

Continuous Delivery with the Bridge

Based on a straightforward and repeatable process, the continuous delivery approach helps with building, testing and releasing software faster. On this page, we will show how you can automate your build, deploy, and testing processes using an automation tool together with the Bridge command line tools.

To take full advantage of this scenario, you need the [xUML Command Line Compiler](#) which is part of Bridge 7.

Prerequisites

The following will guide you through an continuous delivery example. Here, we are using **Jenkins**, an open source automation tool, and **Git**, an open source version control tool. To set up a similar scenario, you need

- a Builder development project that has been checked in into a Git repository
- a Jenkins installation, e.g. 2.89.2.
- the [xUML Command Line Compiler](#)
- the [Bridge Command Line Interface](#)
- the [Regression Test Runner](#)

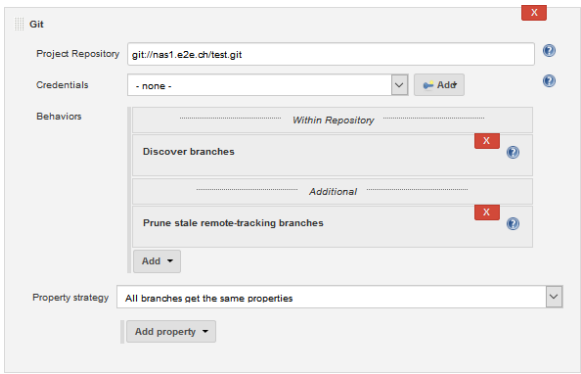
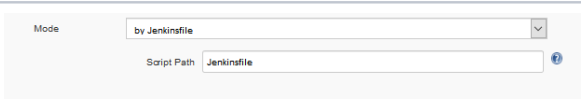
Approach

- Set-up a job in Jenkins that checks minutely for changes to your git repository and triggers a build process in case of changes.
- The build process itself is defined in a Jenkinsfile that is part of the Git repository and contains the following steps:
 1. Building the xUML repository file using the [xUML Command Line Compiler](#).
 2. Deploying the compiled repository using the [Bridge Command Line Interface](#).
 3. Running regression tests on the deployed services with the [Regression Test Runner](#).

Setting Up a Job in Jenkins

First, you need to set up a job in Jenkins that will look for changes to your Git repository regularly and trigger the build process in case of changes.

Create a new Jenkins item: a **Multibranch Pipeline** job.

	Specify where to get the sources from: it will be Git in this example. Select a Git repository and provide credentials if necessary.
	Build configuration will be by Jenkinsfile. This file will be explained further below.

On this Page:

- [Prerequisites](#)
- [Approach](#)
- [Setting Up a Job in Jenkins](#)
- [Setting Up a Jenkins Command File](#)
- [The Build Process](#)
- [Some Jenkins Hints](#)
- [Other Useful Tasks to be Automated](#)

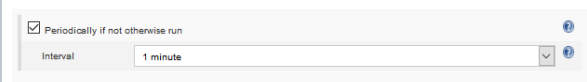
Related Pages:

External Tools:

- [Git Documentation](#)
- [Jenkins User Documentation](#)
- [Apache Groovy Script Documentation](#)

Bridge Tools:

- [xUML Command Line Compiler](#)
- [Bridge Command Line Interface](#)
- [Regression Test Runner](#)

	Specify the triggers that will lead to execution of this job. Check Periodically and select 1 (once per) minute. Jenkins will check every minute for changes on the repository and then do what is configured in the Jenkinsfile.
---	---

