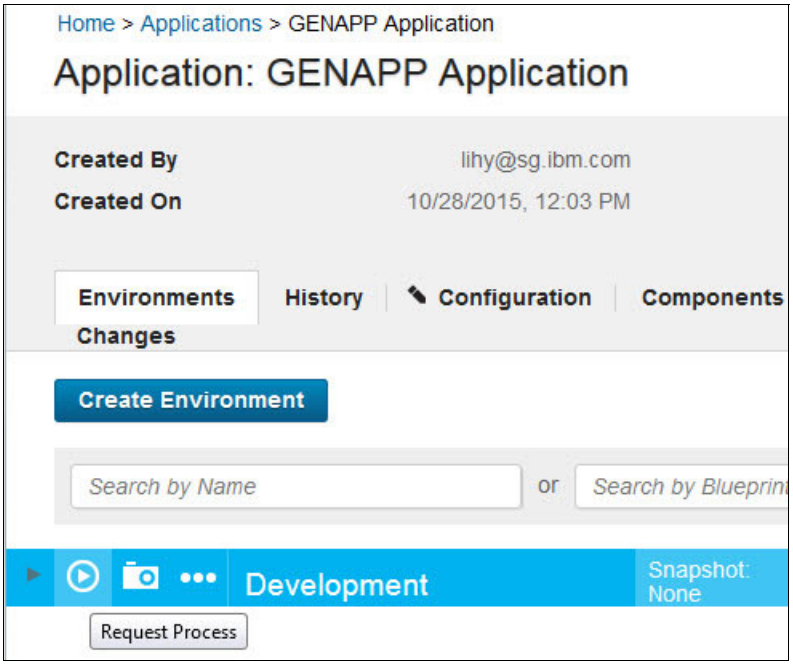


Figure 5-41 Request a process in the environment

3. In the Run Process on the Development window, select **Deploy Application** from the drop-down list as the process.
4. Click **Choose Versions** in the Component Versions window, and add the latest version to be deployed. In this example, as shown in Figure 5-42, it is version 3.

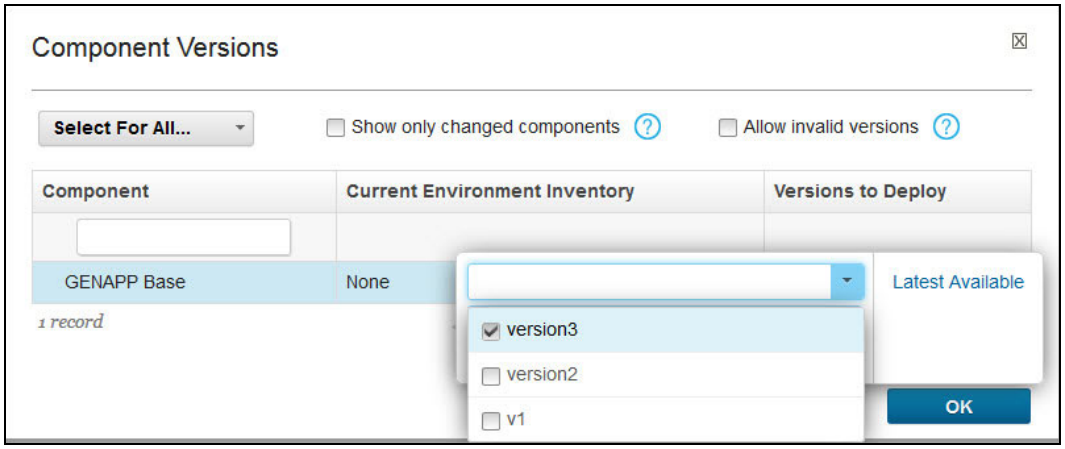


Figure 5-42 Select component version

5. Click **OK** to save and submit the process. The process execution window opens, which shows the progress of the process and the status of each step. Figure 5-43 shows the successful completion of the process.

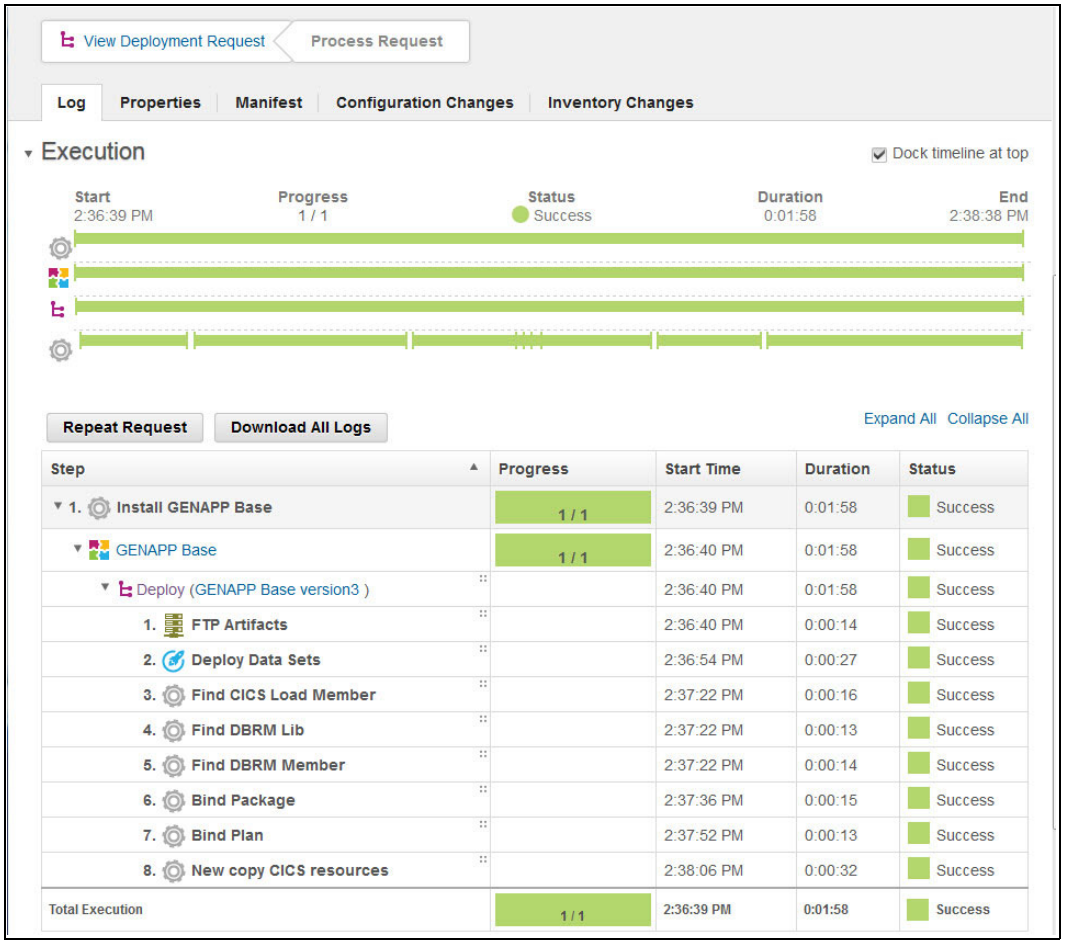


Figure 5-43 Process execution successful

You have completed the build and deployment of the GENAPP Base component.

5.3 Implementing the GENAPP Policy component

This section describes how to add a CICS policy to the GENAPP Base component. Policies were introduced in CICS Transaction Server V5 and are used to protect critical system resources. In this example, we use a policy that generates a system message when the application exceeds a CPU Time threshold.

Assume that the policy bundle is defined and delivered to Rational Team Concert. Focus only on the build and deployment processes.

5.3.1 Creating a build definition

This section describes the steps that are necessary to create a build definition and create a CICS Bundle for the CICS Policy.

The first step is to create a new build definition for a CICS Policy in Rational Team Concert. Complete the following steps:

1. In the Team Artifacts tab of Rational Team Concert, right-click the **Builds** folder and select **New Build Definition**, as shown in Figure 5-44.

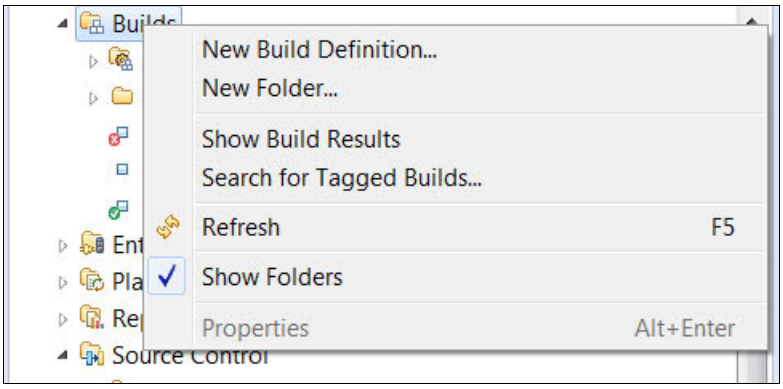


Figure 5-44 New Build Definition

Use the New Build Definition wizard to create a build definition.

2. Click **Next** to define a new build definition, as shown in Figure 5-45.

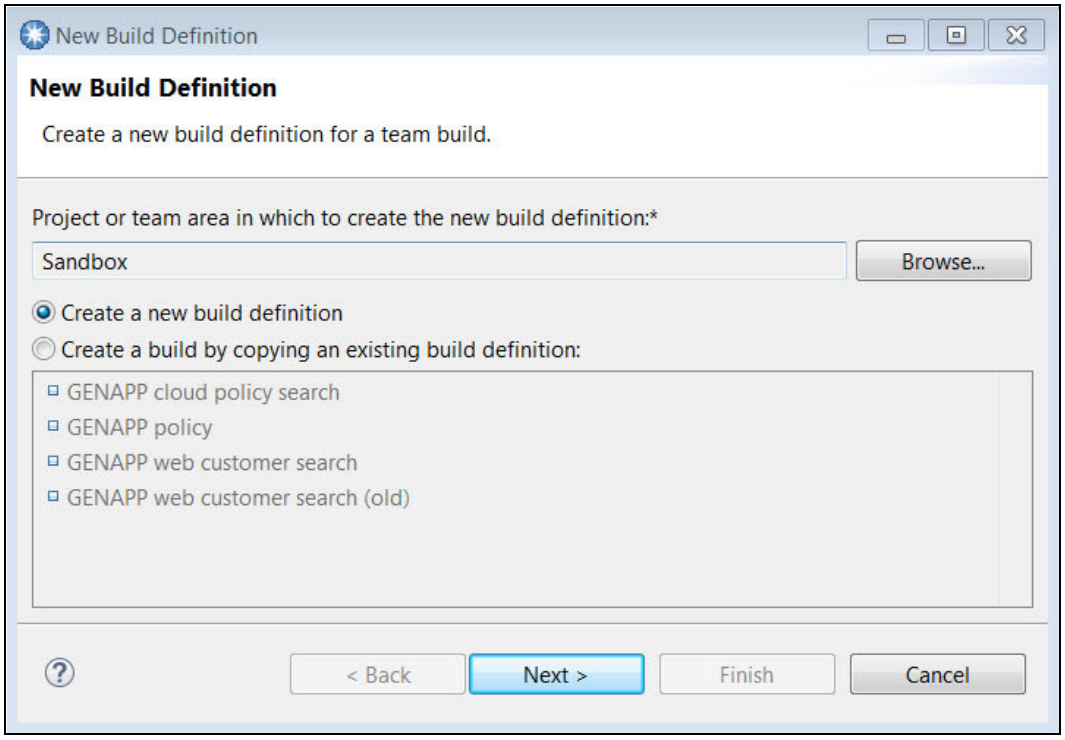


Figure 5-45 Create a build definition

3. Enter GENAPP CICS Policy in both the ID and the Description fields. The ID is the unique name of the build definition. Select the build template from the list of available templates.

4. Select **Ant- Jazz Build Engine** from the Available build templates and click **Next** (see Figure 5-46).

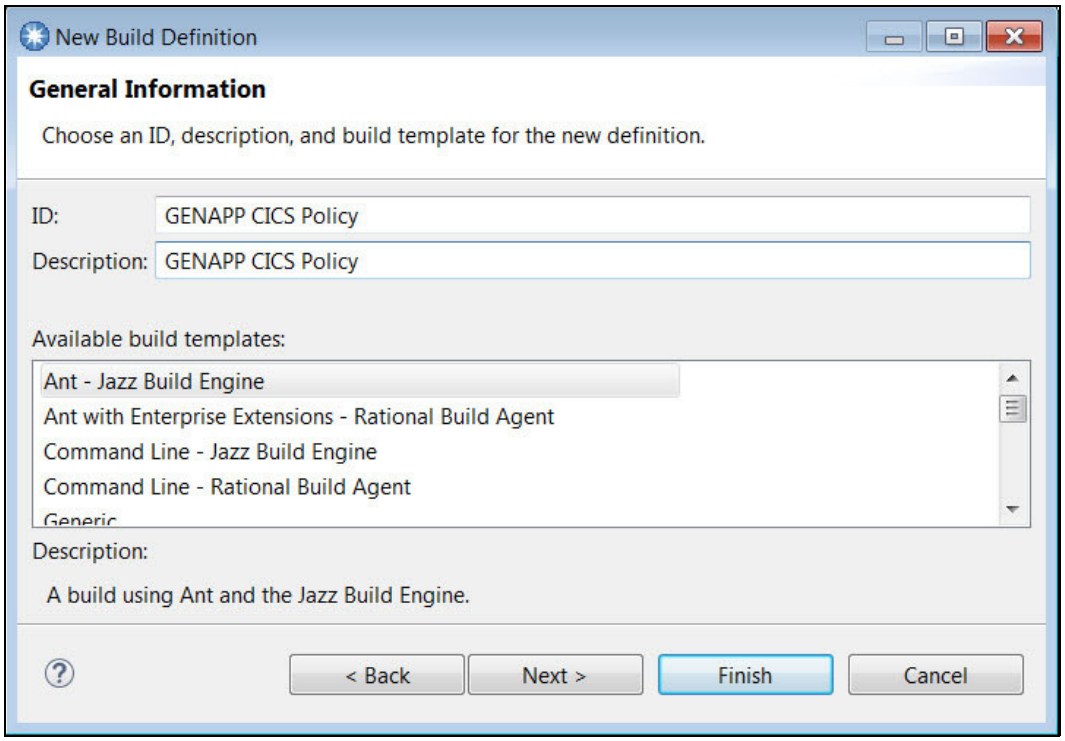


Figure 5-46 Build ID and template

5. The Pre-Build window opens and prompts you for the source code management (SCM) system. In this scenario, use Rational Team Concert and select **Jazz Source Control**, as shown in Figure 5-47 on page 103. Click **Next**.

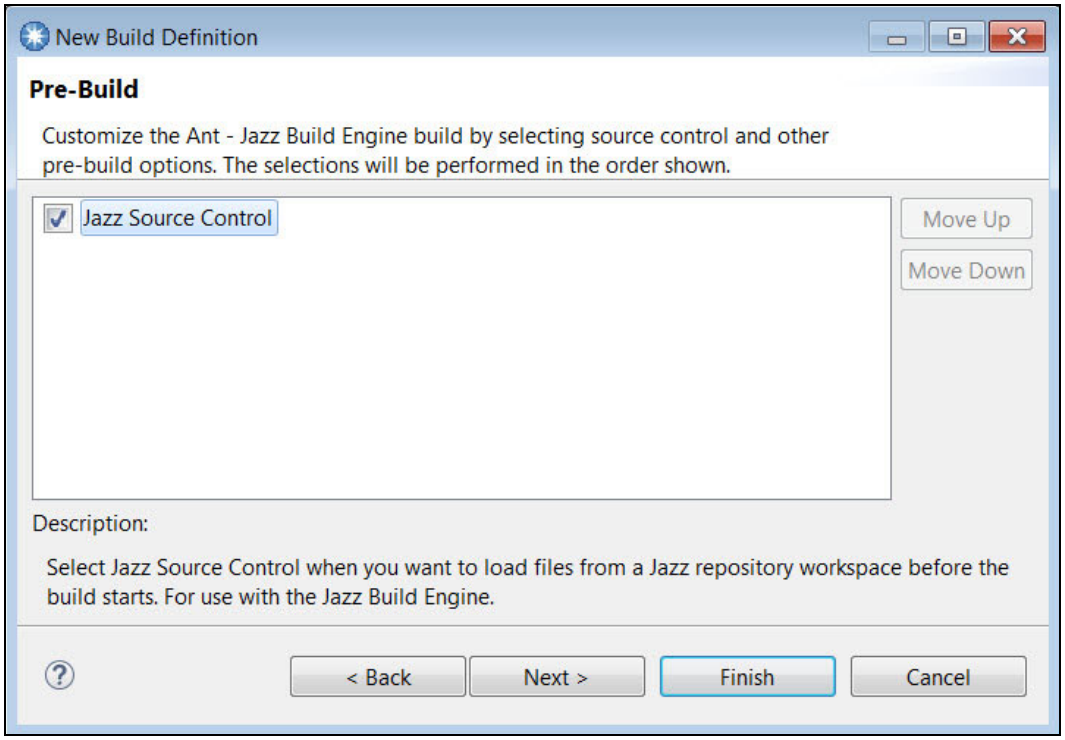


Figure 5-47 Pre-Build option for Jazz Source Control

6. The Post-Build window opens (Figure 5-48). This is an important step for this scenario because you configure the post-build step to publish artifacts to the IBM UrbanCode Deploy CodeStation. Select **Post build Deploy** and click **Finish**.

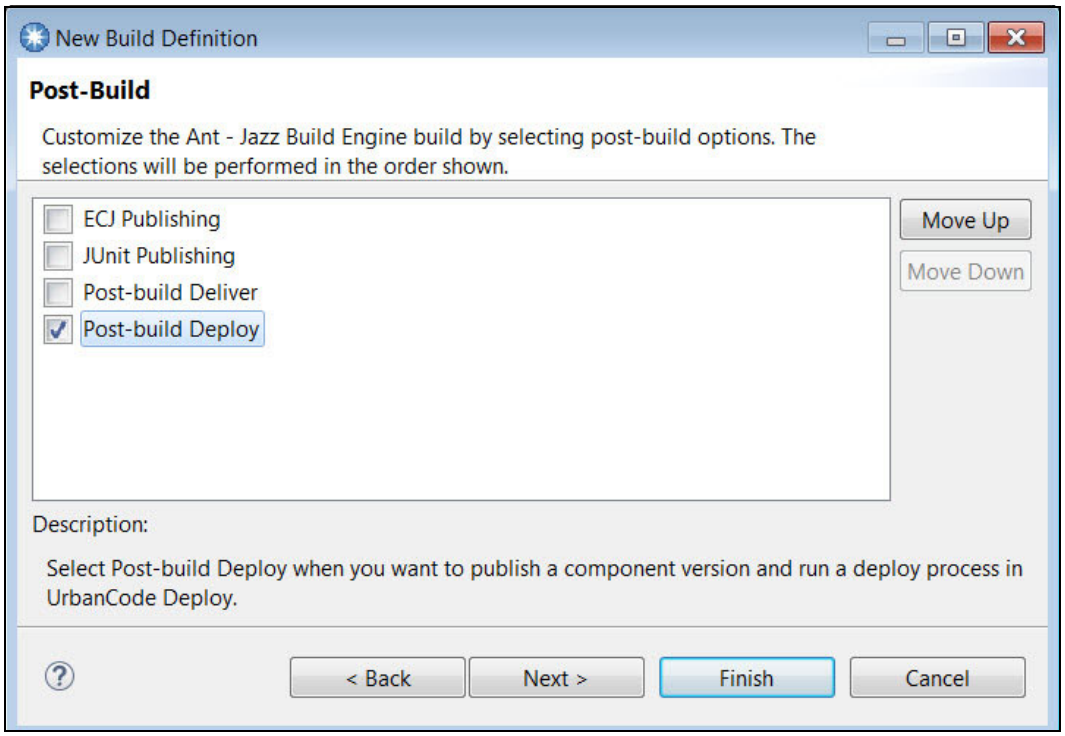


Figure 5-48 Post build Deploy option

With the build definition created, configure each one of the tabs for this scenario. Start with the Overview tab and add the build engines. (The Jazz Build engine was created earlier and is available for use.)

7. Add the build engine by clicking **Add**, as shown in Figure 5-49.

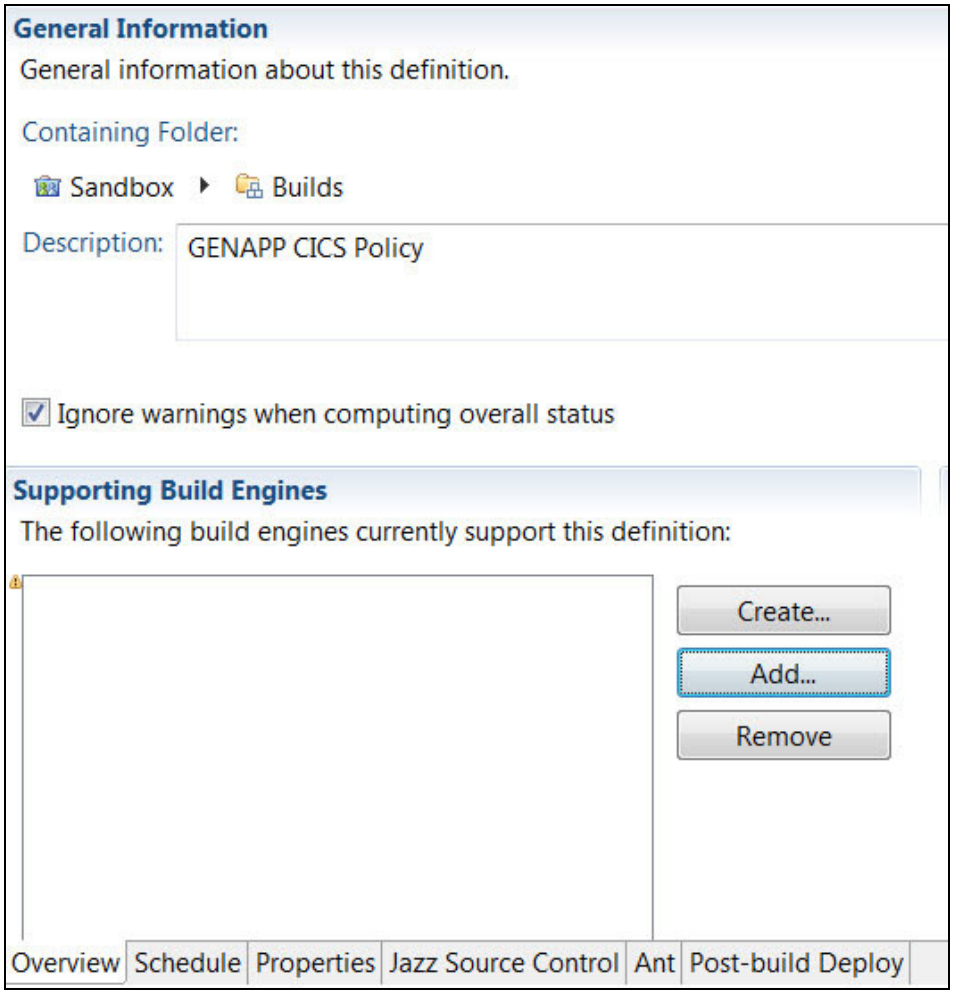


Figure 5-49 Add build engine

8. Select the build engines that the build definition uses. In this example, select the two build engines that are shown in Figure 5-50 and click **OK**.

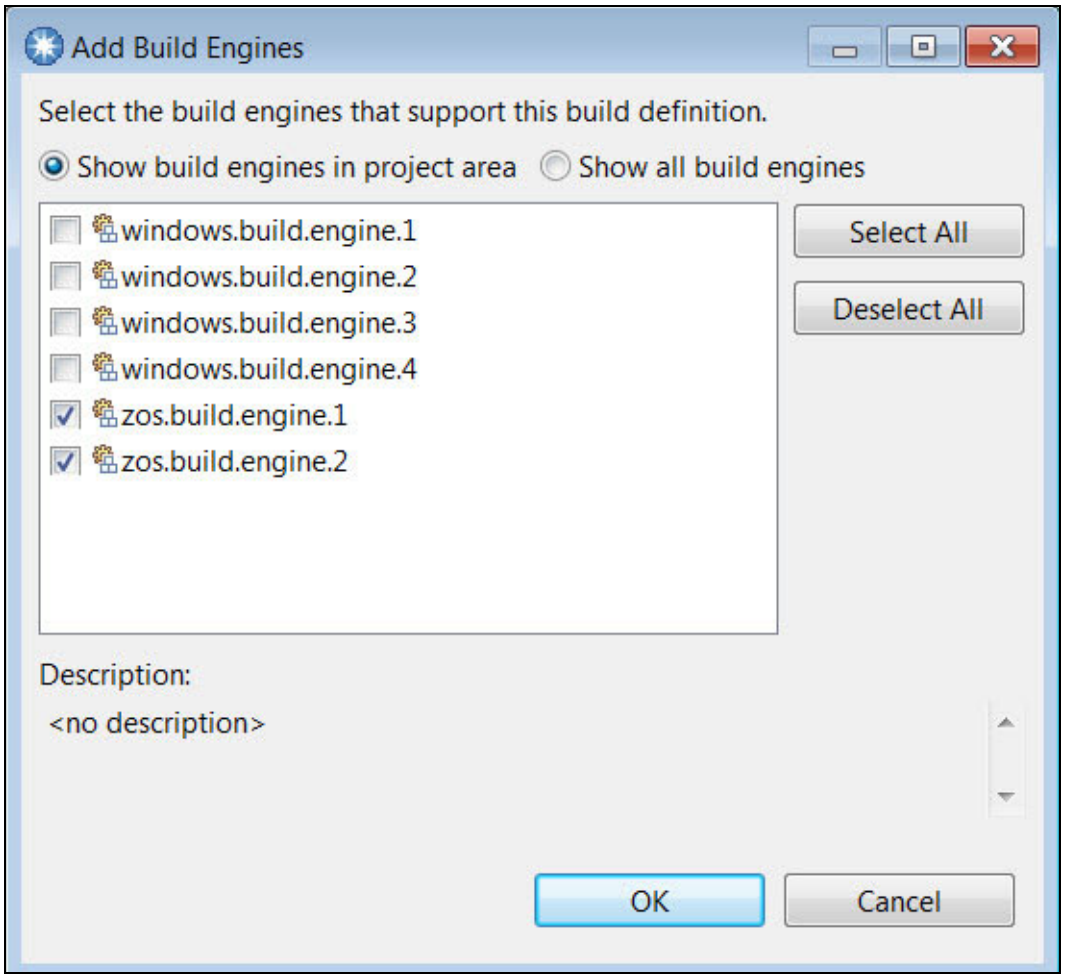


Figure 5-50 Select build engines

The next step is to schedule the build process. In this example, build a continuous build process where new builds are created when new changes are delivered to the Rational Team Concert stream.

9. On the Schedule tab, select **Enabled** and set the Continuous interval in minutes to 1 (Figure 5-51). The engine checks for any changes every minute.

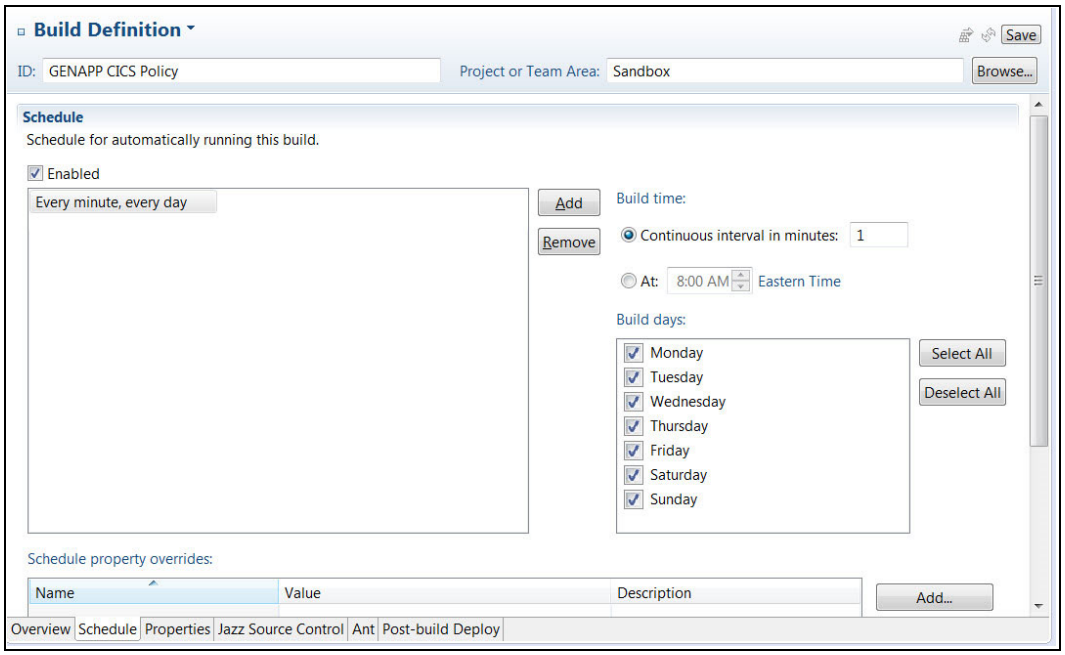


Figure 5-51 Build schedule

The next step is to add properties for the build. Parameterize the source directory, and build the output directory, the CICS Bundle name, and the target platform.

10. Go to the **Properties** tab and click **Add**, as shown in Figure 5-52.

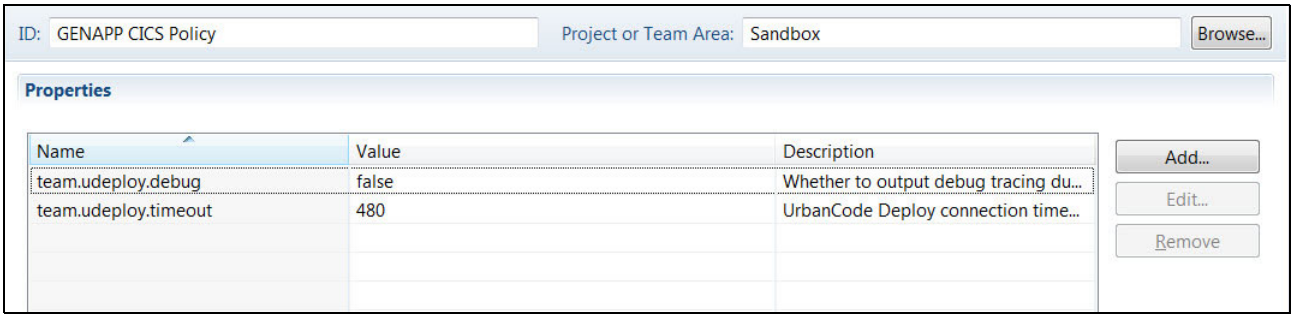


Figure 5-52 Add properties

11. In the next window, which is shown in Figure 5-53 on page 107, select **String** as the property type and click **OK**.

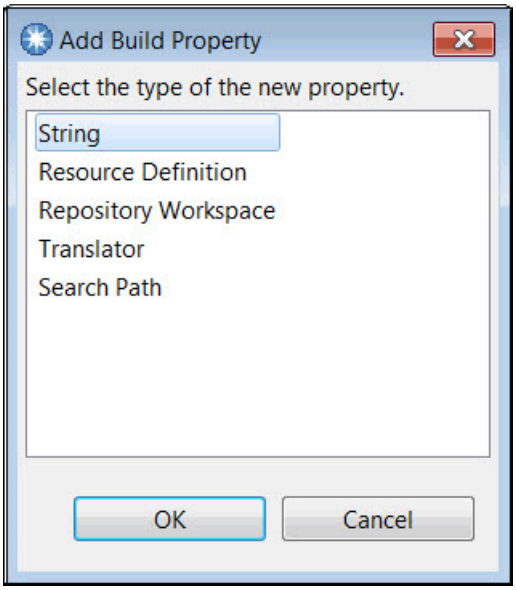


Figure 5-53 Property type

12. Enter the first pair of property names and values, as shown in Figure 5-54.

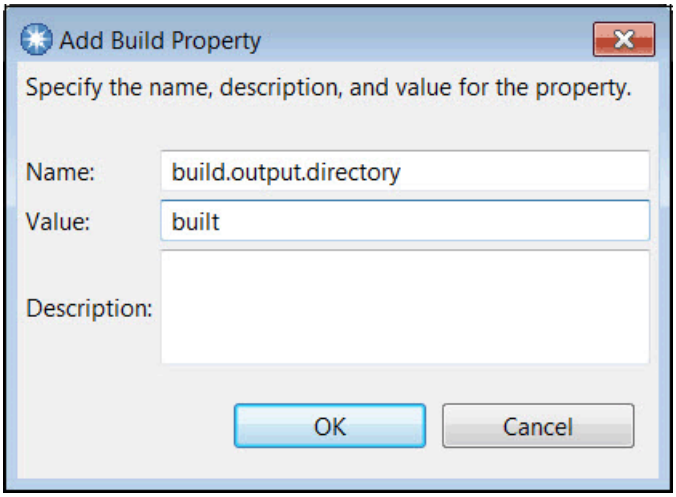


Figure 5-54 Add property name and value

13. Repeat the Add properties steps (always using String as the property type) and add the four pairs of properties and values that are shown in Table 5-4. (You just added `build.output.directory`.)

