

JumpStart to Continuous Integration &
Continuous Delivery (CI/CD)
Workshop PUG UK 2023

Sections

Overview	3
ABLUnit Testing Framework	3
Introduction to ABLUnit	3
Setting up the environment for writing/running ABLUnit tests.....	3
Running your first Unit test	4
Introducing the OpenEdge DevOps Framework.....	4
Build with Gradle	4
Prerequisites	5
Setup the environment	5
Prepare the build.gradle script.....	9
Configure Gradle Tasks.....	10
Deployment CI/CD.....	11
Continuous Integration / Continuous Delivery (CI/CD).....	11
How can I set up a CI/CD pipeline	12
Deploy to Staging Environment / Production using a CI/CD Pipeline	12
Setting up Jenkins as CI Server	12
Creating a new Jenkins Project.....	18
Build the Project.....	19

Overview

This workshop will introduce you to the OpenEdge DevOps Framework and how to set up a CI/CD pipeline for your OpenEdge projects.

You should use a Windows platform on which you install the following:

- OpenEdge 12.2
- OpenEdge DevOps Framework 2.1
- Progress Developer Studio
- Git
- Jenkins

This document guides you through the instruction to perform the workshop's labs.

The theoretical concepts will be presented by your instructor.

ABLunit Testing Framework

This section gives an overview of the ABLUnit test framework and explains how to write end-to-end ABLUnit tests using a simple example.

Introduction to ABLUnit

Unit tests are the tests written to verify a small unit of functionality in a software system. Unit testing provides the ability, for any set of inputs, to determine if the software is returning proper values and will gracefully handle failures during execution.

Unit tests are written and executed by developers to make sure that code meets its design and requirements and behaves as expected.

ABLUnit Testing Framework is a unit testing framework from Progress OpenEdge for testing software written in ABL. This framework is available for customers starting OpenEdge 11.4 and has support for both Classes and Procedures.

Setting up the environment for writing/running ABLUnit tests

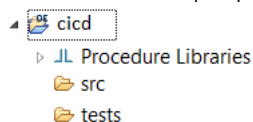
Progress Developer Studio for OpenEdge provides a new perspective for working with ABLUnit called OpenEdge ABLUnit, where it gives a layout with all views required to work with ABLUnit feature in PDSOE. You can add a facet called ABLUnit to existing project or create a new project of type ABLUnit.

This will add to the PROPATH of the project two required libraries:

- **OpenEdge.Core.pl:** Contains Assertions required to work with ablunit.
- **ablunit.pl:** Core framework containing a runner (ablunitcore.p) which is responsible for running unit test cases.

It will also create a new folder, tests as a placeholder to organize all unit tests.

Below is the simple project structure.



src -> Placeholder for source code

tests -> Placeholder for test code (ablunit tests)

PDSOE also provides tooling for writing unit tests through wizards. It will take care of generating test structure for us.

Running your first Unit test

Let's take the example of a greeting procedure written in ABL and write some unit tests for it.

Download the starter project from the [Git repository](#).

Use PDSOE to open the project.

In the src folder, you'll find a procedure called "makegeerting.p" which expects a pcHelloName character input and displays a message 'Hello &1',pcHelloName .

In the tests folder, you'll find the MakeGreeting.cls test class.

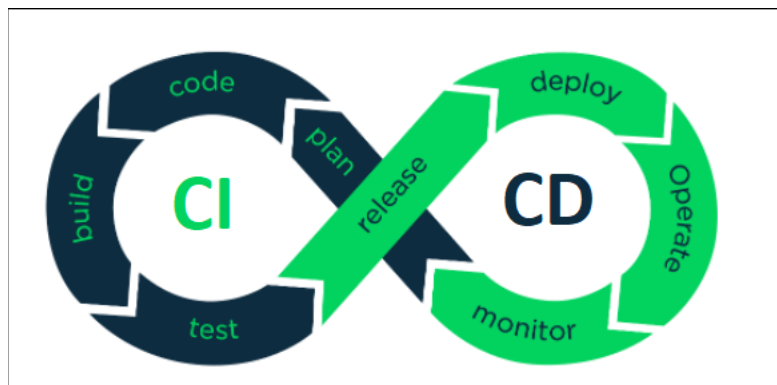
It implements test methods that check whether the greeting returned by the procedure is equal to the expected value, and whether the makegeerting procedure correctly handles the empty input exception.

From PDSOE topbar, run MakeGreeting.cls.

You will notice a new perspective opened. You can rerun a particular test method/procedure, test class, test procedure or a test suite from this perspective.

Introducing the OpenEdge DevOps Framework

Mission critical business applications go through many cycles of incremental changes over the course of their lifespan, complete with code changes, compilation, builds, and validation. Often these cycles are repeated multiple times a day, before a release is made available to users. Continuous integration (CI) is a practice that focuses on optimizing the process of generating these incremental builds and validating them. In the image below, CI represents DevOps and describes a typical software development and delivery process.



For ABL applications, the OpenEdge DevOps Framework is designed to help with implementing an efficient CI pipeline that handles compilation, repository integration, testing, and packaging. It also provides the convenience of sharing the CI pipeline configuration between the development and production build processes.

The OpenEdge DevOps Framework is comprised of a set of plugins designed to address the requirements of two types of users:

1. Users who are new to the CI process and want a simple way to set up their pipeline.
2. Advanced DevOps engineers with a complex CI process and additional flexibility needs.

The OpenEdge DevOps Framework also provides the convenience of sharing the CI pipeline configuration between the development and production build processes.

