

Using Liberty for DevOps, Continuous Delivery, and Deployment

Sebastian Carrizo

Sorin Cucu

Moisés Domínguez García

Sima Modir



 **Cloud**

WebSphere



International Technical Support Organization

**Using Liberty for DevOps, Continuous Delivery, and
Deployment**

November 2015

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

First Edition (November 2015)

This edition applies to IBM WebSphere Application Server Liberty Version 8.5.5.7, Rational Team Concert Version 5.0.2, Rational Quality Manager Version 5.0.2, Rational Test Virtualization Server Version 8.7.0, Rational Test Workbench Version 8.7.0.1, IBM UrbanCode Deploy Version 6.1.1, and Jenkins Version 1.6.0.9.2.

© Copyright International Business Machines Corporation 2015. All rights reserved.

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

Contents

| | |
|---|------|
| Notices | vii |
| Trademarks | viii |
| IBM Redbooks promotions | ix |
| Preface | xi |
| Authors | xi |
| Now you can become a published author, too! | xiii |
| Comments welcome | xiii |
| Stay connected to IBM Redbooks | xiii |
| Chapter 1. Introduction | 1 |
| 1.1 DevOps | 2 |
| 1.1.1 DevOps architecture | 2 |
| 1.1.2 Business value | 3 |
| 1.2 WAS Liberty | 3 |
| 1.2.1 Advantages for development | 3 |
| 1.2.2 Advantages for operations | 4 |
| 1.3 DevOps application lifecycle phases | 4 |
| 1.3.1 Planning | 4 |
| 1.3.2 Development | 5 |
| 1.3.3 Build | 5 |
| 1.3.4 Testing | 5 |
| 1.3.5 Deployment | 6 |
| 1.4 Objective | 6 |
| 1.4.1 DevOps products | 7 |
| 1.5 Overview of the application use case | 7 |
| Chapter 2. Project planning by using Rational Team Concert and Rational Quality Manager | 9 |
| 2.1 Software development | 10 |
| 2.1.1 Requirements management | 10 |
| 2.1.2 Functional specifications | 10 |
| 2.1.3 Architecture and design documents | 10 |
| 2.1.4 Development | 11 |
| 2.1.5 Quality assurance | 11 |
| 2.1.6 Release management | 12 |
| 2.1.7 Documentation | 13 |
| 2.1.8 Delivery | 13 |
| 2.2 Infrastructure | 13 |
| 2.2.1 Scenario | 13 |
| 2.2.2 Monitoring and event management | 16 |
| 2.2.3 Provisioning | 16 |
| 2.3 Planning a software development project with Rational Team Concert | 17 |
| 2.3.1 The role of Rational Team Concert in DevOps | 17 |
| 2.3.2 Scrum project lifecycle and iterations mapping: Development | 17 |
| 2.3.3 User story | 25 |
| 2.4 Integrating the Eclipse Java Platform, Enterprise Edition client with Rational Team Concert and WAS Liberty | 28 |

| | |
|--|-----------|
| 2.5 Rational Quality Manager for test planning | 36 |
| 2.5.1 Acceptance criteria | 36 |
| 2.5.2 Creating a test plan for your project | 37 |
| 2.5.3 Creating test cases | 38 |
| 2.5.4 Linking the test cases to the test plan | 40 |
| 2.5.5 Creating the test scripts | 41 |
| 2.5.6 Linking test scripts to a test case | 42 |
| 2.5.7 Reviewing the test case | 42 |
| 2.5.8 Creating test case execution records | 42 |
| Chapter 3. Code development, source code management, and build | 45 |
| 3.1 How developers code and collaborate by using Rational Team Concert | 46 |
| 3.1.1 Coding | 46 |
| 3.1.2 Unit testing | 47 |
| 3.1.3 Code reviews | 49 |
| 3.1.4 Code delivery | 53 |
| 3.2 Jenkins as an alternative for building code | 53 |
| 3.2.1 Install and configure the Rational Team Concert plug-in on Jenkins | 54 |
| 3.2.2 Configure the Rational Team Concert server connection | 55 |
| 3.2.3 Create a Jenkins build engine | 55 |
| 3.2.4 Create a Jenkins build definition | 56 |
| 3.2.5 Create a Jenkins job by using the Rational Team Concert plug-in | 58 |
| Chapter 4. Continuous testing | 59 |
| 4.1 Rational Quality Manager for system testing | 60 |
| 4.1.1 Structure of the Rational Quality Manager project | 60 |
| 4.1.2 Configure the Rational Quality Manager Adapter | 62 |
| 4.1.3 Create a unit test case | 63 |
| 4.1.4 Create a unit test script | 64 |
| 4.1.5 Associate the test case and test script | 66 |
| 4.1.6 Create a test case execution record | 68 |
| 4.1.7 Execute test cases | 70 |
| 4.2 Rational Test Workbench for test virtualization | 72 |
| 4.2.1 Create and run a database stub | 73 |
| 4.2.2 Edit the data in the database stub | 75 |
| 4.3 Test environment for Liberty configuration testing | 76 |
| Chapter 5. Deployment | 77 |
| 5.1 How to use UrbanCode Deploy for application deployment | 78 |
| 5.1.1 Publishing component artifacts from Jenkins to UrbanCode Deploy | 78 |
| 5.1.2 Configuring UrbanCode Deploy for system deployment | 80 |
| 5.2 How to use Jenkins to deploy applications to IBM Bluemix | 93 |
| Chapter 6. Production environment | 97 |
| 6.1 Solution architecture | 98 |
| 6.1.1 Target environment | 99 |
| 6.1.2 DevOps platform | 99 |
| 6.1.3 Delivery pipeline | 99 |
| 6.2 Liberty profile cluster deployment with UrbanCode Deploy | 100 |
| 6.2.1 Creating a controller component | 100 |
| 6.2.2 Creating a member component | 102 |
| 6.2.3 Defining agents and component distribution | 103 |
| 6.2.4 Configuring the LibertyCloudTrader application and environment | 103 |
| 6.2.5 Adding the required configuration to the server.xml file | 105 |

| | |
|---|------------|
| 6.3 Monitoring and analytics | 106 |
| 6.3.1 Required software products | 106 |
| 6.3.2 Configuration of Data Collector | 107 |
| 6.3.3 Visualize the data in the Application Performance dashboard | 109 |
| 6.4 DevOps Services | 115 |
| 6.4.1 Plan | 115 |
| 6.4.2 Development | 116 |
| 6.4.3 Build and deploy | 116 |
| Chapter 7. Conclusion | 119 |
| 7.1 Advantages of DevOps | 120 |
| 7.2 Advantages of Liberty | 120 |
| Appendix A. Additional material | 123 |
| Locating the web material | 123 |
| Using the web material | 123 |
| Downloading and extracting the web material | 123 |
| Related publications | 125 |
| Online resources | 125 |
| Help from IBM | 126 |

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:

Bluemix™
Cognos®
Concert™
DB2®
IBM®

IBM UrbanCode™
Jazz™
PureApplication®
Rational®
Rational Team Concert™

Redbooks®
Redbooks (logo) ®
TechConnect®
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

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

Linux is a trademark of Linus Torvalds 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.

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

Find and read thousands of IBM Redbooks publications

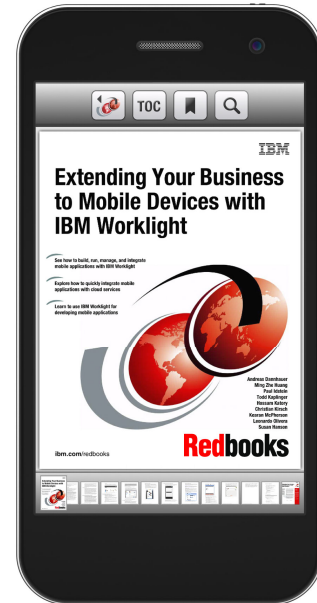
- ▶ Search, bookmark, save and organize favorites
- ▶ Get up-to-the-minute Redbooks news and announcements
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App



Download
Now

iOS



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 for an agile IT team to implement DevOps capabilities in their software delivery of a Java application. We introduce several tools that show how teams can achieve transparency, traceability, and automation in their application lifecycle to all of the stakeholders to deliver a high-quality application that meets its initial requirements. The application that is built highlights the composable and dynamic nature of the Liberty run time. The Liberty run time helps developers to get their applications up and running quickly by using only the programming model features that are required for their applications.

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

Authors

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



Sebastian Carrizo is a Web Middleware Subject Matter Expert (SME) in the IBM Delivery Center Argentina (DCA) and a Middleware SME in the Application Hosting Global Service Line. He is a foundation-level Architect and System Analyst. Sebastian has over 5 years at IBM, and his experience includes the automotive, financial services, steel, and education industries.



Sorin Cucu joined IBM in 2009 in Brno, Czech Republic, as an IBM Tivoli® Monitoring Administrator. He left IBM for an Oracle Senior Support Engineer position in Romania. After 8 months in Oracle, he returned to IBM Romania where he has worked on several projects that are focused on Middleware Products (IBM WebSphere® Application Server, WebSphere Portal, WebSphere MQ, and WebSphere Commerce). In the last 2 years, he has worked as a developer lead (Scrum master) in Cloud Managed Services for the Migration Services offering. He works in the IBM GPSG Romania Center on the Cloud Services Team where he is involved in the Agile and DevOps initiatives, focused on solution design and implementation. He is a WebSphere Application Server Certified Administrator and IBM Cognos® Business Intelligence (BI) Certified Administrator.



Moisés Domínguez García is an award-winning Lead Developer. He has experience as a back-end and front-end developer. He has presented sessions in the Dublin Lab on IBM Bluemix™, Urban Code Deploy, and Continuous Delivery. His IBM Badger project, which was developed with colleagues from IBM Dublin, won the TechConnect® 2014 in Dublin. In 2015, he published an IBM developerWorks article about Rational® License Server. He has more than 9 years of IT experience in software development. He holds a degree in Computer Science Engineering from the University of La Laguna, Spain.



Sima Modir is a Senior IT Architect with extensive experience in sales and delivery. She is a client technical lead in S&D, who is responsible for the architecture and delivery of complex cross-brand IBM solutions. Sima has worked in various roles, including project management, architecture, development, and testing. Sima has over 20 years of client-facing experience. Her industry experience includes telecommunications, media, retail, energy, and utilities. Sima focuses on implementing DevOps Continuous Delivery and tooling solutions. You can follow her on twitter @smodir.

This project was led by Margaret Ticknor, an IBM Redbooks Project Leader in the Raleigh Center. She primarily leads projects about WebSphere products and IBM PureApplication® System. Before Margaret joined the ITSO, she worked as an IT specialist in Endicott, NY. Margaret attended the Computer Science program at State University of New York at Binghamton.

Thanks to the following people for their support of this project:

Deana Coble, IBM Redbooks Technical Writer

Karen Lawrence, IBM Redbooks Technical Writer

Ann Lund, IBM Redbooks Residency Administrator

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

Jeremy Hughes, IBM Hursley Lab, OSGi Apps & DevOps Architect, WebSphere Application Server Development

Marianne L. Hollier, IBM US DevOps Solution Specialist, Certified IT Specialist

Matt Tarnawsky, IBM United Kingdom Limited, Rational Integration Testing Education Lead

William Scott, IBM US, Software Engineer

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>



Introduction

This book describes the benefits of Java development by using IBM WebSphere Application Server Liberty (WAS Liberty) in a DevOps manner. A pragmatic approach is used to take you through the entire delivery pipeline. This approach describes the advantages of WAS Liberty in each phase of the development lifecycle. WAS Liberty is often referred to as *Liberty* in this publication.

In this chapter, the following topics are described:

- ▶ DevOps
- ▶ WAS Liberty
- ▶ DevOps application lifecycle phases
- ▶ Objective
- ▶ Overview of the application use case

1.1 DevOps

DevOps is an organizational capability that focuses on the enhancement of communication and collaboration throughout the delivery pipeline for development and operational teams.

The following list shows the benefits of adopting DevOps practices for an organization:

- ▶ Improve time to market (with tighter traceability from requirements to production)
- ▶ Improve the quality of the outcomes
- ▶ Reduce time to both gather and analyze customer feedback

1.1.1 DevOps architecture

DevOps is a set of practices, tools, and services that are based on agile and lean principles. DevOps enables an organization to release new capabilities to their customers more quickly and with higher quality.

DevOps capabilities span the software delivery lifecycle. An organization determines where and how to implement DevOps based on business objectives, goals, challenges, and gaps in the organization's software delivery capabilities.

The following adoption paths are suggested:

- ▶ Continuous delivery: Automates software delivery to non-production and production environments across the delivery pipeline consistently and frequently. This DevOps practice focuses on reducing the amount of manual labor, resource wait time, and errors.
- ▶ Continuous integration: Tests software earlier in the lifecycle and continuously tests it across the lifecycle. This DevOps practice supports a *Shift-Left* testing approach. Shift-Left testing involves integration of the code by using techniques, such as service virtualization and test automation during development. Defects that are discovered during development are easier to fix and incur less expense than defects that are discovered during production.
- ▶ Continuous feedback: Monitors application usage and provides metric data to various stakeholders, such as operations, development, testing, and line of business owners.

Figure 1-1 shows these IBM DevOps adoption paths across the software delivery lifecycle.

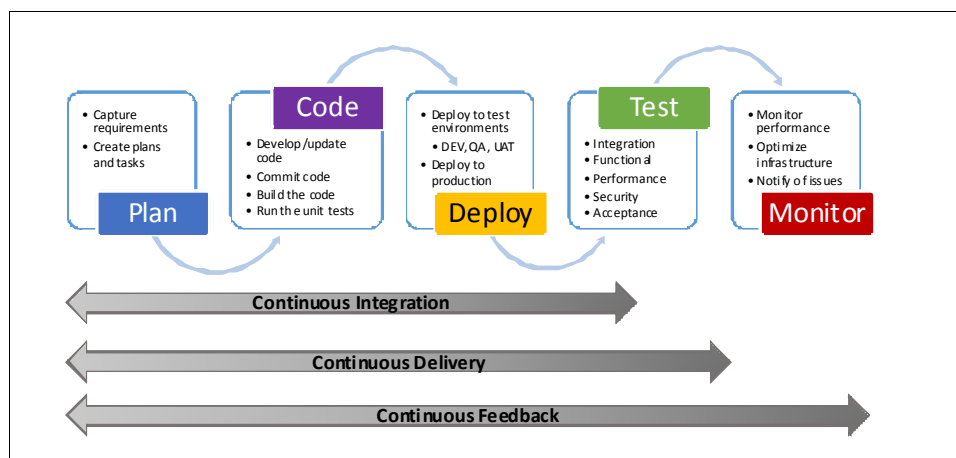


Figure 1-1 DevOps adoption paths

For more information about DevOps, see *DevOps for Dummies*:

http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=BK&infotype=PM&appname=SWGE_RA_VF_USEN&htmlfid=RAM14026USEN&attachment=RAM14026USEN.PDF#loaded

1.1.2 Business value

Businesses are under constant pressure to bring new innovative products and services to their customers much faster. They are also finding that traditional software delivery practices are insufficient for the following reasons:

- ▶ Time-consuming manual software deployments
- ▶ Lack of traceability from requirements to test
- ▶ Inabilities to quickly react to application issue or customer feedback

DevOps practices can enable an organization to transform their software delivery process to address these issues to bring *Continuous Improvement* to their clients.

1.2 WAS Liberty

Liberty is a highly composable, fast to start, and dynamic application server runtime environment. Liberty has the following characteristics:

- ▶ Easy configuration: Configuration is described in an XML file, which can be edited with a text editor and shared across the development team.
- ▶ Dynamic and composable run time: It loads only what the applications that are deployed on the server need.
- ▶ Quick server start time: The server starts in under 5 seconds with a basic web application.
- ▶ Low memory footprint: The server uses less than a 60 MB footprint for Java Platform, Enterprise Edition web application development.
- ▶ Extensible: The Liberty profile provides support for user and product extensions, which can use System Programming Interfaces (SPIs) to extend the run time.
- ▶ Open service support: The Liberty profile is built by using open standards (Open Service Gateway initiative (OSGi) framework).

1.2.1 Advantages for development

Liberty is designed with the developer in mind. The following list shows many benefits of using Liberty for Java development:

- ▶ Simple XML server profile that can be checked into the source code repository and managed as a development artifact.
- ▶ Composable, modular, server run time, which loads the application server with only the components that are needed by the application. Unused functions are not deployed.
- ▶ The Liberty profile and the WebSphere Application Server Developer Tools for Eclipse are available for a developer to download into the Eclipse platform.
- ▶ WAS Liberty also supports seven programming frameworks and provides easy integration with Docker, Node.js, Java Platform, Enterprise Edition (Java EE), and Linux. This support allows your development team to code wherever, whenever, and however they want.

1.2.2 Advantages for operations

Liberty provides a lightweight development environment but it is also ideal as a production environment for applications that do not require the full Java EE stack. Several Liberty benefits for operations are shown:

- ▶ The run time loads only the components that are needed by the application, providing a runtime production environment that is sized correctly.
- ▶ Liberty is easy to use, and it supports any cloud deployment model:
 - Infrastructure as a Service (IaaS)
 - SoftLayer®, Platform as a Service (PaaS)
 - Bluemix
 - OpenShift
 - Heruko
 - Container as a Service (CaaS), such as Docker
- ▶ Low memory usage offers support for more application instances for each production server.
- ▶ The run time can be upgraded with no migration of the user configuration or application.
- ▶ Liberty is configurable for a server cluster for application high availability (HA) and scalability.
- ▶ *Request timing* (a new feature) automatically identifies slow and stopped requests, and it provides detailed information to help identify the root cause.

1.3 DevOps application lifecycle phases

Application Lifecycle Management (ALM) phases are also referred to as the *Delivery Pipeline*. ALM phases are a set of stages that applications move through. The stages are listed:

- ▶ Requirement management to code design
- ▶ Development
- ▶ Testing
- ▶ Release management and deployment
- ▶ Production

The delivery pipeline is managed by different development methodologies, such as *Waterfall* or *Agile*. The key focus of DevOps is to understand the flow of the application and minimize any bottlenecks in its delivery pipeline to shorten the time from planning to production.

1.3.1 Planning

Typically, a new application or enhancement to an existing application originates from a business concept to offer a new option to an existing service or product. The focus of the planning phase is to capture business requirements, analysis, and scoping. Requirements management and traceability to development and testing is a key step in the ALM cycle that often is missed.

After the application scope is agreed to by various stakeholders, application milestones are included in the overall release planning. *Release planning* refers to the overall planning of a set of activities to bring new business functionality from requirements to production. Release planning can also involve updates to a set of new and existing applications. The application development methodology is selected during planning. Liberty fully supports the agile development methodology and it can integrate with planning tools easily, such as IBM Rational Team Concert™ (used in this book's scenario).

1.3.2 Development

During the development phase, developers start coding based on the defined solution design and architecture for the implementation of the new functional use cases. In addition to modeling and design, code development involves source control management (SCM) activities:

- ▶ Code repository
- ▶ Version control
- ▶ Defect tracking

Liberty supports a broad set of programming models and SCM tools, such as Rational Team Concert and GitHub.

1.3.3 Build

The process of compiling and creating an executable program from various code components is managed during the build phase. The build utility needs to compile and link the various files in the correct order. The build utility can link to SCM tools.

Liberty supports several build tools, including open source tools, such as Jenkins (used in our application use case).

1.3.4 Testing

Testing is an integral and important phase of the ALM lifecycle. This phase includes the following areas:

- ▶ Test planning
- ▶ Test case and script creation
- ▶ Test execution
- ▶ Test automation
- ▶ Test virtualization

Several testing stages can occur as part of the ALM lifecycle, such as system, integration, user acceptance, and performance testing.

Liberty offers a dynamic run time and improved performance. These enhancements accelerate the test environment definition and creation, which helps reduce cycle times during testing.

1.3.5 Deployment

Deployment involves the creation of a deployment package. The deployment package includes the executable code, middleware configuration, database schemas, and pushing the deployment package through different test and production environments in the delivery pipeline.

In addition to the dynamic run time and simple XML server configuration, Liberty offers its most significant advantage during the deployment phase.

1.4 Objective

This book covers a Java application use case, which is called LibertyCloudTrader, that is developed in Liberty. This process provides the user with a valuable understanding of the phases of the DevOps model. The application use case is described in 1.5, “Overview of the application use case” on page 7. In this book, a set of DevOps tools (both IBM and other vendors) is used to show the progression of the LibertyCloudTrader Java application through the delivery pipeline. The scenario highlights the use Liberty for advantages in the following areas:

- ▶ To help speed up delivery without sacrificing quality
- ▶ To show the traceability and key capabilities of Liberty for developing Java applications
- ▶ To show Liberty in production

Figure 1-2 shows the tools that are used across the delivery lifecycle in this scenario to show the advantages of using Liberty.

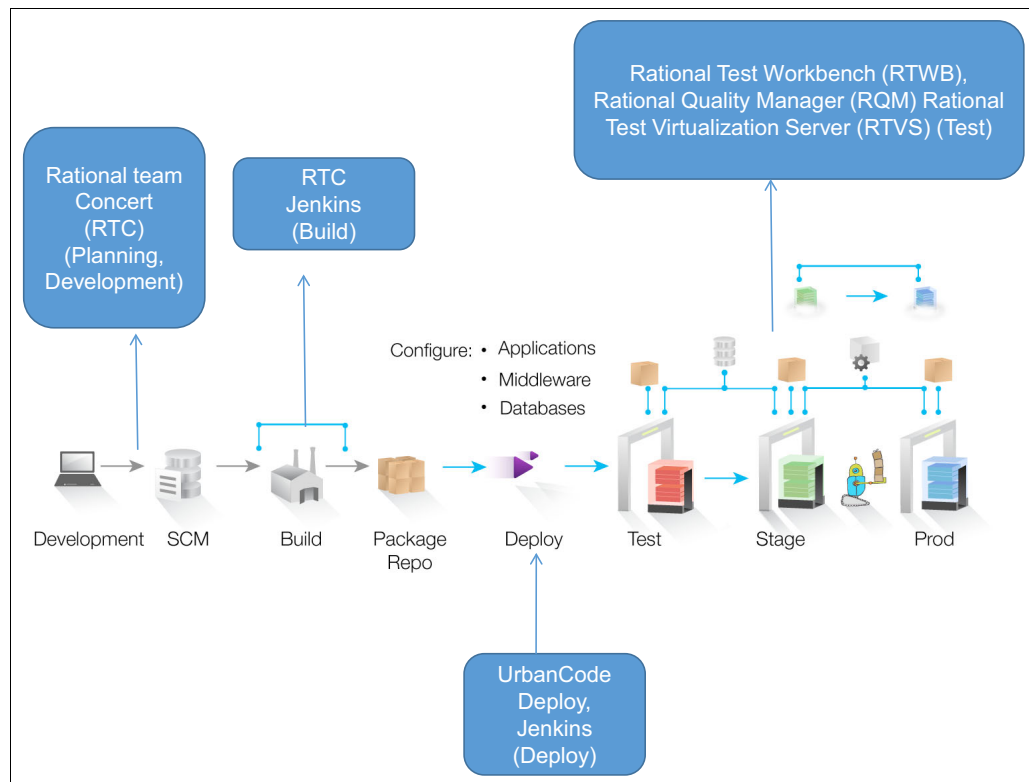


Figure 1-2 DevOps LifeCycle tools that are used in the scenario

1.4.1 DevOps products

The following list shows the DevOps products (both IBM other vendors) that are highlighted throughout this book, in addition to the Liberty profile server:

- ▶ IBM Rational Team Concert
Rational Team Concert provides a collaborative platform for planning, coding, and build and change management.
- ▶ IBM UrbanCode™ Deploy
IBM UrbanCode Deploy is a tool for automating application deployments through your environments.
- ▶ IBM Rational Quality Manager
Rational Quality Manager is a collaborative test creation and management tool.
- ▶ IBM Rational Test Workbench
Rational Test Workbench provides a test automation solution for applications, regression testing, integration technologies, and performance and scalability testing.
- ▶ IBM Rational Test Virtualization Server
Rational Test Virtualization Server enables the deployment of virtualized services for integration testing earlier during the development phase.
- ▶ Jenkins
Jenkins is an open source continuous integration tool that provides code build and deployment functionality.
- ▶ IBM Bluemix
IBM Bluemix is a cloud platform as a service (PaaS) that is developed by IBM. It supports several programming languages, services, and integrated DevOps to build, run, deploy, and manage applications on the cloud. Bluemix is based on Cloud Foundry open technology. Bluemix runs on the SoftLayer infrastructure.

1.5 Overview of the application use case

The application use case, which is called LibertyCloudTrader, is shown in Figure 1-3 on page 8, and it is referred to throughout this book. LibertyCloudTrader simulates an online stock trading system. The scenario application allows users to take the following actions:

- ▶ Log in
- ▶ View their portfolio
- ▶ Look up stock quotes
- ▶ Buy or sell shares

The LibertyCloudTrader application is based on the CloudTrader application, which was developed by using Liberty profile server. The application is built primarily with Java Servlets, JavaServer Pages (JSPs), and JavaBeans. It creates a SOAP web service call to an external DayTrade application to retrieve stock quotes. It simulates trade transactions by populating a local IBM DB2® database. LibertyCloudTrader has a Java Database Connectivity (JDBC) connection to the DB2 database.

The application can be accessed either through an HTTP web interface or a Representational State Transfer (REST) application programming interface (API), which is added to show a mobile interface.

Figure 1-3 shows the LibertyCloudTrader system context diagram.

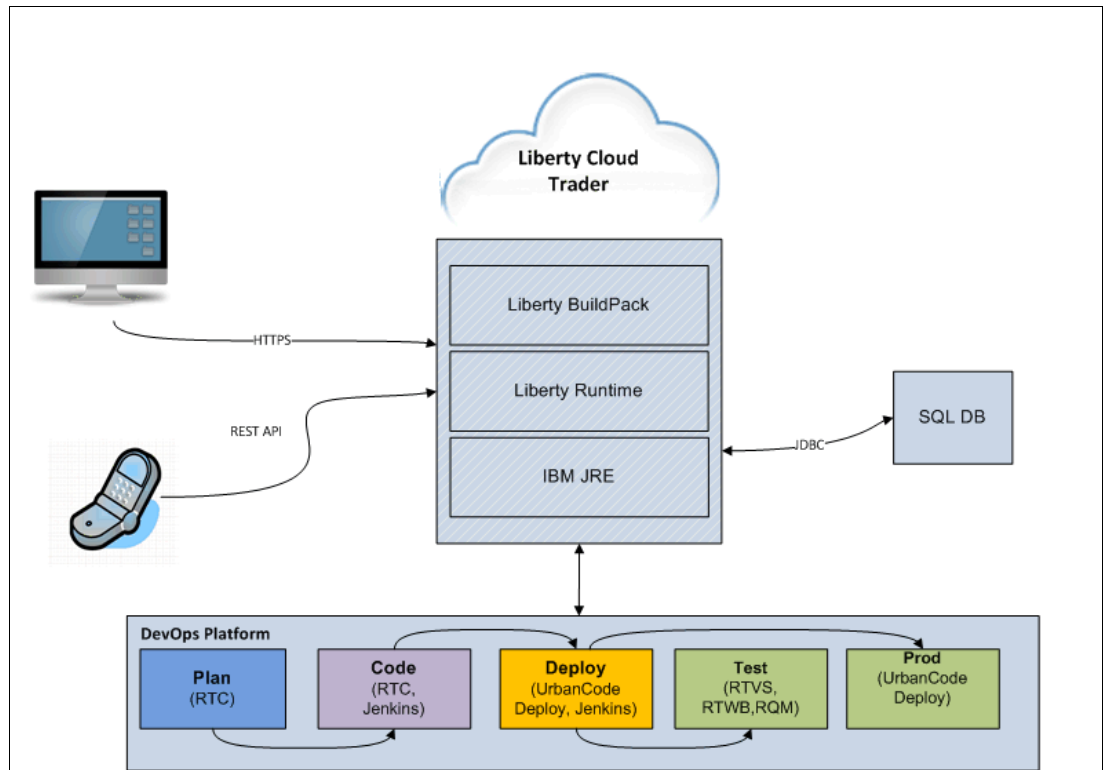


Figure 1-3 LibertyCloudTrader system context diagram



Project planning by using Rational Team Concert and Rational Quality Manager

This chapter describes how to use Rational Team Concert and Rational Quality Manager for project planning.

In this chapter, the following topics are described:

- ▶ Software development
- ▶ Infrastructure
- ▶ Planning a software development project with Rational Team Concert
- ▶ Integrating the Eclipse Java Platform, Enterprise Edition client with Rational Team Concert and WAS Liberty
- ▶ Rational Quality Manager for test planning

2.1 Software development

Software development is a complex process that is structured in multiple stages. Each stage requires attention and close collaboration among the people who are involved in development and those individuals who are involved in project planning.

Each software development project must go through the following stages at a minimum:

- ▶ Requirements management
- ▶ Functional specifications
- ▶ Architecture and design documents
- ▶ Development
- ▶ Quality assurance
- ▶ Release management
- ▶ Documentation
- ▶ Delivery

2.1.1 Requirements management

Requirements management is usually the first phase of software development. This phase starts when a business need is identified and transformed through architecture into a technical solution. This common ground is where architecture and project management come together. They come together to create the artifacts and deliverables that define a method to structure and document the alternatives that achieve the business goals and to fulfill the stakeholders' vision and strategy. However, stakeholders, who will use or benefit from the final product, are not always involved in the project.

Usually, the requirements are handled by a product owner or program owner and any stakeholders. In a software development project, the product owner acts as a *representative of the customer*.

2.1.2 Functional specifications

Functional specifications describe the behavior or functionality of the software product on an abstract level. In larger products, functional specifications might consist of separate documents for each feature of the product. For example, in a web-based software, a functional specification describes the user interface (UI) design of the front end, such as HTML, Cascading Style Sheets (CSS), and JavaScript. Another specification describes the back-end implementation, such as security standards.

Based on approved functional specifications, the developers create the logical model of the software for which they write the code, or they create manuals for users.

One way to write functional specifications is by using an agile method:

As a: <Role> I want to: <functional statement> So that: <business goal is reached>.

2.1.3 Architecture and design documents

When all of the requirements are collected and prioritized, the technical architecture team, which consists of highly qualified technical specialists, starts to work on the product architecture. The architecture defines the components of the product, how they interact with each other, and how they work in concert as a stand-alone product.

Consider the following information when you prepare the business sections of the architecture and design documents:

- ▶ A business footprint (a high-level description of the people and locations that are involved with key business functions)
- ▶ A detailed description of business functions and their information needs
- ▶ A management footprint, which shows the span of control and accountability
- ▶ Standards, rules, and guidelines, which show working practices, legislation, and financial measures
- ▶ A skills matrix and set of job descriptions

Reports and graphics that are generated by modeling tools can be used to demonstrate the key views of the architecture. The architectural artifacts are then reviewed by relevant stakeholders, and their feedback is incorporated.

2.1.4 Development

Software developers use the design documents and development tools to write code. The design documents and development tools are specific, integrated development environments (IDEs), development environments, and static code analysis tools. The code-writing phase is usually the longest phase in the product lifecycle.

Each developer writes code and then collaborates with other developers to ensure that the components can be successfully integrated.

A source control management (SCM) tool is the software component that provides collaboration and coordination between team members of a development team. SCM provides version control and file management, which together ensure that team members do not write over each others' changes, and that only the newest versions of files are identified for use in the repository. After the code is checked in, the build process starts. After a successful build is released, the testing process starts.

2.1.5 Quality assurance

Testing is one of the most important phases in software development, and it affects the end-product functionality and support significantly. If the quality assurance (QA) process is missing, the end-product might have serious problems and might not meet customer and product owner expectations. This situation can be a serious situation that ends in time and financial losses.

This situation can be avoided by performing extensive testing, which is referred to as quality assurance in most software development.

Usually, testing starts as soon as the initial sections of the software are available. Several types of testing exist, and each type is important.

Unit testing

Unit testing is the testing of one part, one feature, or one component of the product, for example, the test of a *submit* button from a UI. An initial test is done by the developer when coding is complete for a specific feature of the product. This testing ensures that the feature performs the expected function. The component is then sent to the testing team, which runs test cases to ensure that the unit works correctly.

Smoke testing

Smoke testing is a low-level check to ensure that all software components are compiled and integrated with each other without any issues. This simple process ensures that the development team provides a working piece of software.

Regression testing

Regression testing is a process that is used for large and long-term software development projects. Regression testing is used to determine and analyze the behavior of a product. Regression testing also ensures that one or more features do not change or break because of other new or changed code.

Functional testing

Functional testing is performed to ensure that the software performs exactly as expected. As with unit testing, this type of testing is required before a final version of the product is released.

This type of testing is performed by dedicated testers. It excludes the developer who wrote the code because the developer tests the software for the purpose for which it was written, but the developer might overlook another malfunction that is inherent in the code.

Another requirement is a test plan that contains test cases. This test plan is necessary to test the functionality of the product. The test plan is divided into test cases that follow an execution plan. Each execution plan has an initial result and an expected result. If those two results are the same, the functional test passed, and the code is confirmed as a working feature.

2.1.6 Release management

When the testing process is finished, the product can officially be released. Therefore, a complete, stable version of the product can be available for use. The development team stops work on this project until new requests are received as fixes or new features. A preferred practice is to fix the problems in the same repository (with the stable version of the product). The new features are handled in a new branch.

Branches

A branch is basically a duplication of a code repository under source control management (SCM). After a stable version of the software is released, future development starts by creating a branch to that specific state of the application. The following list shows examples of this concept:

- ▶ Release 1.0.1 of the application *ABC* can be accessed through a web UI
- ▶ Release 1.0.2 the application *ABC* can be accessed through a web UI and a representational state transfer (REST) application programming interface (API).

In these examples, release 1.0.2 is a branch of release 1.0.1, with the addition of the REST API feature.

Release notes

Usually, the release notes represent a file that contains the metadata of that specific release, which might be included and delivered with the product. A release note contains information about the following characteristics of the product:

- ▶ New functionalities
- ▶ Bug fixes
- ▶ Future enhancements

2.1.7 Documentation

In software development, documentation can be divided in sections: deliverables and artifacts. An *artifact* is used in the development process, and a *deliverable* is part of the product that is being developed. For example, the solution design document for application ABC is an artifact, and the user guide of application ABC is a deliverable.

2.1.8 Delivery

Delivery starts after a product is released and acquired or implemented in response to a business requirement. Delivery is the phase that covers the installation, configuration, and maintenance of the product.

These activities need to be handled by a separate team that can receive support from the development team, if needed.

2.2 Infrastructure

The following section describes the infrastructure requirements for setting up a DevOps environment. Each organization differs with its own types of hardware and software. Therefore, a high-level overview is described without details about the naming conventions, port numbers, firewall rules, and so on.

2.2.1 Scenario

For the scenario that is used in this publication, a single-node deployment server is configured to work with a build server to deploy the application in different environments. The build server (Jenkins) accesses to the SCM that is provided by Rational Team Concert and triggers periodic builds.

When a new version is successfully built, testing can take place (unit testing by using Rational Quality Manager). A successful build is promoted on the deployment server, which deploys the application in different environments.

For this particular example, two environments are used: test and production.

Business requirements

The solution needs to support the following business requirements:

- ▶ Synergy among multiple software components
- ▶ Technology advances and configurations to improve application development
- ▶ An infrastructure in which critical data will not be lost

Technical requirements

The solution needs to meet the following technical requirements:

- ▶ An environment where a DevOps solution can be implemented
- ▶ Support for different technologies and software stacks
- ▶ All of the technical capabilities that are required to support the DevOps features:
 - Automated build and testing
 - Integrated configuration management
 - Integrated change management
 - Continuous integration
 - Integrated deployment planning
 - Continuous deployment
 - Production support
 - Application monitoring
- ▶ The ability to define, implement, and manage the IT resources with systems management standards and processes

Availability

Availability management detects the loss of any critical resources. It can also identify whether a critical resource either reached a minimal operational threshold or exceeded a maximum operational threshold.

Reliability

Reliability is the ability of the system to provide application functionality in a predictable and reliable manner so that the same results are generated consistently under different operating and system conditions.

