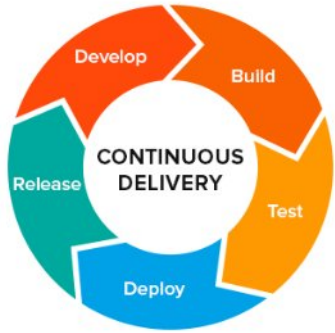




**Northwestern  
Mutual<sup>®</sup>**

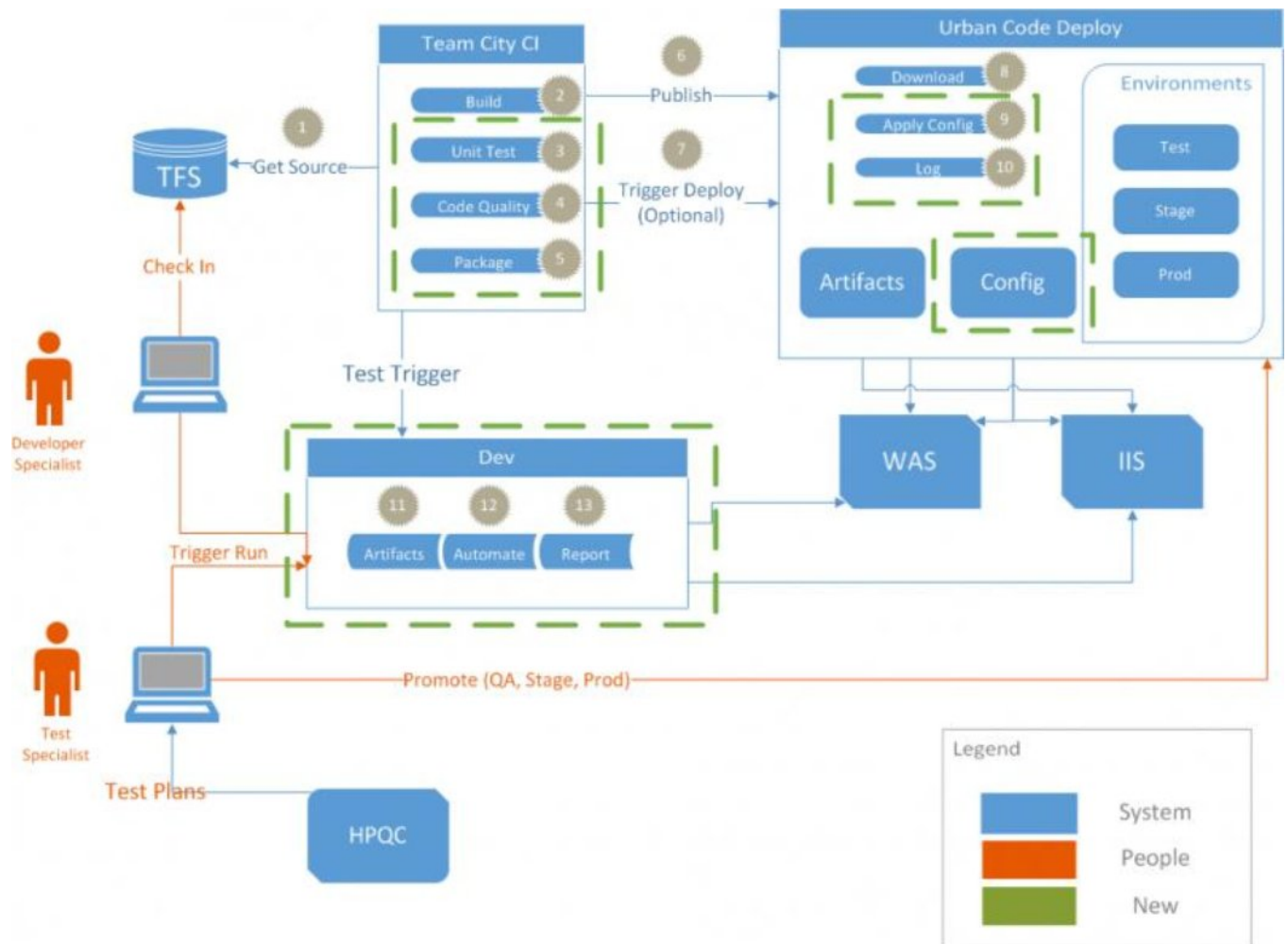
## Introduction:

This document has prepared to provide a comprehensive overview of the CD (Continuous Delivery) team processes in NM.



## Overview:

Continuous Delivery (CD) is a design practice used in software development to automate and improve the process of software delivery. Techniques such as automated testing, continuous integration (CI) and continuous deployment allow software to be developed to a high standard and easily packaged and deployed to test environments, resulting in the ability to rapidly, reliably and repeatedly push out enhancements and bug fixes to customers at low risk and with minimal manual overhead



## Continuous Delivery

What is the aim of continuous delivery?

- It makes every part of the process of building, deploying, testing and releasing software visible to everyone involved, aiding collaboration
- It improves feedback so that problems are identified, and so resolved, as early in the process as possible
- It enables teams to deploy and release any version of their software to any environment at will through a fully automated process

When do you know your organization is doing Continuous Delivery?

- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them

- You can perform push-button deployments of any version of the software to any environment on demand

## Principles of Software Delivery

- Create a Repeatable, Reliable Process for Releasing Software
- Automate almost everything
- Keep everything in version control
- If it hurts, do it more frequently and bring forward the pain
- Build quality in
- Done means released
- Everyone is responsible for the delivery process

## The Release Candidate

A fundamental concept within DevOps and Continuous Delivery is the 'release candidate', and is described as follows:

"While any change may lead to an artifact that can be released to users, they don't start off that way. Every change must be evaluated for its fitness. If the resulting product is found to be free of defects, and it meets the acceptance criteria set out by the customer, then it can be released."

Meaning every change is, in effect, a release candidate. The job of the Continuous Integration workflow is to disprove this assumption, to show that a particular release candidate is not fit to make it into production.

The word 'change' incorporates any modification to the 4 components: executable code, configuration, host environment and data.

## The Continuous Delivery Pipeline:

### Continuous Integration

Continuous Integration servers are responsible for detecting when a change has been committed to TFS and trigger the build process. CI server is then able to execute the build, unit test execution, code quality, and packaging steps of the process. The CI server can also

determine if a build should fail given certain criteria. Teamcity is the selected enterprise continuous integration server.

## Continuous Deployment

Continuous Deployment is responsible for accepting the packaged software from the CI, and taking whatever steps necessary to stand it up in an environment. This may include any necessary installation, server configuration, and dependency management. IBM UrbanCode has been selected as the enterprise deployment tool to replace APLUS and other deployment tools in the future.

## Automated Testing

Automated testing is responsible for validating the success or failure of a deployment. CTS is currently evaluating tools which will be used to perform various testing methods. Integration Testing is needed to have a fully mature pipeline. Currently there are QTP Integration Tests that are based in the UI layer. The modern method of doing this is to place as much of the business logic validation in the code layer, below the UI. There are many ways to do this, but ATDD has been brought in as an experiment to help improve Scrum Team Communication. A side benefit of ATDD is the integration testing it provides.

To form the most accurate assessment of a successfully deployed application, automated integration tests should be run against a deployed, running environment. As such, the current pipeline approach is to run these immediately after TeamCity receives confirmation from UrbanCode that a new application version has been successfully deployed. Using environment information provided by UrbanCode, TeamCity will then execute tests against the fresh environment.

## Continuing the Flow

After integration tests successfully validate the workings of the live environment, TeamCity will apply the appropriate status labels to the application component in UrbanCode, which will allow promotion to higher environments. If desired, TeamCity may even perform the promotion automatically, re-running the integration tests, and so on. This will allow the pipeline to theoretically permit a fully automated channel all the way from build to test to stage to production.

## Summary

1. (TeamCity) Build
2. (TeamCity) Unit test, static analysis
3. (TeamCity) Package
4. (UrbanCode) Deploy/Promote
5. (TeamCity) Integration test

6. (Optional) Go back to step 3

## Continuous Delivery Prerequisites:

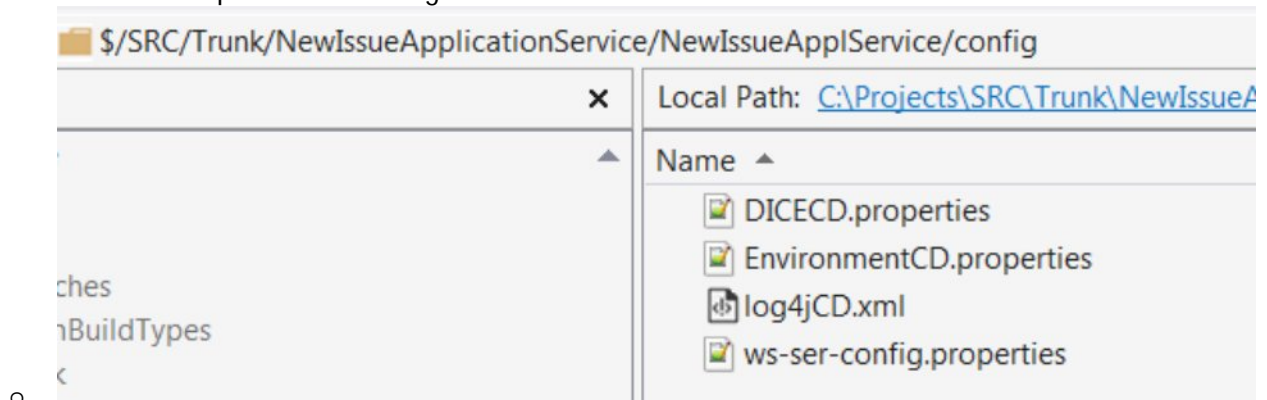
There are changes teams can begin to make or simply be aware of before being converted to the Continuous Delivery pipeline. There are some inconsistent practices in code structures and these instructions will help make our projects' code structures more uniform.

### Starting Off

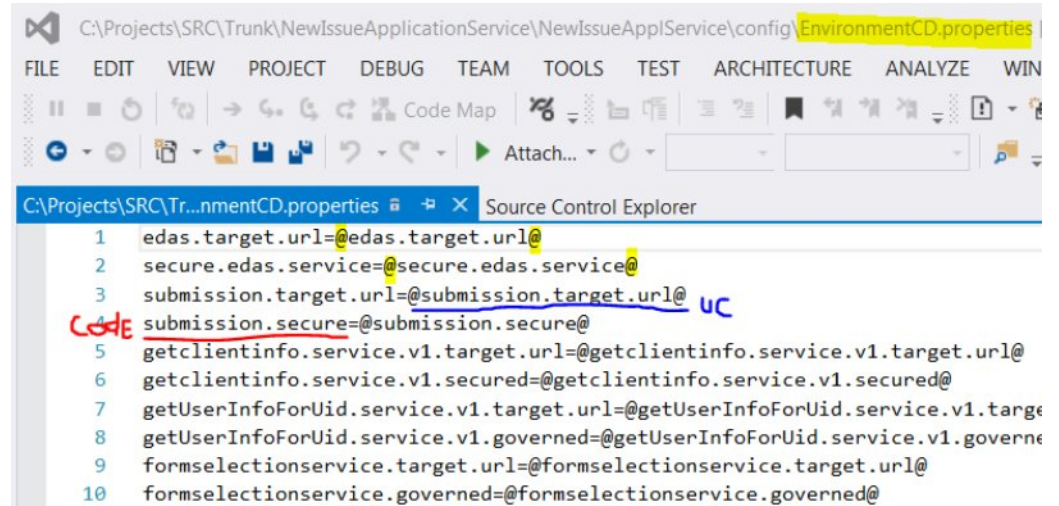
Available Platforms	.NET Application	JEE WAS Application	JSE Batch Application	JavaScript/Node/Grunt
Versions	Visual Studio 2013	Java 7 or 8		
Processes to Adopt or already be doing	Trunk Based Development	Unit Testing	Integration Testing	

## Java Specific Changes:

- Environment Dependent Values extracted into configuration files. This means no environment strings in .java files.
- Again, you should take environment dependent values out of all java files.
- Also, environment values should be consolidated to certain framework configuration files and not proliferated throughout the code in xml files not in the config folder.
- This is a big consolidation effort which will help manage your code in the CD tools, but should also help you moving forward and anyone new to the team by finding these values in one location. The suggested Java Code Structure change is documented below, but here is an example of that config folder:



- Environment.properties files - UrbanCode properties use a @parametername@ delimiter format. Within UrbanCode, you will provide the value for the Red Text code parameters, for each environment. So your code will be totally independent of environment specific values, and those values will be tokenized, or parameterized.



The screenshot shows a code editor with the file path `C:\Projects\Src\Trunk\NewIssueApplicationService\NewIssueAppService\config\EnvironmentCD.properties`. The file contains several lines of properties, each using a tokenized value (e.g., @edas.target.url@). A red bracket on the left side of the editor highlights lines 4 and 5, which are: `submission.secure=@submission.secure@` and `getClientinfo.service.v1.target.url=@getClientinfo.service.v1.target.url@`. The word "CODE" is written in red next to line 4. The text "uc" is written in blue next to line 3.