Build with Gradle

The OpenEdge DevOps Framework provides Gradle plugins for ABL projects.

- ABL base plugin (progress.openedge.abl.base), and
- ABL plugin (progress.openedge.abl)

Plugins are the primary method by which all the useful features are provided in Gradle.

ABL base plugin

The ABL Base plugin provides various capabilities and features required to run or build ABL applications such as running ABL procedures, compiling ABL sources, running ABL unit tests, creating Procedure Library (PL) files, and other such capabilities.

The ABL base plugin provides *Task Types* using which you can create tasks to perform various actions. The name of task types gives an idea about what they are supposed to do.

ABL plugin (progress.openedge.abl)

The ABL plugin depends on the Progress Developer Studio project structure and is recommended for Progress Developer Studio users. This plugin, by default, provides various predefined tasks required to build an ABL project (such as task to compile ABL sources), based on the project's propath and properties. This enables you to build your project with minimum effort and enforce best building practices. These predefined tasks and the necessary parameters required for these tasks (for example, compiling ABL files) can be configured in the build.config file that is part of each Developer Studio project. This ensures that you do not have to maintain separate configuration for development environments and build environments.

Prerequisites

To use OEDF Gradle plugin, you must have:

- Certified OpenEdge 12.2 and later.
- ABL installed with OpenEdge 12.2 and later.
- Gradle 7.3.3.
- Java 11.
- PCT, which is used from DLC or pct by default.
You can optionally specify a different PCT version by providing PCT_VERSION property either as system property, Gradle property, or as an environment variable.
- An ABL project.

Setup the environment

OpenEdge provides the **progradle.bat/sh** script to simplify the OpenEdge DevOps Framework setup in your machine. This script sets up your machine to run the Gradle scripts using the OpenEdge DevOps Framework plugins.

The following activities happen in the background when you execute the progradle.bat/sh script:

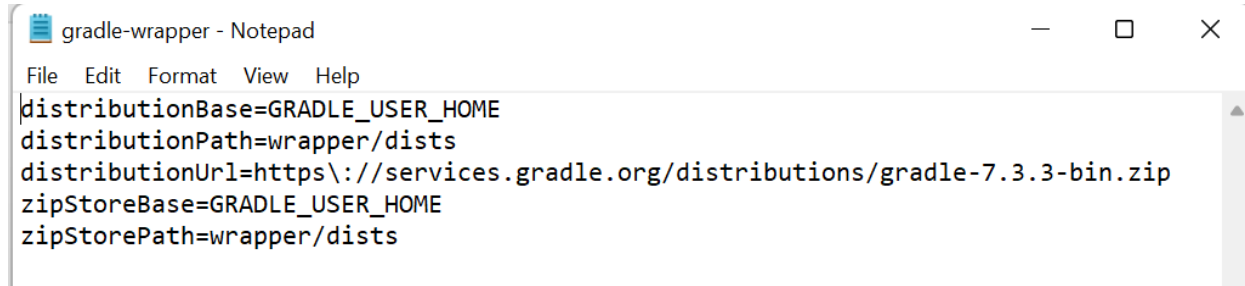
The specified version of gradle is automatically installed when you run the script for the first time.

Note: For OpenEdge 12.2 and later, this script installs Gradle 5.6.x. If you are using OpenEdge DevOps Framework 2.0 or later, then update the default specified version to 7.3.3 at **\$DLC\gradle\wrapper-scripts\gradle-wrapper\gradle-wrapper.properties**.

The JDK used by OpenEdge is automatically configured to run the gradle build scripts.

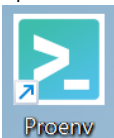
Steps:

Open `$DLC\gradle\wrapper-scripts\gradle\wrapper\gradle-wrapper.properties` file and adjust the `distributionUrl` property as shown below:

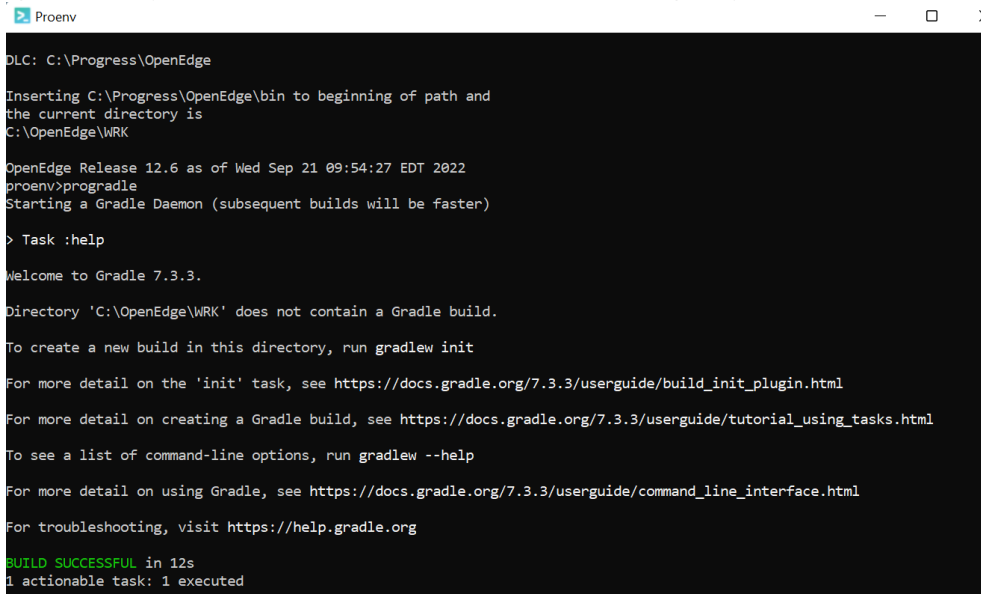


```
gradle-wrapper - Notepad
File Edit Format View Help
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-7.3.3-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

Open Proenv from the Windows start menu.



Run the `progradle` utility to install Gradle. You should see the flowing output.