Table 5-4 Build the definition properties

Property name	Property value
`build.output.directory`	`built`
`bundle.name`	`com.ibm.genapp.cpupolicy.bundle`
`source.dir`	`source`
`target.platform`	`com.ibm.explorer.sdk.web.liberty53.target`

After all the properties are added, the properties tab looks like Figure 5-55.

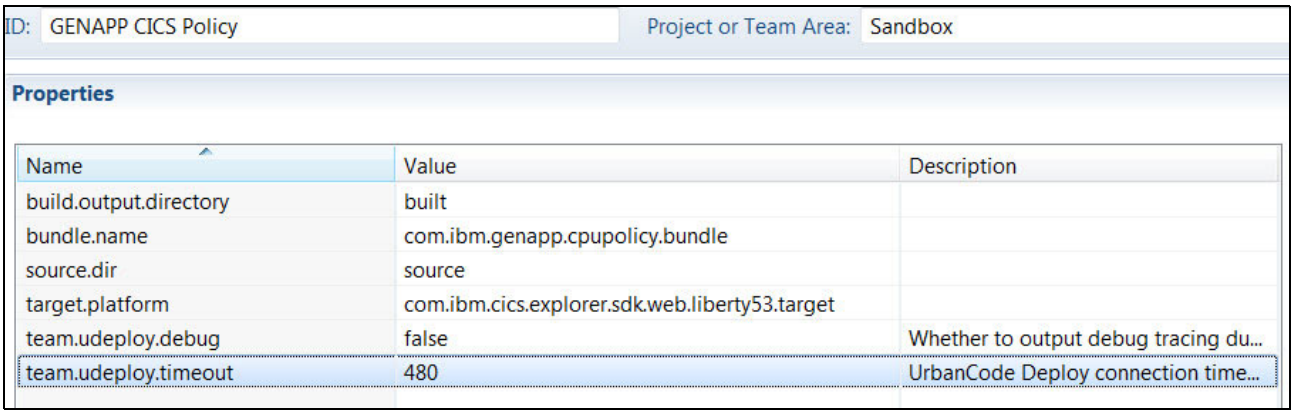


Figure 5-55 Build properties

The next step is to create a repository workspace for the build definition. Repository workspaces are typically used by individual team members to contain their changes in progress. Team members deliver their changes from their repository workspace to the stream and accept changes from other team members into their repository workspace from the stream. Every repository workspace has an owner, and only the owner can make changes in the workspace.

14. On the Jazz Source Control tab, click **Create**, as shown in Figure 5-56.

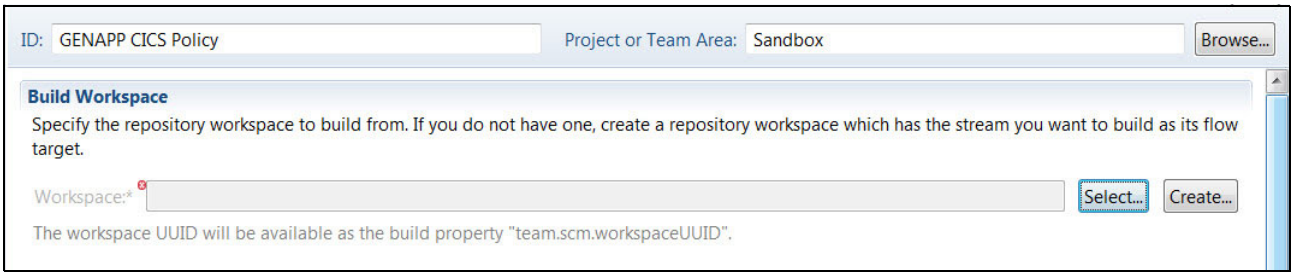


Figure 5-56 Create a workspace repository

A window opens where you select the stream to which you want the changes to flow. Streams are typically used to integrate the work that is stored in repository workspaces.

15. On the New Repository Workspace window, select **Flow with a stream** and select the stream, as shown in Figure 5-57. Click **Next**.

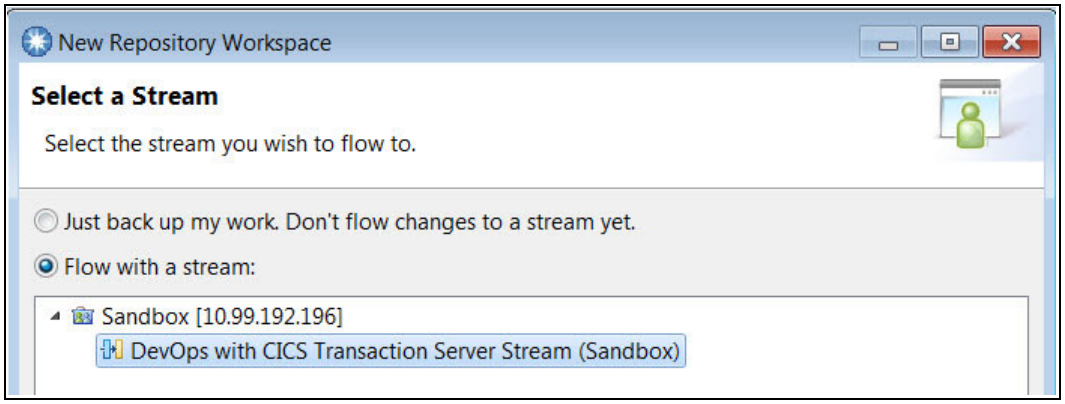


Figure 5-57 Select a stream

16. Set the Repository Workspace Name as BUILD: GENAPP CICS Policy build and click **Next** (Figure 5-58).

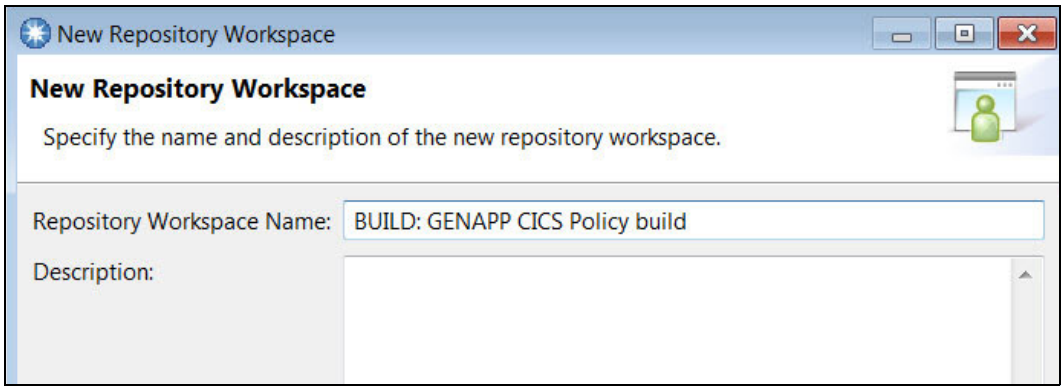


Figure 5-58 New Repository Workspace name

17. Set the Read Access Permission to **Public** and click **Next**, as shown in Figure 5-59.

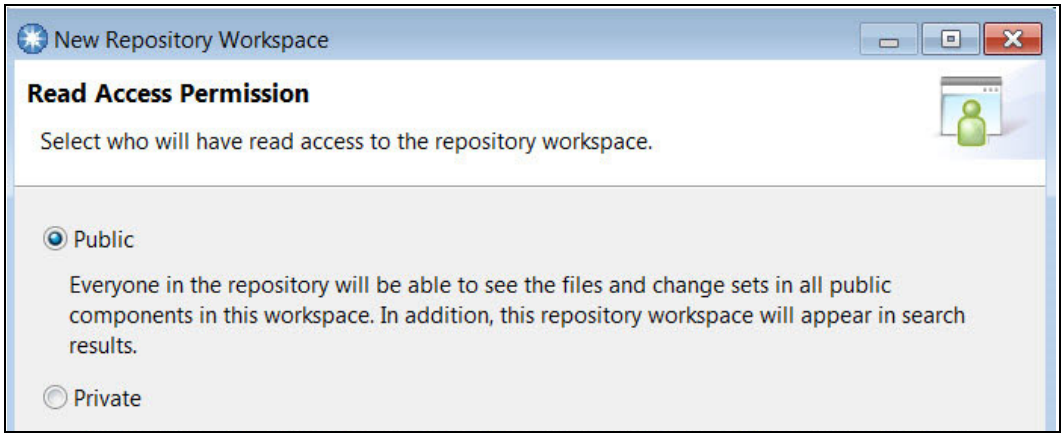


Figure 5-59 Read Access Permission

18. On the Components to Add window, click **Select All** and click **Finish** (Figure 5-60).

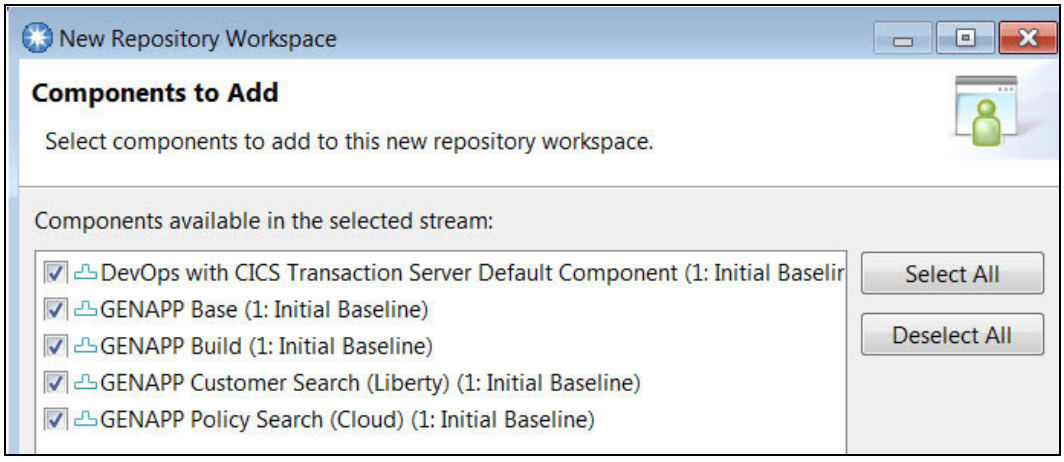


Figure 5-60 Components to add to the repository

The new BUILD: GENAPP CICS Policy build workspace is now selected as the repository workspace for the build definition, as shown in Figure 5-61.

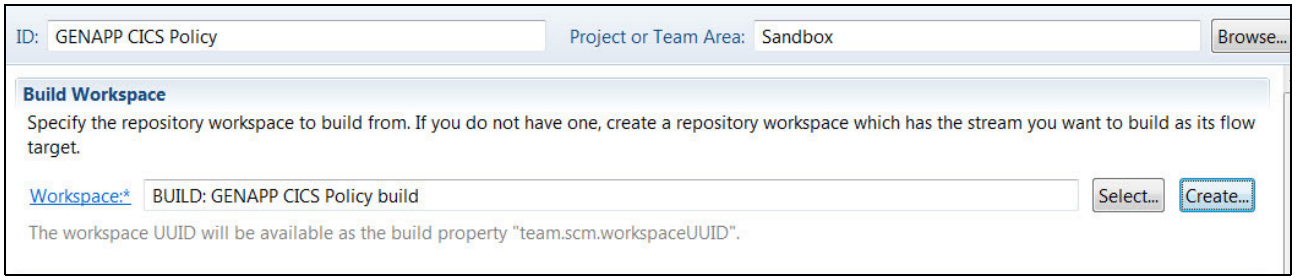


Figure 5-61 Build workspace created

The next step is to specify the load options, and the first item to include is the load directory. The load directory is the folder where the files from the workspace repository are loaded for the build engine. The load directory in this scenario is a property that already is added.

19. On the GENAPP CICS Policy Build Definition window, enter ${source.dir} in the Load directory, as shown in Figure 5-62.

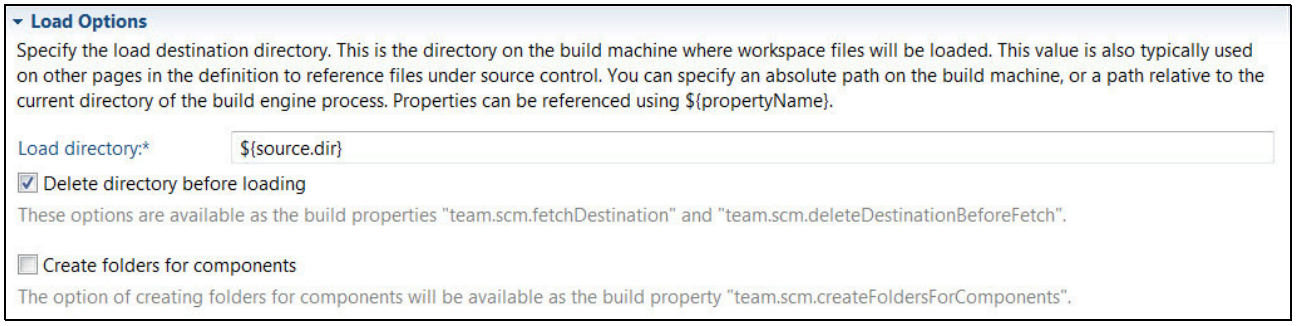


Figure 5-62 Load directory name

You can have many components in the stream, but there might be cases where you might not want to build all of them in one build. So, you can exclude some components from the build definition.

20. Click **Select** to choose the components to exclude (Figure 5-63).

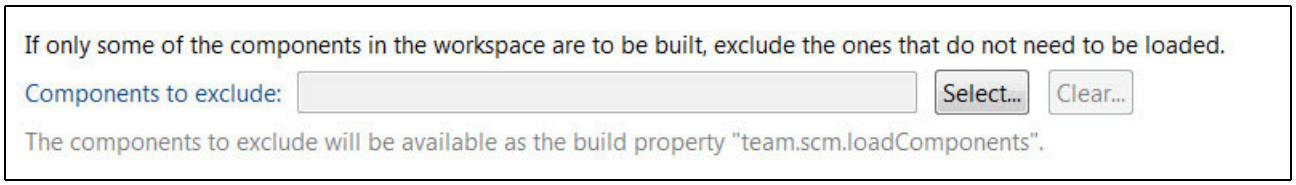


Figure 5-63 Components to exclude

21. From the list of Components to exclude, select those components that you want to exclude in this build definition. Components that are not selected are included in the build. In our scenario, build the GENAPP Base and GENAPP Build components, as shown in Figure 5-64 on page 111.

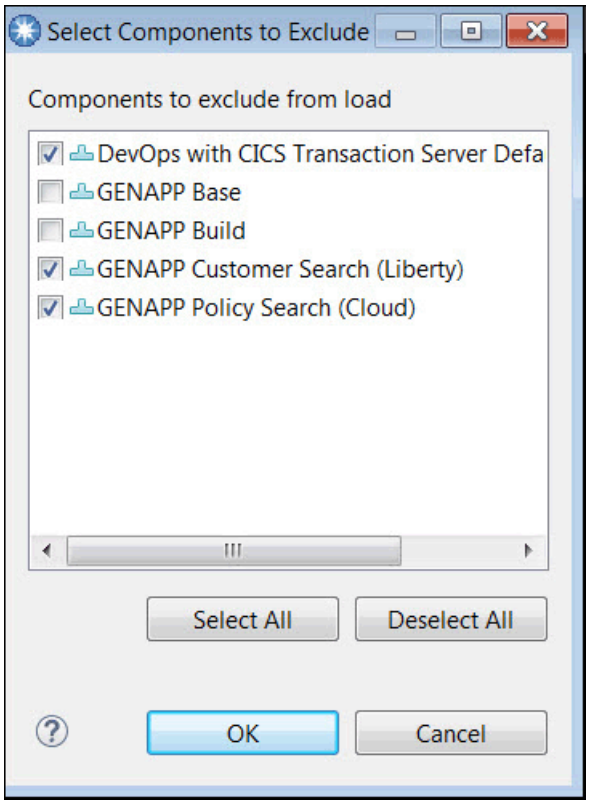


Figure 5-64 Components to exclude from the build

22. Make sure to select both Accept Options check boxes, as shown in Figure 5-65. This selection allows the acceptance of changes from the stream into the build workspace and runs the build only when there are changes in the stream.

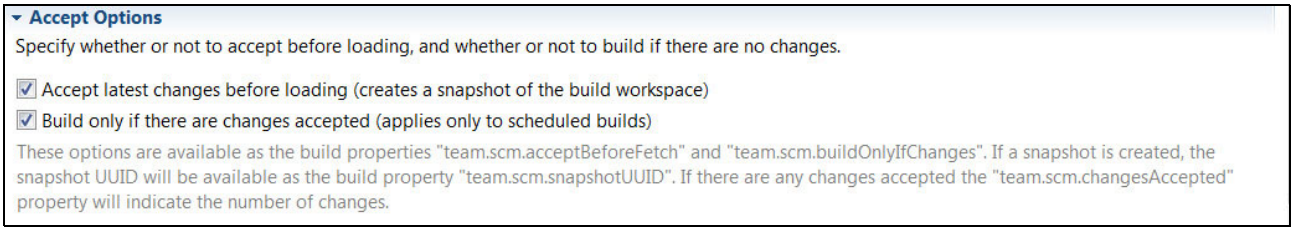


Figure 5-65 Accept options to build only if there are changes

23. Click the **Ant** tab in the build definition to specify what must be run for the build. In this case, we run the CICS build toolkit inside the Ant script. Enter `source/CICS BT/build.xml` as the Build file, as shown in Figure 5-66. You can use specify shell scripts or JCL.

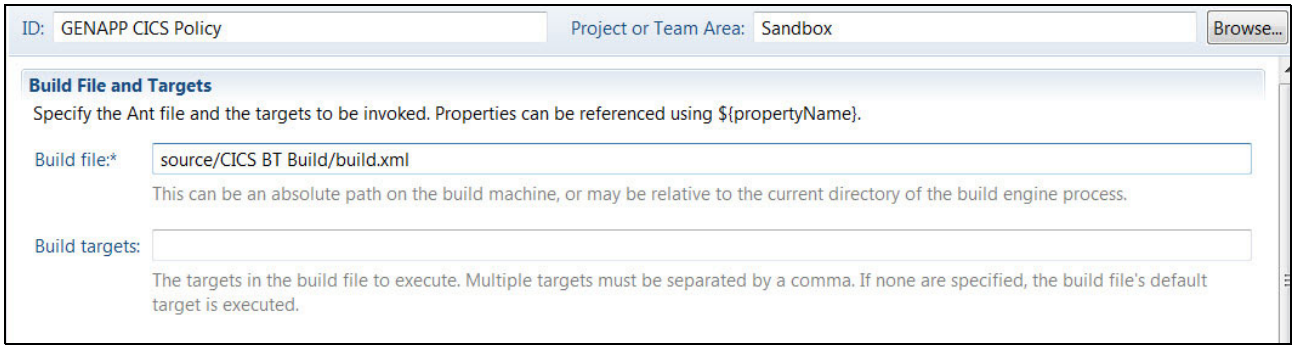


Figure 5-66 Build file location

Example 5-3 shows the `build.xml` script that we are using for the build, which starts the CICS build toolkit.

Example 5-3 Build.xml

```
<project xmlns:contrib="antlib:net.sf.antcontrib" default="cicsbt">

   <!-- Define antcontrib extension -->
   <taskdef resource="net/sf/antcontrib/antlib.xml"
uri="antlib:net.sf.antcontrib" classpath="ant-contrib.jar" />

   <property name="cicsbt_workspace" value="cicsbt_workspace" />
   <property name="userId" value="build_user" />
   <property name="passwordFile" value="${user.dir}/../jbe_password_file.txt"
/>

   <target name="cicsbt">
      <echo level="info">Building with the CICS build toolkit.</echo>
      <echo level="info">source.dir is ${source.dir}.</echo>
      <echo level="info">user.dir is ${user.dir}.</echo>
      <echo level="info">build.output.directory is
${build.output.directory}.</echo>

      <startJazzBuildActivity activityIdProperty="clearingWorkspaceActivityId"
activityLabel="Clearing workspace" />
      <delete includeEmptyDirs="true" failonerror="false">
         <fileset dir="${user.dir}/${cicsbt_workspace}" />
      </delete>
      <completeJazzBuildActivity activityId="${clearingWorkspaceActivityId}" />

      <startJazzBuildActivity activityIdProperty="runningCicsbtActivityId"
activityLabel="Building with CICS build toolkit" />
      <exec executable="${cicsbt.install.dir}/${cicsbt.command}"
failonerror="false" resultproperty="return.code" dir="${user.dir}">
         <env key="IBM_JAVA_OPTIONS"
value="-Xshareclasses:name=cicsbt,groupAccess" />
         <arg line="--input ${source.dir}/* --build ${bundle.name} --output
${build.output.directory} --target ${target.platform} --workspace
${cicsbt_workspace} --verbose" />
```

```
      </exec>
      <fail message="cicsbt failed">
         <condition>
            <contrib:isgreaterthan arg1="${return.code}" arg2="2" />
         </condition>
      </fail>
      <completeJazzBuildActivity activityId="${runningCicsbtActivityId}" />
   </target>


   <taskdef name="startBuildActivity"
classname="com.ibm.team.build.ant.task.StartBuildActivityTask" />
   <taskdef name="completeBuildActivity"
classname="com.ibm.team.build.ant.task.CompleteBuildActivityTask" />

   <macrodef name="startJazzBuildActivity">
      <attribute name="activityIdProperty" />
      <attribute name="activityLabel" />
      <sequential>
         <startBuildActivity label="@{activityLabel}"
activityIdProperty="@{activityIdProperty}" buildResultUUID="${buildResultUUID}"
repositoryAddress="${repositoryAddress}" userId="${userId}"
passwordFile="${passwordFile}" />
      </sequential>
   </macrodef>

   <macrodef name="completeJazzBuildActivity">
      <attribute name="activityId" />
      <sequential>
         <completeBuildActivity activityId="@{activityId}"
buildResultUUID="${buildResultUUID}" repositoryAddress="${repositoryAddress}"
userId="${userId}" passwordFile="${passwordFile}" />
      </sequential>
   </macrodef>


</project>
```

24. Switch to the Post-build Deploy tab to set up automation of publishing artifacts to IBM UrbanCode Deploy CodeStation, as shown in Figure 5-67.

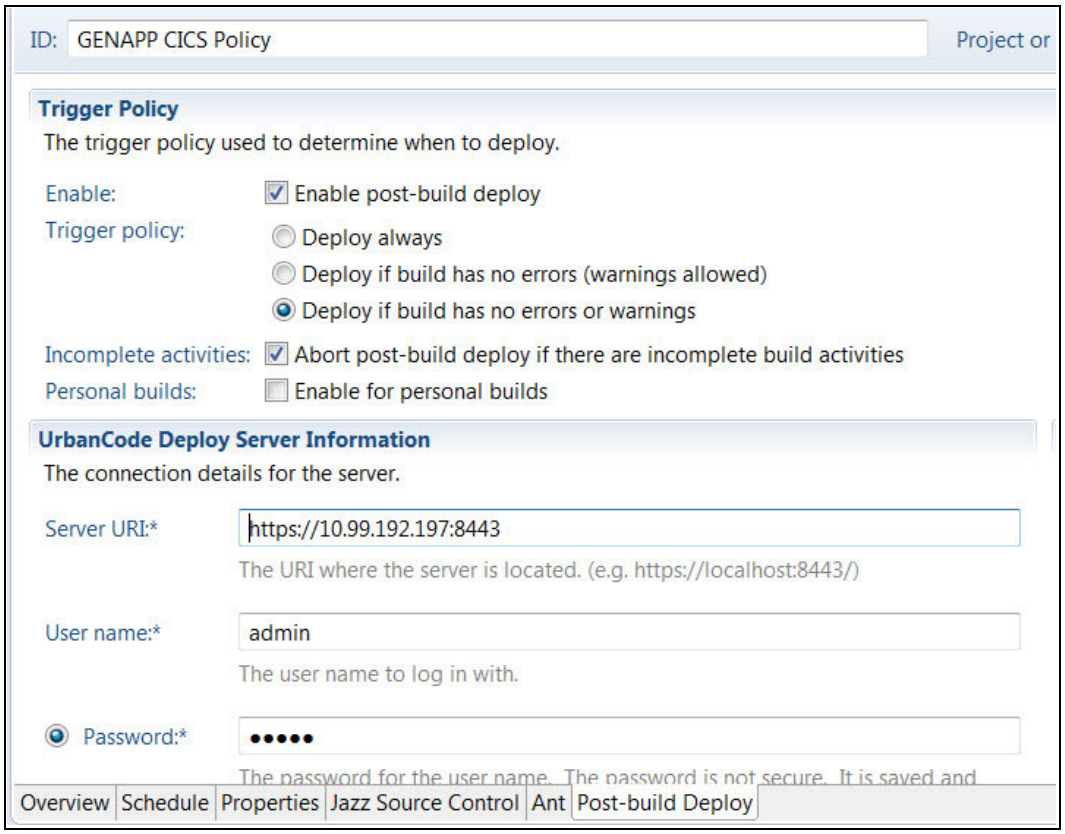


Figure 5-67 IBM UrbanCode Deploy Server Information

25. Next, you must specify the information of the artifact that must be published. You must specify an existing IBM UrbanCode Deploy Component name, which you created for the CICS Policy. For this example, we use GENAPP CICS Policy as the name of the component and ${buildLabel} for the version because we want it to be dynamically created. We also enter the Base directory from which to publish the files. The base directory is the output of the CICS build toolkit run. We specify all files to be included by using **/*, as shown in Figure 5-68 on page 115.

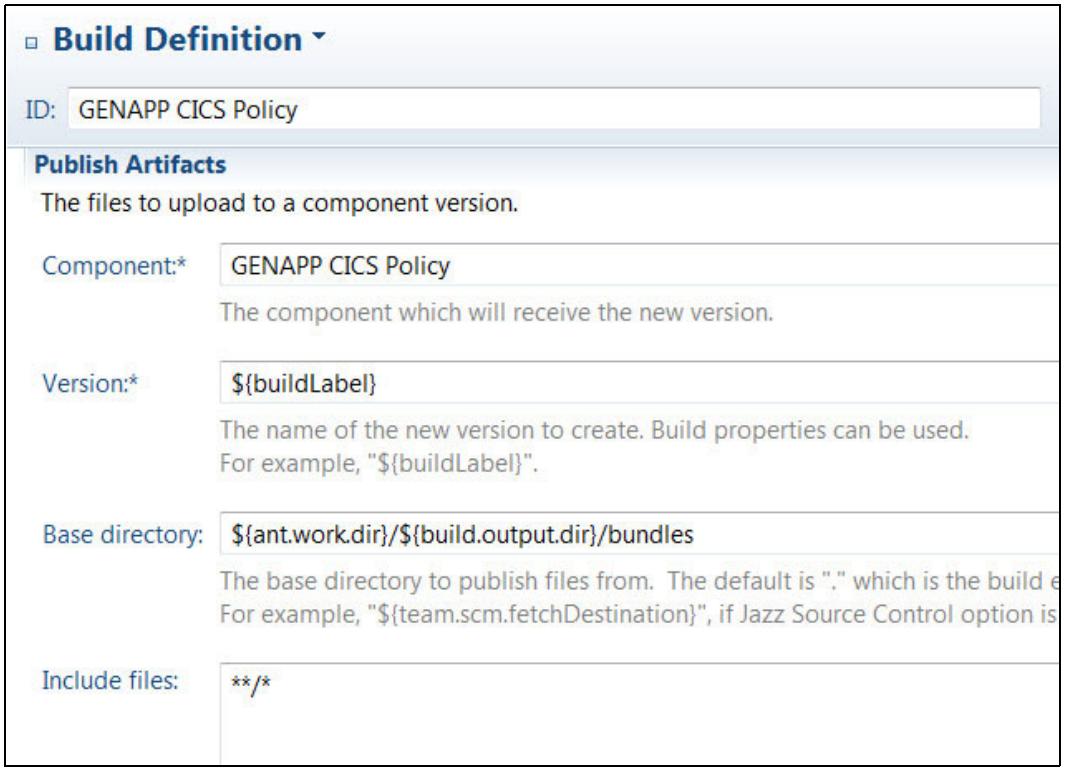


Figure 5-68 Publish Artifacts

5.3.2 Creating a GENAPP CICS Policy component

Creating a policy component must be done before running the build process because the build process creates a new version of the component, which must exist first.

The process for creating a policy component is similar to the steps in 5.2.5, “Defining resources and adding a component to them” on page 88, where you created a component for GENAPP Base. Complete the following steps:

1. Click **Components** → **Create Component**, and create a component that is called GENAPP CICS Policy. Because the policy is a bundle that is on UNIX System Services, the component type should be set to **Standard**.
2. Leave the Source Configuration Type blank because we use the CICS build toolkit post-deployment to put the artifacts into IBM UrbanCode Deploy CodeStation.
3. Click **Save**.

Refer to Figure 5-69.

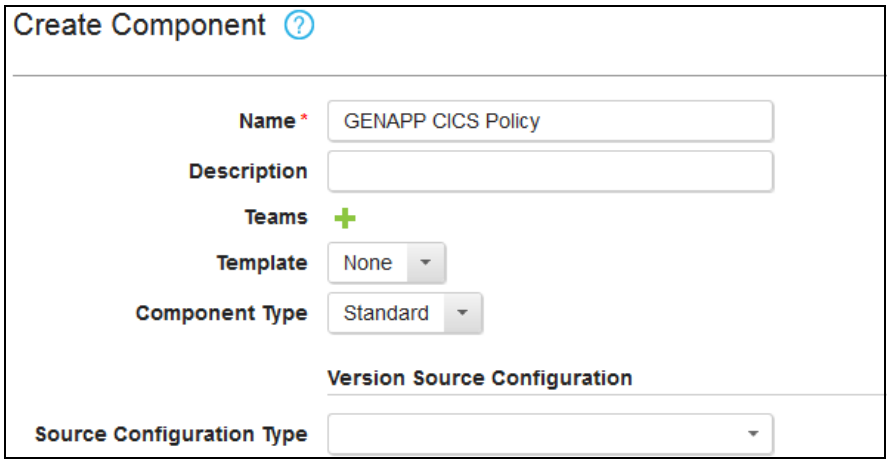


Figure 5-69 Create the GENAPP policy component

5.3.3 Mapping the component to resources

In order for a process to be deployed, it must be mapped to a resource, just like you mapped GENAPP Base to the CICSDA01 region.

Add the GENAPP CICS Policy component to the same CICS region CICSDA01 by completing the following steps:

1. Click **Resources** → **Resource Tree**.
2. Hover your cursor over CICSDA01, and from the Actions menu, select **Add Component**, as shown in Figure 5-70.

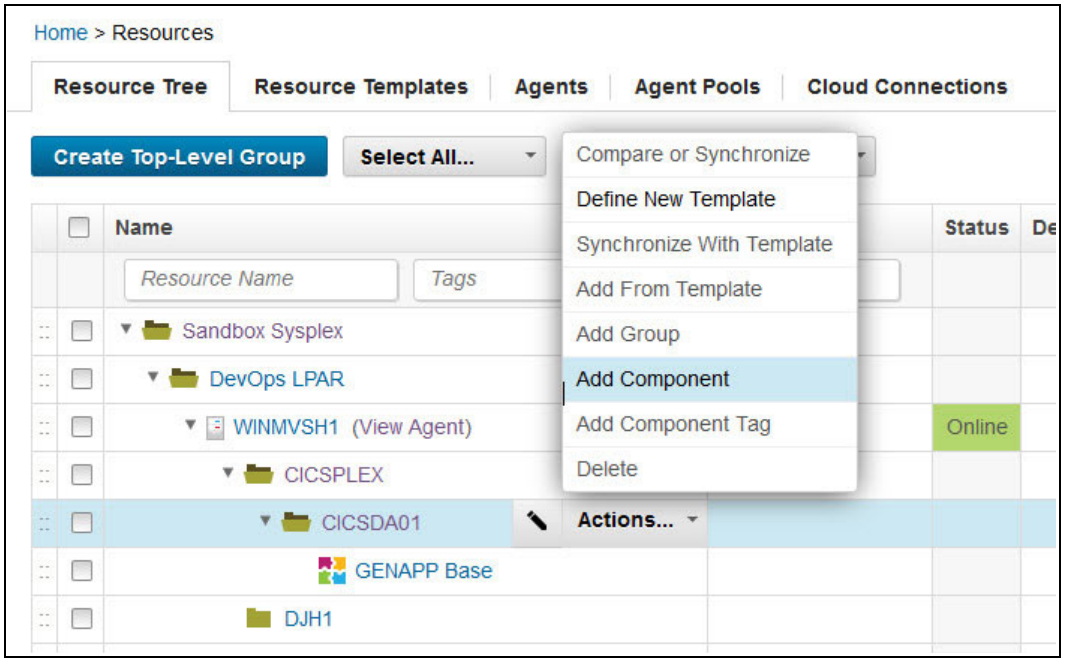


Figure 5-70 Map the GENAPP CICS Policy component to resources

3. Select the GENAPP CICS policy from the Component menu.
4. Click **Save**.

5.3.4 Defining the component process for the GENAPP CICS policy

You must define a deployment process for the GENAPP CICS Policy by completing the following steps:

1. Click **Components** → **GENAPP CICS Policy** → **Processes** (for the current component), and select **Create Process**.
2. Create a deployment process.
3. Select the deployment process that you created and add the necessary steps (see Figure 5-71).

