ThingWorx DevOps with Jenkins

Costin Badici
Principal IoT/Analytics Field Engineer
cbadici@ptc.com

PTC Inc., 121 Seaport Boulevard, Boston, MA 02210 USA

Document Revision History

Revision Date	Version	Description of Change
14/06/2019	1.0	Initial Document
03/07/2019	1.1	Adjustments
11/07/2019	1.2	Adjustments
17/07/2019	1.3	Adjustments
16/09/2019	1.4	Adjustments
25/02/2020	1.5	Adjustments – Integration with SC and Improvements
27/04/2020	1.6	Adjustments – Setup Docker for running tests, trigger pipeline remotely
26/08/2020	1.7	Adjustments – republished by EDC team

Introduction

This document describes how to use a CI/CD Jenkins Pipeline for ThingWorx, providing detailed information about how to setup your ThingWorx instance and how to configure your Jenkins Pipeline. The Pipeline is intended as an example / starting point for managing your DevOps in ThingWorx and it can easily be extended. Please note that this Pipeline is not officially supported by PTC.

DevOps Process

This section outlines the DevOps process in ThingWorx at a high-level. It is assumed that at least a development (DEV) and test (TEST) instance have already been deployed. Typically, customers also have an instance for UAT/performance testing (QA/UAT) and a production instance (PROD, which is the published application).

This example also assumes there is a git repository for source control. If you are planning to use a different source control system (or none at all), you will need to edit some of the services provided below.

In the above diagram, you will find the main steps of the proposed Jenkins Pipeline. This Pipeline assumes that a development sprint has finished, and testing has begun. Now it is time to decide whether to move forward with the deployment (to PROD or a QA/UAT environment) or fix any bugs.

In the first step, the application is exported from the DEV server. This means that your extensions, all the entities developed in your project including system objects you have previously developed will be exported and pushed into your git repository.

The next step is to install the application on a test server. This implies pulling the branch from your git repository on which you pushed the project (assume we always use the master branch), installing all the required extensions, entities related to the project, etc. There is also support available for setting up a Docker container with ThingWorx as a test server.

The last step will involve running the tests you have previously developed (examples also shown in the DevOps project). The output of the tests will be presented in an HTML page in Jenkins. As an optional step, for users of ThingWorx 8.5 or higher, you can also publish your project to Solution Central if all tests have been passed successfully. To read more about Solution Central and how to register your instance, please visit this link.

Based on the results of the tests, you can decide to deploy the application to another environment to perform any further tests or to go back to development and fix any bugs. This

step is not presented here, but it can be easily automated using the services already developed in the example.

Configuration and Usage

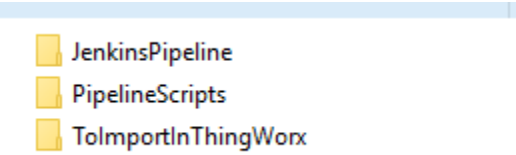
Prerequisites

For the DevOps sample project to work, it is assumed that you are using a git repository for backup and source control purposes. Prior to importing the DevOps entities, please download and install the Git Backup[1] extension from the PTC Git Repository. Another prerequisite for the DevOps is to have Jenkins installed. More information about how to install Jenkins can be found here: https://jenkins.io/doc/book/installing/

Furthermore, the scripts used in the Jenkins Pipeline use cURL commands. Make sure that cURL is available on the machine where you plan to use Jenkins. If you plan to use a test server on Docker, make sure that you have built the corresponding Docker image on the server where you have Jenkins installed. More information about building your Docker image for ThingWorx can be found here.