```
Proenv
DLC: C:\Progress\OpenEdge

Inserting C:\Progress\OpenEdge\bin to beginning of path and
the current directory is
C:\OpenEdge\WRK

OpenEdge Release 12.6 as of Wed Sep 21 09:54:27 EDT 2022
proenv>progradle
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :help

Welcome to Gradle 7.3.3.

Directory 'C:\OpenEdge\WRK' does not contain a Gradle build.

To create a new build in this directory, run gradlew init

For more detail on the 'init' task, see https://docs.gradle.org/7.3.3/userguide/build_init_plugin.html

For more detail on creating a Gradle build, see https://docs.gradle.org/7.3.3/userguide/tutorial_using_tasks.html

To see a list of command-line options, run gradlew --help

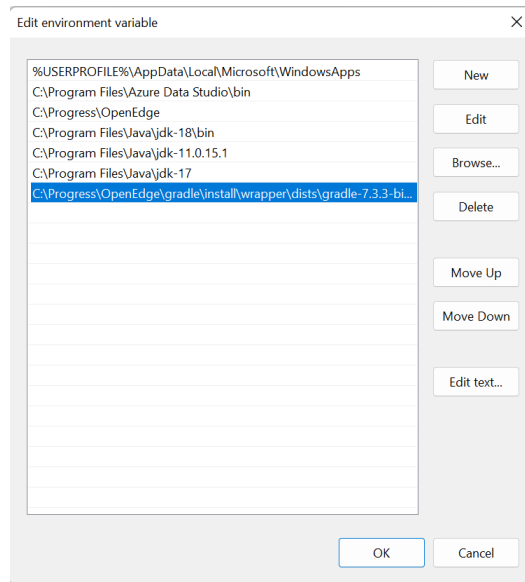
For more detail on using Gradle, see https://docs.gradle.org/7.3.3/userguide/command_line_interface.html

For troubleshooting, visit https://help.gradle.org

BUILD SUCCESSFUL in 12s
1 actionable task: 1 executed
```

Add Gradle to your PATH:

- Go to Edit environment variables.
- Click on "Environment Variable" Button.
- Select "PATH"
- Click on "New"
- Add `$DLC\gradle\install\wrapper\dists\gradle-7.3.3-bin\6a41zxkdtcx8rphpq6y0069z\gradle-7.3.3\bin`
- Click OK to close the window.



Use the OpenEdge project we setup earlier as a starter for your Gradle build.

The gradle init command creates a new Gradle project.

This command also generates all the necessary Gradle build files quickly and easily in our ABL project, which doesn't support Gradle yet.

The gradle init command will run through a setup wizard.

Open a command line and navigate to the root folder of your ABL project.

Type the command gradle init

This starts the Gradle setup wizard, which will prompt several questions:

1. What type of project do you want to create? Select the basic type by typing 1
2. Do you want to write your build scripts in Groovy or Kotlin? Select the Groovy type by typing 1
3. What is the name of your project? By default, Gradle uses the name of the directory you're in. Press Enter to use the default name.

```

C:\Windows\System32\cmd.exe
C:\Users\eddahech\OneDrive - Progress Software Corporation\Desktop\test\CICD>gradle init

Select type of project to generate:
 1: basic
 2: application
 3: library
 4: Gradle plugin
Enter selection (default: basic) [1..4] 1

Select build script DSL:
 1: Groovy
 2: Kotlin
Enter selection (default: Groovy) [1..2] 1

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no]
no

Project name (default: CICD):

> Task :init
Get more help with your project: Learn more about Gradle by exploring our samples at https://docs.gradle.org/7.3.3/samples
es

BUILD SUCCESSFUL in 13s
2 actionable tasks: 2 executed
C:\Users\eddahech\OneDrive - Progress Software Corporation\Desktop\test\CICD>

```

Now we have successful build of Gradle.

Run the command gradlew --help to see some useful getting started information.

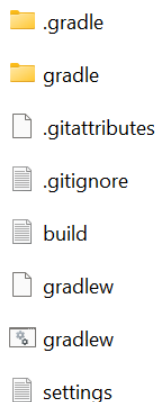
```
C:\Windows\System32\cmd.exe
1 actionable task: 1 executed
C:\Users\eddahech\OneDrive - Progress Software Corporation\Desktop\test\CICD>gradlew --help

USAGE: gradlew [option...] [task...]

