MICRO FOCUS

Using
Visual COBOL in Modern
Application Development

Contents

Using Visual COBOL in Modern Application Development

Development has traditionally been one discrete element in the process of turning users' requirements into working applications. Modern application development processes enable you to move away from a process containing a number of (at best) loosely-connected, discrete steps, and instead adopt a process where each step is handled by a specialized tool, and a combination of these tools and automation result in a more seamless, repeatable, end-to-end process that enables you to build and deliver the right software at the right time.

This documentation describes modern application development in general and summarizes the role that Visual COBOL and other tools from Micro Focus can play in modern application development. It also contains information on how you can use Visual COBOL with the Jenkins CI server in order to implement many of the steps described in the modern application development process.

Note: It is important to bear in mind that even though this documentation covers all of the modern application development process, you can gain significant benefits in terms of quality and productivity by adopting any of the parts of the process in isolation.

Related information

Using Visual COBOL with Jenkins

Introduction to Modern Application Development

The following sections show how modern application development differs from traditional development, outline the benefits of the different stages of a modern application development process, and show how Micro Focus tools can play a key role in your development and deployment processes.

What is Modern Application Development?

For many years the majority of software development followed the waterfall development model. Recently, alternatives to the waterfall model have been devised and subsequently revised in an attempt to make the process of software development more able to respond quickly to changing customer needs.

In the last few years, methodologies such as DevOps and DevSecOps have come into being, bringing even more advantages over older methodologies. In general, modern application development methodologies enable you to reduce your time to market, reduce the complexity of your projects, and ensure a high-quality experience for your users.

The information in this section is concerned with modern application methodologies in general rather than with any specific methodologies. When this document refers to modern application development it is referring to any development methodology that aims to be more responsive to customer needs by automating the different parts of the process as well as the transitions between them.

Related reference

Agile Manifesto: Manifesto for Agile Software Development
DevSecOps: What is DevSecOps?
Micro Focus Enterprise DevOps
TechRepublic: Understanding the pros and cons of the Waterfall Model of software development

Key Concepts in Modern Application Development

Any discussion of modern application development methodologies will include reference to a number of key concepts. The following list provides a very brief summary of the key concepts that are used in this documentation.

Agile software development

A set of principles guiding the production of software that focusses on the following:

- Iterative, incremental, evolutionary delivery
- Face-to-face communication
- Short feedback loops
- Use of automation to promote a focus on quality

Application release automation (ARA)

The use of tools to automate the steps involved to build software and subsequently deploy it to production.

Automated testing

The use of tools to control the running of tests and the comparison of the tests' outcomes with their expected outcomes.

Continuous delivery (CD)

A process whereby every code change results in the building and testing of new software that can then be deployed to production (if appropriate).

Continuous deployment

A process whereby every code change results in the building and testing of new software that is then deployed to production.

Continuous improvement

The process of regularly assessing a team's performance in a rollout period, evaluating what has gone well and what can be improved on.

Continuous integration (CI)

The practice of ensuring that all developers' working copies of code are regularly merged into a shared trunk, and each code change results in the building and testing of new software.

Requirements management

The process of gathering and managing the requirements for an application and ensuring that those requirements are used to effectively drive the efforts of the development team.

Software Change and Configuration Management (SCCM)

Distributed tools that implement a set of disciplines used to stabilize, track and control the versions and configurations of a set of artifacts. This also includes development change management, defect tracking, change automation, development release management, integrated test management, integrated build management and other related processes.

Unit testing

The process where the smallest parts of an application that it is possible to test are tested individually to see if they perform as expected.

You might find some of these terms defined in a number of subtly different ways in different sources. This documentation uses the terms as they are defined here.

Steps Involved in Modern Application Development

This section introduces a workflow diagram showing the various steps involved in a modern application development lifecycle then gives introductory information on each of the diagram's steps.

As said elsewhere, this documentation is concerned with modern application development in general rather than any specific named instance of modern application development.

The following figure shows the steps involved in the definition of modern application development used by this documentation.
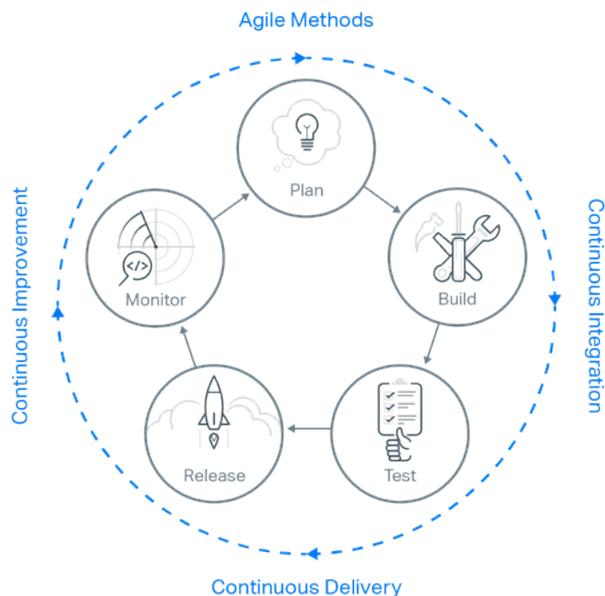
The diagram shows the five main activities that make up the development process:

- Plan
- Build
- Test
- Release
- Monitor

Note that there are no start and end points to this sequence of activities. Whereas one might expect the "Release" activity to mark the end of the process, that activity is followed by the "Monitor" activity where questions are asked about improvements that could be made to the released package as well as to the development process and how it was executed. The information from the "Monitor" activity is then used as input to the "Plan" activity, and the development process continues from there.

Outside the five activities are the broader processes that have evolved to optimize the way in which the individual activities work together. These processes are as follows, and each one is covered in more detail in this documentation:

- Agile methods
- Continuous integration
- Continuous delivery
- Continuous improvement

Related information

Agile Methods
Continuous Integration

Agile Methods

The following sections give an overview of Agile methods, present a typical Agile development workflow, and show the benefits that Micro Focus tools can bring to the Agile development process.

Introduction to Agile Methods

As mentioned in *Key Concepts in Modern Application Development*, agile software development (often referred to as "Agile") is a set of principles guiding the production of software that focusses on:

- Iterative, incremental, and evolutionary delivery
- Face-to-face communication
- Short feedback loops
- Automation to promote a focus on quality

These principles are laid out in a document known as the manifesto for Agile software development.

Agile is intended to be very simple and flexible but it can bring significant benefits such as:

- Increased business agility

  Agile's iterative approach and short feedback loops shorten development cycles and enable you to quickly switch priorities based on changing customer requirements and market conditions.
- Delivering the right product

  An Agile approach facilitates early and frequent feedback, maximizing the likelihood that the software delivered is exactly what the customer needs.

  A combination of the frequent feedback and iterative approach produces another benefit. At the end of an iteration, if stakeholder feedback indicates that the software produced in that iteration is not on target to meet customer requirements, the work done in that iteration can be abandoned and developers can adopt a different approach in the next iteration. The amount of development time lost in such a case would be only one iteration, typically two weeks, which is much less than would be wasted if the same issue arose when following the waterfall model.
- Improved quality

  A key principle of Agile is that testing is integrated throughout the lifecycle.