Scalability

Scalability is the ability to expand the system architecture to handle more users, more transactions, and more data as users and data are added.

Scalability allows existing systems to be extended as far as possible without necessarily needing to replace them.

Scalability requirements address the following domains:

- ▶ Hardware
- ▶ Software
- ▶ Network
- ▶ Applications

To define a scalable infrastructure, we define the anticipated load over a period of time (perhaps two to four years) with projected additional functionalities that might be added to the application over a longer period of time.

Capacity planning

A DevOps architecture must support not only the current anticipated load, but also the projected load on the system over the next few years, according to the strategy of the organization that implements it.

The systems are expected to provide an acceptable level of performance under peak load and to allocate additional resources, if necessary, to satisfy the business requirements.

The DevOps infrastructure must also define the expected capacity, which is based on the number of concurrent or peak service instances, such as code check-in, builds, and deployment processes.

Resource pool requirements (CPU, memory, storage, and network) for a DevOps infrastructure vary from one organization to another organization.

The following configuration was used for the DevOps infrastructure of this book:

- ▶ Deployment server: Eight CPUs, 32 GB RAM, and 500 GB hard disk drive (HDD)
- ▶ SCM server: Four CPUs, 16 GB RAM, and 500 GB HDD
- ▶ Test environment: Four CPUs, 16 GB RAM, and 500 GB HDD
- ▶ Production environment: Four CPUs, 16 GB RAM, and 500 GB HDD

The following configuration was used for the software stack:

- ▶ Deployment server:
 - UrbanCode Deploy (UCD) server 6.1.1
 - Rational License Key Server (RLKS) 8.1.4
 - IBM DB2 for Linux, UNIX, and Windows Version 10.5
- ▶ Test server
Jenkins 1.609.2
- ▶ SCM server
Rational Team Concert 5.02
- ▶ Test environment:
 - WAS Liberty 8.5.5
 - IBM DB2 for Linux, UNIX, and Windows Version 10.5
- ▶ Production environment:
 - WAS Liberty 8.5.5
 - IBM DB2 for Linux, UNIX, and Windows Version 10.5

Integration and network configurations

On the deployment server where UCD is installed, the firewall must allow communication on the local host between the UCD server and the DB2 server on port 50000 and between the UCD server and RLKS on port 27000.

Communications between the build server and the SCM server must be allowed. In addition, communications between the build server and deployment server are required to enable integration between the Jenkins and UCD servers and between Jenkins and Rational Team Concert.

The deployment server communicates with deployment environments by using UrbanCode Deploy agents, so the communication is required on port 7918 for the server and on port 7916 for the agent relay.

Caching artifacts as described in component versions, which is a feature of UCD, is handled by the UCD agent automatically. Caching artifacts occurs on agent relays. Agent relays must be able to initiate connections for an HTTP proxy; the default port is 20080.

Also, relays must be able to receive connections from agents; the default port is 20081.

In other organization environments, the ports might be changed. So, the integration must be performed based on the correct ports. This example uses the default ports.

System requirements

Use the following links to obtain the system requirements:

- ▶ UrbanCode Deploy server 6.1.1:
https://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.install.doc/topics/sysRequire.html?cp=SS4GSP_6.1.1%2F3-0
- ▶ Rational License Key Server 8.1.4:
<http://www.ibm.com/support/docview.wss?uid=swg27038212>
- ▶ Rational Team server 5.02:
<http://www.ibm.com/software/products/en/rtc#othertab1>
- ▶ IBM DB2 for Linux, UNIX, and Windows Version 10.5:
<http://www.ibm.com/support/docview.wss?uid=swg27038033>
- ▶ WAS Liberty profile 8.5.5:
<http://www.ibm.com/support/docview.wss?uid=swg27038218>

2.2.2 Monitoring and event management

Monitoring and event management provides analysis and the correlation of hardware, software, and network events. The infrastructure architecture needs to support proactive monitoring, because monitoring is one of the major requirements in a DevOps solution.

IBM Performance Management provides sophisticated and end-to-end resource performance, availability, and capacity management capabilities to help you optimize your virtualized cloud and hybrid environments. Performance Management reduces your client's management processes to a single tool for virtualized environments. It includes analytics that facilitate capacity planning and proactive right-sizing of cloud environments.

2.2.3 Provisioning

IBM Cloud Orchestrator gives you access to ready-to-use patterns and content packs, which help you to speed up configuration, provisioning, and deployment. It integrates management tools, such as metering, usage, accounting, monitoring, and capacity management, into your cloud services. Go live as quickly as you develop and test applications.

IBM Cloud Orchestrator helps provide these functions:

- ▶ Quickly deploy and scale on-premises and off-premise cloud services
- ▶ Provision and scale cloud resources
- ▶ Reduce administrator workloads and error-prone manual IT administrator tasks
- ▶ Integrate with existing environments by using APIs and tooling extensions

2.3 Planning a software development project with Rational Team Concert

Rational Team Concert is an agile application lifecycle management (ALM) solution that helps companies to build better software and products with an all-in-one agile environment for development teams.

Rational Team Concert can adapt to any kind of project, regardless of size or complexity. In addition, Rational Team Concert provides collaborative change management capabilities. These capabilities are available separately. They can be integrated with popular source control systems.

Rational Team Concert helps teams collaborate for faster software delivery. It enhances team collaboration with its integrated features, including work item, build, and software configuration management. It also provides these functions:

- ▶ Provides high visibility into project activities and team progress with multilevel dashboards and reporting features
- ▶ Facilitates the planning and execution of agile or formal projects with planning tools and templates

2.3.1 The role of Rational Team Concert in DevOps

Together with UCD and Liberty, Rational Team Concert can be used as a development component in a DevOps solution. Rational Team Concert can provide all-in-one development capabilities and integration support with other tools (UCD, Jenkins, and Git), both commercial and open source. The following steps are required for a hypothetical DevOps scenario:

1. Create a project plan in Rational Team Concert.
2. Define the work items according to your project specifications (user stories and tasks).
3. After development is completed, check in the code.
4. Trigger a scheduled build and run unit testing during the build.
5. Validate application functionality and features with regression and functional testing.
6. Deploy the application by using the UCD server.
7. Monitor the application with IBM Performance Monitoring.

One way to plan a software development project is by using an agile project management methodology, such as Scrum¹, Kanban², or XP³. Rational Team Concert can offer support for both traditional project management methodology and agile, as well.

By default, Scrum is the methodology that is used in Rational Team Concert. Scrum is a flexible, holistic, iterative, and collaborative methodology. Scrum is one of the most popular methodologies. It is implemented worldwide in academic, open source, and enterprise projects.

2.3.2 Scrum project lifecycle and iterations mapping: Development

All phases of software development process are handled in Scrum as *sprints*. Whether is planning, development or testing, every phase of the software development lifecycle can be covered in one or many sprints, according to the project size and deadlines. When a sprint is finished, the remaining work items are deferred to the next sprint until that phase ends.

¹ Scrum.org, <https://www.scrum.org/>

² KanbanFlow - Lean project management, simplified, <https://kanbanflow.com/>

³ Extreme Programming: A gentle introduction, <http://www.extremeprogramming.org/>

Iterations

Iterations or *sprints* are predefined periods of time during which different activities occur. A sprint is generally between one week and one month. During a sprint, several meetings take place:

- Day 1: Sprint planning

During the sprint planning meeting, the team members can choose their work for the sprint that is about to begin. The goal of sprint planning is to identify what to develop this iteration and how.

- Day 6: Backlog refinement

During the backlog refinement meeting, the team grooms the product backlog by reprioritizing and reestimating.

Examples of backlog refinement might include the following statements. Beginning the sentence with “Rather is planning,” doesn’t make sense to me. Also, when you say “...every phase”, are you referring to the phases of the software development lifecycle?

- Day 10: Sprint review

During the sprint review meeting, a demo of the deliverable is presented to the product owner and stakeholders. During this meeting, the stories that are completed can be closed after they are validated against the definition of the Done status.

- Day 10: Sprint retrospective

During the sprint retrospective meeting, the Scrum team inspects, adapts to, and responds to a few questions:

- What areas of the project went wrong?
- What do you advise the team to start or stop doing?
- What can be celebrated?

- Daily Scrum

This daily Scrum is a time-boxed stand-up call (a predefined period of time; usually 15 minutes) that occurs each day of a sprint. During the daily Scrum, the following questions are asked:

- What did you accomplish since the last meeting?
- What you are going to do today?
- What is impeding your progress?

The following list overviews the required Scrum roles:

- Product owner

The product owner represents, and is the voice of, the stakeholders. The product owner functions as the end-to-end owner of a specific epic. (An *epic* is a top-level work item that requires a longer time to complete, for example, more than one quarter of a year.) The product owner ensures that the following tasks are completed:

- The stakeholders’ requirements are satisfied for all stages.
- The stakeholders are included in providing direct feedback, where necessary.
- The implemented epic is expected to be profitable.
- Approval is received at end of the defined sprints and the solution design sprints.
- The stakeholders, technical management, and other disciplines on the project anticipate success.
- The product owner is responsible for handling the product backlog.

- Scrum master

A Scrum is facilitated by a Scrum master who is accountable for addressing items that might impede the delivery of the sprint goals or deliverables.

- Development team

The development team is a self-organizing team with cross-functional skills and the responsibility for delivering potentially shippable increments of the project at the end of each sprint.

- Scrum team

The Scrum team consists of the product owner, Scrum master, and development team.

When a running instance of Rational Team Concert exists, ensure that all of the Scrum team members have a user ID and a license that is assigned to the user ID. Then, create a lifecycle project:

1. Navigate to the Rational Team Concert server administration:

<https://yourserver:9443/jts/admin>

2. Click **Create Lifecycle Project**, as shown in Figure 2-1.

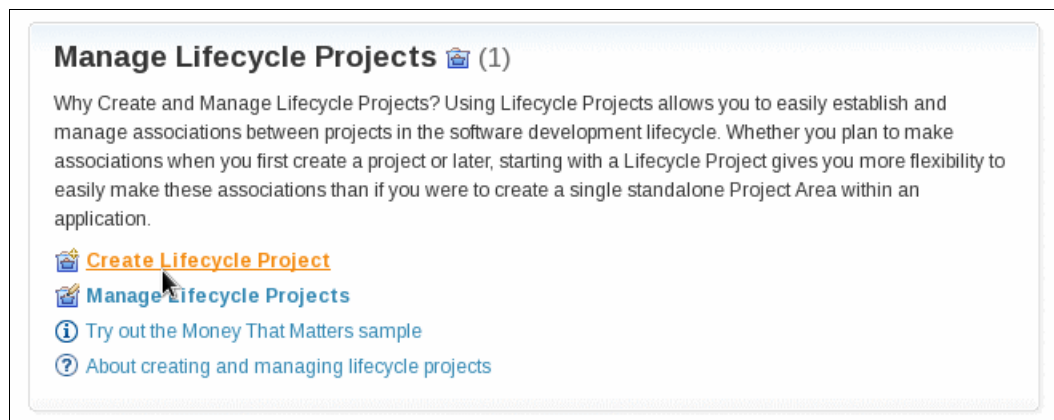


Figure 2-1 Create a lifecycle project in Rational Team Concert

When the task is finished, your lifecycle project is listed in the Lifecycle Projects administration page (you are redirected when the lifecycle project is created), as shown in Figure 2-2.

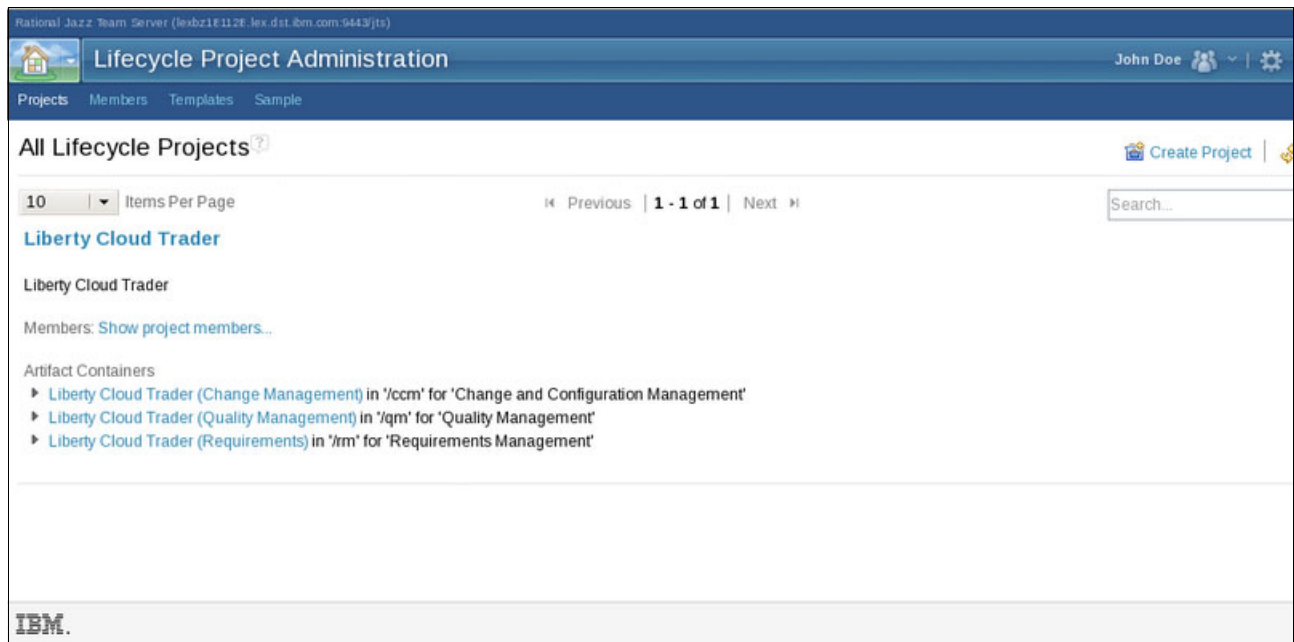


Figure 2-2 Lifecycle Project Administration page

Define the project timelines and iterations

After the project is created, you can start the timeline definition:

1. Click the **gear** icon and select **Manage Lifecycle Projects**, as shown in Figure 2-3.



Figure 2-3 Manage lifecycle projects

2. Select your project from the list. A new page opens, as shown in Figure 2-4.

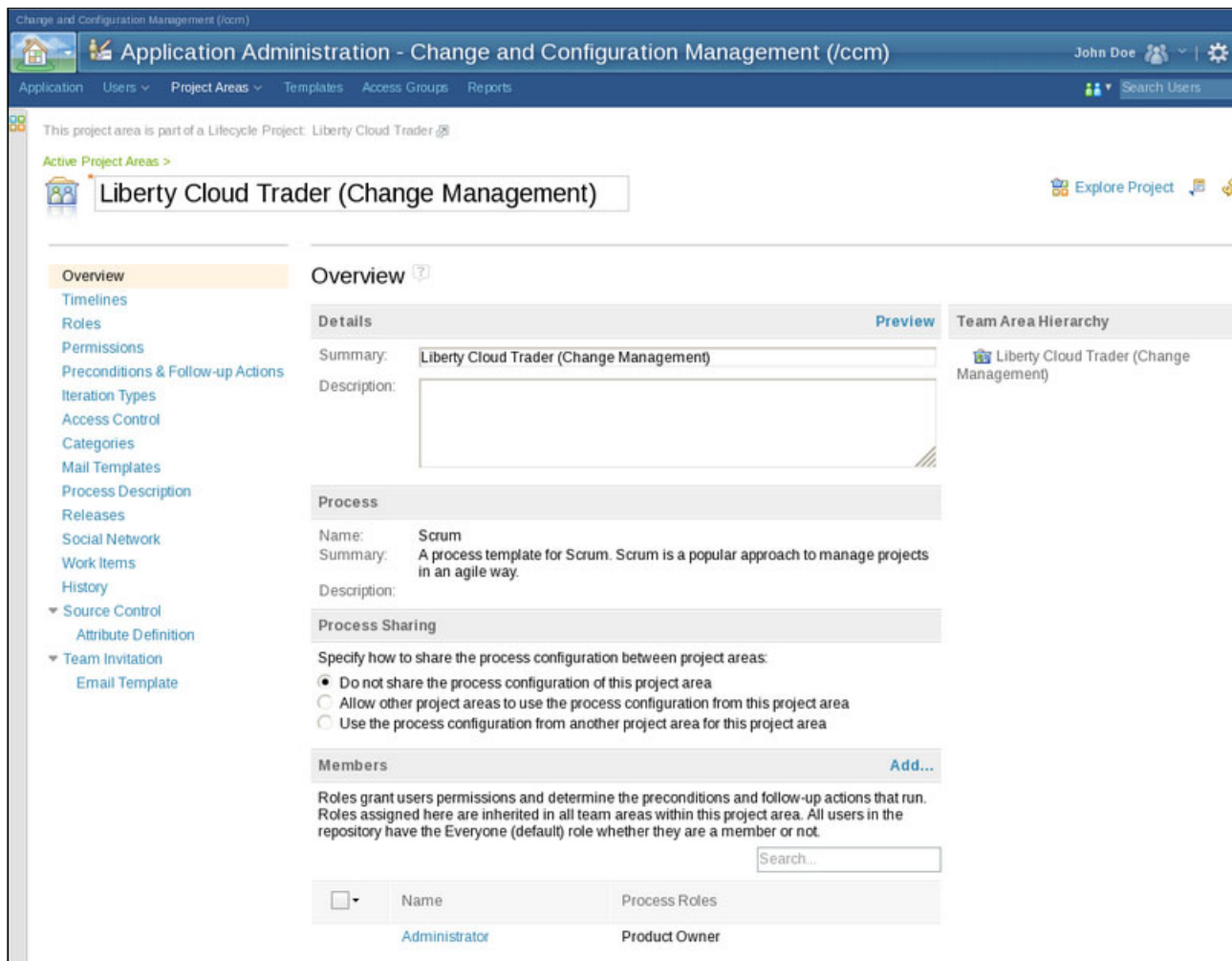


Figure 2-4 Lifecycle Project Administration page

3. From the menu on the left, click **Timelines**, as shown in Figure 2-5.

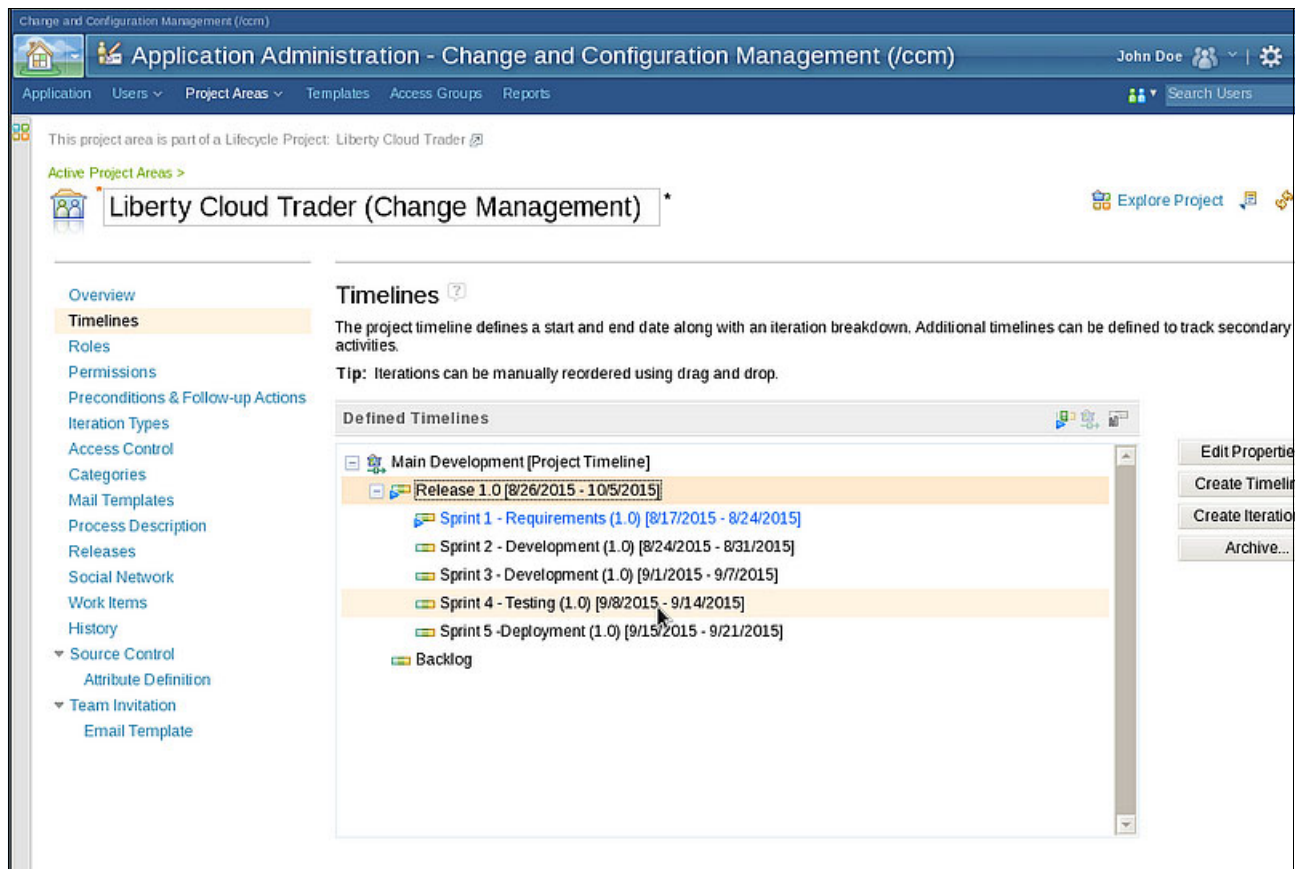


Figure 2-5 Timelines definition

You can edit the default values of a timeline or create new values.

After a lifecycle project is created and the iterations are defined, the product owner creates a top-level epic.

4. Click **Change Management**, and you are redirected to the Change Management area of Rational Team Concert.

5. From the Change Management menu, select **Create a work item** as shown in Figure 2-6.

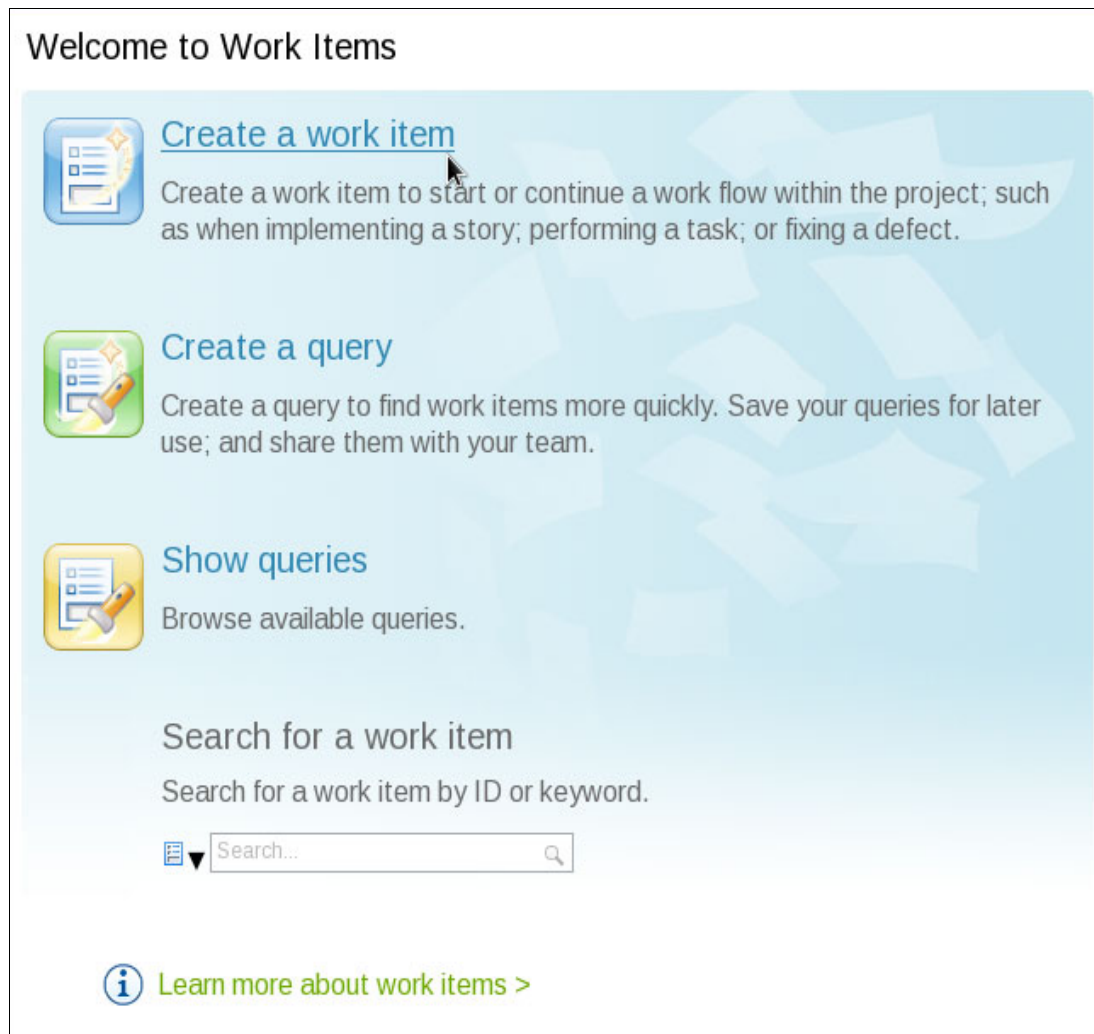


Figure 2-6 Welcome to Work Items page

6. A new form opens, as shown in Figure 2-7. In the Type field, select **Epic** from the drop-down list.

Assign the team members as subscribers to the epic that was created. The development team can start to define the user stories.

The screenshot displays the 'Liberty Cloud Trader (Change Management)' application interface. The top navigation bar includes links for 'Project Dashboards', 'Work Items', 'Plans', 'Source Control', 'Builds', and 'Reports'. The user 'John Doe' is logged in, and the environment is 'DEV00'. The 'Work Items' section is active, showing a form for creating a new 'Epic' work item. The form title is 'Epic <15:53:56> *'. The 'Summary' field is empty, and the status is 'Uninitialized'. The form is divided into several sections: 'Details' (Type: Epic, Filed Against: Unassigned, Progress: No Work, Project Area: Liberty Cloud Trader (Change Management), Team Area: Liberty Cloud Trader (Change Management)), 'Tags' (empty), 'Owned By' (Unassigned), 'Priority' (Unassigned), and 'Planned For' (Unassigned). A 'Quick Information' box on the right indicates 'No Links'. The 'Description' section is empty, and the 'Discussion' section has a text area for comments.

Figure 2-7 Create an Epic work item

2.3.3 User story

A user story (Figure 2-8) is a written description of a piece of functionality that is valuable to a stakeholder. This description uses language that is understandable by business and development people. The user story is also a communication mechanism between the stakeholders and the developers.

The recommended way to write a user story is to use the following format:

As a: <Role> I want to: <functional statement> So that: <business goal is reached>

The screenshot displays the 'Liberty Cloud Trader (Change Management)' application. The main window shows a 'Work Items' view with a specific 'Story' selected. The story's title is 'Story <18:21:42> *'. The summary is 'DEV004: Develop Application Business Logic'. The interface includes tabs for 'Overview', 'Acceptance', 'Links', 'Approvals', and 'History'. The 'Details' section contains fields for 'Type' (Story), 'Filed Against' (Unassigned), 'Story Points' (40 pts), 'Progress' (No Work), 'Project Area' (Liberty Cloud Trader (Change Management)), and 'Team Area' (Liberty Cloud Trader (Change Management)). It also has fields for 'Tags', 'Owned By' (John Smith), 'Priority' (High), and 'Planned For' (Unassigned). The 'Description' section contains the text: 'As a: Developer', 'I need: to develop the core application business logic', and 'So: I can integrate it with the application UI and achieve the project goal'. The 'Discussion' section has a text input field labeled 'Add a comment...'. The interface is clean and professional, with a blue header and a white main area.

Figure 2-8 Create a user story

The following user story attributes are examples:

- ▶ Help with release planning and product roadmap predictability.
- ▶ The basis for collaborative decision making, with a focus on communication and participatory design.
- ▶ A focus on stakeholder goals rather than implementation.

- Backlog (stories are ranked by value and risk, with high-risk, high-value stories ranked the highest).
- A description of the story that is used as an anchor for communication and planning. Comments relay the discussions, assumptions, and acceptance criteria that determine the completion of a story.

In the Acceptance tab, enter the Acceptance Test field, which is used to define the acceptance criteria and non-functional requirements, as shown in Figure 2-9. Use this format:

Given: <context> When: <event> Then: <outcome>

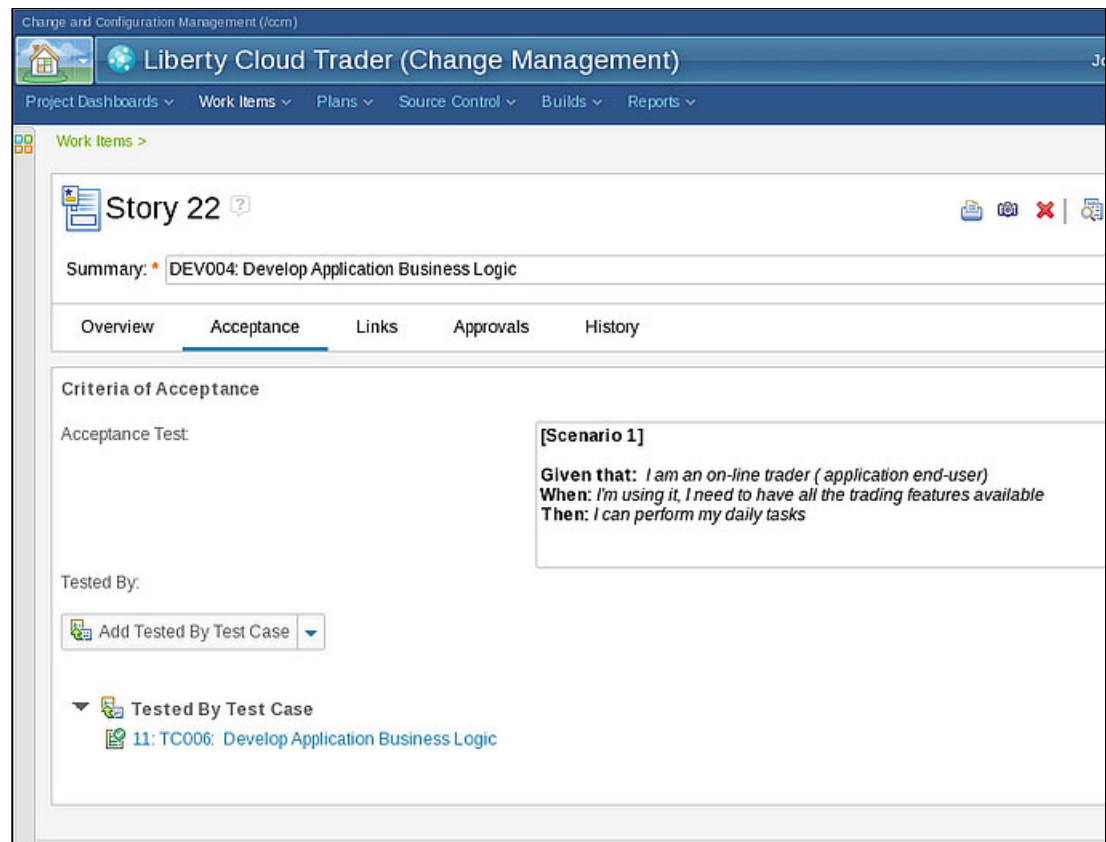


Figure 2-9 Story acceptance criteria

Acceptance criteria scenarios are used to create the test plan in Rational Quality Manager for that specific story. After the story is created, an epic needs to be set as a parent for that story.

You can set an epic on the Links tab by using the Add Related list box, as shown in Figure 2-10.

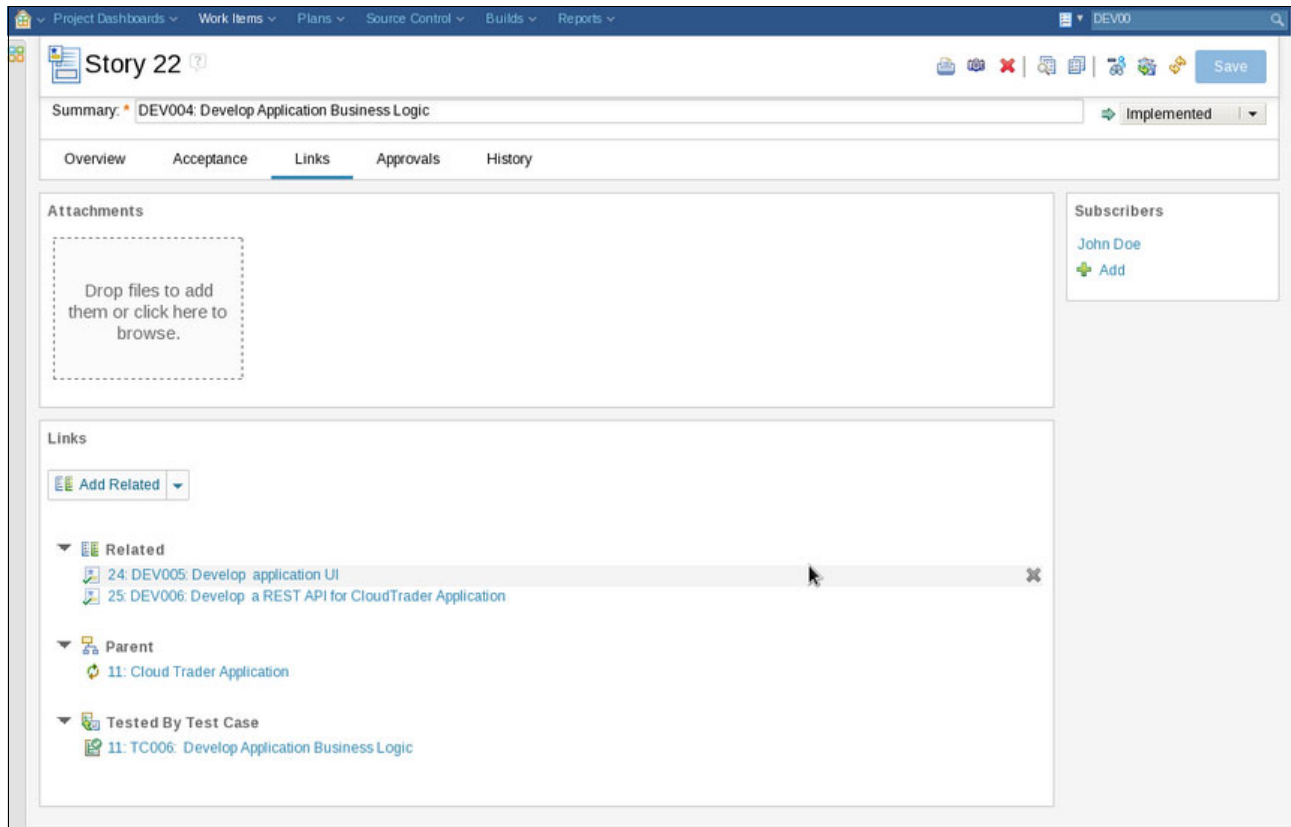


Figure 2-10 Story-related links

Story workflow

During a development iteration, a story status might change several times.

For example, a story is created by the Scrum master. The status, by default, in Rational Team Concert is New. After the story is assigned to a team member, the status is changed to Start Working. Rational Team Concert automatically changes the status again to In progress. When development is complete, the story goes to Complete Development, which changes the status to Implemented. After the testing is completed, Complete testing changes the status to Done.