-?, -h, --help            Shows this help message.
-a, --no-rebuild          Do not rebuild project dependencies.
-b, --build-file          Specify the build file. [deprecated]
--build-cache            Enables the Gradle build cache. Gradle will try to reuse outputs from previous builds
-c, --settings-file      Specify the settings file. [deprecated]
--configuration-cache     Enables the configuration cache. Gradle will try to reuse the build configuration from
previous builds. [incubating]
--configuration-cache-problems Configures how the configuration cache handles problems (fail or warn). Defaults to fail. [incubating]
--configure-on-demand    Configure necessary projects only. Gradle will attempt to reduce configuration time for large multi-project builds. [incubating]
--console               Specifies which type of console output to generate. Values are 'plain', 'auto' (default), 'rich' or 'verbose'.
--continue              Continue task execution after a task failure.
-D, --system-prop        Set system property of the JVM (e.g. -Dmyprop=myvalue).
-d, --debug             Log in debug mode (includes normal stacktrace).
--daemon               Uses the Gradle daemon to run the build. Starts the daemon if not running.
--export-keys           Exports the public keys used for dependency verification.
-F, --dependency-verification Configures the dependency verification mode. Values are 'strict', 'lenient' or 'off'.
--foreground           Starts the Gradle daemon in the foreground.
-g, --gradle-user-home   Specifies the Gradle user home directory. Defaults to ~/.gradle
-I, --init-script        Specify an initialization script.
-i, --info             Set log level to info.
--include-build         Include the specified build in the composite.
```

e.g.: run gradle tasks to display all the available Gradle tasks in your project.

The gradle init command should add the following file to your project:



How to configure a property

OEDF provides options to configure your project build by setting properties. There are numerous properties that are supported by OEDF depending on the project requirements. For example, one such property supported is PCT_VERSION, which, if set, instructs OEDF to use PCT from Maven instead of using the OpenEdge installation path.

You can configure a property in the following ways:

1. Set as a system property.
For example, run the `-DPCT_VERSION=<PCT-version>` CLI command.
2. Set as a Gradle property.
For example, set `PCT_VERSION=<PCT-version>` in the `gradle.properties` file.
3. Set as an environment variable.

The preferred order of precedence (highest to lowest) to configure a property is 1>2> 3.

Create `gradle.properties` file in the root folder of your ABL project.

Enter the following properties:


```
DLC=##YOUR DLC##
description = "CICD"
group = "com.psc.services.cicd"
version = 0.0.1
pluginVersionABLBase=2.1.0
pluginVersionABL=2.1.0
pluginVersionSonarqube=3.0
pluginVersionTasktree=2.1.0
org.gradle.caching=true
org.gradle.parallel=true
```

Settings

Settings declares the configuration required to instantiate and configure the hierarchy of Project instances which are to participate in a build.

There is a one-to-one correspondence between a Settings instance and a `settings.gradle` settings file. Before Gradle assembles the projects for a build, it creates a Settings instance and executes the settings file against it.

Open `setting.gradle` file and add the following setup.

```
pluginManagement {
    plugins {
        id "progress.openedge.abl-base" version "${pluginVersionABLBase}"
        id "org.sonarqube" version "${pluginVersionSonarqube}"
        id "com.dorongold.task-tree" version "${pluginVersionTasktree}"
        id "progress.openedge.abl" version "${pluginVersionABL}"
    }
}
```

Prepare the build.gradle script

We're now ready to write the build script for our ABL project, using the ABL base plugin.

ABL base plugin (progress.openedge.abl-base)

The ABL base plugin defines various capabilities, under task types and extensions. Task types define individual task capabilities like compile ABL code, run ABL unit tests, and other tasks. Extensions are global configurations available to all the task types.

To use the ABL Base plugin, include the following in your build script (`build.gradle` file):

```
plugins {
    id "progress.openedge.abl-base"
}
```

Gradle can identify the plugin version from the `pluginVersionABLBase` property defined in the `gradle.properties` file.

Open `build.gradle` file and add :

```
import com.progress.gradle.abl.tasks.*
plugins {
    id 'base' // Gives base tasks like build, assemble, check, clean...
    id 'distribution' // for zip creation, etc.
    id "com.dorongold.task-tree" // provides 'taskTree' task, which prints out task dependency trees
```

```
id "progress.openedge.abl-base" // base ABL compilation tasks
}
```

Save the changes and run gradlew tasks on your command line to list all the Gradle tasks defined in the project.

Configure Gradle Tasks

Create tasks using these task types:

ABLCompile: The ABLCompile task compiles ABL code.

ABLUnit: Use this task to run ABLUnit tests.

A detailed description of all methods and properties defined in these task types is provided the OpenEdge DevOps Framework documentation.

Sample code snippet of an ABLCompile task

```
tasks.register("myABLCompile", ABLCompile){
    description "My Compile ABL"
    group "ablbase"
    println "OE version: ${openedgeVersion}"
    avmOptions {
        tty.enabled = 'true'
    }
    compileOptions {
        minSize.enabled = 'true'
        strictOptions {
            requireFullNames="Error"
            requireFieldQualifiers="Error"
            requireFullKeywords="Ignore"
            requireReturnValues="Ignore"
        }
    }
    proppath("src", "${abl.dlcHome}/tty/netlib/OpenEdge.Net.pl")
    source("src")
    include("**/*.p", "**/*.cls")
    rcodeDir="bin"
}
```

Sample code snippet of an ABLUnit task

```
tasks.register("myTestCases", ABLUnit){
    description "My ABL Unit Test Cases"
    group "ablbase"
    dependsOn myABLCompile
    avmOptions {
        tty.enabled = true
    }
    arguments = [
        failOnError : true,
        haltOnFailure: true
    ]
    // include() - procedures and classes to include in the 'run'
    // exclude() - procedures and classes to exclude from the 'run'
```

```

include("**/*.p","**/*.cls")
// source() - folder where source code is located, applied after include/exclude
source("tests")
// "baseade" PL files will already be included
propath("tests", "src", "${abl.dlcHome}/tty/netlib/OpenEdge.Net.pl")
// set wrkDir: for logging output
wrkDir = "tests"
// outputDir : where the results.xml will be
outputDir = "tests"
}

```

Now you are ready to build your project. On the command line, navigate to the root folder of your ABL project. Then, run gradlew build.

It is recommended to always execute a build with the Wrapper to ensure a reliable, controlled and standardized execution of the build. Using the Wrapper looks almost exactly like running the build with a Gradle installation. Depending on the operating system you either run gradlew or gradlew.bat instead of the gradle command.

In case the Gradle distribution is not available on the machine, the Wrapper will download it and store it in the local file system. Any subsequent build invocation is going to reuse the existing local distribution if the distribution URL in the Gradle properties doesn't change.

The following console output demonstrates the use of the Wrapper on a Windows machine for the ABL project.