Related information
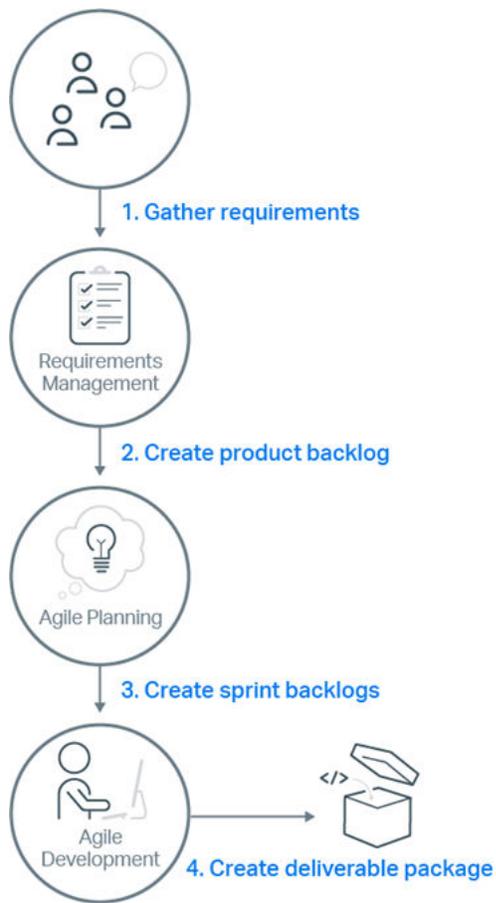
Key Concepts in Modern Application Development

Related reference

Manifesto for Agile Software Development

Agile Development Workflow

The following diagram summarizes the key steps in an Agile development process:

1. Gather requirements

Requirements Management

2. Create product backlog

Agile Planning

3. Create sprint backlogs

Agile Development

4. Create deliverable package

where the numbered steps are as follows:

1. Create a list of requirements. This can include feedback from customers as well as requirements based on your internal needs such as new features, bug fixes, infrastructure and systems work, or technical debt.

2. Use your set of requirements to create a product backlog. The product backlog is a prioritized list of the work that your development team will undertake to add the different items from your customers' and internal requirements.

3. Use the product backlog to create sprint backlogs, which you use to define and track the work that will be done by the development team in a series of sprints. Sprints are also often referred to as "iterations".

   The length of a sprint is typically between one and four weeks and is usually fixed for the duration of a project.

   During each sprint, developers take items from the sprint backlog, work on them, and complete them by the end of the sprint.

   At the end of each sprint, the items on that sprint's backlog must be completed; not just coded, but tested, documented, and integrated into a working product that could be deployed (if required).

4. Produce a deliverable product package, if required.

   At the end of a sprint you return to the Agile planning stage to create the next sprint backlog.

   At the end of the project you return to the requirements gathering stage to create the new product backlog.

Agile Development and Micro Focus Development Tools

The sections *Introduction to Agile Methods* and *Agile Development Workflow* introduce the idea of Agile software development and summarize how Agile development works as a process. This section looks at the Agile development process and shows how different products available from Micro Focus fit into and add value to that process.

The diagram below shows the process presented in the topic *Agile Development Workflow* but has been updated to indicate which Micro Focus products are appropriate at different parts of the process. Although this diagram refers to Micro Focus products, the process described does not require the use of Micro Focus products, so if you are already using a third-party product for one part of the process you can continue to work with that and use Micro Focus products to integrate with it.
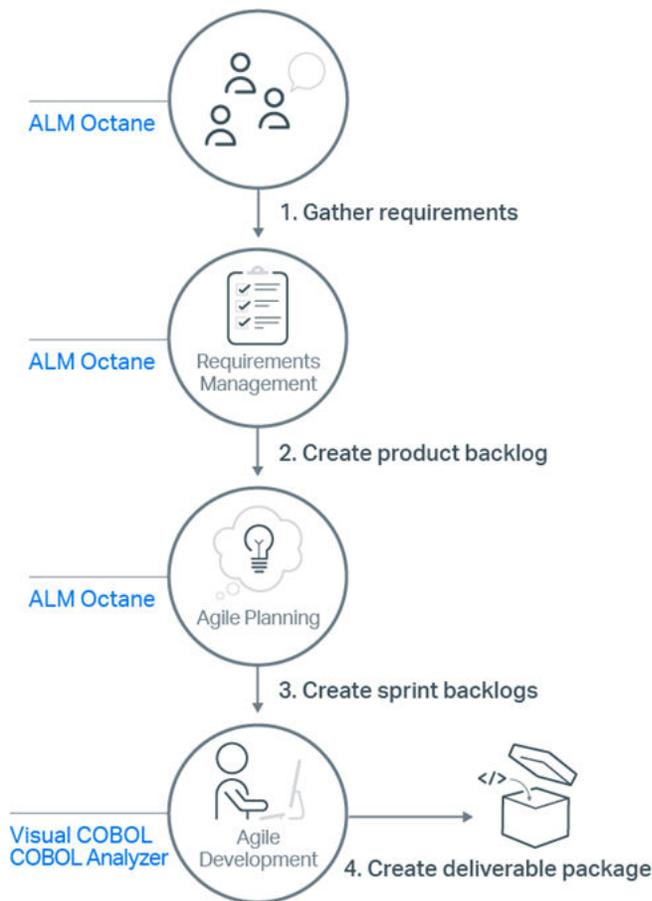
where the numbered steps are as follows:

1. Use Micro Focus ALM Octane for requirements management.

   The Requirements module provides you with a central repository for documenting and tracking all aspects of your project, from conception to delivery. This can include business goals, customer requests, functional requirements, or any other requirements whose approval and progress you want to track.

2. Use Micro Focus ALM Octane to create a product backlog comprising epics, features and stories.

   The product backlog provides you the ability to rank development items and plan development cycles, and the hierarchy of epics, features and stories describes the work required to complete your project. You can also push requirements from the Requirements module into the backlog to be ranked and prioritized against other work.

3. Use Micro Focus ALM Octane to create sprints and releases, which you use to define and track the work on the product backlog that will be done by the development team.

   During each sprint, developers take items from the backlog, work on them, and complete them by the end of the sprint. At the end of each sprint, the items on that sprint's backlog must be completed; not just coded, but tested, documented, and integrated into a working product that could be deployed (if required).

4. Use Visual COBOL to produce a deliverable product package, if required.

   When using Visual COBOL, developers can use the complete array of analysis, intelligence and reporting tools provided by Enterprise Analyzer to quickly gain a full understanding of the applications they are working on. Integration between ALM Octane and Visual COBOL means developers can easily work on code changes and keep ALM Octane's status information up to date at the same time.

   At the end of a sprint you return to the Agile planning stage to create the next sprint backlog. At the end of the project you return to the requirements gathering stage to create the new product backlog.

The following list gives a very brief summary of each of the Micro Focus products that play a part in the Agile development process:

- ALM Octane (Backlog and Team Backlog modules)

  Micro Focus ALM Octane is the enterprise-class planning and tracking solution for Agile software projects.

  ALM Octane is a web-based application lifecycle management platform that enables teams to collaborate easily, manage the product delivery pipeline, and visualize the impact of changes.

  ALM Octane has been designed to integrate easily with Visual COBOL, as well as with third-party tools within the DevOps toolchain.

- ALM Octane (Requirements module)

  Micro Focus ALM Octane enables teams to create requirements in a much more collaborative and flexible way in comparison to other requirements management tools.

  Requirements can be high-level descriptions or formal documentation for your release, depending on your development methodology.

- COBOL Analyzer

  Micro Focus COBOL Analyzer is powerful code analysis and visualization toolset, designed to address the challenges of working with large-scale, complex applications.

  COBOL Analyzer enables you to quickly gain a thorough understanding of your applications, meaning that you reduce the amount of time it takes you to make your changes and you can have more confidence that your changes have the desired effect and do not introduce any new issues.

  You can also use COBOL Analyzer to run queries to determine if your code conforms to your in-house standards. Any code that does not conform to your standards can be flagged as an error following a commit or during the build process.

- Visual COBOL

  Micro Focus Visual COBOL is the next-generation solution for COBOL application development and deployment. It enables you to modernize COBOL systems using Visual Studio and Eclipse as well as deploy COBOL applications and services to new platforms, including .NET, JVM, and the cloud.

  The following features of Visual COBOL make it an invaluable part of the Agile development process:

  - Integration with SCC-compliant source code control systems to enable you to work seamlessly with your source code.
  - Integration into Visual Studio provides useful debugging features such as colorization, IntelliSense, error flagging, and intelligent copybook handling to enable you to quickly track down any issues, establish their cause, and make your edits.

- Reverse Debug and Live Recording are features available on Red Hat Linux x86 platforms that enable you to create a recording of an application's execution then load the recording into the debugger.