When a story can be closed

To define completed work, Scrum uses Definition of Done (DoD). DoD is part of the Scrum Framework and represents the exit-criteria that determines whether a product backlog item (a story, usually) is complete. DoD applies to all user stories in each sprint, for example:

- ▶ **Code Complete:** The code is checked into SCM, and a work item and comment are assigned to that check-in.
- ▶ **Unit Test Written and Executed:** A test that proves the functionality of the code.
- ▶ **Integration Tested:** Ensures that the code that is delivered with the story is not breaking the rest of the application or breaking the build.
- ▶ **Documented:** The feature is documented and the comments exist in the code.

After a story is complete and is set as Done (validated through DoD), the code that was delivered with that story becomes part of the build. Scheduled builds can be triggered by Jenkins, and an automatic deployment can start in the test environment for further testing, such as functional testing and regression testing. When all of the testing passes the criteria, the build is stable. The build can be promoted to production. The same cycle can be reiterated for bug fixing and new features.

2.4 Integrating the Eclipse Java Platform, Enterprise Edition client with Rational Team Concert and WAS Liberty

Follow these steps to integrate the Eclipse Java Platform, Enterprise Edition client with Rational Team Concert and WAS Liberty:

1. Download and extract Eclipse Mars, Release 4.5.0:
<https://eclipse.org/downloads/>
2. Download Rational Team Concert Eclipse Extension, Version 5.0.2:
<https://jazz.net/downloads/rational-team-concert/releases/5.0.2/RTC-Eclipse-Client-Extension-repo-5.0.2.zip>
3. Install Rational Team Concert Eclipse Extension:
 - a. In the Eclipse client, select **Help** → **Install New Software On “Work with” field**. Then, select **Add**. A pop-up window opens. Choose a name, such as RTC, and click **Archive**. Point to a Rational Team Concert extension archive file, and click **OK**.
 - b. Select **Rational Team Concert Client**, and click **Next**, as shown in Figure 2-11.

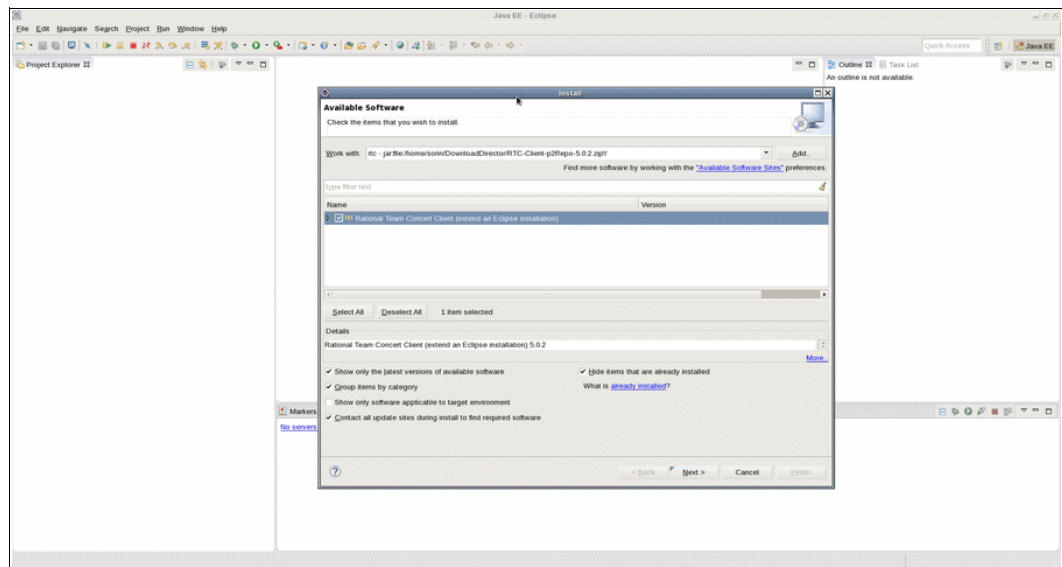


Figure 2-11 The Rational Team Concert Client repository

- c. Review the installation item and click **Next**, as shown in Figure 2-12.

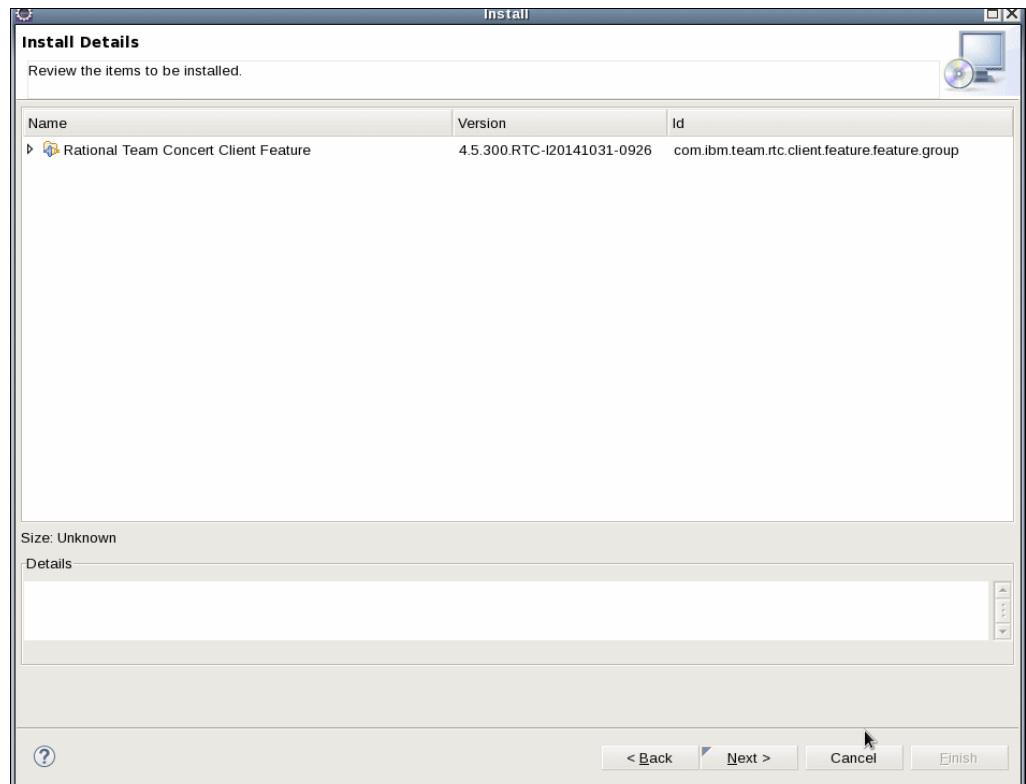


Figure 2-12 Feature to install

- d. When the installation completes, you are prompted to restart Eclipse. After the restart, the Eclipse Welcome page includes the Rational Team Concert features, as shown in Figure 2-13.

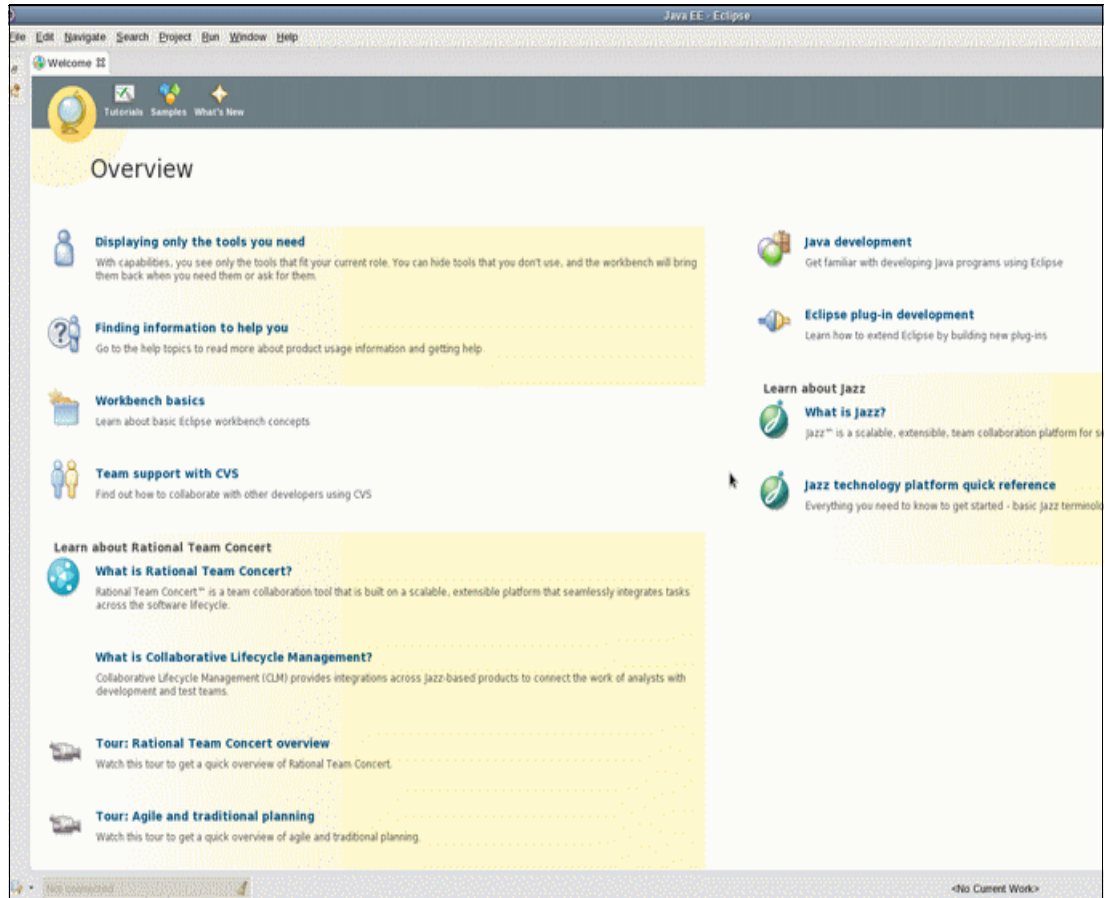


Figure 2-13 Eclipse welcome window

4. Create a repository connection:
 - a. With the installed Rational Team Concert features, you can create a repository connection. Switch the perspective to **Jazz Administration**, as shown in Figure 2-14.

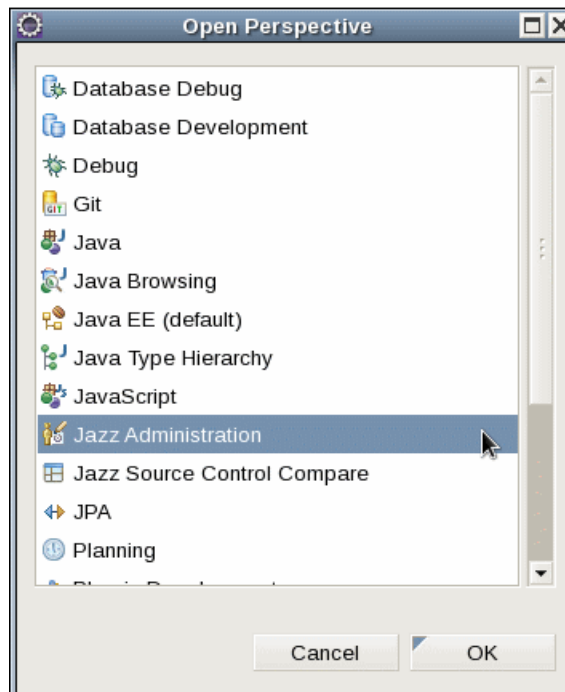


Figure 2-14 Select the Eclipse perspective

- b. Click **Create a new Repository Connection**, as shown in Figure 2-15.

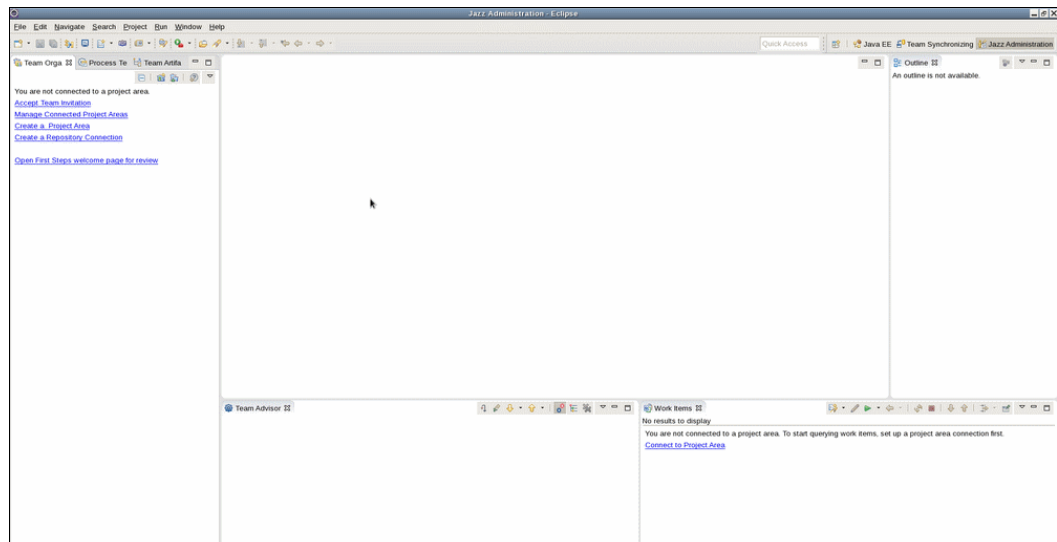


Figure 2-15 Jazz Administration - Eclipse window

In Figure 2-16, provide the connection details to connect to a Rational Team Concert Server. After you add the connection details, click **Finish**.

Create a Jazz Repository Connection

Jazz Repository Connection

Create a new Jazz repository connection.

Location

URI:

Name:

Authentication

Authentication Type:

User ID:

Password:

☒ Remember my password

☒ Automatically log in at startup

Advanced

Connection Timeout (in seconds):

Figure 2-16 Jazz Repository connection windows

- c. You are asked to accept the Secure Sockets Layer (SSL) certificate. Choose **Accept this certificate permanently**. Click **OK**, as shown in Figure 2-17. Now, you can create a local workspace where your Eclipse projects will be created.

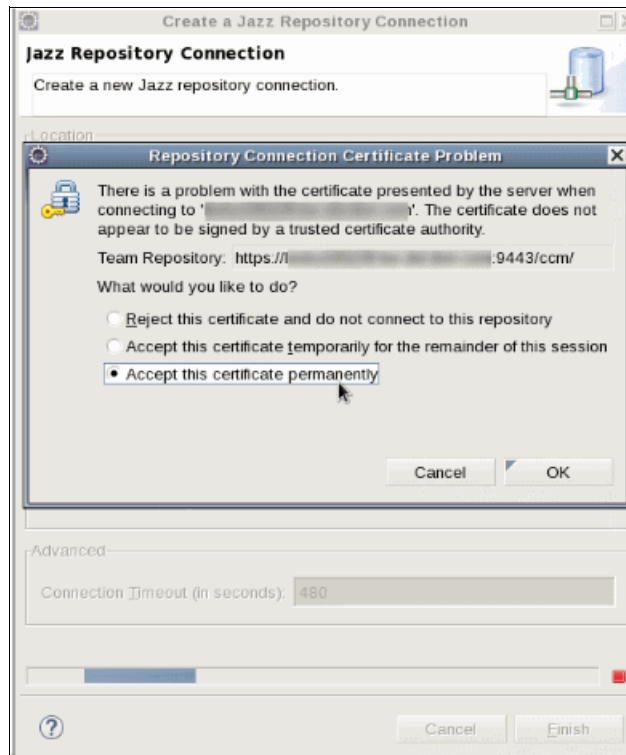


Figure 2-17 Accept the SSL certificate pop-up window

5. Configure a Liberty server in Eclipse:

- a. In the Eclipse client, navigate to **Window** → **Preferences**. Select **Servers** → **Runtime Environments**, as shown in Figure 2-18.

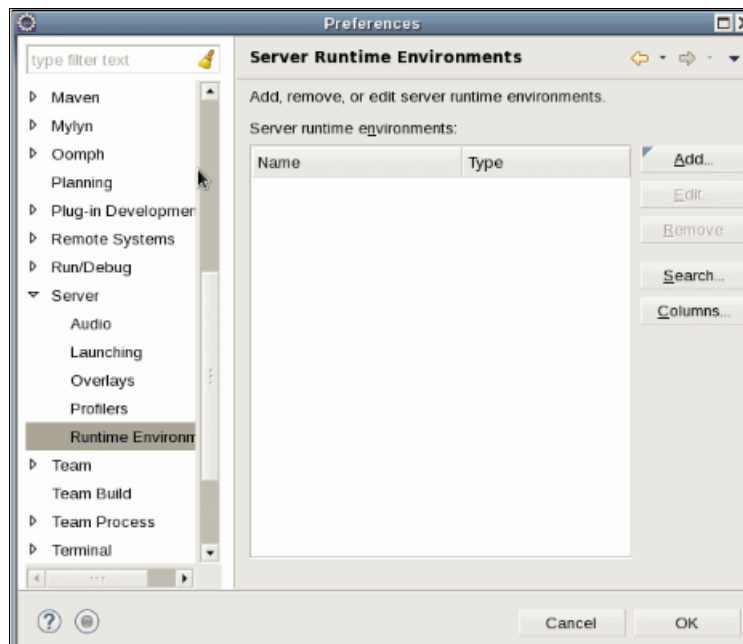


Figure 2-18 Eclipse Runtime Environment window

- b. Click **Add**.

- c. Select **WebSphere Application Server Liberty Profile Tools** from the list, as shown in Figure 2-19.

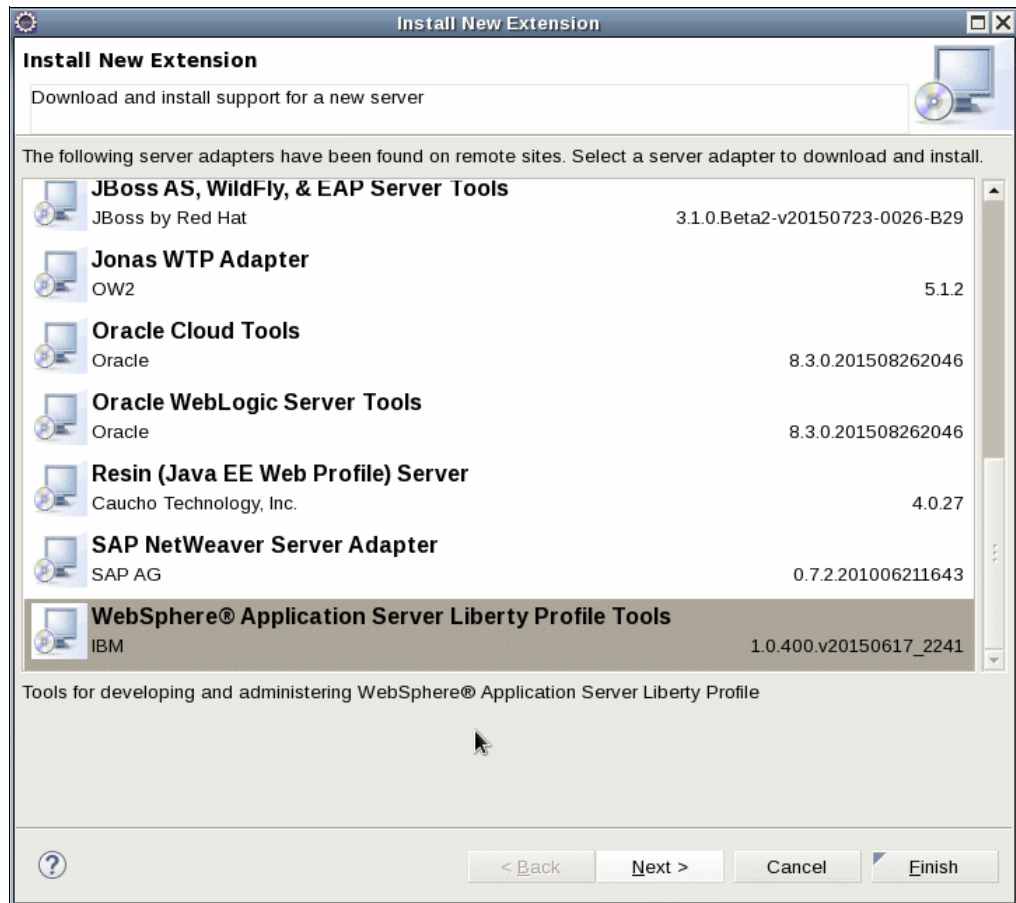


Figure 2-19 List of the application servers

- d. Click **Next** and accept the license agreement. Then, click **Finish** and wait for the installation to complete, as shown in Figure 2-20.



Figure 2-20 Installation completed successfully notification

- e. Click **OK**. The WAS Liberty server is now listed in the Server Runtime Environments window, as shown in Figure 2-21.

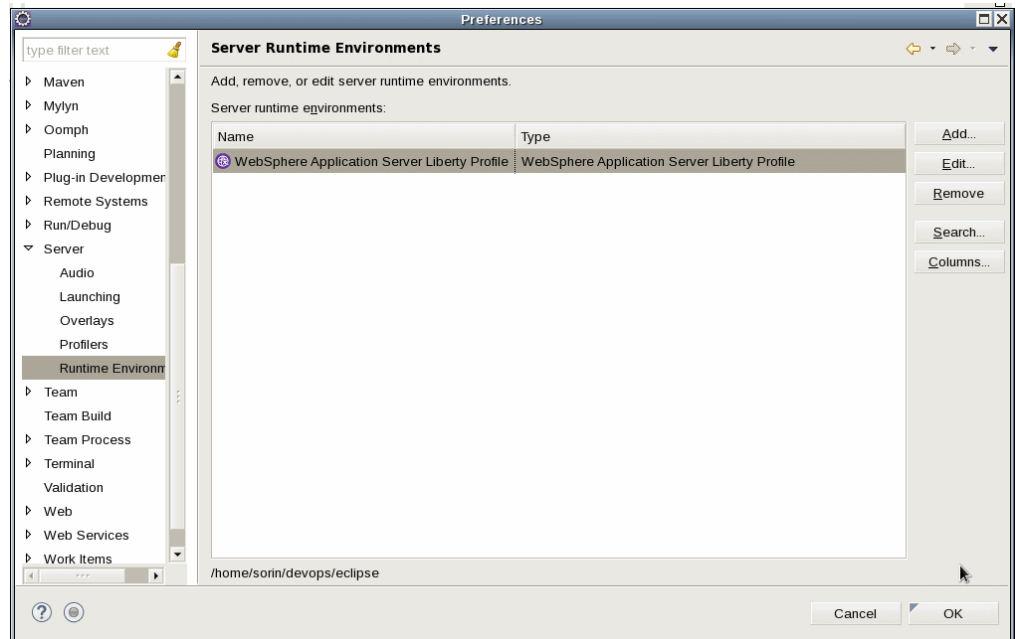


Figure 2-21 Eclipse Server Runtime Environments window

The developed applications can now be tested locally on the WAS Liberty server.

2.5 Rational Quality Manager for test planning

Together with Rational Team Concert and Rational Requirements Management, Rational Quality Manager is part of the IBM Jazz™ Team server, focusing on the test phase of software development projects.

Rational Quality Manager is a collaborative, web-based, quality management solution that offers comprehensive test planning, manual testing, and integration with other automated test tools.

Test planning describes the creation of a test strategy and the use of it, often for a specific period of time, such as an iteration or a sprint, or for a small project.

To explain the process of developing a test plan for a software development project by using Rational Quality Manager, we use the following scenario.

2.5.1 Acceptance criteria

The acceptance criteria section of a test plan is a statement that is agreed upon by the product owner and the testing team before the story is developed. For each user story, it is expected that agreed-upon acceptance criteria are included in Rational Team Concert.

2.5.2 Creating a test plan for your project

To create a test plan, navigate to the Rational Quality Manager menu, click **Planning**, and choose **Create Test Plan**, as shown in Figure 2-22.

The screenshot displays the 'Liberty Cloud Trader (Quality Management)' application interface. The top navigation bar includes 'Project Dashboards', 'Requirements', 'Planning', 'Construction', 'Lab Management', 'Builds', 'Execution', 'Reports', and 'Change Requests'. The 'Planning' menu is active. The main content area is titled 'Test Plans' and shows a form for creating a new test plan. The form includes a 'Summary' section with fields for 'State' (Draft), 'Action' (Change State), 'Originator' (Administrator), 'Owner' (Unassigned), 'Priority' (Unassigned), and 'Description' (Click here to enter a description). A 'Summary' box provides an overview of the test plan, including 'Categories' (Liberty CloudTrader Application) and 'Release' (Release 1.0). The form also features a 'Quality Task' section with a 'Create' button. The interface includes a sidebar with 'Sections' and a right-hand panel with 'Tracked by Quality' and 'Related Sites'.

Figure 2-22 Create a test plan

2.5.3 Creating test cases

After a test plan is created and approved, test cases can be created for each user story. Follow these steps:

1. Choose the test case priority (the same priority as the corresponding user story). Enter the test case weight, as shown in Figure 2-23.

The screenshot displays the 'Liberty Cloud Trader (Quality Management)' application. The main window is titled 'Enter New Test Case Name'. It features a sidebar on the left with a 'Sections' menu where 'Development Items' is highlighted. The main content area contains a form for creating a new test case. The form includes a 'Summary' section with the following details: State: Draft, Originator: Administrator, Priority: Unassigned, and Weight: 100. The 'Categories' section shows 'Development Activity' as the Function and 'Developer Test' as the Test Phase. The form also has a 'Description' field with a placeholder text '< Click here to enter a description >'. The interface includes a top navigation bar with various tabs like 'Project Dashboards', 'Requirements', 'Planning', 'Construction', 'Lab Management', 'Builds', 'Execution', 'Reports', and 'Change Requests'. A right sidebar shows 'Parent Test Plan(s)', 'Related Test Suite', and 'Related Sites'.

Figure 2-23 Create a test case

2. Click **Development Items** from the menu on the left.

3. Click the plus (+) sign and select the corresponding user story, as shown in Figure 2-24.

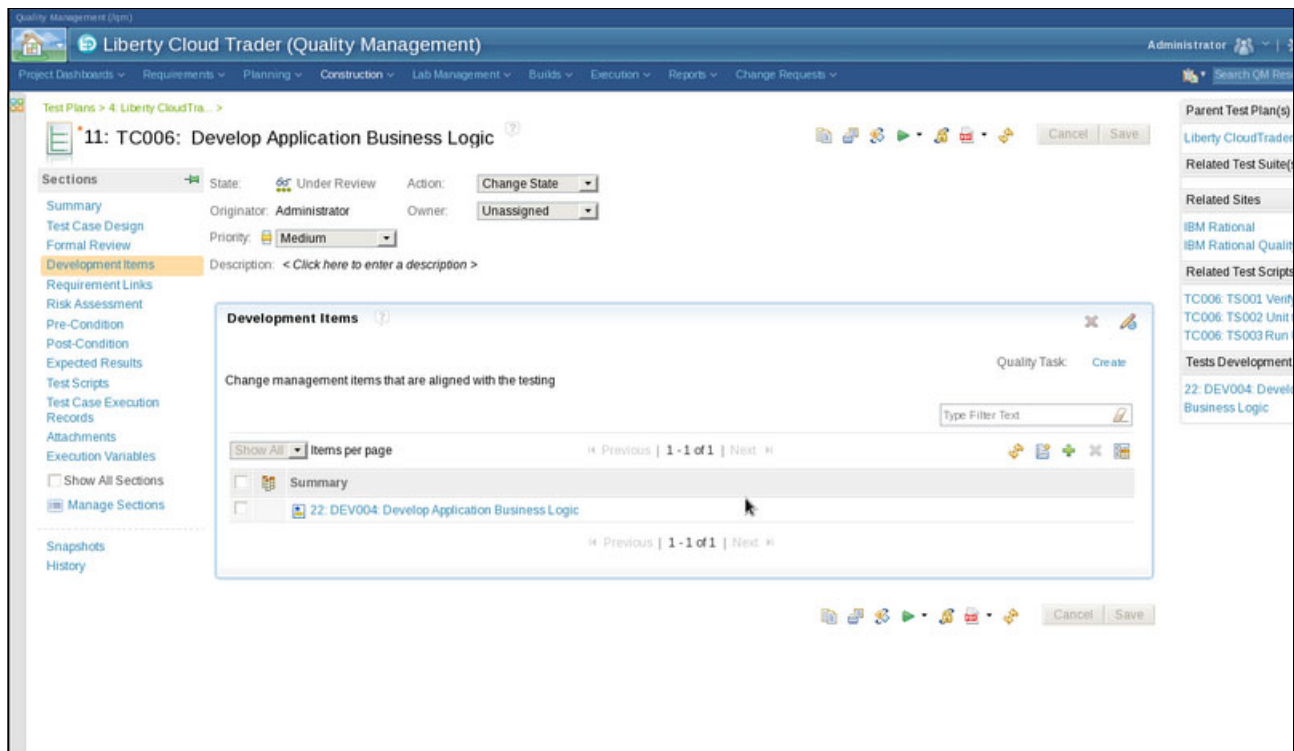


Figure 2-24 Link a test case with an existing user story

2.5.4 Linking the test cases to the test plan

Each test plan contains multiple test cases that belong to several user stories. (Now, the platforms where the test cases are executed can be added to the test plan, if they are known, or they can be selected later, when the test case execution records are created.) See Figure 2-25.

Test Cases ?

Lists the test cases associated with a given plan. You can add and remove associations to test documents and create and associate a new test case. A test case will remove the association to this test plan but not delete the test case.

View As: **General** Group By: **Ungrouped**

Show All Items per page Previous | 1 | Next

| <input type="checkbox"/> | ID | Suspect | Priori | Name ^1 | State | Owner | Created By | Func |
|--------------------------|----|---------|--------|--|-------------|------------|---------------|---------|
| <input type="checkbox"/> | 6 | ◆ | ■ | TC001: Requirement Management: Def... | Under Re... | John Doe | Administrator | Archite |
| <input type="checkbox"/> | 7 | ◆ | ■ | TC002: Create Application Architecture | Under Re... | John Doe | Administrator | Archite |
| <input type="checkbox"/> | 8 | ◆ | ■ | TC003: Define Application UML Model | Under Re... | John Doe | Administrator | Archite |
| <input type="checkbox"/> | 9 | ◆ | ■ | TC004: Identify the third-party tools and I... | Under Re... | John Doe | Administrator | Develo |
| <input type="checkbox"/> | 10 | ◆ | ■ | TC005: Create Database Shema | Under Re... | Unassigned | Administrator | Develo |
| <input type="checkbox"/> | 11 | ◆ | ■ | TC006: Develop Application Business L... | Under Re... | Unassigned | Administrator | Develo |
| <input type="checkbox"/> | 12 | ◆ | ■ | TC007: Develop application UI | Under Re... | Unassigned | Administrator | Develo |
| <input type="checkbox"/> | 13 | ◆ | ■ | TC008: Develop a REST API for Liberty ... | Under Re... | John Doe | Administrator | Develo |
| <input type="checkbox"/> | 14 | ◆ | ■ | TC009: Create a build definition for Libe... | Under Re... | Unassigned | Administrator | Develo |
| <input type="checkbox"/> | 15 | ◆ | ■ | TC010: Create a Mobile Version of Liber... | Under Re... | Jane Smith | Administrator | Develo |

Showing 1-10 of 10 items Previous | 1 | Next

Figure 2-25 List of the test cases that are assigned to a test plan

2.5.5 Creating the test scripts

The test scripts describe the actual test steps to perform during testing to validate the outcome that is produced in a user story. Each test script contains several steps, and each step has expected results. Create a test script. Navigate to the Rational Quality Manager main menu and click **Execution**. From the list, choose **Create New Test Script**, as shown in Figure 2-26.

The screenshot shows the 'Liberty Cloud Trader (Quality Management)' application interface. The top navigation bar includes 'Project Dashboards', 'Requirements', 'Planning', 'Construction', 'Lab Management', 'Builds', 'Execution', 'Reports', and 'Change Requests'. The 'Execution' menu is expanded, showing 'Test Scripts' as the selected option. The main form is titled '< Enter New Test Script Name >'. It includes a 'State' dropdown set to 'Draft', an 'Action' dropdown set to 'Change State', and a 'Work Item' dropdown set to 'Create'. The 'Originator' is 'Administrator' and the 'Owner' is 'Unassigned'. The 'Type' is 'Manual' and the 'Test Data' is 'Unassigned'. A 'Description' field is present with a placeholder '< Click here to enter a description >'. Below the form, there is a 'Manual Steps' section showing 'step 1 to 1 of 1'. The step is numbered '1' and has fields for 'Description' and 'Expected Results'. A 'Quick Tips' section at the bottom provides instructions on how to add new steps and navigate between cells.

Figure 2-26 Creating test scripts

2.5.6 Linking test scripts to a test case

Each test script is associated with a test case. To link test scripts to a test case, navigate to the test case and, from the menu on the left, choose **Test Scripts**. Click the plus (+) sign to add test scripts to a test case, as shown in Figure 2-27.

The screenshot shows the Rational Quality Manager interface. At the top, the breadcrumb is 'Test Plans > 4: Liberty CloudTra... >'. The main header is '11: TC006: Develop Application Business Logic'. On the left, a 'Sections' menu lists various options, with 'Test Scripts' highlighted. The main area displays the 'Test Scripts' section for this test case. It includes a description: 'Automated or manual test scripts that are associated with this test case. Scripts can be reused.' Below this, there's a 'Group By' dropdown set to 'Ungrouped' and a 'Show All' button. A table lists the test scripts:

| | ID | Name | State | Script Type | Owner | Data Record |
|--------------------------|----|-------------------|----------|-------------|-----------|-------------|
| <input type="checkbox"/> | 4 | TC006: TS001 V... | Under... | Manual | Unassi... | Unassigned |
| <input type="checkbox"/> | 5 | TC006: TS002 U... | Under... | Manual | Unassi... | Unassigned |
| <input type="checkbox"/> | 6 | TC006: TS003 R... | Under... | Comman... | John Doe | Unassigned |

Navigation links at the bottom of the table include 'Previous', '1 - 3 of 3', and 'Next'.

Figure 2-27 Linking test scripts to a test case

2.5.7 Reviewing the test case

As an optional step, the story owner and other team members can review and approve the test case.

2.5.8 Creating test case execution records

The test case execution records must be created for each test case (one or more, depending on the number of platforms where the test case is intended to be tested).

Test case execution records serve as proof that test steps were executed on a test case, with the result of passed or failed. Test results of passed confirm the validity of a user story.

If the status is failed, open a defect and include information about the failure, such as execution logs and screen captures. The defect can be created directly in Rational Quality Manager. The defect is assigned automatically to the user story to which the test case is linked. After development fixes the defect, the failing step (or the entire Test Case Execution Record (TCER)) is rerun to ensure that the defect is fixed.

If the status is passed, the user story was successfully tested. The status can be changed to Complete Testing. The deliverables of that user story can be included in an official build, as shown in Figure 2-28.

Description:

Test Case Execution Records

This section lists the related test case execution records for the test plan.

View As: **General** Group By: **Ungrouped**

Show All Items per page

Previous | 1 | Next

| | ID | Priority | Name | Test Case | Test Environment | Iteration | Owner | Created By | Test Script | Last Result | Last Modified |
|--------------------------|----|----------|----------------------------|------------------|------------------|------------|---------------|---------------|----------------------|-------------|---------------|
| <input type="checkbox"/> | 4 | | TC006: Develop Applicat... | TC006: Develo... | Unassigned | Unassigned | Administrator | Administrator | TC006: TS002 Un... | | 4 hours |
| <input type="checkbox"/> | 3 | | TC001: Requirement M... | TC001: Requir... | Unassigned | Unassigned | Administrator | Administrator | Liberty Cloud Tra... | | 5 hours |

Showing 1-2 of 2 items

Previous | 1 | Next

Figure 2-28 Test case execution records



Code development, source code management, and build

This chapter explains the stages of code development by using agile practices. This chapter explains how to improve the quality of a project by focusing the project team on high-quality software development, unit testing, and team collaboration.

To learn more about agile principles, go to this link:

<http://www.ibm.com/software/rational/agile/resources/>

The following topics are described in this chapter:

- ▶ How developers code and collaborate by using Rational Team Concert
- ▶ Jenkins as an alternative for building code

3.1 How developers code and collaborate by using Rational Team Concert

Rational Team Concert is the software development collaboration tool that is used for code development and collaboration activities. We used Rational Team Concert for code development and planning of the LibertyCloudTrader scenario in this publication. This section introduces the “golden” rules of coding and an overview of unit tests, code reviews, and code delivery by using Rational Team Concert.

