Table of Contents

[1 Setting up Continuous Integration and Deployment 3](#_Toc42267304)

[1.1 Continuous Integration and Deployment processes overview 3](#_Toc42267305)

[1.1.1 Continuous integration (CI) 3](#_Toc42267306)

[1.1.2 Prerequisites for CI 3](#_Toc42267307)

[1.1.3 Generic CI algorithm 5](#_Toc42267308)

[1.1.4 Generic Deployment algorithm 6](#_Toc42267309)

[1.1.5 Baseline - Production backup 6](#_Toc42267310)

[1.2 Environment overview 9](#_Toc42267311)

[1.2.1 CI Environment overview 9](#_Toc42267313)

[1.2.2 Hardware requirements 11](#_Toc42267314)

[1.2.3 Software Requirements 12](#_Toc42267315)

[1.3 Manual Deploy 12](#_Toc42267316)

[1.3.1 Manual deployment from Git repository (Deploy.bat) 12](#_Toc42267318)

[1.3.2 Deploy from deployment package 13](#_Toc42267319)

[1.3.3 Setup baseline Innovator instance 13](#_Toc42267320)

[1.3.4 Restore installed instance of Innovator to a baseline state 14](#_Toc42267321)

[1.4 Setup Continuous Integration for Jenkins 15](#_Toc42267322)

[1.4.1 Setup Jenkins on Master node. 15](#_Toc42267324)

[1.4.1 Build node 18](#_Toc42267325)

[1.4.2 Setup SQL Server. 18](#_Toc42267326)

[1.4.3 Deployment node: configure Jenkins Master node, required software, attach to the Master node 22](#_Toc42267327)

[1.4.4 Setup CI job 32](#_Toc42267328)

[1.4.5 Setup deployment job 33](#_Toc42267329)

[1.4.6 Troubleshooting common errors 34](#_Toc42267330)

[1.5 Advanced Continuous Integration settings 36](#_Toc42267331)

[1.5.1 Setup proxy 36](#_Toc42267333)

[1.5.2 Setup feature licenses 38](#_Toc42267334)

[1.5.3 Setup build procedure to work at offline environment 40](#_Toc42267335)

[1.6 Setup Jenkins deploy for UAT and Production 43](#_Toc42267336)

[1.6.1 UAT/Prod: Innovator Components Configuration 43](#_Toc42267338)

[1.6.2 UAT/Prod: Setup Innovator 43](#_Toc42267339)

[1.6.3 UAT/Prod: Configure Innovator: Vault Server 46](#_Toc42267340)

[1.6.4 UAT/Prod: Configure Innovator: Conversion Server 47](#_Toc42267341)

[1.6.5 UAT/Prod: Configure Innovator: Agent Service 48](#_Toc42267342)

[1.6.6 UAT/Prod: Prepare Jenkins deployment job 49](#_Toc42267343)

[1.6.7 UAT: Restore Innovator instance to a baseline state 55](#_Toc42267344)

[1.7 Deploy to production 59](#_Toc42267345)

[1.7.1 UAT/Prod: Run deployment jobs 59](#_Toc42267347)

[1.7.2 Creating baseline after deploy to production 59](#_Toc42267348)

[2 Development with CRT 62](#_Toc42267349)

[2.1 Development processes and standards 62](#_Toc42267351)

[2.1.1 Development process overview 62](#_Toc42267354)

[2.1.2 Development standards 65](#_Toc42267355)

[2.1.3 Tests overview 67](#_Toc42267356)

[2.2 Developer guides 69](#_Toc42267357)

[2.2.1 Setup machine and branch specific settings 69](#_Toc42267359)

[2.2.2 Setup local Innovator instance from the repository 69](#_Toc42267360)

[2.2.3 Run Continuous Integration 70](#_Toc42267361)

[2.2.4 Converting Innovator server methods 70](#_Toc42267362)

[2.2.5 Run Tests 71](#_Toc42267363)

[2.2.6 Add or edit an item/method 71](#_Toc42267364)

[2.2.7 Remove an item/method that exists in a baseline 72](#_Toc42267365)

[2.2.8 Update imports.mf 72](#_Toc42267366)

[2.2.9 Setup solution: generic steps 74](#_Toc42267367)

[2.2.10 Setup solution: Product Engineering 74](#_Toc42267368)

[2.2.11 Setup solution: Program Management 76](#_Toc42267369)

[2.2.12 Setup solution: Component Engineering 77](#_Toc42267370)

[2.2.13 Setup solution: Technical Documentation 79](#_Toc42267371)

[2.2.14 Synchronize source control with production instance 81](#_Toc42267372)

[2.2.15 Fix CI and deployment errors 84](#_Toc42267373)

[3 Reference 85](#_Toc42267374)

[3.1 Tools structure 85](#_Toc42267376)

# Setting up Continuous Integration and Deployment

## Continuous Integration and Deployment processes overview

### Continuous integration (CI)

Continuous integration is a practice, not a tool.

Continuous integration requires that every time somebody commits any change, the entire application is built and a comprehensive set of automated tests is run against it. If the build or test process fails, the development team stops whatever they are doing and fixes the problem immediately. The goal of continuous integration is that the software is in a working state all the time.

### Prerequisites for CI

1. Version control.

Everything in your project must be checked in to a single version control repository: code, tests, database scripts, build and deployment scripts, and anything else needed to create, install, run, and test your application.

1. An automated build.

* You must be able to start your build from the command line.
* You need to be able to run your build process in an automated way from your continuous integration environment so that it can be audited when things go wrong.
* Your build scripts should be treated like your codebase. They should be tested and constantly refactored so that they are tidy and easy to understand.

1. Agreement of the team.

You need everyone to check in small incremental changes frequently to mainline and agree that the highest priority task on the project is to fix any change that breaks the application.

1. You need free machine.

It’s fine to start from unused development machine.

1. CI tool. There several open source options, such as:

* Jenkins from CloudBees.
* CruiseControl.NET from ThoughtWorks.
* GO from ThoughtWorks.
* TeamCity from JetBrains.
* Bamboo from Atlassian.

1. Check in regularly.

The most important practice for continuous integration to work properly is frequent check-ins to trunk or mainline. You should be checking in your code at least a couple of times a day.

1. Create a Comprehensive Automated Test Suite.

If you don’t have a comprehensive suite of automated tests, a passing build only means that the application could be compiled and assembled.

1. Keep the Build and Test Process Short

If it takes too long to build the code and run the unit tests, you will run into the following problems:

* People will stop doing a full build and running the tests before they check in. You will start to get more failing builds.
* The continuous integration process will take so long that multiple commits will have taken place by the time you can run the build again, so you won’t know which check-in broke the build.
* People will check in less often because they have to sit around for ages waiting for the software to build and the tests to run.

1. Managing Your Development Workspace.

Developers should always work from a known good starting point when they begin a fresh piece of work. They should be able to run the build, execute the automated tests, and deploy the application in an environment under their control.

1. Don’t Check In on a Broken Build

If the build breaks, the developers responsible are waiting to fix it. They identify the cause of the breakage as soon as possible and fix it. If one of our colleagues has made a check-in and broken the build as a result, then to have the best chance of fixing it, they will need a clear run at the problem. They don’t want us checking in further changes, triggering new builds, and compounding the failure with more problems.

When this rule is broken, it inevitably takes much longer for the build to be fixed. People get used to seeing the build broken, and very quickly you get into a situation where the build stays broken all of the time. This continues until somebody on the team decides that enough is enough, a Herculean effort ensues to get the build green, and the process starts all over again.

1. Always Run All Commit Tests Locally before committing.

Every commit triggers the creation of a release candidate.

1. Never Go Home on a Broken Build

It is 5:30 P.M. on Friday, all your colleagues are walking out of the door, and you have just committed your changes. The build has broken. You have three options. You can resign yourself to the fact that you will be leaving late and try to fix it. You can revert your changes and return to your check-in attempt next week. Or you can leave now and leave the build broken. If you leave the build broken, when you return on Monday your memory of the changes you made will no longer be fresh, and it will take you significantly longer to understand the problem and fix it.

1. 10 minutes rule

Establish a team rule: When the build breaks on check-in, try to fix it for ten minutes. If, after ten minutes, you aren’t finished with the solution, revert to the previous version from your version control system.

1. Don’t Comment Out Failing Tests

Commenting out tests that fail should always be a last resort, very rarely and reluctantly used, unless you are disciplined enough to fix it right away. It is OK to very occasionally comment out a test pending either some serious development work that needs to be scheduled or some extended discussions with the customer. However, this can push you down a slippery slope.

1. The next step is for everybody to start using the CI server. Here is a simple process to follow.

Once you’re ready to check in your latest change:

1. Check to see if the build is already running. If so, wait for it to finish. If it fails, you’ll need to work with the rest of the team to make it green before you check in.
2. Once it has finished and the tests have passed, update the code in your development environment from this version in the version control repository to get any updates.
3. Run the build script and tests on your development machine to make sure that everything still works correctly on your computer, or alternatively use your CI tool’s personal build feature.
4. If your local build passes, check your code into version control.
5. Wait for your CI tool to run the build with your changes.
6. If it fails, stop what you’re doing and fix the problem immediately on your development machine—go to step 3.
7. If the build passes, rejoice and move on to your next task.

If everybody on the team follows these simple steps every time they commit any change, you will know that your software works on any box with the same configuration as the CI box at all times.

### Generic CI algorithm

CI contains the following steps:

1. Checkout the latest modifications from Git.
2. Run validations against source files.
3. Build and run unit tests.
4. Setup temporary instance of Innovator on Jenkins Agent from production backups.
5. Deploy delta between last release (deploy) and head commits to the instance.
6. Build and run integration tests.
7. Run cleanup.

### Generic Deployment algorithm

Deploy procedure contains the following steps:

1. Evaluate delta between last release (deploy) and head commits to prepare deployment package (code tree and database upgrades).
2. Prepare deployment package based on the delta.
3. Restore Innovator instance to the baseline commit (use RestoreInnovatorFromBackups.bat or Jenkins deployment for SIT and UAT. For development environment use SetupInnovatorHere.bat).
4. Check if deployment parameters are correct (pre-deploy validation step).
5. Upgrade Code Tree of the target Innovator instance.
6. Upgrade Database of the target Innovator instance.
7. (For Production instance only) Prepare production backups

### Baseline - Production backup

Baseline is a combination of backups of production Code Tree and Database, that should be created and scrambled manually. Right after the production instance has been modified (due to either manual changes or automated deployment) the production backups should be updated. Keep in mind that there are several environments dependent from the baseline: development, SIT, UAT.

Next comes explanation of ideal way of collaboration between Git and a Baseline.



Git Strategy: Branches – Remote Customer Git

First you have to tag the ‘Production’ branch (for example **PRD-25-03-2017** as shown on the figure above. Integrator will merge UAT Branch to Production Branch as soon Business will give green light for go-live. Features are now ready to be depoyed to Production Environment. After deployment, Production Branch has to be tagged.

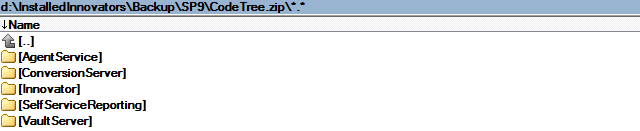
Then you need to perform the following steps on branches, that is not yet deployed to the production environment (for example **Dev-Sprint-X** as shown on figure 2):

1. Specify location of new backups for CI running, or replace old backups with new.
2. Set name of the latest tag on which deployment has occurred to the property ‘Last.Commit.Before.Current.Release’ (for example **PRD-25-03-2017**)

**NOTE: If production backups prepared incorrectly or steps above are not performed – any further result of the CI is invalid!**

In order to prepare a valid code tree backup - navigate to the physical location of a production instance of the Innovator and copy it to a temporary folder. After you have copied code tree from production server you need to remove the following configuration files from the temporary folder:

* ./AgentService/Aras.Server.Agent.Service.exe.config
* ./ConversionServer/ConversionServer.xml
* ./Innovator/Client/Innovator.xml
* ./Innovator/Server/Innovator.xml
* ./SelfServiceReporting/SelfServiceReport.xml
* ./VaultServer/VaultServer.xml
* ConversionServerConfig.xml
* InnovatorServerConfig.xml
* SelfServiceReportConfig.xml
* VaultServerConfig.xml

Now code tree is ready to be added to an archive like this (the structure is important! Pu components to archive’s root): 

Then put resulting archive to a location, accessible for a build script.

The other part pf a baseline is a database backup. To prepare a database backup use the following command line.

sqlcmd -S (local)\MSSQL2014 -Q " BACKUP DATABASE [**ProductionDatabaseName**] TO DISK = '**\\prod-backups-server\prod-backups\last-release\latest\DB.bak'** WITH NOFORMAT, INIT, COMPRESSION".

The database backup recommended to be placed near the code tree backup.

NAnt script settings file has properties that point to a location of the baseline. The default path is \\prod-backups-server\prod-backups\last-release\latest, where **prod-backups-server** - a server, where production backups are put published, **prod-backups** - shared folder name, where the backups are published, **last-release** - expected folder inside the network share. It is required to distinguish releases and **latest** - expected folder (or symbolic link to the folder) with the latest production CodeTree.zip and DB.bak backups.

If backups stored locally you have three options:

1. Setup machine specific includes. This approach is applicable only for developer batch scripts such as SetupInnovatorHere.bat and ContinuousIntegration.bat.
2. Change default value in settings file .\AutomatedProcedures\Default.Settings.include.

NOTE: Do not commit new value to the repository as all environments are dependent from this value. You can ignore this only if you have the same baseline location for all environments.

1. Setup your local machine as prod-backups server

To do this go through these steps:

Put a record for name prod-backups-server into C:\Windows\System32\drivers\etc\hosts

127.0.0.1 prod-backups-server

Make prod-backups-server an optional NetBIOS name for the local machine. This can be made via Windows registry

* Set OptionalNames manually ("New -> String Value" if you need one optional name, or "New -> Multi-String Value" if you need several names)

Windows Registry Editor Version 5.00

[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters]

"OptionalNames"="prod-backups-server"

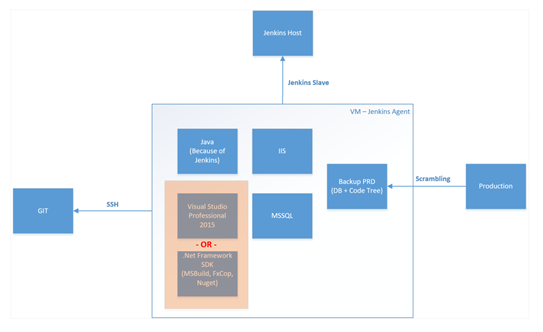
* Restart Server service (Supports file, print, and named-pipe sharing over the network for the computer.)

Create a network share with name "prod-backups" on local machine and give READ access to the share for Windows account which will run restoration process (most likely your Windows account) AND for Windows account under which SQL Server will connect to the network share.

## Environment overview



### CI Environment overview



1. Settings
2. Physical path to an instance of Innovator (required to perform upgrade of code tree)
3. URL of the instance (required to perform database upgrade)
4. Database name in MSSQL server (required to perform upgrade of database)
5. MSSQL server (required by language tool, during restoration instance to a baseline state, and preparation of updated production backups)
6. Location and name of Aras Agent Service (required to perform upgrade of code tree as service holds binaries and should be stopped)
7. Path to code tree and database backups (required during restoration instance to a baseline state, and preparation of updated production backups)

According parameter values have to be specified in the file AutomatedProcedures\Default.Settings.include. Those values depend on branch (dev, sit, uat, production), machine on which deploy is initialized, and user that runs the deploy. Some tips:

1. Make sure that each branch contains appropriate values for:
   1. Physical path and URL to a target Innovator instance.
   2. MSSQL server and database name.
   3. Location and name of Aras Agent Service
2. For branches dev-\*, sit, uat
   1. Check that path to code tree and database backups contain backups, and that the backups are up to date with current production server state (which should be equal to a baseline commit).

Check that every resource and operation performed by the deploy is available for user that runs it. Deployment script may require access to target directory, to manage settings of the directory, to manage Windows Services, to manage IIS

1. Deployment artifacts
2. Deployment package
   1. AutomatedProcedures.
   2. AML-packages.
   3. Code tree.
   4. Pre- and post-scripts.
3. Utility information
   1. Starting Git tag
   2. Head commit hash
   3. Version of installed Innovator
4. Target environment
   1. Physical path to an instance of Innvoator
   2. Url of the instance
   3. MSSQL server
   4. Database name in the MSSQL server
   5. Location an1d name of Aras Agent Service
   6. Path to code tree and database backups
5. Logs

### Hardware requirements

Build node:

|  |  |
| --- | --- |
| Operating system | Windows Server 2008 R2 or higher |
| RAM | 4 GB |
| CPU | 1 core @3GHz |
| The disk’s size | 50GB-150GB |

Deployment node (SIT):

NOTE: values depend on number of estimated instances count.

|  |  |
| --- | --- |
| Operating system | Windows Server 2008 R2 or higher |
| RAM | 8-16 GB |
| CPU | 1-2 core @3GHz |
| The disk’s size | 150GB-1TB |

UAT environment should be equal to production environment. Production environment should be provided by Aras Support.