  With the recording loaded into the debugger you can monitor everything that influenced the running of the program (such as all input, disk access, and keyboard strokes) and because the debugger lets you move backwards and forwards through the execution path you can easily focus on potential causes of crashes or other unexpected behavior in the application.
- The Micro Focus Unit Testing Framework, an xUnit-style testing framework, includes much of the architecture you would expect of an xUnit framework, enabling you to create, compile, run, and debug unit tests from either the command line or the Visual COBOL IDE.
- Core dump debugging. When an application crashes you can arrange for its state to be saved to disk, in a core dump file, which can indicate where the error occurred in the source code, the contents of memory at the time of the error, and the values of any variables and expressions set at the time. You can then use the core dump file to help debug the problems.
- Remote debugging enables you to debug programs that are running on a different computer from the one on which you are using.
- The Consolidated Tracing Facility (CTF) produces detailed diagnostic information that can be invaluable in diagnosing problems when you can't easily attach a debugger.

Related reference

Micro Focus: ALM Octane data sheet
Micro Focus: ALM Octane online help
Micro Focus: COBOL Analyzer data sheet
Micro Focus: COBOL Analyzer online help
Micro Focus: Visual COBOL data sheet
Micro Focus: Visual COBOL online help

Continuous Integration

The following sections give an overview of continuous integration, present a typical continuous integration workflow, and show the benefits that Micro Focus tools can bring to the continuous integration process.

Introduction to Continuous Integration

As mentioned in *Key Concepts in Modern Application Development*, continuous integration (CI) is a software development practice whereby developers on a team regularly integrate their working copies of code to a shared repository. Once a change is integrated to the repository, the application is automatically rebuilt. Automated tests are run before and after the application is rebuilt, to check that no regressions are introduced. If any of the automated tests fail, developers can be notified automatically so that they can provide a fix.

Using continuous integration provides a number of benefits:

- Developers can find problems earlier than they would be able to if builds happened only every day or every week.
- When a problem does occur, pinpointing the cause of the problem is quicker and easier because only a small change should have been made since the last working build.
- The time taken to resolve integration issues is reduced because each change is integrated as required.
- There is always a working version of the product that contains the latest changes.

Tools that provide CI functionality are known as CI servers. There are many tools available that can be used as CI servers, but the following list shows some of the more commonly used:

- Bamboo
- Cruise Control
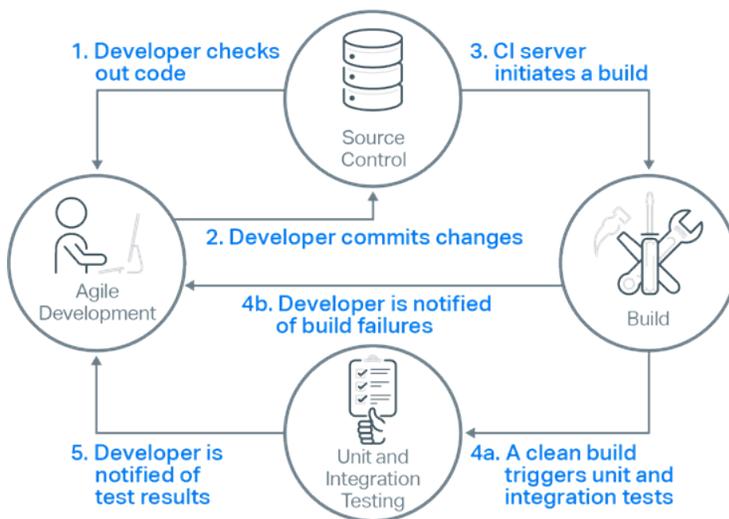- Hudson
- Jenkins
- Team Foundation Server

For detailed information on how to use Visual COBOL with Jenkins, see *Using Visual COBOL with Jenkins*.

Related information

Key Concepts in Modern Application Development
Using Visual COBOL with Jenkins

Continuous Integration Workflow

The following diagram summarizes the key steps in a continuous integration process. Although the process comprises a number of different activities with integration provided between each activity, you do not have to adopt every activity in the process before you can benefit from significant gains in terms of efficiency, effectiveness, and quality. Instead, you might choose to implement this process one activity at a time, taking advantage of the benefits provided by each activity as you add and integrate it with the others.

where the numbered steps are as follows:

1. Developers check out code into their private workspaces. They then make their changes and test them locally.
2. When done, developers check in their changes into the source control repository.
3. The CI server monitors the source control repository and when it detects a change it triggers a build of the relevant sources.
4. After a successful build, the CI server performs some or all of the following activities:

   - makes deployable artefacts available for testing
   - assigns a build label to the version of the code that was just built
   - notifies the relevant team members that a successful build occurred
   - triggers unit and integration testing

   At this point, the changes that were checked in at step 2 have been successfully built and a build label has been applied to the source code that was used for the build, meaning that the build could be recreated if necessary.

In the event of a build failure, the CI server sends notifications to the relevant developers who restart the process from step 1 to make the changes necessary to resolve the build errors.

5. After the unit and integration testing has taken place, the relevant team members are notified of the test results.

    At this point, the changes that were checked in at step 2 have been successfully built and tested, all with little or no manual intervention.

For information on using Jenkins to perform the CI server tasks in the above list, see *Using Visual COBOL with Jenkins*.
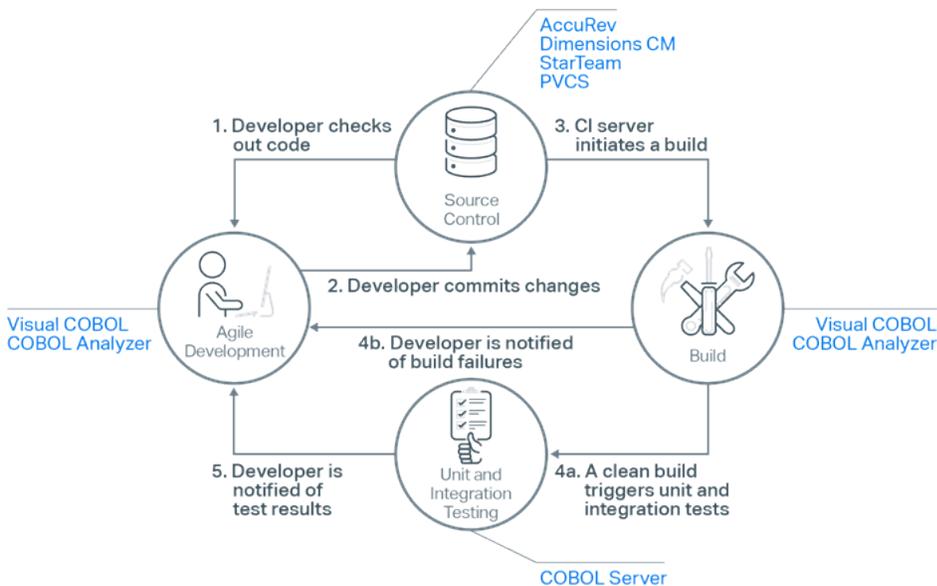
Related information

Using Visual COBOL with Jenkins

Continuous Integration and Micro Focus Development Tools

The sections *Introduction to Continuous Integration* and *Continuous Integration Workflow* introduce the idea of continuous integration and summarize how continuous integration works as a process. This section looks at the continuous integration process and shows how different products available from Micro Focus fit into and add value to that process.

The diagram below shows the process presented in the topic *Continuous Integration Workflow* but has been modified to indicate which Micro Focus products you can use at the different parts of the process. Although this diagram refers to Micro Focus products, the process described does not require the use of Micro Focus products, so if you are already using a third-party product for one part of the process you can continue to work with that and use Micro Focus products to integrate with it.

where the numbered steps are as follows:

1. Developers use Visual COBOL to check out code into their private workspaces. They then make their changes and test them locally using Visual COBOL's unit testing features.