- Log4j.xml files - All environment specific paths can be tokenized following this type of pattern:



The screenshot shows an XML file with the following content:

```

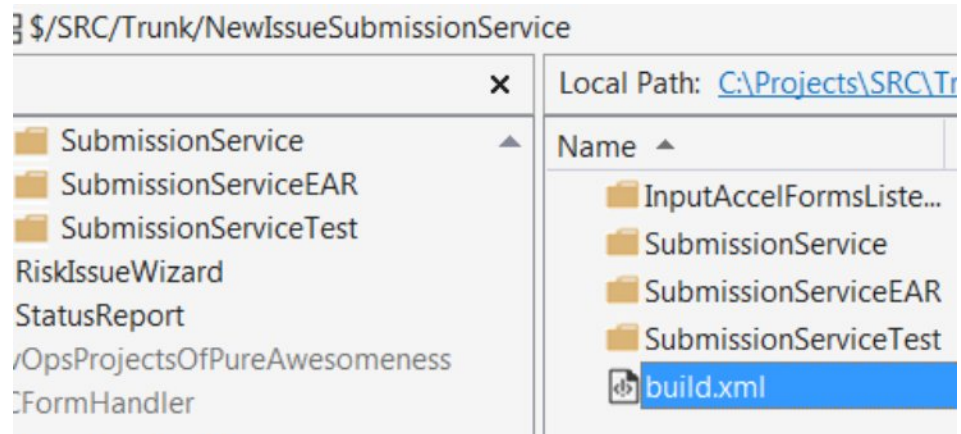
1 <?xml version="1.0" encoding="UTF-8" ?>
2
3 <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
4
5 <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="true">
6
7   <appender name="filelog" class="org.apache.log4j.DailyRollingFileAppender">
8     <param name="File" value="@log4j.filepath@/nbissappsvc@log4j.environment.suffix@.log" />
9     <param name="Threshold" value="debug" />
10   </appender>
11   <layout class="org.apache.log4j.PatternLayout">
12     <param name="ConversionPattern" value="%d{MM/dd/yyyy HH:mm:ss:SSS} | %c:%M:%L - %m%n" />
13   </layout>
14 </log4j:configuration>

```

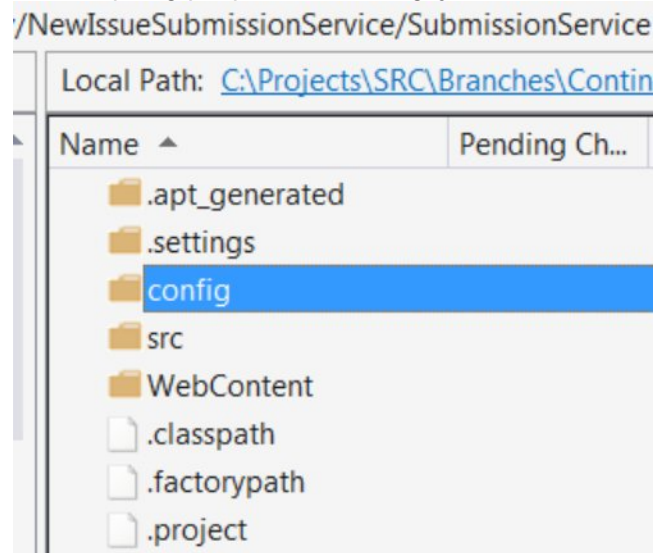
- Upgrade to SER 2.0 jar/dll (if web service consumer). This is to turn dynamic endpoint lookup on and off. This is huge for simplifying the effort to use UrbanCode.

## JEE Code Structure Changes:

- The Build.xml file will be at the root of the project, at the same level as the Project and EAR Folders. It will be drastically more simple looking, as a new ECC Build script has been written for the CD pipeline (ECC\_BUILD\_V3.0.xml). There will no longer be any build.properties files needed. JEE Project Configuration has its own page too.



1. ...
2. In the project folder, create a 'config' folder. Move your log4j.xml and Environment/Spring.properties files to that location. Understandably, some of the connection points in the current build process would need to be updated. This includes any environment specific variations (log4jTest.xml) of those files that might exist too. This Config folder is needed because ECC 3.0 build.xml will copy the Environment/Spring.properties and log4j files from this folder

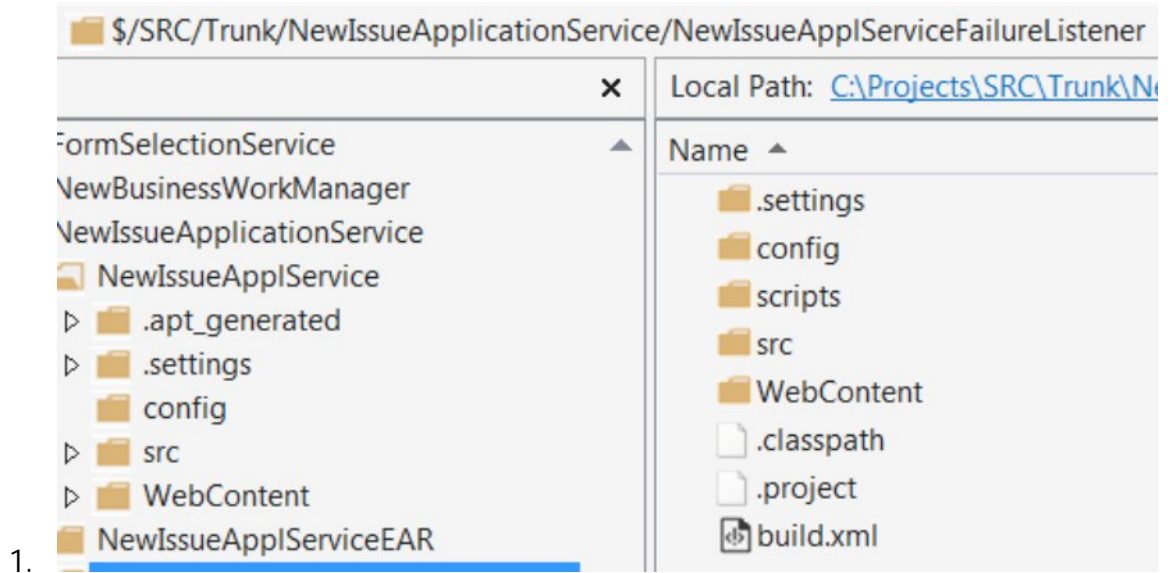


1.

## JSE Code Structure Changes

1. Do not "explode" library jars, and recompile with your source code. This technically works but is against best practices and increases the size of our code artifacts.
2. See the build.xml for JSE below, and that will be all that is needed in the root of the JSE application, like this:





## Example Build.xml

Please note the underscore naming convention, as an area you will need to update with your project's information. Look over all of the values provided, for example if your Java Source directory is "JavaSource" and not "src" make that change as needed.

JEE

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<project name="name_of_source_code_project_here" default="build" basedir=".">

  <!-- Project Properties -->
  <property name="applicationid"
value="name_of_application_usually_unix_id_and_first_word_of_ear_file" />
  <property name="appname" value="name_of_component_usually_next_word_of_ear_file" />
  <property name="virtual.host.name" value="field_host" />

  <!-- Project Convention Overrides -->
  <property name="java.src.directory" value="src" />
  <property name="test.project.srcPattern" value="T*.java" />

  <!-- Import ECCBuild -->
  <property name="eccbuild.path" value="//ho/dfs01/APPLS/Ant/TFS" />
  <import file="${eccbuild.path}/ECC_BUILD_V3.0.xml"/>

  <target name="war-pre-create" description="Hook to perform any actions on the source
directory before the war is created.">
```

<!-- Any source code pointers would go here if do not follow standard code structure, ensure build fails before adding here -->

</target>  
</project>

JSE

```
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<project name="name_of_source_code_project_In_tfs_here" default="build-jar" basedir=".">
```

```
<!-- Build Overrides for different structure -->  
<property name="isJarBuild" value="true" />  
<property name="jar.filename" value="{name_of_jar_created}.jar" />  
<property name="jar.main.class" value="com.nm.{follow_full_folder_name_to_class_name}" />  
</>
```

```
<!-- Project Convention Overrides -->  
<property name="java.src.directory" value="src" />  
<property name="test.project.srcPattern" value="T*.java" />
```

```
<!-- Import ECCBuild -->  
<property name="eccbuild.path" value="//ho/dfs01/APPLS/Ant/TFS" />  
<import file="{eccbuild.path}/ECC_BUILD_V3.0.xml"/>
```

</project>

## Gradle Example

- To use Gradle wrapper, you should have properly configured Gradle Wrapper scripts checked in to your Version Control.
- You should check in build.gradle, gradlew, gradlew.bat, gradle folders in the root of the project. Settings.gradle folder should also be checked in, if it is needed to your application.
- See the Example below:

Local Path: C:\TFS\SRC\Trunk\IPSRulesService		
Name ▲	Pending Ch...	User
gradle		
IPSRulesCommon		
IPSRulesData		
IPSRulesProcessor		
IPSRulesService		
IPSRulesServiceEAR		
IPSRulesServiceTest		
build.gradle		
gradlew		
gradlew.bat		
settings.gradle		

## ASP .Net Specific Changes:

### Supported Patterns

- Applications must have Web.config transformations setup for each environment
- Compilation will be done only one time with Release mode Rebuild
- The deployable artifact (all configs, aspx, ascx, javascript, etc.) should be output to the '[Project name]\obj\Release\Package\PackageTmp' folder. No source code files should be in this folder.
  - Refer to this document for details on how to make these changes: ASP.NET MVC How-To Guide v2.1
- Unit Tests are recommended
- Trunk-based or single branch development

### Supported Features

- MSTests
- Downloading/Restoring NuGet packages
- Code inspection through Resharper

### MS Integration Tests

#### 1.Integration tests need to be moved into their own project within a Visual Studio Solution

- Unit tests should not be included in this new project

## 2. Move all environment-specific values from the Integration Tests config file into UrbanCode

- The values in the config file should be replaced with @[UrbanCode property category].[property name]@
- In UrbanCode, select your application in the Applications tab. For each environment listed, you will need to add the appropriate values that need to be inserted into your config file. Click on an Environment, select the Configuration tab, and select "Environment Properties".
- For example: In the config file shown below, the "environment" value has been replaced with @Environment.environmentBase@.