Setting Up a Jenkins Command File

In your Git repository, you need a **Jenkinsfile** - a Apache Groovy script file that contains the build steps to be performed by Jenkins. The file must be named exactly like that, Jenkinsfile (without extension), and must reside in the root directory of the Git repository.

```
#!/groovy

pipeline {
    agent {
        node {
            label 'Windows'
            customWorkspace "workspace/test-xUML-project"
        }
    }

    options {
        buildDiscarder(logRotator(numToKeepStr: '10',
artifactNumToKeepStr: '1'))
        disableConcurrentBuilds()
    }

    parameters {
        choice(name: 'XUMLC', choices: 'D:/jenkins/userContent/xumlc/xumlc-7.10.0.jar', description: 'Location of the xUML Compiler')
        choice(name: 'REGTEST', choices: 'D:/jenkins/userContent/RegTestRunner/RegTestRunner-nightly.jar', description: 'Location of the Regression Test Runner')
    }

    stages {
        stage('Build') {
            steps {
                dir('Advanced Modeling/E2ELibrary') {
                    bat """
                        java -jar ${XUMLC} -uml uml/librarySQLQuery.xml
                        copy repository\\librarySQLQuery\\librarySQLQuery.
lrep libs\\
                        java -jar ${XUMLC} -uml uml/useLibrarySQLQuery.xml
                        """
                    archiveArtifacts artifacts: 'repository
/useLibrarySQLQuery/UseE2ELibraryExample.rep'
                }

                dir('Advanced Modeling/PState') {
                    bat """
                        java -jar ${XUMLC} -uml uml/pstatePurchaseOrder.xml
                        """
                    archiveArtifacts artifacts: 'repository
/pstatePurchaseOrder/PurchaseOrderExample.rep'
                }
            }
        }
    }
}
```

```

    }
  }
  stage('Deploy') {
    steps {
      dir('Advanced Modeling') {
        bat '''
          e2ebridge deploy E2ELibrary/repository
/useLibrarySQLQuery/UseE2ELibraryExample.rep -h <Bridge host> -u <user> -P
<password> -o overwrite
          e2ebridge deploy PState/repository
/pstatePurchaseOrder/PurchaseOrderExample.rep -h <Bridge host> -u <user> -
P <password> -o overwrite
          '''
        }
      }
    }
  }
  stage('Test') {
    steps {
      dir('Advanced Modeling') {
        bat """
          java -jar ${REGTEST} -project PState -suite "QA
Tests/Tests" -logfile result.xml -host <Bridge host> -port <port> -
username <user> -password <password>
          """
        }
      }
      post {
        always {
          junit 'Advanced Modeling/result.xml'
        }
      }
    }
  }
}

```

Code Snippet	Description
<pre> agent { node { label 'Windows' customWorkspace "workspace/test-xUML-project" } } </pre>	Section agent defines - the name of the agent that should execute the build - the common workspace directory on this agent for all branches within the Git repository. If not specified, Jenkins will create a separate workspace for every branch.
<pre> options { buildDiscarder(logRotator (numToKeepStr: '10', artifactNumToKeepStr: '1')) disableConcurrentBuilds() } </pre>	Section options defines - the count of build logs to keep - the count of build artifacts to keep - whether concurrent builds are allowed
<pre> parameters { choice(name: 'XUMLC', choices: 'D:/jenkins/userContent /xumlc/xumlc-7.10.0.jar', description: 'Location of the xUML Compiler') choice(name: 'REGTEST', choices: 'D:/jenkins/userContent /RegTestRunner/RegTestRunner- nightly.jar', description: 'Location of the Regression Test Runner') } </pre>	Section parameters defines parameters to use further below in the script. If you provide multiple choices, Jenkins will generate a dropdown list to select from. The first list item serves as the default value. This default will be selected, if the script is triggered automatically. A build with newly added parameters will always fail for the first time, because Jenkins needs the first run to add the parameters to the pipeline configuration.

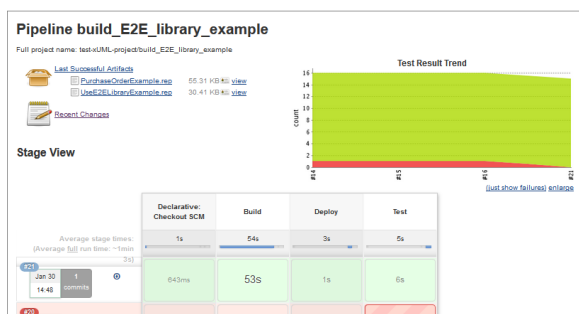
<pre> stages { stage('Build') { [...] } } </pre>	In section stages, you can define named build stages. If the processing of one stage fails, the subsequent stages will not be processed.
<pre> steps { dir('Advanced Modeling /E2ELibrary') { bat """ java -jar \${XUMLC} -uml uml/librarySQLQuery.xml copy repository\\librarySQLQuery\\lib rarySQLQuery.lrep libs\\ java -jar \${XUMLC} -uml uml/useLibrarySQLQuery.xml """ archiveArtifacts artifacts: 'repository/useLibrarySQLQuery /UseE2ELibraryExample.rep' } } </pre>	In section steps, you can define the tasks to process in this stage, e.g. • dir: to change the active directory within the Git repository • bat: to execute batch commands (Windows), e.g. 1. Call the library build. 2. Copy the library repository to the libs folder. 3. Call the xUML model build. Wrap the batch commands in ◦ single quotes ('), if Jenkins variables should not be resolved within the batch command ◦ double quotes ("), if Jenkins variables should be resolved within the batch command ◦ triple single quotes (' ') or triple double quotes ("""") for multiple line batch scripts • archiveArtifacts: to define a list of artifacts (outputs) of this job. These artifacts can be downloaded via the Jenkins console.
<pre> java -jar \${XUMLC} -uml uml /librarySQLQuery.xml </pre>	Call the xUML Command Line Compiler. The location of the compiler is specified via a Jenkins parameter (see above).
<pre> copy repository\\librarySQLQuery\\lib rarySQLQuery.lrep libs\\ </pre>	Copy the compiled library repository to the libs folder of the project, so it will be used when compiling the usage model. For more details, see Compiling Libraries and Library Usage Models.
<pre> e2ebridge deploy E2ELibrary /repository/useLibrarySQLQuery /UseE2ELibraryExample.rep -h <Bridge host> -u <user> -P <password> -o overwrite </pre>	Call the Bridge CLI to deploy the compiled service to a Bridge.
<pre> dir('Advanced Modeling') { bat """ java -jar \${REGTEST} - project PState -suite "QA Tests /Tests" -logfile result.xml - host <Bridge host> -port <port> -username <user> -password <password> """ } </pre>	Call the RegTestRunner to perform regression tests on the newly deployed service. The location of the RegTestRunner is specified via a Jenkins parameter (see above).