    This diagram illustrates the use of AccuRev, Dimensions CM, StarTeam or PVCS as the source code control system but you are not limited to using only those products. Visual COBOL works with any SCC-compliant source code control system, so you can work seamlessly in Visual COBOL with virtually any source code control system you choose to use regardless of whether it is a Micro Focus product or a third-party product.

2. When done, developers check in their changes into the source control repository.
3. The CI server monitors the source control repository and when it detects a change it triggers a build of the relevant sources. Although the build actions are triggered by the CI server, the build actions themselves will be performed by Visual COBOL, typically using Apache Ant or MSBuild scripts.
4. After a successful build, the CI server performs activities such as the following:

   - makes deployable artefacts available for testing
   - assigns a build label to the version of the code that was just built
   - notifies the relevant team members that a successful build occurred
   - triggers unit and integration testing to be run under COBOL Server

   At this point, the changes that were checked in at step 2 have been successfully built and a build label has been applied to the source code that was used for the build (so the build could be recreated if necessary).

   In the event of a build failure, the CI server sends notifications to the relevant developers who restart the process from step 1, using Visual COBOL to make the changes necessary to resolve the build errors.
5. After the unit and integration testing has taken place, the relevant team members are notified of the test results.

   At this point, the changes that were checked in at step 2 have been successfully built and tested, all with little or no manual intervention.

For information on using Jenkins to perform the CI server tasks in the above list, see *Using Visual COBOL with Jenkins*.

The following list gives a very brief summary of each of the Micro Focus products that play a part in the continuous integration process:

- AccuRev

  Micro Focus AccuRev is a software configuration management tool that addresses complex parallel and distributed development environments with stream-based architecture to accelerate development processes and improve asset reuse.

  AccuRev integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.
- COBOL Analyzer

  Micro Focus COBOL Analyzer is powerful code analysis and visualization toolset, designed to address the challenges of working with large-scale, complex applications.

  COBOL Analyzer enables you to quickly gain a thorough understanding of your applications, meaning that you reduce the amount of time it takes you to make your changes and you can have more confidence that your changes have the desired effect and do not introduce any new issues.

  You can also use COBOL Analyzer to run queries to determine if your code conforms to your in-house standards. Any code that does not conform to your standards can be flagged as an error following a commit or during the build process.
- COBOL Server

  COBOL Server is the deployment and execution environment for applications developed using Visual COBOL. It provides a high-performance, platform-portable run time environment in which your customers can execute your COBOL applications, while its small footprint and ease of installation makes it easy for you to use in your testing.

  As well as providing the environment in which your COBOL applications run, COBOL Server includes features to simplify your testing. For example, once you have set up a COBOL Server environment for testing an application you can export the definition of that environment to an XML file, where the XML definition includes details of all aspects of the COBOL Server environment such as region definitions, locations of data files, and settings of environment variables. Once you have exported the definition you can import it to be used during your testing, ensuring that the COBOL Server environment you use in your testing is exactly the same environment as the one you know to be correct.

- Dimensions CM

    Micro Focus Dimensions CM streamlines the complexity of collaborative parallel development and increases team velocity while ensuring a high degree of release readiness.

    Dimensions CM integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.
- PVCS

    Micro Focus PVCS Version Manager is used by thousands of software developers around the world to meet their version control requirements. It is one of the most reliable, trusted, and proven solutions available.

    PVCS integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.
- StarTeam

    Micro Focus StarTeam delivers changes across multiple ALM repositories and tools as the single source of truth. It's an enterprise change management system, serving both centralized and geographically distributed development teams, helping them achieve their highest level of software delivery.

    StarTeam integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.
- Visual COBOL

    Micro Focus Visual COBOL is the next-generation solution for COBOL application development and deployment. It enables you to modernize COBOL systems using Visual Studio and Eclipse as well as deploy COBOL applications and services to new platforms, including .NET, JVM, and the cloud.

    Visual COBOL offers the following features that make it a great fit for using in your CI process:
    - Support for MSBuild means that you can write scripts to enable your CI server to build and run your COBOL applications just as easily as you can build and run your COBOL applications from the Visual COBOL IDE.
    - Integration into an extensible IDE enables you to use a range of third-party functionality to work with different CI servers.
    - Integration with SCC-compliant source code control systems to enable you to work seamlessly with your source code.
    - Integration into Visual Studio provides useful debugging features such as colorization, IntelliSense, error flagging, and intelligent copybook handling to enable you to quickly track down any issues, establish their cause, and make your edits.
    - Reverse Debug and Live Recording are features available on Red Hat Linux x86 platforms that enable you to create a recording of an application's execution then load the recording into the debugger.

        With the recording loaded into the debugger you can monitor everything that influenced the running of the program (such as all input, disk access, and keyboard strokes) and because the debugger lets you move backwards and forwards through the execution path you can easily focus on potential causes of crashes or other unexpected behavior in the application.
    - The Micro Focus Unit Testing Framework, an xUnit-style testing framework, includes much of the architecture you would expect of an xUnit framework, enabling you to create, compile, run, and debug unit tests from either the command line or the Visual COBOL IDE.
    - Core dump debugging. When an application crashes you can arrange for its state to be saved to disk, in a core dump file, which can indicate where the error occurred in the source code, the contents of memory at the time of the error, and the values of any variables and expressions set at the time. You can then use the core dump file to help debug the problems.
    - Remote debugging enables you to debug programs that are running on a different computer from the one on which you are using.
    - The Consolidated Tracing Facility (CTF) produces detailed diagnostic information that can be invaluable in diagnosing problems when you can't easily attach a debugger.

Related information

Using Visual COBOL with Jenkins

Related reference

Micro Focus: AccuRev data sheet
Micro Focus: COBOL Analyzer data sheet
Micro Focus: COBOL Analyzer online help
Micro Focus: COBOL Server data sheet
Micro Focus: Dimensions CM data sheet
Micro Focus: PVCS home page
Micro Focus: StarTeam data sheet
Micro Focus: Visual COBOL data sheet
Micro Focus: Visual COBOL online help

Continuous Delivery

The following sections give an overview of continuous delivery, present a typical continuous delivery workflow, and show the benefits that Micro Focus tools can bring to the continuous delivery process.

Introduction to Continuous Delivery

As mentioned in *Key Concepts in Modern Application Development*, continuous delivery (CD) is a practice where teams make use of automated processes so that any code changes result in the building and testing of new software that can then be deployed to production (if appropriate).

Continuous delivery does not mean that every code change is deployed to production. Instead, a continuous delivery environment typically includes a number of different environments for deployment, where deployments to some environments happen automatically but deployment to other environments requires some manual input or approval.

For example, a continuous delivery environment could include deployment environments for development, testing, staging, and production, where deployment happens automatically for the development and testing environments, but manual intervention is required to approve deployment to the staging or production environments.

Using continuous delivery in this way enables you to deliver application changes quickly and more reliably, leading to improved product quality and user experience.

> **Note:** Continuous delivery is very similar to continuous deployment, with the only difference being the existence (or lack) of manual validation steps. With continuous delivery a code change results in automated building and testing followed by at least one manual validation step before the changes are deployed to production, whereas with continuous deployment there are no manual validation steps, so a code change always results in automated building, testing, and deployment to production.