```
<configSections>
  <appSettings>
    <add key="destinationLocation" value="@ComponentEnvironment.DestinationLocation@" />
    <add key="environment" value="@Environment.environmentBase@" />
  </appSettings>
</configSections>
```

Then in UrbanCode, a property named "environmentBase" has been added to the Environment Properties section in the Test environment. Since this is an Environment Property, you would also need to add this "environmentBase" property to the Stage environment.

The screenshot shows the UrbanCode web interface. The top navigation bar includes links for Dashboard, Components, Applications, Configuration, Processes, Resources, Calendar, Work Items, Reports, and Settings. The breadcrumb trail is: Home > Applications > SRC > Environments > Environment: Test. The main heading is "Environment: Test for SRC". Below this, there's a description "TEST01". A sub-navigation bar includes Resources, History, Calendar, Configuration (selected), and Changes. On the left, a sidebar shows "Basic Settings" and "Environment Properties" (selected). The main content area is titled "Environment Properties" and shows "Version 2 of 2". There are "Add Property" and "Batch Edit" buttons. A table lists the properties:

Name	Value	Description
environmentBase	Test	

At the bottom, it says "1 record - Refresh Print" and a pagination control shows "1 / 1".

## UrbanCode Properties

- Component properties are intended to be used for values that a component needs at all times – for example, public web service urls like google.com.

- Environment properties are meant to be used for values that are specific to an entire environment – such as a generic ID that your application's services will run with.
- Component Environment properties are for situations where a specific component needs a specific value for each environment. For example, if your component needs a database connection URL, that would be a good candidate for a component environment property

## TeamCity:

TeamCity is a continuous integration server produced by JetBrains and is an integral part of the Continuous Delivery pipeline.

### Getting Started

Although several groups have been managing their own independent installations of TeamCity - please engage with CTS to determine what technology best fits your automated build needs.

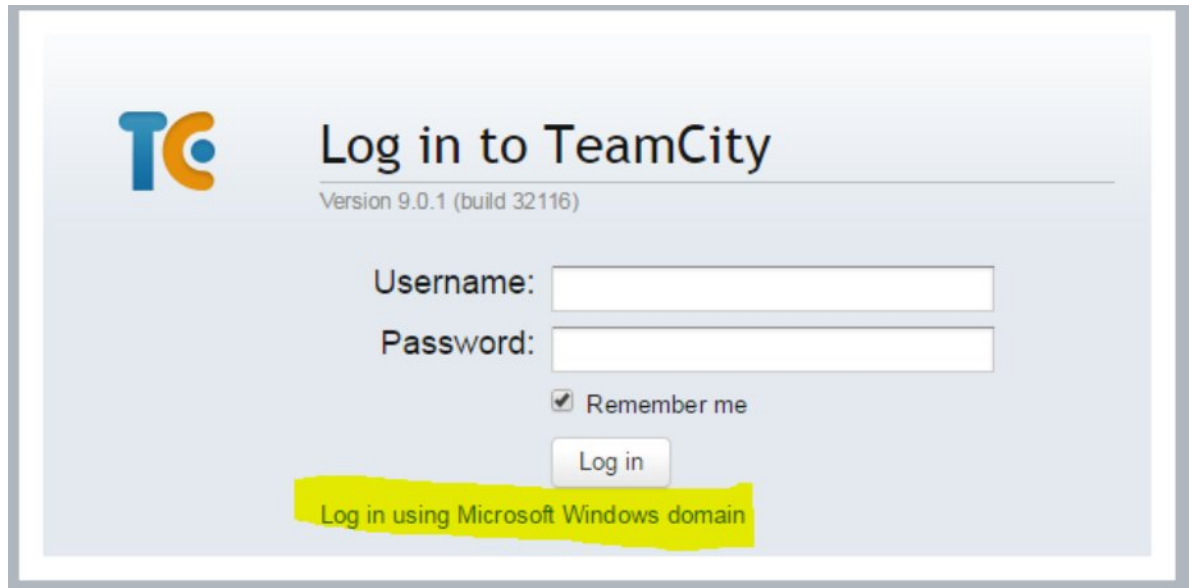
TeamCity First Time Walkthrough is just that, if you have never seen the website, not sure what's what, then this a good first step. This assumes you have been told you have access to TeamCity.

### Complete Documentation

TeamCity is a Continuous Integration Tool made by JetBrains. JetBrains has a tremendous amount of documentation available, [JetBrains' TeamCity Documentation](#) is highly recommended. Obviously Google and other search engines can return quality results as well in response to any question.

### Logging In

1. Navigate to <http://xxxx.xxx.com> (unless you are on the [Enterprise Pipeline](#)).
2. Click the blue link only, no need to enter any credentials or even press log in button:



3. If this is your first time, you'll first login to your "My Settings & Tools" Page.
  1. You can look under the "Version Control Username Settings" pane - make sure your lanid has XX\ before it. You can do that by clicking edit and typing it in before your lanid.
  2. Save changes.
4. To See Projects, click 'Projects' along the top of the window.
5. The Projects you have access to will now appear, this will be the default view each time you log in from now on.

### Starting With No Projects in TeamCity

Refer to the section below, note Steps 1, 5, and 6.

### Working With an Existing TeamCity Project

This will provide some basic navigation points, there is usually more than one way to navigate to an area in TeamCity.

TC Projects | Changes Agents 6 | Build Queue 0

CTS (CTS Applications and POCs)

Overview Change Log Statistics Current Problems Investigations Muted Problems

ATDD | Acceptance Test Driven Development

DotNetSpecflowCalculator |

#1.0.0.255567 Tests passed: 4

1.

2. The Application in this example is "DotNetSpecflowCalculator" clicking on the Name (DotNetSpecflowCalculator) will give you the build history for this configuration.
  1. If you wanted to change some Parameters in that specific build configuration, you would click "Edit Configuration Settings".

Run ... Actions Edit Configuration Settings

1.

2. If you go into this area, refer to [the Users Guild for Build Configuration Settings](#). Parameters would be the primary area to change of an existing build configuration.
3. Notice the Build Number Highlighted, this has been set up to match the TFS Changeset Number.

Source location: C:\Projects\CTS\Trunk\ATDD\DotNet\SpecflowCalculator

Changesets Labels

Changeset	User	Date	Comment
255567	NICK, MICHAEL	3/31/2015 8:11:32 AM	fix build

1.

4. The next item in that line is "Tests Passed" in this example, however if there are no tests, it will say "Success" or a Failure with description as to why it failed.
  1. Clicking this will show you further details on that build.
  2. There are several sub-tabs - but Build Log will be beneficial in seeing exactly what happened in the build and test process.
5. Click Edit Project Settings to add an entirely new build configuration, like if I wanted to add "JavaCucumberCalculator".

## Build Configurations

Build configurations define how to retrieve and build sources of a project. ?

+ Create build configuration

Name
DotNetSpecflowCalculator
JavaCucumberCalculator

1.

6. From this point you would follow the Users Guide for .NET/Java Quick Start

## Users Guide

This is the living document for using TeamCity set up by Centre and CTS for the Enterprise Model.

These links are all prefixed with "TeamCity" to prevent ambiguous pages, like a page titled "LDAP" etc.

- [TeamCity .Net Quick Start Guide](#)

## TeamCity .Net Quick Start Guide

### New Build Configuration

1. In TeamCity, select the project for your build configuration
2. On the upper right hand side of the page, click Edit Project Settings
3. Under Build Configurations, click Create build configuration
4. Enter Name and Description of your build configuration
5. Select .Net Web Project, in the "Based on template" drop down menu
6. Minimum parameters needed for a basic .Net Web Project include:
  - o Project\_Name
  - o Solution\_Path (this should be %Project\_Name%.sln if the sln is at the root of the checkout location, which is set during the Version Control Settings)
  - o UrbanCode\_ComponentName (This is the value that you give in UrbanCode for this component name)
7. Click OK



8. Click on Parameters, delete build.vcs.number
9. Configure Version Control Settings
10. Do not run the build yet, as the UrbanCode part needs to be set up next

Set up UrbanCode next

## TeamCity Java Quick Start Guide:

### JEE - IBM WAS Java Builds

1. In TeamCity, select the project for your build configuration
2. On the upper right hand side of the page, click Edit Project Settings
3. Under Build Configurations, click Create build configuration
4. Enter Name and Description of your build configuration
5. Select JEE Application, in the "Based on template" drop down menu
6. Minimum parameters needed for a basic build
  - o Project\_Name:
  - o UrbanCode\_ComponentName:
7. Click on Parameters, delete build.vcs.number
8. Configure Version Control Settings

### JSE - Oracle Java Builds

1. In TeamCity, select the project for your build configuration
2. On the upper right hand side of the page, click Edit Project Settings
3. Under Build Configurations, click Create build configuration
4. Enter Name and Description of your build configuration
5. Select JSE Application, in the "Based on template" drop down menu
6. Minimum parameters needed for a basic build
  - o Project\_Name:
  - o UrbanCode\_ComponentName:
7. Click on Parameters, delete build.vcs.number
8. Configure Version Control Settings

## TeamCity Enterprise Pipeline JEE Project Configuration:

Enterprise Pipeline JEE Java Projects need configuration changes to work with TeamCity and UrbanCode, here is the JEE Quick Start Steps. These changes can be done in parallel with the current project structures and should not impact a team. The only way these changes would pose an issue to a team, is if the files and folders these instructions explain to add, already exist

in the same location. Obviously, further considerations would be required at that point. This article is just part of the changes for the Pipeline, here is the page for Pipeline Practices.

## Intro

Currently JEE Applications are built with environment dependencies baked into the EAR. For example, a Stage EAR file will only work in Stage. This process removes that dependency from the EAR, so only the source code that project team develops will be in the EAR, the properties that determine environment are abstracted into variables, and they are assigned their environment values by UrbanCode.

Ant is being used to build Java applications at this time. Ant uses a build.xml file for its build direction. The amount of xml in the build.xml file should be greatly reduced and might be shocking for someone used to what a build.xml file might normally look like.

## Build.xml Steps

The template can be found in TFS at this location:

`$/COMMON_ENT/Trunk/scm_config/template_build.xml`

- Copy the file to the root of the JEE application, and rename to 'build.xml'. The build.xml file should be at the same level the project folder, earfolder, etc.
- Example project folder structure:
  - FormSelectionService
    - FormSelectionService
    - FormSelectionServiceEAR
    - FormSelectionServiceTest
    - build.xml
- Change 3 values in the build.xml.

project name	name_of_tfs_source_code_project_here
applicationid	name_of_application_usually_unix_id_and_first_word_of_ear_file
appid	name_of_component_usually_next_word_of_ear_file
java src directory	this the source files directory(generally 'src' or 'JavaSource')

- Check build.xml into TFS/Source Control Tool.
  - If you have the associated project already set up in TeamCity, you can test it at this point if you want, which means run the build in TeamCity.

New 'config' folder

In the above example the new config folder will be located in the service folder:

- - FormSelectionService
    - FormSelectionService
      - config
      - src
      - WebContent

The config folder will have the log4jCD.xml, EnvironmentCD/springCD.properties, and DICECD.properties properties files. It would be best to move all versions of those properties files to this new folder as well. If you have environment specific ws-ser-config files you can put those here as well, but that file requires additional care, seen in the next section.

This section and the next are also covered in the page on replicating the config settings in UrbanCode.

## WS-SER-CONFIG Location

The ws-ser-config.properties file should be located in your projects WebContent/WEB-INF/classes folder. In the event you do not have a classes folder in this WEB-INF folder, create it and copy your properties file there. This is part of the new universal configuration for all Java Projects (JEE and JSE).

## Next Steps

In order to make a seamless transition to UrbanCode, a few other steps are needed to get environment configuration values, found here:

## Import Ant Settings into UrbanCode:

### UrbanCode Import Ant Settings

Importing Ant Settings is an important step in using UrbanCode and TeamCity successfully. The first step of this requires running a Powershell Script locally on your computer, then checking into TFS. After that point, it depends on what else is in place for the next steps. If the TeamCity build configuration is in place, and the component has been initially set up in UrbanCode, then running the build in TeamCity will do transformations in UrbanCode, creating parameters for a

component, but the values for those parameters still need to be copied into UrbanCode. This page will help you navigate this process.

### Ant Settings Replicated in UrbanCode

Individual component environment settings can be viewed by navigating to the Configuration tab in UrbanCode. The Environment Property is declared in the root of the component, ie nbformselsvc. Then in the sub section, you will notice the environments, click on the environment, and that property will be set there for that environment.

Example:

- nbformselsvc - has parameters defined
  - Prod - each environment has a text box to fill in the value for the parameters.
  - Stage
  - Test

### Code Structure (in TFS)

The new format for all projects with Ant builds, is to add or use a "config" folder in the projects main project Code folder. This should not be at the same level as the build.xml file, when converting a code base to run in the Enterprise TeamCity, which is discussed here.

Example:

- JEE
  - Generic Wizard
    - STePFWizWeb
    - config
- JSE
  - DocUploadJob
    - bin
    - config
    - JavaSource/src
    - lib

In that config folder, a file for each environment should exist, to set a common standard. This is not needed as many projects will have these files in a few different locations, based on how that team set them up. The powershell script will handle putting the needed files in this folder to meet the needs of the Continuous Delivery pipeline. One last point is that these environment specific files are only needed if the team needs to revert to a manual build process for, what would be, an unexpected reason.

For example, log4jProd.xml/log4jTest.xml/etc will be present in the code structure somewhere, but could be located here for uniformity. The list of files that will follow this format identified are:

- log4j.xml
- Environment.properties or Spring.properties
- DICE.properties

The build pipeline will automatically update and place ws-ser-config.properties in its proper place, as long as it exists in WebContent/WEB-INF/classes/ws-ser-config.properties. If for whatever reason, the classes folder does not exist, create it and place the file there.

### Update Tokens via PowerShell Script

The process of parameterizing Ant settings and updating them into UrbanCode can be a lot of manual work. This script makes the process easier. In TFS,

```
$/COMMON_ENT/Trunk/deployment/UpdateTokens.ps1
```

script can help transform Ant settings. Note, this is a local process on your computer, so you'll run this and then have to check in the output files for the specific project to TFS.

### PowerShell Set Up

To run this, you will have to ensure you have RemoteSigned ExecutionPolicy. You can do this by first looking at your current settings with 'Get-ExecutionPolicy -List'. If that list returns all 'Undefined', Run this command: 'Set-ExecutionPolicy -ExecutionPolicy RemoteSigned' and then you'll have to confirm with a 'Y'. You can then rerun 'Get-ExecutionPolicy -List' to validate it has changed.

### 4 Parameters to Run Script

Note, quotes are needed when entering in Powershell!

- Name for .properties usually "spring" or "Environment"
- File Path to project's spring.properties or Environment.properties - ex:  
"C:\Projects\SRC\Branches\Continuous  
Delivery\NewIssueApplication\NewIssueAppIService\src"
- File Path for the above mentioned Config folder - ex:  
"C:\Projects\SRC\Branches\Continuous  
Delivery\NewIssueApplication\NewIssueAppIService\config"
- Extension, usually just "" is needed.

## Example