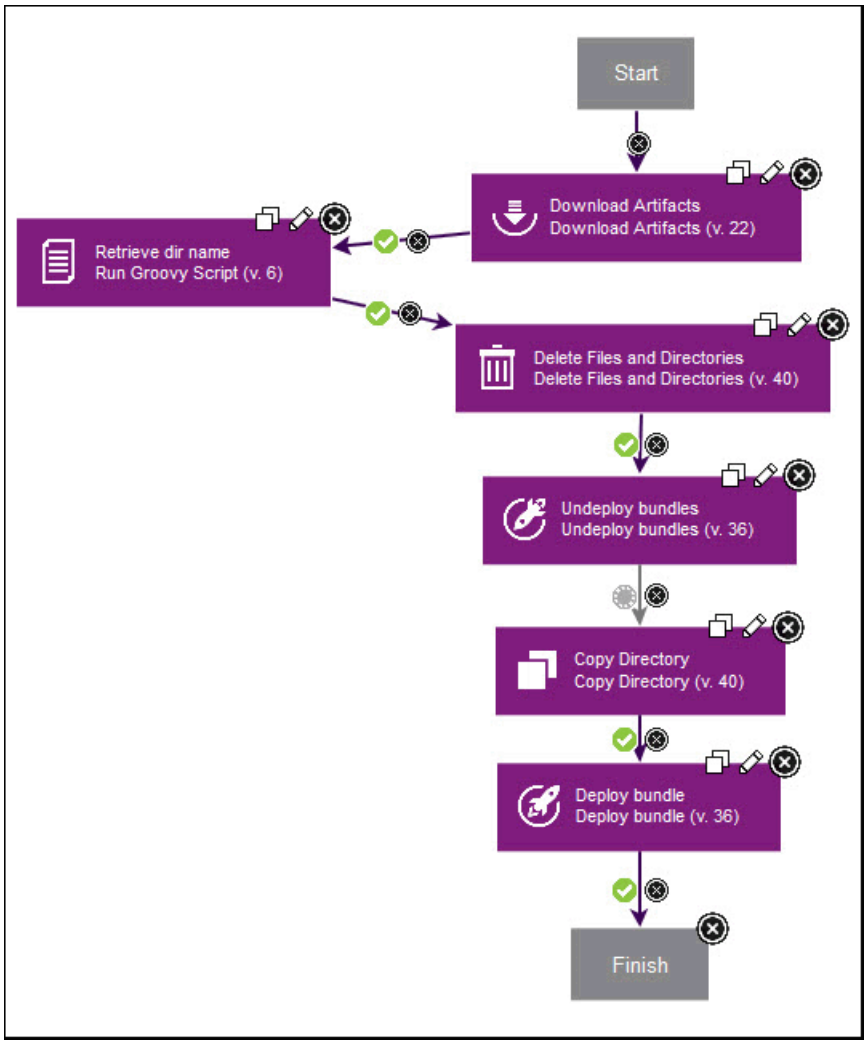


Figure 5-71 GENAPP CICS Policy component process

Because the build bundle already is in the IBM UrbanCode Deploy CodeStation, start with a Download Artifacts step to download the bundle from the code station to a working directory.

The Delete Files and Directories step is used to delete any prior files or directories in the CICS bundle directory with the same name to avoid duplication or overriding when you copy the artifact onto the CICS bundle directory. This step has the following substeps:

1. Retrieve the bundle sub directory by using a Groovy script.
2. Delete the directory with same name in the CICS bundle directory and undeploy the same bundle in CICS if it is installed. Notice the arrow after the undeploy step is marked as “Always run”, so the process continues regardless of the outcome of the undeploy bundle step.
3. Copy the bundle from the working directory onto the CICS bundle directory and perform the actual deployment of the bundle.

The following sections look into the details of each step.

Downloading Artifacts step

As the built artifact already is in the IBM UrbanCode Deploy CodeStation (it was placed there by the post-build deployment process), use a Download Artifacts step to retrieve it and place it in a working directory (Figure 5-72) by using the following values:

- Directory offset = `./built/bundles`: Copy the bundle from the code station into the `/built/bundles` directory under the current working directory.
- Include = `**/*`: Include everything that matches the selected component and version when this process runs (when you run this process, you must select a component and version).

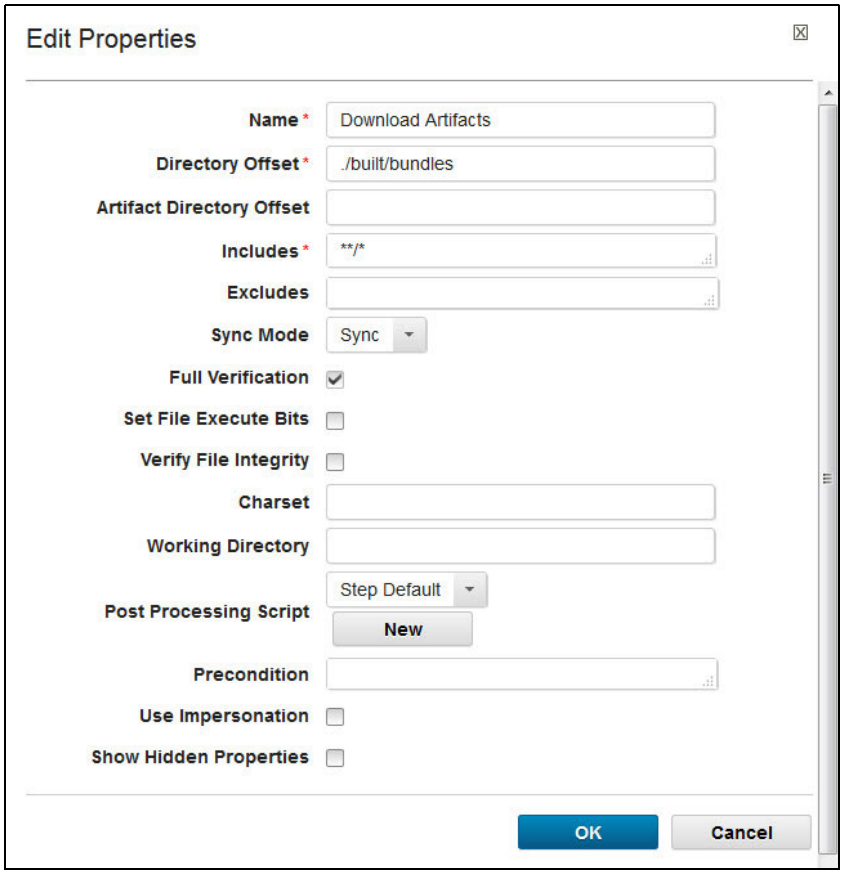


Figure 5-72 Download Artifacts step properties

Retrieving the directory name

This is a preparation step to retrieve the bundle subdirectory each time this process runs. This directory is used in the subsequent steps to construct the complete CICS bundles directory where this bundle is, and in the delete and copy steps.

At the time of writing, there was not a step to perform this particular task, so we developed a Groovy script to do it. IBM UrbanCode Deploy provides several scripting steps such as Groovy and Shell to enable users to write their own steps for specific tasks.

Complete the following tasks:

1. Enter “script” into the Step Palette search box, as shown in Figure 5-73.

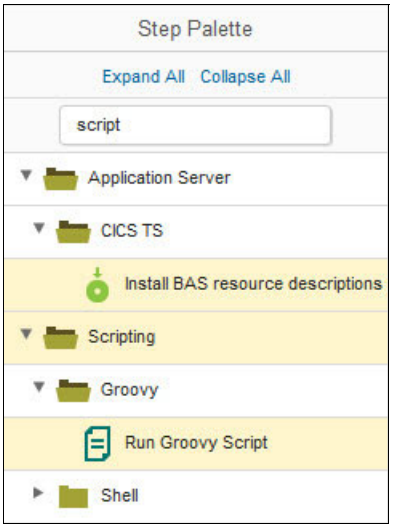


Figure 5-73 Find the script step in the Step Palette

2. Select **Run Groovy Script** and drag it into the main pane.
3. Set the Name as **Retrieve dir name**.
4. Enter the Groovy script, as shown in Example 5-4.

Example 5-4 Groovy code for retrieving the directory

```
   def parentDirectory = "./built/bundles"
   println "Getting first directory inside " + new
File(parentDirectory).getPath()
   def dirs = new File(parentDirectory).listFiles(new FileFilter() {
   public boolean accept(File f) {
          return f.isDirectory()
   }
   })
   def firstDir = dirs.first()
   println firstDir.getPath()
   outProps.setProperty("directory", firstDir.getName())
   println "Done"
```

Figure 5-74 shows the other values for this step.

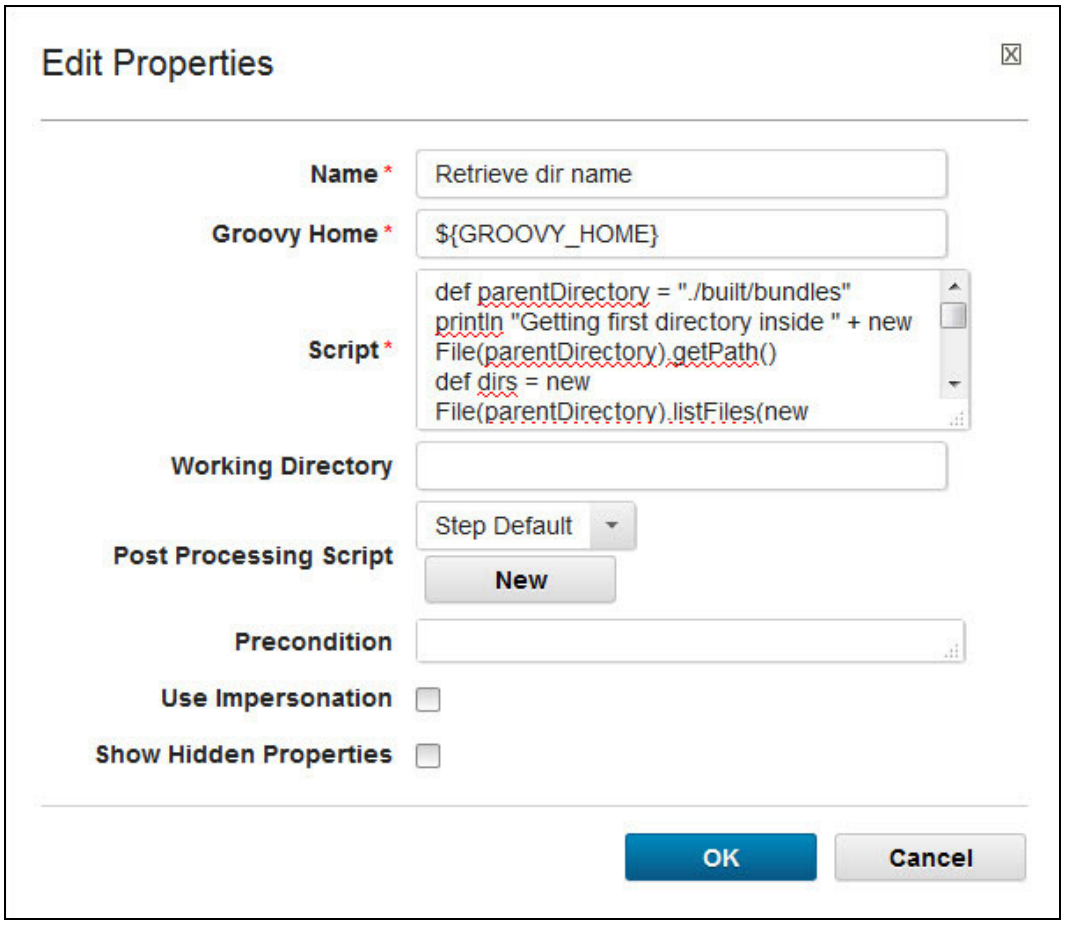


Figure 5-74 Retrieve directory name step properties

The output of this step in this example is a property that is called “directory” with the following value:

`com.ibm.genapp.cpupolicy.bundle_1.0.0`

This is the bundle name that is used in the next build step.

Delete Files and Directories step

This step deletes any files or directories in the destination CICS Bundle directory with the following name before you copy the newly built version to the directory:

`com.ibm.genapp.cpupolicy.bundle_1.0.0`

Create a Delete files and Directories step by using the properties that are listed in Table 5-5.

Table 5-5 Properties explanation for Delete step

Property name	Value	Note
Base Directory*	`${p:app.env.directory}bundles/`	This is the CICS Bundle directory that is used. `App.env.directory` is the directory for this application and is a property that is defined to the development environment.
Include*	`${p:Retrieve dir name/directory}`	This property maps to the output of the previous step.

Figure 5-75 shows the other values for this step.

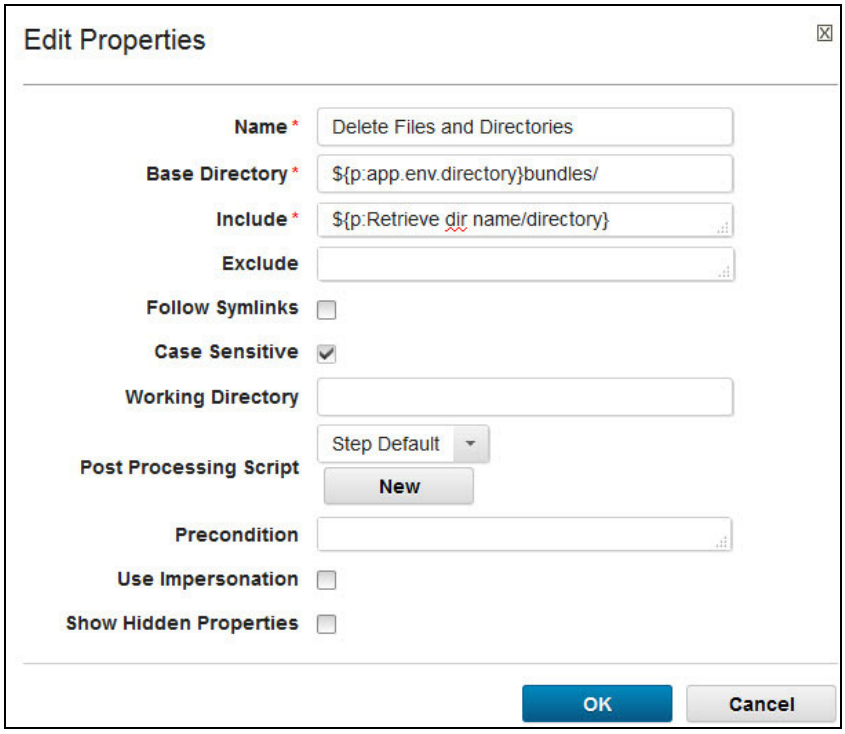


Figure 5-75 Delete Files and Directories step properties

Undeploy Bundles step

This step undeploys any earlier version of the same bundle. The properties that are used in this example are shown in Figure 5-76.

In this example, we use CPUPOLCY and CIPOLICY to name the bundle and resource group in CICS.

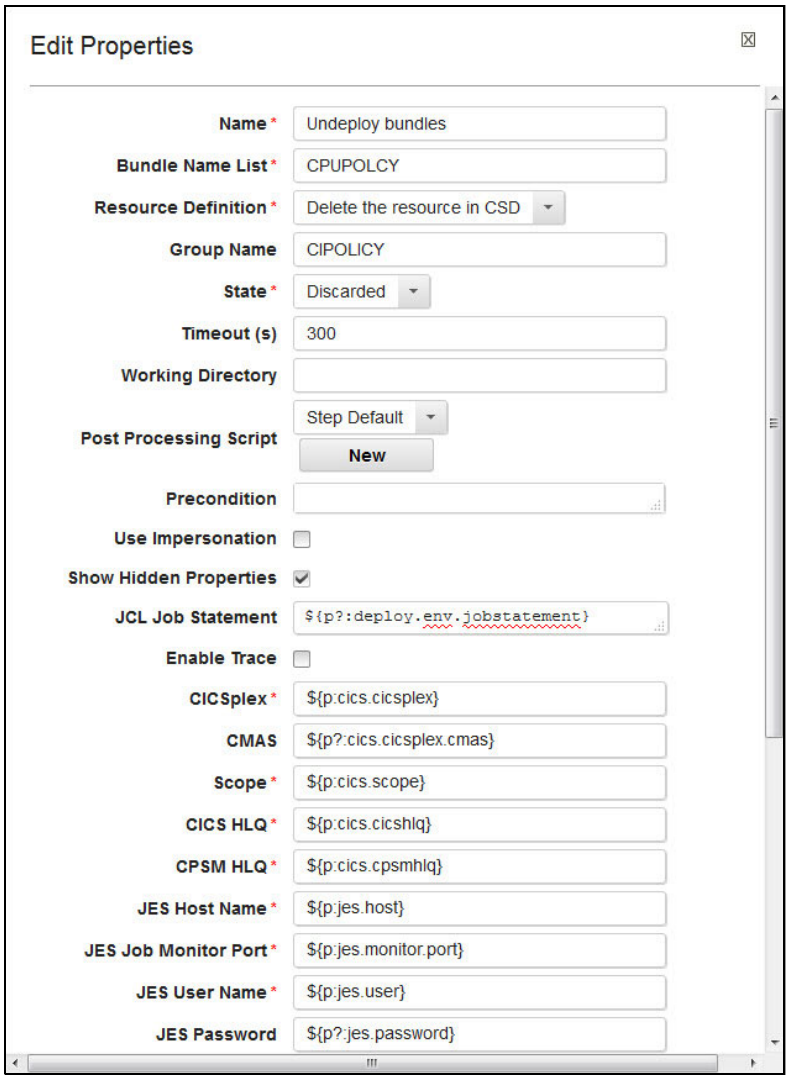


Figure 5-76 Undeploy Bundles step properties

Note the list of properties at the bottom. These are usually properties that are not specific to this component, but apply to a wider scope. So, they are specified in contexts where appropriate so that users do not need to respecify them in every component. However, all of the properties must be resolved successfully for the step to run.

In this particular scenario, most of the properties are CICS specific and are defined on the CICSPlex resource level, as shown in Figure 5-77 on page 123.

The JES-related properties can be defined on an LPAR resource level, where it is more appropriate. For illustration purposes, we include them on CICSPlex properties.

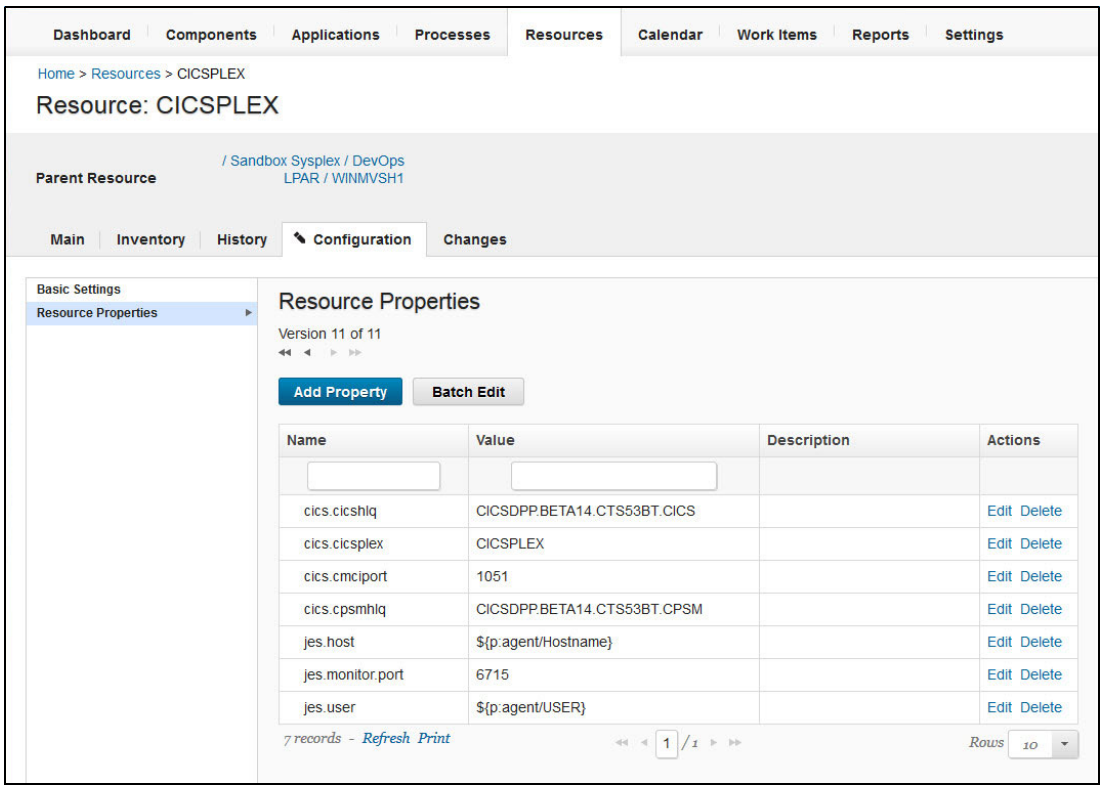


Figure 5-77 Resource properties that are defined on CICSPlex

Copy Directory step

You are ready to copy the bundle from the working directory to the CICS Bundle directory that is used for this application, as shown in Figure 5-78.

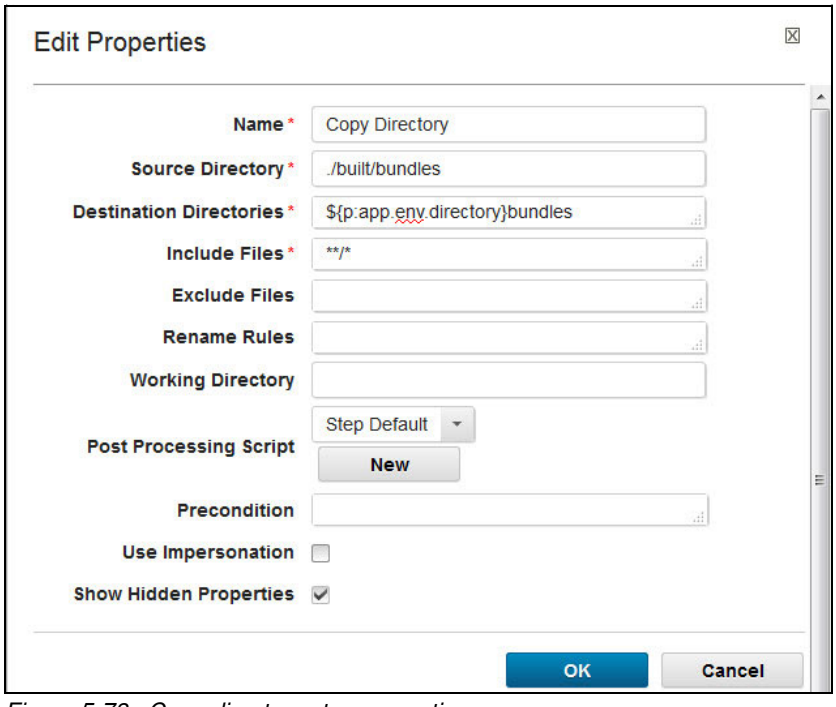


Figure 5-78 Copy directory step properties

Deploy Bundles step

This is the final step to install the bundle in CICS. This step creates CSD definitions under the CIPOLICY group and installs the bundle that is named com.ibm.genapp.cpupolicy.bundle with a status of ENABLED. The Bundle Directory is specified as follows, pointing to the exact bundled that was copied:

```
${p:app.env.directory}bundles/${p:Retrieve dir name/directory}
```

Figure 5-79 shows the other values for this step.

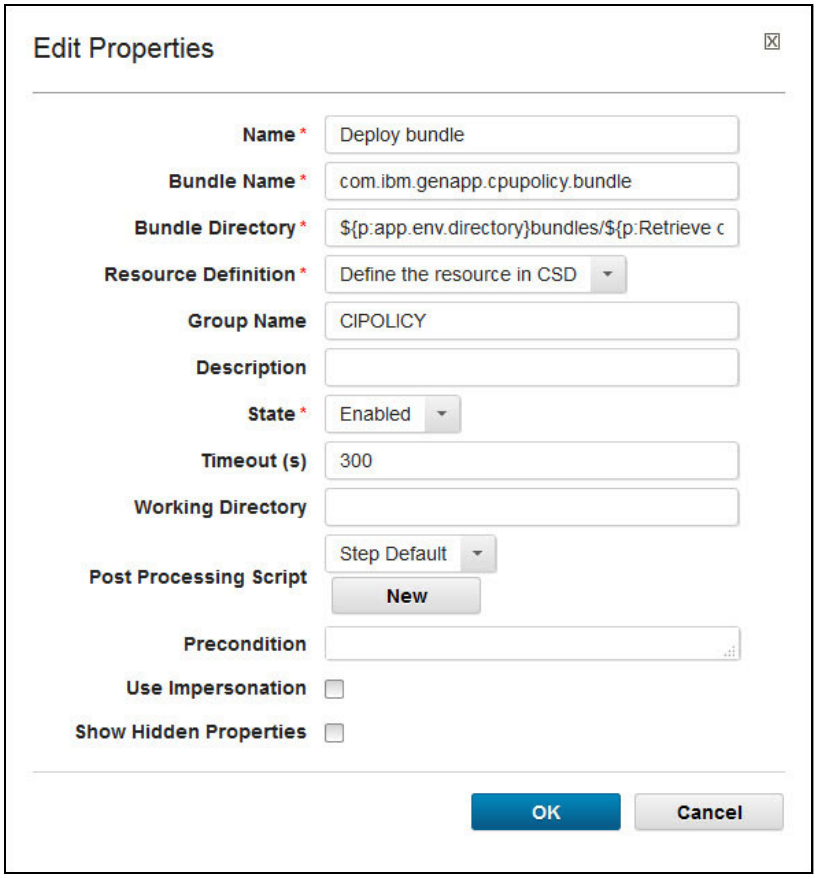


Figure 5-79 Deploy Bundles step properties

5.4 Deploying an application by using the GENAPP Base and CICS Policy

Now that you have both the GENAPP Base and GENAPP CICS Policy components, the next step is to deploy both of them in one application process. This process shows that applications can be used as a higher-level control point for their components.

5.4.1 Adding a GENAPP Policy component to a GENAPP application

To add a GENAPP Policy component to a GENAPP application, complete the following steps:

1. Click **Applications** → **GENAPP Application** → **Components**.
2. Click **Add Component** and select **GENAPP CICS Policy** from the Select a Component menu (see Figure 5-80).

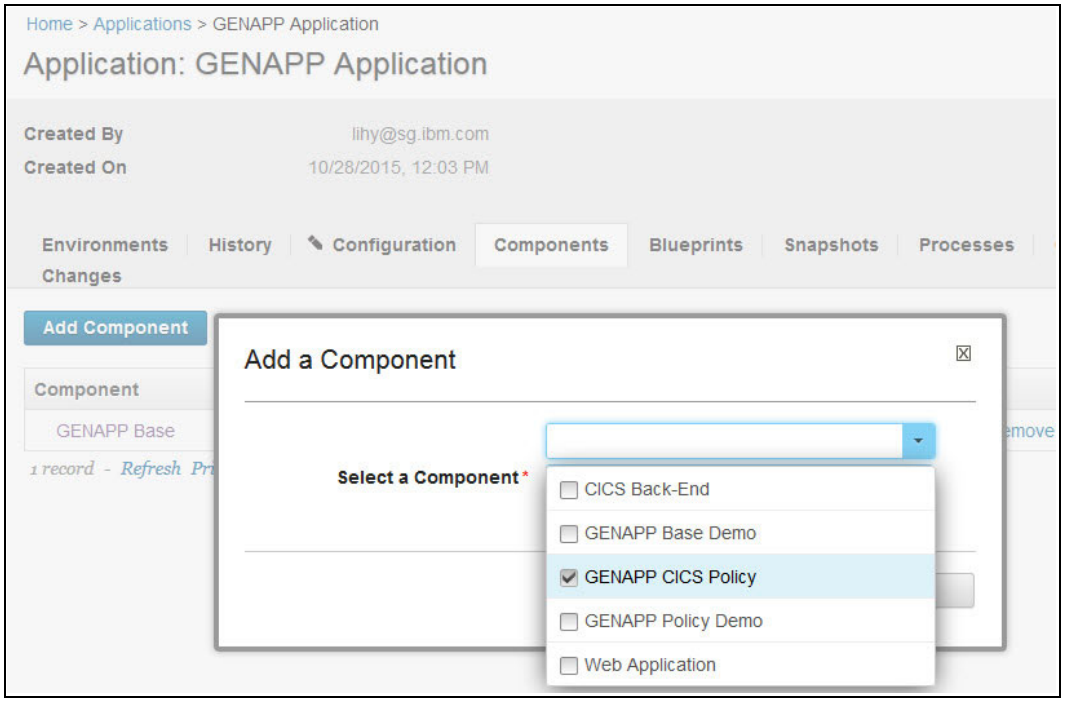


Figure 5-80 Add the GENAPP CICS Policy component to GENAPP Application

5.4.2 Defining an application process to deploy both components

Create another application deployment process by using two steps to deploy each component by running the following steps:

1. Click **GENAPP Application** → **Processes** → **Deploy Application**.
2. Click **Duplicate** next to the Deploy Application process.
3. Click **Deploy Application (copy)** to edit it.
4. Add another Install Component step to start the deployment process for GENAPP CICS Policy. When you select **GENAPP CICS Policy** from the Component menu, the Name field is automatically updated.

For more information, see Figure 5-81.

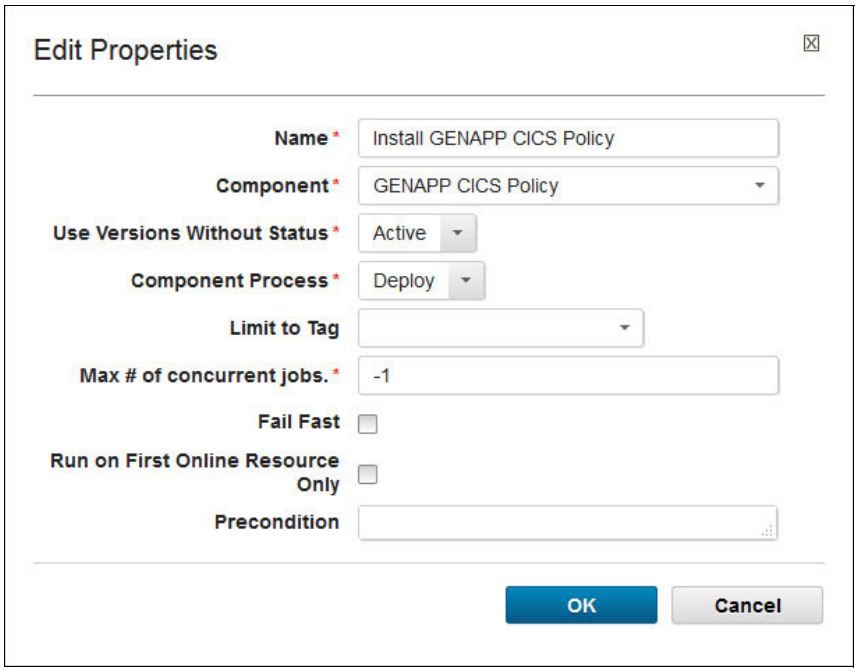


Figure 5-81 Add Deploy step for GENAPP CICS Policy

5. Click **OK** to save the step.
6. Reconnect the steps to make the two steps parallel. In this particular example, these two steps can be defined as either sequential or parallel steps. There is no dependency between the steps. You should get an application that looks like what is shown in Figure 5-82.