Related information
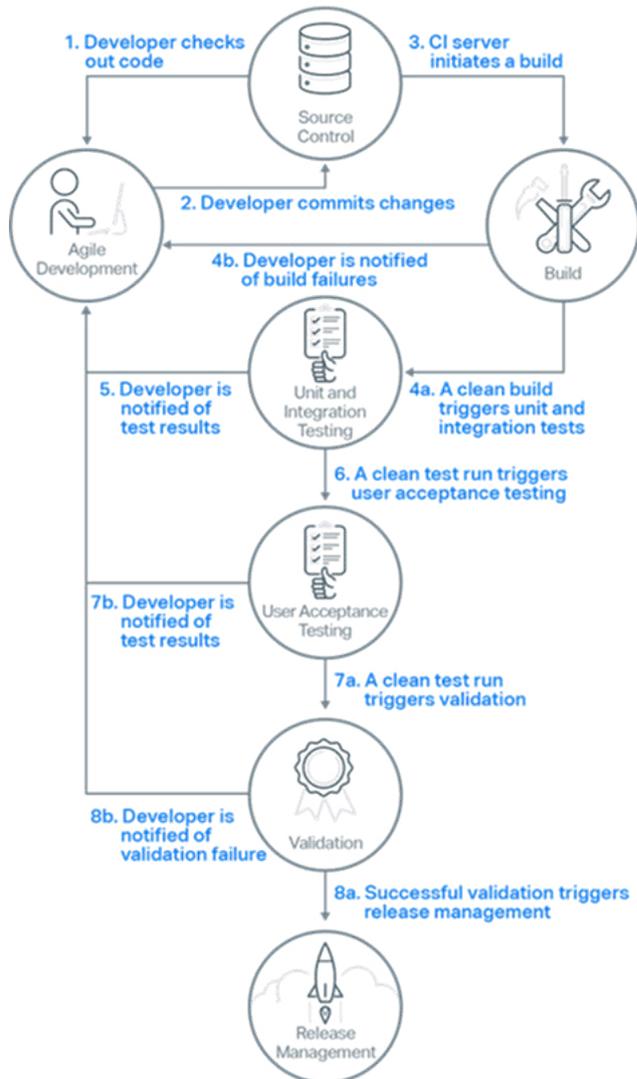
Key Concepts in Modern Application Development

Related reference

Continuous Delivery: What is Continuous Delivery?

Continuous Delivery Workflow

The following diagram summarizes the steps involved in a continuous delivery process. Although the process comprises a number of different activities with integration provided between each activity, you do not have to adopt every activity in the process before you can benefit from significant gains in terms of efficiency, effectiveness, and quality. Instead, you might choose to implement this process one activity at a time, taking advantage of the benefits provided by each activity as you add and integrate it with the others.

Note that because continuous delivery is effectively an extension of the continuous integration process, the first five steps in this diagram are the same as the steps in the diagram presented in *Continuous Integration Workflow*.

where the numbered steps are as follows:

1. Developers check out code into their private workspaces. They then make their changes and test them locally.
2. When done, developers check in their changes into the source control repository.
3. The CI server monitors the source control repository and when it detects a change it triggers a build of the relevant sources.
4. After a successful build, the CI server performs some or all of the following activities:

- makes deployable artefacts available for testing
- assigns a build label to the version of the code that was just built
- notifies the relevant team members that a successful build occurred
- triggers unit and integration testing

At this point, the changes that were checked in at step 2 have been successfully built and a build label has been applied to the source code that was used for the build, meaning that the build could be recreated if necessary.

In the event of a build failure, the CI server sends notifications to the relevant developers who restart the process from step 1 to make the changes necessary to resolve the build errors.

5. After the unit and integration testing has taken place, the relevant team members are notified of the test results.

At this point, the changes that were checked in at step 2 have been successfully built and tested, all with little or no manual intervention.

6. After the unit and integration testing has completed successfully, the CI server triggers the running of more comprehensive, automated acceptance tests.

7. If the acceptance tests all pass, a decision is made whether or not to release.

If the acceptance tests result in failures, the CI server sends notifications to the relevant developers who restart the process from step 1 to make the changes necessary to resolve the test failures.

8. If the validation decision is to release, release management is triggered. This results in the built package being released or deployed to the appropriate environment.

If the validation decision is not to release, the relevant team members are notified and development work continues as normal.

For information on using Jenkins to perform the CI server tasks in the above list, see *Using Visual COBOL with Jenkins*.

Related information

Continuous Integration Workflow
Using Visual COBOL with Jenkins

Continuous Delivery and Micro Focus Development Tools

The sections *Introduction to Continuous Delivery* and *Continuous Delivery Workflow* introduce the idea of continuous delivery and summarize how continuous delivery works as a process. This section looks at the continuous delivery process and shows how different products available from Micro Focus fit into and add value to that process.

The diagram below shows the process presented in the section *Continuous Delivery Workflow* but has been updated to indicate which Micro Focus products are appropriate at different parts of the process. Although this diagram refers to Micro Focus products, the process described does not require the use of Micro Focus products, so if you are already using a third-party product for one part of the process you can continue to work with that and use Micro Focus products to integrate with it.

Note that because continuous delivery is effectively an extension of the continuous integration process, the first five steps in this diagram are the same as the steps in the diagram presented in *Continuous Integration and Micro Focus Development Tools*.

where the numbered steps are as follows:

1. Developers use Visual COBOL to check out code into their private workspaces. They then make their changes and test them locally using Visual COBOL's unit testing features.

   This diagram illustrates the use of AccuRev, Dimensions CM, StarTeam or PVCS as the source code control system but you are not limited to using only those products. Visual COBOL works with any SCC-compliant source code control system, so you can work seamlessly in Visual COBOL with virtually any source code control system you choose to use regardless of whether it is a Micro Focus product or a third-party product.

2. When done, developers check in their changes into the source control repository.

3. The CI server monitors the source control repository and when it detects a change it triggers a build of the relevant sources. Although the build actions are triggered by the CI server, the build actions themselves will be performed by Visual COBOL, typically using Apache Ant or MSBuild scripts.

4. After a successful build, the CI server performs activities such as the following:

   - makes deployable artefacts available for testing
   - assigns a build label to the version of the code that was just built
   - notifies the relevant team members that a successful build occurred

- triggers unit and integration testing to be run under COBOL Server

   At this point, the changes that were checked in at step 2 have been successfully built and a build label has been applied to the source code that was used for the build (so the build could be recreated if necessary).

   In the event of a build failure, the CI server sends notifications to the relevant developers who restart the process from step 1, using Visual COBOL to make the changes necessary to resolve the build errors.

5. After the unit and integration testing has taken place, the relevant team members are notified of the test results.

   At this point, the changes that were checked in at step 2 have been successfully built and tested, all with little or no manual intervention.

6. After the unit and integration testing has completed successfully, the CI server triggers the running of more comprehensive, automated acceptance tests using Micro Focus Silk.

7. If the acceptance tests all pass, a decision is made whether or not to release.

   If the acceptance tests results in failures, the CI server sends notifications to the relevant developers who restart the process from step 1 to make the changes necessary to resolve the test failures.

8. If the validation decision is to release, use Micro Focus Deployment Automation or Micro Focus Release Control to release or deploy the built package to the appropriate environment.

   If the validation decision is not to release, the relevant team members are notified and development work continues as normal.

For information on using Jenkins to perform the CI server tasks in the above list, see *Using Visual COBOL with Jenkins*.

The following list gives a very brief summary of each of the Micro Focus products that play a part in the continuous delivery process:

- AccuRev

   Micro Focus AccuRev is a software configuration management tool that addresses complex parallel and distributed development environments with stream-based architecture to accelerate development processes and improve asset reuse.

   AccuRev integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.

- COBOL Analyzer

   Micro Focus COBOL Analyzer is powerful code analysis and visualization toolset, designed to address the challenges of working with large-scale, complex applications.

   COBOL Analyzer enables you to quickly gain a thorough understanding of your applications, meaning that you reduce the amount of time it takes you to make your changes and you can have more confidence that your changes have the desired effect and do not introduce any new issues.

   You can also use COBOL Analyzer to run queries to determine if your code conforms to your in-house standards. Any code that does not conform to your standards can be flagged as an error following a commit or during the build process.

- COBOL Server

   COBOL Server is the deployment and execution environment for applications developed using Visual COBOL. It provides a high-performance, platform-portable run time environment in which your customers can execute your COBOL applications, while its small footprint and ease of installation makes it easy for you to use in your testing.

   As well as providing the environment in which your COBOL applications run, COBOL Server includes features to simplify your testing. For example, once you have set up a COBOL Server environment for testing an application you can export the definition of that environment to an XML file, where the XML definition includes details of all aspects of the COBOL Server environment such as region definitions, locations of data files, and settings of environment variables. Once you have exported the definition you

can import it to be used during your testing, ensuring that the COBOL Server environment you use in your testing is exactly the same environment as the one you know to be correct.