```
C:\Projects\COMMON_ENT\Trunk\deployment\UpdateTokens.ps1 "Environment"
"C:\Projects\SRC\Branches\Continuous
Delivery\NewIssueApplicationProd\NewIssueAppIService\src"
"C:\Projects\SRC\Branches\Continuous
Delivery\NewIssueApplicationProd\NewIssueAppIService\config" ""
```

## Output from Running Script

in the above config folder, two files are generated and placed in the config folder:

- EnvironmentCD.properties or springCD.properties
- UrbanCodeSettings.txt

These files need to be added/checked into to TFS now for the applicable project.

## Updating Urban Code

The check in will kick off the build in TeamCity, which will then create the settings in UrbanCode, assuming that the component exists in UrbanCode already. You will need to use the values in the UrbanCodeSettings.txt, to copy and paste into the corresponding environment for that component in UrbanCode.

- If the component is not configured in UrbanCode, do that now:

## TeamCity Build Configuration:

### General Settings

Field	Example
Name	Work Manager
Build configuration ID	<TeamCity auto property>
Description	<optional> Build Config for Src's Work Manager (.NET) code
Build number format	%Build_Number%.%build.vcs.number%
Build counter	<TeamCity auto property>
Artifacts paths	%Package_Path% => %Artifact_PublishName%

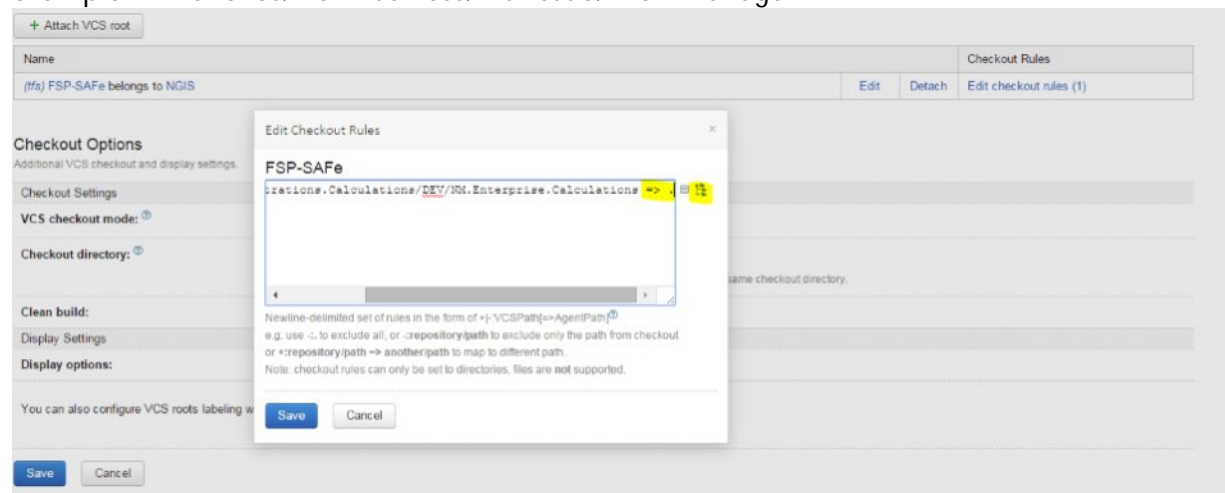
Build options                      No need to change this setting

## Version Control Settings

- Click button "Attach VCS root". The version control system chosen at this time is Team Foundation Server (TFS). The VCS root can be declared at the root project or the individual project level (like SRC, NGIS and etc). The actual reference to the TFS is stored in TFS.url parameter defined at the root project level. The parameter is then inherited to each project and build configuration. Do not overwrite the value in projects and build configurations.

## If Existing Project

- Select appropriate name from "Attach existing VCS root"
- Example: NMCollection/SRC, you would select SRC.
- Click attach
- Click "Edit checkout rules (0)"
- In textbox, type path to actual code project you want to checkout with this build configuration, follow example format +:PATH => .
  - example: +:Branches/NewBusiness/RiskIssue/WorkManager => .



- You can click the right folder structure icon to navigate directly to the project location.
- Click Save

## If New Project Family

- Click Create new VCS root
- Type of VCS: Team Foundation Server
- Click Create
  - VCS Root Name:

- URL
- Repository URL: %TFS.Url%
- Root: \$/SRC (Only Top Group Level Name is needed here, actual path to code is defined in next steps)
- Username: <not needed>
- Password: <not needed>

VCS Root	
VCS root name: <sup>?</sup>	FSP-SAFe <small>A unique name to distinguish this VCS root from other roots.</small>
VCS root ID: <sup>?</sup>	Ngis_FspSAFe <a href="#">Regenerate ID</a> <small>VCS root ID must be unique across all VCS roots. VCS root ID can be used in parameter references to VCS root parameters and REST API.</small>
TFS Settings	
URL: *	%Tfs.Url% <small>URL format: TFS 2010+: http[s]://&lt;TFS Server&gt;:&lt;Port&gt;/tfs/&lt;Project Collection Name&gt; TFS 2005/2008: http[s]://&lt;TFS Server&gt;:&lt;Port&gt; Visual Studio Online: http[s]://{account_name}.visualstudio.com/DefaultCollection</small>
Root: *	\$/FSP-SAFe <small>TFS path to checkout. Format: \$/path.</small>
Username:	 <small>⚠ Leave blank to use the TeamCity server user account. To login to hosted TFS use "ALTERNATE AUTHENTICATION CREDENTIALS" and add "##LIVE##\\" prefix to your username (email address) <sup>?</sup></small>
Password:	 <small>⚠ Leave blank to use the TeamCity server user account</small>
Agent checkout:	<input type="checkbox"/> Enforce overwrite all files
Changes Checking Interval	
Checking interval: <sup>?</sup>	<input checked="" type="radio"/> use global server setting (60 seconds)  <input type="radio"/> custom: 60 seconds

- 
- Click Test connection
- If Success, click Create, otherwise review values typed above.
- Click "Edit checkout rules (0)"
- In textbox, type path to actual code project you want to checkout with this build configuration, follow example format +:PATH => .
  - example: +:Branches/NewBusiness/RiskIssue/WorkManager => .
- Click Save

After it is Created

- You can edit the base path, or the checkout rules by following the related links:





## Build Steps

TeamCity is very modular, the Build Steps portion of the Config is a great example of how modular it can be. Many build steps come out of the box, but many other tools can be manually added for use. See "Custom" for an example of custom, manually added build steps.

- Click Add build step, or you can also click auto-detect build steps.

## .NET

- NuGet Installer
- Visual Studio (sln)
- MSBuild
- MSTest

## JAVA JSE

- Ant
- Gradle
- IntelliJ
- NAnt
- NUnit

## JAVA JEE

- Ant
- Gradle
- IntelliJ
- NAnt
- NUnit

## Unix

- Command Line

## Apple

- XCode Project

## Other

- Command Line
- Powershell

## Custom

- Publish Component to UrbanCode
- Run Specflow Tests [ATDD](#)

## Triggers

- Click Add New Trigger
- VCS Trigger - builds each time code is checked into TFS from path provided for this build
- Other options provided by TeamCity, day/time of day being another often used Trigger

## Failure Conditions

- These should be decided by the team, and are fairly self-explanatory. Leaving this as default will provide an above adequate start for a build configuration.

## Additional Failure Conditions

- Some steps like the Command Line step used to execute a task, may send back a failure, but TeamCity is only judging if it was able to execute the step, not the result of the step. If you cannot send "TeamCity friendly" text back to the build log, you can add a manual failure here. You can fail a build on specific text in the build log.
- TeamCity Build Script Interaction, ie TeamCity friendly text, can be better understood and more eloquently fail or pass the build. It basically follows a `##teamcity[ ]` format, with many different parameters able to be provided inside of the `[ ]`.

## Build Features

- Customizable aspects of the build configuration.

## Dependencies

- Can tie the build configuration to another build configuration. If something needs to occur in order, this is a good option. Additionally, if you have Integration Tests that need to run after the developer has run unit tests, you can create a separate integration test build configuration, and depend it upon the developers actual code passing, with unit tests etc.

## Agent Requirements

- This should be default. This is needed because the server that builds your code, will need to have the appropriate tools to perform that task. For example, if you have .NET code, then the server needs Visual Studio installed as well. This step has already been done, but a context for any future considerations, if a new tool is selected.

## Build LifeCycle

During a build, the order of operations is as follows: Run Build Steps, Publish Artifacts to TeamCity (so it can be available for download/view from "Artifact" tab") and finally run Build Failure Conditions. Note that if build definition contains steps to copy files and folders out to a different network location or tool during the build steps, the build failure as a result of a Build Failure Condition will not revert back the copy steps. Therefore, build definition needs to be setup to validate artifact before it is copied out (if validation is needed). Otherwise, REST api is available to download the artifacts (push vs pull).

## Meta-Runners

Repeatable and reusable build steps can be extracted into a Meta-Runner. TeamCity has the following Meta-Runners in the root project, that can be used by all build definitions if the build steps are applicable:

- CompareBuildArtifactSize - use this meta-runner to compare publish artifact size (artifact to be published to UrbanCode) from current build to last successful build's publish artifact size. If the difference does not fall within the acceptable range (+/- in %), the build will fail. Currently, the JEE, JSE and .NET templates have this meta-runner as a build step immediately before publishing artifact to UrbanCode. The acceptable range can be overwritten at the build definition level as it is an inherited parameter. Generally, this will be configured as the step immediately before PublishComponentToUrbanCode; however, it does not have to be followed by the PublishComponentToUrbanCode; it can run on its own.
- PublishComponentToUrbanCode - use this meta-runner to publish artifact to UrbanCode. Generally, this will be configured as the last build step

## TeamCity NPM/Grunt Steps:

TeamCity has a few different ways that a team using Node.js or javascript can execute NPM and Grunt build steps. You can use the Command Line or Powershell steps to execute basic commands. There was a TeamCity supported package for Node.js and Grunt installed into TeamCity to make these steps even easier than they already are in most cases.

### Build Steps to Add

"Working Directory" is important if you are using a .NET solution and your gruntfile is deeper in your checkout directory that TeamCity is working with. For every JavaScript only project worked with thus far, the checkout directory has been the same level as the working directory and in that case, this field can be left blank. But if your .NET solution has the JavaScript project a folder deeper, that should be specified here. The build steps should be added in the following order.

### Node.js NPM

<b>Runner type:</b>	<div>Node.js NPM</div> <div>Starts NPM</div>
<b>Step name:</b>	<div></div> <div>Optional, specify to distinguish this build step from other steps.</div>
<b>Execute step:</b> ⓘ	<div>If all previous steps finished successfully</div> <div>Specify the step execution policy.</div>
<b>npm commands:</b>	<div><b>Commands:</b></div> <div>install</div> <div>Specify npm commands to run</div>
<b>Execution</b>	
<b>Path to Node.js NPM:</b>	<div></div> <div>Specify path to Node.js NPM executable. Leave blank to use agent-installed one</div>
<b>Working directory:</b> ⓘ	<div>%Web_Project_Name%</div> <div>Optional, set if differs from the checkout directory.</div>
<b>Additional command line parameters:</b>	<div><a href="#">Expand</a></div> <div>Enter additional command line parameters for npm. Put each parameter on a new line</div>

## Grunt

<b>Runner type:</b>	<div>Grunt</div> <div>Executes Grunt tasks</div>
<b>Step name:</b>	<div></div> <div>Optional, specify to distinguish this build step from other steps.</div>
<b>Execute step:</b> <sup>?</sup>	<div>If all previous steps finished successfully</div> <div>Specify the step execution policy.</div>
<b>Grunt:</b>	<div>NPM package from project </div> <div>Specify whether you like to use system-wide or project's npm installed grunt</div>
<b>Grunt File:</b>	<div></div> <div>Specify grunt file path relative to checkout directory. Leave blank to use Gruntfile.{js,coffee}</div>
<b>Grunt Tasks:</b>	<div><b>Commands:</b></div> <div>%Grunt_Commands%</div> <div>Specify grunt tasks to run (new-line separated)</div>
<b>Working directory:</b> <sup>?</sup>	<div>%Web_Project_Name%</div> <div>Optional, set if differs from the checkout directory.</div>
<b>Additional command line parameters:</b>	<div><a href="#">Expand</a></div> <div>Enter additional command line parameters for Grunt. Put each parameter on a new line</div> <div>--teamcity.properties.all and --teamcity.properties parameters are added implicitly pointing to JSON files with all TeamCity properties and</div>

## TeamCity Parameters:

All TeamCity builds will have a mixture of variables. The mixture means build level, system, and environment level definitions; as well as "out-of-the-box" variables TeamCity provides. In addition, Centare has helped create .Net and Java Templates for use, and any build configuration can be turned into template or disassociated from a template.

## Administrator Guide:

## TeamCity Installation

### Setting Up Team City

Refer to installation instructions for TeamCity Server:xxxxxxx.com

Now on TeamCity Server (head) you can navigate to Agent tab, and click Agent Push to begin setting up agents

Click Install Agent, a pop up will show, you need the full server name ( (XXXX).com) or ip

address

If you need to enter credentials, you will likely have to use generic id

## Install Unix Agent

First, a generic id is needed.

Next you need SESU Access to an appropriate location

The link is the form to request SESU access: xxxxxxxxxxxx.com/

In the form, select radio button: "SESU to Other User Authorization"

NOTE: Agent push (the easy way) does not work, because it cannot find the platform name, either because the default platform name is too much text, or it is not a default text expected.

Once you have access to the Unix File location - Go to TeamCity instance, and click Agents, then

Install Build Agents, a mini javascript pop up gives a few options, select the zip file distribution.

Unzip that download.

copy the unzipped "buildAgent" folder to an area on the unix server that is accessible.

For the first time, I used: W:\unixhome\dv19scmt\test\buildAgent - as the folder write options were limited for me.

using the windows view, find a file in that buildAgent folder -

buildAgent/conf/buildAgent.dist.properties and rename it buildAgent.properties.

Then open that file in notepad or similar program

there are two variables that need to be set: serverUrl and name.

for cts I set serverUrl: xxx.com and name=Unix.Cadaman

save and close

then in the unix command line, navigate to that folder buildAgent/bin

run command: "install.sh"

then "agent.sh start"

You should be set up and ready to go, if you go to the teamcity UI, the agent should appear, but not active, go through the steps to assign

it to a Unix Pool and activate the agent.

Team Explorer Everywhere is also needed:

<http://www.microsoft.com/en-us/download/details.aspx?id=30661>

Follow a similar unzip process and again copy that folder to an appropriate location, I used:

W:\unixhome\dv19scmt\test\TEE-CLC-11.0.0

navigate to that folder within the unix command line and simply type "tf" and the install should happen.

## TeamCity LDAP:

Administration - Authentication, Load, LDAP - use basic settings.

Will need a ldap-config.properties file in \ProgramData\JetBrains\TeamCity\config

## External Links

## [TeamCity's LDAP Documentation](#)

### TeamCity Upgrade:

#### Upgrade Process

The upgrade is essentially a simple download of the new version and installing it as if it were new. The install is smart enough to know a version is already installed and can either keep the old version or delete it. If you delete it, you do not lose your build configurations, you just lose the old version of TeamCity.

#### Notes on Upgrade

- Log on to head server vn
- Download updgrade exe
- Backup - (Admin link) - click backup
- Run .exe file, everything looked "normal", press "next" in wizard through different views
  - Note: Since it is essentially an installation, you can change set up settings at this point, or it is noted that might be a way of changing settings (GenericId used to run Service for example)
  - Also since Generic IDs are used to run the server and agents, ESAT will be needed to re-enter the passwords during the install and any upgrades.
- When you launch teamcity again, a maintenance error screen may appear. You need to click "show me details"
- The head server will send a command to the agents to upgrade and it worked successfully
- If any issues occur, it is likely a common issue that can be resolved by going to [TeamCity's Documentation](#)

#### External Links

[TeamCity's Upgrade Documentation](#)

### TeamCity to UrbanCode

#### Introduction

There is a handshake that happens from TeamCity to UrbanCode, and the communication goes in both directions.

## TeamCity

### Build Configuration Parameters

Each TeamCity configuration, automatically inherits three Parameters, two\* of which never change.

UrbanCode.ApiKey*	%secure:teamcity.password.UrbanCode.ApiKey%
UrbanCode.Url*	UrbanCode_ComponentName
UrbanCode_ComponentName	{name that for component as it will appear in UrbanCode}

## Build Step

A meta-runner was created to make this step simple. (A meta-runner is a common step that as been abstracted so that parameters can be entered for changing values). This should always be the last step in a TeamCity build for deploy-able source code. This step is also part of a template and so these values should never change, nor can you change them. In order to change them, a TeamCity administrator would need to change them, and that would only be needed if the process was rewritten for some reason.

Artifacts Path	%teamcity.build.checkoutDir%\%Package_Path%
Component Name	%UrbanCode_ComponentName%
Component Version	%build.number%
Configuration File Path	%teamcity.build.checkoutDir%\%Package_Path%

## Fine Details



The above build step is a convenient way of using a PowerShell script, in conjunction with the udclient. The udclient is the UrbanCode Client .jar file that allows for the API connection to UrbanCode remotely. All of the files related to this process are found in TFS, at this location:

`$/COMMON_ENT/Trunk/deployment/DeployPackage`

The settings above are injected as parameters for the Powershell script, named DeployProject.ps1. The udclient client is clearly in the folder of the same name at the above listed TFS path.

## TeamCity MetaRunners

A TeamCity metarunner is a custom build runner that TeamCity users can write. A custom build step that is used more than once can be turned into a metarunner and be exposed back to build configurations. For general information on metarunners and how to create them, refer to JetBrains website: <https://confluence.jetbrains.com/display/TCD9/Working+with+Meta-Runner>.

### Standards

- All source code for the metarunner should be stored in an independent file, as opposed to the source code being part of the metarunner definition.
- All source code for the metarunner should be unit tested using appropriate framework. For example, if PowerShell was chosen as the scripting language of choice for the task, then Pester unit tests and files should accompany the source
- All source code and corresponding tests for the metarunner should be checked into source control under `$/COMMON_ENT/TeamCity/MetaRunner`
  - A build definition is currently targeting this location from source control. If an update is made to this folder, the build definition will get latest from source control, run unit tests (currently only Pester unit tests are setup to run) and distribute the source to agents if all tests pass.
- Metarunner name should be prefixed with "NM\_" to indicate it is a custom written build runner
- Additional agent requirements should be explicitly mentioned in the metarunner definition.

### Sample metarunner definition

Source:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <meta-runner name="NM_SikuliRunner"> Prefix name with "NM_"
3   <description>To run sikuli test</description>
4   <settings>
5     <parameters>
6       <param name="Sikuli_Jar" value="" spec="text orderNumber='1.0' validationMode='not_empty' label='Sikuli Jar File' display='normal'" />
7       <param name="Sikuli_TestFile" value="" spec="text orderNumber='2.0' validationMode='not_empty' label='Sikuli Test File' display='normal'" />
8     </parameters>
9     <build-runners>
10      <runner name="Sikuli_TestExecution" type="jetbrains_powershell"> This specific metarunner implementation is in
11        <parameters> PowerShell
12          <param name="jetbrains_powershell_bitness" value="x86" />
13          <param name="jetbrains_powershell_errorToError" value="true" />
14          <param name="jetbrains_powershell_execution" value="PS1" />
15          <param name="jetbrains_powershell_scriptArguments" value="-SikuliJar %Sikuli_Jar% -SikuliTest %Sikuli_TestFile%" />
16          <param name="jetbrains_powershell_script_file" value="D:\Tools\Enterprise\MetaRunner\SikuliRunner.ps1" /> Reference to the file on TeamCity
17          <param name="jetbrains_powershell_script_mode" value="FILE" /> agent
18          <param name="teamcity.step.mode" value="default" />
19        </parameters>
20      </runner>
21    </build-runners>
22    <requirements>
23      <exists name="env.JAVA_HOME" /> Additional agent requirements
24    </requirements>
25  </settings>
26 </meta-runner>
27
28
```