### Software Requirements

1. MSSQL Server

* MSSQL Server cannot be installed without .NET 3.5
* Login ‘sa’ should be created during installation
* After the installation you need to create the following logins:
  + innovator (innovator database owner)
  + innovator\_regular (innovator database reader, required since 11.0 SP12)

1. Windows features
   1. .NET Framework 3.5, Web Server (IIS), ASP.NET 4.5/4.6
2. SSH agent
3. Java
4. MSBuild Tools
5. (For Innovator 12.0 and higher) .NET Framework 4.7.2 Developer Pack
6. (For Innovator 12.0 and higher) ASP.NET Core/.NET Core: Runtime & Hosting Bundle 2.1.4

## Manual Deploy



### Manual deployment from Git repository (Deploy.bat)

Prerequisites for this approach:

* All Innovator components exist on one single node within one directory.
* Build agent has all required tools to evaluate diff
* If deployment agent is not a Build agent than Build agent should have access to the following components of Deployment agent: Full access to Innovator directory, IIS, windows service management.
* Production Innovator instance or its equivalent.

This approach evaluates diff since last release to Production and applies delta to a target Innovator server, which components specified in the AutomatedProcedures\Default.Settings.include

Steps to perform manual deployment:

1. Checkout branch that should be deployed, for example sit.
2. If target server (and branch name) **not production** server – [setup an instance of Innovator from production backups](#_Setup_baseline_Innovator_1).
3. It may be required to restart IIS on target server in order to drop existing connections to Innovator.
4. In the AutomatedProcedures\Default.Settings.include specify valid values for:
   1. Path.To.Installed.Innovator
   2. Url.Of.Installed.Innovator
   3. MSSQL.Server
   4. MSSQL.Database.Name
   5. MSSQL.SA.User and MSSQL.Innovator.User
   6. MSSQL.SA.Password and MSSQL.Innovator.Password
   7. Agent.Service.Name
   8. Agent.Service.Host.Name
   9. Url.Of.Deployment.Server
   10. Last.Commit.Before.Current.Release
5. Run Deploy.bat.
6. If target server (and branch name) **is production** server – create tag on the branch and prepare production backups.

### Deploy from deployment package

Deployment from a package allows to prepare zip archive and then use it on any amount of deployment agents. The package contains tools required to perform the deploy, so there are no additional requirements for deployment agents to requirements from Innovator itself.

1. Create a zip archive with deployment package and scripts for deploy
   1. Checkout branch that you want to create the zip from, for example sit.
   2. Run the AutomatedProcedures\CreateZipWithDeploymentPackageAndScripts.bat file. As result you will have the zip in the root of your repository.
2. Deploy deployment package. **Repeat this step for every deployment agent**.
   1. Unzip the package to a temporary folder at deployment agent.
   2. Go to the temporary folder where you unzipped in (further this folder is called as TempFolder).
   3. In the TempFolder\AutomatedProcedures folder you have to modify Default.Settings.include file and put there valid values for passwords, url, etc.
   4. Then run the TempFolder\AutomatedProceduresDeployFromZipWithDeploymentPackageAndScripts.bat file.
   5. The batch script will ask you what components to upgrade. Type ‘Y’ for those component that should be upgraded at current deployment agent.

NOTE: Make sure that last component you upgrade is SQL server.

### Setup baseline Innovator instance

This section describes how to setup Innovator instance using Git repository and production baseline.

Requirements:

1. IIS
2. MSSQL Server
3. Production baseline:
   1. Code Tree backup
   2. Database backup

Steps to do:

1. In the AutomatedProcedures\Default.Settings.include specify valid values for:
   1. Path.To.CodeTree.Zip
   2. Path.To.DB.Bak
   3. Path.To.Installed.Innovator
   4. Url.Of.Installed.Innovator
   5. MSSQL.Server
   6. MSSQL.Database.Name
   7. MSSQL.SA.User and MSSQL.Innovator.User
   8. MSSQL.SA.Password and MSSQL.Innovator.Password
   9. Agent.Service.Name
   10. Agent.Service.Host.Name
   11. Url.Of.Deployment.Server
   12. Last.Commit.Before.Current.Release
2. If required set optional installation path. To do so edit SetupInnovator.bat and uncomment ‘*REM SET PathToSetupRootDirectory = "R:\Setup\Innovator\To\This\Folder\%SetupSpecificName%"*’, and provide custom installation path.
3. Run the AutomatedProcedures\SetupInnovatorFromBaseline.bat, which will create new Innovator instance from release backups.

### Restore installed instance of Innovator to a baseline state

In some cases, it is required to restore state of a certain Innovator instance to its clean state (production like, or baseline state). For example, for testing purposes, when you have to test deployment procedure itself, or when during tests the instance was changed drastically and has to be restored.

Prerequisites for this action are:

1. All Innovator components exist on one single node within one directory.
2. If deployment agent is not a Build agent than Build agent should have access to the following components of Deployment agent: Full access to Innovator directory, IIS, windows service management.
3. You have access to IIS management
4. You have access to MS SQL Server
5. Production baseline:
   1. Code Tree backup
   2. Database backup

Steps to do:

1. In the AutomatedProcedures\Default.Settings.include specify valid values for:
   1. Path.To.CodeTree.Zip
   2. Path.To.DB.Bak
   3. MSSQL.Server
   4. MSSQL.SA.User
   5. MSSQL.SA.Password
   6. Path.To.Installed.Innovator
2. It may be required to restart IIS on target server in order to drop existing connections to Innovator.
3. Run the AutomatedProcedures\RestoreInnovatorFromBaseline.bat, which will rewrite targeted instance’s code tree and database from baseline backups.

## Setup Continuous Integration for Jenkins



### Setup Jenkins on Master node.

You can take necessary installer from <https://jenkins.io/>. Run installer and follow the default steps. After it finished and opened Jenkins in the browser copy and paste initial admin password. Install suggested plugins.

Create a new user.

Make sure that you have the following plugins installed:

[Pipeline](https://wiki.jenkins-ci.org/display/JENKINS/Pipeline+Plugin) minimal version: 2.5. A suite of plugins that lets you orchestrate automation, simple or complex. See [Pipeline as Code with Jenkins](http://jenkins-ci.org/solutions/pipeline/) for more details.

[Pipeline: Multibranch](https://wiki.jenkins-ci.org/display/JENKINS/Pipeline+Multibranch+Plugin) minimal version: 2.12. Enhances Pipeline plugin to handle branches better by automatically grouping builds from different branches.

[Pipeline Utility Steps](https://plugins.jenkins.io/pipeline-utility-steps) minimal version: 1.5.1. Expand possibilities for pipeline jobs, like zip/unzip.

[Credentials Plugin](http://wiki.jenkins-ci.org/display/JENKINS/Credentials+Plugin) minimal version: 2.1.12. This plugin allows you to store credentials in Jenkins.

[Credentials Binding Plugin](http://wiki.jenkins-ci.org/display/JENKINS/Credentials+Binding+Plugin) minimal version: 1.10. Allows credentials to be bound to environment variables for use from miscellaneous build steps.

[Branch API Plugin](http://wiki.jenkins-ci.org/display/JENKINS/Branch+API+Plugin) minimal version: 2.0.7. This plugin provides an API for multiple branch based projects.

[Folders Plugin](https://wiki.jenkins-ci.org/display/JENKINS/CloudBees+Folders+Plugin) minimal version: 5.18. This plugin allows users to create "folders" to organize jobs. Users can define custom taxonomies (like by project type, organization type etc). Folders are nestable and you can define views within folders. Maintained by CloudBees, Inc.

[Git plugin](https://plugins.jenkins.io/git) minimal version: 3.0.5. This plugin integrates [Git](http://git.or.cz/) with Jenkins.

[Windows Slaves Plugin](https://plugins.jenkins.io/windows-slaves) minimal version: 1.2. Allows you to connect to Windows machines and start slave agents on them.

[Email Extension Plugin](https://plugins.jenkins.io/email-ext) minimal version: 2.57. This plugin allows you to configure every aspect of email notifications. You can customize when an email is sent, who should receive it, and what the email says.

[SSH Agent Plugin](http://wiki.jenkins-ci.org/display/JENKINS/SSH+Agent+Plugin) minimal version: 1.15. Allows to perform operations like set Git tag.

[Workspace Cleanup Plugin](http://wiki.jenkins-ci.org/display/JENKINS/Workspace+Cleanup+Plugin) minimal version: 0.34. Provides possibility to easily remove old files from workspace directory.

[Timestamper](http://wiki.jenkins-ci.org/display/JENKINS/Timestamper) minimal version: 1.8.8. Adds timestamps to a log file.

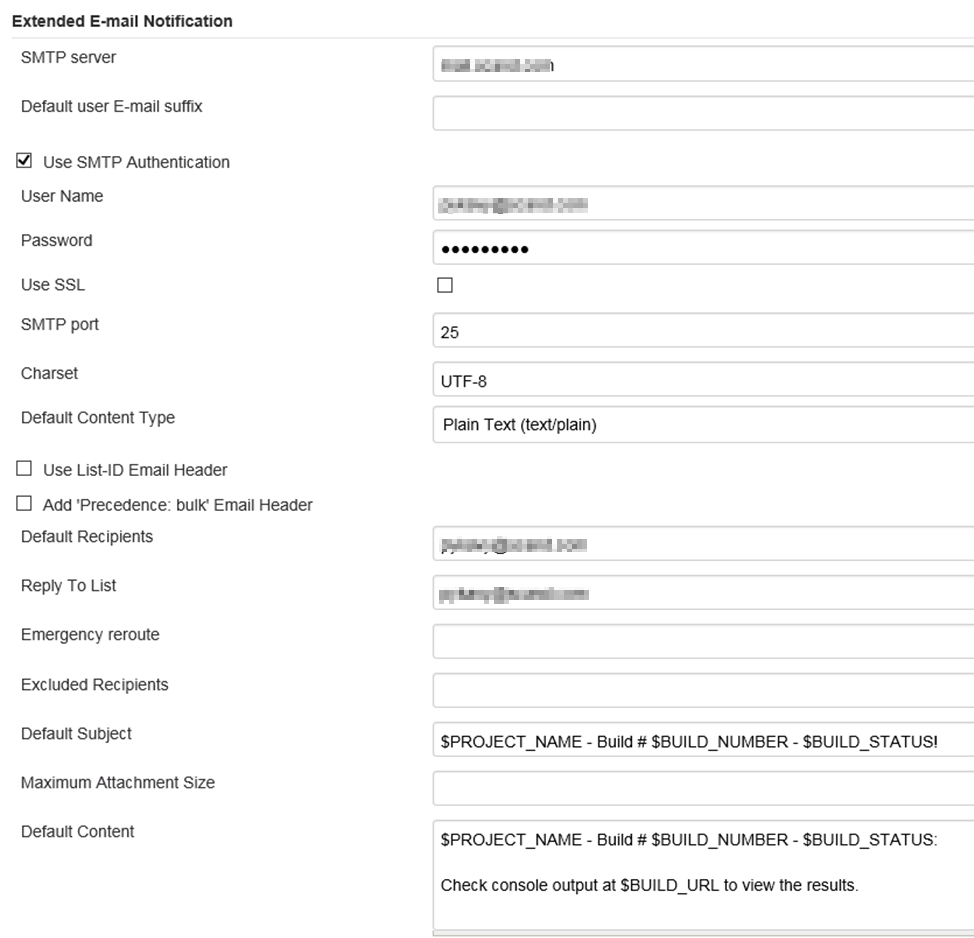
[NUnit](https://plugins.jenkins.io/nunit) minimal version: 0.21. This plugin transforms NUnit test reports so they can be recorded by Jenkins' JUnit features. Required to see unit test results at job results.

[HTML Publisher](http://wiki.jenkins-ci.org/display/JENKINS/HTML+Publisher+Plugin) minimal version: 1.14. This plugin publishes HTML reports. Required to see client test results at job results.

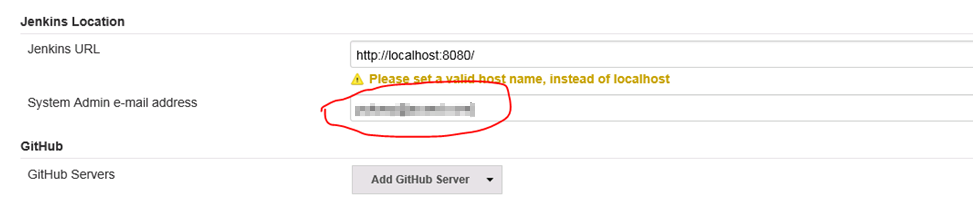
NOTE: plugins could have their own dependencies, which are not resolved automatically. If you install plugins manually you also will have to check plugin’s dependencies and setup them along with plugins from the list above.

If you don’t find some you should install it. To do it ‘Manage Jenkins’ -> ‘Manage Plugins’ select plugins. After you checked all required plugins press ‘Install’. After the installation you need to restart Jenkins. It can be done automatically by check the ‘restart Jenkins after installation’ checkbox, or manually using link <http://your_jenkins_server:8080/Restart>

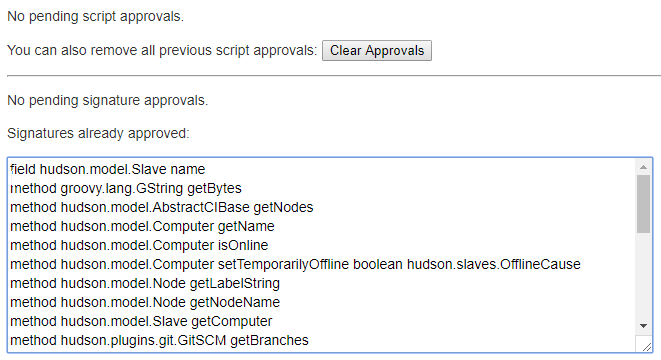
In order to be able to get e-mail with build results open Jenkins select ‘Manage Jenkins’ then ‘Configure System’. Scroll to the ‘Extended E-mail Notification’ section and fill its properties.