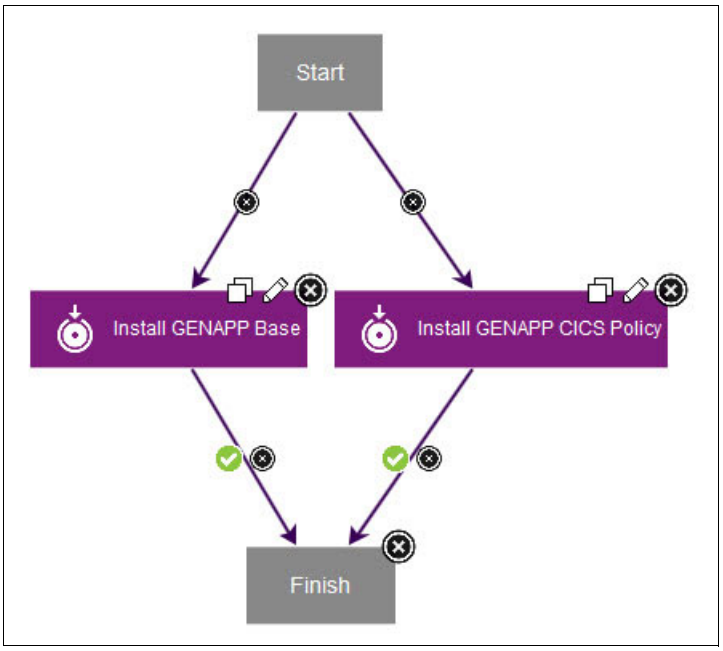


Figure 5-82 Application deployment process with both components

7. Click **Save**.

5.5 Running the process

Now you run the process. Select the new process that you created. Because you have two component steps in the process, select a version of each component to be deployed, as shown in Figure 5-83. You can choose a version for each component to be deployed in this environment.

Complete the following steps:

1. Click **Applications** → **GENAPP Application** → **Environments**.
2. Click the Play button to Request Process, which is marked by a circle with an arrow inside.
3. In the Run Process on Development window, select **Deploy Application (copy)** from the Process menu.
4. Click **Choose Versions**, and add the latest version to be deployed.

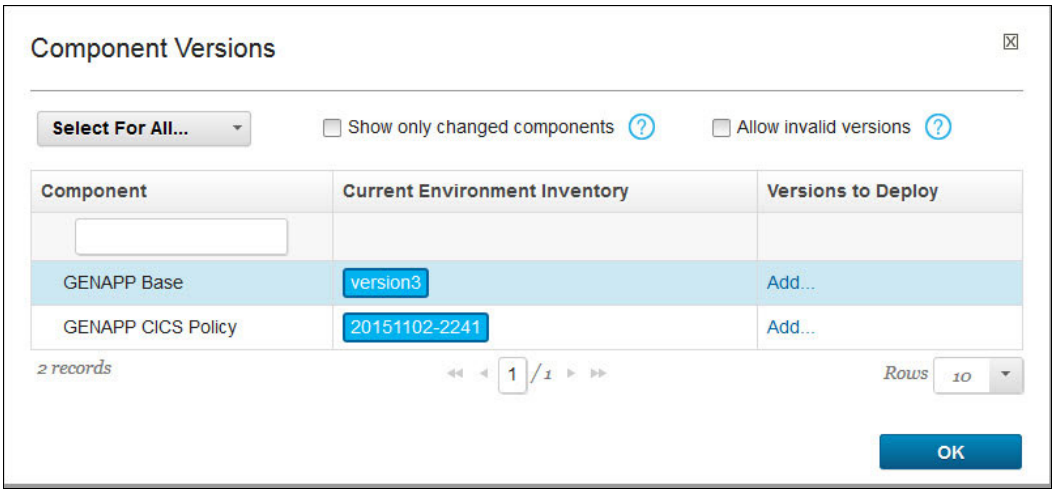


Figure 5-83 Select the component versions to deploy

5. Click **OK** to save and submit the process.

Note: If there is no new version to be deployed for one of the components, then it can be left blank. The process runs only the step that has a version selected.

The first scenario for implementing GENAPP with a CICS policy is now complete.

6

Applying DevOps to Java applications

This chapter demonstrates how the new DevOps tools in CICS Transaction Server V5.3 can be used to create reliable and repeatable automated build and deployment processes for web applications within Liberty JVM servers on CICS Transaction Server.

This chapter covers the following topics:

6.1 Java application scenario

The application that we use in this scenario is the same GENAPP application that is used throughout this book, with an extension that uses the WebSphere Application Server Liberty profile in CICS. The purpose of using the Liberty extension for GENAPP is to demonstrate the ability to use Java and web technologies along with traditional COBOL code. You can download the GENAPP extension from IBM Support at this website:

`http://www.ibm.com/support/docview.wss?uid=swg24031760`

Here are the DevOps tools that you use in this chapter:

- Integrated developer environment (IDE): CICS Explorer
- Source code management (SCM) system: Rational Team Concert (for the web application source code)
- Build: IBM CICS Transaction Server build toolkit (CICS build toolkit)
- Deployment: DFHDPLOY utility

The high-level design of the build and deployment processes is shown in Figure 6-1.

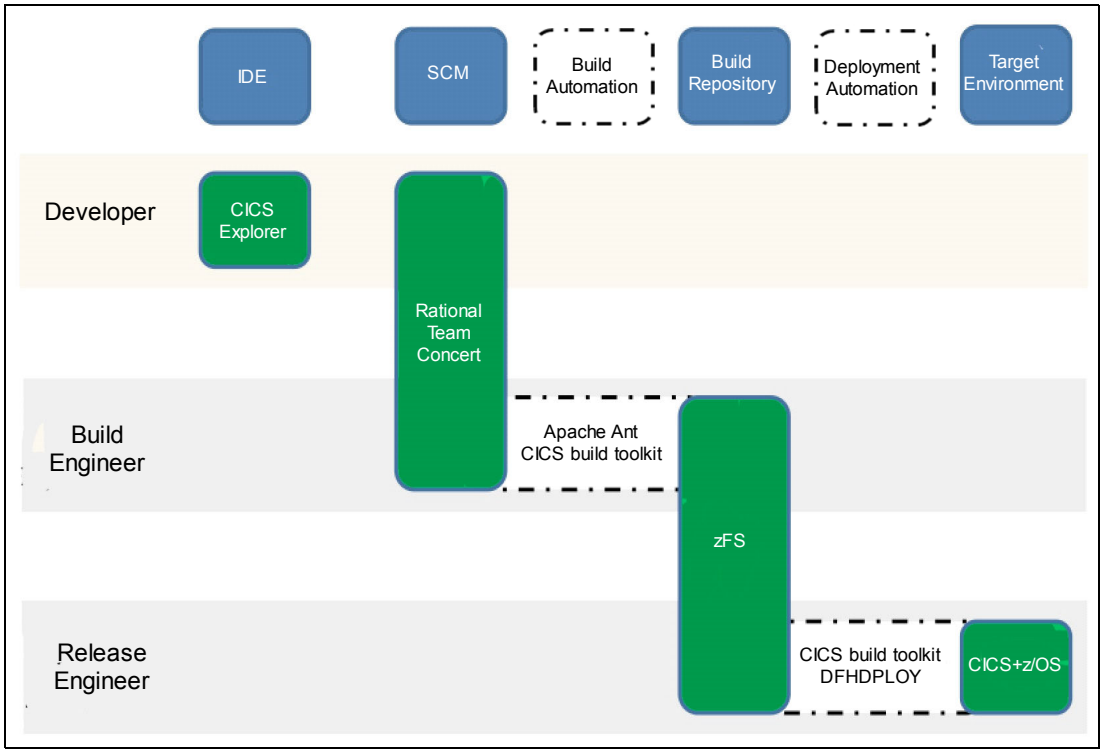


Figure 6-1 High-level design of the build and deployment processes for Java applications

The build process (see the left side of Figure 6-2 on page 131) uses the SCM build engine and the CICS build toolkit's Build command to build the bundle directly within the build repository of zFS. The deployment process (see the right side of Figure 6-2 on page 131) uses the CICS build toolkit's Resolve command to resolve different variables in the resource definitions (for example, the JVM server name attribute), and then uses the DFHDPLOY utility to deploy the bundle in CICS.

In this scenario, we use zFS as a readily available area in which to store built artifacts. However, because it is a simple file system, zFS does not offer features such as versioning, history, or atomic operations, which can make it harder to manage and result in additional work. Therefore, in the long term it might be more appropriate to use a purpose-built build repository such as Rational Asset Manager or Apache Archival. You can also use the IBM UrbanCode Deploy built-in build repository, which is called IBM UrbanCode Deploy CodeStation (in which case, it might be easier to use the IBM UrbanCode Deploy deployment processes rather than DFHDPLOY). If you use a purpose-built build repository, the process of adding and extracting artifacts is different from what we show in this chapter (specifically, you use the repository's APIs rather than standard file system operations).

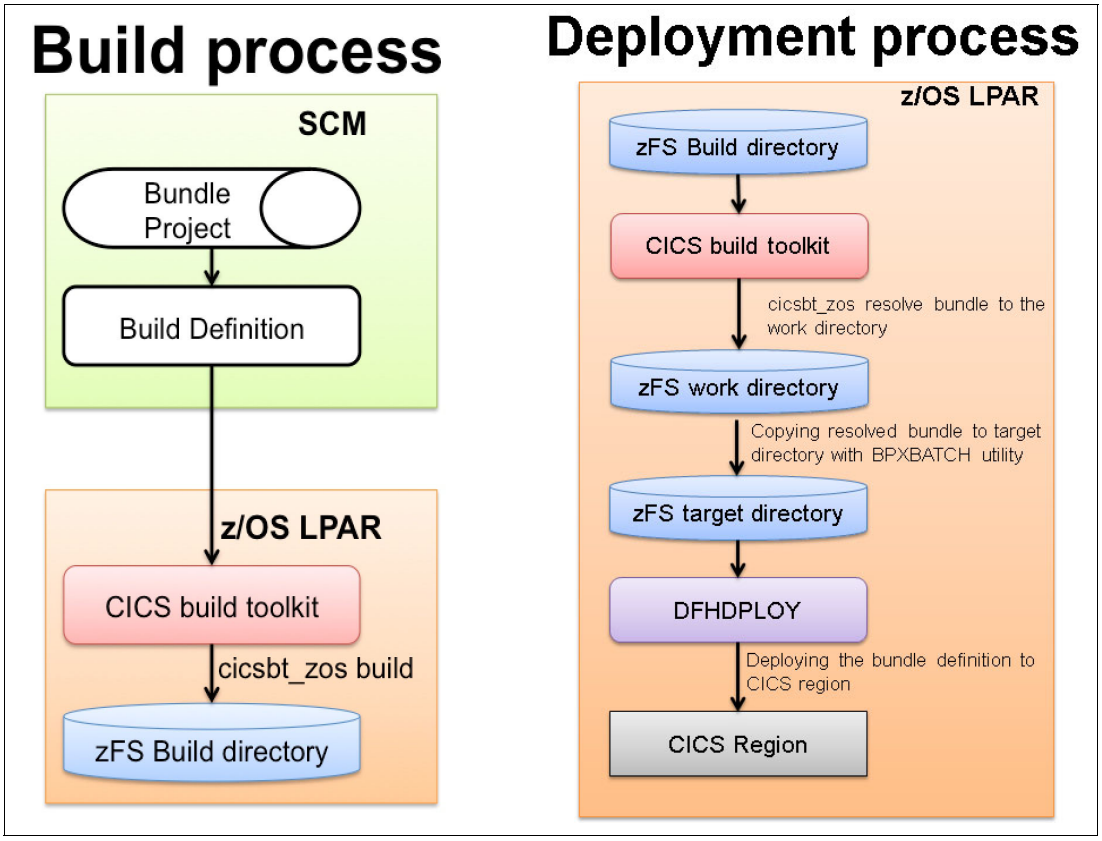


Figure 6-2 Build and deployment architecture for Java application development

6.2 General application description

The GENAPP Liberty Profile extension consists of three components:

- A web user interface (implemented with JSP technology)
- A servlet that is written in Java
- A COMMAREA wrapper that is written in Java

The web user interface is a JSP file that consists of one input field and an output table. In the input field, the user should enter a customer number. On submit, an HTTP POST request is sent with the customer number as a parameter. An output table is provided to display the search results from the HTTP response.

The servlet is a Java program that listens on the HTTP POST URL and then extracts the customer number from the request. The customer number is input into a COMMAREA that is sent as a JCICS **`link()`** parameter to a COBOL program of the GENAPP Base application. The program name in this case is LGICUS01.

The COMMAREA wrapper is a Java class that provides methods to interact with a byte array that represents the COMMAREA that is defined in the LGICUS01 copybook. The wrapper also handles data conversion between UTF-8 and EBCDIC.

Figure 6-3 is a diagram showing all of components that are used in the applications that are described in this chapter.

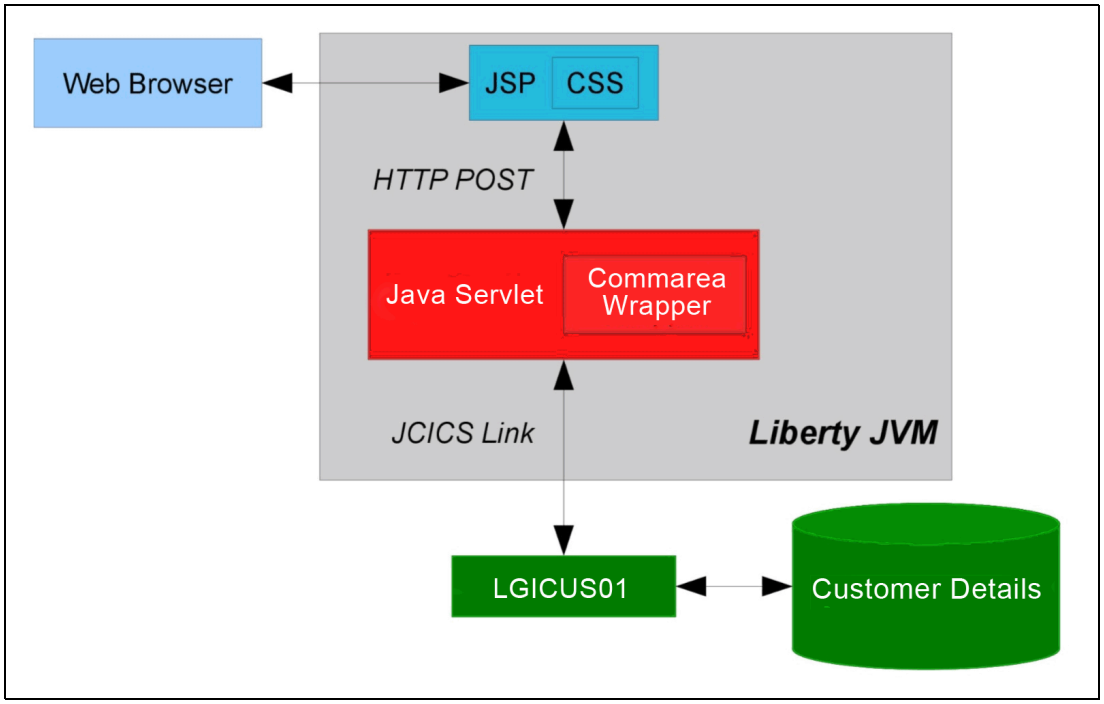


Figure 6-3 Liberty Extension for GENAPP application

6.3 JVM server configuration

Before you can deploy the web application, you must first define a JVM server in CICS, and a corresponding JVM profile. To create the JVM profile, you can use a modified version of the DFHWLP profile (see Example 6-1, which is provided with CICS Transaction Server (in `<CICS_ROOT_DIR>/JVMProfiles/`). This modified version is the version that we use to create the JVM server running in CICS Transaction Server.

Example 6-1 DFHWLP profile

```
################################################################################
# JVM profile: DEVOPSLP                                                        #
#              DEVOPS Liberty Profile                                          #
################################################################################
#                                                                              #
# Licensed Materials - Property of IBM                                         #
#                                                                              #
# 5655-Y04  CICS Transaction Server                                            #
```

```
#                                                                     #
# (c) Copyright IBM Corporation 2012, 2015 All Rights Reserved        #
# US Government Users Restricted Rights - Use, duplication            #
# or disclosure restricted by GSA ADP Schedule Contract               #
# with IBM Corporation                                                #
#                                                                     #
#######################################################################
#
#                      Symbol Substitution
#                      -------------------
#
# The following substitutions are supported:
#   &USSHOME;    => The value of the USSHOME SIT parameter.
#   &CONFIGROOT; => the location of configuration files, such as the
#                   JVM profile.
#   &APPLID;     => The applid of the CICS region.
#   &JVMSERVER;  => The name of the JVMSERVER resource.
#   &DATE;       => Date the JVMSERVER is enabled (localtime). Dyymmdd
#   &TIME;       => Time the JVMSERVER is enabled (localtime). Thhmmss
#
# All variables must be delimited with & and ; for example
#     ENV_VAR=myvar.&APPLID;.&JVMSERVER;.data
#
# Note: The continuation character for use with JVMProfiles is '\'.
#**********************************************************************
#
#                      Required parameters
#                      -------------------
#
# JAVA_HOME specifies the location of the Java directory.
JAVA_HOME=/usr/lpp/java/J7.0_64/
#
# Set the current working directory.
WORK_DIR=/var/cicsts/devops/workdir
#
#**********************************************************************
#
#                WebSphere Liberty Profile server (WLP)
#                --------------------------------------
#
# To enable a Liberty (WLP) server.
WLP_INSTALL_DIR=&USSHOME;/wlp
#
# Optionally define the directory to store shared and server-specific
# configuration.
WLP_USER_DIR=./&APPLID;/&JVMSERVER;/wlp/usr
#
# Optionally define the directory to contain output files for defined
# servers.
WLP_OUTPUT_DIR=./&APPLID;/&JVMSERVER;/wlp/user/servers
#
# Instruct CICS to create and configure the WLP.
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
#
# Instruct CICS to set the port used for HTTP requests.
```

```
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=53040
#
# Instruct CICS to set the port used for HTTPS requests.
-Dcom.ibm.cics.jvmserver.wlp.server.https.port=53041
#

# The following JVM options are optimized for generic Liberty workloads
# -Xms    Initial Java heap size
# -Xmx    Maximum Java heap size
# -Xmso   Initial stack size for native threads (JVM default -Xmso256KB)
-Xms128M
-Xmx256M
-Xmso128K
#

# Set the Garbage collection Policy.
-Xgcpolicy:gencon
#
# The following option enables the "aggressive" level of Just-In-Time
# (JIT) optimization, exploiting the latest hardware optimizations
# available from the IBM 64-bit SDK for z/OS, Java Technology Edition.
#-Xaggressive
#
# Using a shared class cache
# --------------------------
# JVM servers within the same z/OS LPAR can use a common Java
# shared class cache. This reduces the overall virtual storage
# consumption, improves JITing and reduces JVM startup time. The
# following options create a 128 MB shared cache, which can be accessed
# by all CICS regions in the same RACF group.
-Xscmx128M
-Xshareclasses:name=cicsts530%g,groupAccess,nonfatal
#
-Dcom.ibm.tools.attach.enable=no
#
# WebSphere Application Server Liberty profile is only supported when
# running under a JVM with the UTF-8 or ISO-8859-1 character encodings.
-Dfile.encoding=ISO-8859-1
#
```

After you create the jvmprofile in a zFS directory, you should create a JVMSERVER resource definition in CICS that points to the location of the new profile. Make sure that the resource is installed and that the server is enabled.

6.4 Importing the application source code and setting up CICS Explorer

To develop and deploy web applications, you must install the CICS SDK for Servlet and JSP support into CICS Explorer. The steps that are required to install the SDK depend on the method that you chose to install CICS Explorer. For more information, see Chapter 2, “Development environment” on page 11.

After installing the SDK, you can import the source code from the GENAPP Liberty extension that you downloaded by completing the following steps:

1. Switch to the Java perspective by clicking **Window** → **Open perspective** → **Other**, and then clicking **Java** (see Figure 6-4)**.**

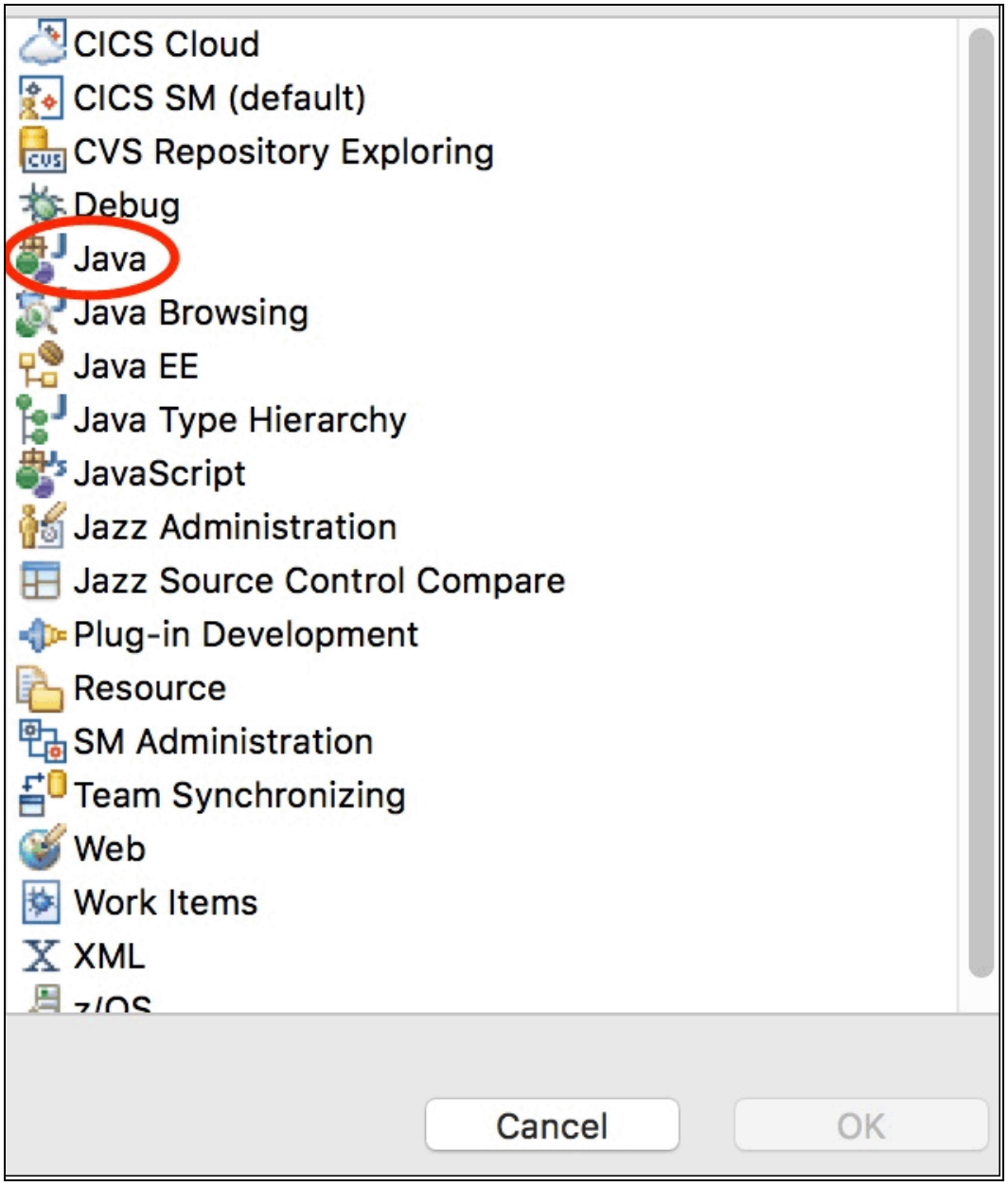


Figure 6-4 Final step of switching to the Java perspective

2. Import the predefined source code that you downloaded:
   a. Click **File** → **Import** and then select **Existing Projects into Workspace** (see Figure 6-5).

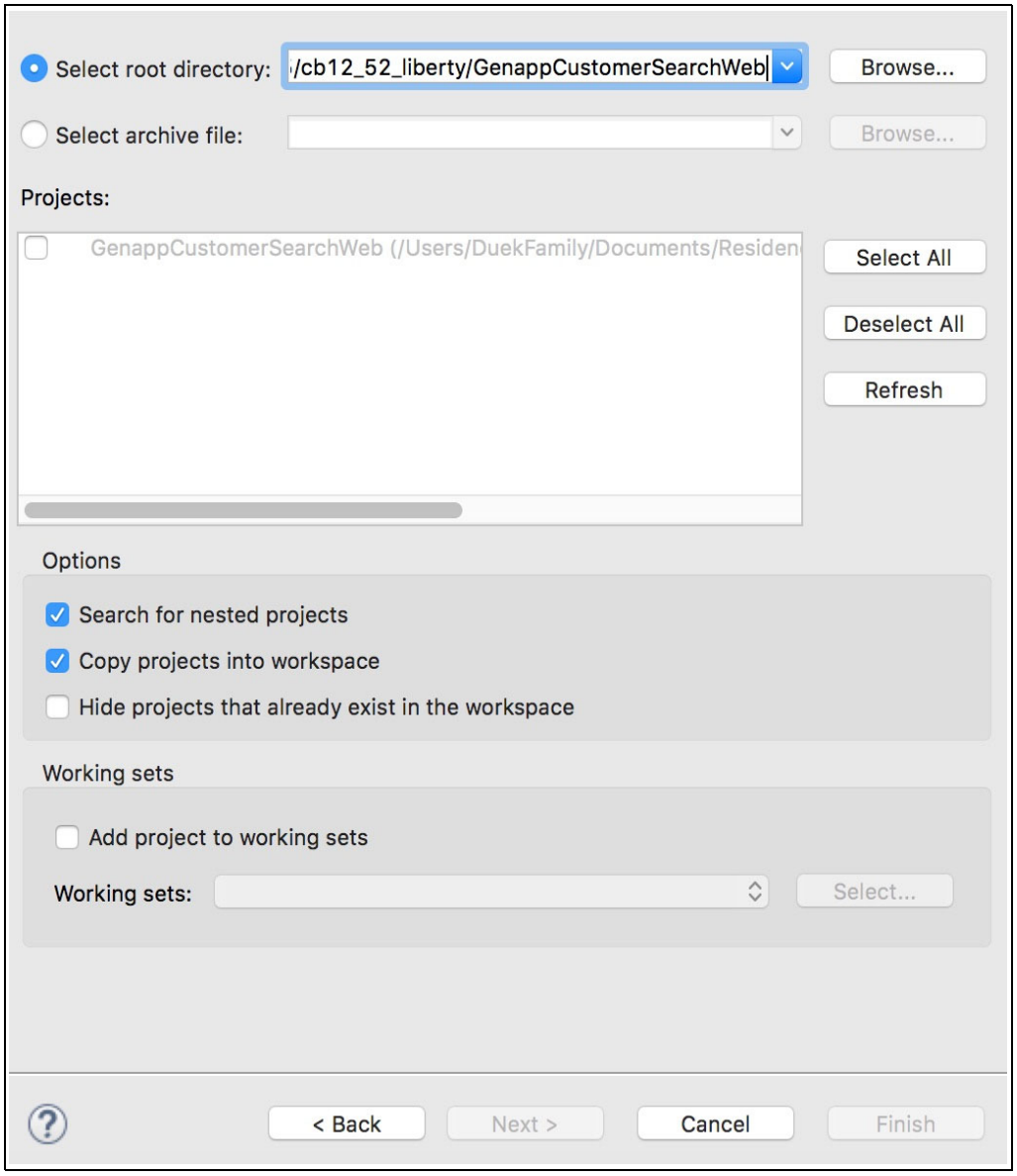


Figure 6-5 Import existing projects into the workspace

   b. Select the folder that contains the downloaded GENAPP Liberty extension, choose the project to import, and click **Finish** (see Figure 6-6 on page 137).

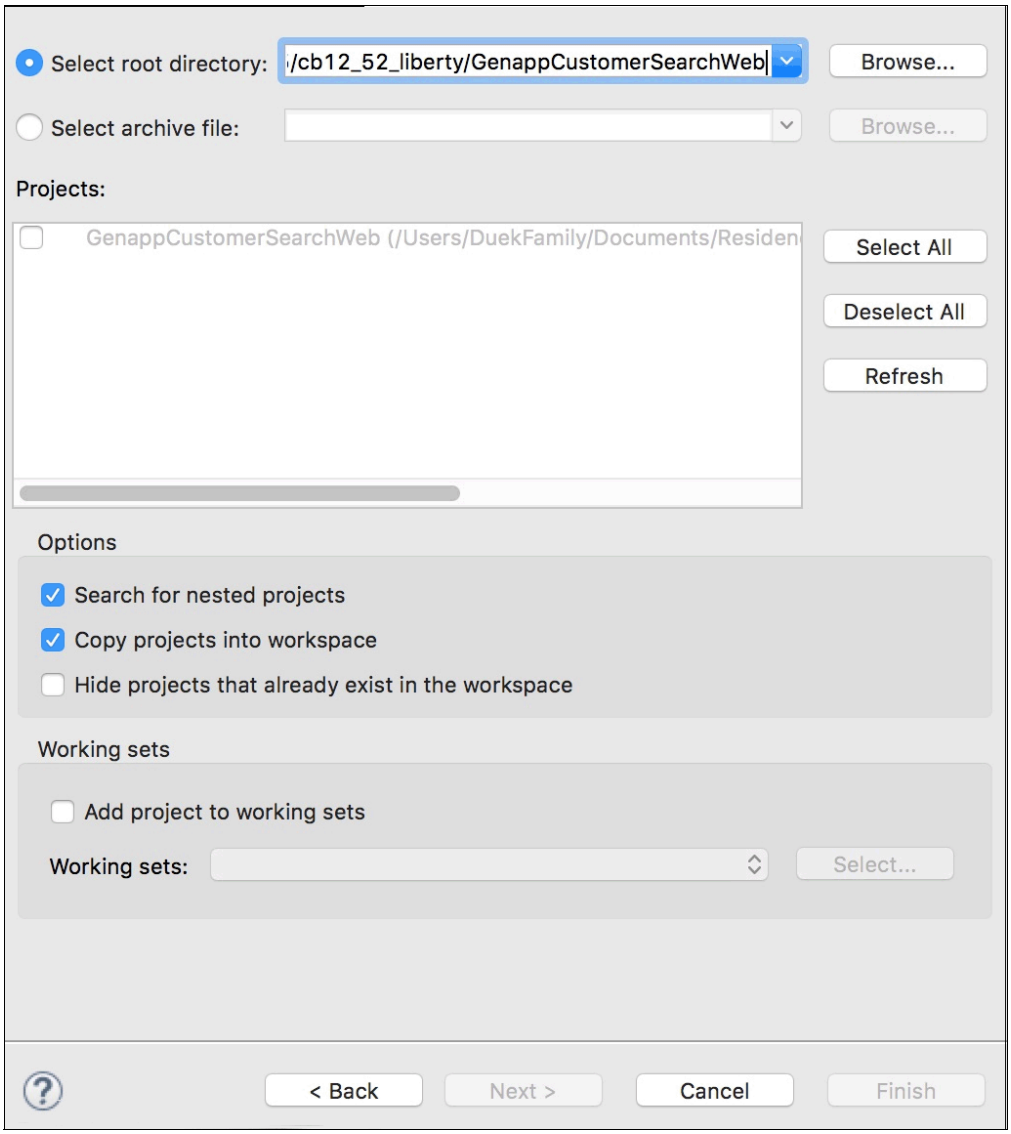


Figure 6-6 Select a project to import

The imported project now is listed in the Package Explorer view.

6.5 Deploying the Liberty extension web application

Note: The GENAPP Liberty extension requires the GENAPP Base application. If you have not installed the GENAPP bAse application, do so before completing the steps in this section.

To deploy the Java application in CICS *without* using automated build and deploy processes, complete the following steps:

1. Create the CICS bundle project, which is exported to the zFS later, and refer to the web project that you already imported.
2. Export the CICS bundle project to a specific location on zFS.

3. Discard any prior versions of the CICS bundle resource (if any exist).
4. Create the CICS bundle definition that points to the new bundle directory (containing the latest version you want to deploy).
5. Install the new CICS bundle.
6. Check the Java application to confirm that it is working.

Based on the steps above, you can deploy the Liberty extension in CICS Transaction Server for the first time by completing the following steps:

1. Create a CICS bundle project:
   a. Click **File** → **New** → **Other** and then select **CICS Bundle Project** (see Figure 6-7).