ThingWorx Jenkins Package

The provided package has the following folder structure:



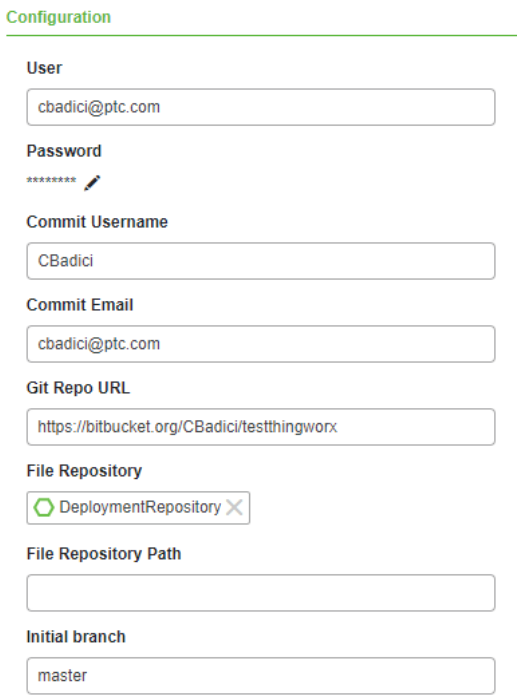
- **JenkinsPipeline**: contains the job that Jenkins will execute, available for both Linux and Windows
- **PipelineScripts**: copy this folder to a location on the machine where Jenkins is installed. The path to the scripts will be referenced in the Pipeline code; This folder contains another subfolder with the Git Extension, DevOps entities and a bash script; these files are relevant only for setting up a Docker container for running the tests
- **ToImportInThingWorx**: contains the entities necessary on the DEV and TEST instances in ThingWorx

DevOps Project in ThingWorx

As a first step, go to the Import/Export menu in ThingWorx and import the DevOps related entities.

[1] The services provided in this project have been tested using the Git Backup Extension v. 1.3.1

Next, create a Thing based on the *GitBackupTemplate*, go to the Configuration tab and add the details of your git repository. Make sure to also add a ThingWorx file repository. *For testing purposes, the services provided in this example will work when the File Repository path from the Git thing is empty, as indicated below.*



As a next step, open the *DeploymentManagerThing* and go to the *Properties and Alerts* tab. Change the *ProjectName* property value to the project that you are using in your application and also the GitThingName, whatever implements the *GitBackupTemplate*. Optionally, you can change the rest of the properties such as the different paths you will use for source control, the app key, etc.

- **AppKey** – Application key name that will be used by Jenkins to connect to your environment. Needs to be the same application key on both DEV and TEST environments
- **EntitiesPath** – Path on the deployment repository where the entities will be saved
- **Extensions** – Infotable; if left empty, all extensions from Dev will be installed on the test system
- **ExtensionsPath** – Path on the deployment repository where extensions will be saved
- **GitThingName** – Thing that will be used to connect to your Git repository
- **PermissionsPath** – Path on the deployment repository where the user permissions will be exported
- **ProjectName** – Entities belonging to this project will be exported and imported on the test system
- **SystemObjectsPath** – Path on the deployment repository where the modified system objects will be exported (these system objects cannot be added to the project and therefore require a separate export)
- **TagSystemObjects** – Tag that is used for the modified system objects

Now, everything is in place for the first two stages of the pipeline, which involves packaging the application and installing it on another environment (e.g. TEST). For these steps, you have the option to add any missing dependencies to your project, to include the passwords in the export, and to remove the old entities from the test/target server.

The next phase involves testing your application. Both unit as well as integration tests can be automated via ThingWorx services. This example establishes a common framework for testing by providing a *TestingTS* Thing Shape with overridable services. These services have the following role:

- **Create the test data** – Should be overridden to create any test data. This is optional.
- **Execute different tests** – ExecuteTest service should be overridden. All tests are executed sequentially to build a test results Infotable. All tests should be marked with the Category "Test"
- **Delete test data** – Should to be overridden to delete the test data
- **Run all tests** – Needs to be overridden to execute all tests. This service will create the test data (optional), execute all the tests (services marked with the "Test" category) and return the result Infotable. Optionally, the test data can be deleted
- **RunAllTestsFormatResult** – Service is not overridable, will run all tests, and convert the result to HTML. This is the service executed in the last step of the Jenkins Pipeline
- **FormatTestResults** – Non-overridable service for formatting the test results from Infotable to HTML

****If the *TestResultData* datashape is modified, please modify the service FormatTestResults accordingly in the *TestingTS****

The developers should implement one or more things using the *TestingTS* and override these services. In the end, we should have a single test thing executing all the tests.