Don’t forget to define your system Admin email address in Manage Jenkins -> Configure System



CI scripts use certain Jenkins methods, that by default not allowed to run by Jenkins. To be able to run CI scripts you need to approve the following scripts in <http://jenkins-host:8080/scriptApproval/>:



To approve them you need to navigate to your Jenkins’ script console page <http://jenkins-host:8080/script> and the following script:

def signaturesToApprove =[

'field hudson.model.Slave name',

'method groovy.lang.GString getBytes',

'method hudson.model.AbstractCIBase getNodes',

'method hudson.model.Computer getName',

'method hudson.model.Computer isOnline',

'method hudson.model.Computer setTemporarilyOffline boolean hudson.slaves.OfflineCause',

'method hudson.model.Node getLabelString',

'method hudson.model.Node getNodeName',

'method hudson.model.Slave getComputer',

'method hudson.plugins.git.GitSCM getBranches',

'method hudson.plugins.git.GitSCM getUserRemoteConfigs',

'method hudson.plugins.git.GitSCMBackwardCompatibility getExtensions',

'method hudson.plugins.git.UserRemoteConfig getUrl',

'method java.lang.String getBytes',

'new hudson.slaves.OfflineCause$UserCause hudson.model.User java.lang.String',

'new java.io.File java.lang.String',

'staticField hudson.model.Result SUCCESS',

'staticMethod hudson.model.User current',

'staticMethod jenkins.model.Jenkins getInstance',

'staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods getAt java.lang.Object java.lang.String',

'staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods getText java.net.URL',

'staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods toURL java.lang.String',

'staticMethod org.codehaus.groovy.runtime.EncodingGroovyMethods encodeBase64 byte[]'

]

for (i = 0; i < signaturesToApprove.size(); i++) {

org.jenkinsci.plugins.scriptsecurity.scripts.ScriptApproval.get().approveSignature(signaturesToApprove[i])

}

### Build node.

Software can be installed through 'AutomatedProcedures\PowershellScripts\SetupEnvironment.ps1' script execution. It uses Chocolatey tool to download and install required software from the Internet. Open PowerShell console as admin in the root of the repository and run the command:

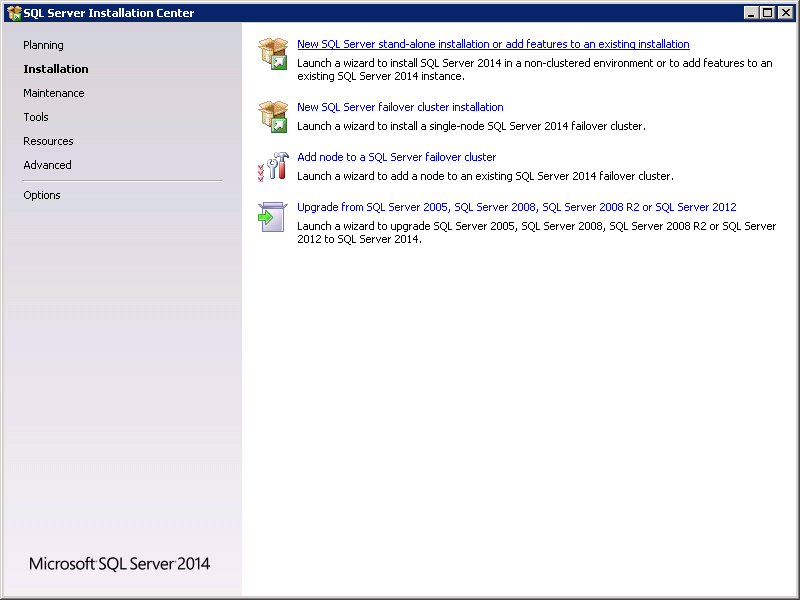
Set-ExecutionPolicy Bypass -Scope Process -Force; & ".\AutomatedProcedures\PowershellScripts\SetupEnvironment.ps1"

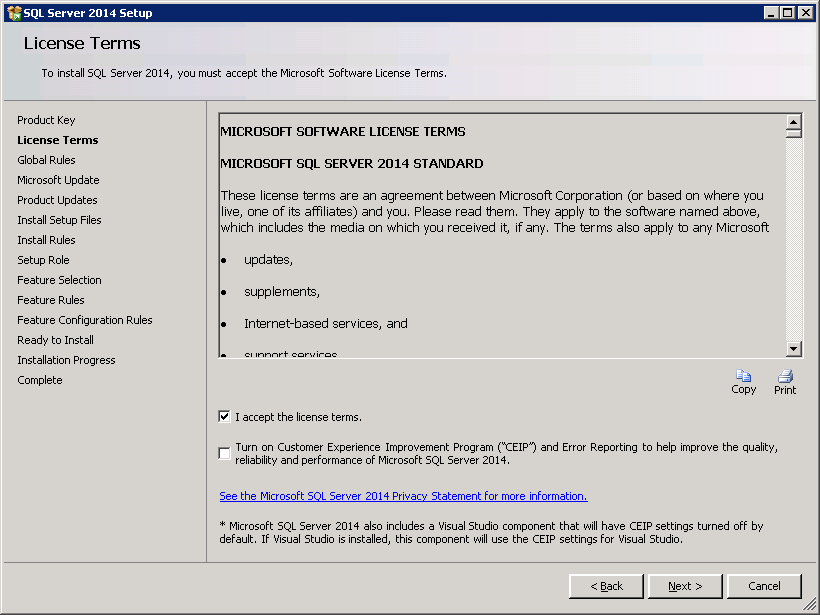
### Setup SQL Server.

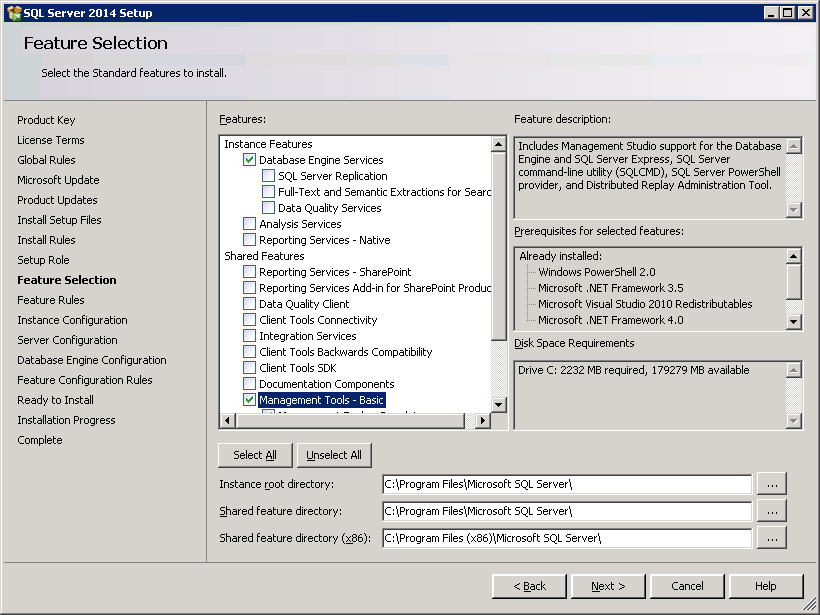
Instance of the SQL Server meets the following requirements:

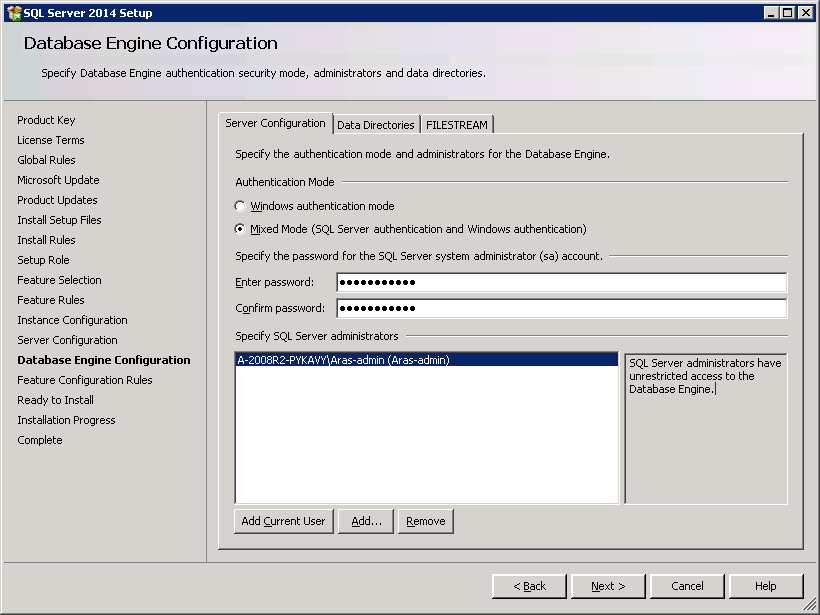
* Installed .Net framework 3.5.
* Supported version of SQL Server is 2008 R2 or above.
* There are no limitations from the deployment procedure for edition of the MSSQL Server, however if your Innovator use the ‘Report Service’ feature it should be at least ‘Express with Advanced Services’ (see <https://msdn.microsoft.com/en-us/library/cc645993(v=sql.105).aspx#Reporting> ). Regardless of an edition’s capabilities, you should check license agreement, so there are no violations.

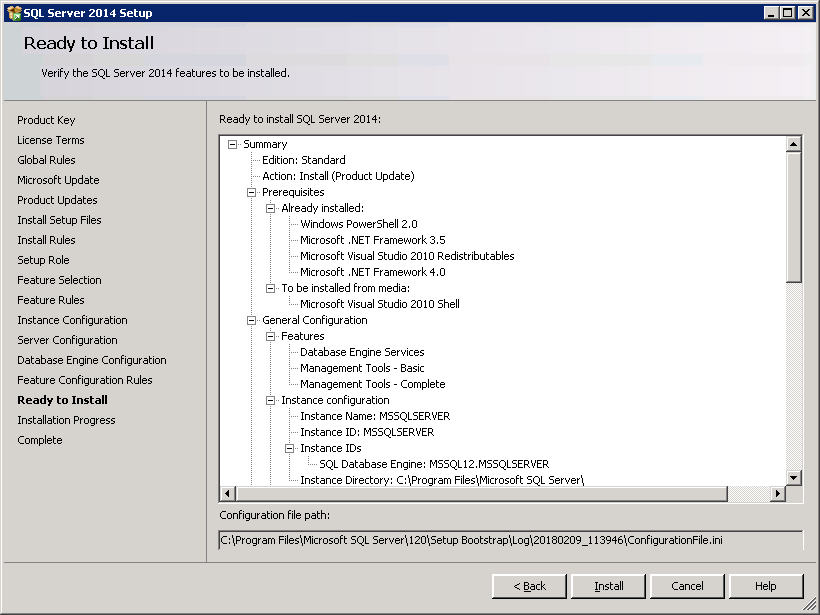
Run setup.exe. Then select ‘Installation’ -> ‘New SQL Server stand-alone installation or add features to an existing installation’ and follow the steps as in the screenshots below.





Select ‘Database Engine Services’ and ‘Management Tools – Basic’

Don’t forget to select “Mixed Mode” for Authentication Mode. Then enter and confirm password for ‘sa’ user. And click the button ‘Add Current User’ to add it in SQL Server administrators.

Click install button to finish the installation.

Then login into DBMS as ‘sa’ with password you specify during installation and create login ‘innovator’.

If you have Innovator version SP12 and later you should also create ‘innovator\_regular’ login.

To create a login - use the following command:

sqlcmd -S (local)\MSSQL2012 -Q "CREATE LOGIN innovator WITH PASSWORD = 'innovator'"

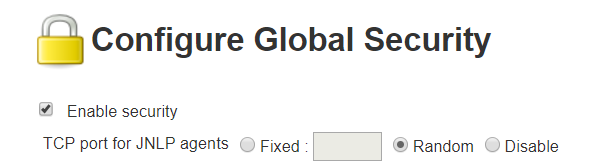
If you use custom password – do not forget to specify password in the deployment procedure settings.

NOTE: it is recommended to limit the SQL server instance (in properties) to consume **not more than 65% of total RAM installed** to prevent overload (for example, if totally OS has 6 GB, SQL server must me limited from UP to maximum 4 GB (4096 MB)).

### Deployment node: configure Jenkins Master node, required software, attach to the Master node

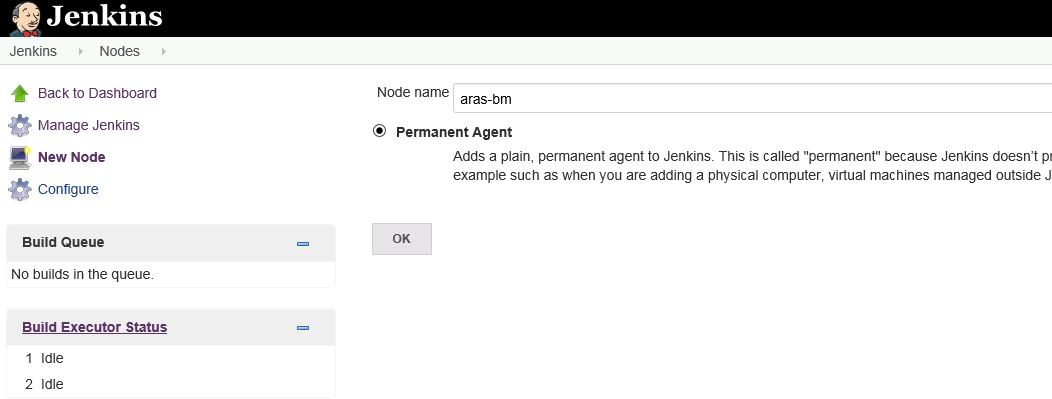
Software requirement for deployment nodes are:

1. Java
2. For Innovator 12.0 and higher only:
   1. Setup .Net Framework 4.7.2: <https://dotnet.microsoft.com/download/thank-you/net472-developer-pack>
   2. Setup .Net core 2.1.4: <https://dotnet.microsoft.com/download/thank-you/dotnet-runtime-2.1.4-windows-hosting-bundle-installer>
   3. Restart deployment agent machine.

First you need to create virtual node at Master node. Follow next steps to accomplish this. Make sure that ‘Random’ radio button is checked for TCP port for JNLP agents. To do it click ‘Manage Jenkins’ from the left menu and then click ‘Configure Global Security’ and select the ‘Random’ radio button as shown below:

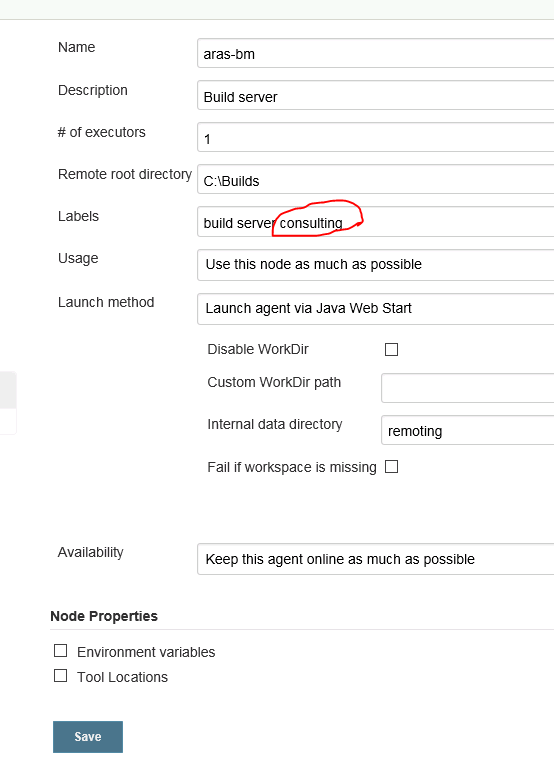
Don’t forget to click ‘Save’ button in the end of page.

Create new node on Jenkins master using following link <http://localhost:8080/computer/new>. Or in Jenkins UI: ‘Manage Jenkins’ -> ‘Manage Nodes’ -> ‘New Node’. Here is an example of new node’s configuration:



After you click ‘OK’ button the new form will appear. Fill it like in the example below.

In Jenkinsfile each node contains your own label that define the machine where the part of pipeline will be execute so you should make sure that Jenkins nodes contains all labels defining in the Jenkinsfile. In the example below, it is ‘consulting’ label;

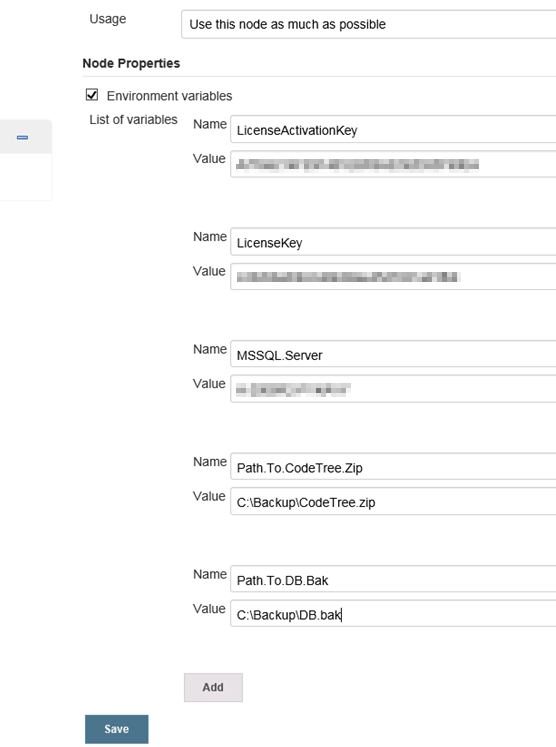


You should configure environment variables for node which will be used during executing pipeline on the specific node. To do it for ‘aras-bm’ go to <http://localhost:8080/computer/aras-bm/configure> and add next environment variables:

1. LicenseActivationKey
2. LicenseKey
3. MSSQL.Server
4. Path.To.CodeTree.Zip
5. Path.To.DB.Bak

You can obtain LicenseActivationKey and LicenseKey from the <http://www.aras.com/support/LicenseKeyService/> .