Sample build step using metarunner

## New Build Step

Runner type:

NM\_SikuliRunner

To run sikuli test

Step name:

Optional, specify to distinguish this build step from other steps.

Sikuli Jar File

sikuli-slides-1.5.0.jar

Value must be specified

Sikuli Test File

SampleRegressionTest.pptx

Value must be specified

Show advanced options

Save

Cancel

## IBM UrbanCode:

IBM UrbanCode is a continuous deployment server produced by IBM and is an integral part of the Continuous Delivery pipeline.

### Getting Started

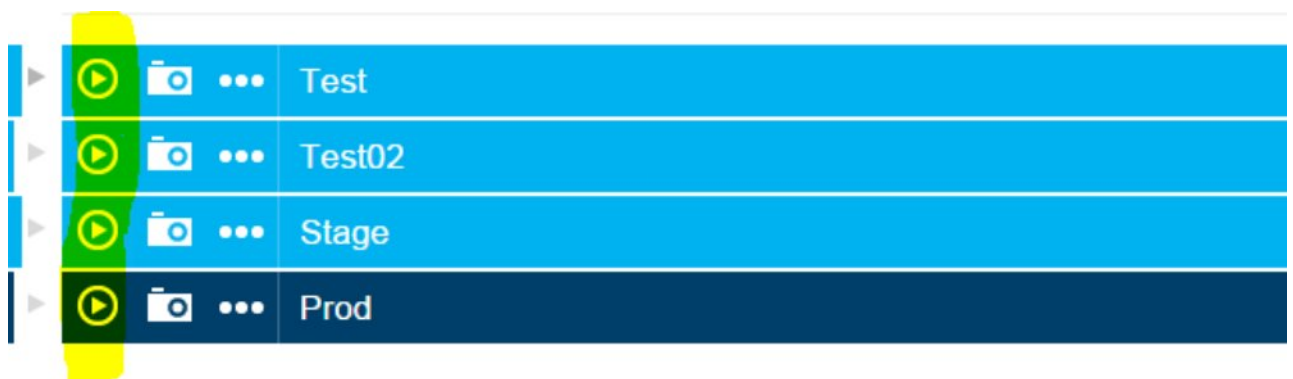
This tool provides the last step in the software delivery cycle, or continuous delivery pipeline. It has access to the servers that host our applications and can deploy code to those servers, whether it is the TEST, STAGE, or PRODUCTION environments. By containing the code in the pipeline, every piece of code that makes its way from developer machine to the production servers can be tracked and precisely identified.

### UrbanCode Deploy

Deploy in IBM UrbanCode is handled a few different ways. This will walk you through how to deploy an application through the tool itself. Meaning this does not cover an auto deploy as part of the continuous deployment model.

### Steps

1. Click on the top level "Applications" tab
2. Select your Application (SRC for example).
3. You will now see the different environments listed in about the middle of the page, click the "Play" button for the environment to which you want to deploy. Note this is the second icon from the left, the first triangle is a carat to expand the environment.



4. After you click the "Play" button, a pop up will appear.
5. Select "Deploy" for the Process, it is the only option but must be selected.

**Run Process on Test**

**Only Changed Versions** ☒

Process \* Deploy

Select a snapshot, or choose versions for individual components.

Snapshot

**Component Versions**

Versions 0 selected [Choose Versions](#)

Schedule Deployment? ☐

Description

**WARNING: Not all required properties have values.**  
[View missing properties...](#)

**Submit** **Cancel**

1. "Only Changed Versions" checkbox should normally be checked, that means it will not redeploy the same changeset if it already exists in that environment. However if you've made a configuration change in UrbanCode and want to redeploy the same changeset, you can uncheck this box.
2. "Schedule Deployment" can be checked and then a calendar will appear if you want to schedule a deployment that way.

**\*NOTE\*** When scheduling a deployment, it will use the environment properties at the time the deployment is scheduled. If you need to update the environment properties for the deployment, you must cancel the scheduled deployment and schedule a new one.

3. "Description" helps if you need to differentiate one deploy from another, but is not a mandatory field.
6. You can chose to deploy a snapshot via a dropdown menu if one is already created, if not select "Choose Versions" at which point another pop up window will appear.

Component Versions

Select For All...

☐ Show only changed components

☐ Allow invalid versions

Component	Current Environment Inventory	Versions to Deploy
<input type="text"/>		
docupload	None	Add...
fidloanwiz	None	Add...
isawiz	None	Add...
nbformselsvc	None	Add...
nbissappsvc	1.0.0.280209	Add...
nbstrptsvc	1.0.0.280131	Add...
nbsubmissionsvc	None	Add...
posvisibilityrules	None	Add...
srcservice	None	Add...
stpfwiz	None	Add...

21 records

<<

<

1

>

>>

Rows

10

OK

- Notice the component names on the left side and you can select the version of that component on the right side, usually you will select "Add => Latest Available"
- Press "OK" and then "Submit" to execute the deploy.
- You can then navigate to the "Dashboard" tab to see the deployment, in that view, click "View" to see the steps execute.

## UrbanCode Deploy to Prod:

Preparing for production deployment

Create your snapshot in Urban Code

- Login to UrbanCode
- Navigate to the Application tab
- Navigate to your application
- Click on the Snapshots tab

## Application: NGIS

Created By	NICK, MICHAEL (nic8304)
Created On	7/28/2015, 4:01 PM

Environments

History

Configuration

Components

Blueprints

Snapshots

Processes

Calendar

Changes

5. If you are not using an existing snapshot, click on "Create Snapshot"
  1. Give the snapshot a name, and optionally a description, and click "Save"

Create Snapshot

Name \*

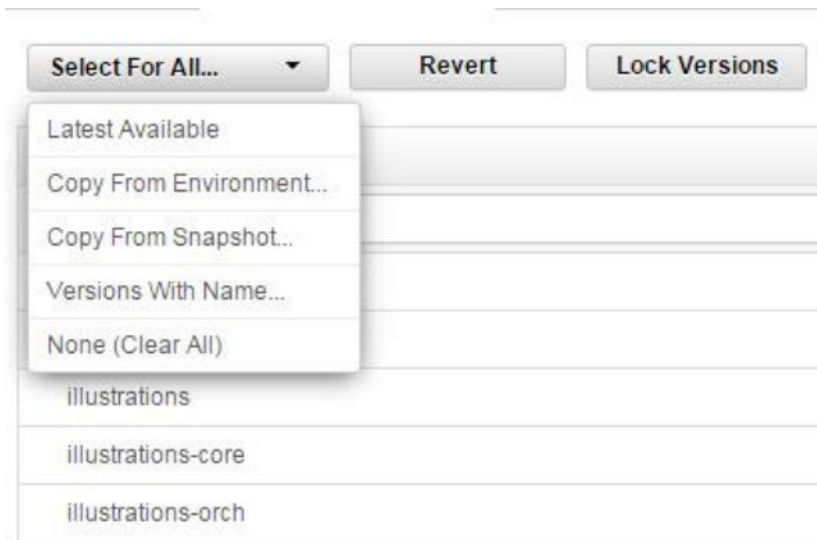
Charles\_Wiki

Description

Save

Cancel

2. Select the versions to include for each component being deployed by doing one of the following
  1. Under "Select For All..."
    - click on "Latest Available" to add the most recent version for each component
    - Click on "Copy From Environment..." to add the versions deployed to a specific environment
    - Click on "Copy From Snapshot..." to use the versions for components from another snapshot
    - Click on "Versions With Name..." to use versions with a specific name



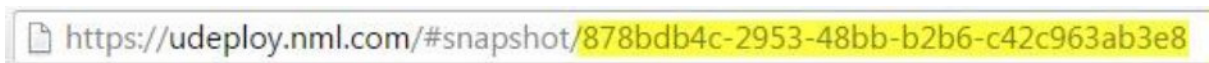
2. Add individual versions to each component by clicking on the "Add..." link

Component	Versions
<input type="text"/>	
calcengine	 <a href="#">Add...</a>
calcui	 <a href="#">Add...</a>
illustrations	 <a href="#">Add...</a>
illustrations-core	 <a href="#">Add...</a>
illustrations-orch	 <a href="#">Add...</a>

6. If you are using an existing snapshot, click on it

NOTE: Do not use the same snapshot across multiple change records. The HPSM integration cannot determine the approval status when the same ID is present on more than one record.