An example is provided in *TestExample1* – RunAllTests. To view the results on a correctly formatted HTML page in Jenkins, Jenkins will call the service RunAllTestsFormatResult. If you plan to modify any fields from the *TestResultData* Data Shape, then please also adjust this service accordingly.

Please note that in Jenkins, with the provided example, all tests are executed as a single operation (sequentially) so it is therefore important to wrap all of them in a single service, such as RunAllTests, even if you plan to run more tests developed in different entities.

Please consider you also have the option to publish your project to Solution Central if all tests have run successfully.

Configuring the Jenkins Pipeline

After installing Jenkins, please ensure that you have the following plugins installed:

- Pipeline
- Pipeline Utility Steps
- HTML Publisher

- Permissive Script Security[2]

The easiest method to install Jenkins plugins is from the web UI, by accessing Manage Jenkins – Manage Plugins. Here you can find other methods for installing plugins: https://jenkins.io/doc/book/managing/plugins/

As a next step, copy the corresponding Jenkins project folder[3] (depending on whether you are on Windows or Linux), "TwJenkins", to the path: *JENKINS_HOME/jobs/.*

Ensure that the *TwJenkins/config.xml* is not a Read-Only file so that you can edit the file from Jenkins.

In addition to this, please add the following option for Java:

```
-Dpermissive-script-security.enabled=true
```

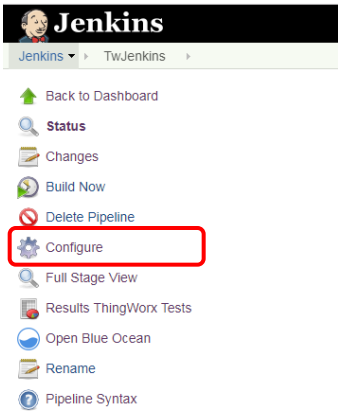
On Windows, this should be added in %JENKINS_HOME%\jenkins.xml in the following section:

```
  if you'd like to run Jenkins with a specific version of Java, specify a full path to java.exe.
  The following value assumes that you have java in your PATH.
  -->
<executable>%BASE%\jre\bin\java</executable>
<arguments>-Dpermissive-script-security.enabled=true -Xrs -Xmx256m -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecyc
  <!--
```

On Linux (CentOs), go to /etc/*sysconfig* and edit the Jenkins file by adding this argument to `JENKINS_JAVA_OPTIONS`.

Restart Jenkins and then login to the web UI. You should now see the TwJenkins job available.

Select the job and then click on Configure:



Go to the Pipeline script section and edit the environment variables used throughout the pipeline script to execute different commands in ThingWorx (see image).

---

[2] Some operations from the Jenkins Pipeline will require administrator approval, such as deleting a file. These approvals are granted by going to Manage Jenkins – In-Process Script Approvals. The signatures approved to run this Jenkins Pipeline are:

- method groovy.lang.GroovyObject getProperty java.lang.String
- method java.io.File delete
- new java.io.File java.lang.String
- staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods append java.io.File java.lang.Object
- staticMethod org.codehaus.groovy.runtime.ScriptBytecodeAdapter unaryPlus java.lang.Object

[3] *TwJenkinsPackage/JenkinsPipeline/Windows/TwJenkins* for Windows or *TwJenkinsPackage/JenkinsPipeline/Linux/TwJenkins* for Linux