MSSQLServer is an instance of the SQL Server used for continuous integration. For example:

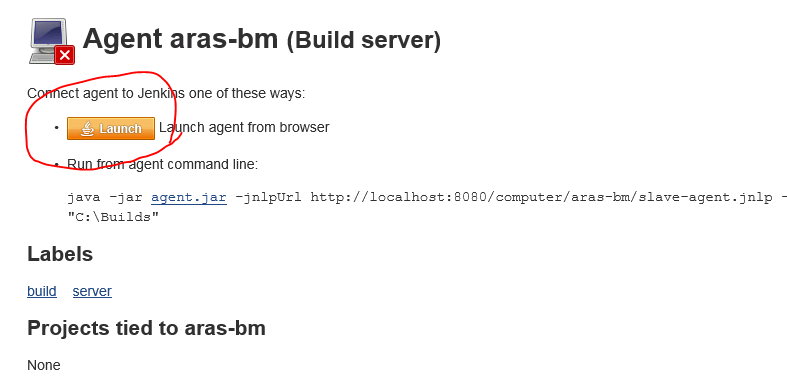


To attach deployment node as Jenkins agent - do the following:

Make sure that new Jenkins agent has Java installed. It is required to have Java on the deployment node. To Install Java - download the latest version from the <https://www.java.com> and go through standard steps.

Create new node on Jenkins master using following link <http://localhost:8080/computer/new>. Or in Jenkins UI: ‘Manage Jenkins’ -> ‘Manage Nodes’ -> ‘New Node’

Configure the node’s environment variables as described in the ‘Configure Jenkins environment variables’ section but use another link (for example for node with name ‘sit’ it will be <http://localhost:8080/computer/sit/>configure).

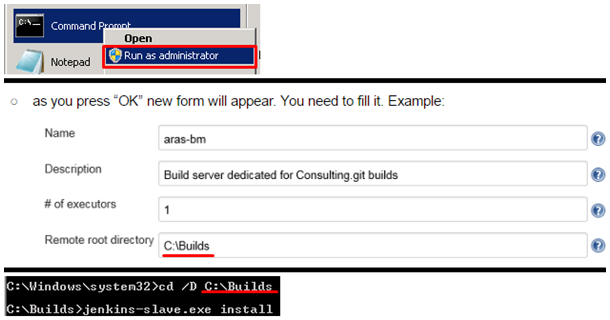
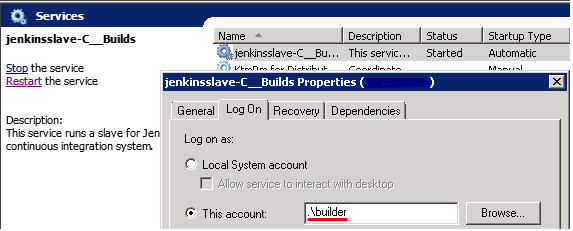
Select created node and connect agent using the option ‘Launch agent from browser’. This step should be initiated from the slave machine. This step requires installed Java on the slave machine. 

After Java launched and completed validation - this warning will appear. press ‘Run’

Then you need to install Jenkins as a Windows service, so the agent slave starts automatically when the machine boots. You need admin privileges to perform this step. The user should have ssh keys in the C:\users\username\.ssh. 

If it failed with error message ‘Access denied’. Run command line as admin. Navigate to the Remote root directory (that was specified in the node’s environment variables earlier) in the opened command line, then type and execute the following command:

jenkins-slave.exe install.

This command will install the service.Confirm that the service runs under user with administrator privileges and with correct ssh keys. 

**Create git tags automatically after successful deployment**

1.You need to have such parameters in the deployment job:

*BranchName* – branch to do deployment from. Also this value will be selected to define which Jenkins node to use. (i.e sit).

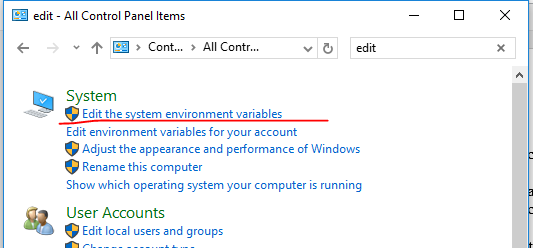
*RepositoryUrl* – your innovator product git repository URL. It will be used to checkout repository and to push tag. (i.e git@myserver:savanchuk\_innovator.git)

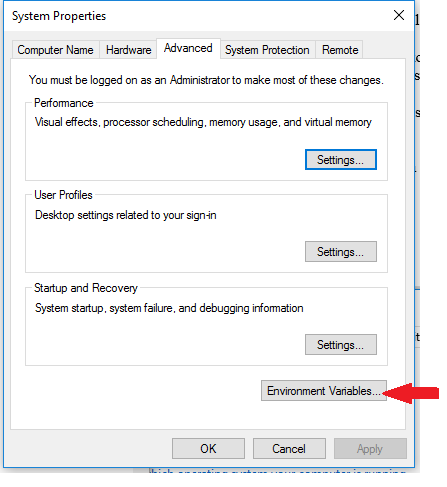
*UserNameForTag* – name of git user that will push the tag. This is user.name git configuration variable that we will set. (i.e John Doe)

*UserEmailForTag* – email of git user that will push the tag. This is user.email git variable that we will set. (i.e mylovelyemail@email.com).

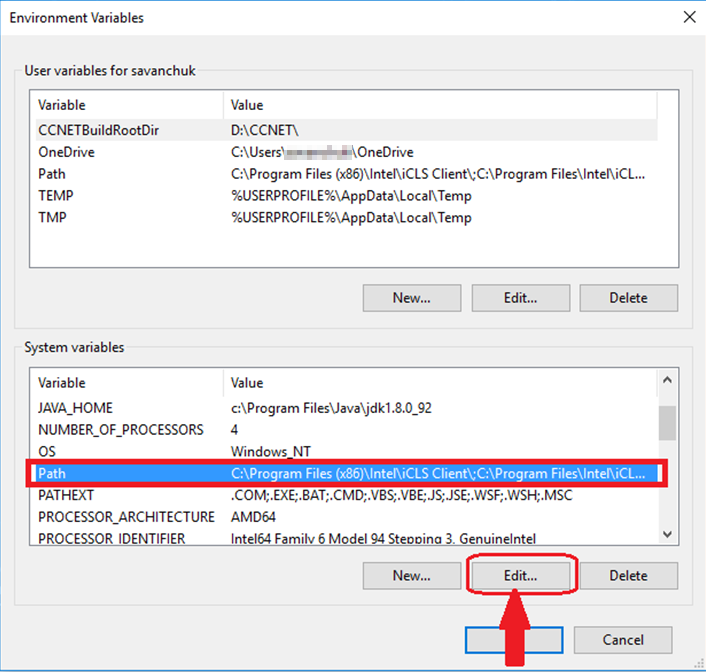
2 Add path to git ssh-agent.exe in the environment system variable PATH on a Jenkins node machine that you will use for deployment

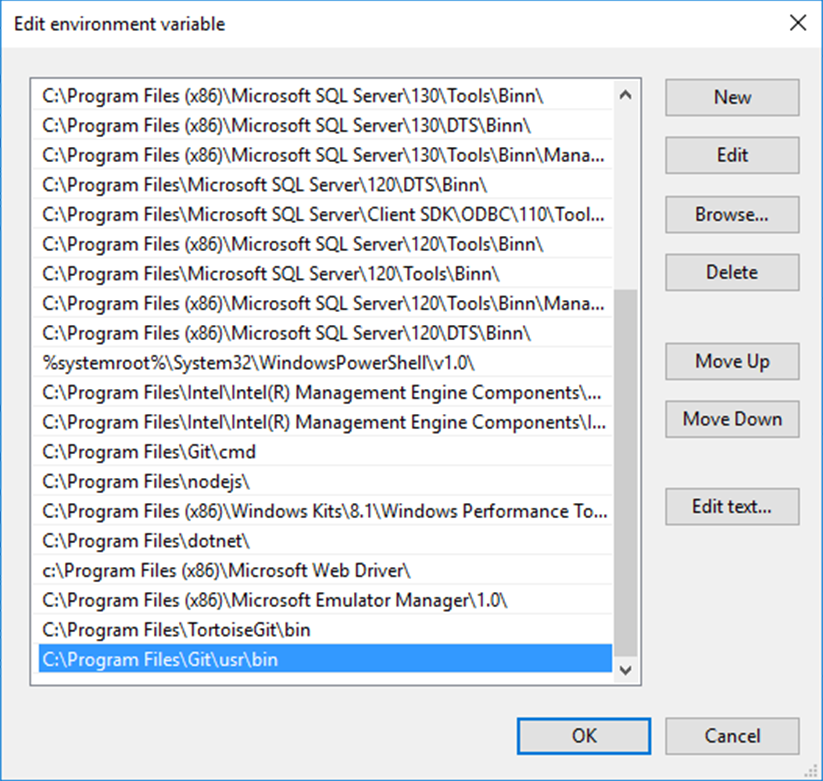
To do this:

1. On the agent node go **Control Panel**
2. Then go Edit the system environment variable
3. Further click the **Environment Variables…** button



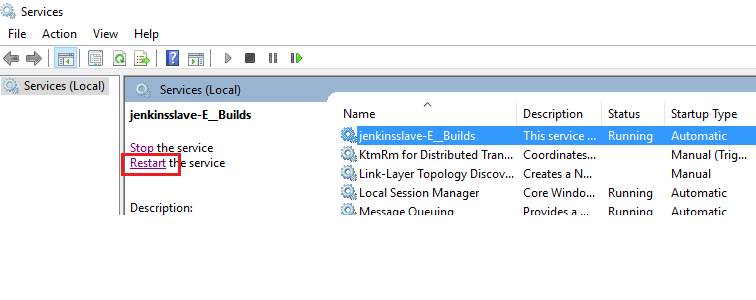
1. In the **System variables** list select the **Path** variable and click the ‘**Edit…’** button



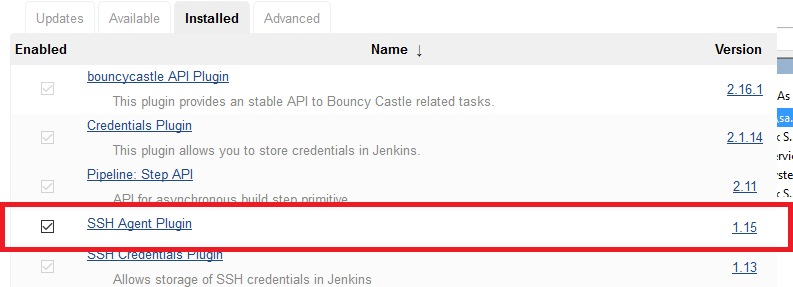
Then you need to click the **New** button and add this path: C:\Program Files\Git\usr\bin

Then press the **OK** button.

After this restart your jenkins agent in **Control Panel - > Services.**



Install the SSH Agent Plugin (Info about how to install Jenkins plugins you can find in the Documentation\Setup Jenkins CI\Setup Jenkins plugins.docx document)

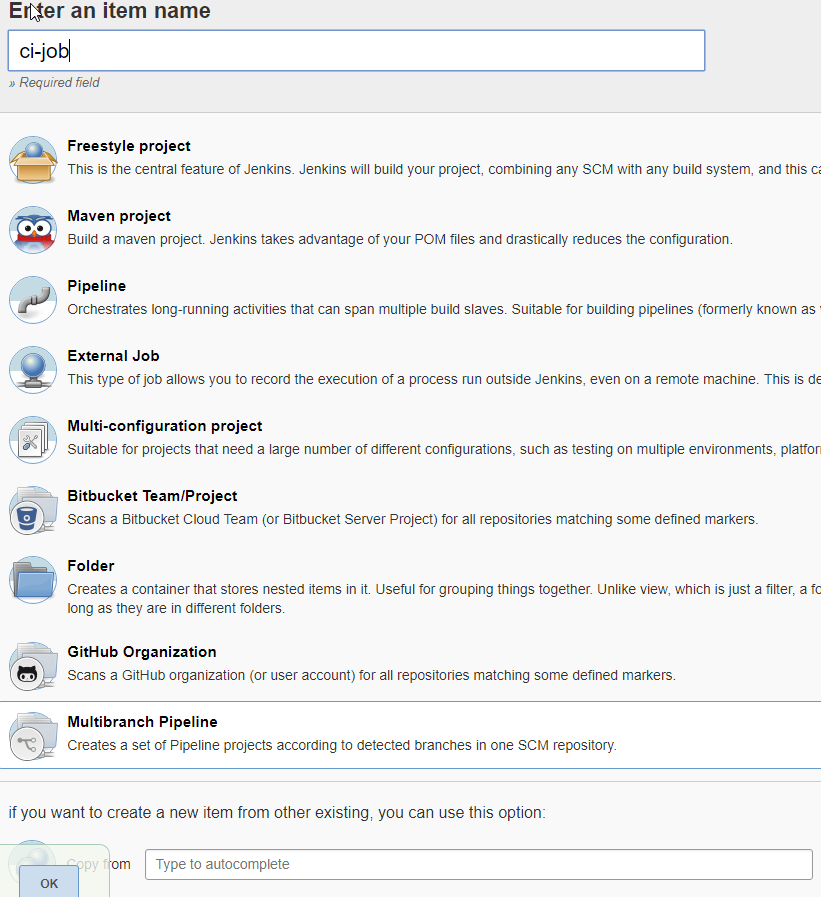


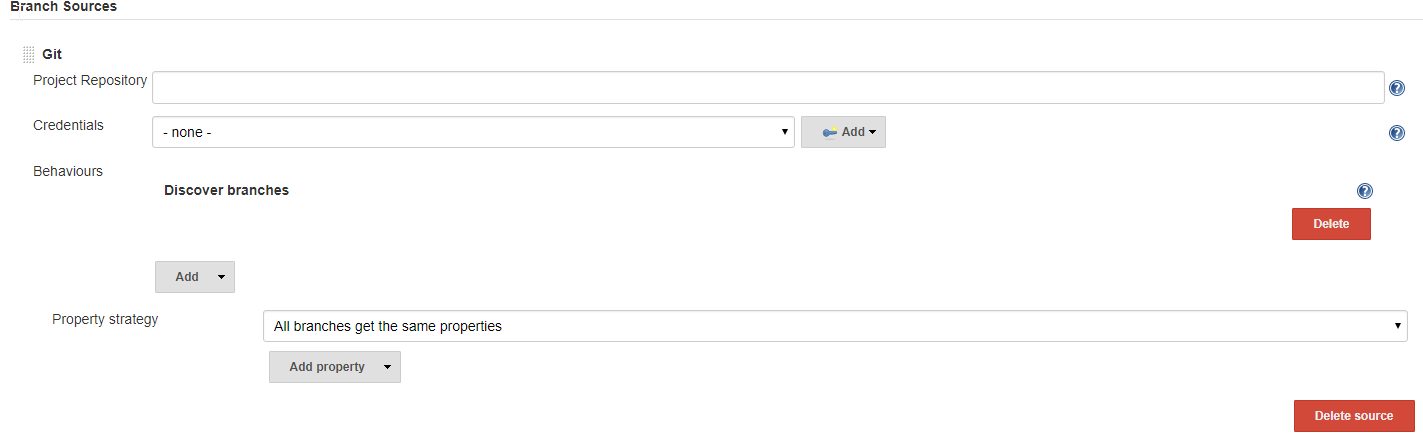
If Jenkins server will ask to restart it, please - do it.

That’s all for it. After this your Jenkins will create git tags when deployment was successful.

### Setup CI job

In TOC select ‘New Item’

Select multibranch pipeline and enter job name

Press ‘add source’ select Git:

Specify URL to repository in ‘Project repository’ field.

Select credential (or create new) that will be used to access the repository.

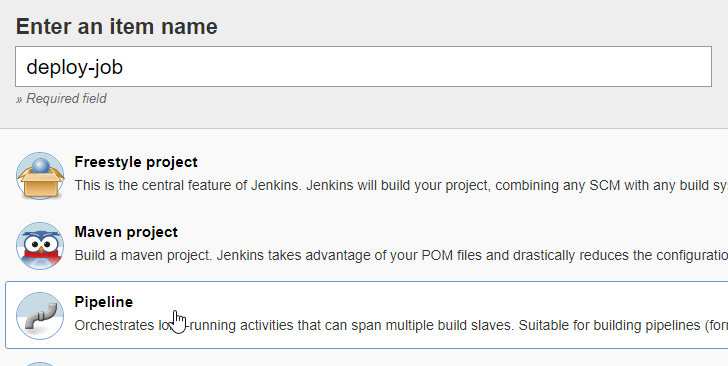
Under ‘behaviors’ press ‘Add’ then select ‘Advanced clone behaviors’. Leave appeared block unchanged.