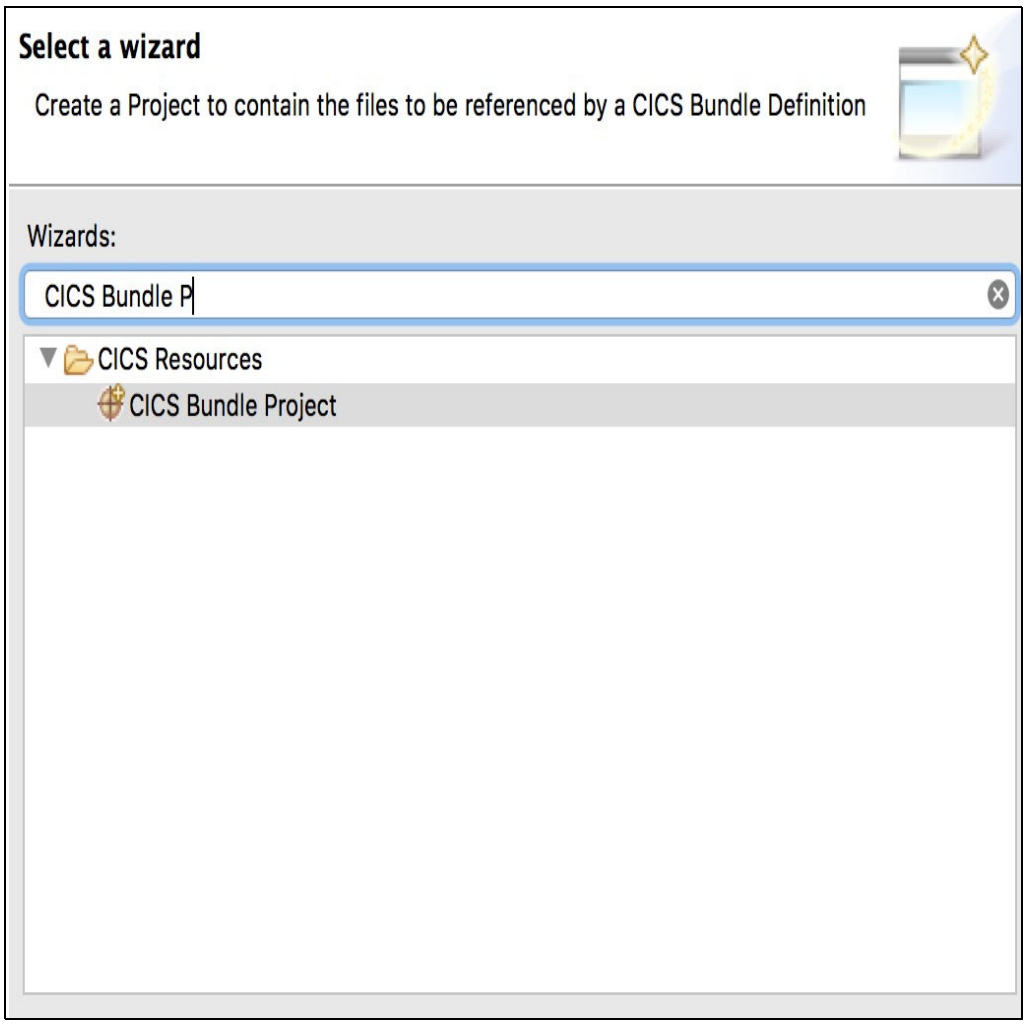


Figure 6-7 CICS Bundle Project

b. Give the project a name and click **Finish** (see Figure 6-8). After you complete this step, the project should appear in the Package Explorer view of your workspace.

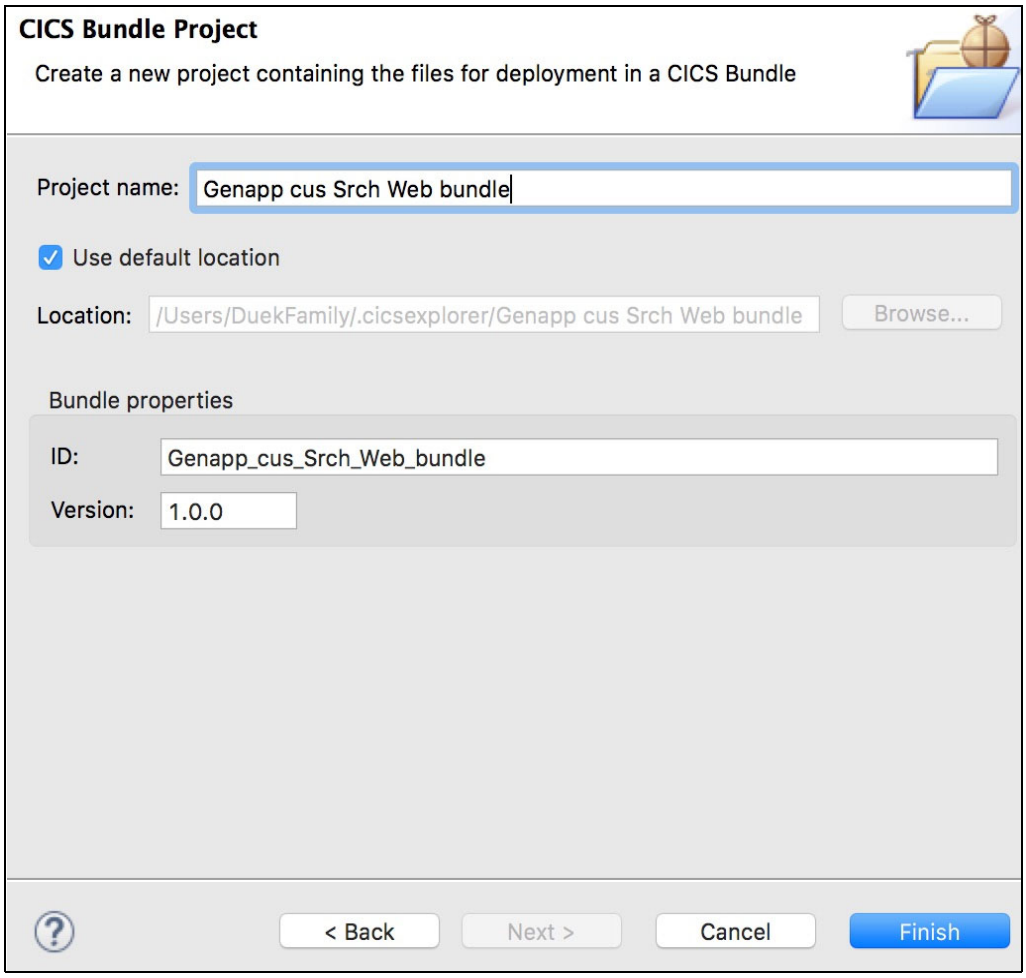


Figure 6-8 CICS Bundle Project attributes

2. Reference your CICS bundle to the web application project:
   a. Open the `cics.xml` file (look in the `META-INF` directory in CICS bundle project that you created in step 1 on page 138). In the Defined Resources section, click **New** and select **Dynamic Web Project Include** (see Figure 6-9).

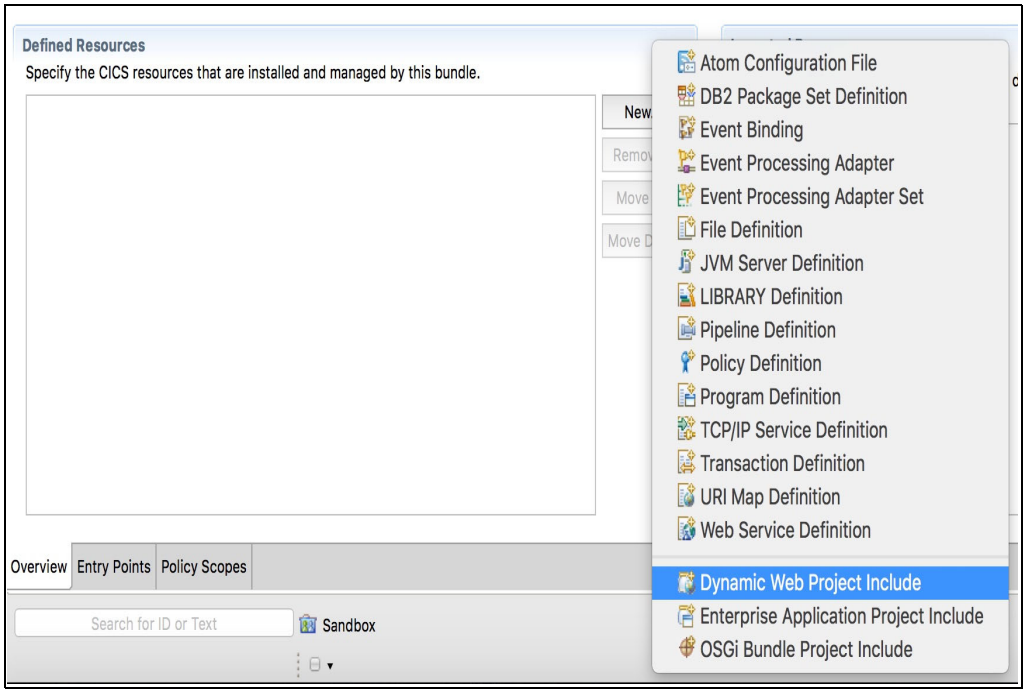


Figure 6-9 Add a reference to the web project

b. Select the imported project containing the web application and specify the JVM server name on which the application should run under CICS. Click **Finish** (see Figure 6-10).

Note: Section 6.6, “Creating an automated build and deployment process by using Rational Team Concert and the CICS build toolkit” on page 146 describes the use of variables instead of hardcoded values. Instead of writing *GENALIBS* as the JVM server name in this scenario, we use a variable that is to the correct value according to the environment to which it is being deployed.

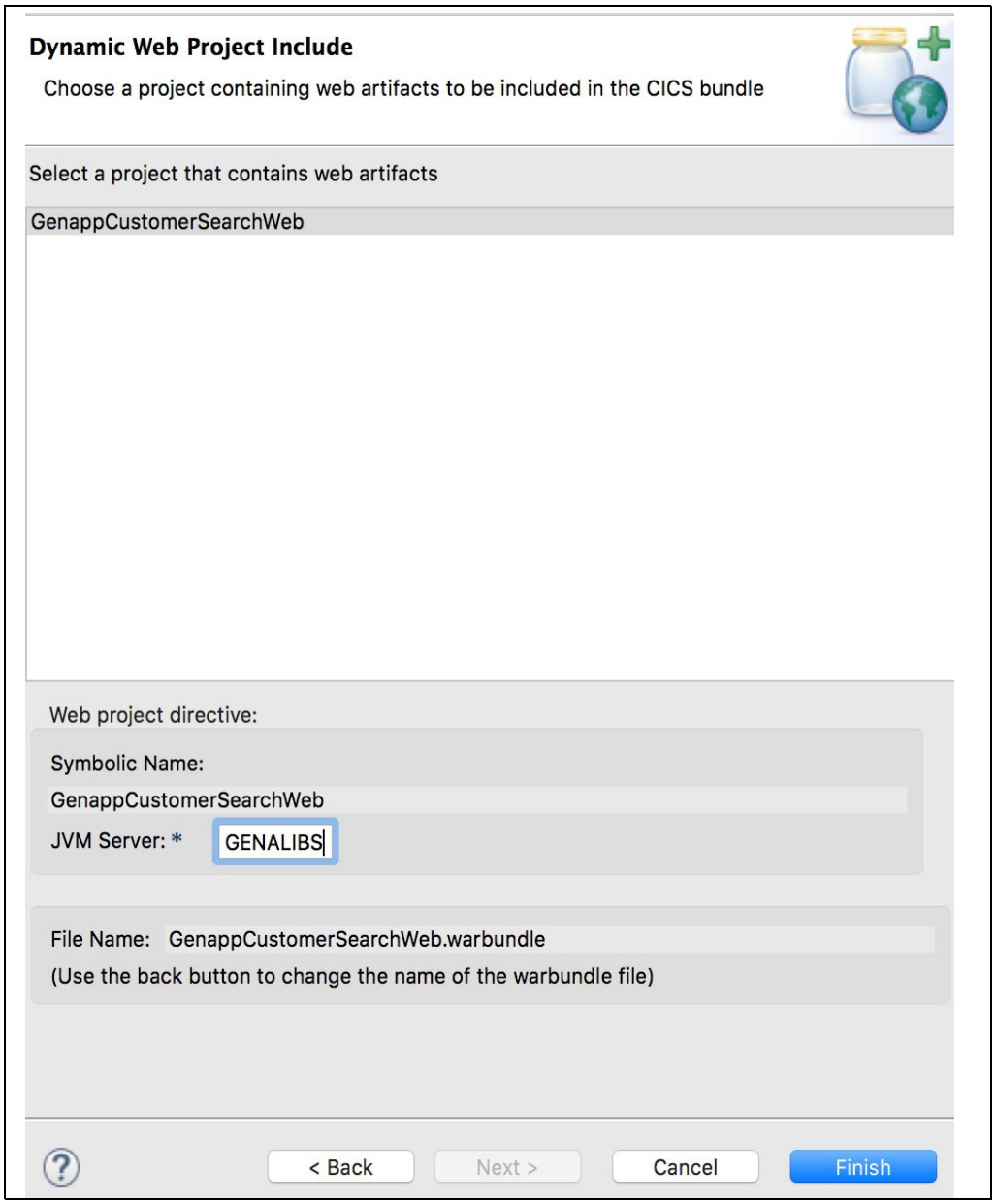


Figure 6-10 Naming the JVM server on which the application will run

3. Export your CICS bundle project to the wanted location:

   a. Right-click the CICS bundle project that you created and select **Export Bundle Project to z/OS UNIX File System** (see Figure 6-11).

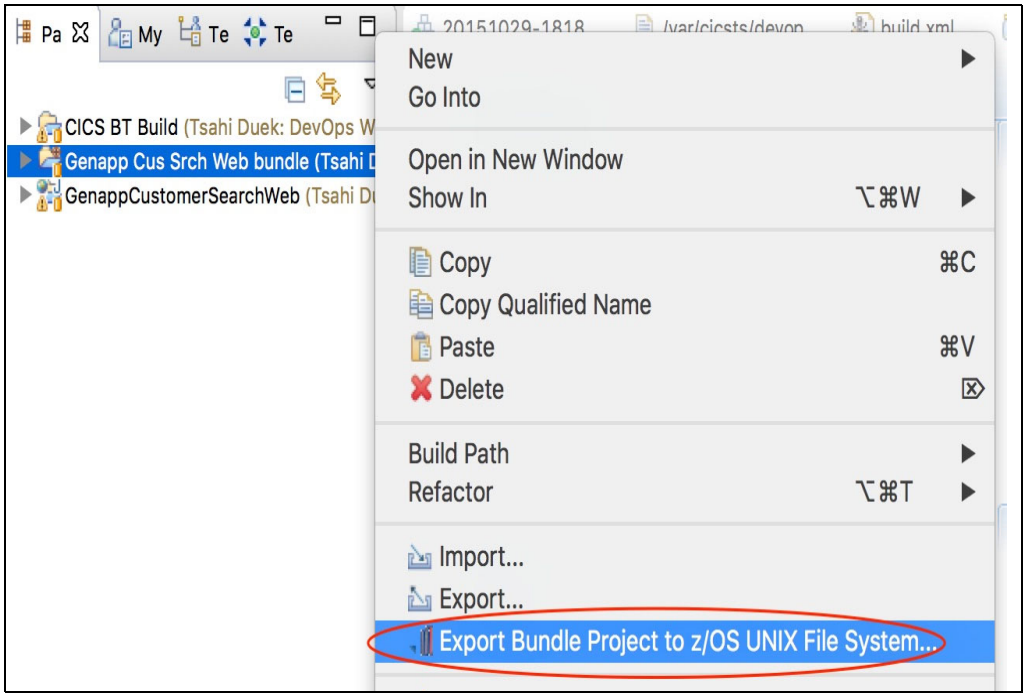


Figure 6-11 Select the export destination

   b. In the window that opens, select **Export to a specific location in the file system**.

   c. Select the directory where you want the bundle definition stored and click **Finish** (see Figure 6-12 on page 143).

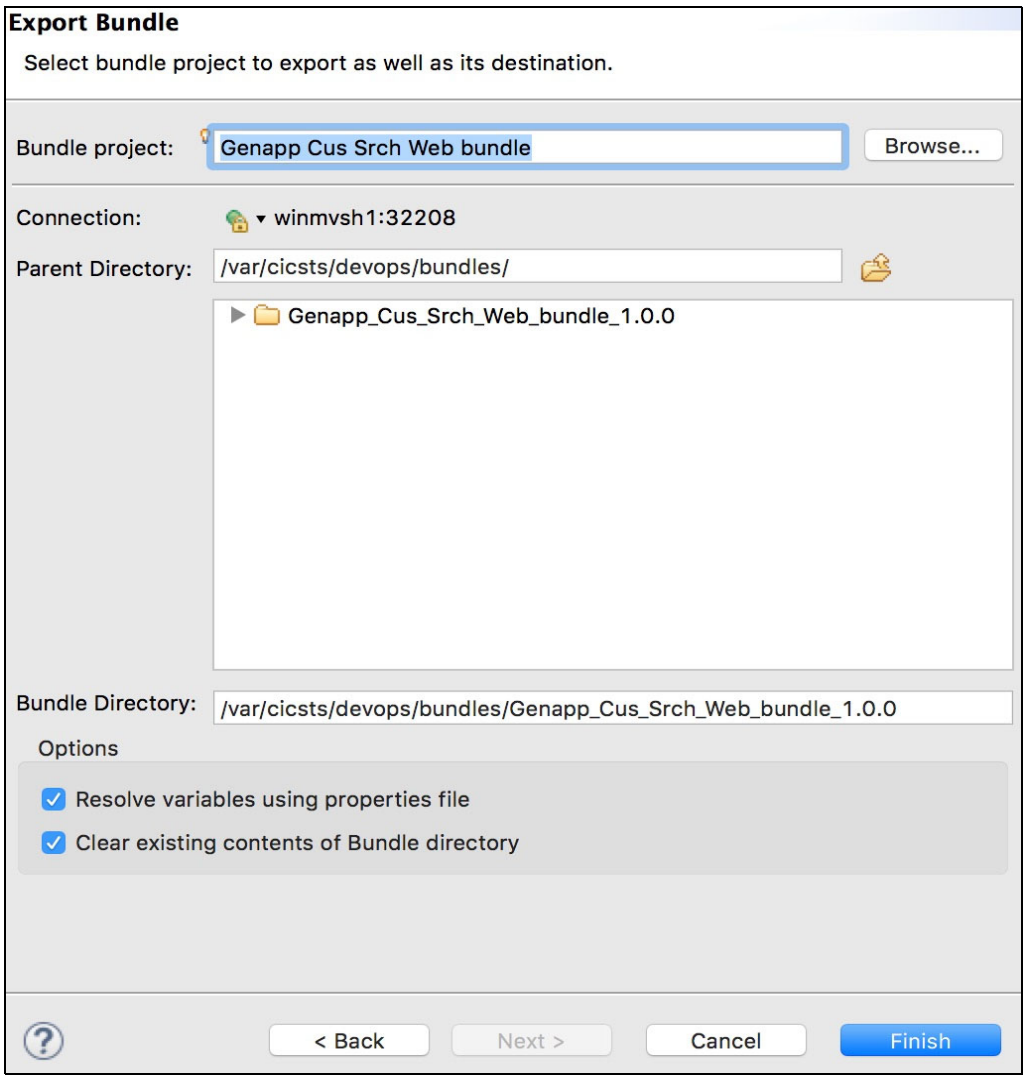


Figure 6-12 Select the folder where the bundle will be stored

4. Define and install the bundle resource definition within CICS:
   a. Switch to the CICS SM perspective.
   b. Open the bundle definition view and click **Definitions** → **Bundle Definitions**.
   c. Initiate creation of the new bundle definition by clicking **File** → **New** → **Other** and then selecting **Bundle Definition**.

d. Define the bundle by specifying the CICSPlex name, region (if you want the bundle definition in your CSD), CSD group, bundle name, and the directory on zFS where the bundle exists (see Figure 6-13).

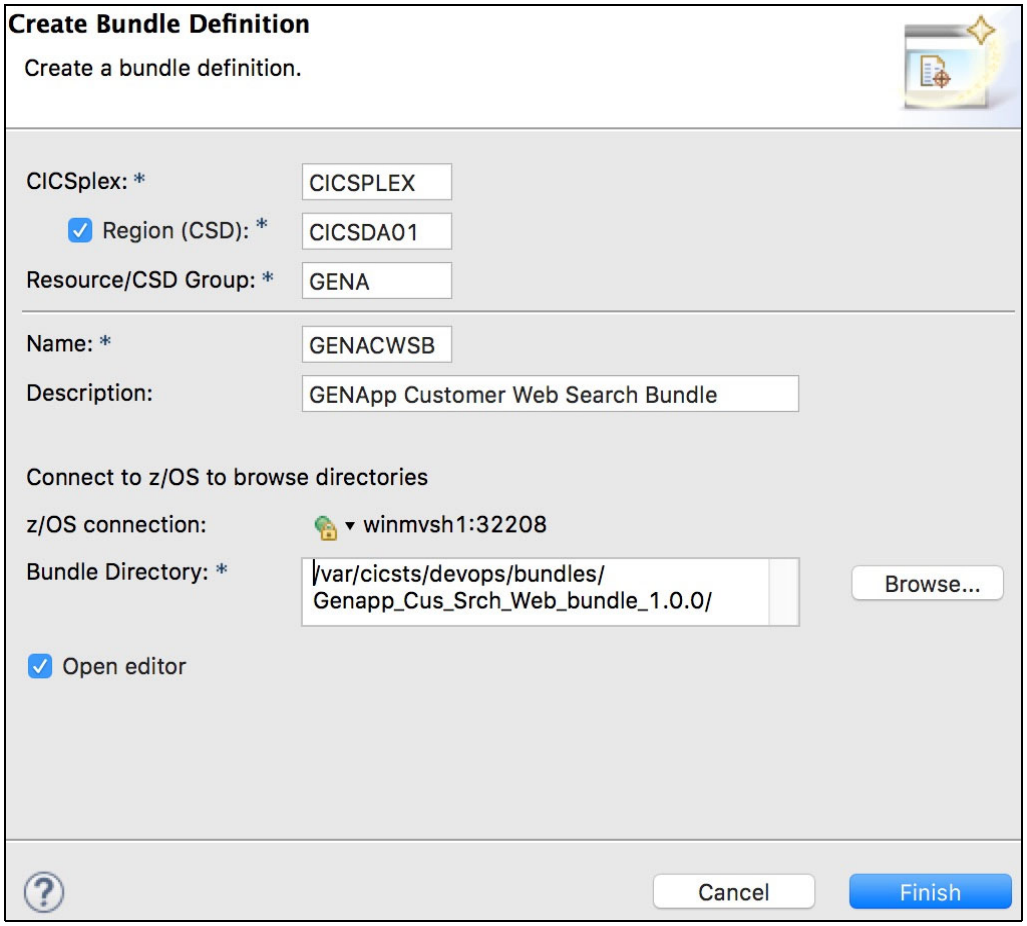


Figure 6-13 Define the CICS bundle

e. Install the bundle by right-clicking the bundle name and selecting **Install**. Then, select the CICS region in which to install the bundle and click **OK**. The bundle is now installed and enabled (to confirm, click **Operations** → **Bundles** to see the Bundle view (Figure 6-14).

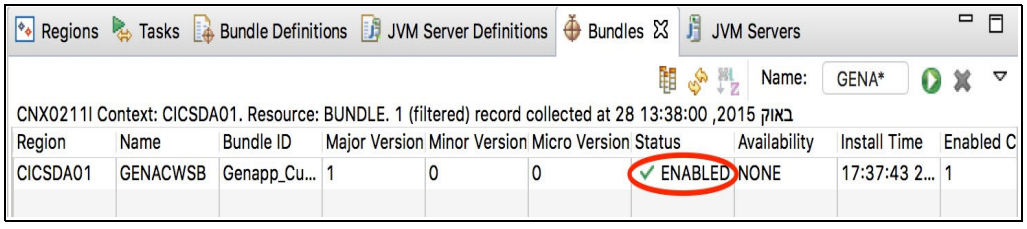


Figure 6-14 Bundle view showing the newly enabled CICS bundle

The web application is now installed in the JVM server within CICS, demonstrating the successful use of the JCICS API and Liberty JVM server with CICS. The application is accessible from the following location and appears in your browser, as shown in Figure 6-15:

```
https://<your_MF_localhost>:<https_port_defined_in_JVMProfile>/Genapp_Customer_Search/
```

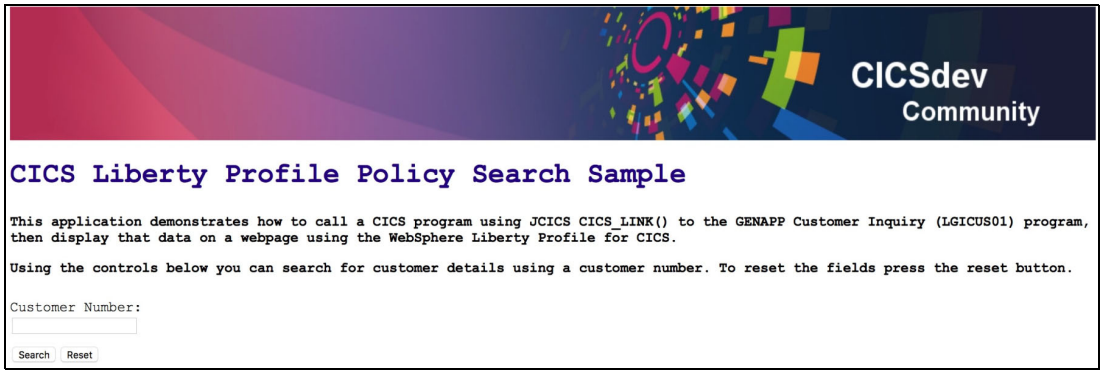


Figure 6-15 Browser view of the GENAPP Liberty extension web application

When you enter a Customer Number (for example, 1), and click **Search**, you see the output that is shown in Figure 6-16.

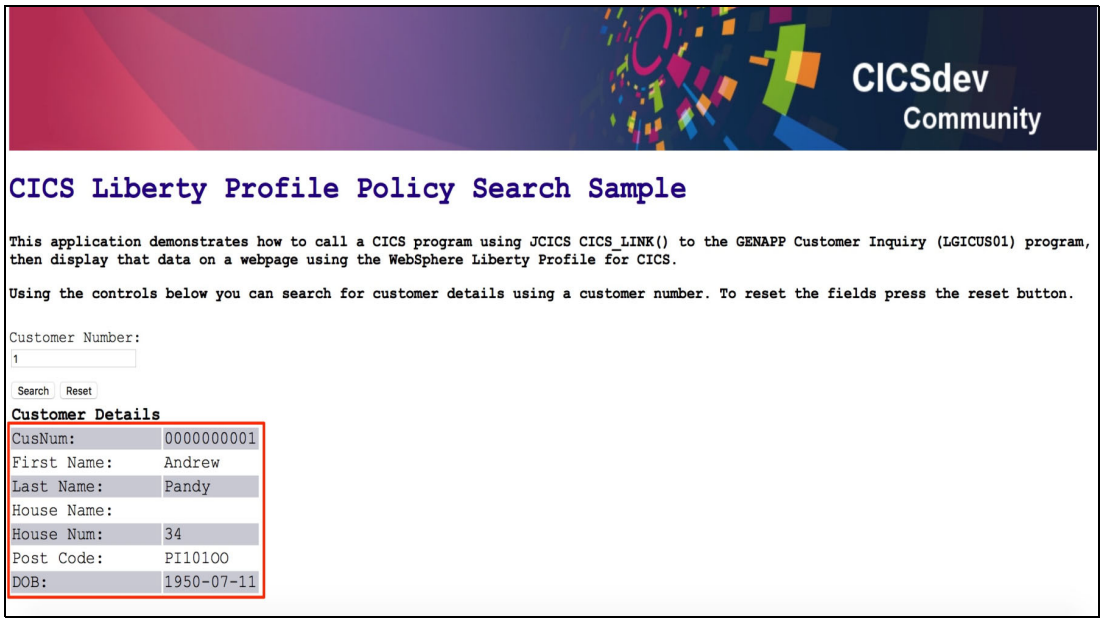


Figure 6-16 Customer search result details when searching for customer number 1

6.6 Creating an automated build and deployment process by using Rational Team Concert and the CICS build toolkit

The build and deploy process involves many manual activities. You must export the CICS bundle project to z/FS, copy the bundle to the correct location based on the environment to which you are deploying, create and install a CICS bundle resource definition that points to the directory of the newest version, and verify that the CICS bundle resource is enabled.

To achieve continuous integration and continuous deployment as addressed by DevOps methodology, you must create an automated build and deployment process. The automated build uses the CICS build toolkit to create deployable artifacts from the CICS bundle project. The automated deployment copies the zFS directory that contains the built bundle from the build process, resolves attribute variables (for example, the JVM server name), disables and discards the current CICS bundle resource, and creates and installs the new CICS bundle resource.

To demonstrate how automated build and deployment processes work, make a change in the application. In this example, the application outputs at table result of the customer details, as shown in Figure 6-17. The change that you make is to add another field, “Phone Mobile”, to the customer details result. By looking at the source code of GENAPP and its Liberty extension, you can tell that the information is already in the web application COMMAREA (see Group-Item `CA-CUSTOMER-REQUEST` in COPY-MEMBER `LGCMAREA`, and the `InquiryCommarea` Java class that is supplied with the Liberty extension).

```
      *    Fields used in INQ All and ADD customer
           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).
```

Figure 6-17 Group definition for the inquiry results

As a result, the only thing that you must do is to add a fields definition that corresponds to the field in the COBOL COPY-MEMBER, and put these fields in the JSP file.

To make these changes, complete the following steps (all the changes are performed under the CICS Explorer IDE):

1. Add the mapping fields `num_policies` and `phone_mobile` to the Java program `InquiryCommarea.java` (see Figure 6-18 on page 147).

```
private final int[] commarea_buffer = {0,32500}; //Length of commarea
private final int[] return_code = {6,2}; //Allocation for return code
private final int[] cus_num = {8, 10}; //Allocation for customer number
private final int[] first_name = {18, 10}; //Allocation for first name
private final int[] last_name = {28, 20}; //Allocation for last name
private final int[] dob = {48, 10}; //Allocation for date of birth
private final int[] house_name = {58, 20}; //Allocation for house name
private final int[] house_num = {78, 4}; //Allocation for house number
private final int[] post_code = {82, 8}; //Allocation for post code
private final int[] num_policies = {90, 3}; //Allocation for num policies
private final int[] phone_mobile = {93, 20}; //Allocation for post code
```

Figure 6-18 InquiryCommarea fields change

2. Add matching getter and setter methods to the `phone_mobile` field in the `InquiryCommarea.java` program (see Figure 6-19 and Figure 6-20).

```
public String getPost_code()
{
    return getData(post_code[0], post_code[1]);
}

public String getPhone_mobile()
{
    return getData(phone_mobile[0], phone_mobile[1]);
}

public void setReturn_code(String value)
{
    setNewValue(value, return_code[0], return_code[1]);
}
```

Figure 6-19 Getter method for new phone_mobile field

```
public void setPost_code(String value)
{
    setNewValue(value, post_code[0], post_code[1]);
}

public void setPhone_mobile(String value)
{
    setNewValue(value, phone_mobile[0], phone_mobile[1]);
}

private void setNewValue(String newValue, int index, int length)
```

Figure 6-20 Setter method for new phone_mobile field

3. Print the new field (Phone Mobile) in the `FetchData.java` program (see Figure 6-21).

```
        {
            //No errors, print out the value of each commarea field
            out.println("<row><name>CusNum:</name><value>" + inqcom.getCus_num() + "</value></row>"
            out.println("<row><name>First Name:</name><value>" + inqcom.getFirst_name() + "</value>
            out.println("<row><name>Last Name:</name><value>" + inqcom.getLast_name() + "</value></
            out.println("<row><name>House Name:</name><value>" + inqcom.getHouse_name() + "</value>
            out.println("<row><name>House Num:</name><value>" + inqcom.getHouse_num() + "</value></
            out.println("<row><name>Post Code:</name><value>" + inqcom.getPost_code() + "</value></
            out.println("<row><name>DOB:</name><value>" + inqcom.getDob() + "</value></row>");
            out.println("<row><name>Phone Mobile:</name><value>" + inqcom.getPhone_mobile() + "</va
        }
        out.println("</record>");
```

Figure 6-21 Print the new phone_mobile field in the FetchData method

4. Change the version number in the `cics.xml` file in the CICS bundle project to 1.1.0 to indicate that this is the newer version of this application.
5. Use variable substitution to replace the hardcoded value for the JVM server name in the `warbundle` file:
   a. Replace the value of the `jvmserver` attribute with the variable *`jvm.server.name`*. Variables are identified by ${*xxx*}, where the xxx between the brackets indicates the variable name (see Figure 6-22). The CICS build toolkit resolves the variable at the correct time.

```
GenappCustomerSearchWeb.warbundle
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<warbundle symbolicname="GenappCustomerSearchWeb" jvmserver="${jvm.server.name}"/>
```

Figure 6-22 Adding a variable to the warbundle file

   b. Add a `variables.properties` file to the CICS bundle project. This file defines a default value for the variable (see Example 6-2).