- Deployment Automation

  Micro Focus Deployment Automation simplifies and automates the deployment of your software. It supports continuous delivery and production deployments by seamlessly enabling deployment pipeline automation, reducing cycle times, and providing rapid feedback. With Deployment Automation, you will be able to deliver high-quality, valuable software in an efficient, fast, and reliable manner.

- Dimensions CM

  Micro Focus Dimensions CM streamlines the complexity of collaborative parallel development and increases team velocity while ensuring a high degree of release readiness.

  Dimensions CM integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.

- PVCS

  Micro Focus PVCS Version Manager is used by thousands of software developers around the world to meet their version control requirements. It is one of the most reliable, trusted, and proven solutions available.

  PVCS integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.

- Release Control

  Micro Focus Release Control enables you to plan, control, and automate your release processes from definition to deployment with a visual release calendar and automated approval process. The automation and integration features that Release Control offers result in improved visibility and tracking, and adherence to your compliance and control procedures.

- Silk

  Micro Focus Silk Test enables you to maintain rigorous quality standards and accelerate application testing on any device and platform. Using Silk Test you can standardize validation efforts by testing web, mobile, rich-client, and enterprise applications with a single, powerful test automation solution.

  Micro Focus Silk Central unifies test assets into one planning, tracking, reporting, and execution hub, enabling you to define quality goals, schedule manual and automated functional and performance tests, and view results in a centralized dashboard.

- StarTeam

  Micro Focus StarTeam delivers changes across multiple ALM repositories and tools as the single source of truth. It's an enterprise change management system, serving both centralized and geographically distributed development teams, helping them achieve their highest level of software delivery.

  StarTeam integrates with Visual COBOL to enable you to commit your changes to the shared repository quickly and easily with a minimum of fuss.

- Visual COBOL

  Micro Focus Visual COBOL is the next-generation solution for COBOL application development and deployment. It enables you to modernize COBOL systems using Visual Studio and Eclipse as well as deploy COBOL applications and services to new platforms, including .NET, JVM, and the cloud.

  Visual COBOL includes the following features that make it particularly suitable for using in your continuous delivery process:

  - Support for MSBuild means that you can write scripts to enable your CI server to build and run your COBOL applications just as easily as you can build and run your COBOL applications from the Visual COBOL IDE.
  - Integration into an extensible IDE enables you to use a range of third-party functionality to work with different CI servers.

- Integration with SCC-compliant source code control systems to enable you to work seamlessly with your source code.
- Integration into Visual Studio provides useful debugging features such as colorization, IntelliSense, error flagging, and intelligent copybook handling to enable you to quickly track down any issues, establish their cause, and make your edits.
- Reverse Debug and Live Recording are features available on Red Hat Linux x86 platforms that enable you to create a recording of an application's execution then load the recording into the debugger.

  With the recording loaded into the debugger you can monitor everything that influenced the running of the program (such as all input, disk access, and keyboard strokes) and because the debugger lets you move backwards and forwards through the execution path you can easily focus on potential causes of crashes or other unexpected behavior in the application.
- The Micro Focus Unit Testing Framework, an xUnit-style testing framework, includes much of the architecture you would expect of an xUnit framework, enabling you to create, compile, run, and debug unit tests from either the command line or the Visual COBOL IDE.
- Core dump debugging. When an application crashes you can arrange for its state to be saved to disk, in a core dump file, which can indicate where the error occurred in the source code, the contents of memory at the time of the error, and the values of any variables and expressions set at the time. You can then use the core dump file to help debug the problems.
- Remote debugging enables you to debug programs that are running on a different computer from the one on which you are using.
- The Consolidated Tracing Facility (CTF) produces detailed diagnostic information that can be invaluable in diagnosing problems when you can't easily attach a debugger.

Related information

Continuous Integration and Micro Focus Development Tools
Using Visual COBOL with Jenkins

Related reference

Micro Focus: AccuRev data sheet
Micro Focus: COBOL Analyzer data sheet
Micro Focus: COBOL Analyzer online help
Micro Focus: COBOL Server data sheet
Micro Focus: Deployment Automation data sheet
Micro Focus: Dimensions CM data sheet
Micro Focus: PVCS home page
Micro Focus: Release Control data sheet
Micro Focus: Silk Central data sheet
Micro Focus: Silk Test data sheet
Micro Focus: StarTeam data sheet
Micro Focus: Visual COBOL data sheet
Micro Focus: Visual COBOL online help

Continuous Improvement

Continuous improvement is strongly linked to one of the principles of the Agile manifesto which states that "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly". The improvements you want to achieve can be split into two distinct areas:

- Improvements to the software being produced.

  The following sources of information could all be used to help you target improvements to your software:

- Test failures. If a particular area of code regularly causes test failures, is there the potential to redesign that section of the code? Test failures would be flagged by the unit and integration testing carried out in the continuous integration and continuous delivery processes of the modern application development process as well as by the acceptance testing carried out in the continuous delivery process.
- Code coverage. The code coverage capabilities of Visual COBOL will highlight any areas of your code that aren't being executed as part of your testing, giving you the opportunity to update your tests to ensure that every line of code is thoroughly tested.
- Customer feedback. Any feedback that you get from your customers about your software can be fed into the requirements gathering step of the Agile development process, giving you the opportunity to assess the feedback and quickly make changes based on it if appropriate. The rapid iterative nature of Agile development ensures that any changes you do make will quickly be delivered to the customer.

- Improvements to the process used to produce the software.

A convenient opportunity for a team to have this discussion is the sprint retrospective that is a part of the Agile development process. There is no fixed format to determine exactly what is discussed at a sprint retrospective, but in general, variants of the following questions are asked:

- What went well?
- What didn't go well?

The scope of these questions is not limited to code-writing activities but covers the whole development process, so from requirements gathering to release. For any aspects of operation that are deemed to have not gone well, solutions can be discussed, and any changes can be implemented for future sprints. If substantial changes are required, they can be handled in the same way as any other requirements, with a tool such as Micro Focus ALM Octane.

As sprints are very short, typically two weeks, any changes that are made to the process will be put into practice very quickly, and future sprint retrospectives will provide opportunities to discuss if the changes made had the desired effect or if further changes need to be made.

Related information

Agile Manifesto: Principles Behind the Agile Manifesto

Using Visual COBOL with Jenkins

This section of the documentation contains information that you need to know to in order to use Visual COBOL 3.0 in conjunction with Jenkins version 2.60.1. It includes information on what software you need to install, how to configure Visual COBOL and Jenkins to get the most from them working together, and tips and guidance for using Visual COBOL alongside Jenkins.

General information about using Visual COBOL in a modern application development process is included in *Introduction to Modern Application Development*.

> ✏️ **Note:**
>
> - The examples in this section of the documentation have been tested with the specified versions of Visual COBOL and Jenkins. The concepts described would still apply to earlier or later releases of the two products, however, some specific releases might require changes to the Jenkins configuration.
> - For full details about how to use Jenkins, see the Jenkins user documentation.

Related information

Introduction to Modern Application Development

Related reference

Jenkins user documentation

Overview of Jenkins

Jenkins is a continuous integration (CI) server that supports a wide range of tools and technologies. Adopting a CI process ensures that all developers' working copies of code are regularly merged into a shared trunk. Once a change is committed to the repository, the product is automatically rebuilt and tested.

With Jenkins you can automate a number of day-to-day tasks such as checking out the sources from source control, building, code analysis, and different levels of testing and deployment.

By setting up Jenkins to run these tasks each time a developer has changed the source code, you can detect any defects much faster meaning that you maintain your applications' quality and reduce time to market.

Jenkins is highly configurable, and there are numerous plugins available that provide access to a range of tools including source control, shell and batch scripts. Jenkins works on multiple platforms, supports Java and integrates with other corporate systems.

Using Projects and Pipelines

Jenkins supports two methods of specifying commands - simple projects and pipelines.