At the bottom of a page press ‘Save’

### Setup deployment job

In TOC select ‘New Item’

Select ‘pipeline’ and enter job’s name

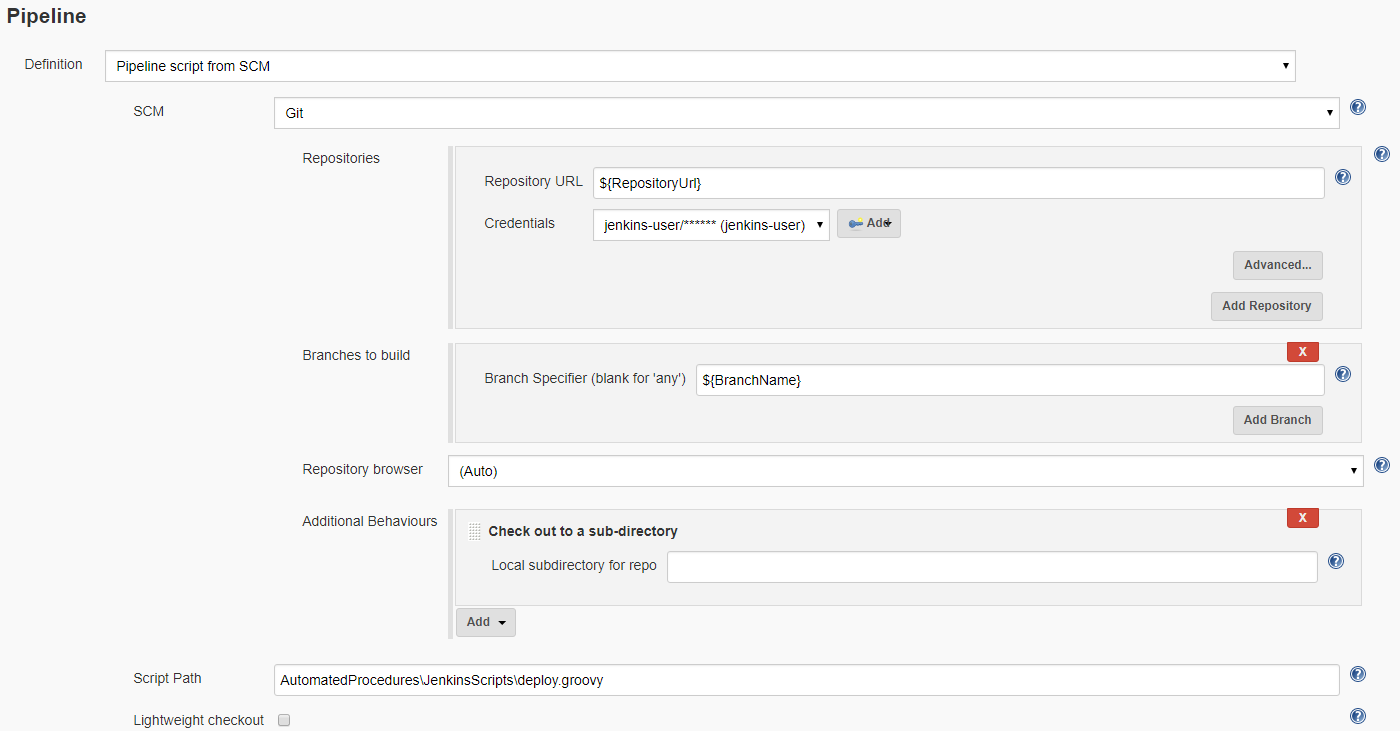


Check ‘Do not allow concurrent builds’

Check ‘This project is parameterized’ and specify the following:

* EmailFailure – those who receive a letter in the case of the fall of the build (String)
* MSSQL.Server – MS SQL server name (String)
* Path.To.CodeTree.Zip – path to appropriate code tree zip (String)
* Path.To.DB.Bak – path to appropriate database backup (String)
* LicenseActivationKey – license activation key that was received from aras.com (String)
* LicenseKey – license key that was received from aras.com (String)
* NodeLabel (deploy agent) – label of jenkins agent where you want to deploy the instance of Innovator (String)
* RepositoryUrl – URL of the repository which will be used as a source repository for build and for setup tag if it is chosen to set one (String)
* UserEmailForTag – user email which will be used during setup tag (String)
* UserNameForTag – user name which will be used during setup tag (String)
* BranchName, TagName– the name of a branch or a tag from the ${RepositoryUrl} which will be checked out during build (String)

Path.To.CodeTree.Zip and Path.To.DB.Bak explanations. For the first deployment you will need to use clean baseline of according Innovator version. The baseline could be found at Aras FTP. If customer already has production you need to get baseline of current production state. The step described in [Baseline - Production backup](#_Baseline_-_Production) section of this document.

In a pipeline section select ‘Pipeline script from SCM’ Definition. 

Fill in:

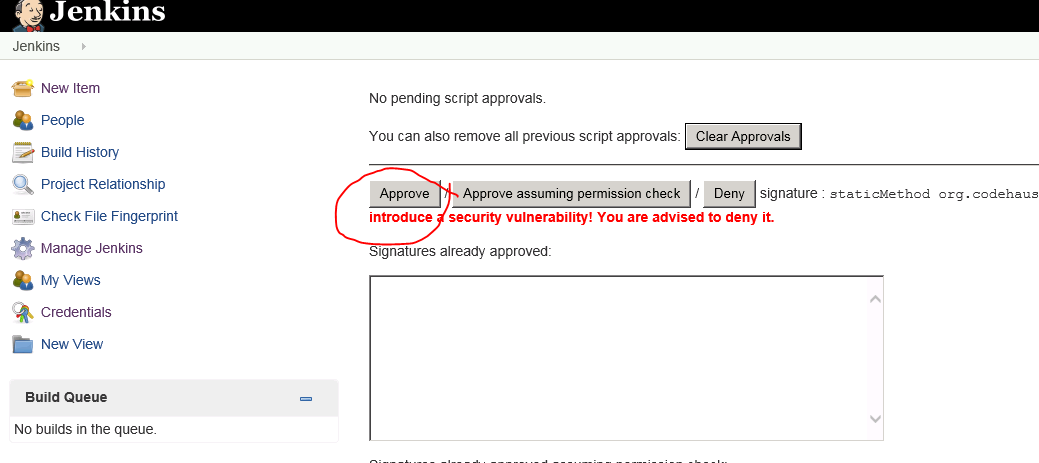
* Repository url: ${RepositoryUrl}
* Credentials: select credentials that will be used to access specified repository.
* Branches to build (depends from what parameter you have created for the job):
  + ${BranchName}
  + ${TagName}
* Script Path : AutomatedProcedures\JenkinsScripts\deploy.groovy
* Lightweight checkout: Uncheck

### Troubleshooting common errors

If you have a Run Time Exception, which states that ‘Scripts not permitted to …’ like this:



Go to ‘Manage Jenkins’ -> ‘In-process Script Approval’ and click ‘Approve’



If you have any troubles in login to innovator (HTTP error 500.21) you can try execute this command from admin cmd to resolve it:

%windir%\Microsoft.NET\Framework64\v4.0.30319\aspnet\_regiis.exe -i

If during deploy you got message “*java.io.IOException: Cannot run program "git.exe"*” – check for all slave nodes that Git is installed and then restart the **jenkisslave-C\_\_\_Builds** service. If you do not have any slave nodes check on master node that Git is installed and restart Jekins service.

## Advanced Continuous Integration settings

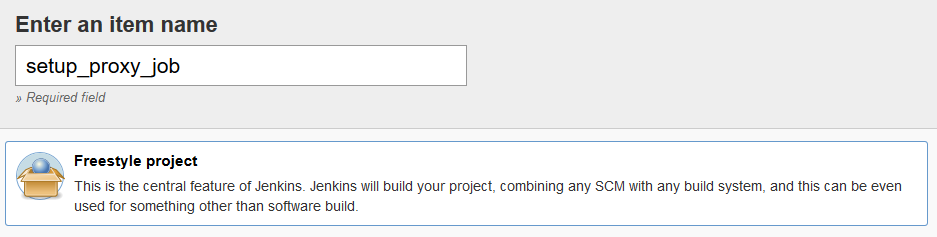


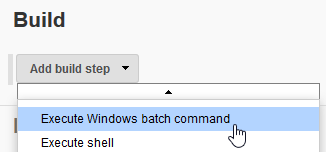
### Setup proxy

There are 2 tools that use internet access to download packages: nuget and npm.

To correctly setup both of them you need to setup configuration from the same user that runs Jenkins process. The easiest way to do this is to create simple Jenkins job and setup proxy from this job.

First you need to create the job:





Add new ‘Execute Windows batch command’ build step to it.

Locate nuget.exe. If you do not have one check in the repository: .\AutomatedProcedures\tools\.nuget\NuGet.exe. For this example imagine that repository root at D:\Work\crt\_work, so nuget will be at D:\Work\crt\_work\AutomatedProcedures\tools\.nuget\NuGet.exe

Next you need to modify Jenkins job. Add the following to the batch command:

D:\Work\crt\_work\AutomatedProcedures\tools\.nuget\**NuGet.exe config -set http\_proxy**=http://my.proxy.address:port

D:\Work\crt\_work\AutomatedProcedures\tools\.nuget\**NuGet.exe config -set http\_proxy.user**=mydomain\myUserName

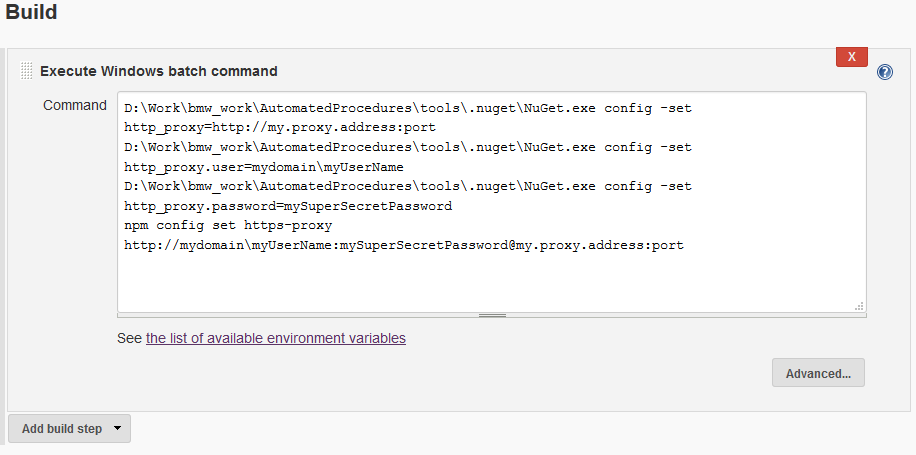
D:\Work\crt\_work\AutomatedProcedures\tools\.nuget\**NuGet.exe config -set http\_proxy.password**=mySuperSecretPassword

NOTE: underscored values should be changed according to your environment.

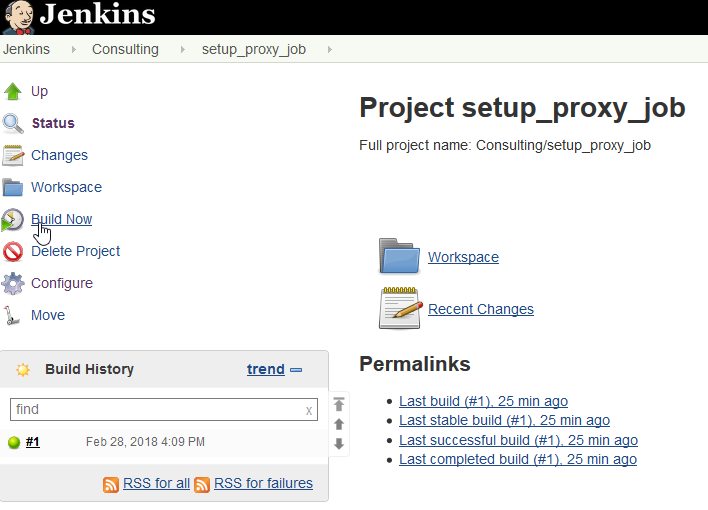
To setup npm proxy use option 1 if you are using http, and option 2 if https. Put chosen option to the batch command.

1) **npm config set proxy** http://mydomain\myUserName:mySuperSecretPassword@my.proxy.address:port

2) **npm config set https-proxy** http://mydomain\myUserName:mySuperSecretPassword@my.proxy.address:port

Resulting configuration should look like so:

Scroll down to ‘Save’ button and press it. Run the job:



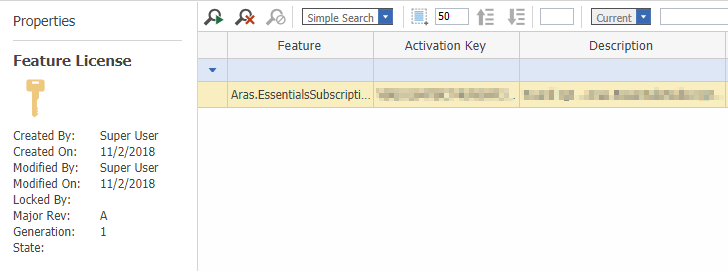
Make sure that job pass successful. As job passes the configuration should be set top work with proxy.

### Setup feature licenses

CI is able to import feature licenses to deployed instances of Aras Innovator. Usually a feature has limited activation count, however, since we are using the same environment for development\CI\deploy, we can reuse activated featured using Import Feature License functionality.

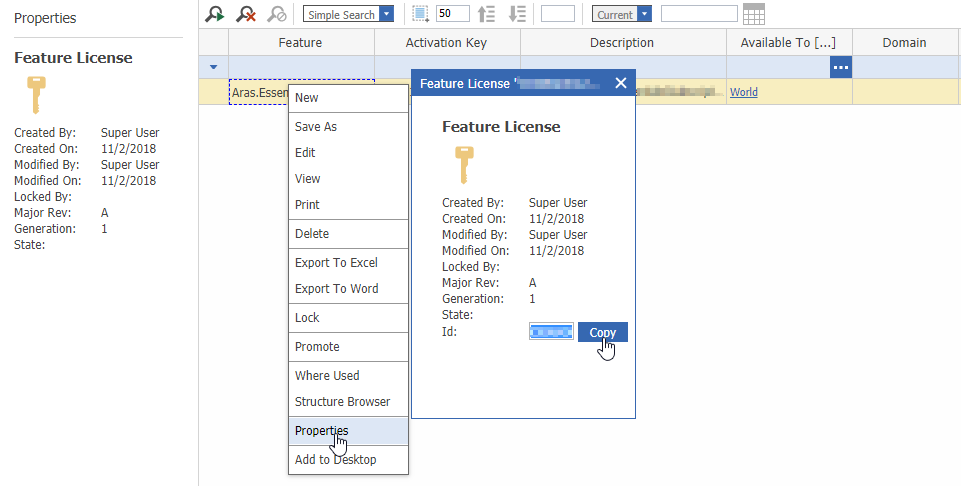
First you need to activate a feature at a target environment to any Aras Innovator instance there.

Then you need to obtain the license\_data property from activated feature. Consider the following example. We have activated Aras.EssentialsSubscription feature. Now we ca see it in the Feature Licenses (Administration section in TOC):



UI do not provide opportunity to get the license data, so we will use nash (default url is [http://localhost/*InnovatorServerAlias*/Client/scripts/nash.aspx](http://localhost/InnovatorServerAlias/Client/scripts/nash.aspx)).

To do that, we need to get an ID of the activated feature: open feature’s properties and copy its ID.



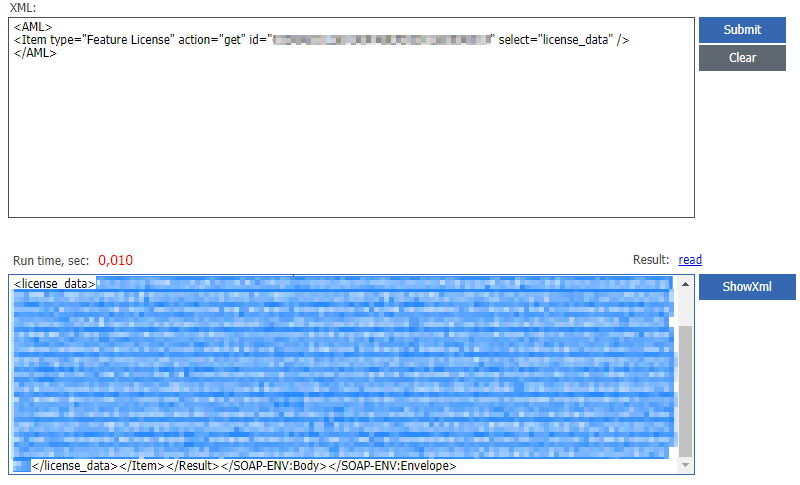
Once it is done open nash and execute the following AML:

<AML>

<Item type="Feature License" action="get" id="COPIED\_ID " select="license\_data" />

</AML>

Get the license\_data from the response:



Get the content of license\_data tag. Reformat the string – remove starting and trailing spaces or new lines (best approach is to use a text editor for this) if there are any.