3.1.1 Coding

Coding is the part of application lifecycle management (ALM) that produces an application executable object. From a DevOps perspective, it is important to write quality code. The following rules apply to writing quality source code. These rules are independent of the programming language.

The golden rules of coding

The following rules guarantee a high level of code quality. Consider them as required characteristics when you write code:

- ▶ **Efficiency.** Combine a simple code with efficiency. Use a framework for your programming language with efficient data structures and algorithms.
- ▶ **Maintainability.** Write descriptive code that makes it simple to read:
 - Create descriptive names and avoid ambiguity. For example, `calculateAverage()` is more descriptive than `calculate()`.
 - Use comments with descriptions at the beginning of each class and method, and remember to write comments as you develop the code.
 - Follow a style guide and indent your code. Spaces and brackets are important to a colleague who is reading your code.
 - Never repeat code. If you have common parts in a class or method, abstract the common code into another class or method and use it with parameters.
 - Code splitting. Divide long code blocks into smaller, consistent sequences, and use similar logic for similar problems.
 - Make your code portable. Never use functionality that is available only in the framework that you are using.
 - Do not include hardcoded values in source code. Environment variables, such as IP addresses, ports, or host names, need to be parameterized. Otherwise, the application cannot run on a different system.

Sample application

LibertyCloudTrader is an application that simulates an online stock trading system. The program enables users to perform these tasks:

- ▶ View the portfolio
- ▶ Look up stock quotes
- ▶ Buy or sell shares

LibertyCloudTrader is built primarily with Java servlets, JavaServer Pages (JSPs), JavaBeans, and web services. The program is based on CloudTrader, a 10-year old trading application.

Figure 3-1 shows the LibertyCloudTrader main window.

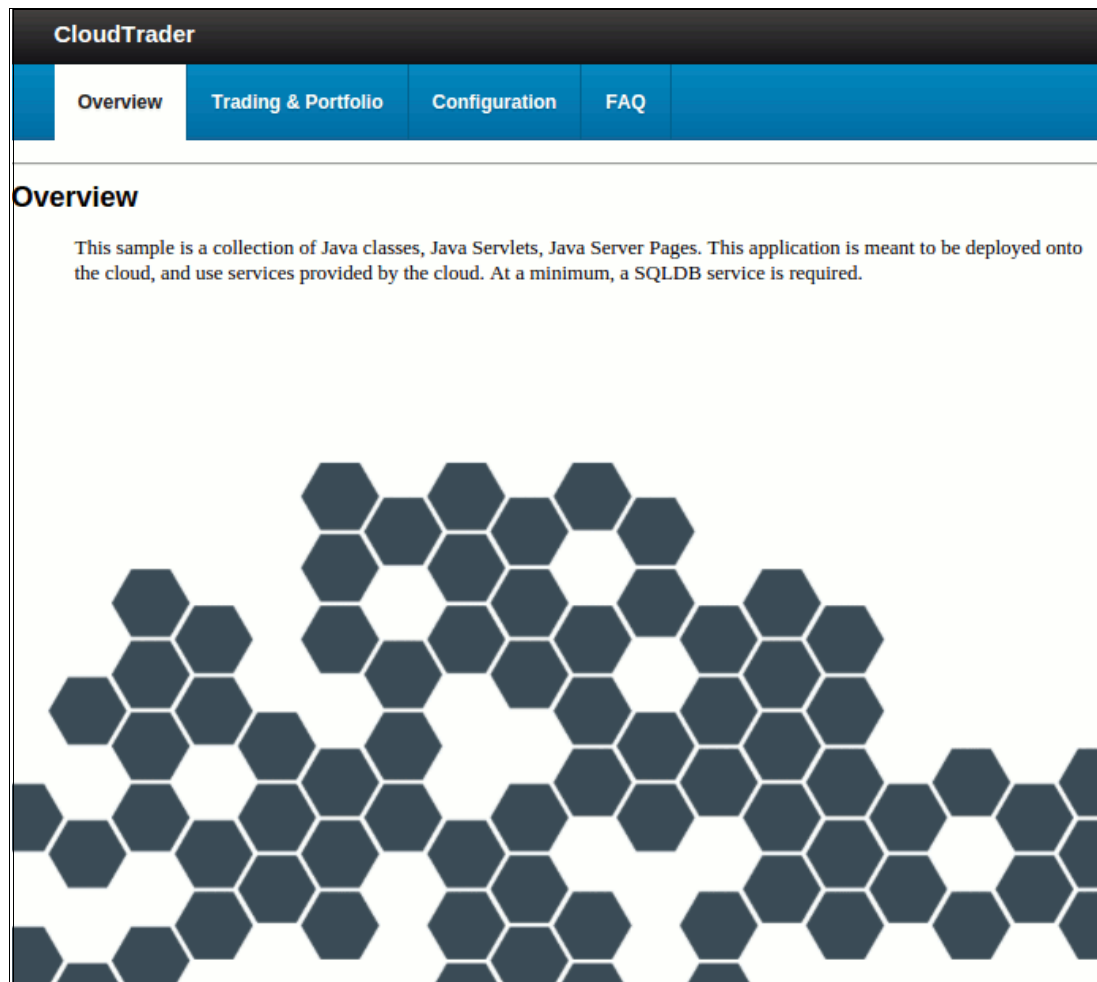


Figure 3-1 The LibertyCloudTrader main window

3.1.2 Unit testing

Unit testing is an important part of code development. Code needs to be tested continuously during the development process.

Unit tests are written from the programmer's perspective. These tests ensure that a particular method successfully performs a set of specific tasks. Each test confirms that a method produces the expected output when a known input is provided.

LibertyCloudTrader unit testing example

In LibertyCloudTrader, the unit tests are in the Java `com.ibm.samples.trade.unitTest` package.

The unit test in the `AccountRestTest` file checks that the `SERVER_URL/api/account` web service is online.

How to run unit tests

The unit test framework that is used in this chapter is JUnit. Verify that the JUnit Java archive (jar) file is in the lib directory in the Java project. If not, download the JUnit jar file from <http://junit.org>, and save the jar file in the lib directory.

To run the unit tests, select **AccountRestTest.java** → **Run As** → **JUnit Test**, as shown in Figure 3-2.

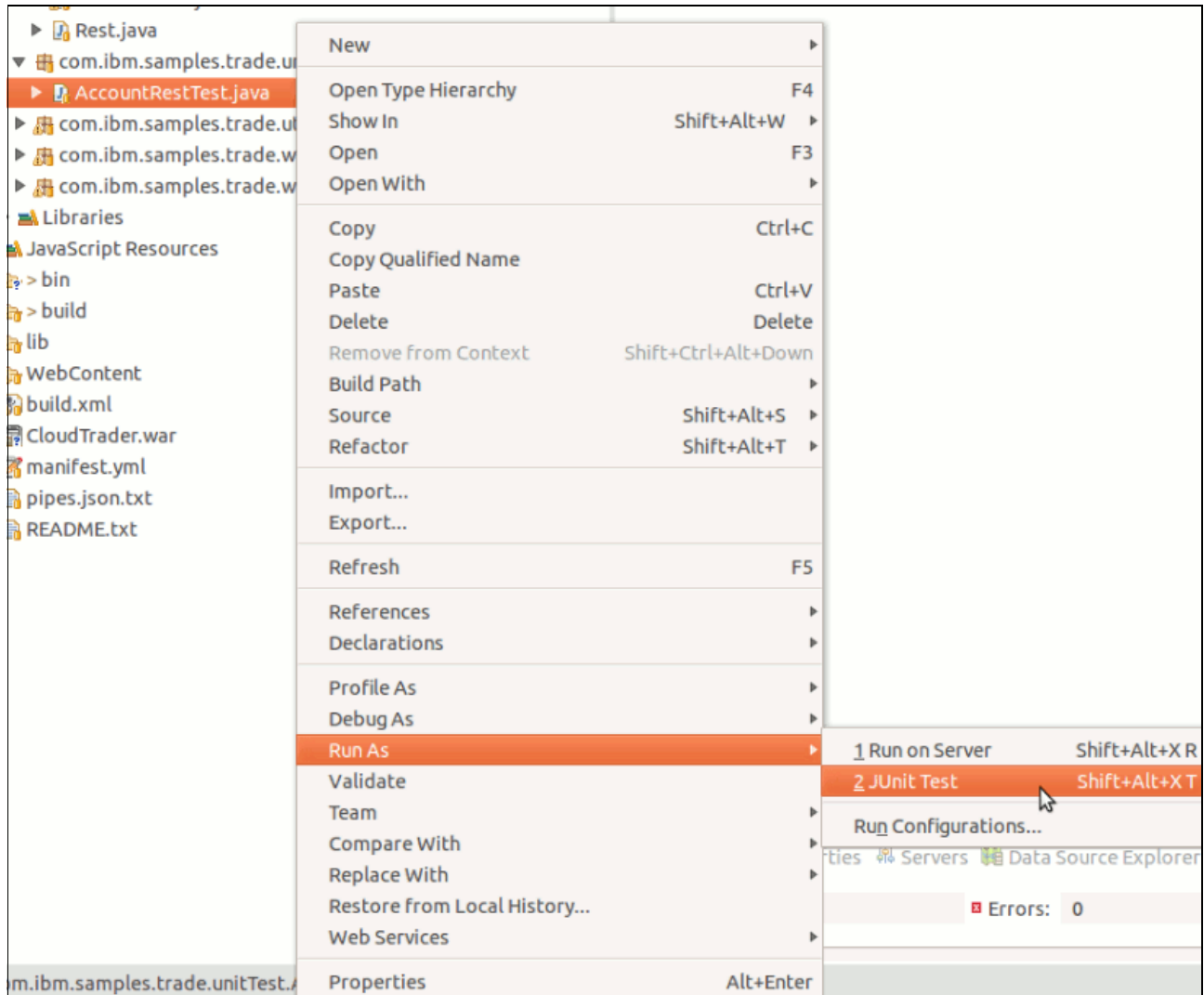


Figure 3-2 Running unit tests

The results of our unit tests are shown in Figure 3-3. If the test is executed successfully, a green list with one item for each unit test is displayed. If errors occur in your unit test, double-click the problematic item to investigate the issue.

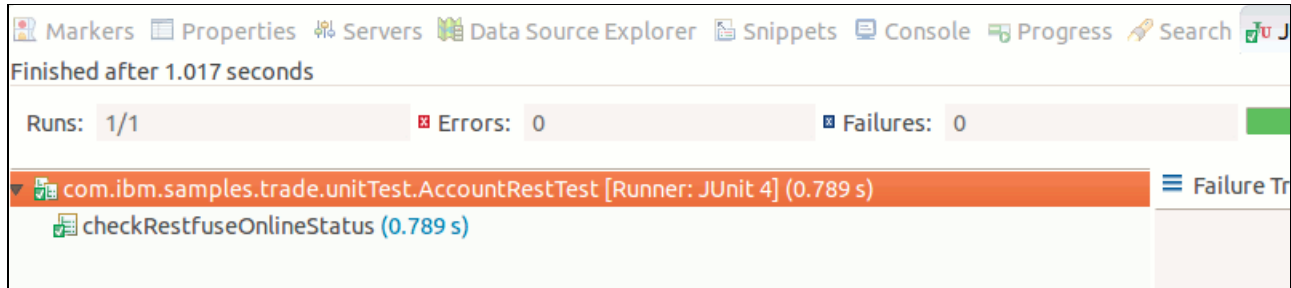


Figure 3-3 Unit test results

3.1.3 Code reviews

Code reviews, which are performed by the developer's peers, are analyses of the source code. Code reviews are intended to identify and fix errors that were overlooked during the development phase, therefore improving both the overall quality of software and the developers' skills. Code reviews are an important aspect of code quality. The code reviews are often overlooked, because formal code reviews require a considerable investment in preparation for the review event and execution time. We describe how the review effort (the code review metric) can be set up as part of the project process. This way, you can gain insight into what you can gain compared to the investment.

The following high-level steps are necessary for code reviews for your teams when you use Rational Team Concert:

1. Open Rational Team Concert and click **Window** → **Open Perspective** → **Work Items** to enable the work items perspective. Select **Pending Changes**, as shown in Figure 3-4.

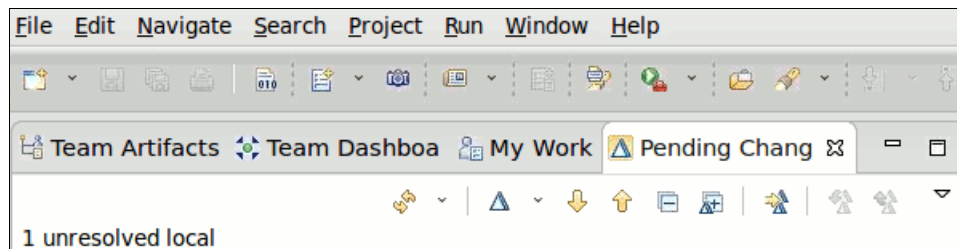


Figure 3-4 Pending changes

A new folder that is named unresolved displays in your workspace folder. This folder has all of the files with your changes.

2. Next, create a change set. Click **Unresolved** → **Check-in** → **New Change set**, as shown in Figure 3-5.

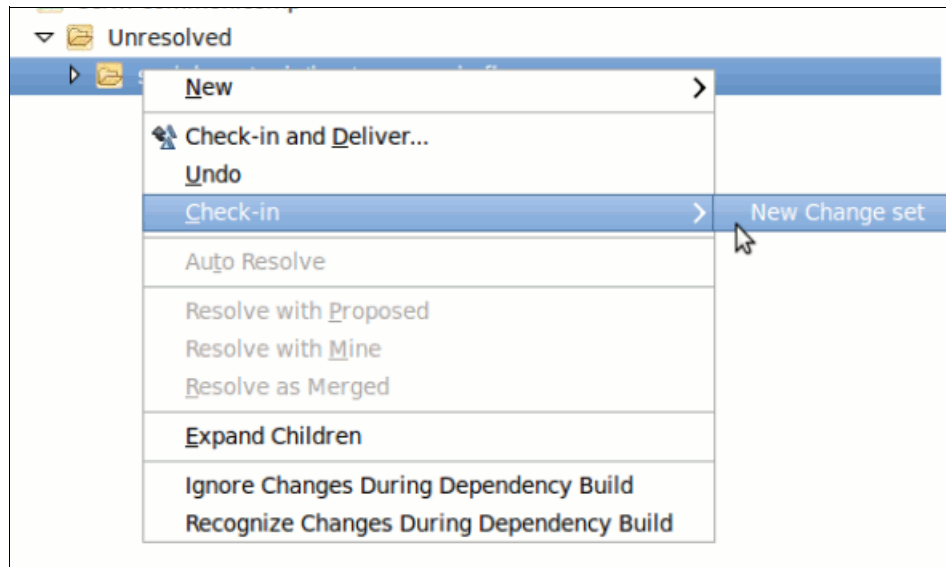


Figure 3-5 Creating a change set

3. Associate the change set with the work item of the story that you are working with, as shown in Figure 3-6. See 2.3.3, “User story” on page 25.

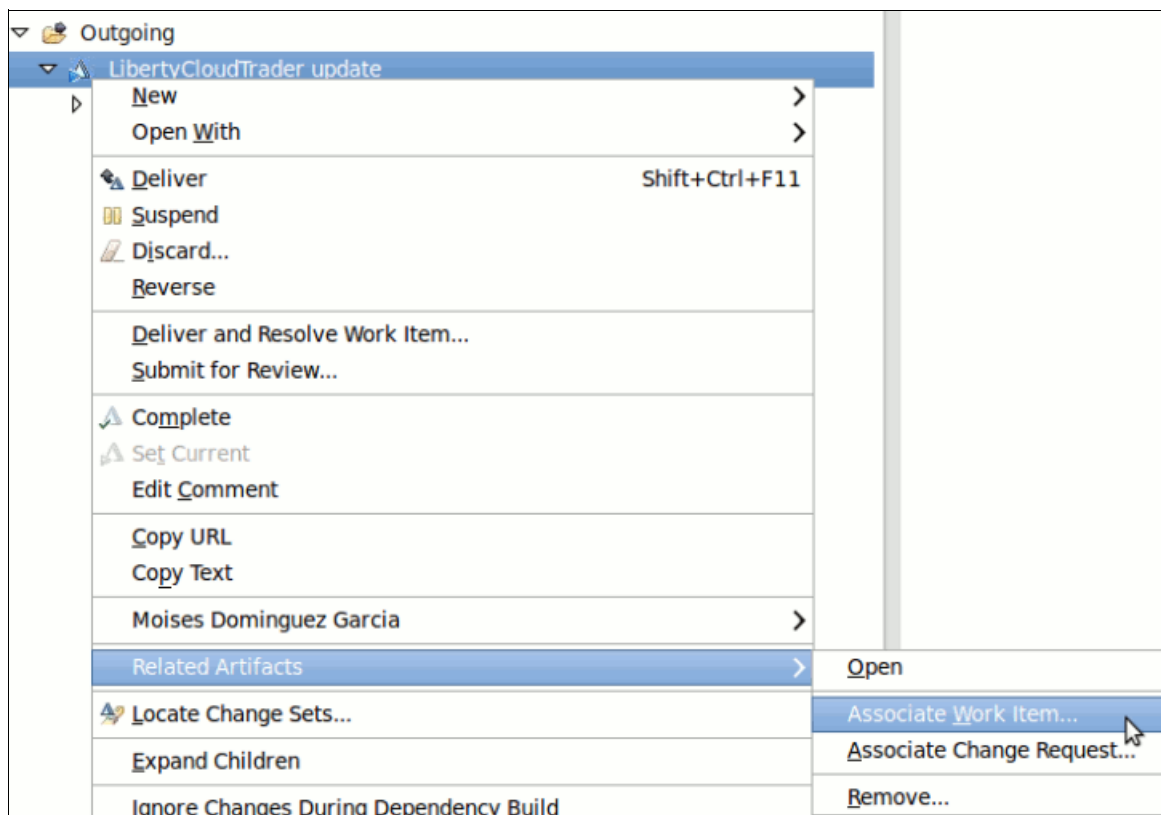


Figure 3-6 Associating a work item with the change set

4. Right-click in the change set and click **Submit for Review** to submit the code for review, as shown in Figure 3-7.

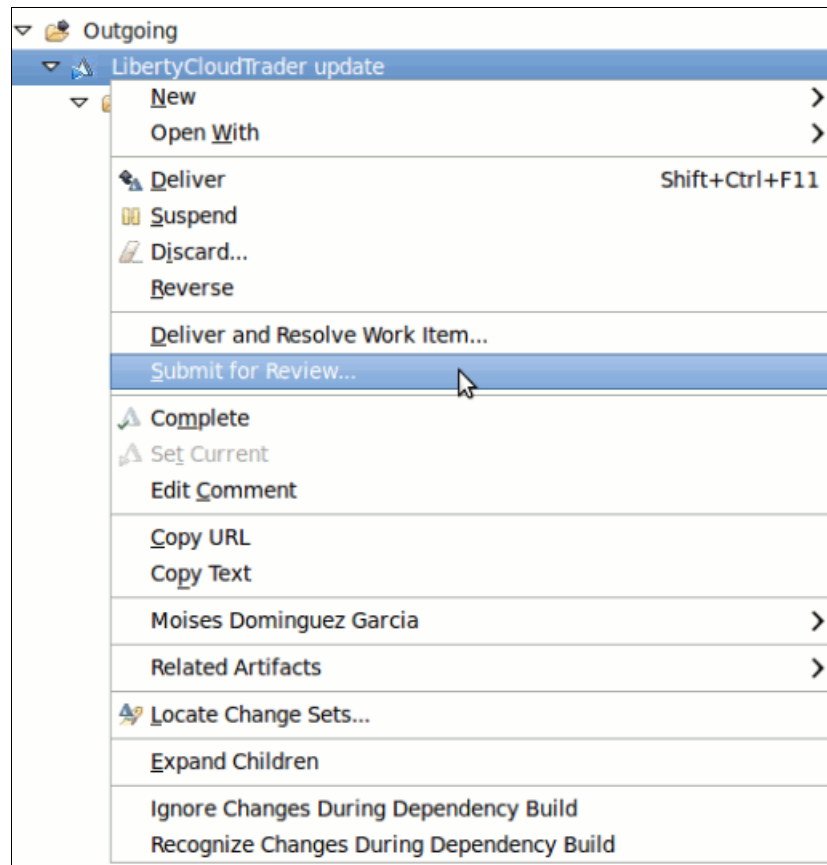


Figure 3-7 Right-click in the change set and select Submit for Review

5. Add a comment to the work item, and select two reviewers, as shown in Figure 3-8.

Submit Change sets for Review

Submit for Review

✖ A reviewer and review subject is required.

☐ Suspend change sets

Add a comment to the work item:

Include here the comments of the changeset

Approval Configuration

Select the approvers for these change sets:

Subject:

Due:

Figure 3-8 Submitting a change set for review and approval

6. Click **Finish** and wait for approvals from the reviewers.

3.1.4 Code delivery

Code delivery is the process of transferring the file changes from a developer's isolated workspace to the code stream that is associated with the workspace. After code delivery, the new versions of the files are stored within the stream and they are available for other developers who are associated with the stream. Other developers are informed by Rational Team Concert of the presence of new content, and then they can accept the changes.

When all of the approvals for the change set are accepted, the developer can deliver the code. Select **Deliver** in the change set to deliver the code, as shown in Figure 3-9.

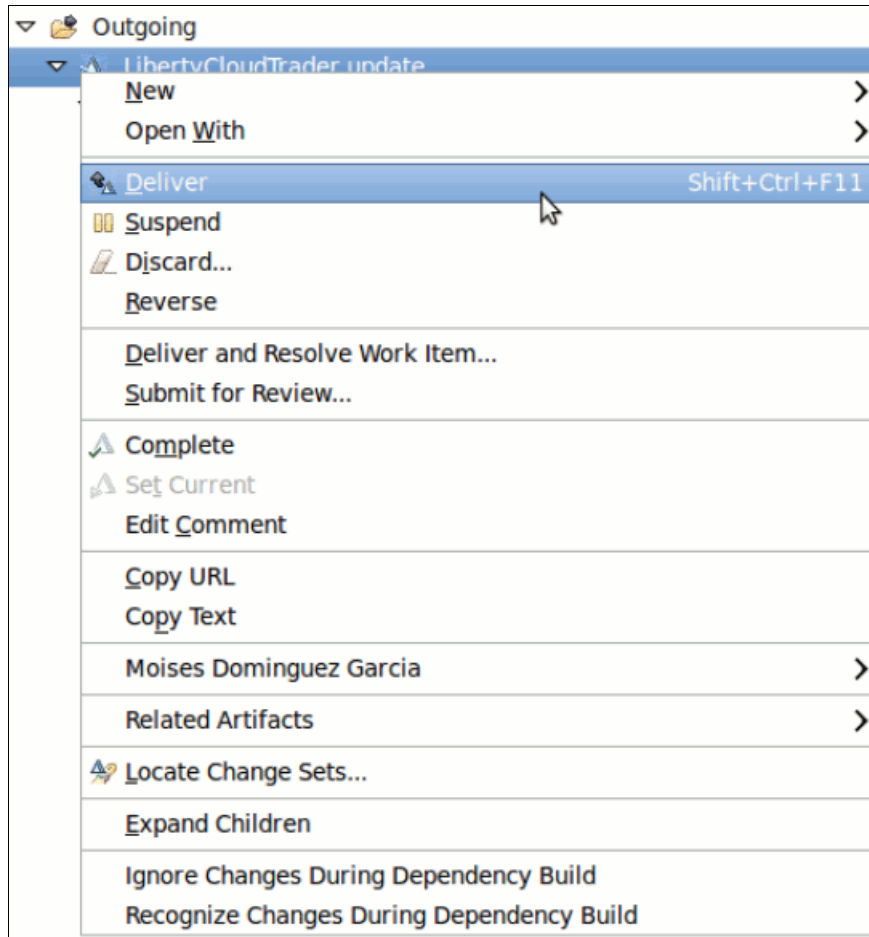


Figure 3-9 Delivering code

3.2 Jenkins as an alternative for building code

This section describes the Rational Collaborative Lifecycle Manager (RCLM) and Jenkins integration for building code. This integration enables Jenkins to build and, if required, deploy application artifacts that are based on an RCLM snapshot.

The following products are prerequisites:

- ▶ Jenkins v1.6 or later
- ▶ RCLM v5 or later
- ▶ Ant v1.8 or later
- ▶ Rational Team Concert Build Toolkit at the same level as RCLM

Note: The installation process for the required products is not included in this section. For installation instructions, see the following links:

- ▶ Jenkins:
<https://jenkins-ci.org/>
- ▶ RCLM:
http://www.ibm.com/support/knowledgecenter/SSYMRC_5.0.2/com.ibm.rational.clm.doc/helpindex_clm.htm
- ▶ Ant:
<http://ant.apache.org/>

3.2.1 Install and configure the Rational Team Concert plug-in on Jenkins

Install the Rational Team Concert plug-in on Jenkins:

1. From the Jenkins dashboard, click **Manage Jenkins** → **Manage Plug-ins**.
2. Select the **Available** tab. Select **Team Concert Plug-in**.
3. Select **Download Now and install after restart**, as shown in Figure 3-10.

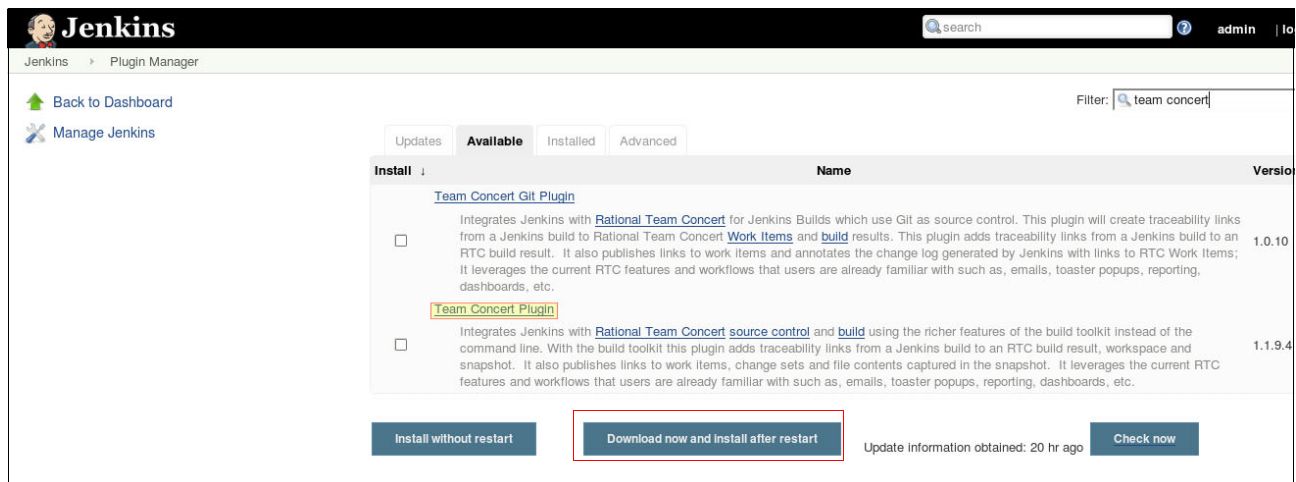


Figure 3-10 Jenkins plug-in search interface that shows the Team Concert plug-in

Configure the Rational Team Concert Build Toolkit

After the Rational Team Concert plug-in is installed, configure the Rational Team Concert Build Toolkit that is used to connect Jenkins with Rational Team Concert:

1. At the Jenkins management console, select **Manage Jenkins** → **Configure System**.
2. Scroll down to the **RTC Build toolkit**, and select **Add RTC Build toolkit**.
3. Define a toolkit name and path (the RTC Build toolkit must be installed locally on the Jenkins server). If the path is invalid, an error message displays.

Figure 3-11 shows a configuration example of Build Toolkit.

The screenshot shows the 'RTC Build toolkit' configuration page. It has a header 'RTC Build toolkit' and a sub-header 'RTC Build toolkit installations'. Below this, there's a form with the following fields: 'Name' (set to 'RTC Build Kit 5.0.2'), 'Home' (set to '/opt/IBM/TeamConcertBuild_5.0.2/buildsystem/buildtoolkit'), and a checkbox 'Install automatically' which is unchecked. There are help icons (question marks) next to each field. At the bottom right is a red button 'Delete RTC Build toolkit'. At the bottom left is a grey button 'Add RTC Build toolkit'. Below the buttons is a small text 'List of RTC Build toolkit installations on this system'.

Figure 3-11 Configuration example of Build Toolkit

3.2.2 Configure the Rational Team Concert server connection

The Rational Team Concert connection is the link that is used by Jenkins to access RCLM repositories and build definitions. Configure the connection:

1. At the Jenkins management console, select **Manage Jenkins** → **Configure System**.
2. Scroll down to **Rational Team Concert**.
3. At the Build toolkit list box, select the toolkit that was defined previously.
4. For Server URI, type the URL for the RCLM server (`https://<host name>:9443/ccm`).
5. Create a credential with the user name and password of the RTC user.
6. Click **Test connection**.

Now, you can use a Rational Team Concert connection to access RCLM from Jenkins.

Figure 3-12 shows an example of a Rational Team Concert server connection definition.

The screenshot shows the 'Rational Team Concert (RTC)' configuration page. It has a header 'Rational Team Concert (RTC)'. Below this, there's a form with the following fields: 'Build toolkit' (a dropdown menu set to 'RTC Build Kit 5.0.2'), a checkbox 'Avoid using build toolkit on Master (experimental)' which is unchecked, 'Server URI' (set to 'https://lexbz181128.cloud.dst.ibm.com:9443/ccm'), 'Connection timeout (in seconds)' (set to '480'), and 'Credentials' (a dropdown menu set to 'administrator/***** (RTC admin user)' with an 'Add' button next to it). There are help icons (question marks) next to each field. At the bottom right is a grey button 'Test connection'.

Figure 3-12 Rational Team Concert server connection definition

3.2.3 Create a Jenkins build engine

Use a build engine to define a tool to use for the application build, in this case, Jenkins. The Rational Team Concert client uses this engine definition to send build jobs to Jenkins.

Follow these steps to create a Jenkins build engine:

1. In the Team Artifacts view, in the project area, expand the **Builds** folder.
2. Right-click **Build Engines**, then click **New Build Engine**.

3. On the New Build Engine page, ensure that **Create a new build engine** is selected, and then click **Next**.
4. On the General Information page, in the ID field, enter a build engine identifier.
5. In the Available build engine types pane, select **Hudson/Jenkins Engine**, and then select **Finish**.
6. In the Build Engine editor, click the **Hudson/Jenkins** tab.
7. Configure the connection settings by entering the following information:
 - Host name
 - Port number
 - User name
 - Password
8. Click **Test Connection**. Figure 3-13 shows an example of the build engine configuration form.

The screenshot shows the 'Build Engine' configuration window. At the top, there's a title bar with the text '*Liberty Profile Cloud Trader Jenkins engine'. Below it, the 'Build Engine' tab is active. The form has two main sections: 'Hudson/Jenkins Server Information' and 'Test Connection'.

Hudson/Jenkins Server Information:

- ID:** Liberty Profile Cloud Trader Jenkins engine
- Project or Team Area:** Liberty Profile Cloud Trader
- Hudson/Jenkins URL:** http://lexbz181142.cloud.dst.ibm.com:8080/
- Validate Hostname:** Checked
- Authorization Required:** Checked
- User Id:** admin
- Password:** (masked with dots)

Test Connection:

- Test Connection:** Button
- Connection test requested.**
- Connecting to:** http://lexbz181142.cloud.dst.ibm.com:8080/
- Found header:** X-Jenkins - 1.609.2
- Found header:** X-Hudson - 1.395
- Test connection SUCCESSFUL!**

At the bottom, there are tabs for 'Overview' and 'Hudson/Jenkins'.

Figure 3-13 Build engine configuration form

3.2.4 Create a Jenkins build definition

A build definition is an element that stores the build configuration and dependencies. In this case, we select a Jenkins job to trigger the build action. Follow these steps:

1. Right-click **Builds**, and click **New Build Definition**.
2. The New Build Definition wizard begins. On the New Build Definition page, accept the default settings, and click **Next**.
3. On the General Information page, define a Build Definition ID, select **Hudson/Jenkins Build** in the Available build templates pane, and then click **Next**.
4. On the Pre-Build page, select **Jazz Source Control**. This setting enables repository access for Jenkins.
5. On the Additional Configuration page, ensure that all of the options are selected, and then click **Finish**.
6. In the Build Definition editor, on the Overview page, in the Supporting Build Engines area, click **Add**.
7. In the Add Build Engines window, select the correct Jenkins build engine, and click **OK**.

8. Click the **Hudson/Jenkins** tab.
9. Click **Get Jobs**, and then define a Jenkins job to run. (This job is the Jenkins job that builds the application. Job creation is explained in 3.2.5, “Create a Jenkins job by using the Rational Team Concert plug-in” on page 58.)
10. Click the **Jazz Source Control** tab, select the workspace that contains the component to build. For the Load Directory, enter the Jenkins job working directory path. (The Jenkins Rational Team Concert plug-in obtains the latest software version on this path.)
11. Click **Save**.
12. To test the build definition, right-click the build definition that was just created, select **Request build**, and click **Submit**. You can see the build result from the Build tab of Rational Team Concert. Figure 3-14 shows the Jenkins configuration tab of the Build Definition.

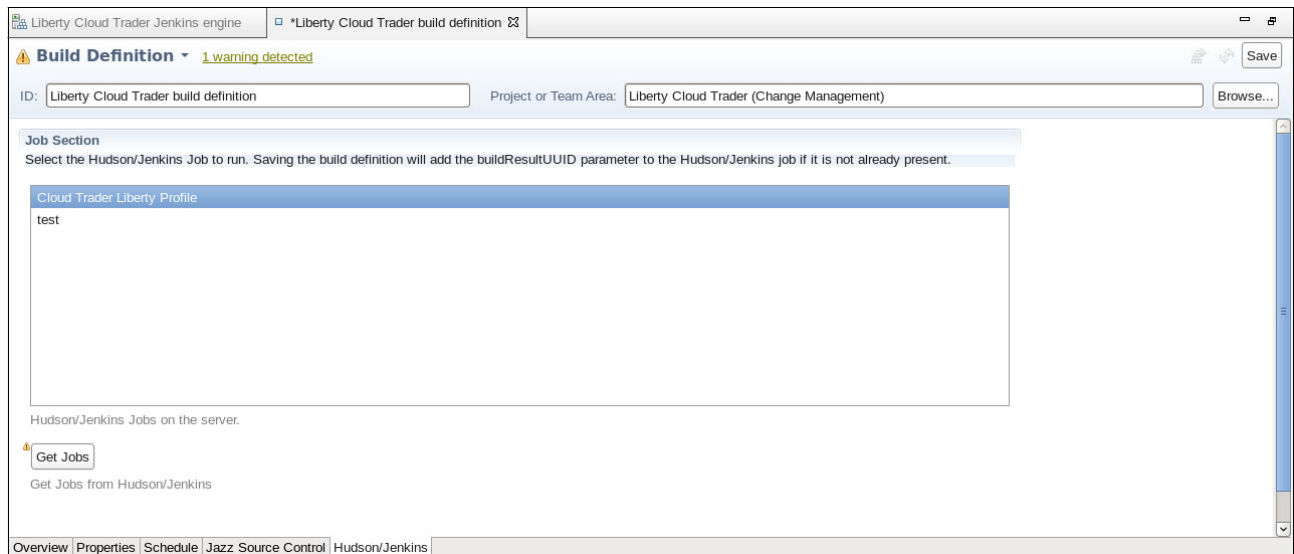


Figure 3-14 Hudson/Jenkins configuration tab of the build definition

Figure 3-15 shows the overview page of the configuration.