7. Locate the snapshot ID in the URL bar. This will be used later on the HPSM change record promotion tab.



Create your change record in HPSM

1. Login to HPSM
2. Go to the Change Queue

3. Open a new change (Planned / For the Record)
4. Fill in all details as normal
5. On the Promotion tab, add a Manual entry in the grid with the snapshot id captured earlier for the Promotion ID and the "WEBWASMQ CO" for the CO Group

Change Details	Testing	Promotion	Implementation	Related Records	Approvals	PCR	Attachments	History
Method								
Method	Promotion ID	LAN ID (lower case) or CO GROUP						
MANUAL	FCF29226-33A8-468A-8C78-830600AAC5E3	WEBWASMQ CO						

### Deploying into production

1. Login to [UrbanCode](#)
2. Navigate to the Application tab
3. Navigate to your application
4. Click on the play button for your production environment



5. Select the Deploy process
6. Select the snapshot you created / used in the prior steps
7. If this is a future deployment, select the Schedule Deployment checkbox and enter the date and time for the deployment to occur. You can either select from the popup menus, or manually enter them.
8. Click Submit



Run Process on Prod

Only Changed Versions ☒

Process \* 

Deploy

Select a snapshot, or choose versions for individual components.

Snapshot 

r1

Schedule Deployment? ☒

Date \* 

11/28/2015

Time \* 

2:00 PM

Make Recurring ☐

Description 

R1 deployment

Submit

Cancel

## UrbanCode Set Up AutoDeploy:

AutoDeploy is a fundamental aspect of Continuous Delivery. It is a long term goal to grow into this functionality. Your team will discover pros and cons to doing this and can adjust accordingly.

- There are settings in both TeamCity and UrbanCode to enable autodeploy, however it is a process performed by UrbanCode. The part in TeamCity is needed so it can be passed to UrbanCode via the REST API.

## TeamCity

Navigate to your component's Parameters in TeamCity. The only value you'll need to ever change is deploy.application. All other values are handled in the TeamCity Template.

Variable Name	Value	Notes
deploy.application	-DeployComponent or nothing	nothing is "off", - DeployComponent is "on". Nothing is

leaving the text box empty not 'nothing'.

deploy.application.environment Test

This should not be changed

deploy.application.environment %system.teamcity.projectName%

This should not be changed

deploy.application.process Deploy

Will always be 'Deploy'

## UrbanCode

1. Navigate to your component in UrbanCode. (Components - then find and click component name).
2. There is a "sub" menu, find Configuration and click it, note UrbanCode is confusing and there are 2 Configuration tabs, this one is inbetween Usage and Calendar.
3. In Basic Settings (default) section, scroll down and find Run Process after a Version is Created, and check it.
4. Two additional drop down menus should appear
5. For Application Process select SRC - Deploy (or your given option with - Deploy).
6. For Environment select SRC - Test (or your given option for Test).

The screenshot displays the 'Run Process after a Version is Created' configuration section in UrbanCode. The 'Run Process after a Version is Created' checkbox is checked and highlighted in yellow. Below it, the 'Application Process' dropdown menu is set to 'SRC - Deploy' and the 'Environment' dropdown menu is set to 'SRC - Test', both also highlighted in yellow. The 'Save' button is green and the 'Cancel' button is grey.

Import Versions Automatically ☐

Copy to Code Station ☒

Default Version Type \* Full

☒ Use the system's default version import agent/tag.  
☐ Import new component versions using a single agent.  
☐ Import new component versions using any agent with the specified tag.

Cleanup Configuration

Inherit Cleanup Settings ☒

Run Process after a Version is Created ☒

Application Process \* SRC - Deploy

Environment \* SRC - Test

Save Cancel

1.

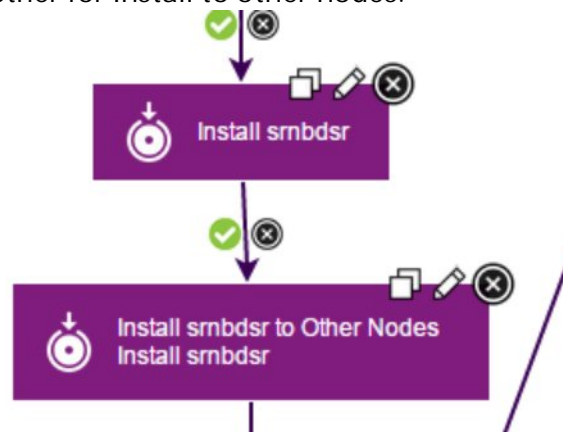
- Note Stage and Production options are available, although there are preventative measures in place that would fail the process if an attempt was made to select Production. Currently you could in theory select Stage if your team decided that made the most sense for your area.

## UrbanCode New .NET Component Deployment:

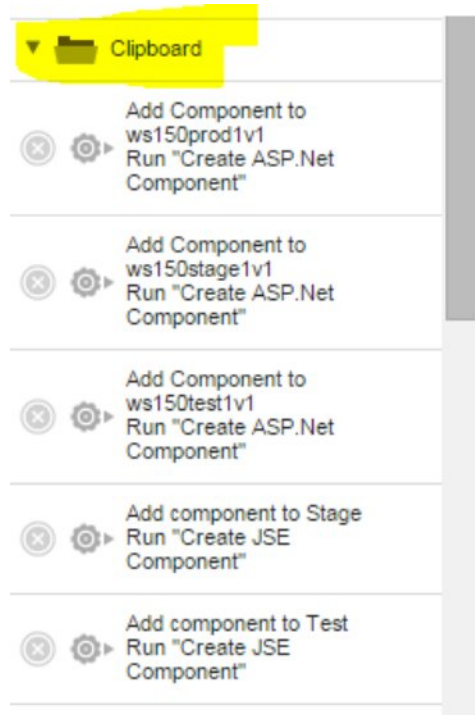
UrbanCode has a processes that are built into other processes. This is apparent when you add a new .NET component to your overall application area. If you overall application is 'SRC' and you have a brand new .NET application, you need to specifically add that application to your SRC Deployment. This uses SRC as an example but applies to any application area with a .NET component that is new or not already set up in their UrbanCode Deployment.

### Steps

1. Click Applications, and then your Application Area
2. On the 'Sub' menu, click Processes, the one that is in between Snapshots and Calendar
3. Click Deploy
4. Make sure you are in the "Design" View
  1. Note You may not have access to edit this process
5. You'll notice each .NET component for your application has two tiles in the Design, one for Install and another for Install to other nodes.



- 1.
6. You can copy an existing components two tiles, this is done by clicking the White Squares icon, to the left of the Pencil Icon on the tile's upper right-side corner.
7. Click the copy button on both the first install tile, and the second tile below it. You can pick any component that is already set up to copy, it does not matter.
8. After both are copied, find the Clipboard in the left hand tools expandable menu. The two you just copied will appear in the Clipboard after you expand it. Don't do anything with them yet.



- 1.
9. Look in the middle section again and delete the last arrow from the lowest tile connected to Finish by clicking the X near the middle of the arrow.
10. Drag Finish Tile a little lower to give yourself room for two more tiles
11. Drag the first Install step from the clipboard, it will immediately prompt you to edit it once you drag it on to the diagram.
12. Change the component from the drop down menu to your new component. Change the name as appropriate.

**Edit Properties** ✕

---

**Name \***

**Component \***

**Use Versions Without Status \***

**Component Process \***

**Limit to Tag**

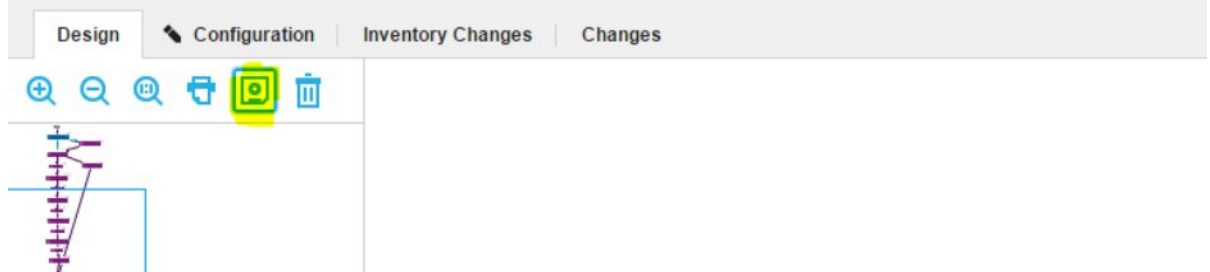
**Max # of concurrent jobs. \***

**Fail Fast** ☐

**Run on First Online Resource Only** ☐

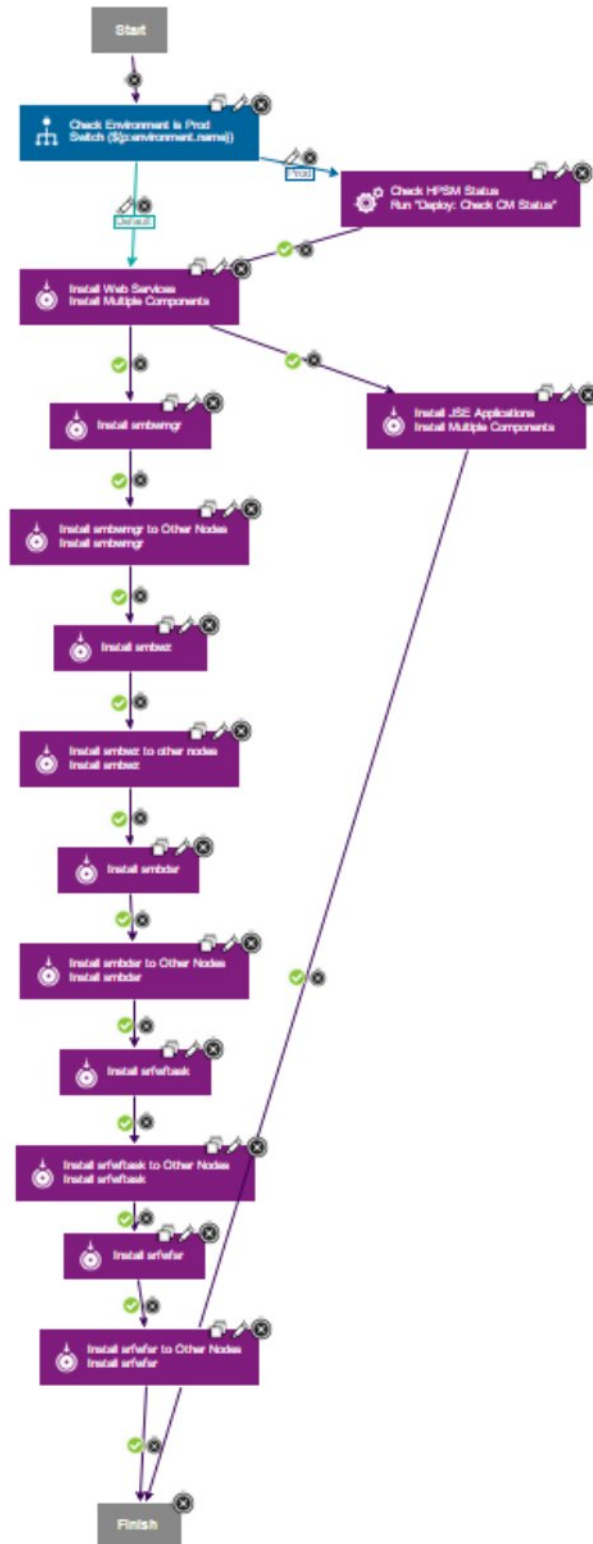
1.

13. Do the same thing for the other tile, ...Other Nodes
14. If you hover over the tiles, you'll notice a blue arrow, click that and drag down to the tile below.
15. Do that for the previous last tile to the new tiles, until your last tile is connected to Finish
16. Click Save which is a tricky icon in the top of the left hand menu.



1.

17. Full Process After September 21, 2015 Edit:



1.

## UrbanCode Environment Gates:

Environment Gates in UrbanCode prevent components from being deployed that do not meet a criteria set.

### How To

To see the environment gate, or to change them, you navigate to your Application's configuration:

1. Click Applications
2. Click your Application name (ie 'SRC')
3. Click the sub-menu Configurations
4. Click the left-side Environment Gates

#### Application: SRC

The screenshot shows the UrbanCode Configuration page for Application: SRC. The top section displays 'Created By: PIESENS, DAN (pie15)' and 'Created On: 12/10/2014, 3:40 PM'. Below this is a navigation bar with tabs: Environments, History, Configuration (selected), Components, Blueprints, Snapshots, Processes, Calendar, and Changes. The left sidebar shows a menu with 'Basic Settings', 'Application Properties', and 'Environment Gates' (selected). The main content area is titled 'Environment Gates' and contains a 'Save Conditions' button. Below the button, there are four sections: 'Test', 'Test02', 'Stage', and 'Stage02', each with an 'Add a new condition...' dropdown. Below these are two sections: 'Deployed to Stage' (with a red 'X' icon) and 'And...' dropdown, followed by '- or -', 'Prod' (with a blue 'X' icon), 'Deployed to Test' (with a red 'X' icon), 'And...' dropdown, followed by '- or -', and 'Add a new condition...' dropdown.

1.