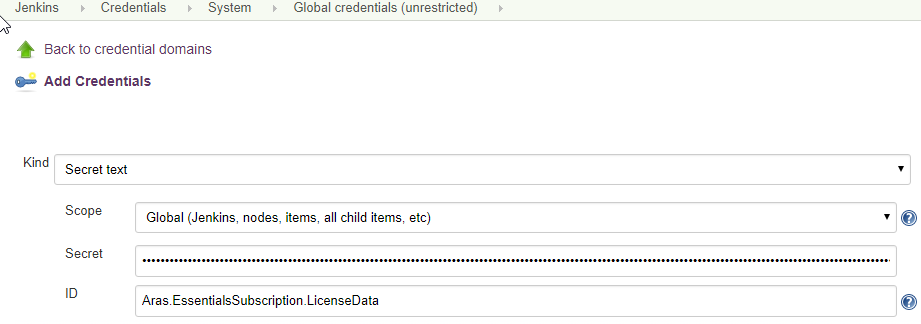
You can use obtained license data at local development environment or at Jenkins.

For local use paste the license\_data to local machine specific include as is. To do so create the Feature.License.Strings.List property and put the license data to the value attribute:

<property name="Feature.License.Strings.List" overwrite="true" value="LICENSE\_DATA” />

If you use several licenses – simply add another license data to the value tag, separating license data string by commas like so:

<property name="Feature.License.Strings.List" overwrite="true" value="LICENSE\_DATA, LICENSE\_DATA\_2, LICENSE\_DATA\_3” />

It is also possible to specify the license data for Jenkins its credentials. For that you need to create a secret text credential in your Jenkins (default link is <http://localhost:8080/credentials/store/system/domain/_/newCredentials>) For example we have created Aras.EssentialsSubscription.LicenseData credential in Jenkins and pasted license data to the Secret field. Once you pressed OK the credential will be available for Jenkins jobs (CI/Deploy). To use the credential put its Id to the AutomatedProcedures\JenkinsfileProperties.json file in the repository. Put the credential id to the featureLicenseCredentialIDs array (Added value marked as green):

{

"credentials": [

{ "$class": "StringBinding", "credentialsId": "MSSQL.Innovator.Password", "variable": "MSSQL.Innovator.Password" },

{ "$class": "StringBinding", "credentialsId": "MSSQL.SA.Password", "variable": "MSSQL.SA.Password" },

{ "$class": "StringBinding", "credentialsId": "InnovatorLicense110", "variable": "InnovatorLicense110" }

],

"featureLicenseCredentialIDs": [

“Aras.EssentialsSubscription.LicenseData"

]

}

### Setup build procedure to work at offline environment

It is possible that development, CI, SIT or another environment do not have any Internet access. Usually build procedure uses nuget to download packages used by a build procedure. In a case when an environment does not have Internet access - you need to setup build procedure to run builds, even without the Internet. There are a few things to setup:

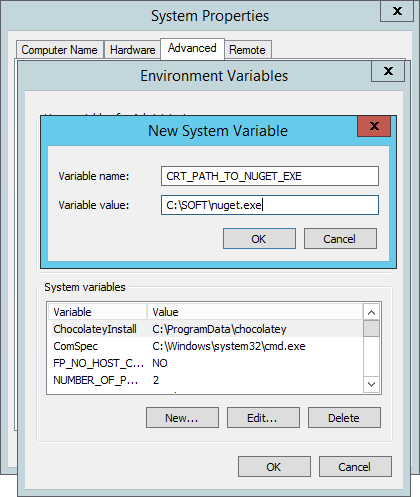
**Setup nuget.**

Download nuget from <https://dist.nuget.org/win-x86-commandline/latest/nuget.exe>.

**Make nuget available for build procedure.**

Put nuget to the C:\Program Files (x86)\NuGet\NuGet.exe

… or anywhere else, but specify the NuGet.exe’s path to the ‘CRT\_PATH\_TO\_NUGET\_EXE’ system environment variable:



**Download packages required by a build procedure and put them to a nuget cache.**

Get all required packages at environment that has access to the Internet and copy package to any convenient storage for you (for example we will use path C:\Program Files (x86)\NuGet\Packages). The list of packages to prepare is following:

7-zip.commandline

microsoft.experimental.io

nant

nant.contrib.portable

nsubstitute

nunit

nunit.runners

opencover

reportgenerator

stylecop.analyzers

system.data.hashfunction.core

system.data.hashfunction.interfaces

system.data.hashfunction.xxhash

Packages versions are dependent from Innovator version. (see packages.config of related projects inside tests and src folders)

Do not forget to add location of the folder to the nuget configuration file. To set the location for all users and processes please create a configuration file C:\Program Files (x86)\NuGet\Config\nuget.config with the following content:

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<packageSources>

<add key="crt-packages" value=" C:\Program Files (x86)\NuGet\Packages\" />

</packageSources>

</configuration>

**Setup npm packages.**

First you need to download required packages. For this action you need access to the internet, which should be performed on a different machine with internet access, or with temporary internet access at machine itself. The sequence is:

1. Clone the repository and navigate to the .\AutomatedProcedures\tools\NodeJS folder using command line (e.g. “cd /d C:\CustomerRepositoryTemplate\AutomatedProcedures\tools\NodeJS”)
2. (with Internet access) in the command line run “npm install”
3. If you did previous steps at build agent skip this step. Locate downloaded cache folders npm and npm-cache in current user’s roaming data at %userprofile%\AppData\Roaming
4. Next you need to find out user that running Jenkins (press “start” button, start to type ‘services’, locate Jenkins or Jenkins-slave service and see the user name at logon as column. As an example, we will use ‘techrunner’ user name.

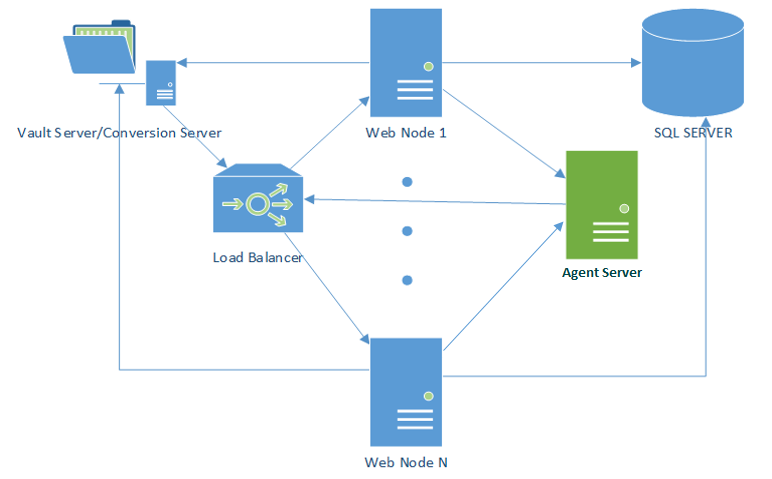
At the build agent locate the techreunner’s roaming data (user that is running Jenkins service at build agent) and copy npm and npm-cache folders from step 3 to the C:\Users\techrunner\AppData\Roaming folder.

## Setup Jenkins deploy for UAT and Production



### UAT/Prod: Innovator Components Configuration

Typical configuration resented below:



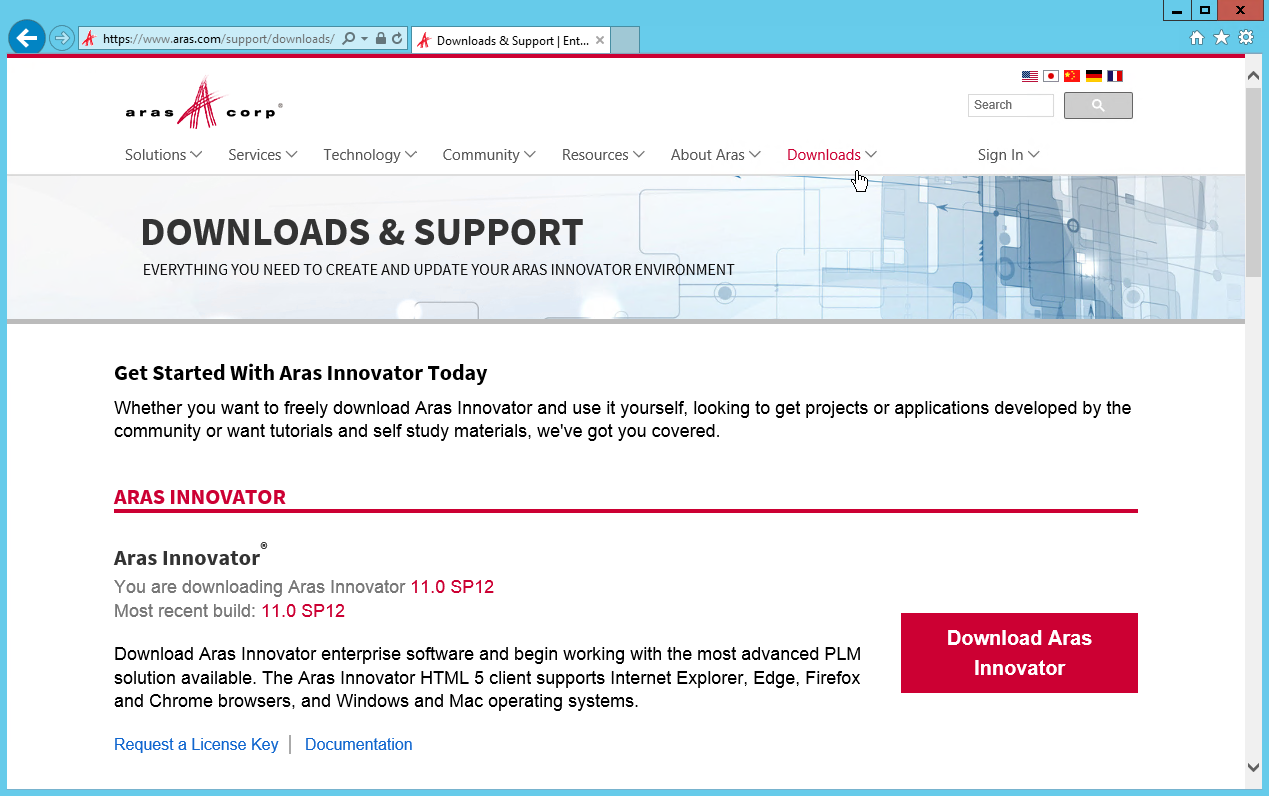
In most cases suggested configuration is:

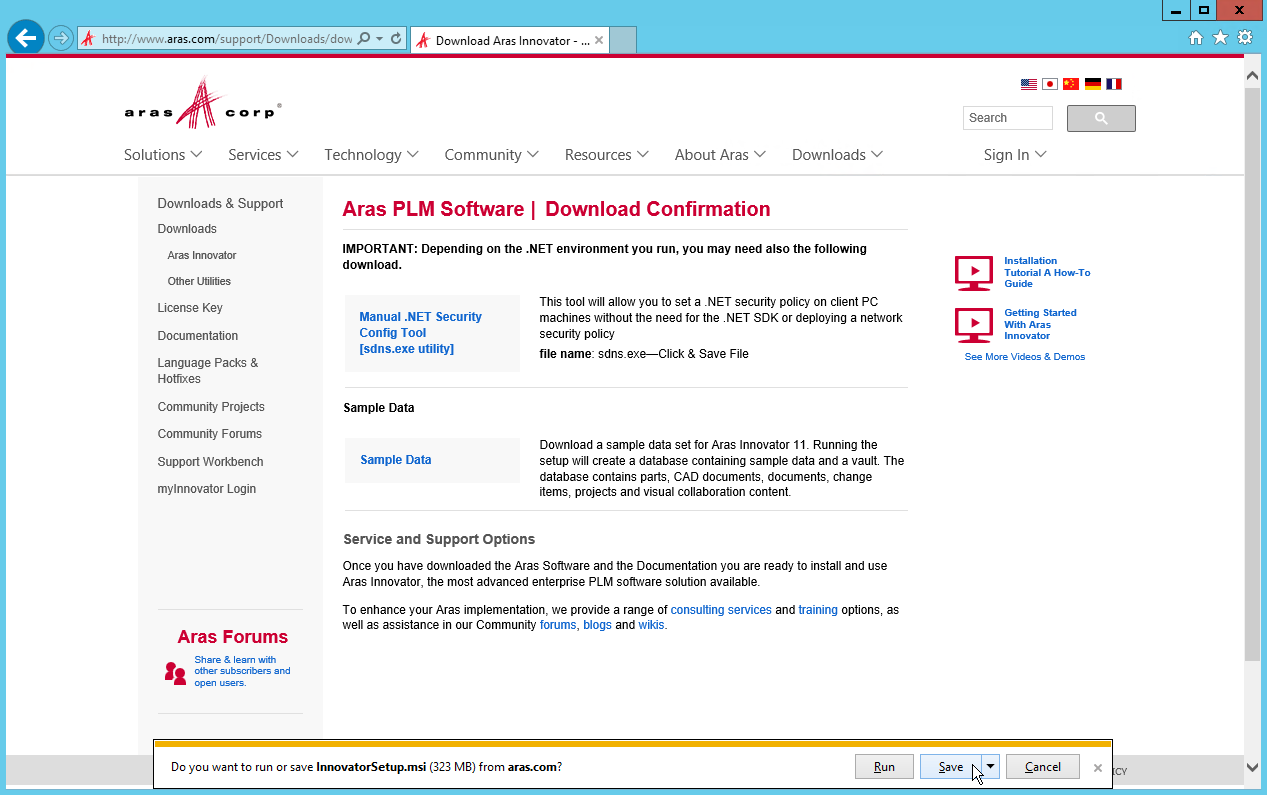
1. Load Balancer for 2 Innovator application nodes to provide high availability of innovator web application:
   1. Innovator application Web Node 1
   2. Innovator application Web Node 2
2. Dedicated SQL Server to provide high performance of SQL requests from innovator applications.
3. Dedicated server for each of the following components: Agent Service Server/ Vault Server/ Conversion Server/ OAuth Server.

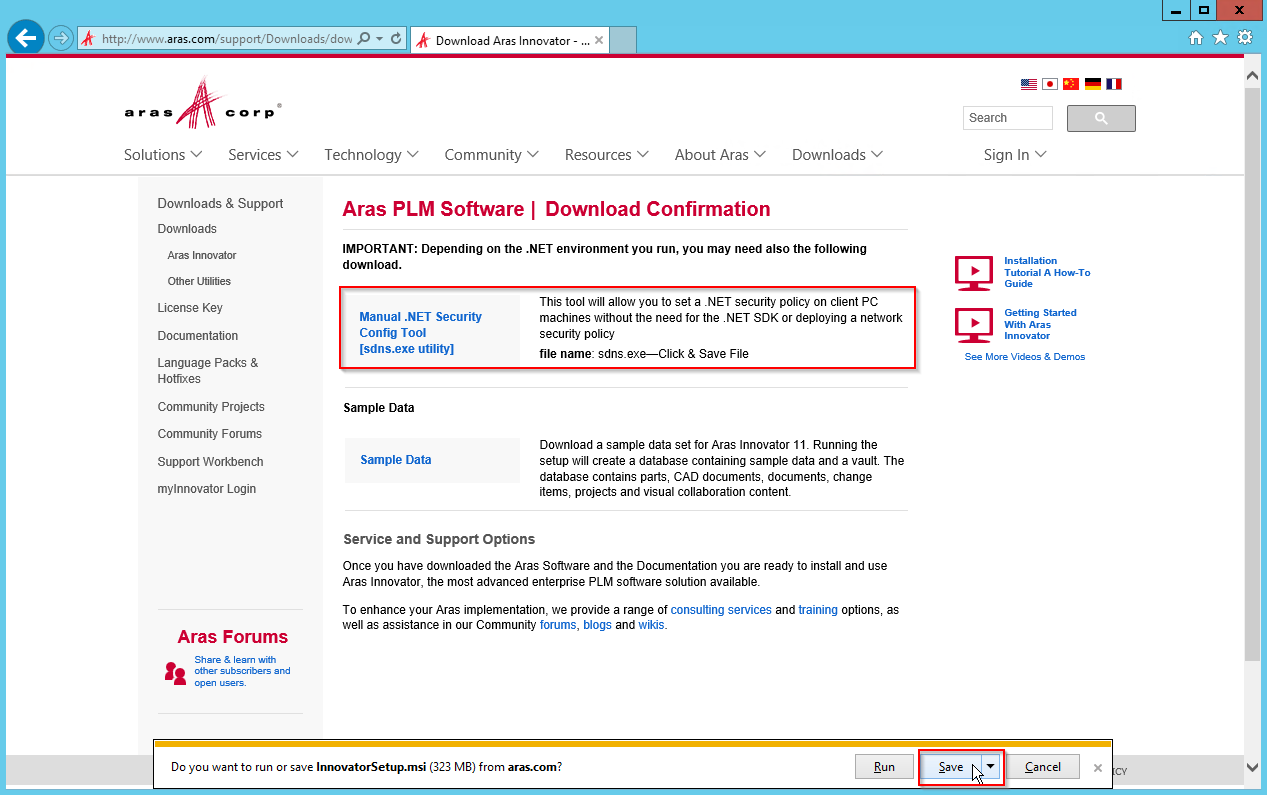
NOTE: Configuration of these nodes depends on expected load for its components. It is possible that some of these nodes are not required and could be removed from the configuration. It is also possible to put more than one component to a node, if they do not require much resources.