```

C:\Windows\System32\cmd.exe
OE version: 12.6.0.0
[ant:ABUnit] QUIT statement found

> Task :myTestCases
[ant:ABUnit] Total tests run: 2, Failures: 0, Errors: 0

> Task :compileAbl
Executing compile tasks for each src entries...

> Task :dbConnections
Executing dbConnections tasks...

> Task :compileAbl-root-tests
Execution optimizations have been disabled for task ':compileAbl-root-tests' to ensure correctness due to the following reasons:
- Gradle detected a problem with the following location: 'C:\Users\eddahech\Progress\Developer Studio 12.6\workspace\cicd-web\tests'. Reason: Task ':compileAbl-root-tests'
uses this output of task ':myTestCases' without declaring an explicit or implicit dependency. This can lead to incorrect results being produced, depending on what order the
tasks are executed. Please refer to https://docs.gradle.org/7.3.3/userguide/validation_problems.html#implicit_dependency for more details about this problem.
[ant:PCTCompile] PCTCompile - Progress Code Compiler

[ant:PCTCompile] 1 file(s) compiled
[ant:PCTCompile] 1 file(s) compiled

> Task :compileAbl-root-src
Execution optimizations have been disabled for task ':compileAbl-root-src' to ensure correctness due to the following reasons:
- Gradle detected a problem with the following location: 'C:\Users\eddahech\Progress\Developer Studio 12.6\workspace\cicd-web\tests'. Reason: Task ':compileAbl-root-src'
uses this output of task ':myTestCases' without declaring an explicit or implicit dependency. This can lead to incorrect results being produced, depending on what order the
tasks are executed. Please refer to https://docs.gradle.org/7.3.3/userguide/validation_problems.html#implicit_dependency for more details about this problem.
[ant:PCTCompile] PCTCompile - Progress Code Compiler

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.3.3/userguide/command_line_interface.html#sec:command_line_warnings

Execution optimizations have been disabled for 2 invalid unit(s) of work during this build to ensure correctness.
Please consult deprecation warnings for more details.