Example 6-2 Variables.properties file in the bundle project provides a default value for the JVM server name

```
jvm.server.name=GENAJVM
```

   c. Create a properties file for each environment to provide the variable value that applies there. When the variable is resolved, the value in the environment-specific properties overrides the default value that is provided in the bundle project `variables.properties` file (see Example 6-3 for an example properties file for a test environment). These files can be stored in a project in source code management (SCM) system alongside the application source, but must be made available on zFS at deployment time.

Example 6-3 Test.properties file provides a variable value for the test environment

```
jvm.server.name=TESTJVM
```

When the changes are completed, the next thing you must do is to build the application.

Here, you can enjoy the benefit of continuous integration. Instead of exporting the changes manually to the z/FS, you can use a build engine to start the CICS build toolkit to create the new bundles. The build engine in our example is Rational Team Concert Build with Ant integration, which you can use to run the **cicsbt** command to create the bundle in the build directory in z/FS.

For this example, you must create a similar definition based on the Ant – Jazz Build Engine template that is provided in Rational Team Concert. We schedule this build definition to run every 5 minutes. We use a build engine that is installed in z/OS so that we can build the bundle directory directly to z/FS. The properties for this build definition are shown in Table 6-1.

Table 6-1 Build definition properties

Name	Value	Use of property
ant.work.dir	workdir	Working directory for the Ant script
build.output.dir	/var/cicsts/devops/builds/genappweb	Output directory for the CICS build toolkit build command
bundle.name	Genapp_Cus_Srch_Web_bundle	Bundle name to build
load.dir	${ant.work.dir}/${source.dir}	Load directory to be use in Rational Team Concert build definition
source.component	GENAPP Customer Search (Liberty)	Source component in Rational Team Concert
source.dir	source	The name of the source directory for the CICS build toolkit build command
target.platform	com.ibm.cics.explorer.sdk.web.liberty53.target	The name of the target platform for the CICS build toolkit build command

In this example, we use an Ant build script to run the **cicsbt_zos** command in z/OS.

After we create the build definition, we request a build so that Rational Team Concert builds the bundle in the correct z/FS build output directory (`/var/cicsts/devops/builds/genappweb`). The suggested structure is to have a subdirectory for each logical application. In this case, the directory contains the `Genapp_Cus_Srch_Web_Bundle_1.0.0/ bundle` directory that has the WAR file `warbundle` file, and the `cics.xml` file (see Figure 6-23).

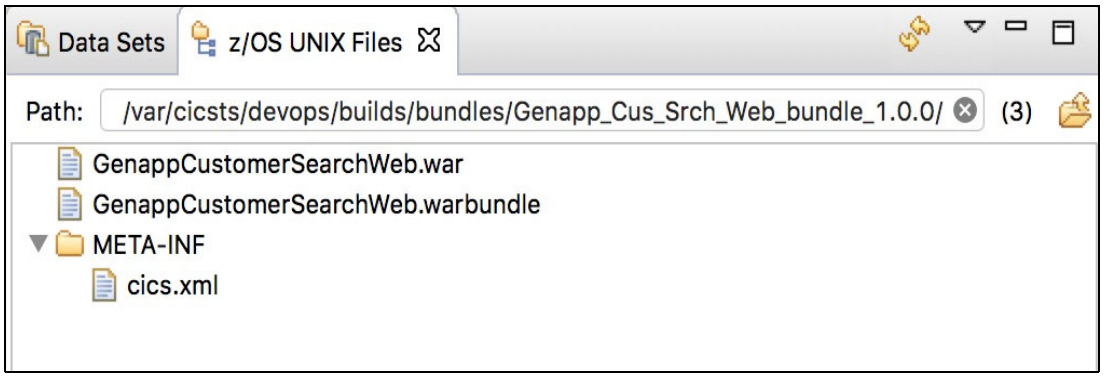


Figure 6-23 Bundle that is built by the build definition

6.7 Deploying the build

Next, deploy the bundle to an appropriate bundle directory (for example, the one in your test environment), create a CICS bundle definition, and install it. This example assumes that you have a previous version of the bundle that is installed with a lower semantic version.

For these processes, use the new DFHDPLOY utility in CICS Transaction Server V5.3, which you can use to define, install, discard, and set the availability status of bundles. The DFHDPLOY utility can be started from JCL and integrated with your existing deployment and automation procedures. Use z/OS commands to copy zFS directories and the CICS build toolkit to resolve variables in the bundle.

Complete the following steps:

1. Copy the specific bundle directory that you want to deploy to a temporary work directory. This step is important because when you use the CICS build toolkit resolve command, it resolves *all* the bundles in the input directory, and you want to resolve only a specific bundle. For this step, use the BPXBATCH utility, which you can use to issue SHELL scripts or UNIX programs.
2. Clean the work directory that you will use for the resolved bundle.
3. Resolve the variables in the bundle by running the CICS build toolkit resolve command. After you run this command, the bundle is in a work directory, which ensures that if the resolve step fails, the existing target directory remains intact. Use an environment-specific properties file to provide the appropriate value for the JVM server name.
4. Copy the resolved bundle to the target directory if the resolve step succeeds. This is the directory from which you install the bundle definition, and it depends on the environment to which you are deploying.
5. Use the DFHDPLOY utility to make two changes:
   a. Undeploy the current bundle.
   b. Deploy the new bundle from the appropriate bundle directory and enable it.

   If you have multiple bundles to deploy, you need to perform several instances of this step.

DFHDPLOY takes care of discarding the old CSD definition for the BUNDLE resource and creating the new one in the specified group. Because your are deploying a new version of an existing bundle, assume that the CSD group is added to an appropriate group list to ensure that the bundle is installed on a cold start of the CICS system. When you deploy a bundle to a group that has not been used before, you can add a step that calls the DFHCSDUP utility to add the group to a list. In general, your deployment script should be robust enough to be rerun if there is a failure, for example, if the set of CICS regions in the scope change during the course of deployment.

It is important to understand that this deployment process results in a short loss of service for the application. This loss occurs in the short time frame from when you undeploy the old version of the bundle until the new version of the bundle is enabled. If clients attempt to access the web page during this time, they get an HTTP 404 Not Found response.

If you must deploy a web application with no outage, you can create a version of the application with a different URI for the context root. You can then deploy the new bundle alongside the old version, and use the new URI to reach the new version. Later, you can undeploy the old version. For this method to work, you must use two different names for the BUNDLE resource, rather than the same name described above. For more information about processes for lifecycling and versioning web applications, see *CICS and Liberty: What You Need to Know*, SG24-8335.

Another consideration when this process is applied to production deployments is quiescing the workload. Before undeploying any version of a web application, you should allow workload to drain away. For example, if you have a load balancing configuration, you can temporarily direct work away from the systems where the deployment occurs. When the bundle is disabled, any in-flight work completes, but any new requests fail.

Example 6-4 shows the JCL that is used to complete the steps that were described. The example uses symbols in in-stream data, which was first supported in z/OS V2.1. If you are running an earlier version of z/OS, you must modify the example as needed.

Example 6-4 Deploy a JCL by using the DFHDPLOY utility

```
//GENADPLY JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//EXPORT    EXPORT SYMLIST=*
//*
//          SET  CPSMHLQ=CICSDPP.BETA14.CTS53BT.CPSM
//          SET  CICSHLQ=CICSDPP.BETA14.CTS53BT.CICS
//          SET  CICSPLEX=CICSPLEX
//          SET  JAVAHOME='/java/J8.0_64/'
//          SET  BUNWRKDR='/var/cicsts/devops/workdir'
//          SET  BUNBLDDR='/var/cicsts/devops/builds/bundles'
//          SET  PROPSDIR='/var/cicsts/devops/properties'
//          SET   ENVNAME='test'
//          SET  TARGETDR='/var/cicsts/TESTPLEX'
//          SET  BUNDNAME='Genapp_Cus_Srch_Web_bundle_1.1.0'
//          SET  CICSBTCM='/usr/lpp/cicsts/cicsbt/cicsbt/cicsbt_zos'
//*
//          SET  CIBUNDNM=GENACWSB
//          SET  CISCOPE=CICSDA01
//          SET  CSDGROUP=GENAWEB
//*
//* First step is to clean up the work directory
//*
```

```
//CLEAN   EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
//STEPLIB  DD   DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV   DD   *,SYMBOLS=JCLONLY
JAVA_HOME=&JAVAHOME
/*
//STDPARM  DD   *,SYMBOLS=JCLONLY
SH
rm -r &BUNWRKDR/*;
//*
//* Second step is to resolve the bundle using cicsbt_zos
//*
//RESOLVE EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
//STEPLIB  DD   DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV   DD   *,SYMBOLS=JCLONLY
JAVA_HOME=&JAVAHOME
/*
//STDPARM  DD   *,SYMBOLS=JCLONLY
PGM
&CICSBTCM
   --resolve &BUNBLDDR
   --output &BUNWRKDR
   --properties &PROPSDIR/&ENVNAME..properties
   --verbose
//*
//* Third step is to copy the resolved output to the target dir
//*
//COPY   EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G,COND=(0,NE,RESOLVE)
//STEPLIB  DD   DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV   DD   *,SYMBOLS=JCLONLY
JAVA_HOME=&JAVAHOME
/*
//STDPARM  DD   *,SYMBOLS=JCLONLY
SH
cp -r
&BUNWRKDR/*
&TARGETDR
//*
//* Final step is UNDEPLOY the old bundle
//* then DEPLOY and set available the new bundle
//*
//DFHDPLOY  EXEC PGM=DFHDPLOY,REGION=1OOM,COND=(0,NE,COPY)
//*
//STEPLIB   DD   DISP=SHR,DSN=&CPSMHLQ..SEYUAUTH
//          DD   DISP=SHR,DSN=&CICSHLQ..SDFHLOAD
//*
//SYSTSPRT  DD   SYSOUT=*
//*
//SYSIN  DD   *,SYMBOLS=JCLONLY
*
```

```
   SET CICSPLEX(&CICSPLEX);

UNDEPLOY BUNDLE(&CIBUNDNM)
          SCOPE(&CISCOPE)
          CSDGROUP(&CSDGROUP)
          STATE(DISCARDED);
*
*   Define a CICS bundle resource and move it towards an
*   ENABLED state within the current CICSplex.
*
*
 DEPLOY BUNDLE(&CIBUNDNM)
        SCOPE(&CISCOPE)
        CSDGROUP(&CSDGROUP)
 BUNDLEDIR(&TARGETDR/bundles/&BUNDNAME/)
        STATE(ENABLED);
/*
```

Note: The JCL that is shown in Example 6-4 on page 151 is fully parameterized and can be easily modified for different environments and bundles. It can also be a baseline for automatically generated JCLs. The final steps run only if the previous step succeeded.

Table 6-2 provides details about the steps that are involved in the JCL that is shown in Example 6-4 on page 151.

Table 6-2 JCL steps that are involved in Example 2

Step	Description
CLEAN	Deletes the contents of the work directory `/var/cicsts/devops/workdir/`.
RESOLVE	Resolves the bundle, by using the CICS build toolkit, to the work directory (in this case, `/var/cicsts/devops/workdir`) with the appropriate properties files for the environment (in this case, `/var/cicsts/devops/properties/test.properties`).
COPY	Copies the resolved bundle from the work directory to the target location (`/var/cicsts/devops/workdir` to `/var/cicsts/CICSPLEX`) by running the **cp** command.
DFHDPLOY	Undeploys the current bundle (disables and discards the resource and then deletes the definition from the CSD) and deploys the new bundle (points to the new bundle directory, Version 1.1.0).

After submitting the JCL, the changes should be applied to the application and the result should be as shown in Figure 6-24.

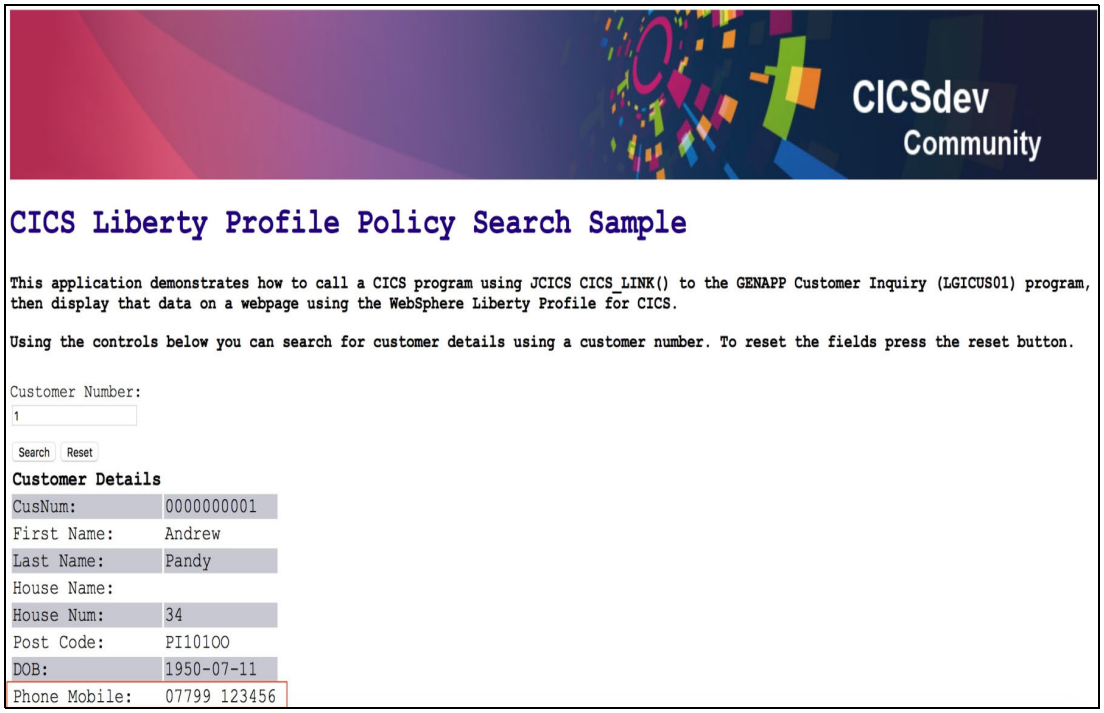


Figure 6-24 Web application after the change was applied (Version 1.1.0)

Here is a summary of what was done in this chapter:

1. Installed the first version of the example application with no automatic build or deployment process. Everything was done manually (exporting the bundle, disabling and discarding the CICS bundle resource, defining a CICS bundle with the appropriate zFS directory, and installing the bundle in the wanted CICS region).
2. Created a build definition that integrates the Rational Team Concert Build engine, Ant build script, and CICS build toolkit's build command to build the wanted bundle in a zFS directory.
3. Created a deployment process that uses the CICS build toolkit's resolve command to resolve variables within resource definition attributes, undeploys the CICS bundle resource definition, defines the CICS bundle with the newer version of the deployed bundle, and installs it in the target environment.

7

Applying DevOps to CICS cloud applications

This chapter demonstrates how the new DevOps tools can be used to deploy a CICS cloud application to multiple environments by using a repeatable and automatic process.

This chapter covers the following topics:

7.1 CICS and cloud applications

Introduced in CICS Transaction Server V5.1, CICS cloud applications provide a mechanism for encapsulating all of the resources that comprise a business application in one entity, which can be deployed, enabled, disabled, and removed as a unit. Certain resources can be tagged as entry points to the application so that you can analyze the resources that are used by the application and make activities such as chargebacks easier.

CICS Transaction Server V5.2 introduces CICS cloud application multi-versioning, in which you can have numerous versions of your application running concurrently, and bring a new version of an application into your production environment without service outages. You also can roll back versions of the application without major service interruptions. You can use multi-versioning to release new application functions to just a subset of your users and prepare and test the installation of the new application without deprovisioning the in-service application. You can also maintain support for particular versions of applications for migrating users.

For more information about CICS cloud applications, see *Cloud Enabling IBM CICS*, SG24-8114.

7.2 The scenario that is used in this chapter

The Policy Search GENAPP Extension with Cloud Enablement is used to illustrate the DevOps concepts that are described in this chapter. This scenario uses the following tools:

- CICS Explorer
- Rational Team Concert
- Apache Ant
- IBM CICS Transaction Server build toolkit (CICS build toolkit)
- DFHDPLOY

This GENAPP extension comes with two application bindings to demonstrate deploying the application to a single-region platform or a multi-region platform. When you use the single-region platform, the application binding installs all of the application into one region. When using the multi-region platform, the application binding installs each tier into a different region type. It also installs remote programs to allow program calls between regions.

You can download the GENAPP extension from this website and import it in CICS Explorer:

`http://www.ibm.com/support/docview.wss?uid=swg24031760`

For this scenario, we primarily use the CICS Cloud perspective in CICS Explorer, which provides views to develop CICS cloud applications. You will see how variables can be extracted to make the application as portable to different environments as possible, and the code changes delivered to Rational Team Concert.

Rational Team Concert is used as a CI server and the Jazz Build Engine is used as a build engine. Apache Ant is the build utility, so you can start the CICS build toolkit for the build.

When deploying the application, JCL scripting provides the automation, which calls the CICS build toolkit to resolve the application's variables for the intended platform. Finally, the DFHDEPLOY utility is used to deploy the CICS cloud application to the target environment.

Figure 7-1 on page 157 shows the full CICS cloud application DevOps scenario for this chapter.

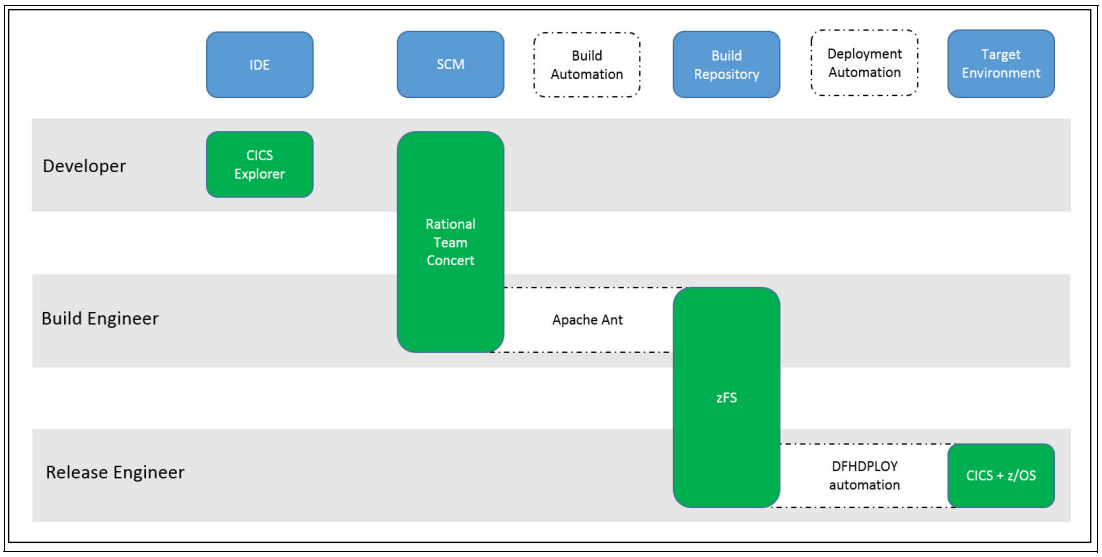


Figure 7-1 CICS cloud application DevOps scenario

We make several changes to the application that we are deploying to make it suitable for multiple environments:

- The CEDF transaction is a useful tool in development environments but is disruptive to testing and production environments. Use variable substitution to enable only CEDF for program definitions in the development environment and disable it in the testing environment.
- The name of the JVM server where OSGi bundles are installed might differ between environments. Use variable substitution to use different JVM server names in different environments.

It is also important to know which actions are performed by the individuals filling each role in the process:

- Application developer:
  - Use CICS Explorer wizards to change the CEDF and JVMSERVER program definition attributes to reference variables.
  - Set the variable default for development.
  - Check in changes to the source code management (SCM) system and request an automated build and deployment.
- Build engineer: Write build automation to run the CICS build toolkit that builds the application binding and related artifacts.
- Release engineer:
  - Create application bindings for the test environment and other environments
  - Write automation to run the CICS build toolkit that resolves the application by using properties for the test environment
  - Write automation to run the DFHDPLOY utility that undeploys the old version of the application and deploys the new, updated one

This scenario also demonstrates the architecture of the build and deployment processes.

The automated build process uses the SCM build engine and the CICS build toolkit's build command to build the application bundle directly in the location on zFS that is used as a build repository. The deployment process uses the CICS build toolkit's resolve command to resolve different variables in the resource definitions (for example, CEDF and JVM Server name attributes), and then uses the DFHDPLOY utility to deploy the application bundle to the target environment.

Figure 7-2 shows the scenario's build and deployment architecture.

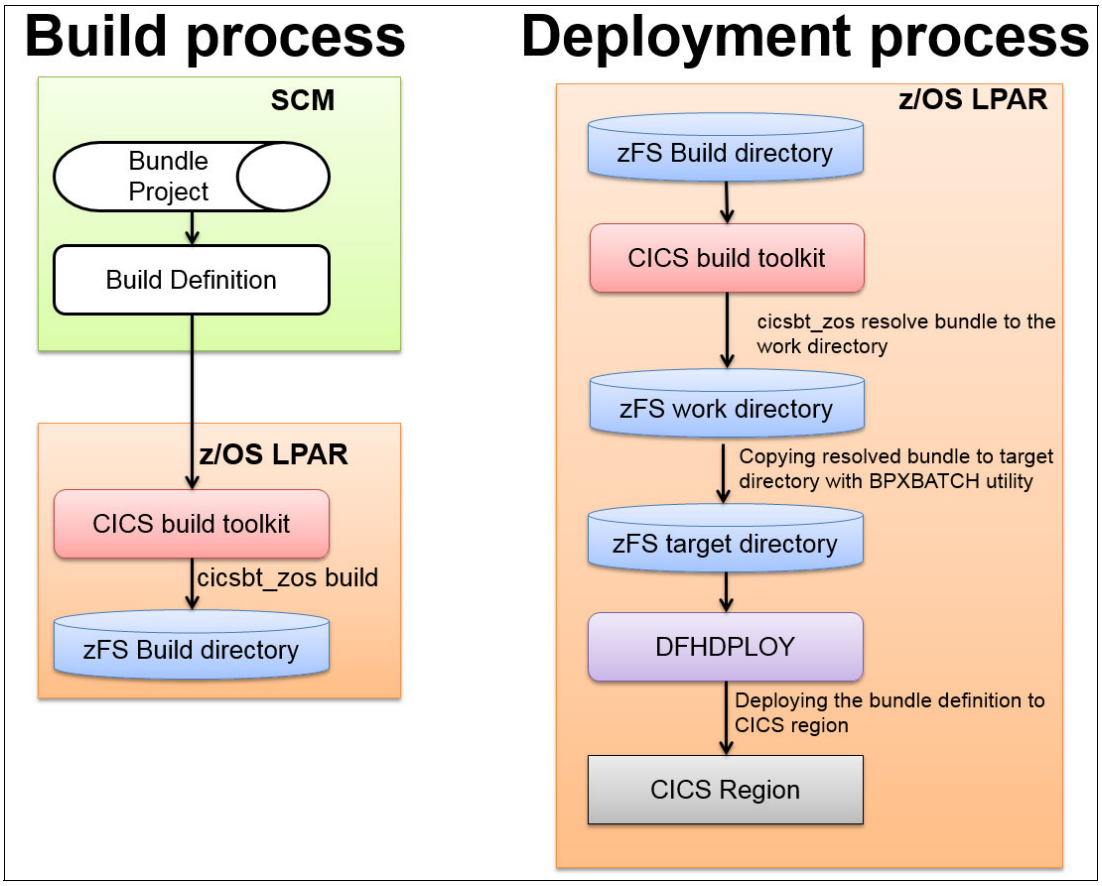


Figure 7-2 Build and deployment architecture for CICS cloud application DevOps scenario

7.3 Using the Policy Search GENAPP Extension with Cloud Enablement

The Policy Search GENAPP Extension with Cloud Enablement demonstrates how the support for cloud applications in CICS allows the resources that compose an application to be deployed together, across a CICSPlex, and then managed as part of a combined lifecycle. Without changing the application, it can be bound to different topologies, which are known as *platforms*.

The application consists of three tiers. The first tier installs transaction SSC2, which is a 3270 terminal transaction that takes input and displays the associated policies. The middle tier installs a Java program that takes the input request in a COMMAREA and moves it into a container, and vice versa. The final tier installs a Java program that interacts with a DB2 database by using JDBC. When using the single-region platform, the application binding simply installs the entire application into one region. When using the multi-region platform, the application binding installs each tier into a different region type. It also installs remote programs to allow program calls between regions.

7.3.1 Importing the GENAPP extension projects into CICS Explorer

The GENAPP extension provides the application, application binding, platform, and CICS bundle projects. To start developing with these projects, download the extension from the location that is described in 7.2, “The scenario that is used in this chapter” on page 156. Then, in CICS Explorer, click **File** → **Import** and select **Existing Projects into Workspace** to import all the discovered projects (see Figure 7-3).

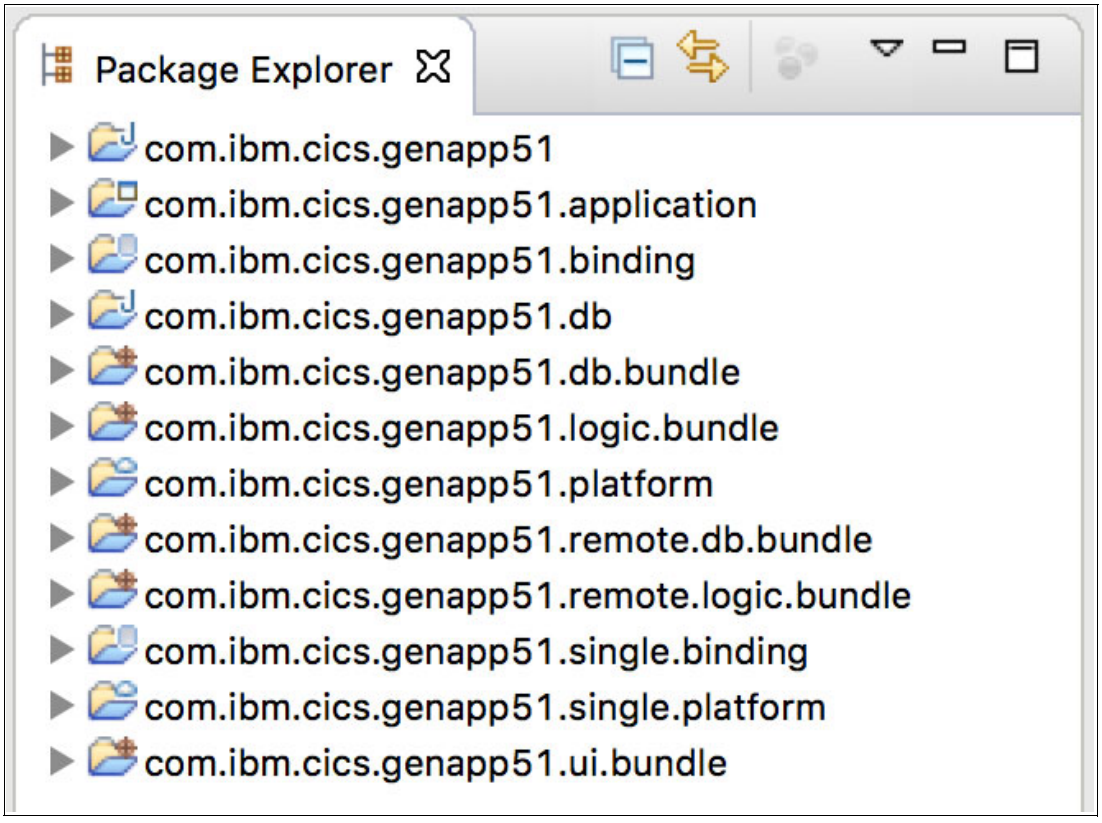


Figure 7-3 Projects that are imported from the GENAPP extension

To use the projects in an automated build and deployment pipeline, they must be shared to an SCM system. Right-click the projects and select **Team** → **Share Project**. Several different options are presented, depending on the source control plug-ins that are installed in CICS Explorer. For Rational Team Concert, select **Jazz Source Control** and choose a repository workspace and component to which to share (see Figure 7-4).

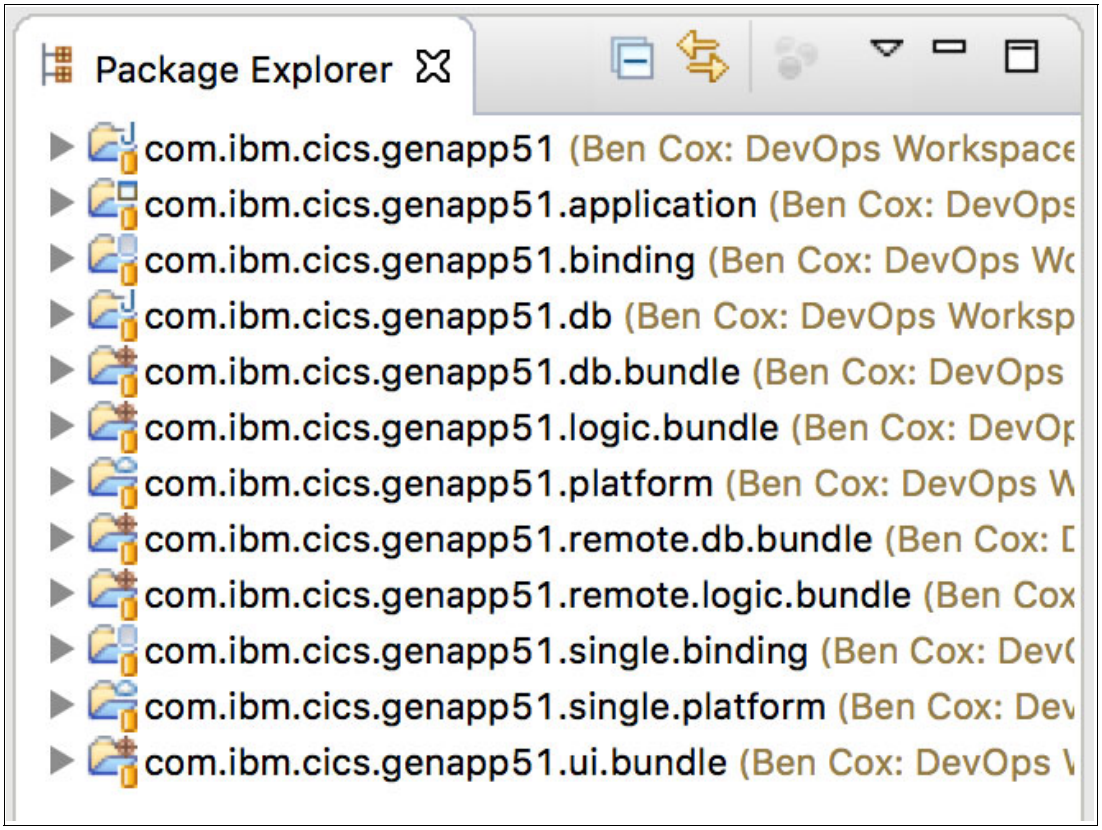


Figure 7-4 The projects that are shared to SCM - the repository icon decorator indicates that they are shared

With the projects shared to the SCM system, developers can now work on the application in their own workspaces.

During development, you can test the application by using tools in the CICS Cloud perspective of CICS Explorer. Right-click the listing for the application or application binding project and select **Export Application/Application Binding Project to z/OS UNIX File System**. Then, create the application definition and install the application.

7.3.2 Setting up the target platform for team development

The Java projects in the application are failing to compile because they have missing dependencies.

Developing OSGi-based applications in CICS Explorer (either for OSGi JVM servers or Liberty JVM servers with OSGi applications) involves the *target platform*. The target platform provides a set of OSGi bundles (and associated Java interfaces and classes) that are assumed to exist in a particular environment. By using the target platform, the applications can be compiled for each specific environment by using the APIs that the environment provides. It also provides many built-in target platform templates for each of the environments that CICS Transaction Server provides, including OSGi JVM servers and Liberty JVM servers, at each release of CICS Transaction Server.

Target platforms can consist of more than just the APIs for CICS Transaction Server, OSGi, and Liberty. When you use an OSGi JVM server, setting the environment variable OSGI_BUNDLES in the JVM profile allows additional OSGi bundles to be installed at the start of the JVM server, which means that applications can assume that they exist and use their APIs. Common OSGi bundles that are added to the OSGI_BUNDLES environment variable include the OSGi bundles for connecting to IBM DB2, such as `db2jcc4.jar` and `db2jcc_license_cisuz.jar`. The GENAPP extension needs the DB2 OSGi bundles to exist in the JVM server, so the target platform must be altered to include them for development purposes and to allow it to be built.

To store these external OSGi bundles, complete the following steps:

1. Create a project by clicking **New** → **Project**, click **General** → **Project** , and name the new project Target Platform.
2. Add a folder that is called `bundles` to the project and add the OSGi bundles to the folder.