### UAT/Prod: Setup Innovator

Get msi from https://www.aras.com/support/downloads/



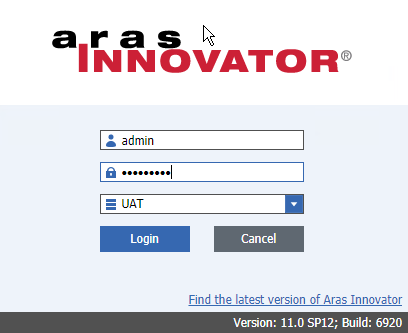




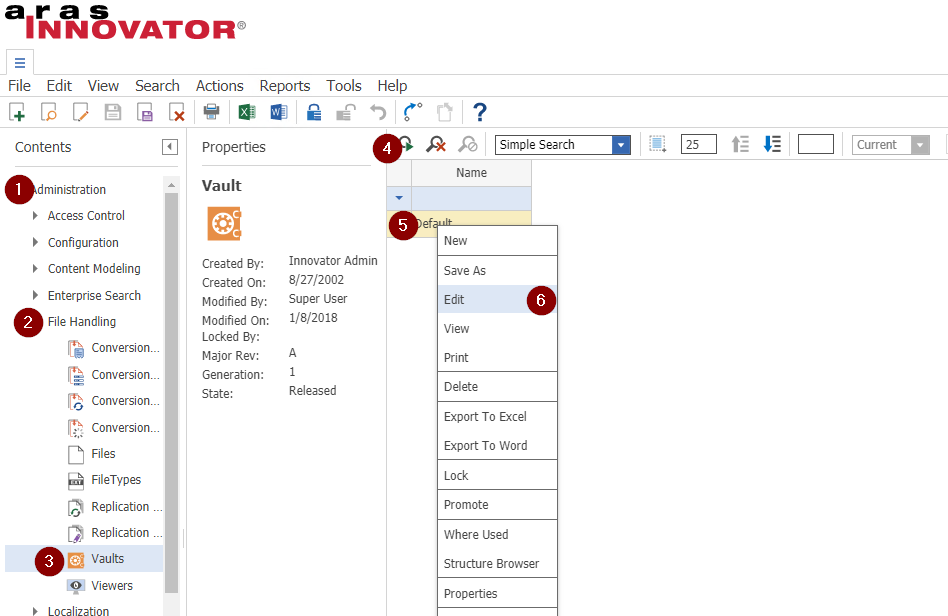
Copy msi file from build server to a target node

Follow instructions from the ‘Aras Innovator 11.0 - Installation Guide.pdf’ which could be found in CD image in the ‘Documentation’ folder.

### UAT/Prod: Configure Innovator: Vault Server

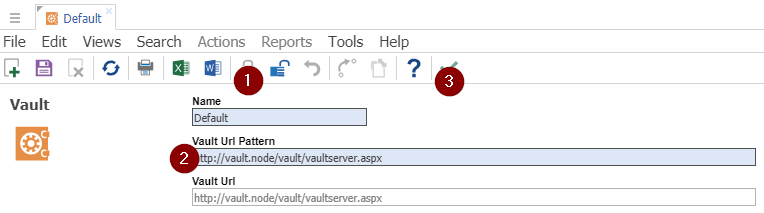
1. Log in to Innovator instance as an admin.
2. Edit Default vault

Under Administration->File Handling->Vaults. Run search and Edit the Default Vault.



In Vault Url Pattern specify URL the Vault server virtual path like so: <http://vault.node/vault/vaultserver.aspx>

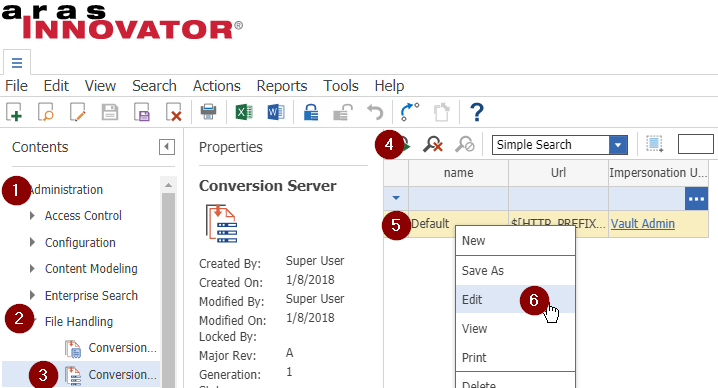
The pattern for URL is: **http://[vault server node name]/vault/vaultserver.aspx**



### UAT/Prod: Configure Innovator: Conversion Server

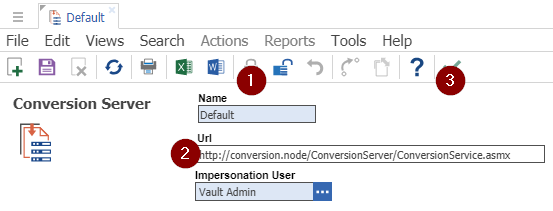
Under Administration->File Handling->Conversion Servers

Run search and Edit the Default Conversion Server.



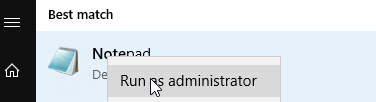
In Url field specify URL of Conversion Server: <http://conversion.node/ConversionServer/ConversionService.asmx>

The pattern for URL is: **https://[conversion server node name]/ConversionServer/ConversionService.asmx**



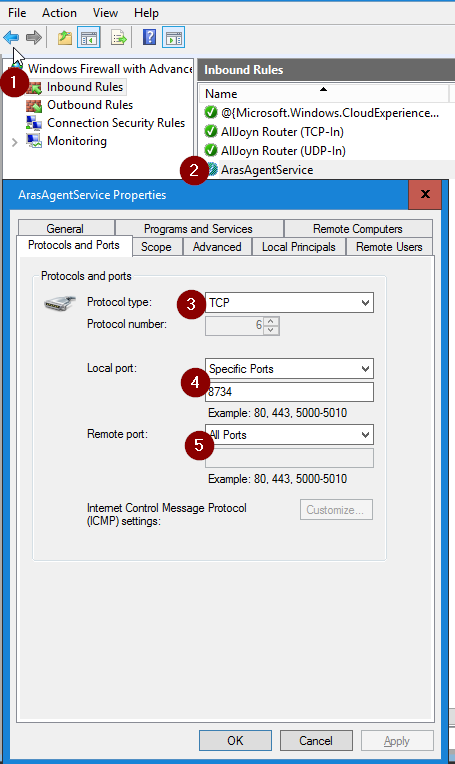
### UAT/Prod: Configure Innovator: Agent Service

For each Web Application Node: log in to a node and run notepad as admin:



Open InnovatorServerConfig.xml. Default path is: “C:\Program Files (x86)\Aras\Innovator\InnovatorServerConfig.xml”

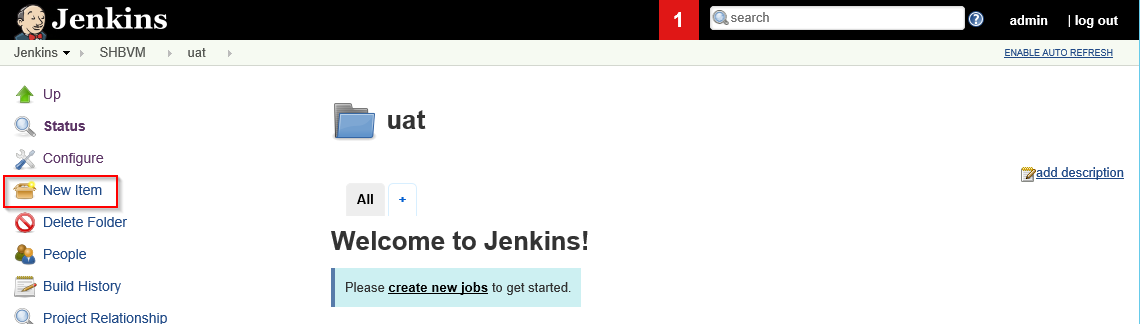
Modify <AgentService> tag – specify to InnovatorToServiceAddress attribute url address of installed Agent Service: [http://**agent.service.node**:8734/AgentService\_UAT](http://agent.service.node:8734/AgentService_UAT)

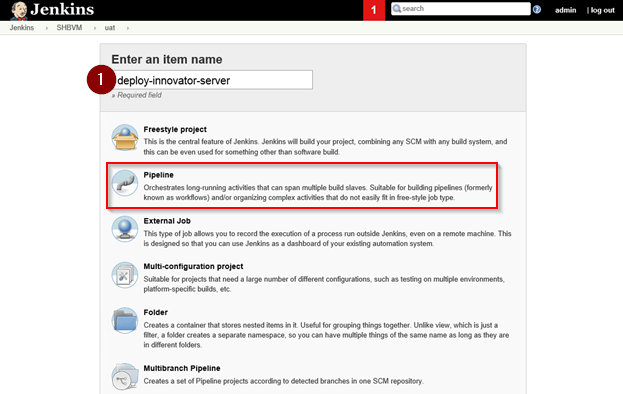
It is required to enable access to the agent service port (8734) in firewall at agent service node:

### UAT/Prod: Prepare Jenkins deployment job

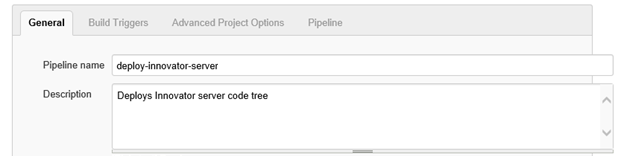
We need to create a Jenkins job for each Innovator component:

* Innovator server node 1
* Innovator server node 2
* Innovator database
* Conversion server
* Agent service
* Vault server node

1. Innovator server node 1

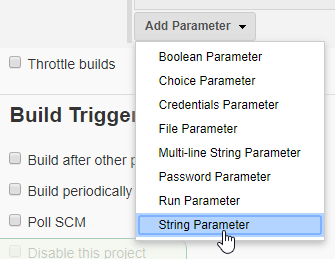
Create a new pipeline job. Enter appropriate name to field 1 and choose pipeline type (2).

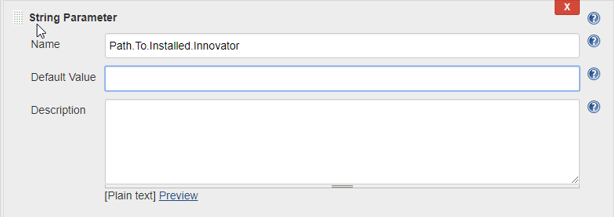
Set description if needed



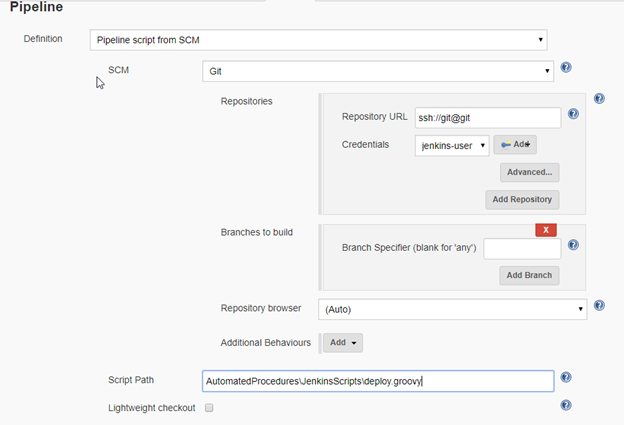
Add/Edit the following properties:

* Property name: **NodeLabel**; value: **consulting**
* Property name: **CustomBuildAgentNodeLabel**; value: **consulting**
* Property name: **TagName**; value: **none**.
* (For all jobs, but Innovator Database) Property name: CustomMssqlInnovatorPasswordCredentialId; value: MSSQL.Innovator.Password
* Property name: CustomBuildAgentNodeLabel; value: MSSQL.Innovator.Regular.Password
* Property name: **Path.To.Installed.Innovator**; value: “**\\web.node.1\C$\Program Files (x86)\Aras\Innovator”** (do not forget the quots)





To add a parameter press “Add Parameter” and select “String Parameter”. Then specify its name and value (in default value field)



In the pipeline section ensure that Repository url is correct and branch specifier is **uat**. Script Path depends from a component that the job should deploy.

In a case of use of custom MSSQL users (default users are: sa, innovator, innovator\_regular) , you should edit default setting include file in the repository .\AutomatedProcedures\Default.Settings.include and specify appropriate MSSQL login ids.

Make sure that URL of installed innovator points at load balancer url. Specify appropriate Agent service name (to get the name go to 'Computer management' -> 'Services' and find ‘Aras Innovator Agent’ service, open its properties and the name will be highlited) and its host server (agent.service.node).

To put modifications under source control - run git bash from the repository root and type the following commands:

git add AutomatedProcedures/Default.Settings.include

git commit -m "Setup UAT environment default values"

git push

1. Innovator server node 2

Copy job from Innovator server node 1 (deploy-innovator-server-1) to create for the second node:

Open prepared job for Innovator server node 1

Copy its full project name and close the job

Create a new job, but this time do not choose type but put the full project name to a “Copy from” field for a new job.

1. Conversion server

Copy job from Innovator server node 1 (deploy-innovator-server-1)

Keep properties unchanged

Specify different Script path under Pipeline section: AutomatedProcedures\JenkinsScripts\deploy.conversion.server.groovy



1. Agent service

Copy job from Innovator server node 1 (deploy-innovator-server-1)

Keep properties unchanged

Specify different Script path under Pipeline section: AutomatedProcedures\JenkinsScripts\deploy.agent.service.groovy

1. Vault server node

Copy job from Innovator server node 1 (deploy-innovator-server-1)

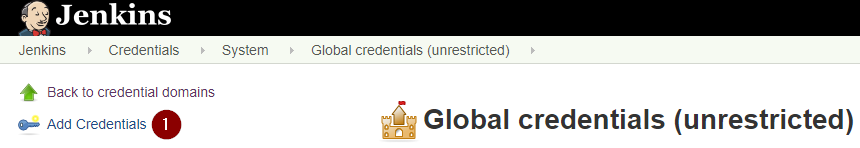
Change the Path.To.Installed.Innovator parameter to, for example: “\\vault.node\C$\Program Files (x86)\Aras\Innovator” (do not forget the quotes)

Specify different Script path under Pipeline section: AutomatedProcedures\JenkinsScripts\deploy.vault.server.groovy

1. Innovator database
   1. Setup credentials

First of all, we need to specify credentials for MSSQL user innovator and Innovator user ‘root’.

Navigate to <http://build.server:8080/credentials/store/system/domain/_/> (server name may differ)



Click ‘Add Credentials’ (1)

Fill in the following:

1. Kind: Secret text
2. Scope: Global
3. Secret: put a password of an innovator db owner (login innovator by default)
4. ID: MSSQL.Innovator.Password.**SQLSERVER** (just example, for different SQL instance it is recommended to specify its name, marked in **bold**)

For innovator root user:

1. Kind: Secret text
2. Scope: Global
3. Secret: put a password of the root user of an innovator server
4. ID: Innovator.Root.Password.**UAT** (just example, for different server SQL instance it is recommended to specify its name, marked in **bold**)

For MSSQL sa user:

1. Kind: Secret text
2. Scope: Global
3. Secret: put a password of a MSSQL server admin login (login sa or vmcr by default)
4. ID: MSSQL.SA.Password.**SQLSERVER** (just example, for different server SQL instance it is recommended to specify its name, marked in **bold**)

Create a jenkins job for innovator database:

Copy job from Innovator server node 1 (deploy-innovator-server-1)

Remove redundant parameters Perform.Import.Of.SampleData.