BUILD SUCCESSFUL in 3m 11s
6 actionable tasks: 5 executed, 1 up-to-date

```

Deployment CI/CD

Continuous Integration / Continuous Delivery (CI/CD)

As developers, we are constantly looking into improving our development process.

A way to improve the development process is by adopting CI/CD in the organization.

Continuous Integration (CI) is a development practice where developers integrate frequently into a source code repository and the software is built and tested automatically.

Continuous Delivery (CD or CDE) is the ability to deploy the integrated code to a production environment. For example, the application deployment artifacts are ready to be downloaded and installed from a Repository Manager.

Continuous Deployment (CD) is the ability to automatically deploy to a production environment.

A key benefit of CI/CD is continuous feedback, from the build and test environment as well as from users when deploying beta or production level versions.

Overall, the development process can become more efficient with high quality and productivity.

How can I set up a CI/CD pipeline

A way to get started is to set up a basic automated pipeline then iterate:

SCM → Build → Test → Staging → Production

A maturity Model can help to understand the state of CI/CD in the organization:

- <http://bekkopen.github.io/maturity-model/>
- <https://dzone.com/articles/continuous-delivery-the-holy-grail-of-cloud-app-ma>

Depending on the organization requirements you can focus on the areas that need more attention.

Follow best practices as you develop the pipeline:

- Automate the Build / IaC
- Everyone Commits to the Mainline Every Day
- Isolate and Secure Your CI/CD Environment
- Build Only Once and Promote the Result Through the Pipeline

The following articles provide information on best practices:

- <https://www.martinfowler.com/articles/continuousIntegration.html>
- <https://www.digitalocean.com/community/tutorials/an-introduction-to-ci-cd-best-practicesGeneral>

Deploy to Staging Environment / Production using a CI/CD Pipeline

A possible approach to a CI/CD pipeline includes the following components:

- Source Control / SCM
- CI Server
- CI Agent
- Repository Manager (Development)
- Repository Manager (Deployment)
- Infrastructure and Configuration Managers

In this section of the workshop, we use GitHub to provide SCM and Jenkins as CI Server support.

In practice, you can use any SCM and CI Server depending on the requirements of your organization.

Setting up Jenkins as CI Server

Jenkins provides built-in support for CI/CD.

You can install Jenkins directly on the machine or run it as a Docker container.

Link: [Jenkins download and deployment](#)

The simplest way to install Jenkins on Windows is to use the Jenkins Windows installer.

This will install Jenkins as a service using a 64-bit JVM chosen by the user.

To run Jenkins as a service, the account that runs Jenkins must have permission to login as a service.

Prerequisites

Minimum hardware requirements:

- 256 MB of RAM
- 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Software requirements:

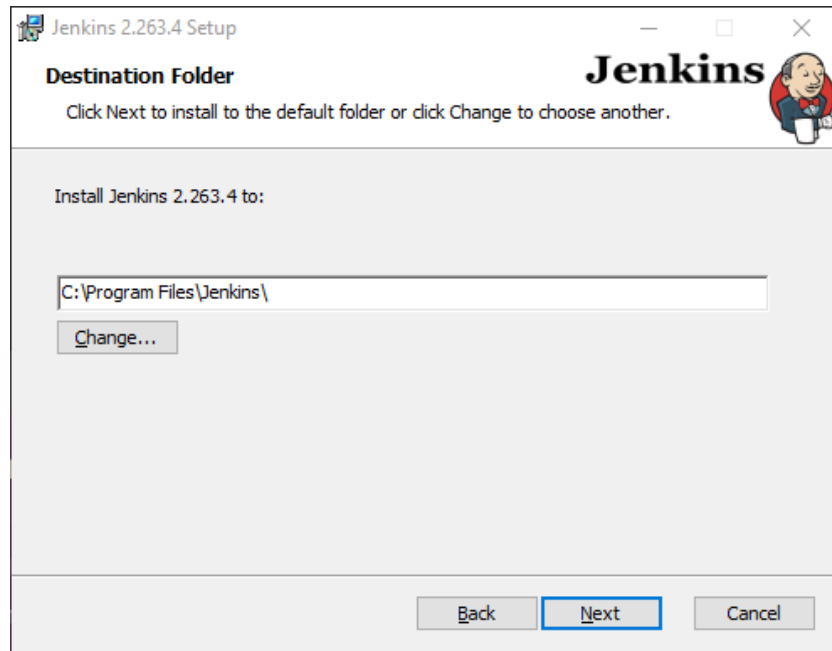
- Java: Jenkins requires Java 11 or 17 since Jenkins 2.357 and LTS 2.361.1
- Web browser
- [Windows Support Policy](#)