Pipeline

Definition: Pipeline script

Script:

```
pipeline {
  agent any
  environment {
    DEV = 'http://172.27.163.209'
    TEST = 'https://pp-19052309559c.portal.ptc.io'
    TEST_THING = 'TestExample1'
    SCRIPT_PATH = 'C:\\Users\\cbadici\\Documents'
    TEST_RESULT_PATH = 'C:\\Users\\cbadici\\Documents\\testresult\\Thingworx'
    APPKEY = 'de367fd7-aca6-4b7d-adac-cfe218c430bb'
  }
```

- DEV – protocol://hostname for your development instance
- TEST – protocol://hostname for your test instance
- TEST_THING – A single thing from the test instance that will execute all tests using the RunAllTestsFormatResult service
- SETUP_DOCKER – Either true or false, depending on whether you would like to run a Docker container with ThingWorx for your automatic tests; in case you set this up to true, make sure you have previously built the Docker image on the server where Jenkins is running
- INSTALL_EXTENSIONS – Whether to install the extensions as well. Should be set to 'true' whenever you have updated some extensions on the DEV server or when you are importing the application for the first time on the TEST server
- ADD_DEPENDECIES - If set to true, will add any missing dependencies to the project before the Package step; should be set to true only for ThingWorx 8.5 or higher
- INCLUDE_PASSWORDS – Option to include passwords; please note that setting this option to true will mean that the passwords will be exported in clear text. It is not recommended to turn this setting to true
- REMOVE_OLDPROJECT – If set to true, will erase the project on the target (TEST) instance before importing the application; this ensures that old entities are removed
- PUBLISH_TO_SC – If all tests have passed successfully, user has the option to publish the solution to Solution Central; this option is applicable for ThingWorx 8.5 or higher
- ARTIFACTID – Artifact ID for the solution published to Solution Central; this option is applicable for ThingWorx 8.5 or higher
- GROUPID – Group ID for the solution published to Solution Central; this option is applicable for ThingWorx 8.5 or higher
- PACKAGE_VERSION – Package version for the solution published to Solution Central; this option is applicable for ThingWorx 8.5 or higher
- MIN_PLATFORM_VERSION – Minimum ThingWorx Platform version for the solution published to Solution Central; applicable for ThingWorx 8.5 or higher
- SCRIPT_PATH – The path on the machine where the batch/shell scripts that Jenkins will execute are present
- TEST_RESULT_PATH – A path on the machine where Jenkins will save the result of the test executions. This path needs to include the provided CSS folder for the result to be properly formatted in an HTML page. **In the initial package, the CSS folder is a package of the PipelineScripts folder**
- DOCKER_PATH **–** The path where the ThingWorx Docker files are available, including the "*docker-compose.yml*" file
- APPKEY – The app key Jenkins will use to authenticate to ThingWorx. This app key is already provided in the DevOps project, but you can replace it. The app key needs to be the same on both the DEV and TEST environments. If you plan to use different

application keys, please create an additional parameter in the Jenkins pipeline and provide it as a command line parameter in the corresponding stage in Jenkins, when executing the batch or shell script.

Setting up a Docker Container

In some situations, you might want to use ThingWorx running on a Docker container for your automatic tests. In this situation, you need to follow these steps:

1. Make sure you have built the Docker image for ThingWorx, following the instructions available on the Help Center
2. In your Jenkins job configuration, make sure SETUP_DOCKER is set to 'true' and that the DOCKER_PATH points to where you have your ThingWorx Docker files.
3. On your DEV server, where you have initially configured your *GitBackup* Thing, make sure you add it to the DevOps project in ThingWorx and export the DevOps project entities (binary, universal export). Place this file in the *twdevopssetup* folder that is present in your *PipelineScripts* folder
4. Make sure that you copy the *twdevopssetup* folder from the *PipelineScripts* folder to the DOCKER_PATH. A bash script is present in this folder that will import the Git Extension and the DevOps entities
5. Open the bash script in edit mode and make sure all the variables have the correct values. Make sure you adjust the values for the Git Extension name, DevOps entities file name, ThingWorx host and port as well as the initial Administrator password