- Your default should have only restrictions placed on Production that says, if a component has not been to Test or Stage, it cannot go to Production. Ideally we will change this to say Test and Stage.
- You can then go into a given component and navigate to its Versions and see that it has been tagged with the environment it has been in

Home > Components > nbissappsvc

### Component: nbissappsvc

**Created By** PIESENS, DAN (pie15)

**Created On** 5/11/2015, 12:06 PM

**Template** WebSphere Enterprise Application

**Used By** SRC

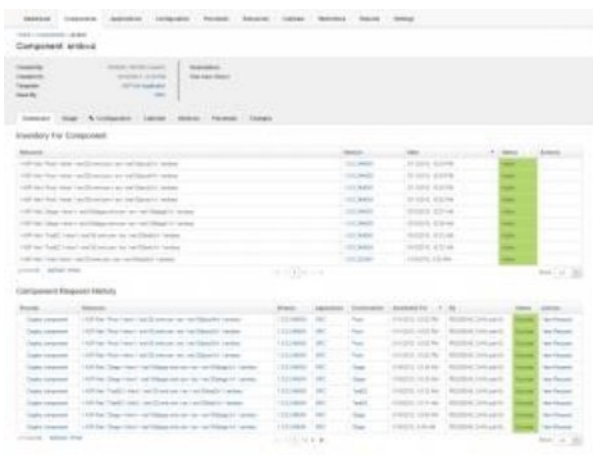
**Description:** New Issue Application Service

Dashboard Usage **Configuration** Calendar **Versions** Processes Changes

Version	Statuses	Type	Created By
1.0.0.299068	Deployed to Test	Full	SRC Build Agent (src-builder)
1.0.0.299053	Deployed to Stage	Full	SRC Build Agent (src-builder)
1.0.0.298989	Deployed to Test	Full	SRC Build Agent (src-builder)
1.0.0.298786		Full	SRC Build Agent (src-builder)
1.0.0.298771		Full	SRC Build Agent (src-builder)
1.0.0.298764		Full	SRC Build Agent (src-builder)

## UrbanCode .NET Quick Start Guide:

This is a quick start guide for IBM UrbanCode which is a continuous deployment tool.



## UrbanCode Component View

### High Level Overview

1. Run Add Application process (Not needed if your top level application exists, ex. if SRC, this is not needed. If you think you need to do this, email Continuous Delivery PDL)
2. Run Create {ASP.Net, JEE} Component for each node(server) in each environment
3. Add additional Environment Property Definitions
4. Configure Environment Property values for each environment available to that component



## .NET Quick Start Guide

1. Click 'Processes' tab
2. Run Create ASP.NET Component notice 'Run' to right side in Actions Column
3. Fill in the Following Parameters, examples from SRC have been provided  
Note: This will be run for each server (per environment) this component is deployed to
  1. Component -- srnbwz (the name of the app for deployment purposes, for example, this is used to dynamically give your component its name on the environment server, if going to Test02, the deployment would auto name this example srnbwz02)
  2. Ping Package -- ASPPing-Intra
  3. Server -- ws150test1v1 (one server per environment per run)
  4. App Pool -- src
  5. Site -- ws150test.nml.com
  6. Farm -- intra1
  7. Environment (one environment per run) -- Test
  8. Agent -- bana01
  9. Application -- SRC
  10. Team Name -- SRC (just a tag within urbancode)
  11. Resource -- click 'Set', open first carat for Agents and click radio button for Generic Agents
4. Click Submit
5. Process will begin to run, there are 7 main steps that will need to run, notice that the name of the step indicates whether or not a 'Success' is needed. For example, if It Checks If a Resource Exists, and that resource exists, then it will Not Start 'Create Resource.'
  1. If there are any errors, the step that failed will have a log that can be viewed. UrbanCode sometimes has issues in the browser, so if you do not see the log after clicking it, it is usually best to clear your cache. It is a flaw in UrbanCode. After each environment has been added for this component, begin next steps.
6. Add Environment Property Definitions to Component
  1. Navigate to 'Configuration' on top menu
  2. Click on Component name in expandable menu on left hand side
  3. Click 'Add Property'
  4. To Supply value per each environment, expand carat next to component name and click into the environment listed below.
  5. Find property name and enter value

## UrbanCode JEE Quick Start Guide:

### High Level Overview

1. List of Apps in Each Environment , helps for validating naming of components: WAS
2. Run Add Application process (If Application does not already exist)

3. Run Create JEE Component for each node(server/agent) in each environment
4. Add additional Environment Property Definitions
5. Configure Environment Property values for each environment available to that component

## JEE Quick Start Guide

1. Click upper level Processes tab
2. Find Create WAS Component Process and click
3. Fill in the following fields: (SRC Examples) Note: This will be run for each agent (per environment) this component is deployed to
  1. Component - nbsubmissionsvc
  2. Application - SRC
  3. Environment one environment per run - Test
  4. Agent - bana01
  5. WAS Cell - DMIntra4
  6. Cluster - stpmenu
  7. Team Name - SRC
  8. Component Description - Deploy component for the New Issue Submission Service
  9. Resource -- click 'Set', open first carat for Agents and click radio button for Generic Agents
4. Click Submit - This will be done the number of environments you have multiplied by the number of agents
5. Process will Run - 14 Steps, however note that yellow page steps are just comments on what is happening.
  1. If any errors, find logs:
    1. Each step that has a log, that line item will have a blue terminal icon appear when you hover over it, click on that icon and the log appears. There are sometimes browser issues with this in UrbanCode, so the log might appear blank, if that is the case, try another browser or clear your cache. This is a known UrbanCode issue.
    2. The global logs can be found, if you have permissions, via Settings, under System, the first option is "Logging."
6. After success, navigate to 'Components' tab and find newly created component
7. Hover over line, notice black tag icon, and click on it, assign newly created component application tag

## SRC Version

There is a process Add WAS Component to SRC which is the above process, but has an additional step of running a Test02 Environment set up, with many values hard coded.

These are the only needed components to add a new SRC component:

1. Component Name
2. Component Description
3. Cell Name
4. Cluster (defaulted to stpmenu)
5. Resource
6. Hard Coded Values for Test02 process
  1. Environment - Test02
  2. Cell - DMIntra30
  3. Cluster - stpmenuext

## Manual Finishing Steps

### Configuration Final Steps

1. Click Components top level tab
2. Click on given component name in list
3. Click on 'Configuration' sub-level tab
4. Notice These values and ensure they are filled:
  1. Application Type: Application
  2. WebSphere Application Name: \${p:appName}-\${component.name}\${p:environment/environmentSuffix}-\${p:environment/environmentBase}
  3. Application Name: stpmenu (SRC example)

### Agent Mapping Verification

1. Click on Resources
2. Enter component name in the search text box near top of screen, should have shadow text reading "Resource Name."
3. Verify that the component appears in expected areas, Test, Stage, Prod

## UrbanCode JSE Quick Start Guide:

### High Level Overview

This deployment for a batch application is essentially only doing a copy of the binary, and other aspects of the application would already be set up if you are running this step, so there is only one process to run.

1. Run Create JSE Component for each node(server/agent) in each environment

## JSE Quick Start Guide

1. Click upper level Processes tab
2. Find Add WAS Component to Application Process and click
3. Fill in the following fields: (SRC Examples)
  1. Component - newissue-listener-failure
  2. Application - SRC
  3. WAS Cell- DMIntra4
  4. Cluster - stpmenu
  5. Team Name - SRC
  6. Component Description - New Issue Application Service Failure Listener
  7. Test Agent - bana01 (defaulted)
  8. Stage Agent - bana02 (defaulted)
  9. Prod Agent - bana03 (defaulted)
  10. Resource -- click 'Set', open first carat for Agents and click radio button for Generic Agents
4. Click Submit
5. Process will Run its steps, note that yellow page steps are just comments on what is happening.
  1. If any errors, find logs:
    1. Each step that has a log, that line item will have a blue terminal icon appear when you hover over it, click on that icon and the log appears. There are sometimes browser issues with this in UrbanCode, so the log might appear blank, if that is the case, try another browser or clear your cache. This is a known UrbanCode issue.
    2. The global logs can be found, if you have permissions, via Settings, under System, the first option is "Logging."
6. After success, navigate to 'Components' tab and find newly created component
7. Hover over line, notice black tag icon, and click on it, assign newly created component application tag

## Component Properties

1. Navigate to 'Components' and then click on your JSE component name.
2. In the sub-menu click Configuration
3. Look over the values to make sure they make sense, many of them are based on a parameterized naming pattern that worked for SRC RI/DSR.
4. JSE Applications typically have associated 'Script' files that run as autosys jobs. Script Name is a property under this Configuration tab. You can add one or many script file names. Use a single space as a delimiter.
  1. Example: Script1 Script2

## SRC Version

There is a process Add JSE Component to SRC which is the above process, but has default values in place. This has Application and Team hardcoded to "SRC" and the Test01/02 Agent hard coded to bana01, Stage01/02 Agent hard coded to bana02, and Prod hard coded to bana03. Only two fields required to fill, with the description being driven off of the Component name.

- Component
- Resource (Set, Agents, Radio Button for Generic Agents)

## Manual Finishing Steps

### Agent Mapping Verification

1. Click on Resources
2. Enter component name in the search text box near top of screen, should have shadow text reading "Resource Name."
3. Verify that the component appears in expected areas, Test, Stage, Prod.

## Deployment Flows

- An important note in UrbanCode is that workflows or processes can be created for more than deployments, and can be created for sub-steps within another process too.

## Deploy EAR With WASPING:

### Intro

For any WAS Application, navigate to the 'Components' tab, click on the WAS Application, click the sub-tab "Processes" (not at top of screen), and then click "Deploy EAR with WASPING."

### Steps

1. Start
2. Parallel Tracks
  1. Track 1
    1. Download Artifacts
  2. Track 2
    1. Delete Files and Directories
    2. Create File
    3. Create Directories
3. Unzip
  1. Track 1

1. Get WAR Name
2. Track 2
  1. UnZip WAR
4. Replace Tokens
5. Update WSRR Config
6. Copy WAS Ping
7. RePackage WARFile
8. Delete unzipped WAR Dir
9. Update XML File with XPath
10. RePackage EAR File
11. Check Application is Installed
12. Is App Installed?
  1. True
    1. Check Application is not running
    2. Stop Application
13. Install or Update Application (Also False Path)
14. Map Resource References to EJB for Application
15. Map Users and Groups to Roles for Application (not working)
16. Wait for Application
17. Start Application
18. Finish

## Details

### 2.1.1 Download Artifacts

Include:	**/*
----------	------

### 2.2.1 Delete Files And Directories

Include:	unZipped/**/*
----------	---------------

### 2.2.2 Create File

UrbanCode WAS Ping Page Code

### 2.2.3 Create Directories

Directories:	unZipped
--------------	----------

### 3 Unzip

.zip files:	\${p:earFile}
Include Files:	**/*

#### 3.1.1 Get WAR Name

Groovy Home	\${GROOVY_HOME}
Script:	<pre>def fileFinder = new FileNameFinder()  def warFile = fileFinder.getFileNames('unZipped', '*.war').first() def javaFile = new File(warFile)  outProps['warFile'] = javaFile.name outProps['warFileBase'] = javaFile.name.split("\\.", 2)[0]</pre>

#### 3.2.1 Unzip WAR

.zip files:	unZipped/*.war
Include Files:	**/*

### 4 Replace Tokens

Include Files:	**/*
Start Token Deliminator	@
End Token Deliminator	@
Property File Name	replace_tokens.properties

Property List	<code>\${p:environment/allProperties}</code>
---------------	--

## 5 Update WSRR Config

Directory Offset:	<code>unZipped/unZippedWAR/WEB-INF/classes</code>
Include Files:	<code>ws-ser-config.properties</code>
Add/Update Properties :	<code>com.nm.aa.sr.wsrr.endpointproxy.address=\${p:environment/wsrr.endpoint.address}</code> <code>com.nm.aa.sr.wsrr.consumer.usedynamicendpoint=\${p:environment/wsrr.usedynamicendpoint}</code> <code>com.nm.aa.sr.wsrr.includewsheader=\${p:environment/wsrr.includewsheader}</code>

## 6 Copy WASPING

Source Directory:	<code>.</code>
Destination Directories	<code>unZipped/unZippedWAR</code>
Include Files	<code>wasping.jsp</code>

## 7 Repackage WARFile

.zip File Name	<code>unZipped/\${p:Get WAR Name/warFile}</code>
Base Directory	<code>unZipped/unZippedWAR</code>
Include	<code>**/*</code>

## 8 Delete unZippedWAR Dir



Base Directory	unzipped
Include	unZippedWAR/**/* unZippedWAR

## 9 Update XML File with XPath

Source Directory Offset	unZipped/META-INF
Destination Directory Offset	unZipped/META-INF
File Includes	application.xml
Replace with text	/:application/:module/:web/:context-root/text()->\${p:Get WAR Name/warFileBase}\${p:environment/environmentSuffix}

## 10 Repackage EAR File

.zip file name	\${p:earFile}
Base Directory	unZipped
Include	**/*

## 11 Check Application is Installed

Application Name	\${p:websphereAppName}
Application Edition	\${p?:appedition}

### 12.1.1 Check Application is Not Running

Application Name	<code>\${p:websphereAppName}</code>
Application Edition	<code>\${p?:appedition}</code>
Application Type	<code>\${p:appType}</code>

### 12.1.2 Stop Application

Application Name	<code>\${p:websphereAppName}</code>
Application Edition	<code>\${p?:appedition}</code>
Script File	<code>\${p:request.id}.py</code>

### 13 Install or Update Application

Application Source	<code>\${p:earFile}</code>
Application Name	<code>\${p:websphereAppName}</code>
Application Edition	<code>\${p?:appedition}</code>
Editor Description	<code>\${p?:appEditDesc}</code>
Script File	<code>\${p:request.id}.py</code>

### 14 Map Resource References to EJB for Application

Application Name	<code>\${p:websphereAppName}</code>
Application Edition	<code>\${p?:appedition}</code>

Resource References	<code>\${p:resourceMappings}</code>
Script File	<code>\${p:request.id}.py</code>
Precondition	<code>(properties.get("resourceMappings") != "")</code>

#### 15 Map Users and Groups to Roles for Application

Application Name	<code>\${p:websphereAppName}</code>
Application Edition	<code>\${p?:appedition}</code>
Role Mappings	<code>\${p:roleMappings}</code>
Script File	<code>\${p:request.id}.py</code>
Precondition	<code>(properties.get("resourceMappings") != "")</code>

#### 16 Wait for Application

Application Name	<code>\${p:websphereAppName}</code>
Application Edition	<code>\${p?:appedition}</code>
Timeout(seconds)	600
Interval(seconds) - misspelled in UrbanCode	10

#### 17 Start Application

Application Name	<code>\${p:websphereAppName}</code>
------------------	-------------------------------------

Application Edition	<code>\${p?:appedition}</code>
Script File	<code>\${p:request.id}.py</code>

## Deploy JSE Application:

### Intro

For any JSE Application, navigate to the 'Components' tab, click on the JSE Application, click the sub-tab "Processes" (not at top of screen), and then click "Deploy JSE Application."