Setup wizard

On opening the Windows Installer, an **Installation Setup Wizard** appears, Click **Next** on the Setup Wizard to start your installation.



Select the destination folder to store your Jenkins Installation and click **Next** to continue.

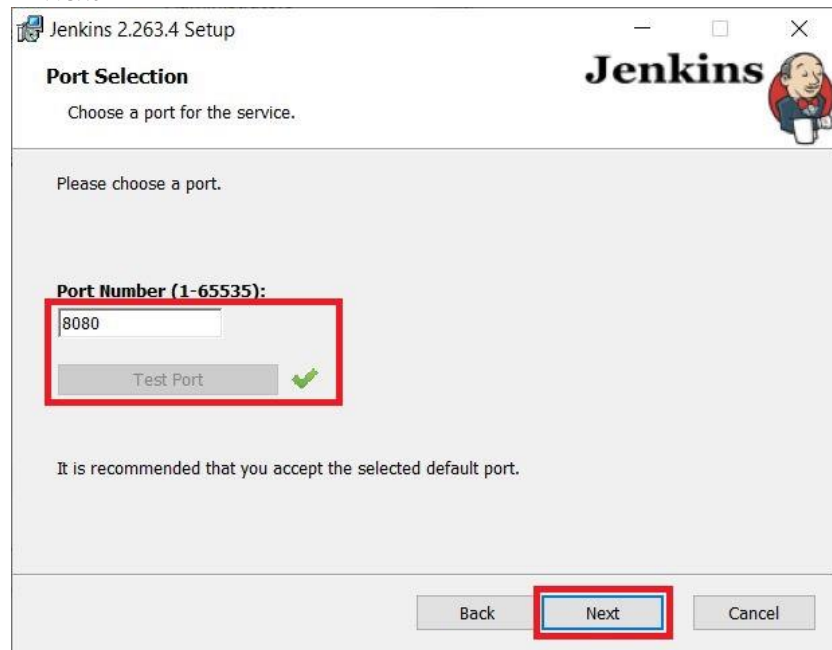


When Installing Jenkins, it is recommended to install and run Jenkins as an independent windows service using a local or domain user as it is much safer than running Jenkins using LocalSystem(Windows equivalent of root) which will grant Jenkins full access to your machine and services. To run Jenkins service using a local or domain user, specify the domain username and password with which you want to run Jenkins, click on Test Credentials to test your domain credentials and click on Next.

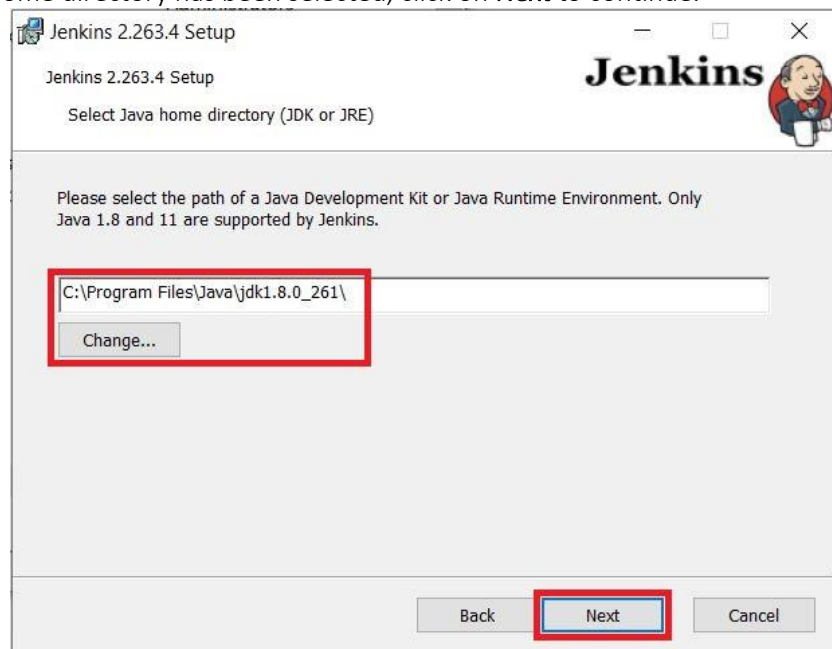


If you get **Invalid Logon** Error pop-up while trying to test your credentials, follow the steps explained [here](#) to resolve it.

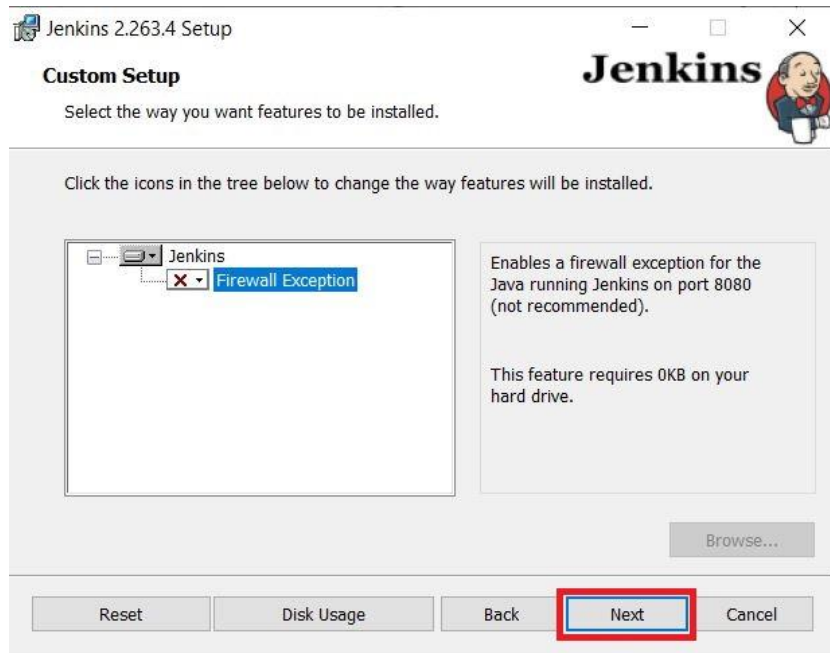
Specify the port on which Jenkins will be running, **Test Port** button to validate whether the specified port is free on your machine or not. Consequently, if the port is free, it will show a green tick mark as shown below, then click on **Next**.



The installation process checks for Java on your machine and prefills the dialog with the Java home directory. If the needed Java version is not installed on your machine, you will be prompted to install it. Once your Java home directory has been selected, click on **Next** to continue.



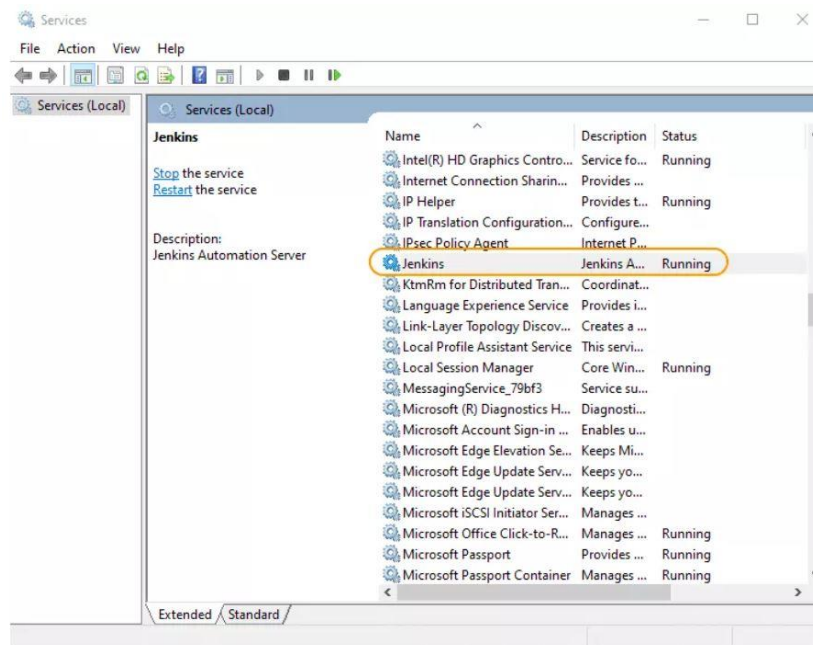
Select other services that need to be installed with Jenkins and click on **Next**.



Click on the **Install** button to start the installation of Jenkins.



Once the installation is completed, click on **Finish** to complete the installation. Jenkins will be installed as a **Windows Service**. You can validate this by browsing the **services** section, as shown below:



Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically generated password.

Browse to <http://localhost:8080> (or whichever port you configured for Jenkins when installing it) and wait until the Unlock Jenkins page appears.

The initial Administrator password should be found under the Jenkins installation path.

For default installation location to C:\Program Files\Jenkins, a file called **initialAdminPassword** can be found under C:\Program Files\Jenkins\secrets.

If a custom path for Jenkins installation was selected, then you should check that location for **initialAdminPassword** file.

Open the highlighted file and copy the content of the **initialAdminPassword** file.

On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**.

After unlocking Jenkins, the **Customize Jenkins** page appears. Here you can install any number of useful plugins as part of your initial setup.

Click **Install suggested plugins** - to install the recommended set of plugins, which are based on most common use cases. You can install (or remove) additional Jenkins plugins at a later point in time via the **Manage Jenkins > Plugins** page in Jenkins.

After customizing Jenkins with plugins, Jenkins asks you to create your first administrator user.