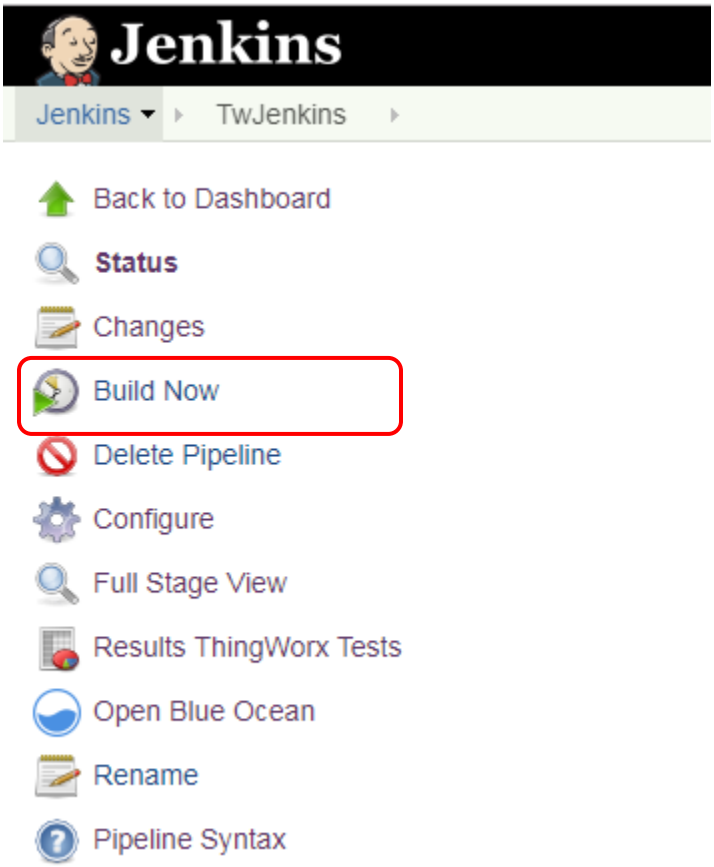
Please note that after the tests have run, the Docker container is not automatically deleted. If you plan to reuse the same Docker container for a new round of tests, make sure to set "SETUP_DOCKER" to false.

The current pipeline only includes a Linux Docker example, but follow the same instructions and adjust your pipeline accordingly with a "Build Docker" stage on Windows as well.

Build the Pipeline and View the Results

Once you have done all the previous steps, you are ready to build the Jenkins Pipeline, which will automatically export the application from DEV, push it into your git repository, import it on your TEST environment, and perform the tests you have previously developed.

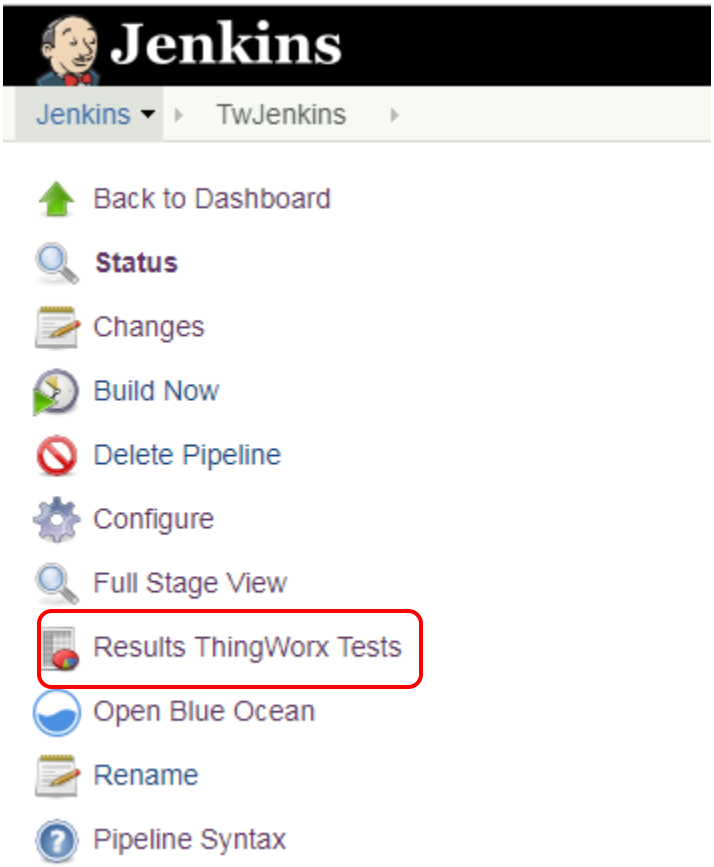
Navigate to the TwJenkins job and click on Build Now.