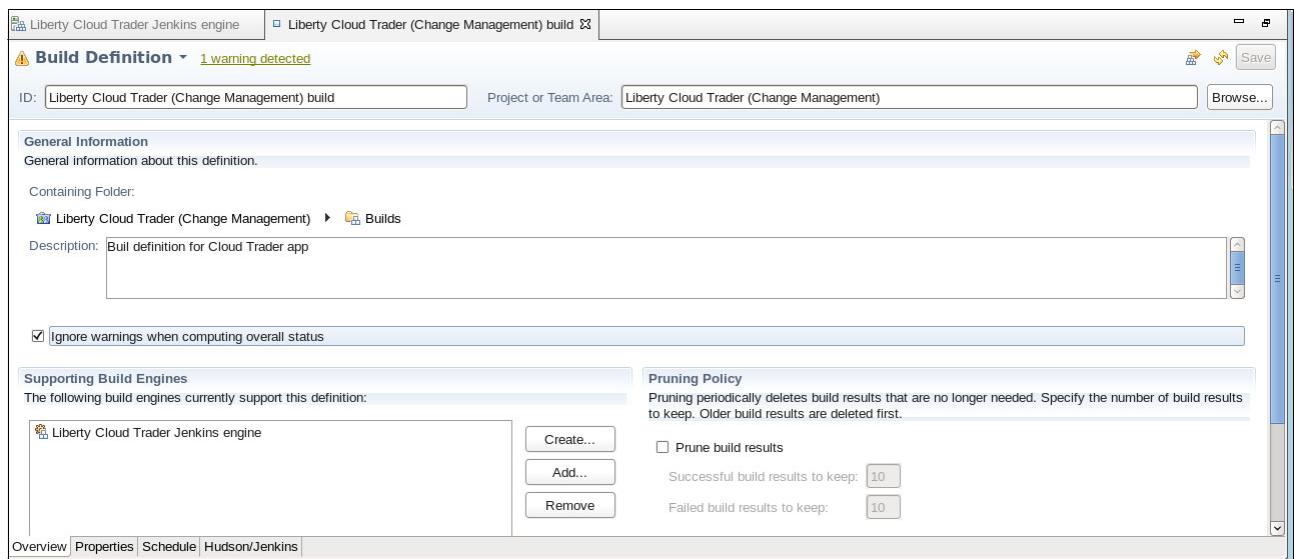


Figure 3-15 Overview tab of the build definition process

3.2.5 Create a Jenkins job by using the Rational Team Concert plug-in

The Jenkins job is the final piece of the Rational Team Concert and Jenkins integration. This job takes the software source from the RCLM repository and executes an Ant script to build the application. Follow these steps:

1. At the Jenkins management console, select **New Item**. Define the job name as Cloud Trader Liberty Profile. Select **Multi-configuration project**, and click **OK**.
2. Under Source Code Management, select **Rational Team Concert**.
3. In the Build definition text box, enter the name of the Build Definition that we created, and then click **Validate**. (An error message displays if the build Definition ID does not exist.)
4. Click **Save**.
5. Test the job by clicking the **job clock** icon on the Jenkins dashboard, as shown in Figure 3-16.




| All | + | | | | |
|---|---|--|-----------------------------------|----------------------------------|---|
| S | W | Name ↓ | Last Success | Last Failure | Last Duration |
|  |  | Cloud Trader Liberty Profile | 1 hr 44 min - #11 | 6 hr 51 min - #8 | 3.2 sec |
| | | | | |  |

Figure 3-16 Jenkins job for Rational Team Concert integration



Continuous testing

Continuous testing is a software development practice in which tests are run automatically by a continuous testing tool. This method reduces the time that is needed for manual testing.

In this chapter, the following topics are described:

- ▶ Rational Quality Manager for system testing
- ▶ Rational Test Workbench for test virtualization
- ▶ Test environment for Liberty configuration testing

4.1 Rational Quality Manager for system testing

IBM Rational Quality Manager is test management software that provides quality assurance (QA) teams with a method to manage and track the QA lifecycle.

Rational Quality Manager is a collaborative and customizable tool for teams to perform these tasks:

- ▶ Create and maintain a test plan
- ▶ Test, design, and run manual or automated tests
- ▶ Manage goals
- ▶ Schedule, track, and report metrics about the health of the test phase overall

Rational Quality Manager simplifies all aspects of test management and includes support for managing requirements and defects. We use Rational Quality Manager as a unit test solution in this book.

4.1.1 Structure of the Rational Quality Manager project

Use the following steps to create the structure of the Rational Quality Manager project:

1. Create a folder that is named `unitTest` in the root partition of the test server. In that folder, create the following subdirectories:
 - `adapter`
 - `build`
 - `code`
 - `jar`
2. Download the unit test Java archive (jar) file and the hamcrest framework from <http://junit.org>, and save them to the `jar` folder. In this example, we used the following JAR files:
 - `junit-4.12.jar`
 - `hamcrest-core-1.3.jar`
3. Create the Java application (Example 4-1), and save the `HelloRqm.java` file in the `/unitTests/code` directory.

Example 4-1 HelloRqm.java file

```
public class HelloRqm {  
  
    public String customcode() {  
  
        return("RQM Test Hello World");  
  
    }  
  
}
```

4. Create the test application (Example 4-2) and save the TestHelloRqm.java file in the /unitTests/code directory.

Example 4-2 TestHelloRqm.java file

```
import junit.framework.TestCase;

public class TestHelloRqm extends TestCase {

    public TestHelloRqm(

        String name) {

        super(name);

    }

    public void testReturn() {

        HelloRqm hello = new HelloRqm();

        assertEquals("RQM Test Hello World", hello.customcode());

    }

    public static void main(String[] args) {

        junit.textui.TestRunner.run( TestHelloRqm.class);

    }

}
```

5. Compile the project by issuing the following commands:

```
export MAIN_TEST_DIR="/unitTests/"
export DATASTORE_DIR="$MAIN_TEST_DIR/build"
export
CLASSPATH="$MAIN_TEST_DIR/jar/junit-4.12.jar:$MAIN_TEST_DIR/jar/hamcrest-core-1
.3.jar:$DATASTORE_DIR"
javac /unitTests/code/* -d /unitTests/build/
```

6. Execute the test case by issuing the following commands:

```
cd $DATASTORE_DIR
java org.junit.runner.JUnitCore TestHelloRqm
```

7. Create a `junit_test.sh` script with the content that is shown in Example 4-3, and save the script in the `/unitTests` directory.

Example 4-3 Content of `junit_test.sh`

```
export MAIN_TEST_DIR="/unitTests/"
export DATASTORE_DIR="$MAIN_TEST_DIR/build"

export
CLASSPATH="$MAIN_TEST_DIR/jar/junit-4.12.jar:$MAIN_TEST_DIR/jar/hamcrest-core-1
.3.jar:$DATASTORE_DIR"
cd $DATASTORE_DIR
java org.junit.runner.JUnitCore TestHelloRqm
```

4.1.2 Configure the Rational Quality Manager Adapter

Rational Quality Manager provides a command-line adapter, the Rational Quality Manager Adapter, to execute tests. Use the following steps to configure the adapter:

1. Locate the adapter from the Rational installation directory. The default path for the adapter is `/opt/IBM/JazzTeamServer/server/conf/qm/adapters/RQMCommandLineAdapter.zip`.
2. Extract the command-line adapter into the `/unitTest/adapter` directory.
3. Execute the following commands from the command-line prompt to connect to the Rational Quality Manager server:

```
cd /unitTests/adapter
./start.sh -repository https://lexbz181128.lex.dst.ibm.com:9443/qm -user
administrator -password smart321way -adapter RQMAdapter -adapterName
Moi-RQMJUnit -projectArea "Liberty Cloud Trader (Quality Management)"
-configFile Junitconfig.ini
```

After the successful creation of the adapter, the messages “The adapter is now connected” and “Created Commandline Adapter” display. A configuration file that is named `Junitconfig.ini` is populated with information about the Rational Quality Manager connection.

4. In Rational Quality Manager, select **Execution** → **Adapter Console**. Ensure that the adapter is connected, as shown in Figure 4-1.

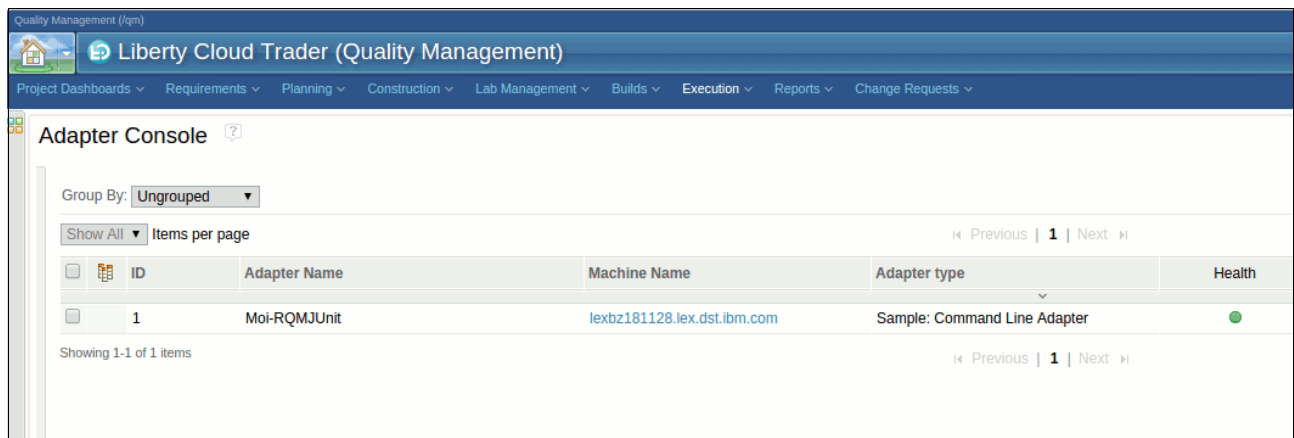


Figure 4-1 The adapter console

4.1.3 Create a unit test case

To create a unit test case in Rational Quality Manager, navigate to **Construction** → **Create a new Test Case**, as shown in Figure 4-2.

The screenshot displays the Rational Quality Manager (RQM) interface for creating a new test case. The top navigation bar shows the 'Construction' menu item selected. The main content area is titled 'Test Cases >' and contains a form for 'junit test case 01'. The form is in the 'Draft' state and is created using the 'Default Test Case Template'. The 'Summary' section is expanded, showing a description and categories. The 'Function' and 'Test Phase' are both set to 'Unassigned'. The 'Estimate' field is empty, and the 'Weight' is set to '100 Points'. The 'Originator' and 'Owner' are both 'Administrator', and the 'Priority' is 'Low'. The 'Description' field contains a link to enter a description. The left sidebar shows a list of sections, including 'Summary', 'Test Case Design', 'Formal Review', 'Development Items', 'Requirement Links', 'Risk Assessment', 'Pre-Condition', 'Post-Condition', 'Expected Results', 'Test Scripts', 'Test Case Execution Records', 'Attachments', and 'Execution Variables'. The bottom of the form has a 'Contains Unsaved Changes' warning.

Figure 4-2 Creating a test case

4.1.4 Create a unit test script

Use the following steps to create a unit test script in Rational Quality Manager:

1. Navigate to **Construction** → **Create a new Test Script**, as shown in Figure 4-3.
2. Type a name for the test case, and click **Save**.

Quality Management (lqm)

Liberty Cloud Trader (Quality Management)

Project Dashboards ▾ Requirements ▾ Planning ▾ Construction ▾* Lab Management ▾ Builds ▾ Execution ▾ Reports ▾ Change Requests ▾

Test Scripts >

Contains Unsaved Changes

2: junit test script 02

Sections

- Summary
- Formal Review
- Command Line Script Details
- Execution Variables
- Associated E-Signatures
- Show All Sections

Snapshots

History

State: Draft

Action: Change State ▾

Originator: Administrator

Owner: Unassigned ▾

Type: Command Line

Preferred machine: lexbz181128.lex.dst.ibm.com

Description: < Click here to enter a description >

Command Line Script Details

Command Line script

This will create a Test Script that references external test resources. Select the type of test resource to use:

☐ Use test resources from a shared location

Choose test resources from a shared location.
At execution time, the resources will be copied to the test machine.

☒ Use test resources that are local to a test machine

Choose test resources that are on a test machine already.
At execution time, the resources at the location you choose will be used.

Select Adapter

Command

Arguments

Contains Unsaved Changes

Figure 4-3 Creating a unit test script

3. Type a name for the test script. Select **Use test resources that are local to a test machine**.
4. Click **Select Adapter**, and select the adapter **Moi-RQMJUnit** in the next window. Click **Next**.

5. For the Project Path, type /unitTests, and click **Go**.
6. Select the **junit_test.sh** script, as shown in Figure 4-4.
7. Click **Finish** and save the test script.

Quality Management (/qm)

Liberty Cloud Trader (Quality Management)

Project Dashboards ▾ Requirements ▾ Planning ▾ Construction ▾* Lab Management ▾ Builds ▾ Execution ▾ Reports ▾ Change Requests ▾

Test Scripts >

Contains Unsaved Changes

2: junit test script 02

Sections: Summary, Formal Review, **Command Line Script Details**, Execution Variables, Associated E-Signatures

State: Draft Action: Change State ▾

Originator: Administrator Owner: Unassigned ▾

Type: Command Line

Preferred machine: lexbz181128.lex.dst.ibm.com

Description: < Click here to enter a description >

Import Test Script: Use local test resources

Step 2: Select Project path and Test Scripts

Project Path: /unitTests

☐ When the selected machine is available, use it as the preferred machine to run the test scripts

Table:

| Name ^ | Location | Type |
|---------------|------------|--------------|
| junit_test.sh | /unitTests | Command Line |

Figure 4-4 Selecting the `junit_test.sh` script

4.1.5 Associate the test case and test script

Use the following steps to associate the test case and test script:

1. Open the junit test case that was created in 4.1.3, “Create a unit test case” on page 63.
2. Select **Test scripts** from the menu on the left.
3. Click the plus sign (+) icon to add the test scripts to the list, as shown in Figure 4-5.

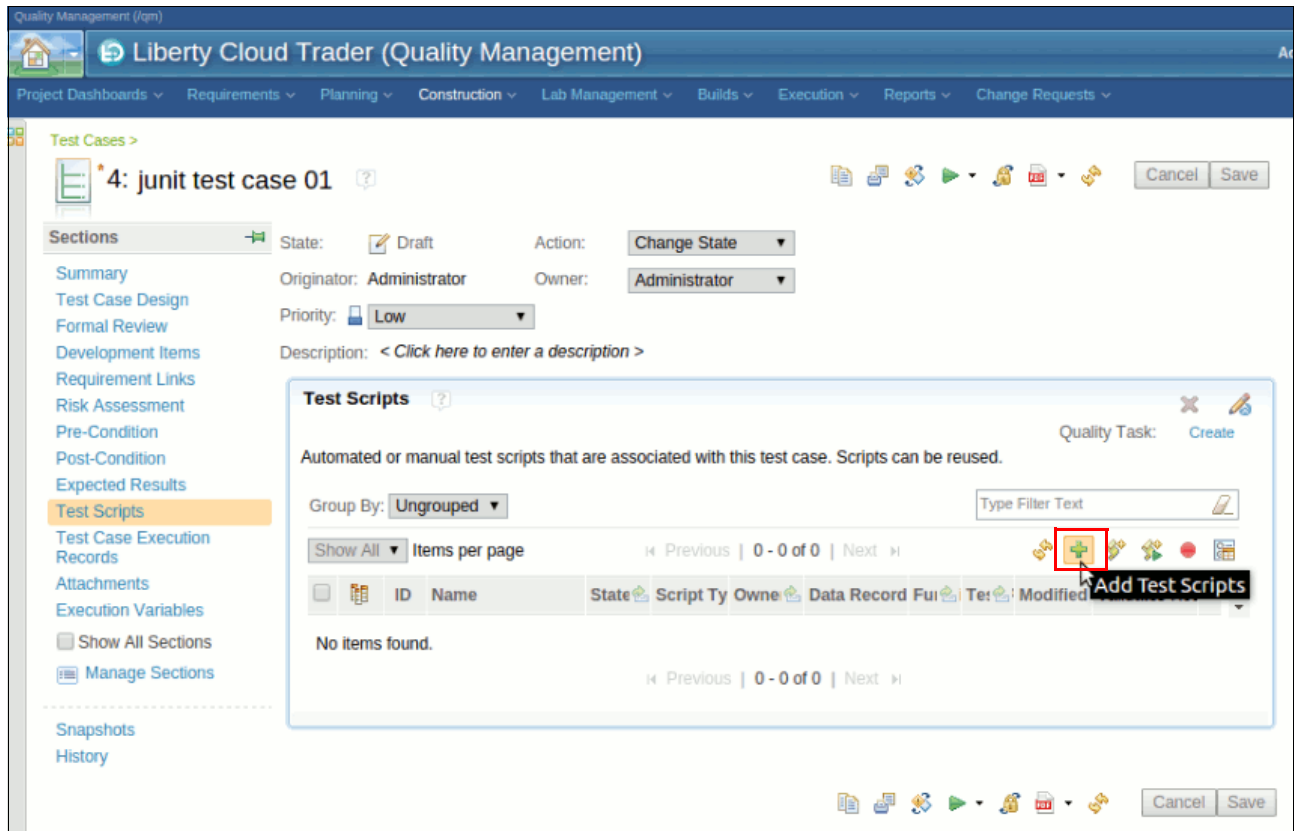


Figure 4-5 Associating the test case and test script

4. Select **junit test script 02**. The window in Figure 4-6 opens.
5. Click **Add and Close**.
6. Click **Save** to save the test case.

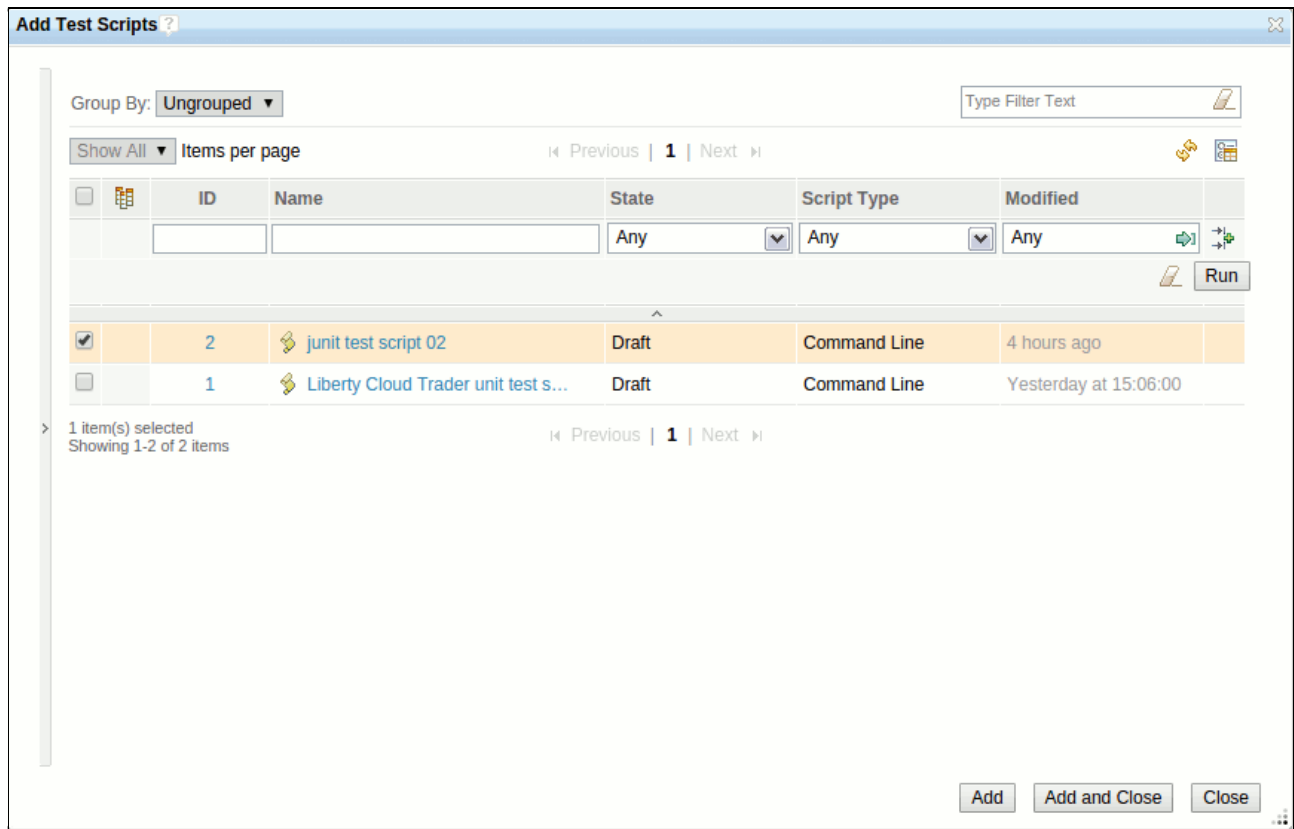


Figure 4-6 Adding the test script

4.1.6 Create a test case execution record

Use the following steps to create a test case execution record to execute the test case:

1. In the junit test case, click **Test Case Execution Records** in the left pane (Figure 4-5 on page 66).
2. Click the **Generate New Test Case Execution Records** icon, as shown in Figure 4-7.

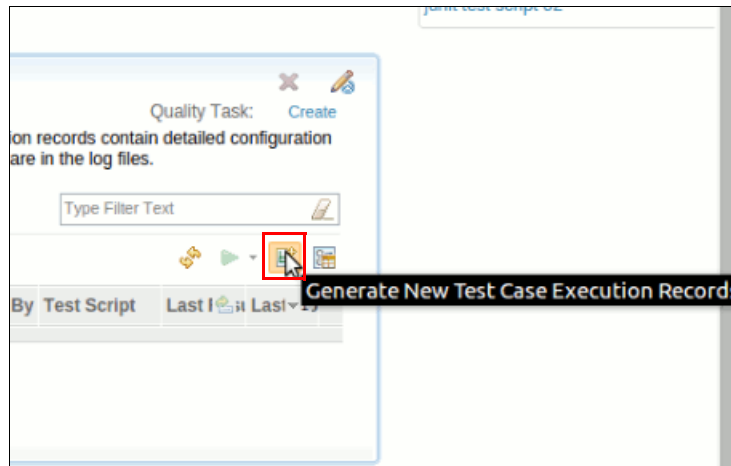


Figure 4-7 Generating a new test case

3. The window in Figure 4-8 opens.

Figure 4-8 Generating a test case execution record

4. Click the **Generate Test Environments** tab.
5. Click the **Sample Command Line Adapter** check box.
6. Click **Next**, and the window that is shown in Figure 4-9 opens.
7. Select the **Sample Command Line Adapter**.

| Name | Browsers | Test Adapter | Database | Application Server | Management Agent | Operating System | CPU |
|--------------------|----------------|--------------------|----------------|--------------------|------------------|------------------|----------------|
| Sample: Command... | Not Configured | Sample: Command... | Not Configured | Not Configured | Not Configured | Not Configured | Not Configured |

Figure 4-9 Setting the command-line adapter

8. Click **Next**, and the window that is shown in Figure 4-10 opens.
9. Select **junit test case 01**, and click **Finish**.



Figure 4-10 Selecting the junit test case 01

The setup for the test case in Rational Quality Manager is now complete.

4.1.7 Execute test cases

Use the following steps to execute test cases in Rational Quality Manager:

1. Click **Construction** → **Browse** → **junit test case 01**.
2. Click the **Play** icon. Click **Run** to execute the test case, as shown in Figure 4-11.

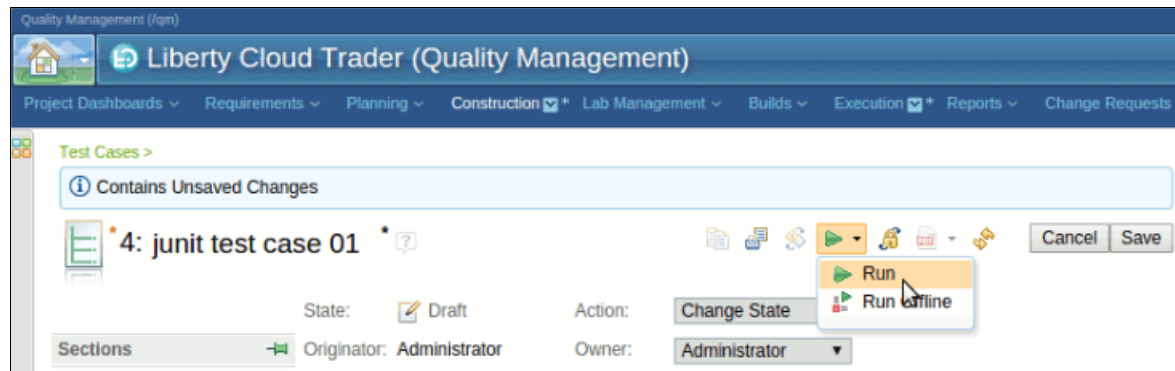
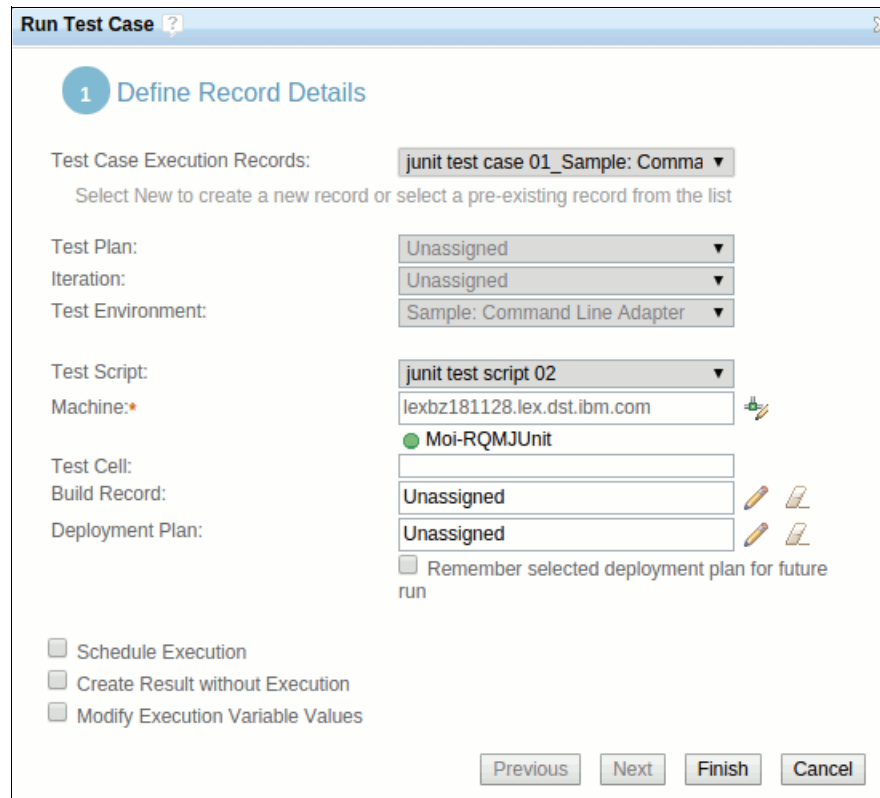


Figure 4-11 Running the junit test case 01

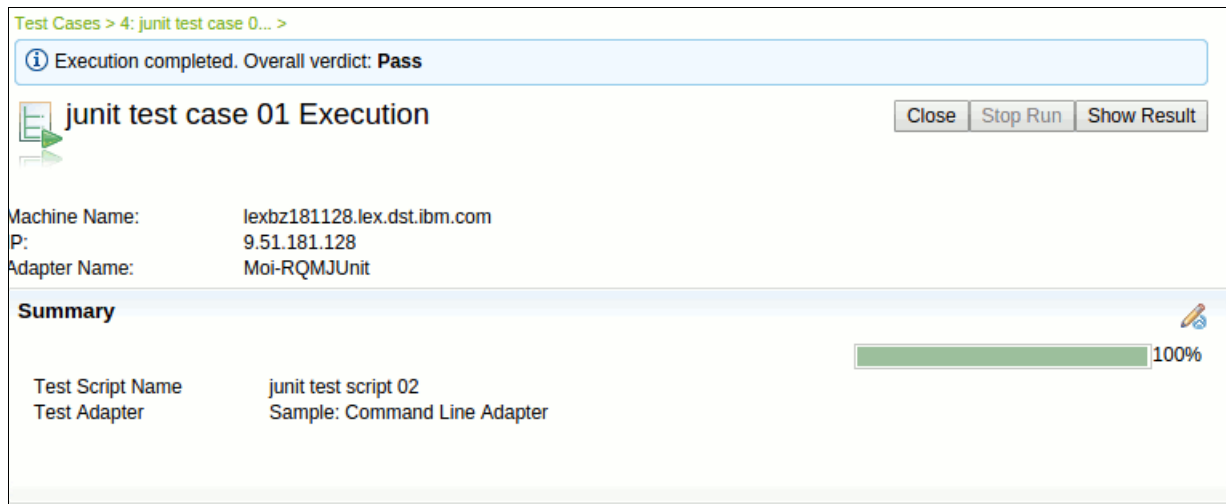
3. The window that is shown in Figure 4-12 opens. Select **junit test case 01**, and click **Finish** to run the test case.



The 'Run Test Case' dialog box is shown with the title bar 'Run Test Case'. It contains a section '1 Define Record Details'. Under 'Test Case Execution Records', there is a dropdown menu showing 'junit test case 01_Sample: Comma' and a note 'Select New to create a new record or select a pre-existing record from the list'. Below this are fields for 'Test Plan:' (Unassigned), 'Iteration:' (Unassigned), and 'Test Environment:' (Sample: Command Line Adapter). The 'Test Script:' field shows 'junit test script 02'. The 'Machine:' field shows 'lexbz181128.lex.dst.ibm.com' with a green dot icon. The 'Test Cell:' field is empty. The 'Build Record:' field shows 'Unassigned' with a pencil icon. The 'Deployment Plan:' field shows 'Unassigned' with a pencil icon. There is a checkbox 'Remember selected deployment plan for future run' which is unchecked. At the bottom, there are three checkboxes: 'Schedule Execution', 'Create Result without Execution', and 'Modify Execution Variable Values', all of which are unchecked. At the bottom right, there are four buttons: 'Previous', 'Next', 'Finish', and 'Cancel'.

Figure 4-12 Running the junit test case

4. The window that is shown in Figure 4-13 opens. Click **Show Result** to see more details about the execution of the junit test case.



The 'JUnit test case 01 Execution' window is shown. It has a title bar 'Test Cases > 4: junit test case 0... >'. Below the title bar, there is a blue bar with an information icon and the text 'Execution completed. Overall verdict: Pass'. Below this, there is a section 'junit test case 01 Execution' with a green play button icon. To the right of this section are three buttons: 'Close', 'Stop Run', and 'Show Result'. Below this, there is a table with the following data:

| | |
|---------------|-----------------------------|
| Machine Name: | lexbz181128.lex.dst.ibm.com |
| P: | 9.51.181.128 |
| Adapter Name: | Moi-RQMJUnit |

Below the table, there is a section 'Summary' with a green progress bar and the text '100%'. To the right of the progress bar is a pencil icon. Below the progress bar, there is a table with the following data:

| | |
|------------------|------------------------------|
| Test Script Name | junit test script 02 |
| Test Adapter | Sample: Command Line Adapter |

Figure 4-13 Showing the results of the junit test case

5. The window that is shown in Figure 4-14 displays additional information about the execution of the `junit` test case.

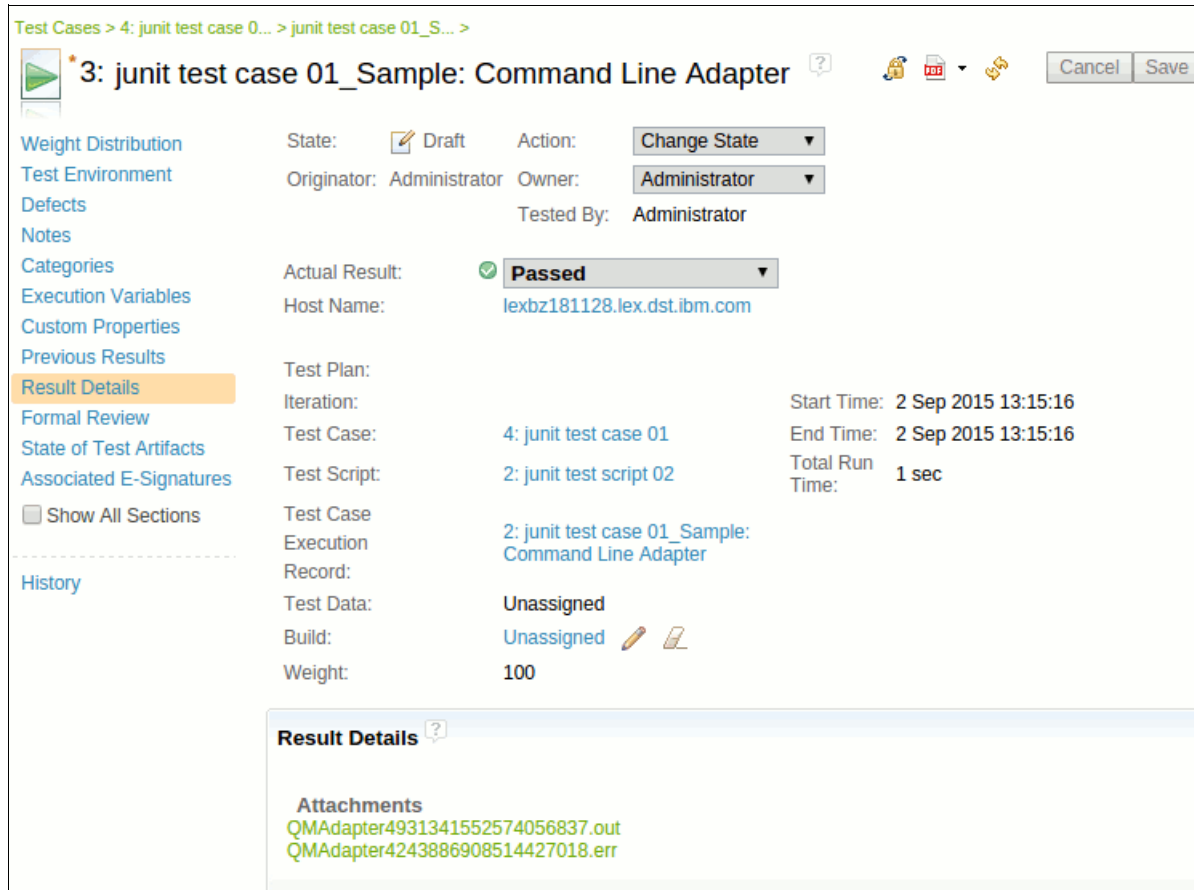


Figure 4-14 Showing the additional results of the `junit` test case

By using Rational Quality Manager, we ran the `junit` test case successfully.

4.2 Rational Test Workbench for test virtualization

IBM Rational Test Workbench provides a comprehensive test automation solution for applications. Testing covers model-based, functional, and performance testing.

To test an application end-to-end, you need to understand the databases and services that are associated with the system. The Rational Test Workbench server helps to simulate the back-end services that you can use during integration testing.

Rational Integration Tester (RIT) is the integration testing and application virtualization tool in the Rational Test Workbench. This section shows how to use RIT to test an application database without any modifications in the code by using virtual services that are called *stubs*.

For more information about test virtualization, see the *IBM Rational Integration Tester Testing Training Guide* at the following website:

http://www.slideshare.net/Darrel_Rader/rit-850-integration-testing-training-students-guide

4.2.1 Create and run a database stub

In the following section, you use RIT to intercept and record database access requests and to create a virtual database that is based on those requests. This direct recording method is called *Live Mode*.

Recording and creating the database stub

Use the following steps to record and create a database stub to behave as an actual “live” DB2 database:

1. Open RIT. In the Architecture School perspective, go to the **Physical View**, and expand the tree structure to show the details for the localhost.
2. Double-click the **IBM DB2 (Universal)** entry for the Liberty Cloud Trader database to display its configuration details, as shown in Figure 4-15.