Earlier versions of Jenkins only provided projects (previously known as jobs) for managing your tasks. Projects need to be created and configured manually in the UI and, if you need to create multiple projects, could require more maintenance work. Configuring projects is independent from your code.

More recent versions of Jenkins also support pipelines where all build, test, analysis and deployment tasks are stored in a single pipeline and saved as a Jenkinsfile. Micro Focus recommends using pipelines as they are suitable for organizing complex activities running on multiple machines. Storing a pipeline in a file format also means you can follow common CI best practices and save the pipeline in your source control code system with the rest of your code as another artefact.

For simplicity, the examples in this guide are created using projects and not pipelines. For details about the pipeline syntax, see the Jenkins user documentation.

Related reference

Jenkins user documentation

Terminology

This documentation uses a number of terms as defined in the Jenkins UI. For a full list of the Jenkins terms, see the Jenkins user documentation.

Agent   A machine which connects to a Jenkins master and executes tasks when directed by the master.[1]

Job   A deprecated term, synonymous with project.

Master   The central, coordinating machine which stores configuration, loads plugins, and renders the various user interfaces for Jenkins.[2]

Node   A machine which is part of the Jenkins environment and capable of executing pipelines or projects. Both the master and agents are considered to be nodes.

Pipeline   A user-defined model of a continuous delivery pipeline.[3]

Plugin   An extension that provides additional functionality that is not provided by standard Jenkins.

Project   A user-configured description of work which Jenkins should perform, such as building a piece of software.

Related reference

Jenkins user documentation

Scenarios for Using Jenkins with Visual COBOL

You can integrate Jenkins into your continuous integration environment in a variety of ways. The approach that you decide to implement depends on your particular development process.

Using a Master and Agent Machines

The simplest scenario is to install Jenkins on the same machine on which you build and test your source code.

Real-world development and delivery processes typically require a more complex scenario than this, however, and you often need to ensure that your applications operate as expected on a variety of platforms. You can use Jenkins to its full potential in a multiple machine environment where Jenkins is installed on one machine (master) and controls a number of tasks that execute on various other machines (agents).

---

[1] Agent machines are also referred to as Slaves.

[2] Typically, this is the machine that has Jenkins installed. The actual work specified in a project would then be performed on one or more agent machines (slaves).

[3] The examples in this documentation are created using projects and not pipelines.

Automating Your Processes

You can use Jenkins to automatically start the next task in your application development process when the previous one has completed successfully. For example, this is how you can create a sequence of projects:

1. Your projects examine the source code control system and trigger a checkout and build after a developer commits a change.
2. At the end of a successful build, Jenkins can trigger other projects such as ones that run the MFUnit tests or copy the executables to another location for additional testing or deployment.

Software Requirements

The examples in this documentation are created with the software versions specified below:

- Jenkins 2.60.1 - see the Jenkins user documentation for information about the software prerequisites. In a scenario where you have a master and a number of agent machines, Jenkins must be installed on the master machine. You can obtain Jenkins from the Jenkins download page.
- Visual COBOL 3.0 for Visual Studio - see the product's release notes for any software prerequisites.

  In a scenario where you have a master and a number of agent machines, Visual COBOL must be installed on each agent machine. You can obtain Visual COBOL from Micro Focus SupportLine.
- Apache Ant 1.9.4 - this is required if you are working with COBOL applications created with Visual COBOL for Eclipse. Download Ant from the Apache Ant Web site.

The rest of this documentation assumes that you have installed and licensed the specified software as required.

Related reference

Apache Ant Web site
Jenkins download page
Jenkins user documentation
Micro Focus SupportLine
Visual COBOL 3.0 release notes

Installing and Configuring Jenkins

The following is a brief overview of how to install and configure Jenkins and how to create a project.

Installing and Accessing Jenkins

Jenkins is supported on a variety of platforms. See the Jenkins user documentation for detailed instructions on how to install and start it on the platform that you will be using.

You can access Jenkins from any machine using `http://<mymachine>:8080`, where `<mymachine>` is the name of the machine where Jenkins is installed.

Configuring Jenkins

You can specify general Jenkins options by clicking **Manage Jenkins** in the Home page of the Jenkins interface.

Use plugins to extend Jenkins and to enable support for tools or systems - click **Manage Plugins**. Use this page to check which plugins are installed and to install or update any as required. Widely-used plugins include those used for source control systems (such as Git and Subversion). Other plugins you might find useful include:

- Conditional Buildstep
- Copy Artifact
- JUnit Attachments and xUnit
- Node and Label Parameter
- PowerShell
- SCTMExecutor - works with Micro Focus MFUnit support and Micro Focus Silk Central
- vSphere - if you are using agent machines that are stored in VMWare vSphere
- Warnings Plug-in

Creating a Project

Click **New Item** in the Home page of Jenkins interface to create a project such as a freestyle project.

Use the project's configuration page (click **Configure**) to modify various aspects of what it does. Some aspects that might be appropriate for your build system are:

- **Restrict where this project can be run** - specify the agents (such as remote machines) where the work should be performed.
- **Source Code Management** - enter details of where your project is in source control. There are plugins available that allow Jenkins to work with most major source control systems.
- **Build Triggers** - define events that automatically start the execution of the project.
- **Build** - specify what the project should do such as executing Windows or UNIX commands, or Ant scripts.
- **Post-build Actions** - specify the commands that the project should perform upon completion. These could be emailing reports, compiling build results, or triggering other projects based on the output of the current one.

Related reference

Jenkins user documentation

Advanced Configuration

The following sections provide guidance on how to specify more advanced configuration details to integrate Jenkins even more closely with your Visual COBOL development process.

Configuring Email Reporting

You can configure Jenkins so that it sends reports at the end of a build or a test, for example a report listing any test failures.

To configure Jenkins to send email notifications:

1. Click **Manage Jenkins** in the Jenkins Home page.
2. Click **Configure System**.
3. Specify an SMTP server, any recipients and any other required details such as whether to always send an email or to only email about build failures - see the **Extended E-mail Notification** and **E-mail Notification** lists.

To enable email notifications for a project:

1. Navigate to the configuration area of your project.
2. In the **Post-build Actions** section, click **Add post-build action**.
3. Add **Editable Email Notification** and **E-mail Notification** and specify a recipient email and other settings as required.

4. For the **Editable Email Notification**, click **Advanced** and configure email as well as what triggers it (such as when a certain failure has occurred).

Using Sources from Source Control

You can configure Jenkins to access the sources you store in a source control system, of which there are many types. The simplest way to do this is to use a file system watcher which triggers an action when a change occurs in the source control repository.

To enable Jenkins to access the source control, install a required plugin:

1. In the Jenkins Home page, click **Manage Jenkins > Manage Plugins**, and then click the **Available** tab.
2. Locate **File system SCM**, and then install the required plugin by enabling the check box and clicking **Install**.
3. Restart Jenkins (you can select **Ctrl+C** in the command window that is running Jenkins and then start Jenkins again).

The following example shows how to configure your project to access the sources from Subversion:

1. Navigate to the project's configuration area.
2. Under **Source Code Management**, click **Subversion**.
3. In the **Repository URL** area, specify the URL path of the source control that you want to work with.
4. Specify the credentials and a local check-out directory as required.
5. Click **Apply** to save the configuration.

   You have now created a source configuration for your project, but you also need to change the **Build Triggers** section to tell Jenkins to monitor that location.

Triggering Builds Automatically

Jenkins supports triggers to automatically start running your projects. You can use various events as triggers - a successful build started by another project, detecting a change in the code following a commit in the SCM system, or a change as a result of monitoring a URL or a network location to name a few.

You specify triggers in the **Build Triggers** section of the project's configuration. For example:

- **Build after other projects are built** - starts execution after successfully building another project.
- **Poll SCM** - starts execution after a commit to a source code control system. Alternatively, instead of building on a source code change, you could specify that the project is to be executed at a scheduled time.