If an error occurs in the export application or import application stages, the Pipeline will fail. The ThingWorx error will be shown in the Console Output in Jenkins:

```
[Pipeline] { (ExecuteTests)
Stage "ExecuteTests" skipped due to earlier failure(s)
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: Error in Import Application stage Error ReferenceError: "ErrorOnPurpose" is not defined.
Finished: FAILURE
```

Once your build is finished, if successful, an HTML report will be generated with the results of the test that you can directly view in Jenkins (after the first build, a window reload is necessary):



Click on Results ThingWorx Test to view the report:

Back to TwJenkins index

RunAllTests

Comments	Expected	Inputs	Passed	Test Name	
passed	{"result":1100}	{"consumer3":100,"consumer2":2000,"consumer1":1200}	true	AveragePowerConsumption	Thu
passed	{"result":false}	{"powerConsumed":1200,"powerStored":2000,"powerProduced":5000}	true	EstEnoughPower	Thu

Each build will have a Results ThingWorx Tests HTML page displayed. On the project page you will be able to view the results from the latest build. To view additional information, go back and click on the build number and then click on Console Output. The Console Output is particularly important if you have issues with your build:

Jenkins › TwJenkins › #25

- Back to Project
- Status
- Changes
- **Console Output**
  - View as plain text
- Edit Build Information
- Delete build '#25'
- Open Blue Ocean
- Restart from Stage
- Replay
- Pipeline Steps
- Workspaces
- Previous Build

Console Output

```
Started by user unknown or anonymous
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\Program Files (x86)\Jenkins\workspace\TwJenkins
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (PackageApplication)
[Pipeline] bat

C:\Program Files (x86)\Jenkins\workspace\TwJenkins>C:\Users\cbadici\Documents\packageApp.bat http://172.27.163.209 de367f

C:\Program Files (x86)\Jenkins\workspace\TwJenkins>curl -X POST -H "Content-type: application/json" -H "appkey: de367fd7-
\"http://172.27.163.209\"}"  http://172.27.163.209/Thingworx/Things/DeploymentManagerThing/Services/PackageApplication
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100    37    0     0  100    37      0     30  0:00:01  0:00:01 --:--:--    30
100    37    0     0  100    37      0     16  0:00:02  0:00:02 --:--:--    16
100    37    0     0  100    37      0     11  0:00:03  0:00:03 --:--:--    11
100    37    0     0  100    37      0      8  0:00:04  0:00:04 --:--:--     8
```