When the **Create First Admin User** page appears, specify the details for your administrator user in the respective fields and click **Save and Finish**.

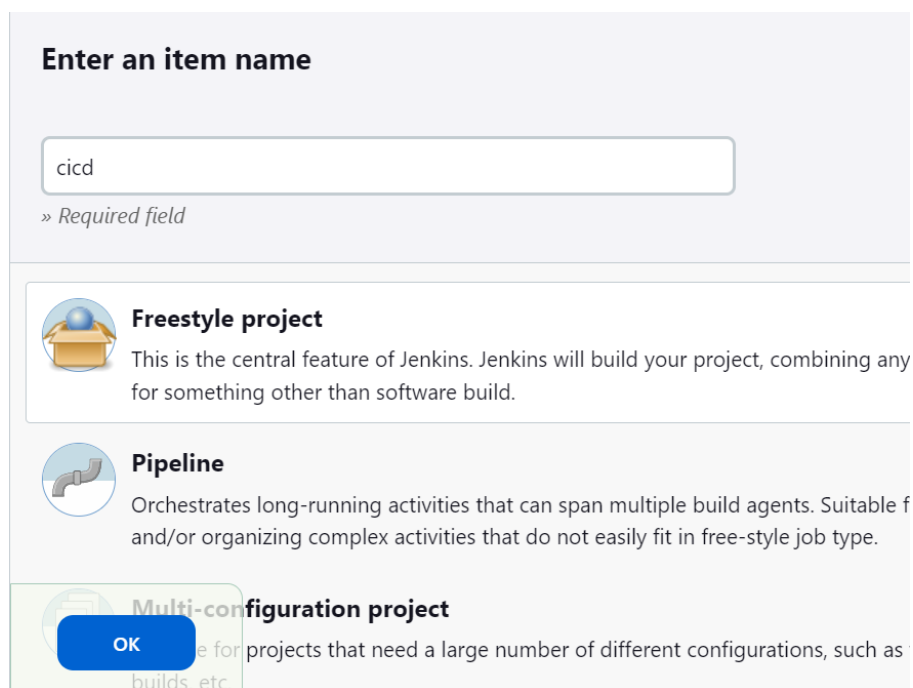
When the **Jenkins is ready** page appears, click **Start using Jenkins**.

Creating a new Jenkins Project

You now have access to your own Jenkins environment.

Now, you can create a new project. Click on “New Item” from the main menu, enter an item name “cicd-workshop” and select the “Freestyle project” then click on “OK”.

The freestyle projects are meant to automate simple jobs, such as running tests, creating and packaging applications, producing reports, or executing commands.



Enter an item name

cicd

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Useful for projects that need a large number of different configurations, such as builds, etc.

OK

The configuration page lets you define the tasks performed by the Jenkins project.

Enter a description for the project.

Check GitHub project and enter the URL for the GitHub hosted project:

<https://github.com/SoumayaEddahech/cicd-workshop-2023/>

In Source Code Management, select Git and specify the URL or path of the git repository. This uses the same syntax as your git clone command.

<https://github.com/SoumayaEddahech/cicd-workshop-2023.git>

If your Git repository is public, leave the "credentials" field empty, otherwise configure the required access information.

Specify the branches if you'd like to track a specific branch in a repository. Enter: */main
Specify the HTTP URL for this repository's GitHub page in the Repository browser section.
Select githubweb and insert <https://github.com/SoumayaEddahech/cicd-workshop-2023/tree/main> in the Url field.

For projects that use Gradle as the build system. This causes Jenkins to invoke Gradle with the given tasks.
Under the Build Steps > Invoke Gradle script, select the option Use Gradle Wrapper and set the Wrapper location field to ./



In the Task field specify a list of Gradle tasks to be invoked: build

Then click on the Save Button.

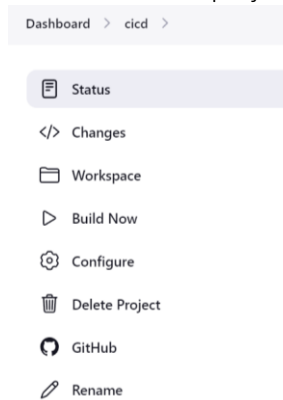
Ready to build your project with Jenkins.

Build the Project

Your project will be displayed on the Dashboard.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		cicd	N/A	N/A	N/A

A Build can be scheduled from the dashboard or from the project view.



The first build may take some time, as Jenkins will clone the source code from Git and perform a full build of the project.

A rebuild will only consider the changes made to the project and will therefore be quicker.

Check the Console Output of your build under Dashboard>{Project}>{#BuildId}> Console Output.

It should show the Executed Gradle Tasks.

Check your build:

From Jenkins Dashboard menu, navigate to Dashboard > Manage Jenkins > System.

Jenkins stores by default all its data in the <Home directory> on the file system.

Open the <Home directory>\workspace\<JenkinsProject>

(e.g.: C:\ProgramData\Jenkins\.jenkins\workspace\cicd). You will notice that Jenkins downloaded your code source from GitHub and built it. Check <Home directory>\workspace\<JenkinsProject>\build to see the output of your build and <Home directory>\workspace\<JenkinsProject>\tests to see the output of your tests.

Congratulations!

You have completed the workshop.