Follow these steps:

- a. In the database configuration window, select the **Stub Settings** tab. Ensure that the **Use integrated database** check box is selected. For this data source, the system uses the integrated database for recording as the basis for stubbing rather than a vendor-specific database.
- b. Click **OK** to save the changes, and close the window.

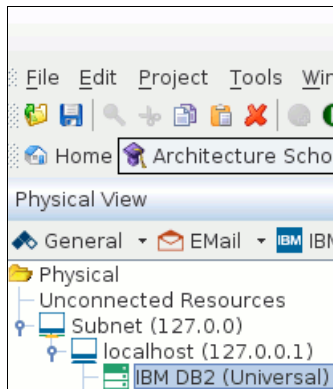


Figure 4-15 IBM DB2 entry

3. Switch to the **Recording Studio** perspective. Clear any monitor configurations and events from previous recordings.
4. From the Monitor Configuration toolbar, click **Select operations or resources to monitor**. Select **XA DataSource**, and click **OK**.

5. In the Events view, click **Recording** to start recording. Follow these steps:
 - a. When you are prompted, select **Yes, record the SQL and create a virtual database**.
 - b. Select **Create a database stub but don't start it**. When the recording stops, you can specify what to do with the stub.
 - c. For the Stub name, enter VirtualDatabase, as shown in Figure 4-16.
 - d. Click **Start Recording** to record the SQL queries.



Figure 4-16 Database virtualization

6. Return to your web browser. Open the Liberty Cloud Trader Application.
7. Click **Trading & Portfolio**, and log in to the application with user ID uid:0.
8. Return to RIT and pause the recording of the stub. When you are prompted to open the resource in the Test Factory perspective, click **OK**.
9. The Test Factory perspective displays information about the database stub. The Queries tab shows the SQL queries that were recorded as a result of your browser actions. The Tables tab shows the tables that were queried and created.

4.2.2 Edit the data in the database stub

Before you use the virtual database in operation, you edit the data that it contains to illustrate that data is sourced from the database stub and not from the live database:

1. From the toolbar, click **Use a spreadsheet to edit simulation database tables**, as shown in Figure 4-17.

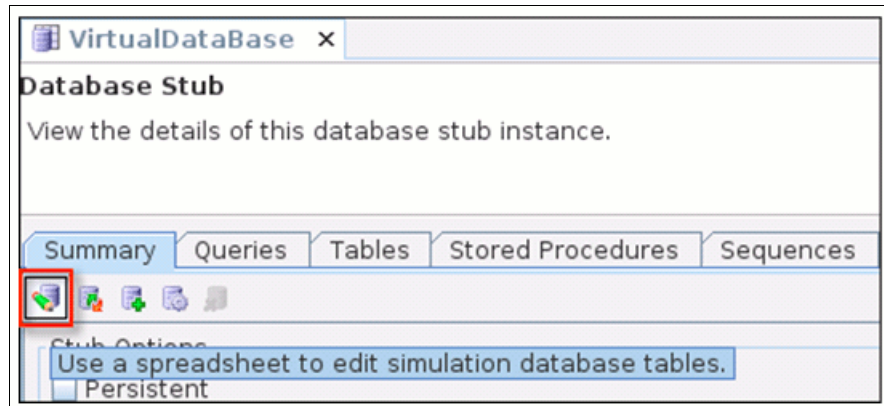


Figure 4-17 Use a spreadsheet to edit the simulation database tables

2. The Edit Database stub wizard opens. On the first page, select **Edit the contents of specific tables**, and then click **Next**.
3. In the pop-up window, select the check box for **SELECT * FROM ACCOUNTPROFILEEJB** to edit the data that is associated with that query and table. Click **Next**.
4. In a few moments, a spreadsheet opens with data. Locate the row with user ID uid:0, and change the following data:
 - Address: Tenerife St number 2, Spain
 - Password: teide
 - Full Name: uid:0
5. Click **Save** to save the spreadsheet, and accept any defaults. Close the spreadsheet window.
6. Return to RIT. The Edit Database Stub window is still active; it is waiting for confirmation that you completed the changes to the data. Click **Finish** to complete the editing.
7. At the top of the Summary tab, select the **Persistent** check box. This setting retains any changes to the simulated database because it is run for future executions of the database stub. Otherwise, any changes during execution are discarded when the stub is stopped.
8. Click **Save** to save the database stub.
9. Locate your stub in the tree. Select **Logical** → **IBM WebSphere Application Server** → **XA Data Source** → **VirtualDatabase**. Right-click and select **Run** to run your stub.
10. Wait until the stub is Ready, then return to your web browser. Open Liberty Cloud Trader.
11. Log in to Liberty Cloud Trader with the user name uid:0 and password teide.

You successfully observed, recorded, and created a database stub that is based on the SQL queries and data that are returned from the Liberty Cloud Trader application. You then ran the database stub and observed that the stub can simulate the database and store data persistently, also.

4.3 Test environment for Liberty configuration testing

The configurations of the development environment and test environment are similar. The Liberty profile configuration file is described.

The major Liberty configuration file is the `server.xml` file. It describes the application server configuration. You can configure one or more attributes of each element.

The configuration elements of the `server.xml` file are described in this link:

https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/autodita/rwlp_metatype_4ic.html



Deployment

This chapter provides a practical example for deploying applications, configurations, and middleware products. By using IBM UrbanCode Deploy and Jenkins as deployment automation tools, we demonstrate how the WAS Liberty profile application server enables continuous delivery by reducing the required installation and deployment time.

In this chapter, the following topics are described:

- ▶ How to use UrbanCode Deploy for application deployment
- ▶ How to use Jenkins to deploy applications to IBM Bluemix

5.1 How to use UrbanCode Deploy for application deployment

Part of the objective of the continuous delivery process is to reduce the time that is between code delivery and application installation. In UrbanCode Deploy (UCD), *components* are the representation of the system artifacts to deploy (files, directory structures, and binaries) and their manipulation processes. UrbanCode Deploy provides all of the necessary tools to fetch the different versions of components, modify each component (if necessary), and deploy them over several environments. The following software is used in this section:

- ▶ UCD Plug-in 1.2.4 for Jenkins:
<https://developer.ibm.com/urbancode/plugin/jenkins>
- ▶ WAS Liberty plug-in 3.641636 for UCD:
<https://developer.ibm.com/urbancode/plugin/websphere-liberty-ibmucd/>
- ▶ Jenkins Server v1.609.2:
<https://jenkins-ci.org/>
- ▶ IBM UrbanCode Deploy v6.1.1:
<https://developer.ibm.com/urbancode/products/urbancode-deploy/>

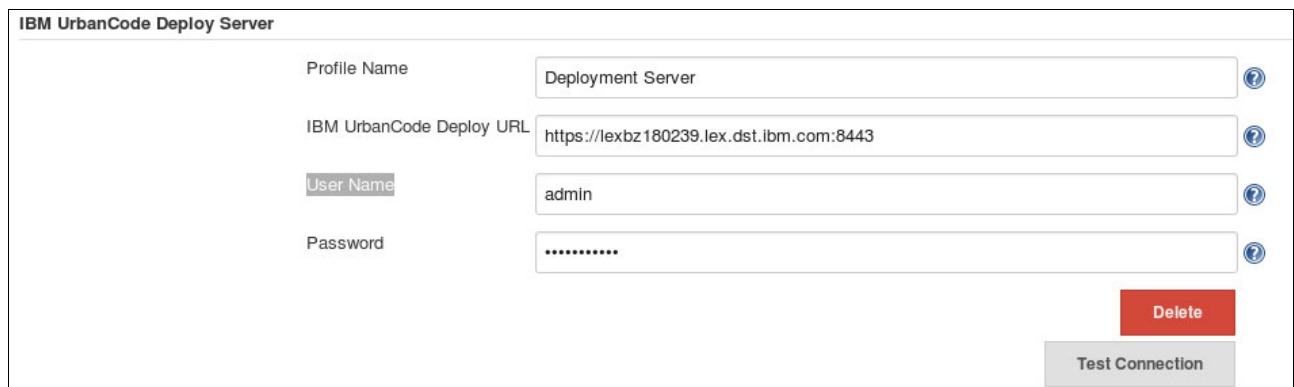
5.1.1 Publishing component artifacts from Jenkins to UrbanCode Deploy

Chapter 3, “Code development, source code management, and build” on page 45 describes how to build a web application by using Rational Team Concert and Jenkins. In this chapter, we explain how to publish artifacts that are built by Jenkins to UCD.

Installing and configuring the Jenkins UCD plug-in

The Jenkins UCD plug-in adds another UCD artifact to publish that is called Post-build Action to the Jenkins catalog. This action allows us to publish artifacts that are generated during the build phase to UCD as component artifacts. Follow these steps:

1. At the Jenkins management console, select **Manage Jenkins** → **Configure System**.
2. Scroll down to IBM UrbanCode Deploy Server, and select **Add**.
3. Enter the Profile Name and the IBM UrbanCode Deploy URL server connection parameters (the UrbanCode Deploy URL, User Name, and Password), and click **Test Connection**. If the test connection is successful, click **Save**. Figure 5-1 shows a server connection example.



The screenshot shows the 'IBM UrbanCode Deploy Server' configuration form. It contains four input fields: 'Profile Name' with the value 'Deployment Server', 'IBM UrbanCode Deploy URL' with the value 'https://lexbz180239.lex.dst.ibm.com:8443', 'User Name' with the value 'admin', and 'Password' with masked characters. Each field has a help icon to its right. At the bottom right, there are two buttons: a red 'Delete' button and a grey 'Test Connection' button.

Figure 5-1 UCD plug-in for Jenkins server connection example

Configuring a Jenkins job to publish artifacts to UrbanCode Deploy

The plug-in and the UCD server connection are in place, so we must configure a job to publish artifacts to UCD. This procedure applies to any build job. Follow these steps:

1. From the Jenkins management console, click **Manage Jenkins** → **Manage Plug-ins**.
2. Select the **Advanced** tab, and click **Browse** in the Upload Plug-in section. Select the UCD plug-in file (**ibm-ucdeploy-publisher-XXX.hpi**), and click **Upload**.
3. Go to the configuration page of the Jenkins job that was created in Chapter 3, “Code development, source code management, and build” on page 45. Scroll down to the post-build actions section and create a new post-build action of type **Publish artifacts to IBM UrbanCode Deploy**.
4. Select a server connection from the IBM UrbanCode Deploy Server list box.
5. Enter the name of the component to publish. (Later, we define the same component name at the UCD console.)
6. Enter the environment variable **\${WORKSPACE}** in the Base Artifact Directory box.
7. Enter the environment variable **\${BUILD_NUMBER}** in the Version box. This variable passes the Rational Team Concert build number to the UCD component version.
8. In the Include text area, enter the relative path of the artifacts to publish. Use ****/*** to include all files in the working directory. You can use Exclude to define the pattern of the components to exclude.
9. If you want to execute a deployment process after you publish a new component version, check the **Deploy** check box. This function can be useful if you want to deploy a new version to a development server, for example, to execute a unit test over this new component version. In this case, we left it unchecked.
10. Click **Save**.

The Jenkins job compiles and publishes the Rational Team Concert components and artifacts to UCD. You can test the job by requesting a new build on Rational Team Concert or directly from the Jenkins console. Both actions publish a new component version on UCD.

Figure 5-2 shows an example of the post-build actions configuration.

Post-build Actions

Publish artifacts to IBM UrbanCode Deploy

IBM UrbanCode Deploy Server: Deployment Server

Use Another User: ☐

User Name:

User Password:

Component: CloudTrader

Base Artifact Directory: \${WORKSPACE}

Directory Offset:

Version: \${BUILD_NUMBER}

Version Description:

Version Properties:

Include: CloudTrader/CloudTrader.war

Exclude:

Skip publishing (e.g. temporarily): ☐

Deploy: ☐

Figure 5-2 Example of UCD publishing post-build actions configuration

5.1.2 Configuring UrbanCode Deploy for system deployment

UrbanCode Deploy provides a framework to define the applications, environments, components, and processes that are necessary for systems deployment. In this section, we define an application that contains three components (application server, application server configuration, and enterprise application). These components are installed in two environments (test and production).

Note: The documentation about the naming that is used in this section is available at the following website:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.doc/topics/glossary.html?lang=e

Installing the Liberty profile plug-in

The Liberty profile plug-in provides a set of tools for managing the Liberty profile server and application from UrbanCode Deploy. Follow these steps:

1. From the UCD administration dashboard, click **Settings** → **Automation Plug-ins**, and click **Load Plug-in**.
2. Browse the Liberty profile plug-in file (see the Note that follows this step) and click **Submit**.

If the plug-in installation is successful, the WAS Liberty plug-in is included in the plug-in list.

Note: You can download the WAS Liberty plug-in from the following website:

<https://public.dhe.ibm.com/software/products/UrbanCode/plugins/ibmucd/WebSphereLiberty/WebSphere-Liberty-3.641636.zip>

Installing the UrbanCode Deploy agent remotely

The UCD agent is responsible for executing the deployment process and commands to the target servers. For the Liberty Cloud Trader use case, we define two target servers with one agent each. The solution architecture can vary depending on the complexity and the required availability level.

The following steps are for the installation of a single agent. Repeat these steps for each target server. Follow these steps:

1. From the UCD administration dashboard, click **Resources** → **Agents**, and click **Install New Agent**.
2. Choose Secure Shell (SSH) for UNIX systems and WinRS for Microsoft Windows systems. For our example, we choose SSH.
3. In the Target Hosts text box, enter the host name or IP address of the target server.
4. In the SSH Port field, leave the default value (22), except that you need to use a different port for SSH connections.
5. The User name field must contain the name of the user who will install and execute commands on the target servers.
6. You can choose to **Use Public Key Authentication** or **Password** authentication. In our example, we choose password authentication.
7. Enter an Agent Name, which must identify the target server. We suggest that you include the environment name in the agent name. For example, use TestAgent1 to identify to which environment the agent belongs.
8. Click **Save**, and the agent installation process starts automatically. After a few seconds, the agent status displays.

Note: The UCD agent is based on Java. The `$JAVA_HOME` environment variable must be defined on the user profile of the user who installs and executes the UCD agent.

For troubleshooting instructions, see this website:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.doc/topics/agent_installremote.html?cp=SS4GSP_6.1.1%2F3-3-0&lang=en

Defining components

We define the UrbanCode Deploy *components* to represent our artifacts. We define three components for this case:

- ▶ Liberty App Server: Contains the JAR file of the Liberty profile server core
- ▶ Cloud Trader Server Config: Contains the server.xml file and the db2 driver libraries
- ▶ Cloud Trader: Contains the application .war file

Note: Any configuration field can contain references to element properties, such as `${p:server_name}`.

For more information about the use of property references, see this website:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.doc/topics/ud_properties_using.html

Creating the Liberty Application Server component and installing the Liberty profile server

This section describes how to create the Liberty Application Server component and then install the Liberty profile server on the Liberty Application Server component. You can also create processes to stop, start, and uninstall the Liberty Application Server by following the same logic.

Follow these steps to create the Liberty Application Server component:

1. From the UCD administration dashboard, click **Components** → **Create Component**.
2. In the Name field, enter the name of the component, in this case, Liberty App Server.
3. Accept all of the default settings, and click **Save**.

We create the Install Liberty profile server process design on the Liberty Application Server component, which executes the following actions:

- ▶ Stops the previously running Liberty profile
- ▶ Deletes the previous Liberty profile installation
- ▶ Downloads the Liberty Install artifact to a component default working directory on the target system
- ▶ Installs the Liberty profile
- ▶ Executes the shell script to harden or secure the app server
- ▶ Creates the Liberty profile server

Follow these instructions to create the process steps:

1. Create the Liberty profile server process:
 - a. In the Components Configuration window, select the **Processes** tab and click **Create Process**.
 - b. In the Process Creation dialog box, enter the process name, in this case, Install Liberty Profile Server, and click **Save**.
 - c. Click the new process element to display the process flow editor.

Figure 5-3 shows the process design flow for installing the Liberty profile server.

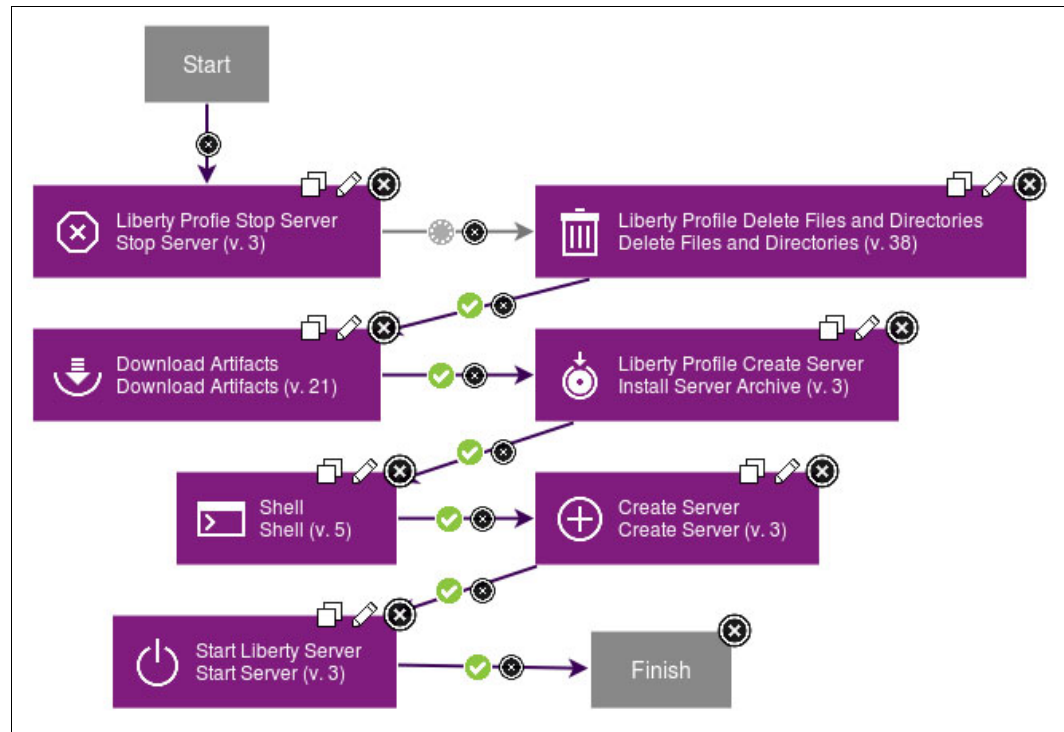


Figure 5-3 Install Liberty Profile Server process design

2. Drag a **Stop Server** step, configure it in the following manner, and click **Save**:
 - Server name: `${p:liberty_serverName}`
 - WAS Liberty installation directory: `${p:liberty_directory}`
3. Drag a **Delete Files and Directories** step, configure it in the following manner, and click **Save**:
 - Base directory: `.`
 - Include: `*/**`
 - Working directory: `${p:liberty_directory}`
4. Drag a **Download Artifact** step, accept the default setting, and click **Save**.
5. Drag an **Install Server Archive** step, configure it in the following manner, and click **Save**:
 - Source file: `Cloud Trader Liberty App Server/${p:liberty_jar_name}`
 - WAS Liberty installation directory: `${p:liberty_install_directory}`
6. Drag a **Shell** step, configure it in the following manner, and click **Save**:
 - Directory offset: `.`
 - Shell script: `chmod -R 770 bin`
 - Working directory: `${p:liberty_directory}`
7. Drag a **Create Server** step, configure it in the following manner, and click **Save**:
 - Server name: `${p:liberty_serverName}`
 - WAS Liberty installation directory: `${p:liberty_directory}`
8. Drag a **Start Server** step, configure it in the following manner, and click **Save**:
 - Server name: `${p:liberty_serverName}`
 - WAS Liberty installation directory: `${p:liberty_directory}`

9. Click the **Save** icon in the left toolbar to save the flow.

Creating the Cloud Trader Server Config component

To create the Cloud Trader Server Config component, follow these steps:

1. From the UCD administration dashboard, click **Components** → **Create Component**.
2. Enter the name of the component, in this case, Cloud Trader Server Config, in the name field.
3. Accept all of the default settings, and click **Save**.

We create the Install configuration process design for the Cloud Trader Server Config component, which executes the following actions:

- ▶ Downloads the Cloud Trader Server Config artifact to a component default working directory on the target system.
- ▶ Modifies the server.xml files by replacing tokens for environment properties.
- ▶ Copies the server.xml file.
- ▶ Copies the required library .jar files (database drivers and framework libraries) into the <server_dir>/lib directory.

Follow these instructions to create the process steps:

1. In the components configuration window, select the **Processes** tab and click **Create Process**.
2. In the process creation dialog box, enter the process name, in this case, Install configuration, and click **Save**.
3. Click in the new process element, and a process flow editor displays.

Figure 5-4 shows the flow of the Install configuration process design.

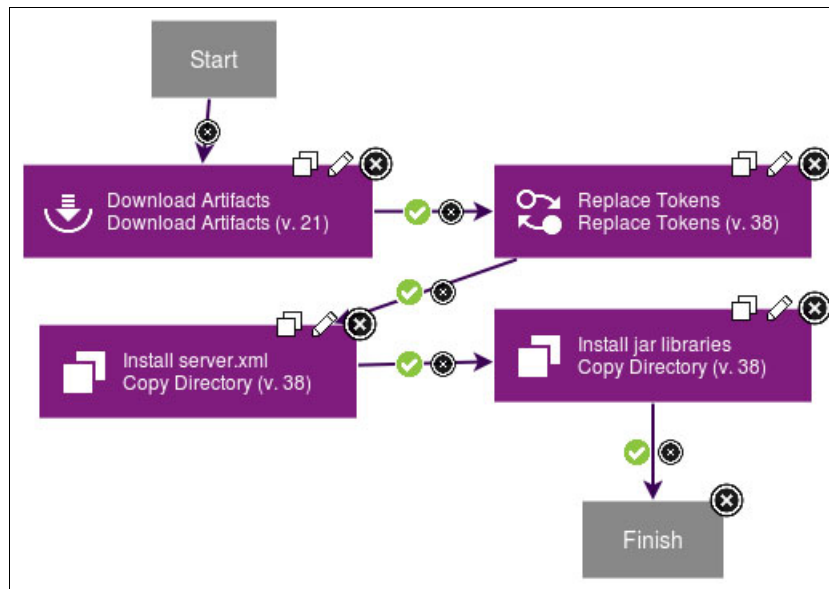


Figure 5-4 Install configuration process design

4. Drag a **Download Artifact** step, accept the default settings on all fields, and click **Save**.

5. Drag a **Replace Tokens** step, configure it in the following manner, and click **Save**:
 - Include files: `**/*.xml`
 - Start token delimiter: `@`
 - End token delimiter: `@`
 - Property file name: `replace_tokens.properties`
 - Property list: `${p:environment/allProperties}`
6. Drag a **Copy Directory** step, configure it in the following manner, and click **Save**:
 - Source directory: `./Liberty Profile Configuration/`
 - Destination directories: `${p:liberty_directory}/usr/servers/${p:liberty_serverName}/`
 - Include files: `*.xml`
7. Drag a **Copy Directory** step, configure it in the following manner, and click **Save**:
 - Source directory: `./Liberty Profile Configuration/`
 - Destination directories: `${p:liberty_directory}/usr/servers/${p:liberty_serverName}/lib/`
 - Include files: `*.jar`
8. Click the **Save** icon in the left toolbar to save the flow.

Creating the CloudTrader component

Follow these steps to create the CloudTrader component:

1. From the UCD administration dashboard, click **Components** → **Create Component**.
2. Enter the name of the component, in this case, `CloudTrader`, in the name field.
3. Accept all of the default settings, and click **Save**.

We create the Install Cloud Trader Application process design for the CloudTrader component, which executes the following actions:

- ▶ Downloads the CloudTrader application artifact to a component default working directory on the target system.
- ▶ Installs or updates the application artifact by using *Dropins*.

Follow these instructions to create the process steps:

1. In the components configuration window, select the **Processes** tab, and click **Create Process**.
2. In the process creation dialog box, enter the process name, in this case, `Install Cloud Trader Application`, and click **Save**.

3. Click the new process element, and a process flow editor displays.

Figure 5-5 shows the flow of the Install Cloud Trader Application process design.

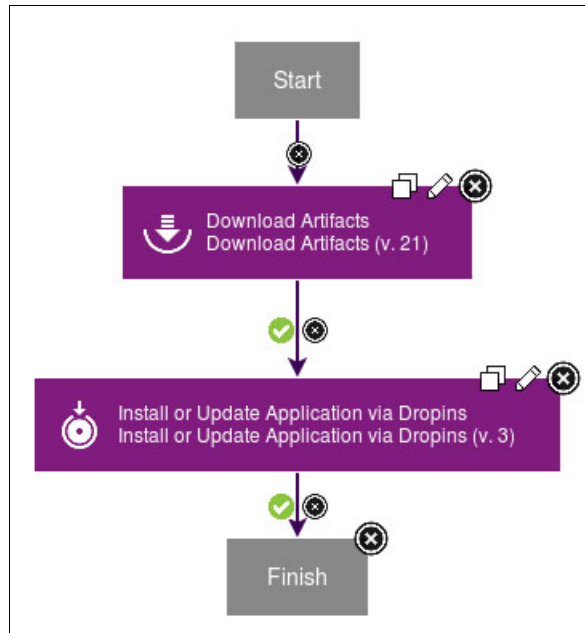


Figure 5-5 Install Cloud Trader Application process design

4. Drag a **Download artifact** step, accept the default setting, and click **Save**.
5. Drag an **Install or Update Application via Dropins** step, configure it in the following manner, and click **Save**:
 - Source file: CloudTrader/CloudTrader.war
 - Server name: `${p:liberty_serverName}`
 - WAS Liberty installation directory: `${p:liberty_directory}`
6. Click the **diskette** icon on the left toolbar to save the flow.

Defining resources

UCD *resources* are the deployment targets, such as a server. All systems with installed UCD agents are displayed in the **Resources** → **Agents** panel of the administration dashboard. Follow these steps:

1. On the Resources tab, define two top-level groups.
2. Include an agent in each group by using the Actions menu to make the agents available for application deployments.

3. Add the components that we defined in “Creating the CloudTrader component” on page 85 to each agent. Select the **Add component** option of the Actions menu.

Figure 5-6 shows the resource groups layout.

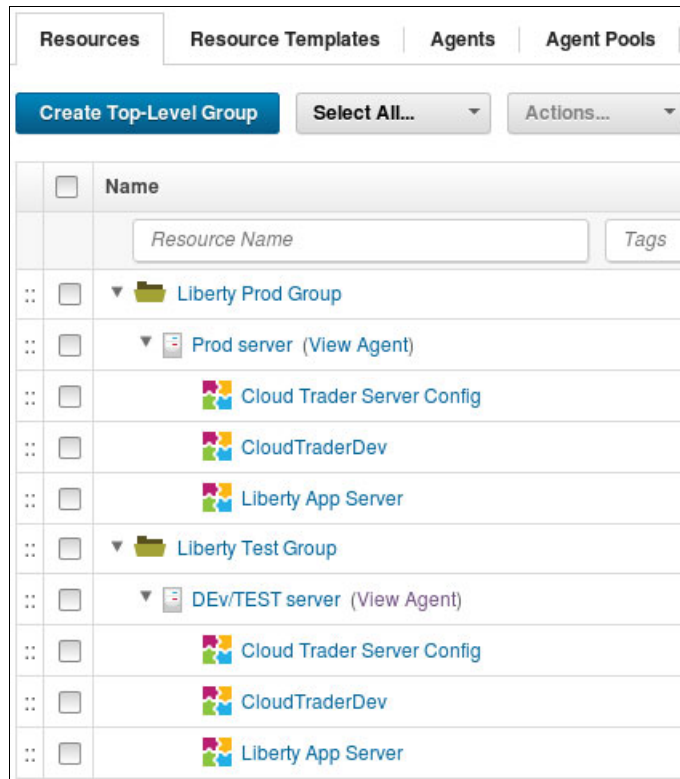


Figure 5-6 Resource groups layout

Defining an application

A *UCD application* is the collection of the necessary components for automated deployment. *Artifacts* are the necessary application files for installation, and *components* are the artifacts for installation and the deployment process.

Create the LibertyCloudTrader application on the UCD dashboard:

1. Click **Create New Application**, and enter an application name. In our example, we use LibertyCloudTrader.
2. On the Configuration tab, and create the following application properties. These properties are referred to during the component and application installation:
 - liberty_directory = /opt/IBM/WebSphere/wlp/
 - liberty_install_directory = /opt/IBM/WebSphere/
 - liberty_jar_name = wlr-nd-8.5.5.jar
 - liberty_serverName = CloudTrader

Figure 5-7 shows an example of the application properties definition.

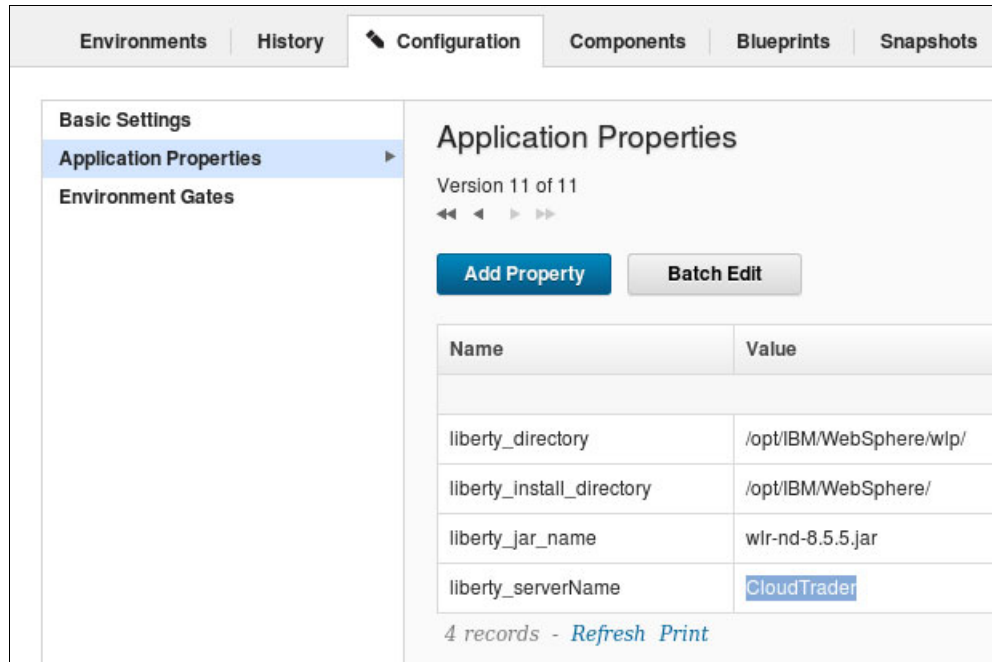
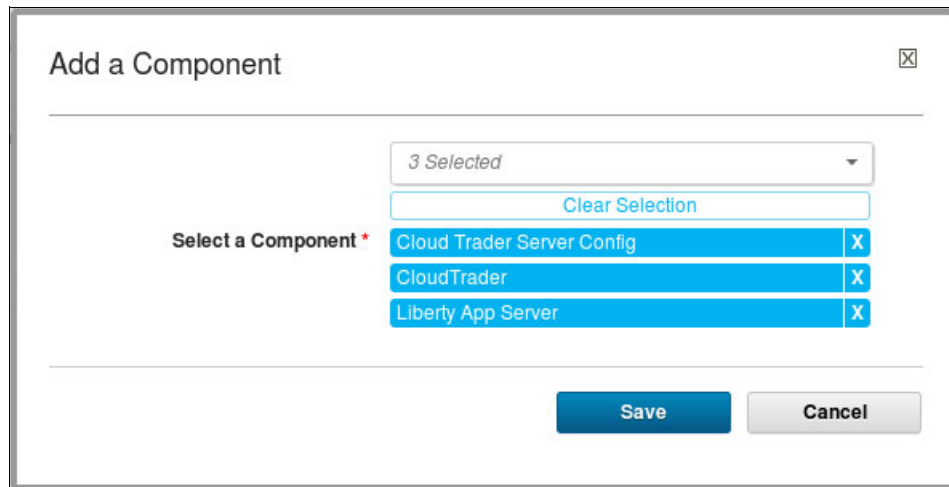


Figure 5-7 Application properties definition example

3. Associate the components with the LibertyCloudTrader application. In the application, click **Add a Component** and select the component name. Figure 5-8 shows the Add a Component dialog box.

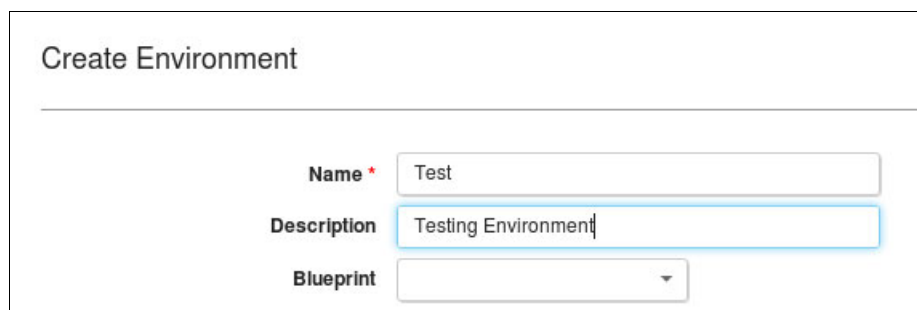


The 'Add a Component' dialog box features a title bar with a close button. Below the title bar is a horizontal line. Underneath, there is a dropdown menu showing '3 Selected' and a 'Clear Selection' button. To the left of the component list is the label 'Select a Component *'. The component list contains three items: 'Cloud Trader Server Config', 'CloudTrader', and 'Liberty App Server', each with a blue highlight and a red 'X' in a box to its right. At the bottom right are 'Save' and 'Cancel' buttons.

| Component Name | Action |
|----------------------------|--------|
| Cloud Trader Server Config | X |
| CloudTrader | X |
| Liberty App Server | X |

Figure 5-8 Add a Component dialog box

4. With the components in place, create two application environments (test and production (prod)) for LibertyCloudTrader. Figure 5-9 shows the Create Environment dialog box. You can identify each environment by assigning a color to it.



The 'Create Environment' dialog box has a title bar with a close button. Below the title bar is a horizontal line. The form contains three fields: 'Name' with the value 'Test', 'Description' with the value 'Testing Environment', and 'Blueprint' with a dropdown arrow. The 'Description' field is highlighted with a blue border.

| Field | Value |
|-------------|---------------------|
| Name * | Test |
| Description | Testing Environment |
| Blueprint | |

Figure 5-9 Create Environment dialog box

- To complete the creation of the environments, add the top-level group resources that were defined in “Defining resources” on page 86 to the corresponding environment. Click the name of the environment that you want, and click the **Resources** tab. Then, click **Add Base Resources**. Figure 5-10 shows the resource group that was added to the prod environment.

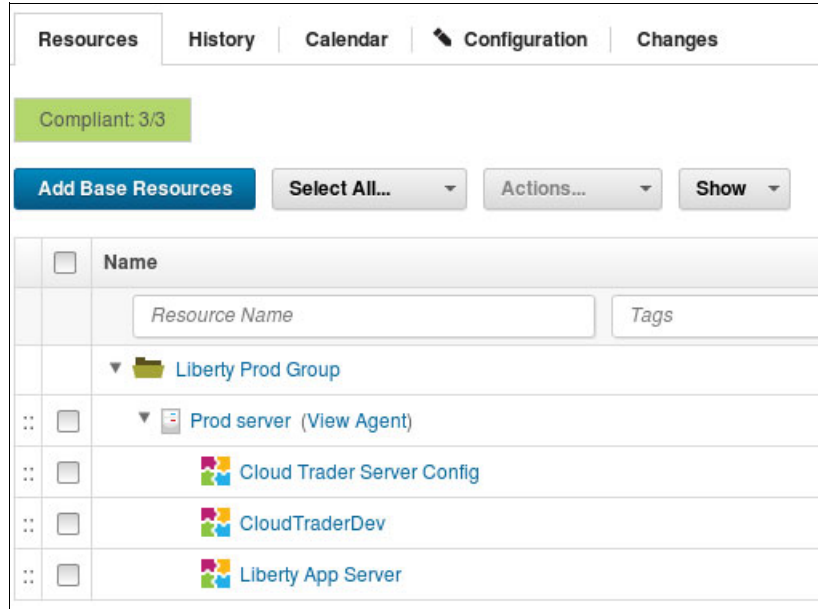


Figure 5-10 Prod environment resource group

- Each environment is linked to its own services and resources. In our example, a database instance exists for each environment. We define properties at the environment level. These properties are used during the replace token step that was defined in “Creating the Cloud Trader Server Config component” on page 84. Define these properties:

```

- trader_db_name = traderProd
- trader_db_passwprd = password
- trader_db_user = db2usr
- trader_db_port = 50000
- trader_db_server = database.server.hostname

```

Repeat Step 6 in each environment (test and prod). Example 5-1 shows a tokenized server.xml data source configuration.