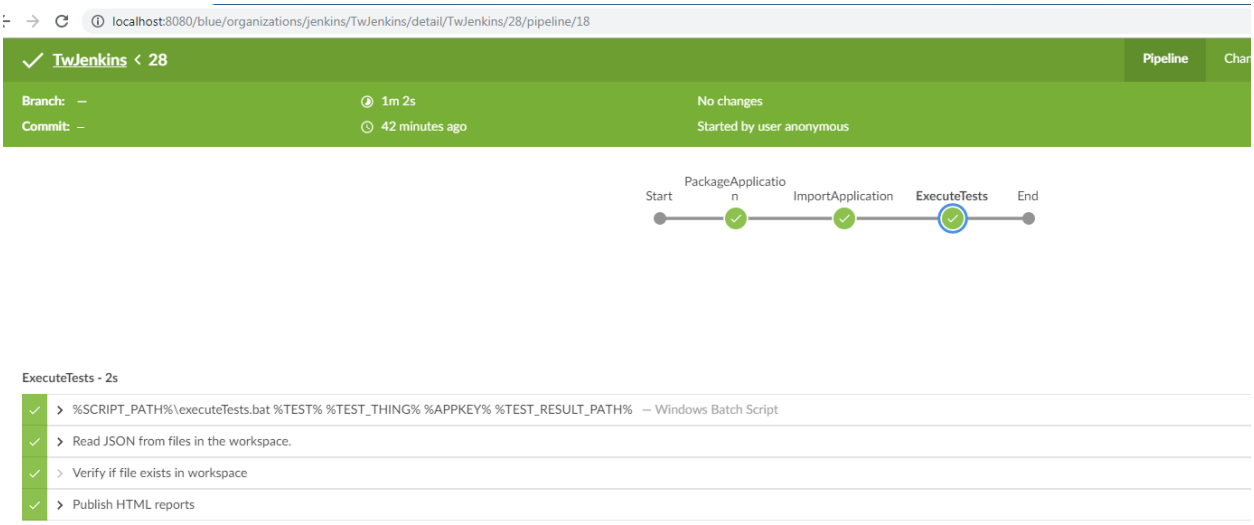
You can also click on the Pipeline Steps to view the different steps performed by the Jenkins pipeline and their execution time:

Jenkins › TwJenkins › #25 › Pipeline Steps

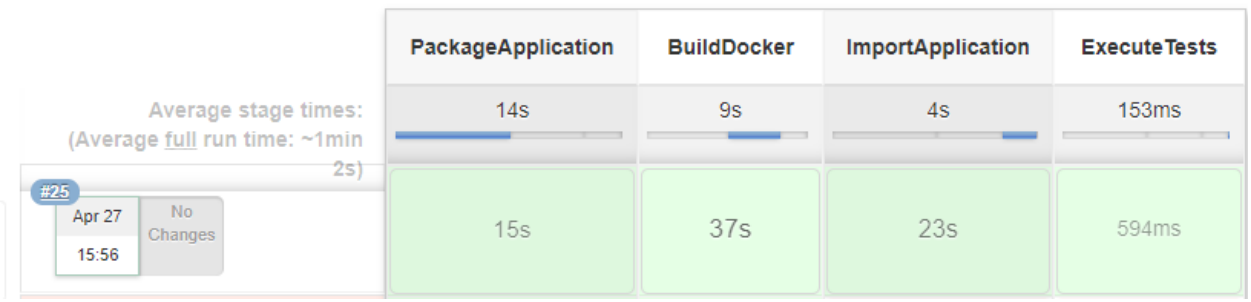
- Back to Project
- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#25'
- Open Blue Ocean
- Restart from Stage
- Replay
- **Pipeline Steps**
- Workspaces
- Previous Build

Step
Start of Pipeline - (41 sec in block)
Allocate node : Start - (41 sec in block)
Allocate node : Body : Start - (41 sec in block)
Set environment variables : Start - (41 sec in block)
Set environment variables : Body : Start - (41 sec in block)
Stage : Start - (25 sec in block)
PackageApplication - (25 sec in block)
Windows Batch Script - (25 sec in self)
Stage : Start - (14 sec in block)
ImportApplication - (14 sec in block)
Windows Batch Script - (14 sec in self)
Stage : Start - (1.2 sec in block)
ExecuteTests - (1.1 sec in block)
Windows Batch Script - (0.84 sec in self)
Run arbitrary Pipeline script : Start - (0.17 sec in block)
Run arbitrary Pipeline script : Body : Start - (96 ms in block)
Read JSON from files in the workspace. - (44 ms in self)
Publish HTML reports - (77 ms in self)

You can also track your build using the Blue Ocean UI. A prerequisite for this is installing the Blue Ocean plugin. If you have already installed this plugin, you can open the Blue Ocean UI to view information about your build:



If running a Docker container from the Pipeline, an additional Pipeline stage will appear, like in the picture below:



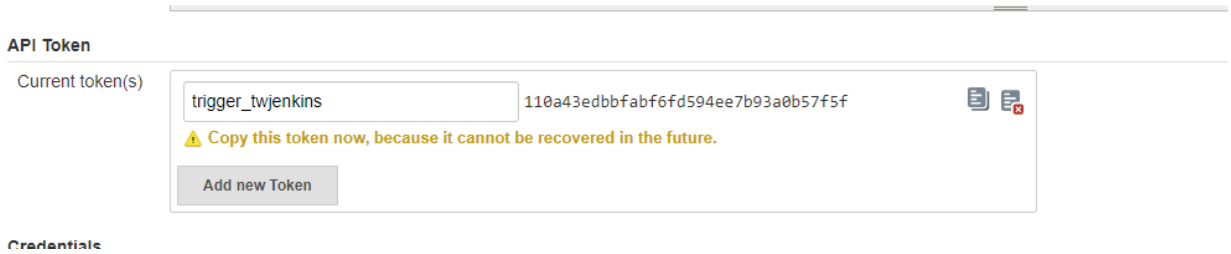
Triggering the Jenkins Pipeline Remotely

The Jenkins Pipeline can be triggered via the UI, by clicking on Build Now or remotely with an HTTP request. More information about triggering a Jenkins Pipeline remotely can be found here.

You can trigger the Jenkins Pipeline from ThingWorx, using the Content Loader Functions. An example snippet can be found below:

```
var result = Resources["ContentLoaderFunctions"].PostJSON({
      url: "http://192.168.189.135:8080/job/TwJenkins/build" /* STRING */,
      timeout: 300 /* NUMBER */,
      password: "110a43edbbfabf6fd594ee7b93a0b57f5f" /* STRING */,
      username: "admin" /* STRING */
});
```

The password is an access token that needs to be generated in Jenkins. In the upper corner where the username appears in Jenkins, expand the dropdown and click on Configure. Then go to the API Tokens section and generate a new token.



Troubleshoot the ThingWorx Jenkins Pipeline

You might have different issues when building the Pipeline for the first time. Please find a list of possible issues that you may encounter below:

1. **Missing plugins**

   Certain plugins are required to build this Pipeline successfully:
   - Pipeline
   - Pipeline Utility Steps
   - HTML Publisher
   - Permissive Script Security

2. **cURL command is not recognized**

   In this situation you need to install cURL commands on the machine where Jenkins is installed. You can find information about installing cURL here

3. **ThingWorx related issues**

   If an error is generated in one of the services that Jenkins calls in ThingWorx, please test the services outside the Jenkins environment to ensure they are working correctly. Also, ensure the custom tests are not producing any errors

4. **Script approvals**

   The first time you run the Jenkins Pipeline, an error might occur related to the script. Some method signatures require administrator approval. To do this, go to Manage Jenkins – In Process Script approvals and approve the method signatures.

   The following method signatures are used in the Pipeline and require approval:
   - method groovy.lang.GroovyObject getProperty java.lang.String
   - method java.io.File delete
   - new java.io.File java.lang.String

- staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods append java.io.File java.lang.Object
- staticMethod org.codehaus.groovy.runtime.ScriptBytecodeAdapter unaryPlus java.lang.Object

5. Certificate verification error in Jenkins

```
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
[Pipeline] }
```

To work with self-signed certificates, add the "*-insecure*" option to the cURL command in the batch or shell script. For example, for editing the Package App step, edit "*packageApp.bat*" or "*.sh*" and add "*-insecure*": *(e.g.* `curl -X POST -insecure`*)*

6. Failed to write to the *PipelineScripts* folder

The result of each stage is written to the *PipelineScripts* folder. Make sure that the Jenkins user has permissions to write to this folder.