### Steps

1. Start
2. Download Artifacts
3. Replace Tokens
4. Copy contents to directory share
5. Create Root Directories
6. Update Directory Permissions
7. Update Custom Permissions
8. Finish

### Details

#### 2 Download Artifacts

Includes:	<code>**/*</code>
-----------	-------------------

#### 3 Move Replace Tokens

Include Files:	<code>**/*</code>
Start Token Deliminitor	<code>@</code>
End Token Deliminitor	<code>@</code>

Property File Name	replace_tokens.properties
Property List	\${p:environment/allProperties}
Explicit Tokens	environment.name->\${p:environment.name}

#### 4 Copy contents to directory share

Destination Directories	\${p:component/deployRoot}
Include files:	**/*
Working Directory	\${p:component/deployRoot}

#### 5 Create Root Directories

Directories:	backup bin config data docroot include lib logs rpts scripts src sql sysln
Working Directory	\${p:component/deployRoot}

#### 6 Update Directory Permissions

Directory Offset:	.
Shell Script	#chown -R \${p:component/applicationName} ./**/*.*
Working Directory	\${p:component/deployRoot}

#### 7 Update Custom Permissions

Groovy Home:	\${GROOVY_HOME}
--------------	-----------------

Script	<pre>println "Updating Custom Permissions"  String fileExceptions = "\${p:component/fileExceptions}"</pre>
Working Directory	<code>\${p:component/deployRoot}</code>

## UrbanCode Rename a Component:

Rename a Component is not as straightforward as it might appear in UrbanCode. The case for this is if you made a simple mistake typing or have a need to rename a component. There are 3 spots you will have to check to rename the component - as a way of ensuring that the old name does not persist.

Click on Components Tab

1. Find the name of the component, click it
2. Click on the 'Configuration' sub-menu
3. Type in the new name, and click save.

Click on Resources Tab

1. Click on the Type of App - .NET, JEE, JSE, etc
2. Click on the Environment it is in (might be all environments so this will need to be repeated)
3. Then find the Cell or Cells it is hooked up to, and ensure the name is the same as above change.
4. If not, find the drop down arrow in the row and edit.

Click on Applications Tab

1. Click on App (SRC)
2. Click on Configuration sub-menu
3. Ensure component has new name, if not click on and edit similar to above 'Components' tab step.

## UrbanCode Setup & Administration:

- UrbanCode topology diagram

- How to get UrbanCode license information
  - Log into the Rational License Server (RDP into the Windows server). To confirm what license server UCD is using, log into the UCD UI, navigate Settings tab -> System Settings link -> Licensing section. The RCL server path textbox contains the license server URL.
  - Follow the steps in IBM's Technote on how to use LMTools

## Future and Current Continuous Delivery Tooling

This page focuses on selection of tools for a Continuous Delivery Pipeline. The primary goals are to clarify the current state, identify key decision points, and establish a road map for addressing critical questions.

A PowerPoint version of this information is available.

### Definitions

A continuous delivery pipeline will provide a set of capabilities. The following is a high-level listing, but sufficient as a basis for determining decision points that must be addressed.



#### Plan

Planning work items, managing a backlog of features, etc.

#### Code

Producing source code and configuration.

#### Build

Building application artifacts.

#### Test

Verifying functional and quality requirements are met.

#### Release

Releasing application artifacts ready for deployment.

#### Deploy

Deploying applications to environments where they can be run.

#### Operate

Maintaining operational applications.



# Continuous Integration



CI Pipeline

"**Continuous Integration (CI)** systems provide automation of the software build and validation process driven in a continuous way by running a configured sequence of operations every time a software change is checked into the source code management repository." (*Gartner*)

This diagram illustrates the fact that the scope of **Continuous Integration** is limited to the building and testing of delivered code.

# Continuous Delivery

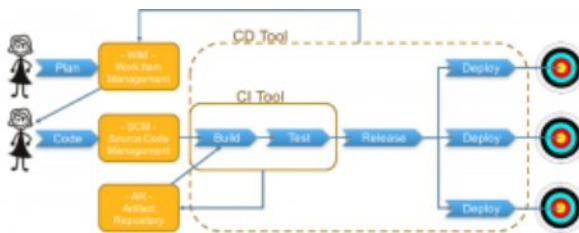


CD Pipeline

"**Continuous Delivery (CD)** is a set of principles and practices to reduce the cost, time, and risk of delivering incremental changes to users. (*Jez Humble and David Farley in Continuous Delivery*)

This diagram illustrates that **Continuous Delivery** broadens the scope of CI to include up-front planning as well as the release and deployment (delivery) of functional applications.

# Continuous Delivery Pipeline



Continuous Delivery Pipeline

For the purposes of this discussion, a **Continuous Delivery Pipeline** is a process by which an application is delivered by means of a coordinated set of tools providing Continuous Delivery capabilities. The application delivered by a pipeline may consist of coordinated

delivery of parts of the application to multiple targeted platforms, but the deployment should be considered as a single process.

Numerous "pipelines" may be operational throughout the organization, one per deliverable application.

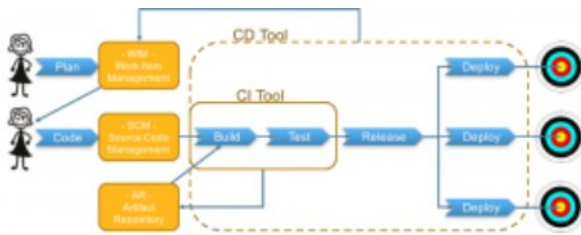
This diagram gives a high level view of the constituent parts of a Continuous Delivery Pipeline. The intent is to show the coordination between the constituent capabilities of Continuous Delivery and the points where tools are involved in this process.

## Current State

The following image identifies tools currently in use at Northwestern Mutual for each of the pipeline capabilities identified above. Click on individual tools for further detail.



## Pipeline High-Level Components

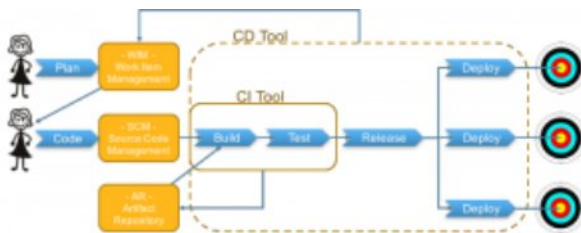


Pipeline High-Level Components

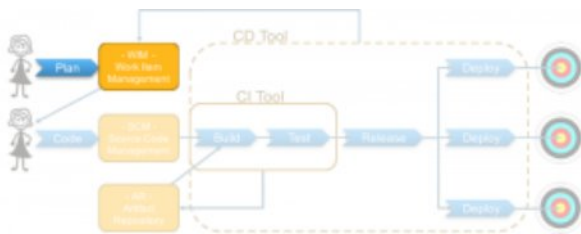
[\[Expand\]](#)

Decisions

## Single or Multiple Pipelines



## Work Item Management



Work Item Management (WIM)

## Primary Candidates

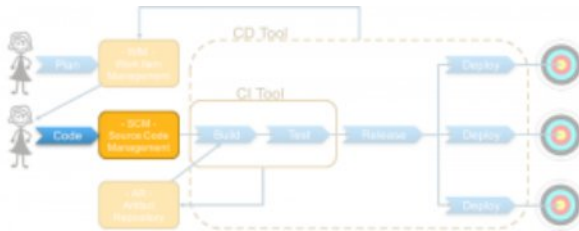
**[Criteria**

- Ability to track work items through build, test, release and deployment
- Ability to manage a backlog of work items
- Support

**Interesting**

- TFS provides both WIM and SCM functionality. Does it make sense to use TFS for one but NOT the other?

## Source Code Management



Source Code Management (SCM)

**Source Code Management (SCM)** is the management of the digital materials produced by developers and used for building applications. This includes both text (computer programs and configuration), and binary (visual and audio) files. This typically does NOT include materials that are obtained from other repositories (internal and external) and not produced as part of development of the current application.

There are two primary types of SCM:

### Centralized

All revision control functions take place on a shared server. Simultaneous changes are controlled by file locking and version merging.

This approach is often adopted because it is perceived as more easily comprehended. However, there are significant problems that arise in use, particularly in agile projects.

- It is not possible for more than one person to work on a particular artifact at the same time.
- It is necessary to "check out" (and consequently lock) a collection of related artifacts if work is done that could potentially impact all of them. This exacerbates the prior issue, since it is now not possible for more than one person to work in a particular are of the application at the same time.
- It is necessary to do this check out ahead of time, when one can only guess at the range of artifacts that must be included.
- Since it is necessary to keep this collection of artifacts checked out during the entire time work is done, they tend to remain checked out for extended periods.
- It is common to forget about checkouts since a period of time lapses and other issues arise in the mean time.

### Distributed

A peer-to-peer approach is taken with each peer having a bona-fide copy of the repository. Revision control is accomplished by exchanging change-sets from peer to peer.

This approach is more suited to modern agile development practices for several reasons:

- There is less need to worry about dependent sections. Developers make necessary changes on artifacts, and only modifications are committed.
- Independent modifications are merged back into the repository. Any issues are identified and addressed when changes are committed and "pushed" to the shared repository. This is more realistic than guessing what needs to be locked at checkout.
- Nothing is locked, so there is never a pause while waiting for an artifact to be available.
- Commits are made and merged locally, so operations are quick and can be done offline.

## **Primary Candidates**

### **Team Foundation Server (TFS)**

TFS provides both Work Item Management and Source Code Management. It is a Centralized SCM.

- As a Centralized SCM, TFS requires pre-update locking. See the discussion of Centralized SCM and Distributed SCM above for a discussion of why this complicates agile development.
- Though TFS provides both Work Item Management and Source Code Management, there is no technical reason that both need to be used together. Just because TFS is used for WIM does NOT mean that it should also be used for SCM (and vice versa). Both of these capabilities may be considered independently.
- TFS has a Command Line Interface (CLI) that can be accessed via Java and Node. This provides a means of using TFS on platforms (Linux, OSX, etc.) that do not support TFS directly. However, these have been difficult to work with in practice. The net of this is that TFS is NOT a viable option for Unix/Linux based source code management at Northwestern Mutual.

### **GitLab**

GitLab is a vendor that provides Git services. Git is a Distributed SCM

- As a Distributed SCM, GitLab requires post-commit merging. See the discussion of Centralized SCM and Distributed SCM above for a discussion of why this facilitates agile development.
- A feature comparison with GitHub is available on the GitLab web site.
- Git client support is available in all IDE's and platforms currently in use at Northwestern Mutual. Both command line CLI and graphical (GUI) interfaces are available.

## GitHub

GitHub is a vendor that provides Git services. Git is a Distributed SCM

- As a Distributed SCM, GitHub requires post-commit merging. See the discussion of Centralized SCM and Distributed SCM above for a discussion of why this facilitates agile development.
- A feature comparison with GitLab is available on the GitHub web site.
- Git client support is available in all IDE's and platforms currently in use at Northwestern Mutual. Both command line CLI and graphical (GUI) interfaces are available.

## Others

- Git is free Open Source software which could be installed at Northwestern Mutual. This would provide all of the basic benefits of Git but without the additional capabilities and interface provided by GitLab and GitHub. However, since Git clients are available for VisualStudio and Eclipse-based IDE's, as well as many good independent clients and tools, the need for these additional services may not be sufficient to justify licensing costs. Also, Git is extremely stable and has VERY broad community support, so vendor support provide by GitLab or GitHub may not be a sufficient benefit.
- TFS provides Git capability. Work would be required to verify whether this is a full implementation of Git, and whether it is distributed as opposed to centralized (see above). It will also need to be determined whether this affects licensing costs, and whether there is any advantage to this approach if TFS is used for WIM.

## Criteria

- Branch/Merge capabilities
- Platform support (Windows / Linux / OSX)
- Vendor support
- Licensing costs
- Interaction with artifact repositories (for security scanning)