Example 5-1 Tokenized server.xml data source configuration

```

<dataSource id='db2Datasource' jdbcDriverRef='db2-driver'
jndiName='jdbc/TradeDataSource' statementCacheSize='30' transactional='true'>
  <properties.db2.jcc id='db2-SecureMessageDB-props'
databaseName='@trader_db_name@' user='@trader_db_user@'
password='@trader_db_passwprd@' portNumber='@trader_db_port@'
serverName='@trader_db_server@'/>
</dataSource>
<jdbcDriver id='db2-driver' libraryRef='db2-library'/>
  <library id='db2-library'>
    <fileset id='db2-fileset' dir='${server.config.dir}/lib'
includes='db2jcc4.jar db2jcc_license_cu.jar'/>
  </library>

```

7. With the declared components, resources, and environments in the LibertyCloudTrader application, we can create the System Install process design for the LibertyCloudTrader application:

- Install Liberty App Server.
- Install Cloud Trader Server Config.
- Install CloudTrader.

Follow these steps:

- In the application configuration window, select the **Processes** tab and click **Create New Process**.
- In the process creation dialog box, enter the process name, in this case, System Install, and click **Save**.
- Click the new process element, and a process flow editor displays.

Figure 5-11 shows the flow for the System Install process design.

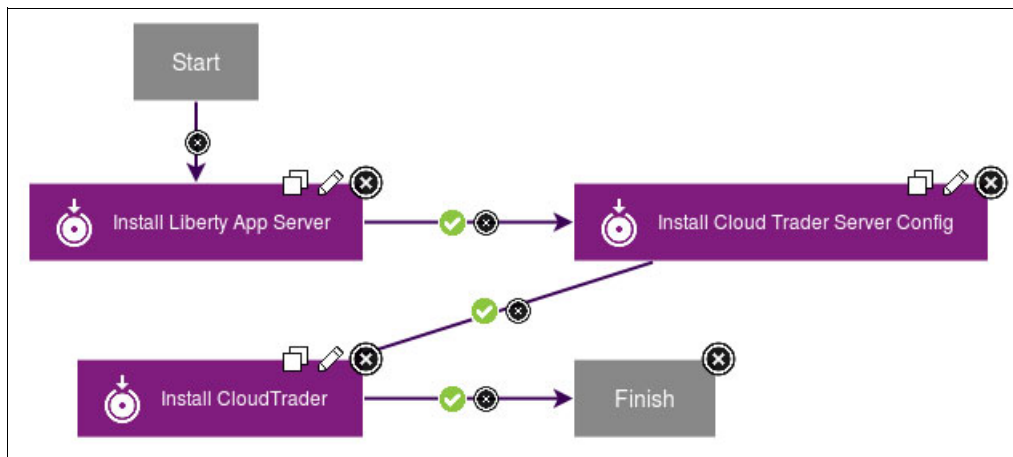


Figure 5-11 System Install process design

- Drag an **Install component** step, configure it in the following manner, and click **Save**:
 - Name: Install Liberty App Server
 - Component: Liberty App Server
 - Component process: Install Liberty Profile Server
- Drag an **Install component** step, configure it in the following manner, and click **Save**:
 - Name: Install Cloud Trader Server Config
 - Component: Cloud Trader Server Config
 - Component process: Install Configuration
- Drag an **Install component** step, configure it in the following manner, and click **Save**:
 - Name: Install CloudTrader
 - Component: CloudTrader
 - Component process: Install Cloud Trader Application
- Click the **Save** icon on the left toolbar to save the flow.

Running the deployment process

We are ready to run the deployment automation process:

1. Click the **request process** icon next to the environment definition to start the deployment. Figure 5-12 shows the highlighted request process icon.

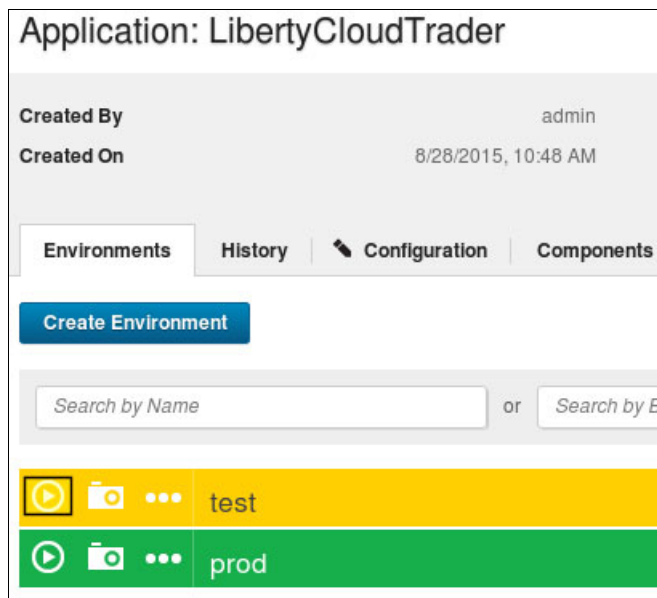


Figure 5-12 The test request process icon

2. Clear the **Only Changed Versions** check box to force the overlay of previous deployments. This function is useful for deployment automation testing. Figure 5-13 shows the Run Process on test dialog box.

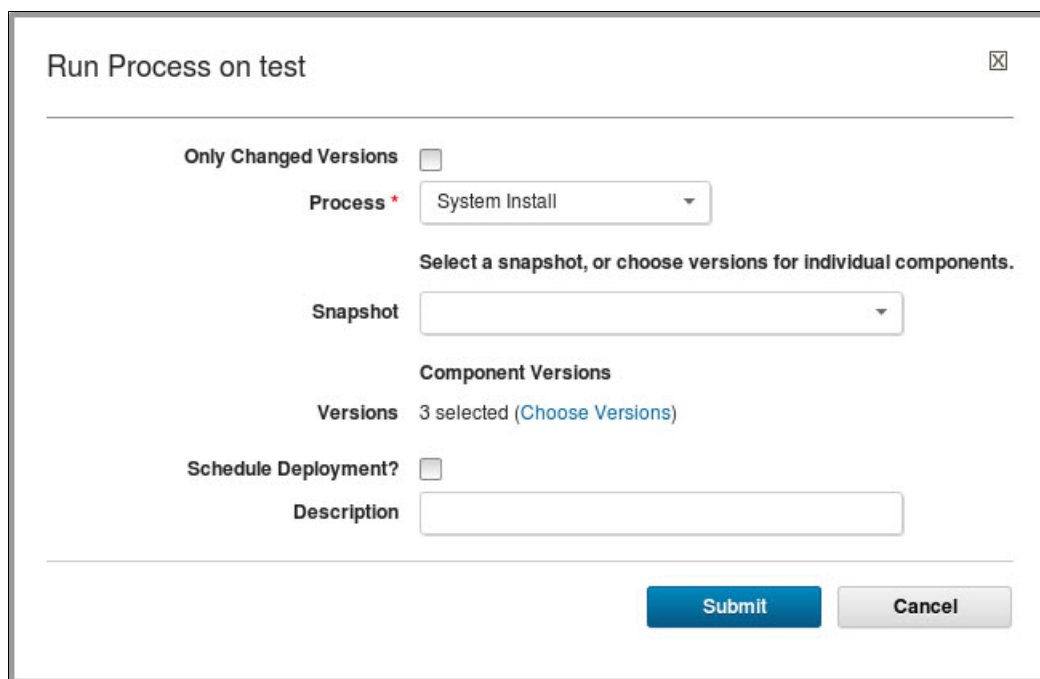


Figure 5-13 Run Process dialog box for the test environment

3. The submitted process automation displays in an execution panel. Figure 5-14 shows that the deployment process is successful.

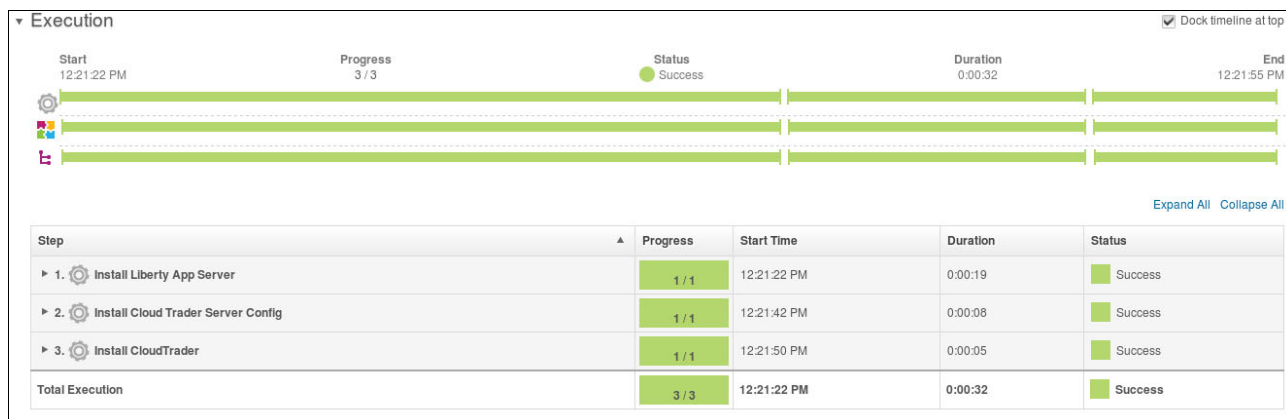


Figure 5-14 Successful deployment process

A single Liberty profile application deployment is complete. To deploy across more servers, add additional agents to a new or existing top-level group resource. Applications are deployed across multiple resources by adding them to an existing or new environment. Creating additional UCD environments is another way to group systems and control deployments, because environments are a specified option in running the application process.

5.2 How to use Jenkins to deploy applications to IBM Bluemix

In Chapter 3, “Code development, source code management, and build” on page 45, we use Jenkins to automate the application build and integration phases, and UCD and Jenkins for application deployment.

In this section, we describe how to use IBM Bluemix in association with Jenkins to enable *continuous delivery*. Our goal is to enable continuous delivery by reducing the time between defining system requirements (for example, server, database, and nodes) and installing the first version of the application.

We also want to reduce the dependency between developers and operations by providing a set of easy-to-use tools that define runtime environments and services. By combining these capabilities with the Jenkins automation tool, we obtain a powerful and flexible way of creating and deploying applications with a minimum of operational intervention.

For this process, we use the following resources:

- ▶ A Bluemix account
- ▶ Jenkins v1.609.2
- ▶ The Cloud Foundry plug-in for Jenkins v1.4.2

Defining an application in IBM Bluemix

Defining an application environment in IBM Bluemix is as easy as choosing the type of application run time and selecting the services that are associated with it. To define a Bluemix environment for LibertyCloudTrader application, follow these steps:

1. From the Bluemix dashboard, create a Cloud Foundry application by clicking **Create App**.
2. Select **Web App**, and then, click **Liberty for Java**.
3. Define an application name, and click **Finish**.

When the staging process completes, a Liberty Application server is ready to run a Java application. The next step is to create a database service to store the Cloud Trader application data.

Creating and binding services in Bluemix

Bluemix offers various services, such as SQL and non-SQL databases, analytics, web services, and data storage. The number of available services is always increasing. In our example, we bind a simple SQL service to our new Liberty application container:

1. From the Bluemix dashboard, scroll down to the application section, and click the application that was created in “Defining an application in IBM Bluemix” on page 94.
2. On the application panel, click **ADD A SERVICE OR API**. The API and service catalog display.
3. Locate and click the **SQL Database** service.
4. In the Add Service panel on the left, enter a name for the service, in our case, primarydb, and select the service plan that you prefer. Each plan provides different service levels, depending on the service that you select.

The necessary cloud infrastructure components are ready to run our CloudTrader application. Next, we configure our Jenkins job and project to publish our application to Bluemix.

Configuring a Jenkins job to publish an application to Bluemix

In 5.1.1, “Publishing component artifacts from Jenkins to UrbanCode Deploy” on page 78, we explained how to publish an application build to UCD. Here, we show how to define a post-build action to use the Jenkins Cloud Foundry plug-in to publish an application to Bluemix. Follow these steps:

1. From the Jenkins dashboard, click **Manage Jenkins** → **Manage Plug-ins**.
2. Click the **Available** tab and search for the Cloud Foundry plug-in.
3. Select **Download Now and install after restart**.
4. From the Jenkins dashboard, select the **Cloud Trader Liberty Profile** that we created in 3.2.5, “Create a Jenkins job by using the Rational Team Concert plug-in” on page 58, then click **Configure**.
5. Scroll down to the Post-build Actions section, and click **Add post-build action** to create an action of type **Publish to Cloud Foundry**.
6. Complete the Action fields in the following manner:
 - Target: `https://api.ng.bluemix.net`. This target can vary, depending on the Bluemix site for which you are creating your application.
 - Credentials: Create a credential that includes your Bluemix user information.
 - Organization: Enter your Bluemix organization. Typically, use your email address.
 - Space: The space where to create the Bluemix application, usually, dev.

7. Click **Test Connection**.
8. Check **Enter configuration in Jenkins**.
9. Configure the application options in the following manner:
 - Memory: 256 MB
 - Hostname: Your application name in Bluemix.
 - Instances: 1
 - Timeout: 60 (seconds)
10. Click **Add** next to the Service label.
11. Introduce the SQL database service name, primarydb, that we defined in “Creating and binding services in Bluemix” on page 94.
12. Click **Save**.

You can test this new configuration by running the Jenkins job or by requesting a new build from Rational Team Concert. See Figure 3-16 on page 58. For more information about running a Jenkins job, see 3.2.5, “Create a Jenkins job by using the Rational Team Concert plug-in” on page 58.



Production environment

This chapter describes the production environment characteristics of IBM WAS Liberty. The following topics are included:

- ▶ Solution architecture
- ▶ Liberty profile cluster deployment with UrbanCode Deploy
- ▶ Monitoring and analytics
- ▶ DevOps Services

6.1 Solution architecture

Figure 6-1 shows the solution architecture for the LibertyCloudTrader application. The architecture consists of the following components.

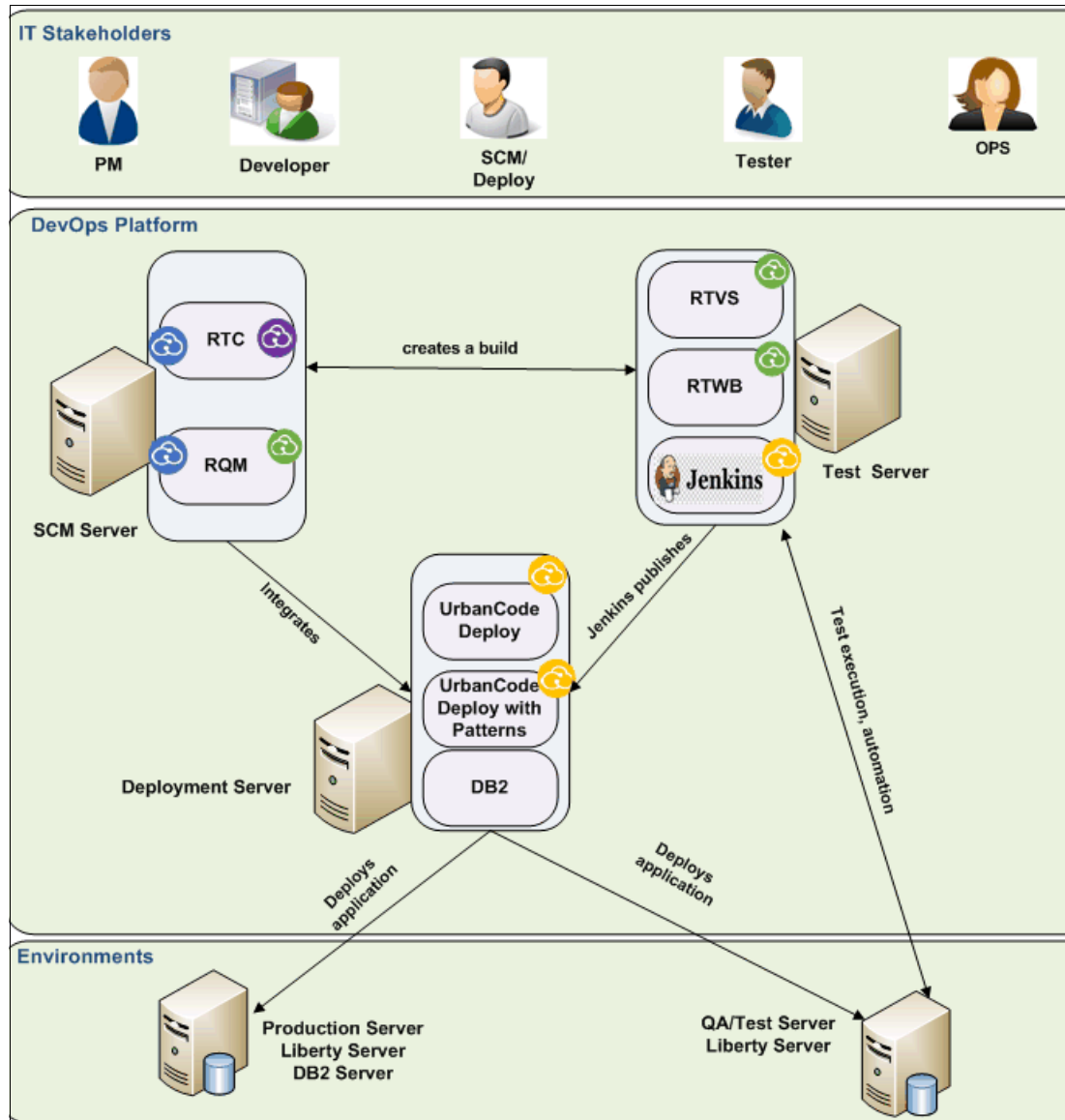


Figure 6-1 Liberty Cloud Trader Solution Architecture

6.1.1 Target environment

The following target environments are included in the solution:

- Quality Assurance (QA) Server

An instance of Liberty server Version 8.5.5 is installed on the QA Server. In a specific project, multiple QA environments are used for different stages of testing.

- Production Server

An instance of Liberty server Version 8.5.5 and IBM DB2 Version 10.5 software are installed on the Production Server.

Because this application is fairly simple, we assumed that developers are using their notebook computers as the development environment. But in a typical project, you must provision a development environment that is shared by developers.

6.1.2 DevOps platform

We installed DevOps toolset on the following three servers:

- Source control management (SCM) server

IBM Rational Team Concert Version 5.0.2 and IBM Rational Quality Manager Version 5.0.2. software are installed on the SCM server.

- Test server

IBM Rational Test Virtualization Service Version 8.7.0, IBM Rational Test Workbench Version 8.7.0, and open source Jenkins Version 1.6.0.9.2 software are installed on the test server.

- Deployment server

UrbanCode Deploy (UCD) Version 6.1.1 and UrbanCode Deploy with Patterns Version 6.1.1 are installed on the deployment server. On this server, we also installed an instance of IBM DB2 Server Version 10.5 and IBM Rational License Key Server (RLKS) Version 8.1.4, which are used by the DevOps tools.

6.1.3 Delivery pipeline

In this example, the iterative flow of taking the LibertyCloudTrader application through the delivery pipeline is shown:

1. The project or release manager creates a project/release plan, which includes development activities.
2. The developer uses Rational Team Concert for collaboration. The developer uses the Liberty developer tool for Eclipse and starts coding assigned application components.
3. The developer performs unit testing by using JUnit and Rational Team Concert.
4. After the developer is satisfied with the application, the developer checks the code into the Rational Team Concert code repository.
5. The developer runs an entire set of JUnit test cases in Rational Quality Manager. A code review is conducted.
6. The SCM subject matter expert (SME) uses Rational Team Concert to send a build creation request to Jenkins and the code components that need to be included in that build.

7. Jenkins creates a build with a link to the code artifacts in Rational Team Concert and publishes the build to the Deployment Server.
8. The SCM SME creates a deployment process and snapshot in UCD that includes the latest build that was created by Jenkins and deploys it to QA Server.
9. The tester runs the system and integration test cases that were defined in Rational Quality Manager and creates any defects in Rational Team Concert.
10. The tester uses test automation and test service virtualization. In our example, we virtualized a Java Database Connectivity (JDBC) transaction.
11. If any defects were generated during this testing, defects are created in Rational Team Concert and assigned to the developer.
12. The developer obtains the defects from Rational Team Concert and adds fixes to the code. This cycle continues until all defects are addressed.
13. The Deployment Server deploys the code snapshot that was tested and deploys it to the Production Server.

Important: Selected DevOps tools must be integrated and they must provide traceability across various application lifecycle management (ALM) phases to allow the iterative flow of the application through the pipeline faster. In this scenario, we selected primarily Rational DevOps tools but we were able to show traceability across these phases.

6.2 Liberty profile cluster deployment with UrbanCode Deploy

One of the most important requirements for a production environment is high availability (HA). This section explains how to deploy a WAS Liberty profile collective cluster by using UCD.

What is the difference between a controller and a member? The answer is configuration. In Chapter 5, “Deployment” on page 77, the Cloud Trader application configuration was defined as an UCD component that contains configuration artifacts. Now, we split that component into two variations: a Liberty configuration for the controller, that is, the collective cluster controller, and a Liberty configuration for the member that is a collective cluster member. The main idea is to deploy one controller and as many members as you need. The following example shows one of each component that is deployed on a production environment.

6.2.1 Creating a controller component

To reuse the process and setting that were previously defined on the Cloud Trader Server configuration, copy it by using these steps:

1. Go to components on the UCD dashboard and select **Copy** from the Cloud Trader Server Config component menu.
2. Define the new component name as Cloud Trader Server Config Controller.

Edit the new controller configuration component to include the controller creation steps in the install configuration process by using these steps:

1. Go to the **Process** tab for the Cloud Trader Server Config Controller component.
2. Click **Edit** on the install configuration process.
3. Drag a **shell** step from the step palette, and configure its fields in the following manner, and then, click **save**:
 - Working directory: `${p:liberty_directory}`
This variable defines the liberty root path.
 - Shell script, which is shown in Example 6-1. This command configures our server as a collective controller and creates a cluster config file in the server instance root.

Example 6-1 Shell script

```
export
JVM_ARGS=-Dcom.ibm.websphere.collective.utility.autoAcceptCertificates=true
bin/collective create ${p:liberty_serverName}
--keystorePassword=Keystorepassword
--createConfigFile=${p:liberty_directory}/usr/servers/${p:liberty_serverName}
}/cluster-config.xml
```

4. Drag a **Start Server** step from the step palette, and configure its fields in the following manner:
 - WAS Liberty installation directory: `${p:liberty_directory}`
This variable defines the liberty root path.
 - Server name: `${p:liberty_serverName}`

Figure 6-2 shows the process flow design.

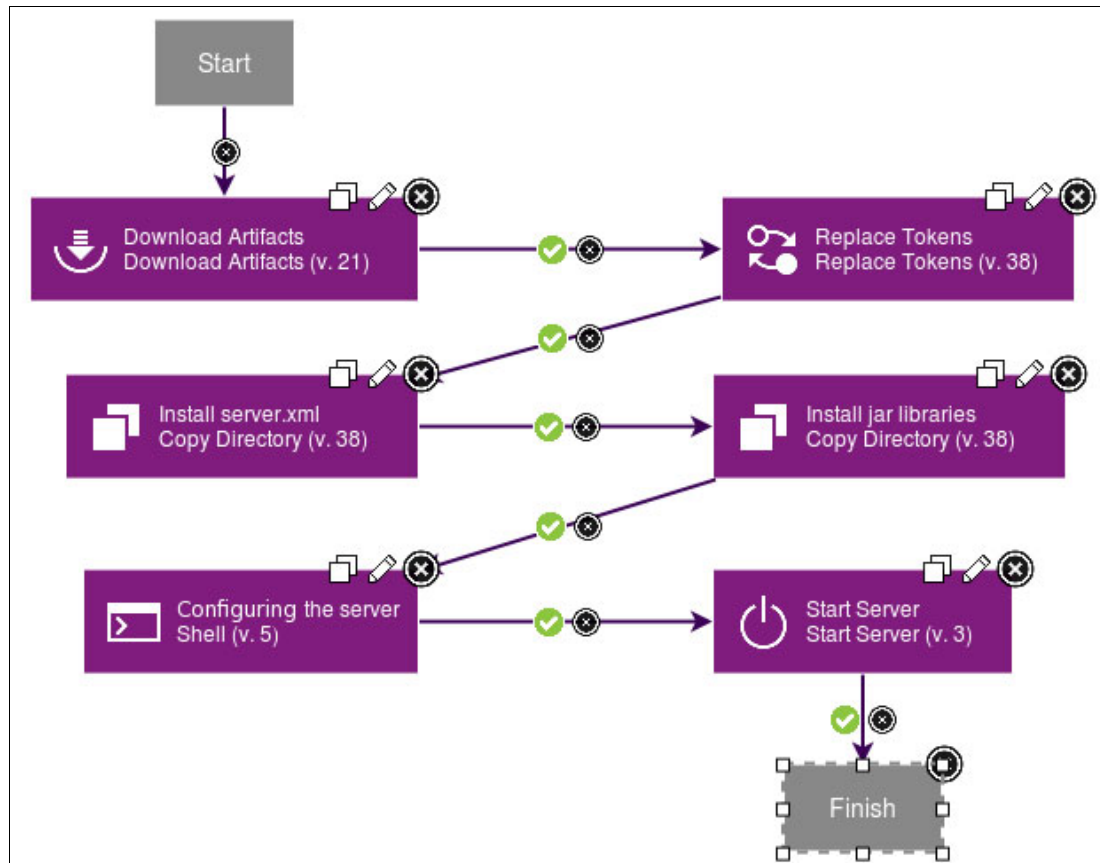


Figure 6-2 Install configuration process flow design

6.2.2 Creating a member component

For the member component, edit the Cloud Trader Server Config component to add the necessary steps to join a Liberty server as a collective member:

1. Go to the **Process** tab of the Cloud Trader Server Config component.
2. Click **Edit** at the Install configuration process.
3. Drag a new **shell** step from the step palette, and configure its fields in the following manner:
 - Working directory: `${p:liberty_directory}`
This variable defines the liberty root path.
 - Shell script, which is shown in Example 6-2. Most of the properties that are used in this script are defined in the following section.

Example 6-2 Shell script

```

export
JVM_ARGS=-Dcom.ibm.websphere.collective.utility.autoAcceptCertificates=true
bin/collective join ${p:liberty_serverName}
--host=${p:liberty_controller_address} --port=${p:liberty_controller_port}
--keystorePassword=password --password=${p:liberty_controller_password}
--user=${p:liberty_controller_user}

```

```
--createConfigFile=${p:liberty_directory}/usr/servers/${p:liberty_serverName}/cluster-config.xml
```

4. Add a **Start Server** step, and configure it in the following manner:

- WAS Liberty installation directory: \${p:liberty_directory}

This variable defines the liberty root path.

- Server name: \${p:liberty_serverName}

Figure 6-2 on page 102 shows the process flow design.

Note: Collective command parameters can vary depending on your requirements. For more information about collective options, see the following web address:

<http://ibm.co/1knNKHY>

6.2.3 Defining agents and component distribution

This section redefines the production resource group that was created in Chapter 5, “Deployment” on page 77 and adds an agent to host the collective member. Then, you must redistribute the components to install the correct configuration, depending on whether the server is a controller or a member. Follow these steps:

1. Go to the **Resources** tab on the UCD dashboard. Expand the **Liberty Prod Group** that is defined in Chapter 5, “Deployment” on page 77. Add an agent. We describe how to define an agent in “Defining components” on page 81.
2. Add the following components to the added agent:
 - **Cloud Trader Server Config Controller** (controlled configuration component)
 - **CloudTrader** (CloudTrader application component)
 - **Liberty App Server** (Liberty profile application server component)

By altering the Cloud Trader Server Config, the agent that was defined in Chapter 5, “Deployment” on page 77 for production now contains the member configuration component. You can replicate that structure to create as many collective members as you want. If you intend to deploy an environment with only one server, ensure that you deploy the controller component to prevent errors when the deployment process tries to join the member to the controller.

6.2.4 Configuring the LibertyCloudTrader application and environment

This section describes the creation of the necessary environment properties and processes to deploy the collective cluster.

Defining the environment properties for a controller connection

The following steps show how to define the environment properties for the controller connection:

1. Go to the **Applications** tab on the UCD dashboard, and select **LibertyCloudTrader**.
2. Edit the environment configuration by clicking the environment name at the Environment tab. In our case, we edited the Prod environment.

3. Go to the **Configuration** tab, and select **Environment Properties**.
4. To create the following properties, click **Add Property**:
 - liberty_controller_address = *<controller host name>*
 - liberty_controller_password = *<administrator user password of the controller server>*
 - liberty_controller_user = *<administrator user name of the controller server>*
 - liberty_controller_port = *<controller ssl port>* . This port is usually port 9443.

These properties are used by the install configuration process to join new collective members to the controller.

Configuring the application deployment process

The final step from the UCD application perspective is to modify the deployment process that is defined in Chapter 5, “Deployment” on page 77 to deploy the Controller configuration component. Follow these steps:

1. Go to the **Applications** tab on the UCD dashboard. Select **LibertyCloudTrader**.
2. Go to the **Process** tab, and select **System Install**.
3. In the process designer, add an **install component** step, and configure it to execute the Install configuration process of the Install Cloud Trader Server Config Controller component.

Figure 6-3 shows the System Install process flow design.

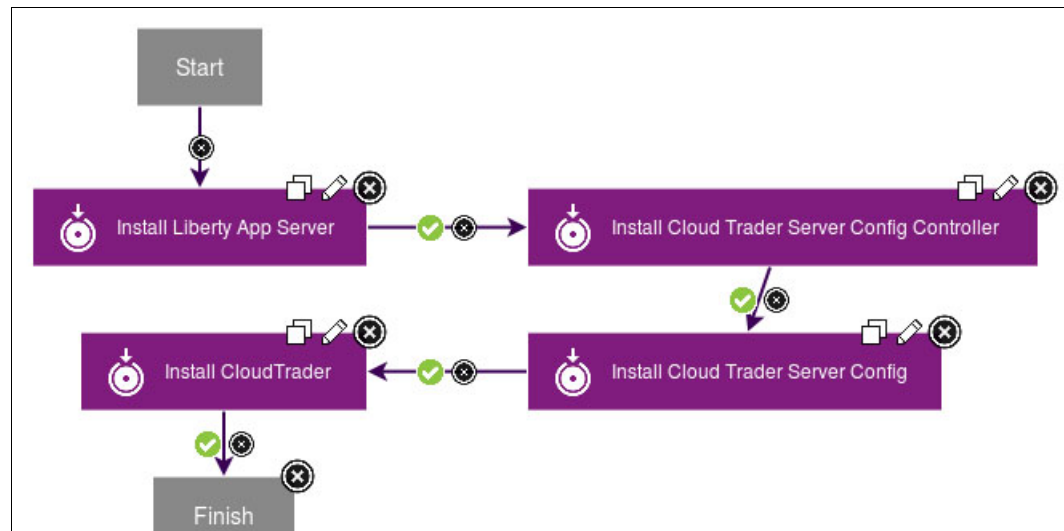


Figure 6-3 System Install process flow design

The UCD application is configured. Next, you need to modify the server.xml file to include the cluster feature and the administrator user registry entry. (This entry is necessary to allow a member to join the collective controller.)

6.2.5 Adding the required configuration to the server.xml file

To prepare the server configuration to work as a cluster member or controller, you must add several entries to the `server.xml` file that is included in the Cloud Trader Server Config component. In this book, we use the same version of the file to set both the controller and the member. You can create separate versions of the file for the controller and the member, if you prefer.

The following entries must be added to the `server.xml` file:

- ▶ Include the following feature inside the `featureManager` tags: `clusterMember-1.0`
- ▶ If no entry exists, create a new `basicRegistry` after the `featureManager` tag and include the following entries: `<user password="password" id="wasadmin" name="wasadmin"/>`
- ▶ If no entry exists, create a new `administrator-role` tag after the `basicRegistry` tag and include the following entry inside: `<user>wasadmin</user>`
- ▶ Include the location of the configuration file:
`<include location="${server.config.dir}/cluster-config.xml" />`

The location must contain the path of the cluster configuration file that was created during the controller or member creation.

Example 6-3 shows the `server.xml` file after the required entries were added.

Example 6-3 Server.xml file

```
<server description="dblogin server">
<!--
(C) COPYRIGHT International Business Machines Corp., 2014
-->
<!--
All rights reserved * Licensed Materials - Property of IBM
-->
<!-- Enable features -->
<featureManager>
<feature>jsp-2.2</feature>

<feature>localConnector-1.0</feature>
    <feature>jaxrs-1.1</feature>
    <feature>servlet-3.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>ssl-1.0</feature>
<feature>monitor-1.0</feature>
<feature>jdbc-4.0</feature>
<feature>clusterMember-1.0</feature>
</featureManager>
<httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" >

    </httpEndpoint>

<basicRegistry>
<user password="password" id="wasadmin" name="wasadmin"/>
</basicRegistry>
```

```
<administrator-role>
<user>wasadmin</user>
</administrator-role>

<include location="${server.config.dir}/cluster-config.xml" />
</server>
```

Now, all of the components can be deployed to the production environment. The result consists of two servers, a controller, and a member running a default cluster of the CloudTrader application.

6.3 Monitoring and analytics

Use IBM Performance Management to monitor and troubleshoot application stacks and visualize the data in a browser console. Performance Management can work with both Performance Management Agents and IBM Tivoli Monitoring Agents. You can configure these agents to monitor individual components in a data center, public cloud, or hybrid combination.

The data is displayed in the Application Performance dashboard, which is a web-based console that offers a comprehensive overview of your applications, the health of their components, and runtime environment statistics.

For more information about IBM Performance Management, see the following web address:

<http://www.ibm.com/support/knowledgecenter/SSHLNR/welcome>

6.3.1 Required software products

For WAS Liberty profile monitoring, IBM Tivoli Composite Application Manager Agent for WebSphere Applications Version 7.10.0 was configured to receive the data from the Data Collector of the Monitoring Agent for WebSphere Applications Version 7.30.09. For the operating system, the Monitoring Agent for Linux OS Version 6.35.09 was used with the Monitoring Agent for DB2 Version 7.10.03 for DB2.

6.3.2 Configuration of Data Collector

After you install the Performance Management server and the agents, the next step is to configure the Data Collector of the Monitoring Agent for WebSphere Applications to use IBM Tivoli Composite Application Manager Agent:

1. Go to `yndchome` on the Monitoring Agent for WebSphere Applications installation home directory. By default, the path is `/opt/ibm/apm/agent/yndchome`, as shown in Figure 6-4.

```
[root@       bin]# pwd
/opt/ibm/apm/agent/bin
[root@       bin]# cd ..
[root@       agent]# ls
auditlogs  bin  classes  clienttime  config  ExitPoint  JRE
[root@       agent]# cd yndchome
[root@       yndchome]# cd 7.3.0.9.0/
[root@       7.3.0.9.0]# cd bin
[root@       bin]# pwd
/opt/ibm/apm/agent/yndchome/7.3.0.9.0/bin
[root@       bin]# ls
ampassInst.sh  configSOAWASClientDC.sh  createcfg.py
config_i5.sh   configtemplate.sh        createcfg_z0S.sh
config.py      configtemplatexd.py      deletetemplate.sh
config.sh      config_z0S.sh            deletetemplatexd.py
[root@       bin]#
```

Figure 6-4 Monitoring Agent for WebSphere Applications Data Collector home directory

2. Run the **config.sh** script, which is in the data collector `bin` directory, from the command line:

```
/opt/ibm/apm/agent/yndchome/bin
```

Note: You must set the `JAVA_HOME` variable to use the same Java Runtime Environment (JRE) that is used by Liberty profile.