For more details on Apache Groovy, refer to the [Apache Groovy Script Documentation](#). You can check the script's syntax before execution using Pipeline Linter, a command line tool that is coming with Jenkins. Refer to the [Jenkins Documentation](#) for more information on Linter.

The Build Process

And that's all? Yes, it is. Every time you will push changes to the related Git repository, the multibranch pipeline job will be triggered automatically and perform the defined steps.

On the Jenkins console, you can find a nice overview on each test run:



```
[Pipeline] }
[Pipeline] bat
[Advanced Modeling] Running batch script

D:\jenkins\workspace\test-xMML-project\Advanced Modeling>e2ebridge deploy E2ELibrary\repository
\useLibrarySQLQuery\UseE2ELibraryExample.rep -h e2ebridge.e2e.ch -u admin -P admin -o overwrite
Working, please wait.
D:\jenkins\workspace\test-xMML-project\Advanced Modeling>E2ELibrary\repository
\useLibrarySQLQuery\UseE2ELibraryExample.rep: [32mSUCCESS
[39m[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] dir
Running in D:\jenkins\workspace\test-xMML-project\Advanced Modeling
[Pipeline] {
[Pipeline] bat
[Advanced Modeling] Running batch script

D:\jenkins\workspace\test-xMML-project\Advanced Modeling>java -jar D:\jenkins\userContent
\RegRunner\RegTestRunner-nightly.jar -project F8State -suite "QA Tests/Tests" -logfile
result.xml
Running Test '<startService>Start PurchaseOrderExample'.
Running Test '<testCaseTest>Create Purchase Order 1'.
Running Test '<testCaseTest>Create Purchase Order 2'.
Running Test '<testCaseTest>Add Item 1'.
Running Test '<testCaseTest>Add Item 2'.
Running Test '<testCaseTest>Add Gratifications for Wishes Unlimited'.
Running Test '<testCaseTest>getCallPurchaseOrders'.
Running Test '<testCaseTest>Get Purchase Order 1'.
Running Test '<testCaseTest>Get Purchase Order 4'.
Running Test '<testCaseTest>Check Out Order 1'.
Running Test '<testCaseTest>Check Out Order 2'.
Running Test '<testCaseTest>Get Purchase Orders from 1970-11-01'.
Running Test '<testCaseTest>Close Purchase Order 1'.
Running Test '<testCaseTest>Close Purchase Order 2'.
Running Test '<stopService>Stop PurchaseOrderExample'.
Finished: 4.235 seconds
[Pipeline] }
[Pipeline] // dir
Post stage
[Pipeline] junit
Recording test results
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // ws
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Hint	Description
first run of a Jenkins job	The first run of a Jenkins job may take a little longer, depending on the size of the Git repository that is checked out.
additional stages	You can not only use Jenkins to automate your building, deploying and testing, but also to • publish the build result to a delivery endpoint • send mails containing the build results to all who are concerned
parameters	You can use parameters with your multibranch pipeline (see also Jenkinsfile above). • Parameter definition: Jenkinsfile • Parameter value specification: Jenkins job configuration or first item if choices (default value), if triggered automatically • Parameter usage: Jenkinsfile • The first run of a parameterized pipeline will fail, because Jenkins needs the first run to add the parameters to the pipeline configuration. Same, when parameters change.

concurrent builds	You can use disableConcurrentBuilds() to prevent the multibranch pipeline to be triggered while another build is still running. Nevertheless, this will not prevent multiple branches within the same job being processed in parallel. To prevent this, you could e.g. allow certain build steps - that cannot run in parallel - for specific branches only.
--------------------------	---

Other Useful Tasks to be Automated

Task	Description
change service preferences	You can use the Bridge CLI to change the preferences of a service, e.g. to set the flag for automatic startup: <pre>e2ebridge preferences <service name> --pref. automaticStartup=true -u <user> -P <password></pre> You can list all available preferences with <pre>e2ebridge preferences <service name> -u <user> -P <password></pre>
change service settings	You can call the Bridge API to change the settings of a service: <pre>e2ebridge settings <service name> [-n --nodejs] [set <setting name> <settings value>]... [Bridge connection]</pre>