Creating Environment Variables

There are different ways to specify environment variables in Jenkins. To specify an environment variable that applies globally, go to the Jenkins configuration pages. You can also specify environment variables that only apply to individual nodes, or ones that are set as an external step before starting Jenkins.

In a scenario where you have Visual COBOL installed on multiple platforms on agent machines, using environment variables for common installation paths can greatly simplify the creation of new projects.

For example, you can have an environment variable for the Visual COBOL installation directory on Windows, and another one for the corresponding location on UNIX. Other environment variables could point to locations such as the installation location of Visual Studio, the default source code control system check-out folder, or the location of the samples.

To set global environment variables in Jenkins:

1. From the Jenkins Home page, click **Manage Jenkins**.
2. Click **Configure System**.

3. In the **Global properties** section, check **Environment variables**.
4. Click **Add** and set any environment variables for paths that might be the same across a number of slave machines.

To set environment variables for a node:

1. In the configuration page of the node, check **Environment variables** in the **Node Properties** section.
2. Click **Add** and specify any environment variables as required.

Related information

Best practices - Environment Variables

Using Agents

You can have Jenkins installed on one machine, known as a master, and use it to run projects that control your day-to-day development processes on other machines, known as agents. This can be very helpful if you need to ensure that your applications run correctly on a variety of different test configurations.

Note: An agent can either be a physical or a virtual machine.

The process of setting up agents in Jenkins is as follows:

1. In Jenkins on the master, create a node that defines a connection to the machine you want to use as an agent. See the example below for instructions.
2. On the agent, open the Jenkins UI. Open the page for the node and follow the link to launch a Jenkins agent which will establish a connection between Jenkins on the master machine and the agent machine.
3. In Jenkins on the master, create a project and configure it to execute on the agent.

Example of Creating Agent Machines

The following example shows how to create a connection to an agent machine which is stored in a vSphere cloud. This requires that you install the vSphere plugin in Jenkins.

Ensure that Jenkins is connected to the vSphere cloud:

1. From the Jenkins Home page, click **Manage Jenkins**.
2. Click **General**.
3. Under the **Cloud** section, click **Add new cloud > vSphere cloud**.
4. Specify the configuration and login details for connecting to the vSphere host that contains your virtual machines.

To create a connection to an agent machine, first create a node in Jenkins as follows:

1. Click **Manage Jenkins**.
2. Click **Manage Nodes**.

   This opens the **Nodes** page showing the machine that has Jenkins installed as the master.
3. Click **New Node**.
4. Specify a name for the node and click **Permanent Agent**, then click **OK**.
5. Fill in the details as required - for example, specify a **Remote root directory** if this is needed for checking out sources on the agent.
6. Click **Save**.

To configure the agent to connect to the master:

1. On the agent machine, load the Jenkins Home page (`http://mastermachine:8080`).
2. From the Jenkins Home page, click the node for the machine that appears under the Build Execution Status box.

3. From the Jenkins page for the node, click **Launch** to start the agent that connects the machine to Jenkins.

Configure an existing project to execute on the agent as follows:

1. On the master machine, go to the project configuration page.
2. Check **Restrict where this project can be run** in the **General** section.
3. In the **Label Expression**, specify the name of the node that you just created and connected to.
4. Save the project.

   Next time you build the project, it will execute on the agent.

Using Jenkins to Build COBOL Applications

The following sections describe aspects of Visual COBOL's behavior that you need to configure to enable your Visual COBOL system to work with Jenkins.

In a Jenkins environment you often use Jenkins installed on a master machine to control processes that execute on a number of other machines (agents) running different operating systems that also have Visual COBOL installed. In such configurations, it is useful to have the same Visual COBOL configuration settings on all agents.

Micro Focus recommends that you install Visual COBOL using the same default settings and locations on all Windows and UNIX machines where you will execute your build, test, analysis and deployment tasks. This will make it easier to configure your Jenkins projects.

Setting up the Environment

In most cases, your Jenkins projects execute tasks in a command line environment and not inside the Visual COBOL IDE. This means that the COBOL environment is not automatically preconfigured as it would be if you used the IDE. As a result, when you use Jenkins projects you typically need to manually set the COBOL environment for your projects.

> **Note:** If you use scripts to build your applications, then the scripts might already include the commands for setting the COBOL environment. In this case, you do not need to specify the environment in Jenkins - you only need to configure Jenkins to run the scripts.

To configure the environment for applications created with Visual COBOL for Visual Studio:

1. Ensure Visual COBOL is installed on the machine you are going to build, test or analyze the COBOL sources.
2. In Jenkins, in the **Build** section of the project's configuration page, click **Add build step > Execute Windows batch command**.
3. Enter the following in the **Command** field:

   ```
   call "VSCMDpath\VsDevCmd.bat"
   ```

   Where *VSCMDpath* is the location of the Visual Studio command prompt batch file. For example, for Visual Studio Professional this is `C:\Program Files (x86)\Microsoft Visual Studio\ \Professional\Common7\Tools`. Visual COBOL integrates with the Visual Studio development environment so executing this sets the environment for building COBOL projects created with Visual Studio.
4. In the same **Command** field, specify any commands that you want to be executed in this environment. For example, use MSBuild to compile your projects.

Best Practices When Using Jenkins

This section includes some best practices that you might want to adopt when working with Visual COBOL and Jenkins. Although you do not have to adopt the processes described here, Micro Focus recommends

that you do as they are known to be successful in helping to create an efficient and effective development process.

Using Jenkins With Source Control

Micro Focus recommends that you adopt some common best practices in your commit and build process such as:

- Deliver small changes of code by checking in your code frequently. This ensures that if build problems occur, it is easier to pin-point the exact reason for them.
- Do not check in any changes that are not tested or that contain any errors.
- Do not check in any new changes before fixing the error that caused an earlier Jenkins build failure.

Specifying any Environment Variables in a Project's Configuration

Specifying required environment variables within a Jenkins project's configuration makes the project more portable between different operating systems and reduces the chances of any external factors causing the build to fail.

Using this approach might mean that there could be some duplication in projects so choose the best method depending on how your particular project runs.

Creating Separate Projects for Building and Testing Your Code

A large benefit of continuous integration is receiving feedback quickly on whether a source code commit has introduced any failures. As such it can often be better to use separate projects for your build and test tasks. In this way, if a build fails, Jenkins will not attempt to run any tests and you will receive an error report much faster.

Using Pipelines to Build Your Applications

When you organize your build, test, code analysis and deployment tasks with Jenkins, Micro Focus recommends that you use pipelines in preference to projects.

You can save your pipelines as Jenkinsfiles and store them in your source code control system alongside your code. This means you can use different build instructions in the Jenkinsfile for the different branches of your code that represent the versions of the application. You only configure Jenkins once to use the sources from a particular branch in the source code control system and Jenkins will read the instructions in the pipeline about how to build your application.

Troubleshooting

The following sections describe some issues that you might encounter when using Jenkins with Visual COBOL with suggested workarounds.

Ant Error "Can't find mfant.jar" When Building COBOL Projects

If you receive this error when building any of your COBOL projects the most likely cause is that you have not specified the location of `mfant.jar` to Ant.

To resolve this error perform the operation described in *Making mfant.jar Available to Enable Building*.

Related information

Making mfant.jar Available to Enable Building

COBOL Projects Don't Build

If some of your COBOL projects build successfully but others fail unexpectedly, it suggests that the problem is with the project (or a project-based configuration in either Visual COBOL or Jenkins) rather than with the build system itself.

To investigate what the possible causes might be:

* Open the COBOL project inside the IDE supplied with Visual COBOL and ensure that it builds correctly.
* Check the Jenkins project's configuration and any of the build steps defined in it.

A Build Failure isn't Reported as a Failure

You can do one of the following to check why failures are not reported:

* Check the Jenkins configuration and the project configuration to ensure that any errors will be reported correctly. See *Configuring Email Reporting*.
* Check the project's console log for any error messages.
* Build your application inside the Visual COBOL IDE and check whether any errors are reported.

Related information

Configuring Email Reporting