With the dependencies available in the workspace, you can create and apply a target platform for CICS Transaction Server V5.3 and those dependencies. Complete the following steps:

1. Open the **Preferences** menu and click **Plug-in Development** → **Target Platform**.
2. Click **Add** to add a target definition and use the wizard to choose a platform template. The list includes CICS Transaction Server V5.3 and prior target platform definitions for OSGi JVM servers, and if the Eclipse Web Tools Platform is installed, it also shows the Liberty JVM server target definitions.
3. Click **Next** and CICS Explorer shows the content of the selected definition.

4. Rename the target platform to “CICS Transaction Server V5.3 with DB2”, as shown in Figure 7-5.

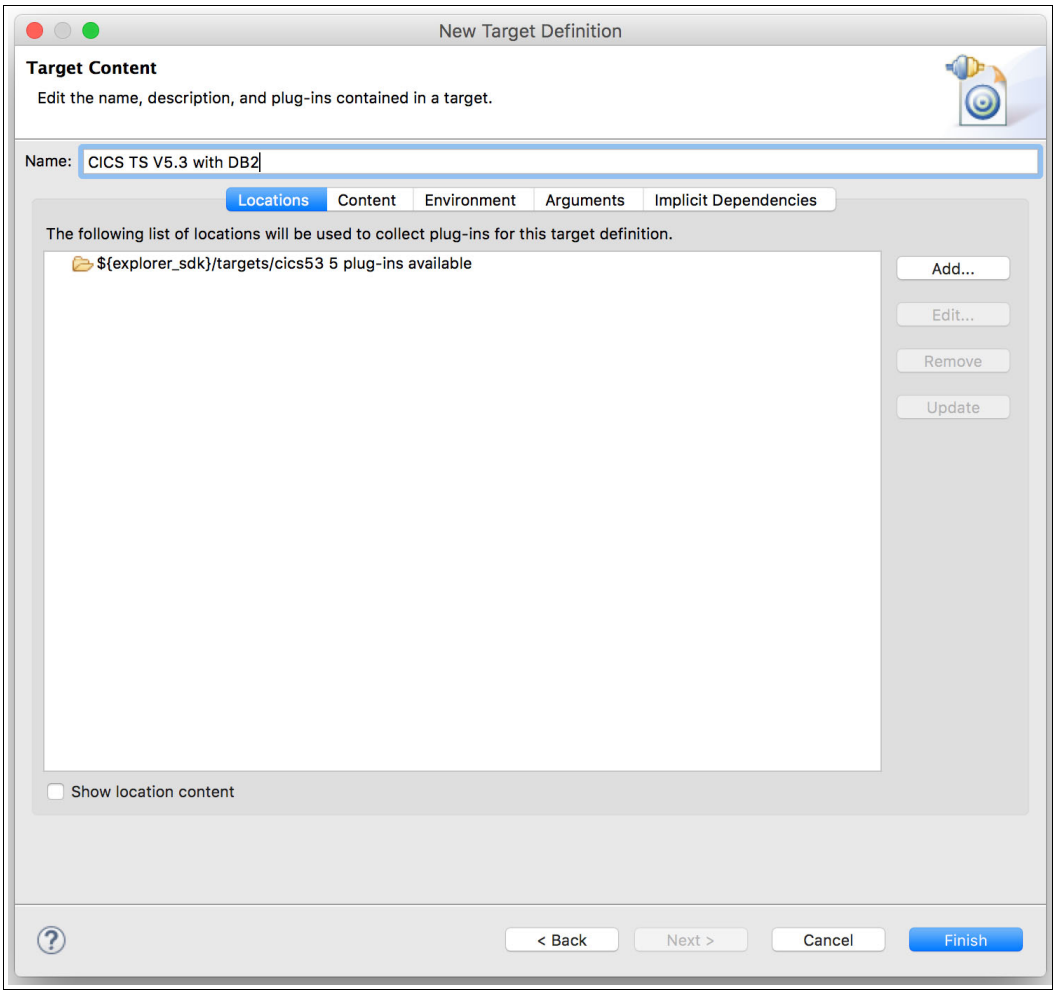


Figure 7-5 Rename a target definition that is produced from a template

5. The target definition also must refer to the DB2 bundles. Click **Add** to open a dialog box and select **Directory** (Figure 7-6) because the bundles are in a workspace directory.

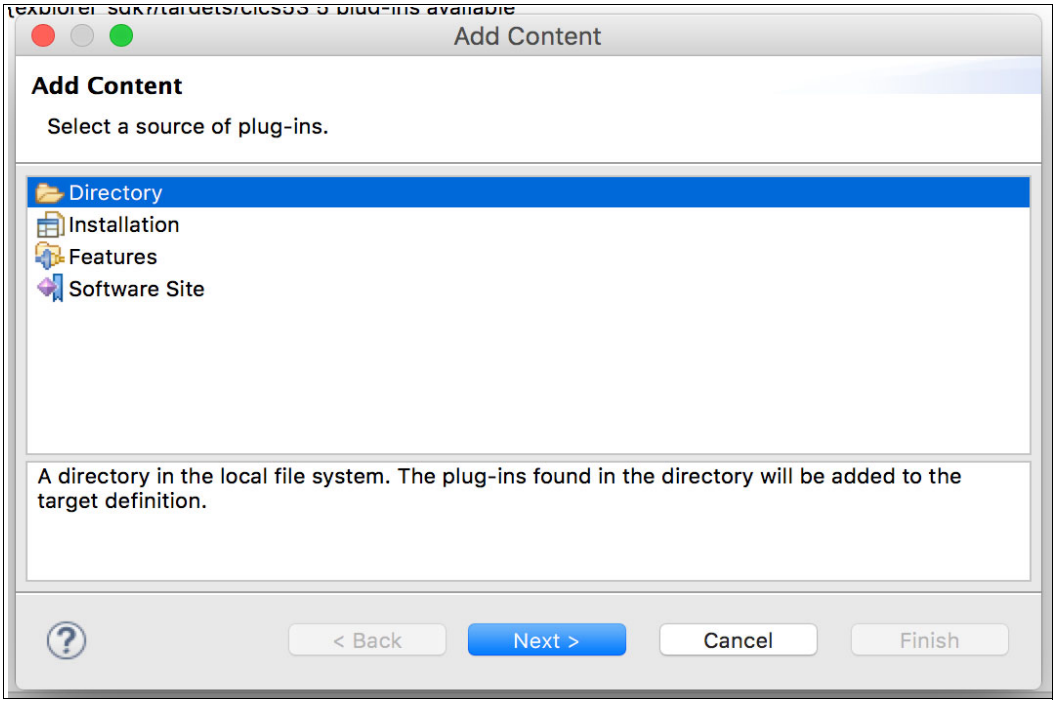


Figure 7-6 Add directory content to the target definition

6. In this scenario, we want our target definition to be flexible so that developers can use it on different machines and with automation features, such as those provided with the CICS build toolkit. To achieve this flexibility, use a workspace variable to refer to the required bundles directory, either by entering the variable directly or clicking **Variables** and following the prompts. The *resource_loc* variable, which is predefined in Eclipse and can be used within both CICS Explorer and the CICS build toolkit, resolves the location of the argument workspace-relative path to an absolute file path (see Figure 7-7).

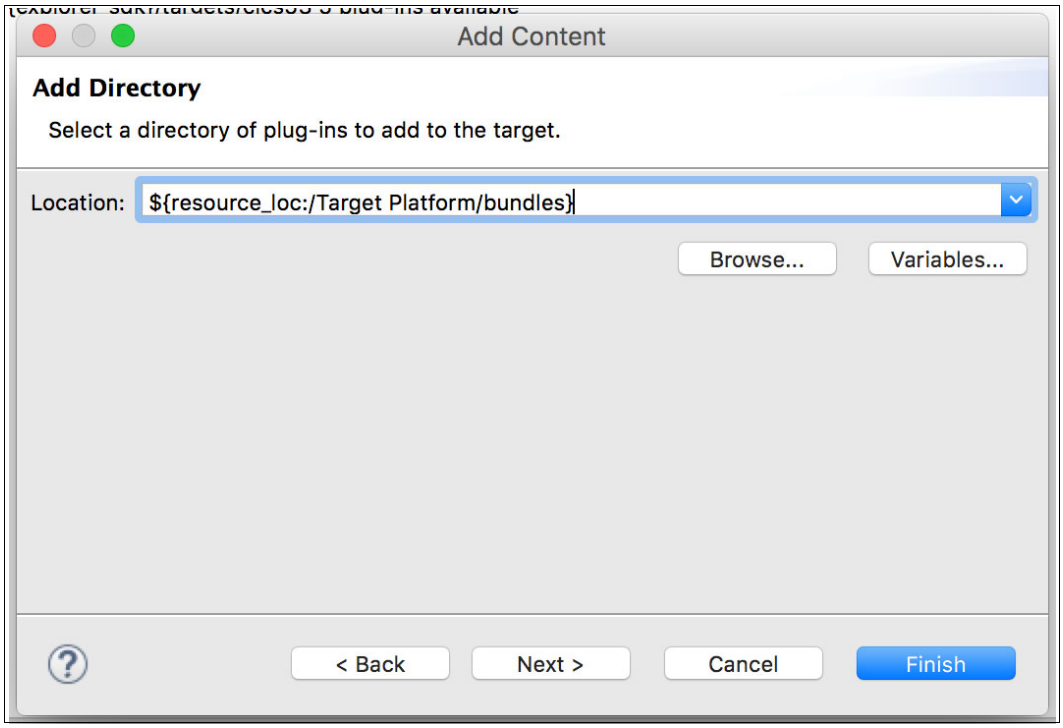


Figure 7-7 Specify the directory in an environment-neutral manner

7. With the workspace variable designated, click **Next** to see the OSGi bundles that are contained in the directory. In our example, there is just the `com.ibm.db2.jcc` bundle. Complete the steps in the Add Content window and then click the **Content** tab of the new target definition wizard (see Figure 7-8). Select the **com.ibm.db2.jcc** check box to ensure that the bundle is used by the target definition, and then click **Finish**.

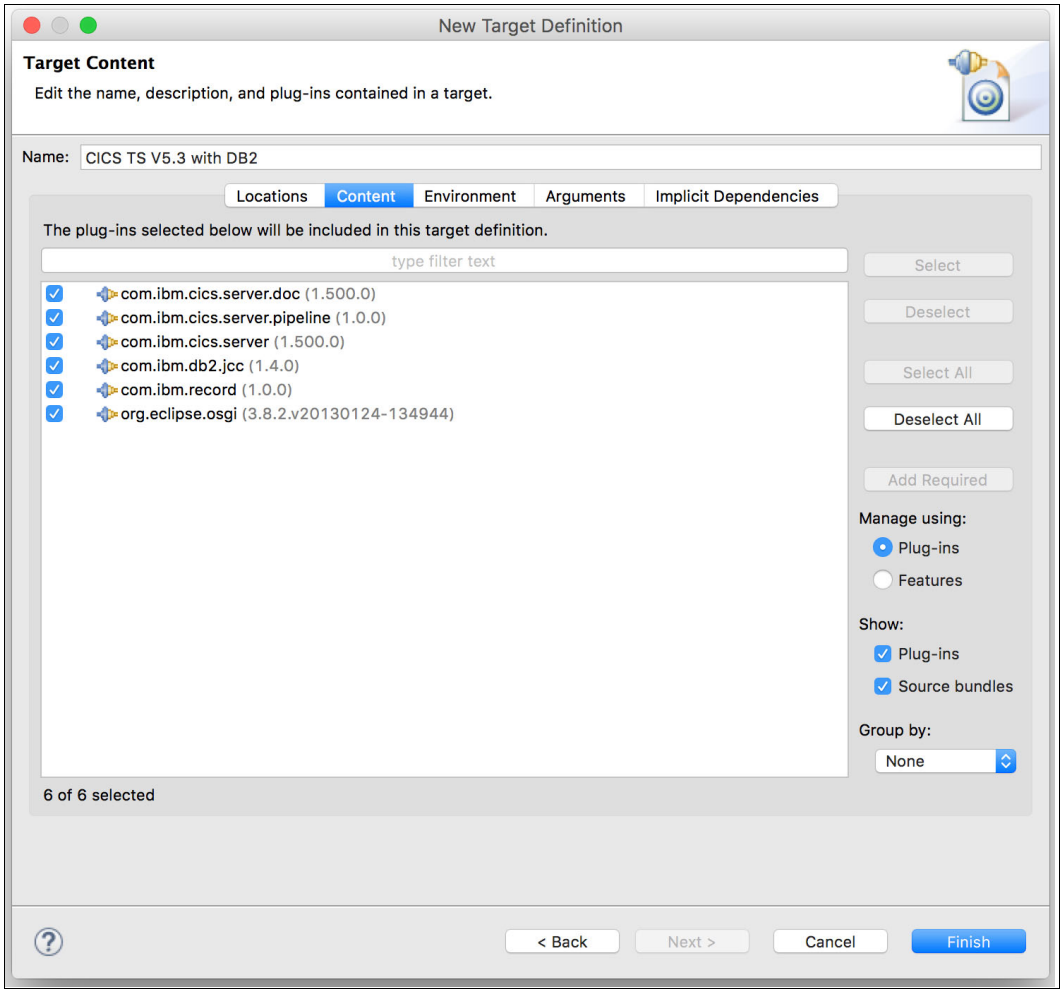


Figure 7-8 Ensure that the DB2 bundle is selected in the target content

8. With the bundle selected, the new target platform appears in the list of all target platforms. Select the check box beside the new target platform and click **Apply** (see Figure 7-9). The workspace is rebuilt by using the new target platform.

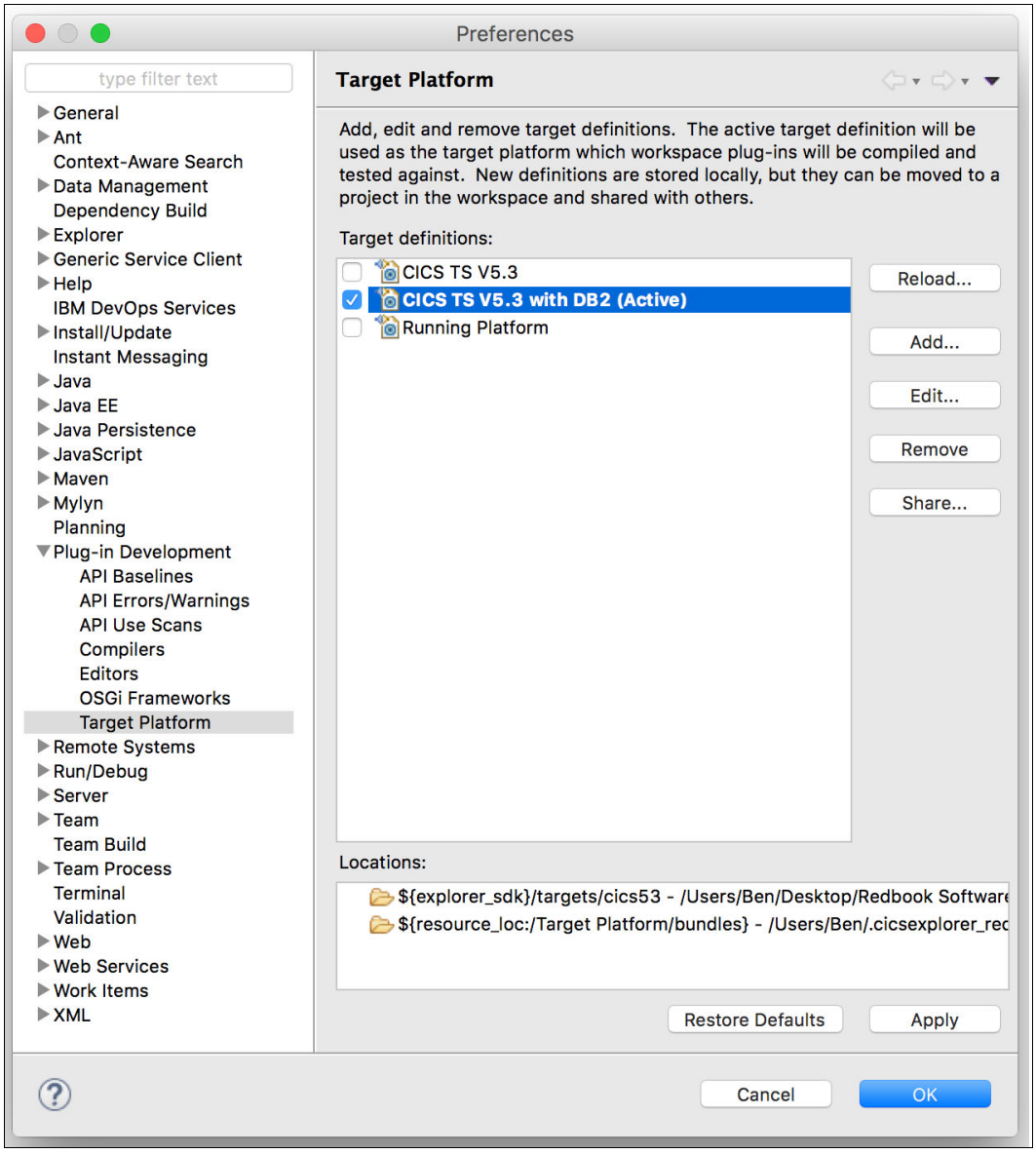


Figure 7-9 The new target platform before it is shared into a project

9. The target platform is held locally by the workspace of CICS Explorer. It must be shared, by using source control, so that others can access it. To do this task, click **Share** to choose where to put it, and then select the Target Platform project.
10. Give the file a name, such as `CICS Transaction Server V5.3 with DB2.target`, and put it in the Target Platform project. A new target definition file is created in that project.
11. The Target Platform project and its content must now be checked in to the SCM system, Now, other developers can develop with a consistent target platform. They simply must check out the Target Platform project, open the target definition file, and click **Set as Target Platform** (see Figure 7-10 on page 167).

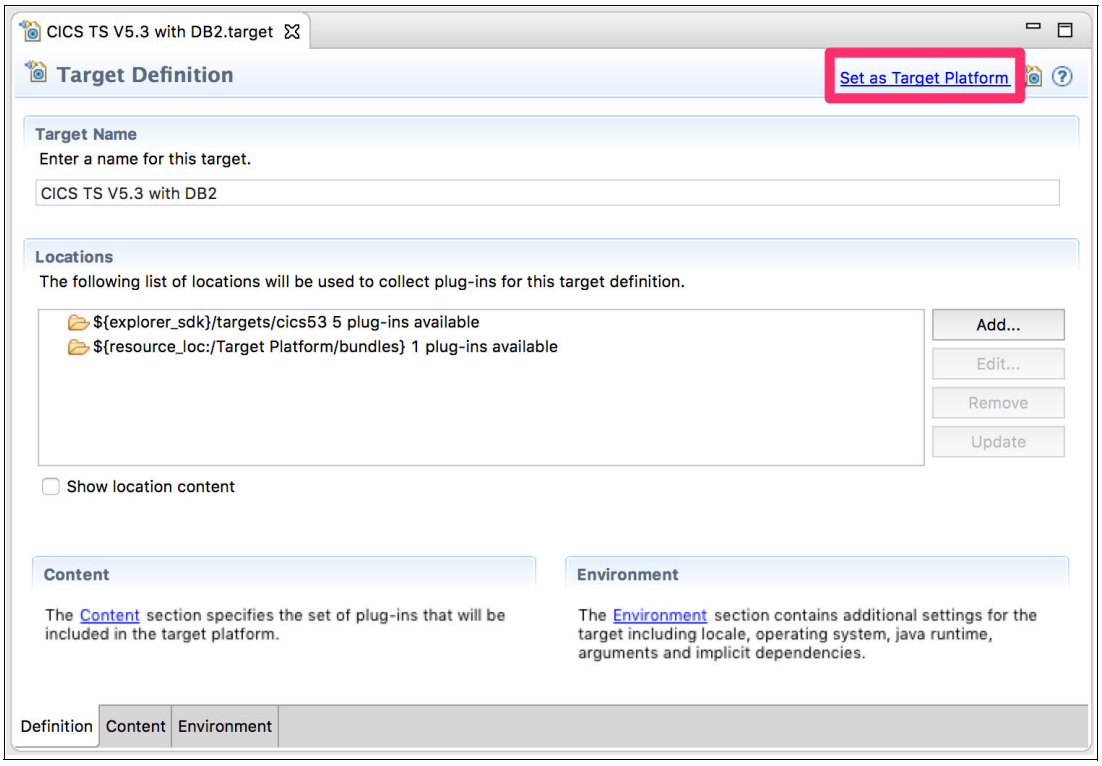


Figure 7-10 Set the target platform from the target definition file

Later in the development pipeline, when automating the build by using the CICS build toolkit, you load the Target Platform from the SCM system alongside the application, and pass the location of the target definition file to the CICS build toolkit by using the **`--target`** argument. In this way, both development and build use the same target platform, reducing errors caused by inconsistent platform definitions.

7.4 Making changes in development

To get started with changing the application, complete the following steps:

1. Open the CICS Explorer Cloud perspective, as shown in Figure 7-11, in which the GENAPP extension application bundles are displayed. Update both bundles by using variable substitution.

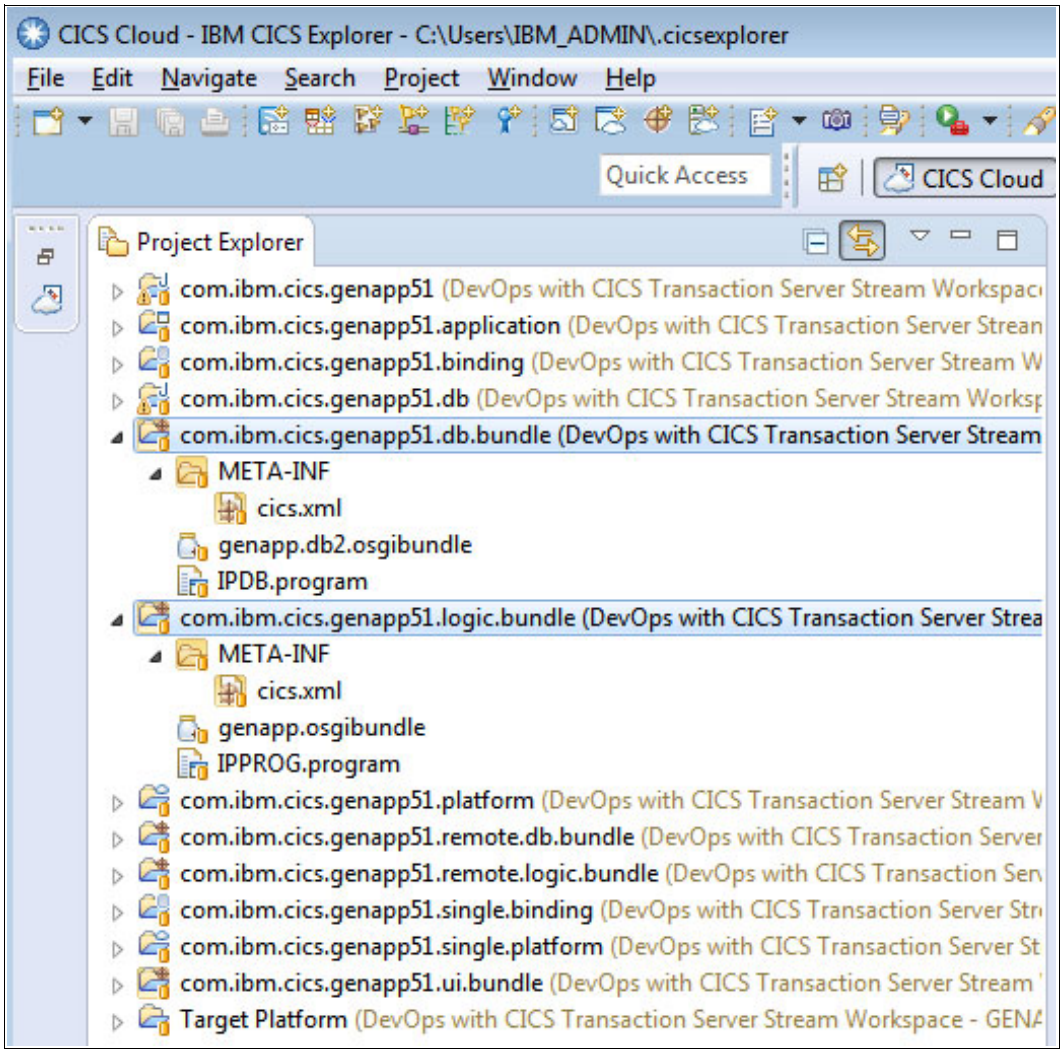


Figure 7-11 CICS Explorer Cloud perspective showing the GENAPP extension application bundles

2. Double-click the listing for `IPDB.program` and the program attributes are displayed, as shown in Figure 7-12 on page 169. CEDF is set to YES. Leave it as YES in the development environment, but set it to NO in the testing or production environments.

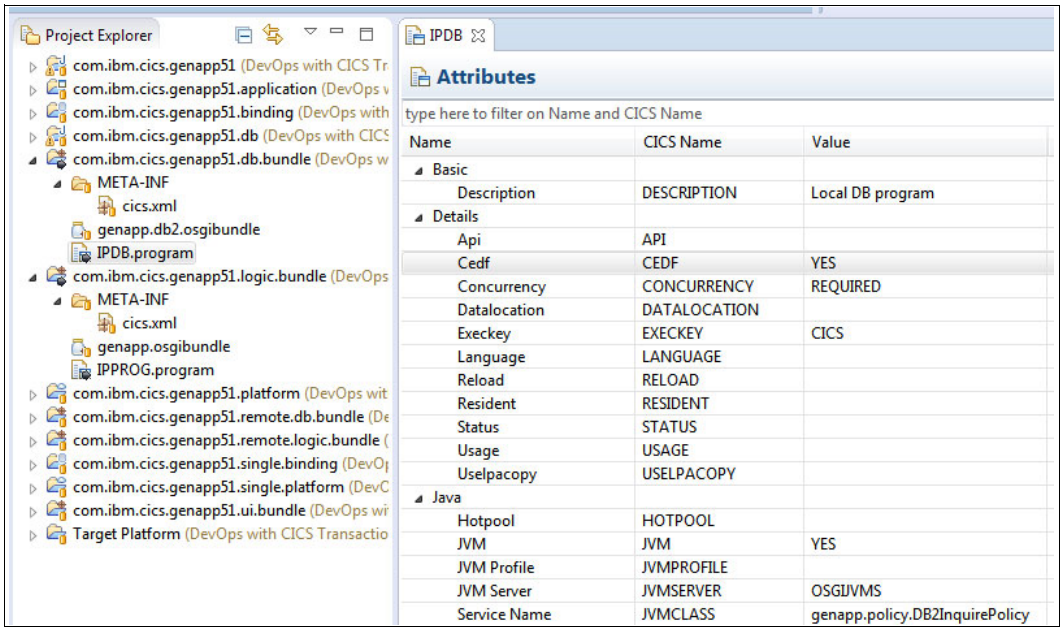


Figure 7-12 IPDB.program attributes

3. Right-click the CEDF attribute item and select **Extract value to variable**. The window that is shown in Figure 7-13 opens, where you can do variable substitution. Set CEDF(YES) for the single.binding test region and CEDF(NO) for the testing or production regions.

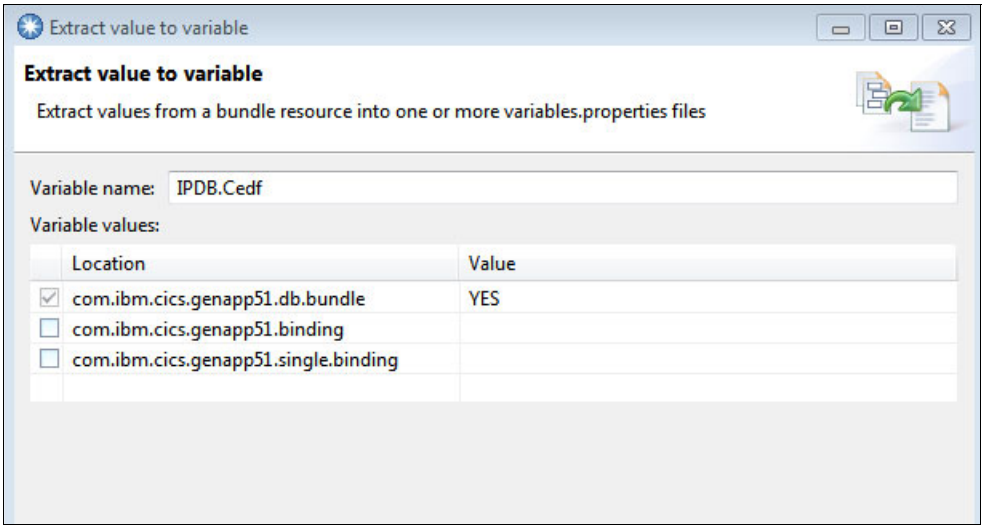


Figure 7-13 Extract a value to a variable

4. Select com.ibm.cics.geanapp51.single.binding (development) and com.ibm.cics.genapp51.binding (test or production) and set the CEDF values to NO and YES, respectively (see Figure 7-14). Click **Finish**. If you leave the single-binding option unchanged, it defaults to the db.bundle value of YES.

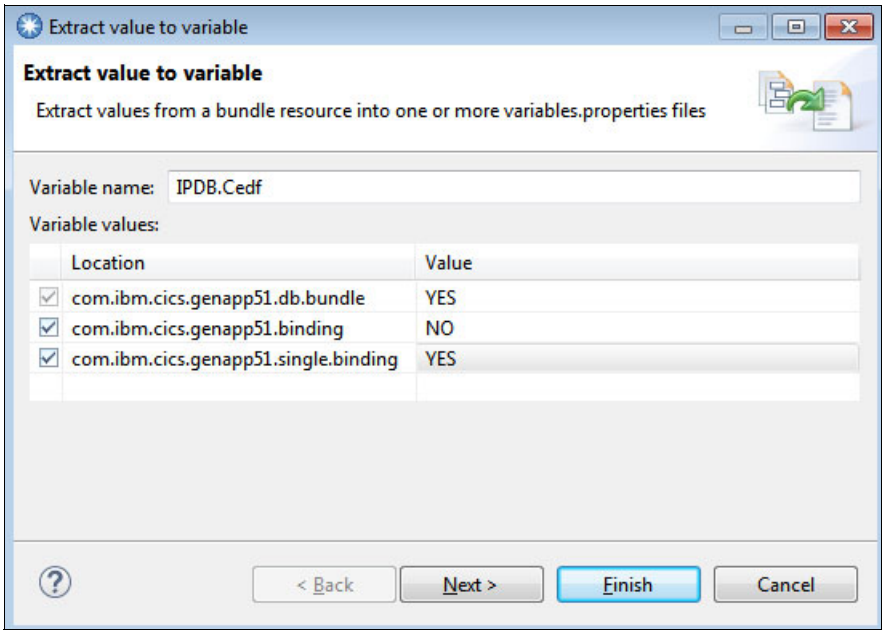


Figure 7-14 Extract value to variable changes

In Figure 7-15, a variable (*${IPDB.Cedf}*) takes the place of the CEDF attribute of the `IPDB.program` file.

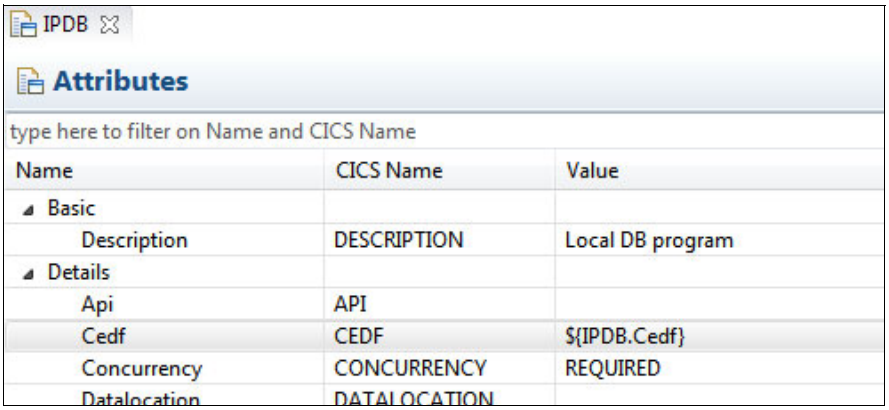


Figure 7-15 CEDF attribute with substituted variable

5. Use the same steps and apply variable substitution to the JVM server name, specifically the `db.bundle` (see Figure 7-16). For our multi-environment, we select `genapp51.binding`, give it a value of JVM1, and click **Finish**. In this case, we do not change the development application binding, com.ibm.cics.genapp51.single.binding, which allows the bundle's value to be used.