3. Follow the step-by-step configuration (Figure 6-5) and restart the application server after you finish.

```
Data collector configuration summary

-----
Each of the servers will be configured for data collection

1) List of servers selected

- WAS server: opt. IBM.WebSphere.wlp.CloudTrader()
  WAS node: opt. IBM.WebSphere.wlp
  WAS cell: opt. IBM.WebSphere.wlp

  WebSphere Profile home      :
    /opt/IBM/WebSphere/wlp/usr

    WAS version : 8.5.5.6
    Deployment  : Standalone
    JVM mode    : 64
    Server alias : CloudTrader
    Configuration home : /opt/ibm/apm/agent/yndchome/7.3.0.9.0

2) Integrate with ITCAM Agent for WebSphere Applications : Yes

  Config app server for TEMA : Yes
  Application Performance Diag : Yes
  TEMA hostname or IP address : 127.0.0.1
  TEMA port number : 63335

3) Integrate with Transaction tracking : Yes

  Transaction Framework Extension hostname : 127.0.0.1
  Transaction Framework Extension port number : 5457

4) Advanced settings :

  Set Garbage Collection log path : No

Configuration sections:

1) List of servers selected
2) Integrate with ITCAM Agent for WebSphere Applications
3) Integrate with Transaction tracking
4) Advanced settings

To modify a section, enter the number. To modify all sections, enter "***". To accept your configuration without modifying, enter "a". To
a

Do you want to backup current WebSphere configuration? [1 - YES, 2 - NO] [default is: 2]:
2
Processing Configuration call for WebSphere Liberty profile: opt. IBM.WebSphere.wlp
Successfully configured data collector for server CloudTrader
Application server (CloudTrader) should be restarted
Summary:
  CloudTrader (OK)
Please ensure the account that was used to run the application server has the read and write privileges to DCHome/runtime directories.
```

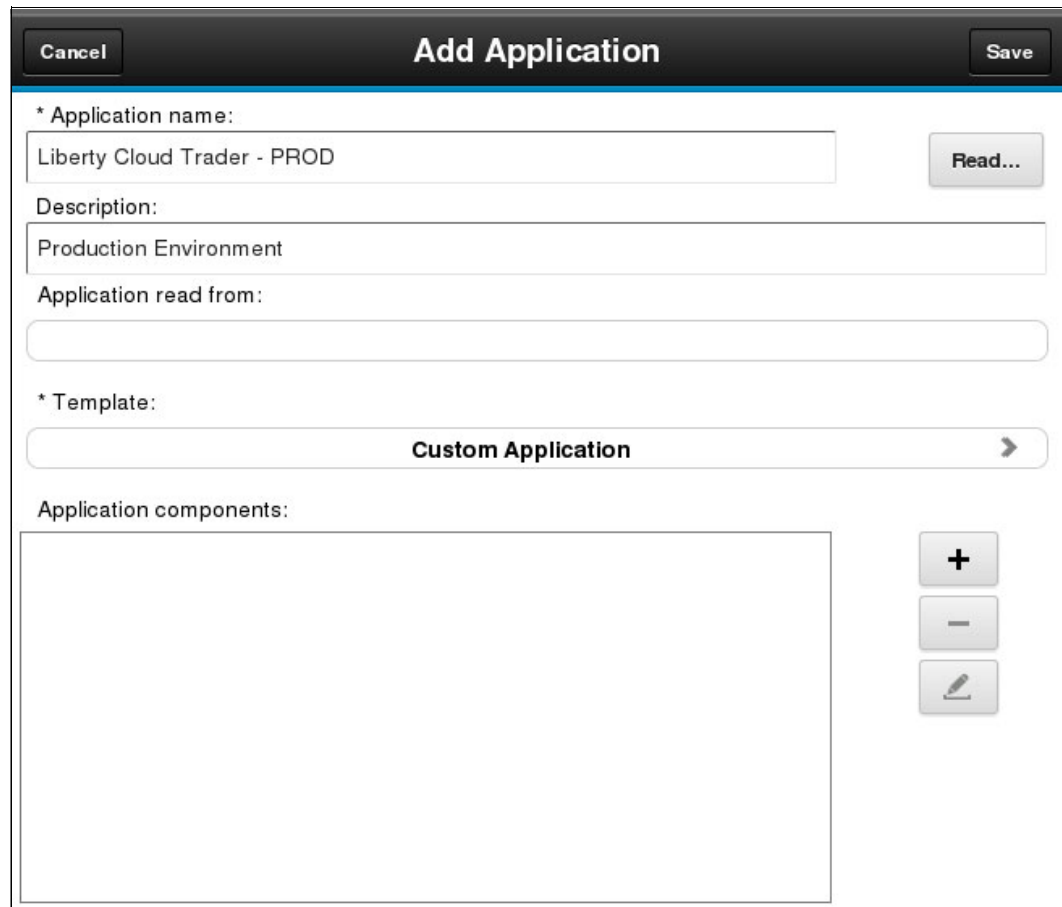
Figure 6-5 Data Collector configuration

4. Optional: Install and configure the OS and DB2 agents. For a full stack monitoring, you can configure the OS and DB2 agents. For more information, see this web address:
http://www.ibm.com/support/knowledgecenter/SSHLNR_8.1.1/com.ibm.pm.doc/install/config_agents_intro.htm?lang=en

6.3.3 Visualize the data in the Application Performance dashboard

To visualize the data in the Application Performance dashboard, use the following steps:

1. Log on to Application Performance dashboard. Click the plus sign (+) to configure a new application, as shown in Figure 6-6.



The screenshot shows a dialog box titled "Add Application" with a dark header bar containing "Cancel" and "Save" buttons. The main content area is white and contains the following fields and controls:

- * Application name:** A text input field containing "Liberty Cloud Trader - PROD". To its right is a "Read..." button.
- Description:** A text input field containing "Production Environment".
- Application read from:** An empty text input field.
- * Template:** A dropdown menu showing "Custom Application" with a right-pointing arrow.
- Application components:** A large empty rectangular box. To its right are three vertically stacked buttons: a plus sign (+), a minus sign (-), and a pencil icon.

Figure 6-6 Add application components

2. Select the components that are part of your application environment, as shown in Figure 6-7 and Figure 6-8 on page 111. Click **Save**.

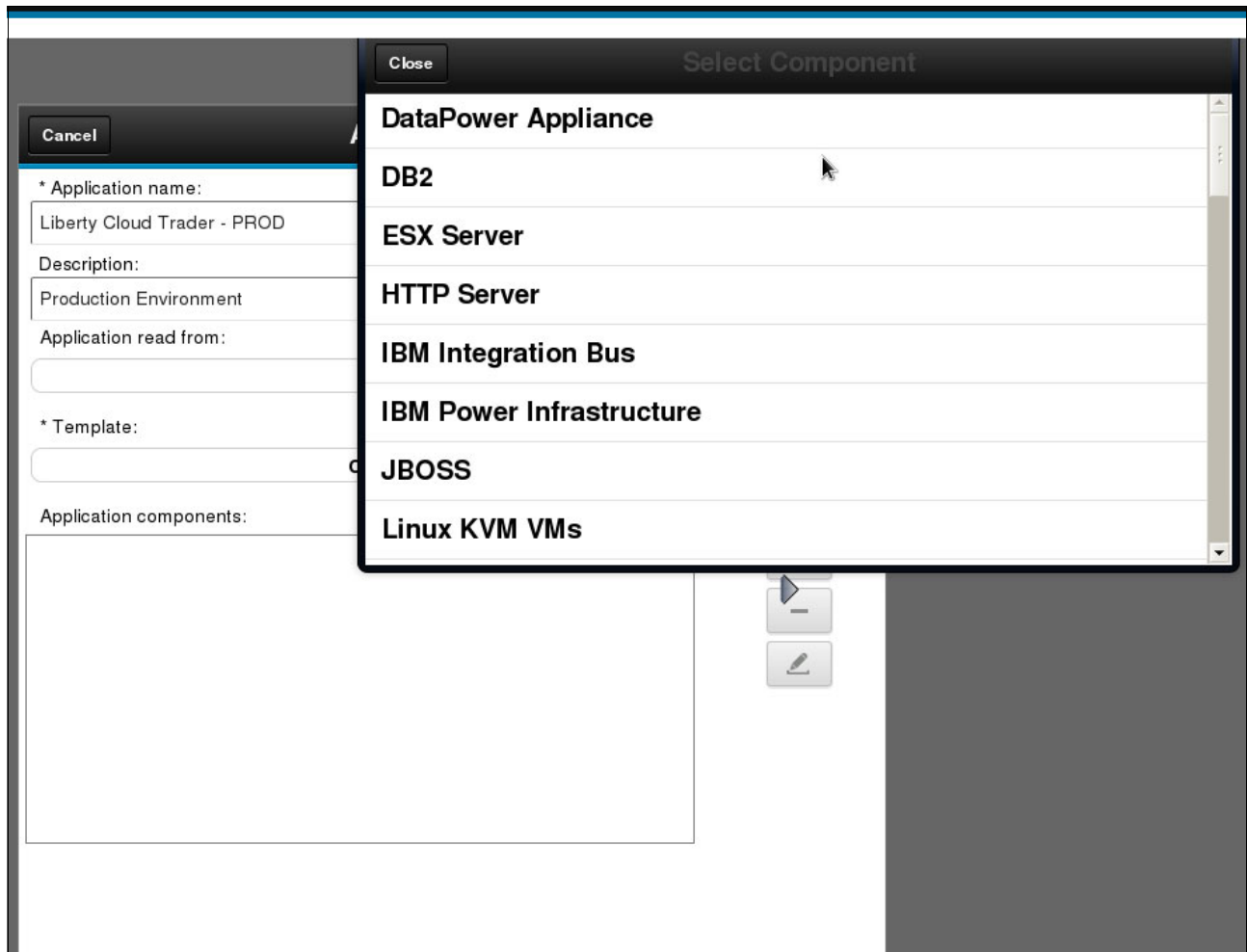


Figure 6-7 Select the components that are part of your application

Note: You can create a new application to monitor multiple components, such as the OS, DB2 database, and application server. Or, you can create a new application to monitor single components only, such as an application server or a database.

Cancel

Add Application

Save

* Application name:

Liberty Cloud Trader - PROD

Read...

Description:

Production Environment

Application read from:

* Template:

Custom Application

Application components:

lexbz181071 - Linux OS(1)

lexbz181071:LZ

db2inst1:lexbz181071 - DB2(1)

db2inst1:lexbz181071:UD

CloudTrader:lexbz181071 - WAS(1)

CloudTrader:lexbz181071:KYNS

+

-

Figure 6-8 Application components

Chapter 6. Production environment 111

3. You can see the new application data, which is displayed on the dashboard, as shown in Figure 6-9.

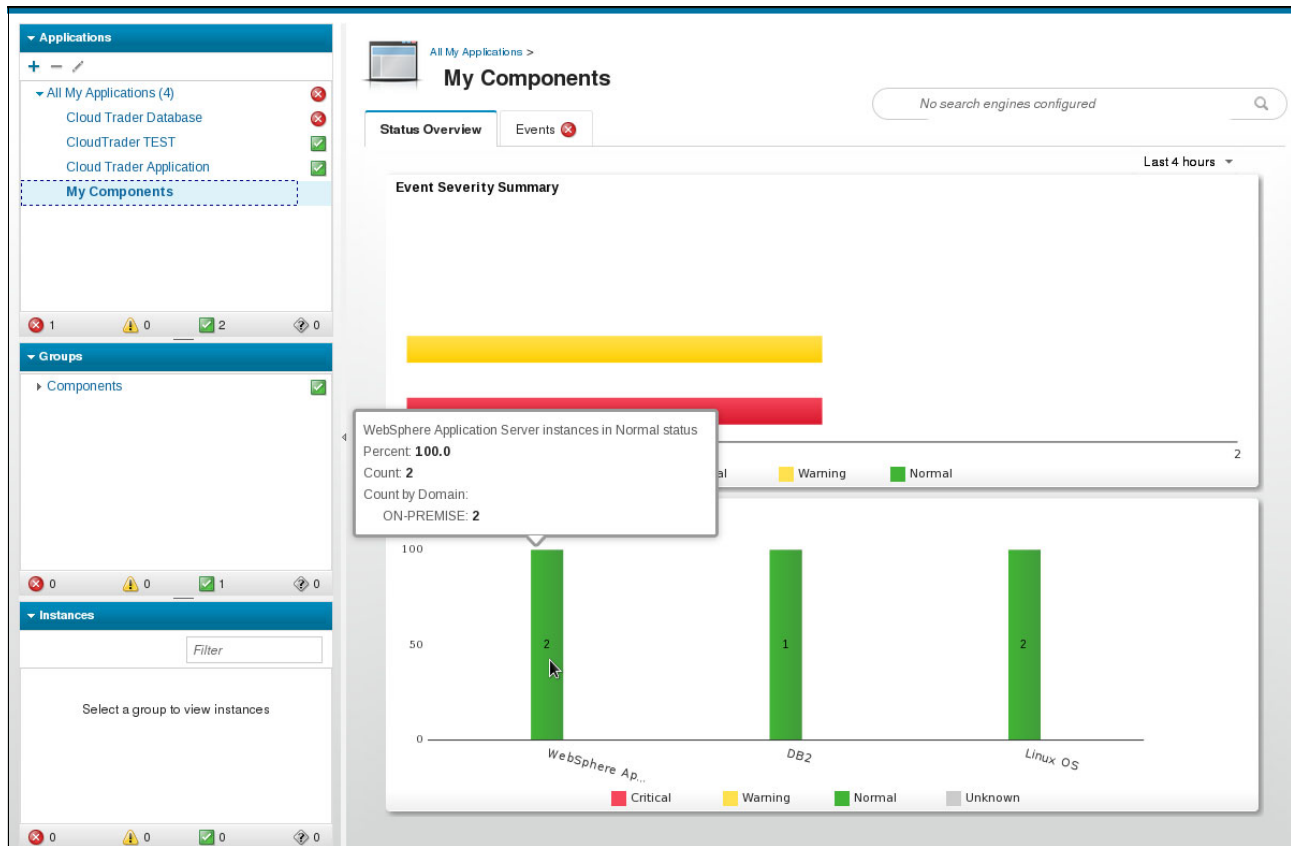


Figure 6-9 Application dashboard

You can monitor several resources for the WAS Liberty profile:

- ▶ WAS Liberty profile information
- ▶ Average response time
- ▶ JVM garbage collector
- ▶ Enterprise JavaBeans (EJB) containers
- ▶ Slowest web services
- ▶ Log messages
- ▶ Busiest DB connection pools

Figure 6-10 shows a quick overview of the Java virtual machine (JVM) status.

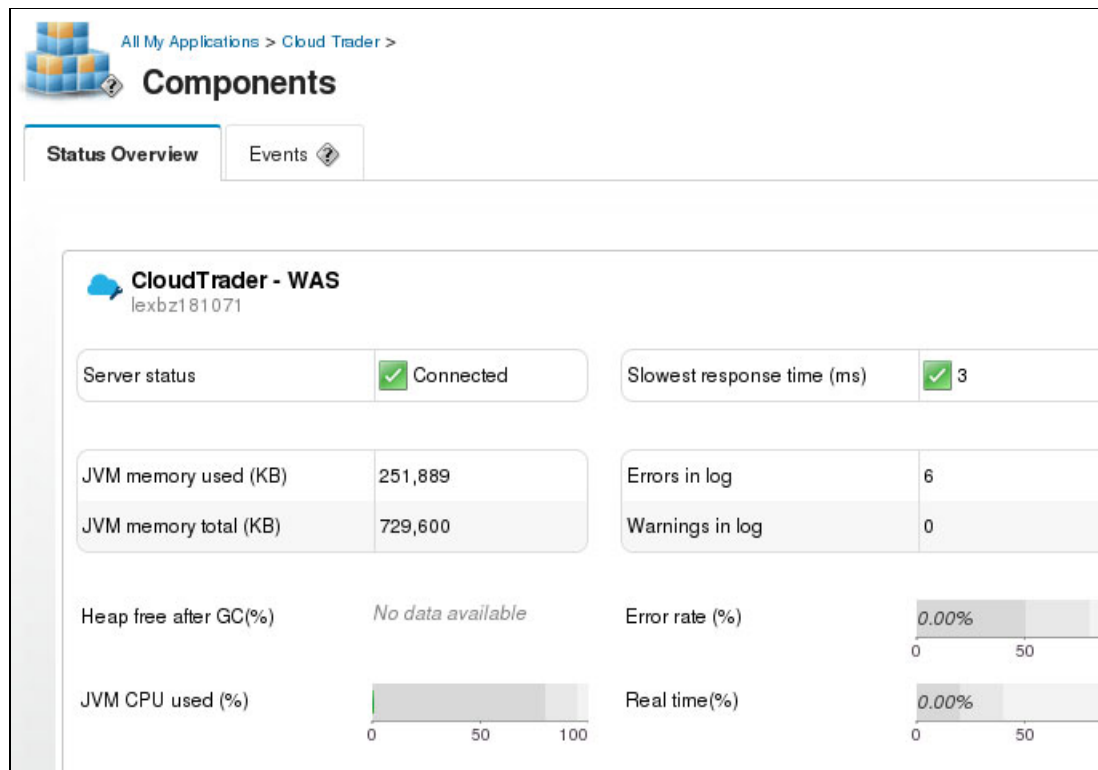


Figure 6-10 Liberty JVM status

Figure 6-11 shows the Liberty servlet container status.

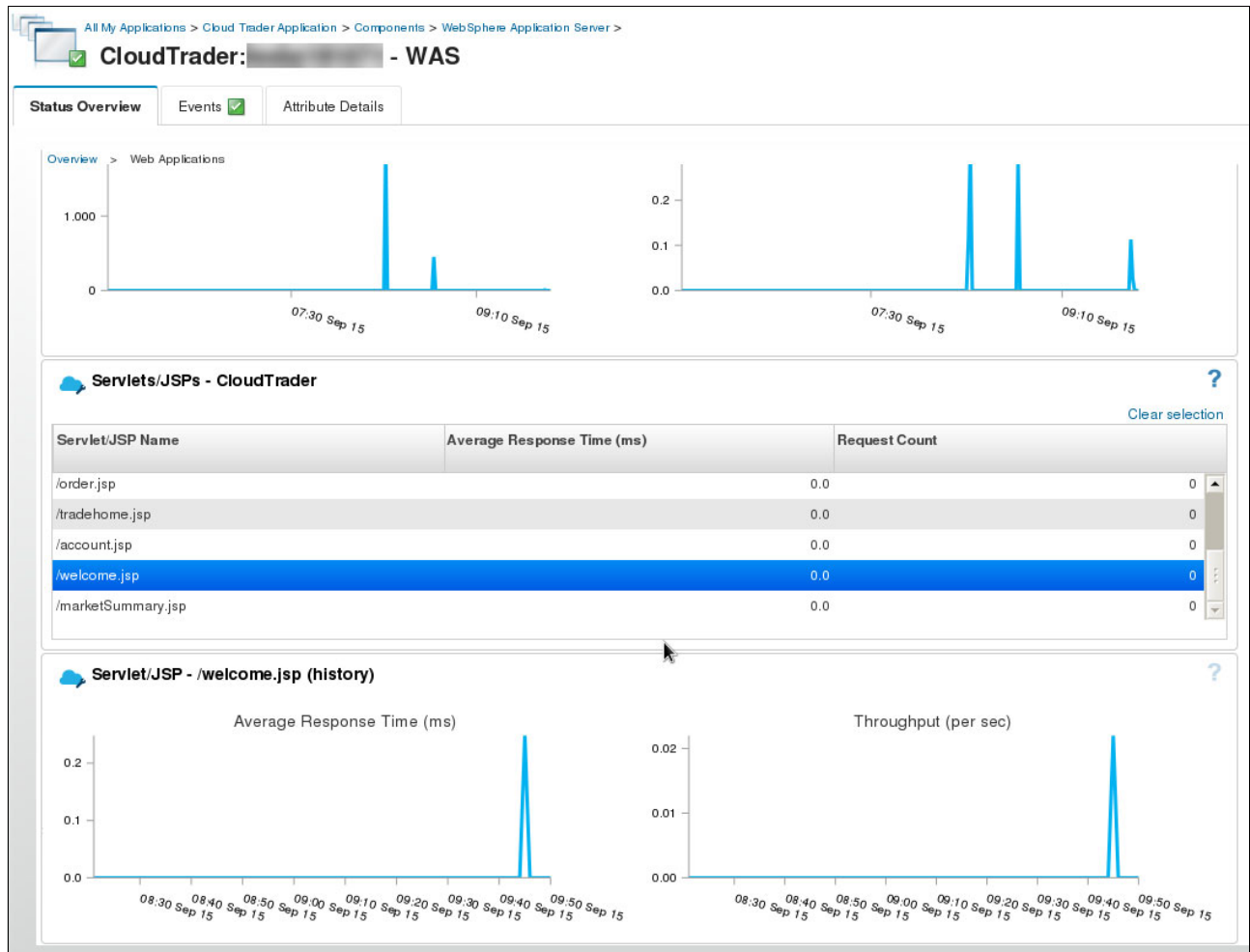


Figure 6-11 Liberty servlet container status

Figure 6-12 shows the database events.

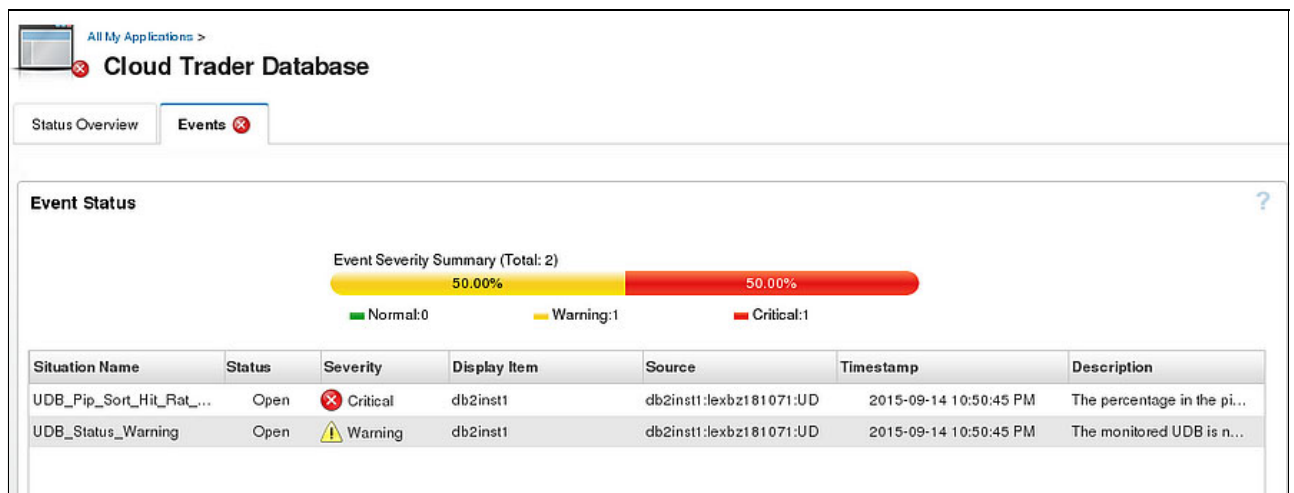


Figure 6-12 Database events

6.4 DevOps Services

IBM Bluemix is an open standards-based cloud Platform as a Service (PaaS) offering. Bluemix is based on Cloud Foundry running on IBM SoftLayer. Bluemix provides a Liberty for Java Runtime buildpack for Java development. It supports both web archive (WAR) and enterprise archive (EAR) files. The Liberty buildpack provides a WAS Liberty container that can run Java Platform, Enterprise Edition 7 and Open Services Gateway initiative (OSGi) applications. It supports popular frameworks, such as Spring, and it includes the IBM JRE.

Even though our team was unable to get a chance to take the LibertyCloudTrader application through the delivery pipeline within Bluemix, apparently, it is possible to get through the full pipeline from within Bluemix. Certain gaps might exist in stages of the lifecycle but this area is continuously evolving.

IBM Bluemix provides DevOps Services that includes the Track and Plan service and the Build and Deploy service, which is also called the *Delivery Pipeline*. You can handle most application lifecycle management (ALM) phases from within the Bluemix platform (that is, plan, develop, build, and deploy a Java application all within the Bluemix environment).

Figure 6-13 shows a list of DevOps Services in the Bluemix catalog.

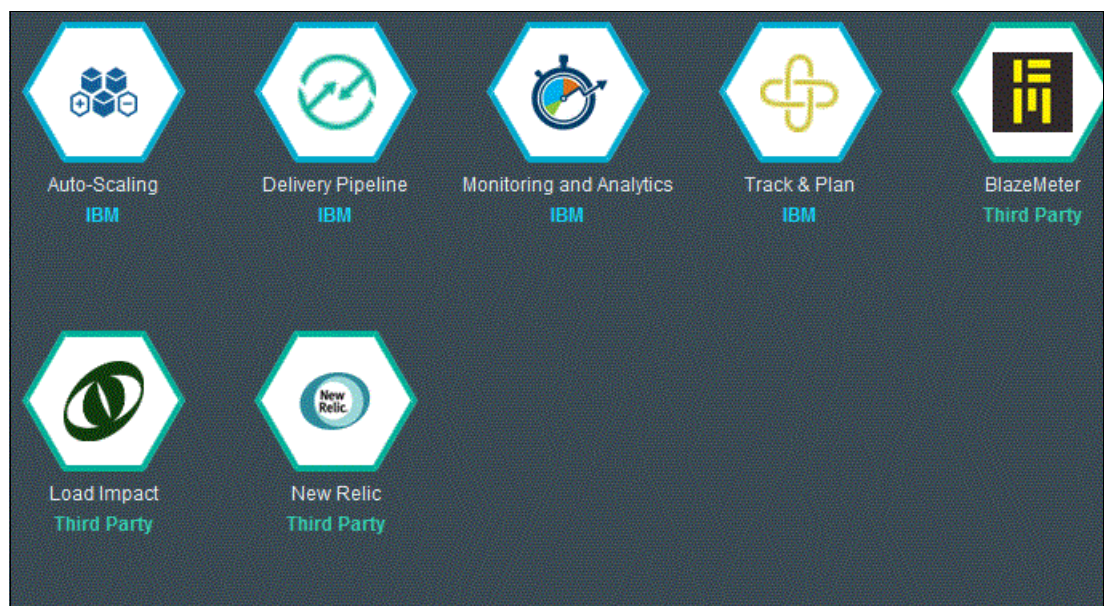


Figure 6-13 Bluemix DevOps Services

6.4.1 Plan

With the IBM Bluemix DevOps Services Track and Plan feature, you can create project work items and track them throughout the project. A project can be created and linked to a code repository.

Several code repository options are available within Bluemix. You can use GitHub (if your team is already familiar with it), Git repo, which is built into Bluemix, or Jazz source control management (SCM) for developers who prefer to use Rational Team Concert and Jazz tooling.

6.4.2 Development

For code development, the developer can use Web integrated development environment (IDE), which is a browser-based development environment. If the developer wants to use another IDE environment, Bluemix Live Sync synchronizes the changes in the developer's local file system with the cloud workspace in IBM Bluemix DevOps Services.

Bluemix also provides the option to use the local Eclipse environment for coding. In this case, you must install the Rational Team Concert plug-in. From the Eclipse environment, you can connect to Bluemix DevOps Services. The project must use the Jazz SCM repository in this case.

Certain change and configuration management features that are in Rational Team Concert might not be available in DevOps Services. For more information, see the following web address:

https://hub.jazz.net/docs/jazz_scm_client/#connect_to_your_devops_services_projects_from_eclipse

6.4.3 Build and deploy

The IBM Bluemix DevOps Services Build and Deploy feature, which is also known as the *Delivery Pipeline*, automates the continuous deployment of the application. Stages are defined that organize the input and jobs as the application is built, deployed, and tested (that is, as the application goes through the pipeline).

The simplest pipeline has two stages: Build Stage and Deploy Stage. The Build Stage takes the code changes from the code repository, such as GitHub, and after the build completes, the Deploy Stage, which deploys the application to Bluemix, is initiated.

The following methods are available to deploy a Liberty application to Bluemix:

- ▶ Pushing a stand-alone application
- ▶ Pushing a server directory
- ▶ Pushing a packaged server

Stand-alone application

Stand-alone applications, such as WAR or EAR files, can be deployed to Liberty in Bluemix by using the **cf push** command with the **-p** parameter that points to a WAR file or an EAR file.

To deploy a stand-alone application, run the **cf push** command with the **-p** parameter that points to your WAR or EAR file, for example, **\$ cf push <yourappname> -p myapp.war**.

Server directory

If the Liberty profile is installed on the developer workstation, you can create a server directory with the application. A **server.xml** file can be pushed to Bluemix by using the following command:

```
$ cf push <yourappname> -p defaultServer
```

Packaged server

A packaged server file is created by using Liberty's server package command, which can be pushed to Bluemix. The packaged server file can contain shared resources and Liberty user features in addition to the application and configuration files that are needed by the application.

For example, if your Liberty server is `defaultServer`, run the command:

```
$ wlp/bin/server package defaultServer --include=usr
```

Under DevOps Services in the Bluemix catalog, an Integration Testing service, which is an experimental service at the moment, is available. Integration testing needs to synchronize with Bluemix. It uses an oauth-token to send space and application requests to Bluemix in the same manner as the `cf` command-line tool. You can test application services by using this testing service. You can add a test stage to DevOps Build and Deploy that runs automation or service tests after the build is complete and before the application is deployed to Bluemix.

For more information, see the following web address:

<https://console.ng.bluemix.net/catalog/integration-testing/>

The Monitoring and Analytics service in Bluemix provides a service to monitor and perform diagnostics of a Liberty or Node.js application during development, test, or production.

For more information, see the following web address:

https://www.ng.bluemix.net/docs/services/monana/index.html#monana_overview



Conclusion

In this book, we reviewed the end-to-end Java development lifecycle and the use of Liberty in a DevOps manner. The following sections summarize the key advantages of Liberty and DevOps that were explored in this publication:

- ▶ Advantages of DevOps
- ▶ Advantages of Liberty

7.1 Advantages of DevOps

DevOps is an organizational capability that is focused on the development and operations teams collaborating and communicating throughout the entire application development lifecycle. Both the development and operations teams are active and equal participants throughout the application lifecycle management (ALM) lifecycle.

DevOps is a journey that differs for every organization, depending on the organization's objectives. DevOps is targeted to improve time to market and reduce feedback loops from customer needs through development.

Different DevOps adoption paths exist, and each path focuses on working with specific gaps and bottlenecks. Continuous deployment is designed around the automation of software delivery across the pipeline. Continuous integration is designed for performing integration testing earlier in the ALM lifecycle (during development as part of the build process). Continuous feedback is designed for monitoring application usage in production and proactively handling customer feedback.

Important: Ensure that the selected DevOps toolset is fully integrated and provides traceability across different ALM lifecycles. The toolset can be flexible in design, and the tools can come from various vendors. The reality for many organizations is that they have a heterogeneous environment from the tooling perspective. However, ensure that selected tools support an integrated pipeline.

In the interest of time and scope, the exercises in this book primarily focused on the following tools:

- ▶ IBM products that were pre-integrated
- ▶ Jenkins, to show that an open source tool can also be part of the DevOps toolset

7.2 Advantages of Liberty

Liberty provides a combination of IBM technology and open source software, which makes it ideal for developers and operations teams. Liberty offers many advantages across the ALM lifecycle.

The following list summarizes the key benefits when you use Liberty in the DevOps manner:

► Planning

Liberty is designed for agile development. Liberty can be used to iteratively and quickly flow changes through the delivery pipeline.

► Development:

- Liberty is a dynamic, composable application server for Java development that provides advantages for both development and operations teams.
- Liberty can be part of the DevOps toolset for Java development shops to use for the development of system of engagement or system of record applications. Its small footprint makes it ideal for developing mobile or Internet of Things (IOT) applications.
- DevOps tools can be on premises or hosted in the cloud. It is also possible for an organization to use a Platform as a Service (PaaS), such as IBM Bluemix, for the development of certain stages or potentially the full lifecycle. Liberty supports all of these deployment models.
- Liberty provides developer tools to use within the Eclipse environment.
- Liberty supports many open source frameworks, such as Spring and Tapestry.

► Building

You can use Git for source control and open source tools, such as Maven or Jenkins, to create a build. You can use a tool, such as Rational Team Concert, as a source code repository and a build engine.

► Deployment:

- A simple XML server configuration makes it easier to track the server configuration and share it from development through operations.
- Server configuration and the application code can be packaged. By packaging the server configuration and application code, you can refresh both application infrastructure and application code easily as part of the DevOps flow.

► Testing

With the lightweight Liberty tools and dynamic run time, the changes flow through the pipeline quickly and allow for a shorter iterative code-build-deploy-test cycle.

► Production:

- Liberty has a new management model with the collective controller. The central management point is the collective controller. No policy enforcement ensures that configuration across the cluster is in sync, which offers greater flexibility for the user and holds the user to a higher level of responsibility. With Liberty, this process can also be automated during deployment.
- If the application server is updated, applications continue working. This design is called “*Zero Migration*”.

► Monitoring

Liberty produces monitoring statistics for use with standard Java Management Extensions (JMX) clients, such as JConsole.



A

Additional material

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

Locating the web material

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at this website:

<ftp://www.redbooks.ibm.com/redbooks/SG248286>

Alternatively, you can go to the IBM Redbooks website:

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

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248286.

Using the web material

The additional web material that accompanies this book contains the following file:

| <i>File name</i> | <i>Description</i> |
|---------------------|---|
| SG248286.zip | Archive file that contains the sample application files of the Liberty Cloud Trader application |

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material SG248286.zip file into this folder.

Download the unit test jar and hamcrest from <http://junit.org> and save it into the jar folder. In this example, the jars that are used are junit-4.12.jar and hamcrest-core-1.3.jar.

Related publications

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

Online resources

These websites are also relevant as further information sources:

- ▶ *DevOps for Dummies*:
<http://ibm.co/20phBQY>
- ▶ Eclipse Mars, Version, Mars Release 4.5.0:
<https://eclipse.org/downloads/>
- ▶ Rational Team Concert Eclipse Extension, Version: 5.0.2:
<https://jazz.net/downloads/rational-team-concert/releases/5.0.2/RTC-Eclipse-Client-Extension-repo-5.0.2.zip>
- ▶ Agile principles:
<http://www.ibm.com/software/rational/agile/resources/>
- ▶ Jenkins:
<https://jenkins-ci.org/>
- ▶ Rational Collaborative Lifecycle Manager:
http://www.ibm.com/support/knowledgecenter/SSYMRC_5.0.2/com.ibm.rational.clm.doc/help/index_clm.htm
- ▶ Ant:
<http://ant.apache.org/>
- ▶ *IBM Rational Integration Tester Training Guide* at the following website:
http://www.slideshare.net/Darrel_Rader/rit-850-integration-testing-training-students-guide
- ▶ UrbanCode Deploy Server 6.1.1:
https://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.install.doc/topics/sysRequire.html?cp=SS4GSP_6.1.1%2F3-0
- ▶ Rational License Key Server 8.1.4:
<http://www.ibm.com/support/docview.wss?uid=swg27038212>
- ▶ Rational Team Server 5.02:
<http://www.ibm.com/software/products/en/rtc#othertab1>
- ▶ IBM DB2 10.5:
<http://www.ibm.com/support/docview.wss?uid=swg27038033>
- ▶ WAS Liberty profile 8.5.5:
<http://www.ibm.com/support/docview.wss?uid=swg27038218>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



SG24-8286-00

ISBN 0738441163

Printed in U.S.A.

Get connected