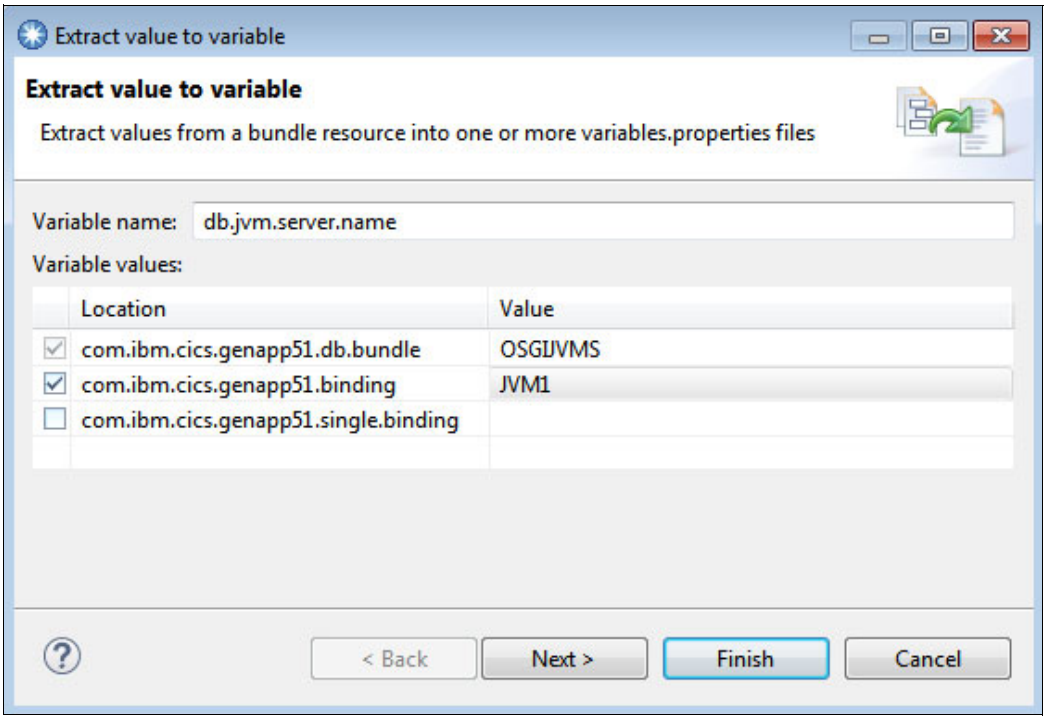


Figure 7-16 Extract value to variable for JVM1

The `IPDB.program` attributes now show value substitutions for both CEDF and JVM Server (see Figure 7-17).

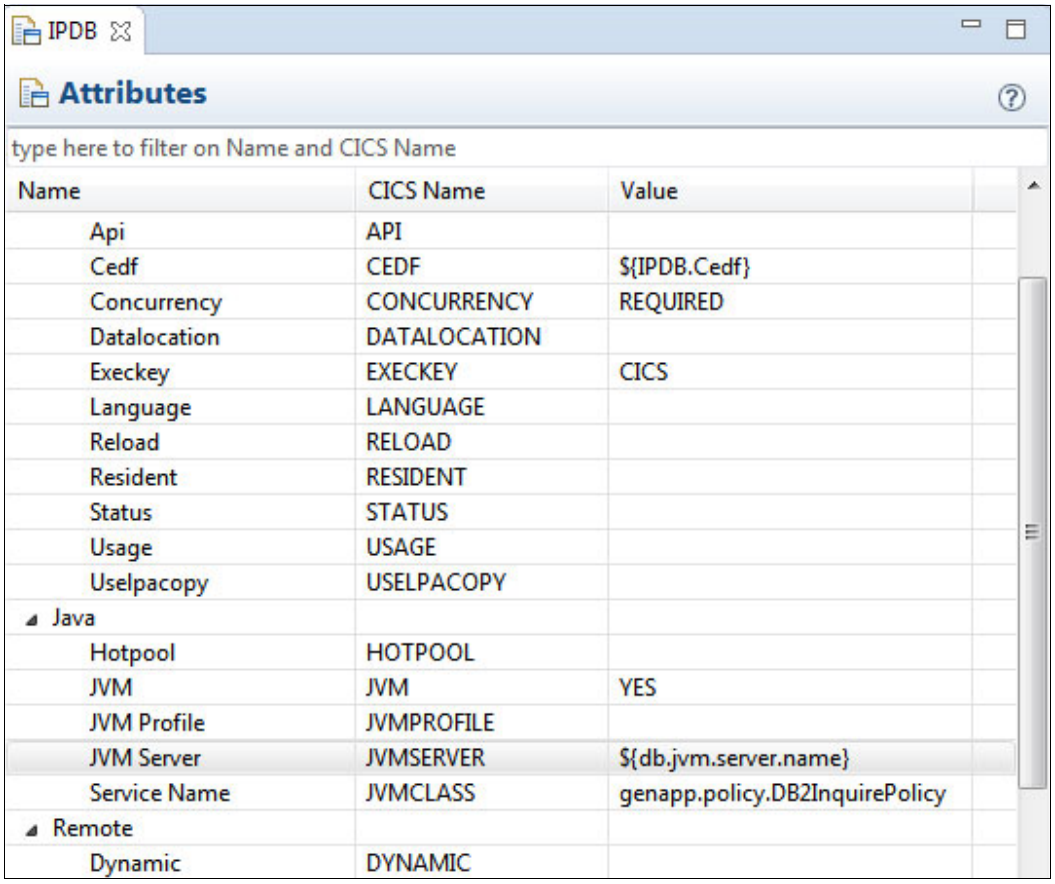


Figure 7-17 IPDB.program attributes with substituted variables

6. Change `logic.bundle` in the same way you did for the binding bundle. Figure 7-18 on page 173 shows the results. The value of the CEDF attribute is substituted with the variable *${IPPROG.Cedf}*, and the value of the JVMSERVER attribute is substituted with the variable *${logic.jvm.server.name}*. We set the value of the variable in the test application binding to JVM2 as a way to distinguish it from the other OSGi bundle, which used JVM1.

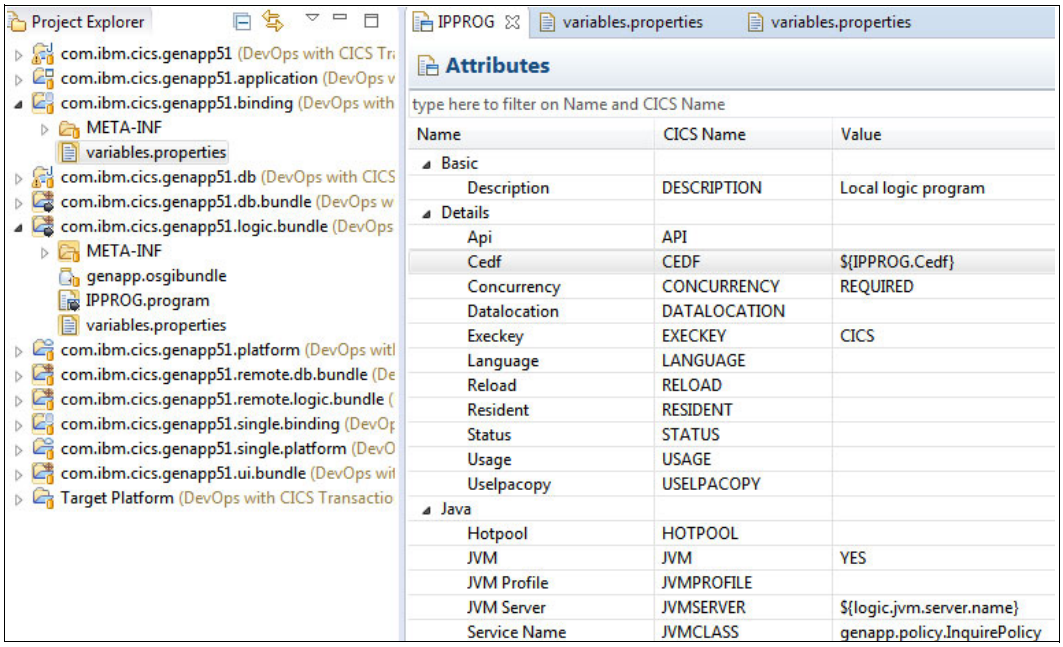


Figure 7-18 IPPROG.program attributes with substituted variables

With these changes made, you can now check them in and deliver them to SCM.

7.5 Creating an automated build definition by using Rational Team Concert and the CICS build toolkit

To provide continuous building of the application, create a Rational Team Concert build definition for it. When Rational Team Concert detects changes in the application source code, a new build is triggered.

There is no need to write new build scripts because the Ant build script that you created and used earlier in this book can also be reused to build the CICS cloud application. However, the parameters that you define on the build definition must be changed, as shown in Table 7-1. There are three changes:

- Change the bundle name to the name of the application binding.
- Change the Rational Team Concert source component to the component for this application.
- Change the target platform to use the target definition file that you created by using both CICS Transaction Server V5.3 and DB2 libraries.

Table 7-1 Build properties for the CICS cloud application

Name of property	Value	Uses
`ant.work.dir`	`workdit`	Working directory for the Ant script
`build.output.dir`	`/var/cicsts/devops/builds/${bundle.name}`	Output directory for the CICS build toolkit build command
`bundle.name`	`com.ibm.cics.genapp51.single.binding`	Bundle name to build

Name of property	Value	Uses
load.dir	${ant.work.dir}/${source.dir}	Load directory to use in the Rational Team Concert build definition
source.component	GENAPP Policy Search (Cloud)	Source component in Rational Team Concert
source.dir	source	The name of the source directory for the CICS build toolkit's build command
target.platform	"${source.dir}/${source.component}/Target Platform/CICS Transaction Server V5.3 with DB2.target"	The name of the target platform for the CICS Build toolkit build command

In this process, you use a subset of the same zFS area that you used in Chapter 6, "Applying DevOps to Java applications" on page 129 because you use zFS as a common build repository between the bundles. Be sure to read the part about the disadvantages of zFS as a build repository compared to purpose-built alternatives.

7.6 Deploying the application with the CICS build toolkit and DFHDPLOY

With new versions of the application created and placed into zFS, you can now deploy the application. In this example, deployment consists of three steps:

1. Copying the application from its location in zFS.
2. Using the CICS build toolkit's resolve command to resolve variables by using the relevant application binding.
3. Using the DFHDPLOY utility to change the lifecycle of the application.

All of these deployment steps can be run by using JCL. In this example, we use a single job to perform the entire process. The JCL of the job is shown in Example 7-1.

Example 7-1 Application deployment JCL

```
//CLODDPLY JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
//*
//EXPORT  EXPORT SYMLIST=*
//*
//     SET CPSMHLQ=CICSDPP.BETA14.CTS53BT.CPSM
//     SET CICSHLQ=CICSDPP.BETA14.CTS53BT.CICS
//     SET CICSPLEX=CICSPLEX
//     SET JAVAHOME='/java/J8.0_64/'
//     SET BUNWRKDR='/var/cicsts/devops/workdir'
//     SET BUILDDIR='/var/cicsts/devops/builds'
//     SET TARGETDR='/var/cicsts/devops/platform'
//     SET CICSBTCM='/usr/lpp/cicsts/cicsbt/cicsbt/cicsbt_zos'
//     SET BINDNAME='com.ibm.cics.genapp51.single.binding'
//     SET PLATNAME='com.ibm.cics.genapp51.single.platform'
//     SET PLATDEF='SINGPLAT'
```

```
//     SET BINDVRNM='com.ibm.cics.genapp51.single.binding_1.0.0'
//     SET APPLNAME='com.ibm.cics.genapp51.application_1.0.0'
//*
//     SET CISCOPE=CICSDA01
//*
//* First step is to copy only the bundle you want to resolve
//*
//COPY  EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
//STEPLIB DD  DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV  DD  *,SYMBOLS=JCLONLY
JAVA_HOME=&JAVAHOME
/*
//STDPARM DD  *,SYMBOLS=JCLONLY
SH
rm -r &BUNWRKDR/bundles/*;
cp -r
&BUILDDIR/&BINDNAME/*
&BUNWRKDR/bundles
//*
//* Second step is to resolve the bundle using cicsbt_zos
//*
//RESOLVE EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
//STEPLIB DD  DSN=CEE.SCEERUN,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV  DD  *,SYMBOLS=JCLONLY
JAVA_HOME=&JAVAHOME
/*
//STDPARM DD  *,SYMBOLS=JCLONLY
PGM
&CICSBTCM
  --resolve &BUNWRKDR/bundles
  --output &TARGETDR/
  --properties &TARGETDR/bundles/variables.properties
  --verbose
//*
//* Third step is DEPLOY and set available the new bundle
//*
//DEPLOY EXEC PGM=DFHDPLOY,REGION=100M
//*
//STEPLIB  DD  DISP=SHR,DSN=&CPSMHLQ..SEYUAUTH
//     DD  DISP=SHR,DSN=&CICSHLQ..SDFHLOAD
//*
//SYSTSPRT DD  SYSOUT=*
//*
//SYSIN DD  *,SYMBOLS=JCLONLY
SET CICSPLEX(&CICSPLEX);
*
*  Define a CICS bundle resource and transition it towards an
*  ENABLED state within the current CICSplex.
*
UNDEPLOY APPLICATION(GENASING)
     VERSION(1.0.0)
```

```
     SCOPE(&CISCOPE)
     PLATFORM(&PLATDEF)
     STATE(DISCARDED);
*
DEPLOY APPLICATION(GENASING)
    APPLDIR(&TARGETDR/
        &PLATNAME/
        applications/
        &APPLNAME/)
    BINDDIR(&TARGETDR/
        &PLATNAME/
        bindings/
        &BINDVRNM/)
    STATE(AVAILABLE)
```

Each of the deployment tasks is a job step: **COPY**, **RESOLVE**, and **DEPLOY**. In both the **COPY** and **RESOLVE** steps, the BPXBATCH utility is used, allowing UNIX System Services commands to be run as specified in the **STDPARM DD** statement.

Throughout the job, we rely heavily on the use of symbols to avoid repetition of variables and to allow substitution, which includes using substitution within in-stream data sets, which is possible in z/OS V2.1 and later versions. Some of these variables are related to system setup (for example, the location of Java (`JAVAHOME`), the location of the CICS build toolkit (`CICSBTCM`), and the location of CICS data sets (`CPSMHLQ` and `CICSHLQ`)), and are likely to be shared across all installation scripts. However, other variables can change, such as those relating to the exact application, platform, and application binding that we are targeting. By passing different parameters to this script, different applications can be deployed.

The third step of this script, in which **DFHDPLOY** runs, undeploys the existing application before deploying the new application. Be careful because these steps can result in a small loss of service for the application. However, if application multi-versioning is used, both applications can coexist simultaneously, with the higher-versioned application taking the workload when it is available and the lower-versioned application being removed when appropriate, such as when another higher-versioned application is deployed or sufficient testing of the higher-versioned application is conducted. In this example, the application is not suitable for multi-versioning, thus leading to a loss of service.

When the job is run, the output looks like what is shown in Example 7-2.

Example 7-2 Output of the application deployment job

```
1          J E S 2 J O B L O G -- S Y S T E M M V H 1 -- N O D E W I N M V S H O
0
 16.14.08 JOB48117 ---- WEDNESDAY, 02 DEC 2015 ----
 16.14.08 JOB48117 IRR010I USERID REDS01  IS ASSIGNED TO THIS JOB.
 16.14.08 JOB48117 ICH70001I REDS01  LAST ACCESS AT 16:14:07 ON WEDNESDAY, DECEMBER 2, 2015
 16.14.08 JOB48117 $HASP373 CLODDPLY STARTED - INIT 12  - CLASS A    - SYS MVH1
 16.14.08 JOB48117 IEF403I CLODDPLY - STARTED
 16.14.11 JOB48117 -                     --TIMINGS (MINS.)--      ----PAGING COUNTS---
 16.14.11 JOB48117 -JOBNAME STEPNAME PROCSTEP  RC  EXCP  CPU  SRB CLOCK  SERV PG  PAGE  SWAP
VIO SWAPS STEPNO
 16.14.11 JOB48117 -CLODDPLY     COPY    00   52  .00  .00  .05  315  0   0   0   0   0   1
 16.14.25 JOB48117 -CLODDPLY     RESOLVE   00   74  .00  .00  .22  366  0   0   0   0   0   2
 16.14.43 JOB48117 -CLODDPLY     DFHDPLOY  00  421  .00  .00  .30  3476  0   0   0   0   0   3
 16.14.43 JOB48117 IEF404I CLODDPLY - ENDED
```

```
 16.14.43 JOB48117 -CLODDPLY ENDED. NAME-           TOTAL CPU TIME=  .00 TOTAL ELAPSED TIME=
.59
 16.14.43 JOB48117 $HASP395 CLODDPLY ENDED
0------ JES2 JOB STATISTICS ------
- 02 DEC 2015 JOB EXECUTION DATE
-      85 CARDS READ
-     187 SYSOUT PRINT RECORDS
-       0 SYSOUT PUNCH RECORDS
-      11 SYSOUT SPOOL KBYTES
-     0.59 MINUTES EXECUTION TIME

!! END OF JESMSGLG SPOOL FILE !!
    1 //CLODDPLY JOB CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID            JOB48117
     //*
     IEFC653I SUBSTITUTION JCL - CLASS=A,MSGCLASS=A,NOTIFY=REDS01
    2 //EXPORT  EXPORT SYMLIST=*
     //*
    3 //     SET CPSMHLQ=CICSDPP.BETA14.CTS53BT.CPSM
    4 //CPSMHLQ EXPORT EXPSET=CICSDPP.BETA14.CTS53BT.CPSM GENERATED STATEMENT
    5 //     SET CICSHLQ=CICSDPP.BETA14.CTS53BT.CICS
    6 //CICSHLQ EXPORT EXPSET=CICSDPP.BETA14.CTS53BT.CICS GENERATED STATEMENT
    7 //     SET CICSPLEX=CICSPLEX
    8 //CICSPLEX EXPORT EXPSET=CICSPLEX          GENERATED STATEMENT
    9 //     SET JAVAHOME='/java/J8.0_64/'
    10 //JAVAHOME EXPORT EXPSET=/java/J8.0_64/       GENERATED STATEMENT
    11 //     SET BUNWRKDR='/var/cicsts/devops/workdir'
    12 //BUNWRKDR EXPORT EXPSET=/var/cicsts/devops/workdir GENERATED STATEMENT
    13 //     SET BUILDDIR='/var/cicsts/devops/builds'
    14 //BUILDDIR EXPORT EXPSET=/var/cicsts/devops/builds  GENERATED STATEMENT
    15 //     SET TARGETDR='/var/cicsts/devops/platform'
    16 //TARGETDR EXPORT EXPSET=/var/cicsts/devops/platform GENERATED STATEMENT
    17 //     SET CICSBTCM='/usr/lpp/cicsts/cicsbt/cicsbt/cicsbt_zos'
    18 //CICSBTCM EXPORT EXPSET=/usr/lpp/cicsts/cicsbt/c... GENERATED STATEMENT
    19 //     SET BINDNAME='com.ibm.cics.genapp51.single.binding'
    20 //BINDNAME EXPORT EXPSET=com.ibm.cics.genapp51.si... GENERATED STATEMENT
    21 //     SET PLATNAME='com.ibm.cics.genapp51.single.platform'
    22 //PLATNAME EXPORT EXPSET=com.ibm.cics.genapp51.si... GENERATED STATEMENT
    23 //     SET PLATDEF='SINGPLAT'
    24 //PLATDEF EXPORT EXPSET=SINGPLAT          GENERATED STATEMENT
    25 //     SET BINDVRNM='com.ibm.cics.genapp51.single.binding_1.0.0'
    26 //BINDVRNM EXPORT EXPSET=com.ibm.cics.genapp51.si... GENERATED STATEMENT
    27 //     SET APPLNAME='com.ibm.cics.genapp51.application_1.0.0'
    28 //APPLNAME EXPORT EXPSET=com.ibm.cics.genapp51.ap... GENERATED STATEMENT
     //*
    29 //     SET CISCOPE=CICSDA01
    30 //CISCOPE EXPORT EXPSET=CICSDA01          GENERATED STATEMENT
     //*
     //* First step is to copy only the bundle you want to resolve
     //*
    31 //COPY  EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
    32 //STEPLIB DD  DSN=CEE.SCEERUN,DISP=SHR
    33 //STDOUT DD SYSOUT=*
    34 //STDERR DD SYSOUT=*
    35 //STDENV  DD  *,SYMBOLS=JCLONLY
    36 //STDPARM DD  *,SYMBOLS=JCLONLY
```

```
     //*
     //* Second step is to resolve the bundle using cicsbt_zos
     //*
    37 //RESOLVE EXEC PGM=BPXBATCH,REGION=0M,MEMLIMIT=6G
    38 //STEPLIB DD  DSN=CEE.SCEERUN,DISP=SHR
    39 //STDOUT DD SYSOUT=*
    40 //STDERR DD SYSOUT=*
    41 //STDENV  DD  *,SYMBOLS=JCLONLY
    42 //STDPARM DD  *,SYMBOLS=JCLONLY
     //*
     //* Third step is DEPLOY and set available the new bundle
     //*
    43 //DFHDPLOY EXEC PGM=DFHDPLOY,REGION=100M
     //*
    44 //STEPLIB  DD  DISP=SHR,DSN=&CPSMHLQ..SEYUAUTH
     IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=CICSDPP.BETA14.CTS53BT.CPSM.SEYUAUTH
    45 //     DD  DISP=SHR,DSN=&CICSHLQ..SDFHLOAD
     //*
     IEFC653I SUBSTITUTION JCL - DISP=SHR,DSN=CICSDPP.BETA14.CTS53BT.CICS.SDFHLOAD
    46 //SYSTSPRT DD  SYSOUT=*
     //*
    47 //SYSIN DD  *,SYMBOLS=JCLONLY

!! END OF JESJCL SPOOL FILE !!
 ICH70001I REDS01  LAST ACCESS AT 16:14:07 ON WEDNESDAY, DECEMBER 2, 2015
 IEF236I ALLOC. FOR CLODDPLY COPY
 IEF237I BC01 ALLOCATED TO STEPLIB
 IEF237I JES2 ALLOCATED TO STDOUT
 IEF237I JES2 ALLOCATED TO STDERR
 IEF237I JES2 ALLOCATED TO STDENV
 IEF237I JES2 ALLOCATED TO STDPARM
 IEF142I CLODDPLY COPY - STEP WAS EXECUTED - COND CODE 0000
 IEF285I  CEE.SCEERUN                 KEPT
 IEF285I  VOL SER NOS= PHSY02.
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000106.?     SYSOUT
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000107.?     SYSOUT
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000101.?     SYSIN
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000102.?     SYSIN
 IEF373I STEP/COPY  /START 2015336.1614
 IEF032I STEP/COPY  /STOP 2015336.1614
     CPU:   0 HR 00 MIN 00.01 SEC  SRB:   0 HR 00 MIN 00.00 SEC
     VIRT:  164K SYS:  296K EXT:   132K SYS:  10540K
     ATB- REAL:          16K SLOTS:           0K
       VIRT- ALLOC:  2409M SHRD:    0M
 IEF236I ALLOC. FOR CLODDPLY RESOLVE
 IEF237I BC01 ALLOCATED TO STEPLIB
 IEF237I JES2 ALLOCATED TO STDOUT
 IEF237I JES2 ALLOCATED TO STDERR
 IEF237I JES2 ALLOCATED TO STDENV
 IEF237I JES2 ALLOCATED TO STDPARM
 IEF142I CLODDPLY RESOLVE - STEP WAS EXECUTED - COND CODE 0000
 IEF285I  CEE.SCEERUN                 KEPT
 IEF285I  VOL SER NOS= PHSY02.
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000108.?     SYSOUT
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000109.?     SYSOUT
```

```
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000103.?     SYSIN
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000104.?     SYSIN
 IEF373I STEP/RESOLVE /START 2015336.1614
 IEF032I STEP/RESOLVE /STOP 2015336.1614
     CPU:   0 HR 00 MIN 00.01 SEC  SRB:   0 HR 00 MIN 00.00 SEC
     VIRT:  164K SYS:  304K EXT:   132K SYS:  10544K
     ATB- REAL:          16K SLOTS:           0K
       VIRT- ALLOC:  2409M SHRD:    0M
 IEF236I ALLOC. FOR CLODDPLY DFHDPLOY
 IGD103I SMS ALLOCATED TO DDNAME STEPLIB
 IGD103I SMS ALLOCATED TO DDNAME
 IEF237I JES2 ALLOCATED TO SYSTSPRT
 IEF237I JES2 ALLOCATED TO SYSIN
 IEF142I CLODDPLY DFHDPLOY - STEP WAS EXECUTED - COND CODE 0000
 IGD104I CICSDPP.BETA14.CTS53BT.CPSM.SEYUAUTH     RETAINED, DDNAME=STEPLIB
 IGD104I CICSDPP.BETA14.CTS53BT.CICS.SDFHLOAD     RETAINED, DDNAME=
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000110.?     SYSOUT
 IEF285I  REDS01.CLODDPLY.JOB48117.D0000105.?     SYSIN
 IEF373I STEP/DFHDPLOY/START 2015336.1614
 IEF032I STEP/DFHDPLOY/STOP 2015336.1614
     CPU:   0 HR 00 MIN 00.07 SEC  SRB:   0 HR 00 MIN 00.00 SEC
     VIRT:  16K SYS:  300K EXT:   2384K SYS:  10544K
     ATB- REAL:          28K SLOTS:           0K
       VIRT- ALLOC:  2409M SHRD:    0M
 IEF375I JOB/CLODDPLY/START 2015336.1614
 IEF033I JOB/CLODDPLY/STOP 2015336.1614
     CPU:   0 HR 00 MIN 00.09 SEC  SRB:   0 HR 00 MIN 00.00 SEC

!! END OF JESYSMSG SPOOL FILE !!

IBM's internal systems must only be used for conducting IBM's business
or for purposes authorized by IBM management.

>> WINMVSH1 at z/OS 1.01.00

>> Wednesday 02 December 2015, JDay: 336


!! END OF STDOUT SPOOL FILE !!
Successfully resolved /var/cicsts/devops/workdir/bundles to /var/cicsts/devops/platform

!! END OF STDOUT SPOOL FILE !!
16:14:25.360784 : DFHDPLOY CICS TS APPLICATION DEPLOYMENT 2015/12/02 4:14pm
16:14:25.363714 : SET CICSPLEX(CICSPLEX);
16:14:25.368988 : DFHRL2013I Connection to CICSPLEX(CICSPLEX) successful.
16:14:26.389328 :
16:14:26.389382 : *
16:14:26.389396 : *  Define a CICS bundle resource and transition it towards an
16:14:26.389404 : *  ENABLED state within the current CICSplex.
16:14:26.389413 : *
16:14:26.389421 : UNDEPLOY APPLICATION(GENASING)
16:14:26.389431 :      VERSION(1.0.0)
16:14:26.389440 :      SCOPE(CICSDA01)
16:14:26.389448 :      PLATFORM(SINGPLAT)
16:14:26.389458 :      STATE(DISCARDED);
```

```
16:14:29.423585 : DFHRL2081I Application(GENASING) Version(1.0.0) in
Platform(com.ibm.cics.genapp51.single.platform) does not exist.
16:14:29.434298 : DFHRL2056I CICS application definition for APPLICATION(GENASING) has been
removed.
16:14:29.434357 : DFHRL2037I UNDEPLOY command successful.
16:14:29.452419 :
16:14:29.452461 : *
16:14:29.452473 : DEPLOY APPLICATION(GENASING)
16:14:29.452482 :     APPLDIR(/var/cicsts/devops/platform/
16:14:29.452490 :         com.ibm.cics.genapp51.single.platform/
16:14:29.452499 :         applications/
16:14:29.452509 :         com.ibm.cics.genapp51.application_1.0.0/)
16:14:29.452518 :     BINDDIR(/var/cicsts/devops/platform/
16:14:29.452526 :         com.ibm.cics.genapp51.single.platform/
16:14:29.452535 :         bindings/
16:14:29.452544 :         com.ibm.cics.genapp51.single.binding_1.0.0/)
16:14:29.452553 :     STATE(AVAILABLE)
16:14:29.452561 :
16:14:30.510030 : DFHRL2044I CICS application definition(GENASING) created.
16:14:31.521346 : DFHRL2045I CICS application definition(GENASING) installed.
16:14:37.610669 : DFHRL2046I Setting application state to ENABLED.
16:14:40.660380 : DFHRL2046I Setting application state to AVAILABLE.
16:14:43.717402 : DFHRL2012I DEPLOY command completed successfully.
16:14:43.763215 :
16:14:43.763324 : DFHRL2007I Processing complete.
16:14:43.798314 : DFHRL2014I Disconnecting from CICSPLEX(CICSPLEX).

!! END OF SYSTSPRT SPOOL FILE !!
```

The output shows the three steps of the job running: **COPY** moves files into place, **RESOLVE** uses the CICS build toolkit to resolve the application binding, and **DFHDPLOY** deploys the application to CICS. In this case, the application does not exist, so the **UNDEPLOY** command to DFHDPLOY has no effect. However, if the application existed, it is removed and then the new application is installed.

The SYSTSPRT output of **DFHDPLOY** for this example is shown in Example 7-3.

Example 7-3 SYSTSPRT output of DFHDPLOY when the application exists

```
21:25:16.085229 : DFHDPLOY CICS TS APPLICATION DEPLOYMENT 2015/12/03 9:25pm
21:25:16.088220 : SET CICSPLEX(CICSPLEX);
21:25:16.104461 : DFHRL2013I Connection to CICSPLEX(CICSPLEX) successful.
21:25:17.130505 :
21:25:17.130563 : *
21:25:17.130575 : *  Define a CICS bundle resource and transition it towards an
21:25:17.130584 : *  ENABLED state within the current CICSplex.
21:25:17.130592 : *
21:25:17.130600 : UNDEPLOY APPLICATION(GENASING)
21:25:17.130610 :      VERSION(1.0.0)
21:25:17.130620 :      SCOPE(CICSDA01)
21:25:17.130628 :      PLATFORM(SINGPLAT)
21:25:17.130638 :      STATE(DISCARDED);
21:25:20.170062 : DFHRL2092I Application(com.ibm.cics.genapp51.application) Version(1.0.0) found
on Platform(com.ibm.cics.genapp51.single.platform).
21:25:20.170217 : DFHRL2064I APPLICATION(GENASING) VERSION(1.0.0) state is ENABLED and
availability is AVAILABLE.
```

```
21:25:20.170281 : DFHRL2046I Setting application state to UNAVAILABLE.
21:25:24.244958 : DFHRL2122I Quiescing tasks for application(GENASING) version(1.0.0).
21:25:26.271643 : DFHRL2091I No active tasks found for application(GENASING) version(1.0.0).
21:25:26.271756 : DFHRL2061I Quiesce of tasks for application(GENASING) version(1.0.0) completed
successfully.
21:25:26.271821 : DFHRL2046I Setting application state to DISABLED.
21:25:30.346731 : DFHRL2042I Discarding APPLICATION(GENASING).
21:25:32.388240 : DFHRL2056I CICS application definition for APPLICATION(GENASING) has been
removed.
21:25:32.388317 : DFHRL2037I UNDEPLOY command successful.
21:25:32.433415 :
21:25:32.433467 : *
21:25:32.433478 : DEPLOY APPLICATION(GENASING)
21:25:32.433488 :     APPLDIR(/var/cicsts/devops/platform/
21:25:32.433497 :         com.ibm.cics.genapp51.single.platform/
21:25:32.433505 :         applications/
21:25:32.433514 :         com.ibm.cics.genapp51.application_1.0.0/)
21:25:32.433523 :     BINDDIR(/var/cicsts/devops/platform/
21:25:32.433532 :         com.ibm.cics.genapp51.single.platform/
21:25:32.433540 :         bindings/
21:25:32.433551 :         com.ibm.cics.genapp51.single.binding_1.0.0/)
21:25:32.433560 :     STATE(AVAILABLE)
21:25:32.433570 :
21:25:33.476387 : DFHRL2044I CICS application definition(GENASING) created.
21:25:34.487638 : DFHRL2045I CICS application definition(GENASING) installed.
21:25:40.572146 : DFHRL2046I Setting application state to ENABLED.
21:25:43.641345 : DFHRL2046I Setting application state to AVAILABLE.
21:25:46.683510 : DFHRL2012I DEPLOY command completed successfully.
21:25:46.740392 :
21:25:46.740493 : DFHRL2007I Processing complete.
21:25:46.810790 : DFHRL2014I Disconnecting from CICSPLEX(CICSPLEX).
```

With this kind of automation in place, CICS cloud applications can be built and deployed without human intervention.

For brevity, in this scenario we have not covered the automatic testing that should take place at each stage of the DevOps pipeline. With automated testing, approval for changes to proceed to the next stage can be automatically assigned, with full confidence that they passed the tests.

Redbooks

CICS and DevOps: What You Need to Know

(0.2”spine)
0.17”<->0.473”
90<->249 pages

SG24-8339-00

ISBN 0738441384

Printed in U.S.A.