* 1. Create the following string parameters:
* MSSQL.Server: **SQLSERVER**
* MSSQL.Database.Name: **UAT**
* CustomMssqlInnovatorPasswordCredentialId: **MSSQL.Innovator.Password.SQLSERVER**
* CustomMssqlSaPasswordCredentialId: **MSSQL.SA.Password.SQLSERVER**
* Path.To.Installed.Innovator : \\innovator.node.1\C$\Program Files (x86)\Aras\Innovator
* CustomInnovatorRootPasswordCredentialId: **Innovator.Root.Password.UAT**

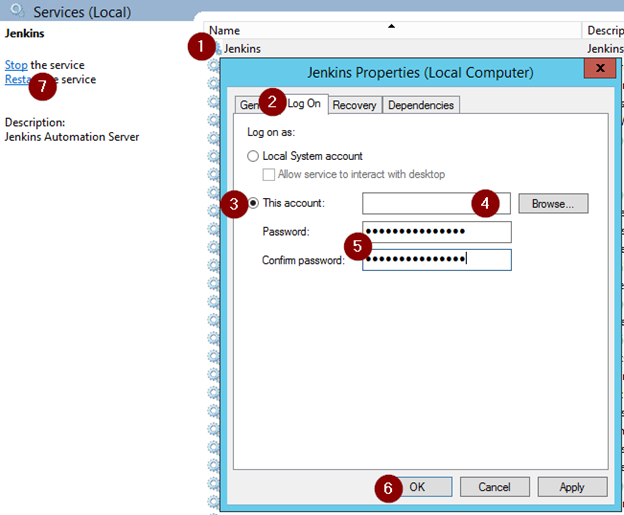
NOTE: as we do not specify MSSQL.Innovator.Password and Password.Of.Root.Innovator.User directly in plain text, we use parameters CustomMssqlInnovatorPasswordCredentialId, CustomInnovatorRootPasswordCredentialId that contain credentials that we created earlier for this invironment (UAT)

* 1. Specify different Script path under Pipeline section:

AutomatedProcedures\JenkinsScripts\deploy.innovator.db.groovy

* 1. Setup Jenkins to access network share

In order to have access Jenkins server needs to run under specific user. To set this user navigate to 'Computer management' -> 'Services'.



Find “Jenkins” service (1) open its properties. On Log On (2) tab choose This account (3) specify custom account name (4), fill in password fields (5), press OK (6) and restart the service (7).

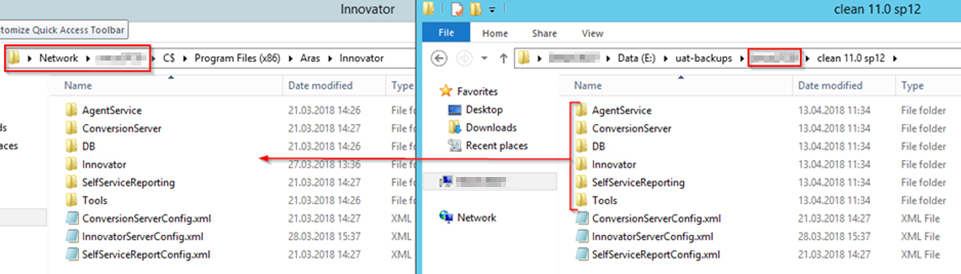
### UAT: Restore Innovator instance to a baseline state

The first step before deploy the latest modifications to UAT you need to restore existing instance to a ‘baseline’ state, which should be equivalent of current production instance. To reduce possibility of deployment failures on production environment, it is required to run deployment on the same instance as production.

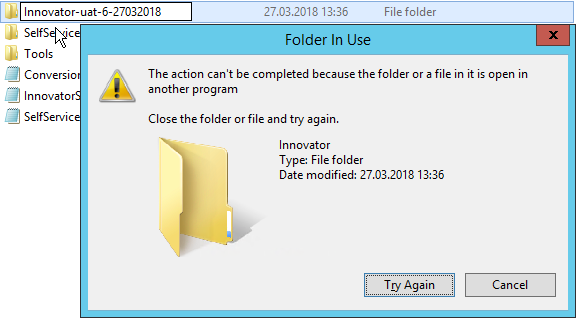
Components to be restored are:

1. CodeTree:
   1. Innovator server node 1
   2. Innovator server node 2
   3. Conversion server
   4. Agent service
   5. Vault server node 1
   6. Vault server node 2
2. Database
3. Restore code tree

To restore a code tree component - you need to do the following:

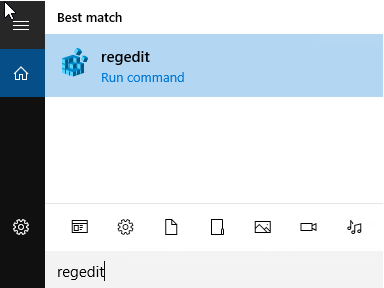
1. Create a backup of a component
2. Remove it
3. Copy component from a backup to a target node like so: 

NOTE: Innovator server nodes have registered fonts. When you try to remove/rename ‘Innovator’ folder you will see the error like shown below:



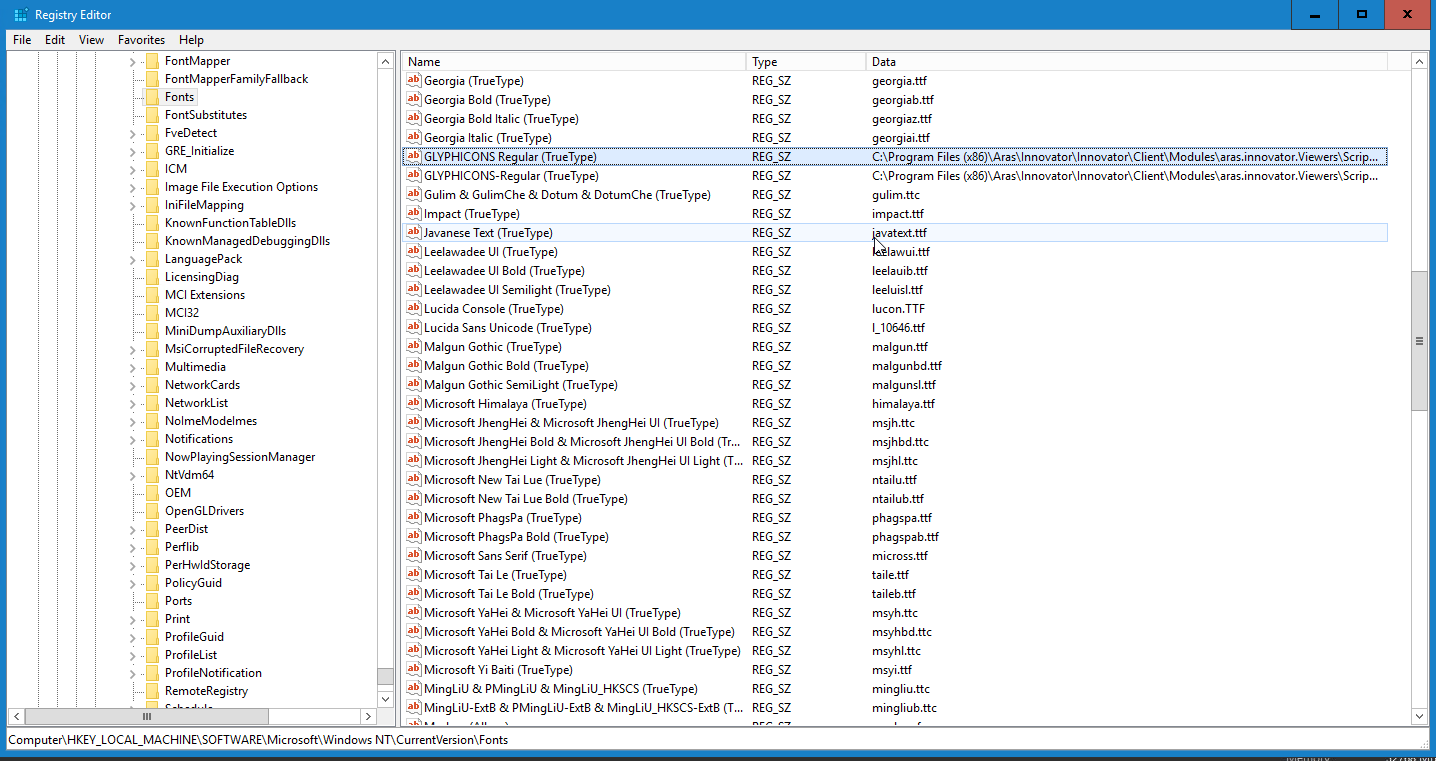
In order to fix it you need to remove registered fonts using regedit.

1. Open regedit (press Start, start to type regedit)



1. Locate and remove registered fonts

All fonts can be found in the registry key: HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts



Remove the following fonts from the key:

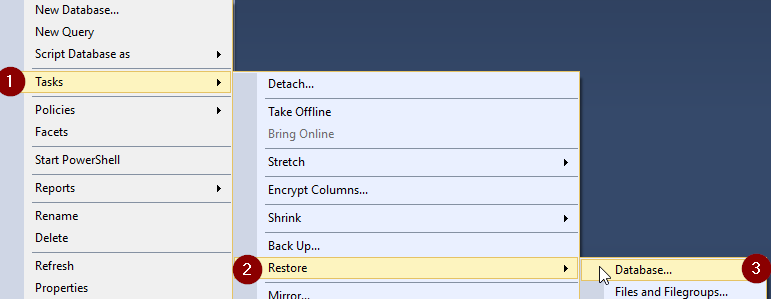
* GLYPHICONS Regular (TrueType)
* GLYPHICONS-Regular (TrueType)
* PDFTronGlyphicons (TrueType)
* tinymce (TrueType)
* tinymce-small (TrueType)

As we modified system registry we have to restart Innovator application node

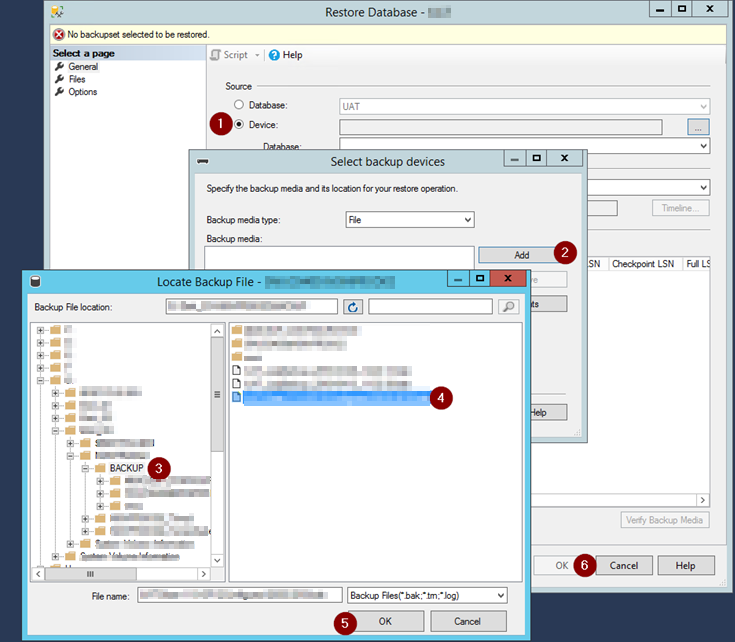
NOTE: fonts have to be removed from registry for both Innovator application nodes

1. Restore database

Open Management Studio and connect to SQLSERVER. Select uat database in the TOC and choose Tasks->Restore->Database in its context menu:



In opened dialog choose Source->Device (1) and press ‘…’ button to select a device. In ‘Setup backup devices’ dialog press ‘Add’ (2). In TOC (3) locate the latest production database backup (4) and press ‘OK’ for all opened dialogs.



## Deploy to production



### UAT/Prod: Run deployment jobs

Before you run you need to ensure that in Git all modifications are merged from uat branch to production branch.

In Jenkins go to a folder with configured Jenkins deployment jobs. The folder has 7 Jenkins deployment jobs for the following components: 2 Innovator applications, 2 Vault servers, Conversion Server, Agent Service, Database.

In every job you have to specify tag of the latest production commit. Tag template is *production-x*, where *x* is a release number.

First you need to apply jobs that upgrade code tree:

* deploy-agent-service
* deploy-conversion-server
* deploy-innovator-server-1
* deploy-innovator-server-2
* deploy-vault-server

After that you need to run the

* deploy-innovator-database

Database should be upgraded at the latest moment, as during code tree upgrade deployment procedure performs steps to prepare working Innovator instance, which is required for successful database upgrade.

### Creating baseline after deploy to production

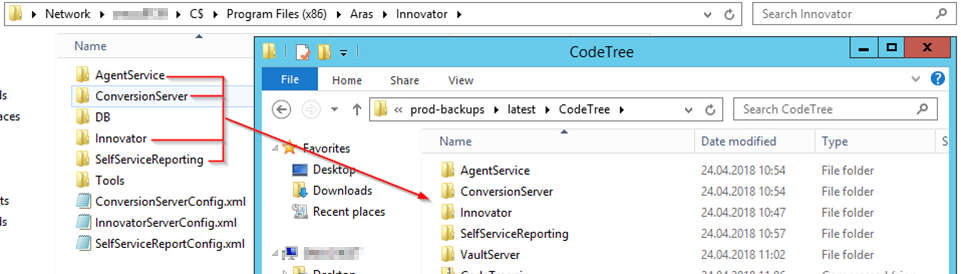
1. Backup old baseline

To backup previous baseline – simply rename the ‘latest’ folder located at build server at D:\prod-backups\latest to D:\prod-backups\[latest-prod-tag-name]

NOTE: [latest-prod-tag-name] is a name of a tag that of previous release. In case of first deployment it is CleanInnovator**XX**SP**YY**, where XX is a major version and YY is a service pack.

1. Put code tree from all nodes to a single temporary folder at build server

For example, create a new folder D:\prod-backups\latest\CodeTree at build server. Then copy code tree from production nodes (folders AgentService, ConversionServer, Innovator, SelfServiceReporting and VaultServer) to this temporary folder. Note that second application and vault server nodes can be skipped as they are the same as first ones.



After you have copied code tree to the build server you need to remove redundant configuration files from the temporary folder. It is required, because baseline will be used for several environments: CI, SIT, UAT and we need to configure them separately. The list of redundant configuration files is:

AgentService\Aras.Server.Agent.Service.exe.config

AgentService\OAuth.config

ConversionServer\ConversionServer.xml

Innovator\Client\Innovator.xml

Innovator\Client\OAuth.config

Innovator\Server\App\_Data\Certificates\InnovatorServer.pfx

Innovator\Server\App\_Data\Certificates\OAuthServer.cer

Innovator\Server\Innovator.xml

Innovator\Server\OAuth.config

OAuthServer\App\_Data\Certificates\AgentService.cer

OAuthServer\App\_Data\Certificates\InnovatorServer.cer

OAuthServer\App\_Data\Certificates\OAuthServer.pfx

OAuthServer\App\_Data\Certificates\SelfServiceReporting.cer

OAuthServer\App\_Data\Certificates\VaultServer.cer

OAuthServer\OAuth.config

OAuthServer\Web.config

SelfServiceReporting\App\_Data\Certificates\OAuthServer.cer

SelfServiceReporting\App\_Data\Certificates\SelfServiceReporting.pfx

SelfServiceReporting\SelfServiceReport.xml

SelfServiceReporting\OAuth.config

VaultServer\App\_Data\Certificates\OAuthServer.cer

VaultServer\App\_Data\Certificates\VaultServer.pfx

VaultServer\VaultServer.xml

VaultServer\OAuth.config

ConversionServerConfig.xml

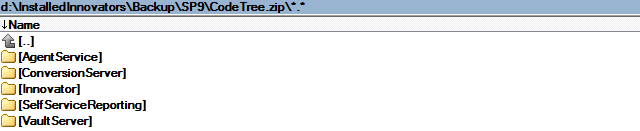
InnovatorServerConfig.xml

SelfServiceReportConfig.xml

VaultServerConfig.xml

Innovator\Server\Data\FeatureTree.xml

Innovator\Server\Data\FeatureTreeInfo.xml

Next you need to create an archive with name CodeTree.zip and with structure like this one:

… and put the archive to the E:\prod-backups\latest folder.

1. Create database backup

To create a database backup use the following command line:

sqlcmd -S (local)\MSSQL2014 -Q " BACKUP DATABASE [**ProductionDatabaseName**] TO DISK = '**\\prod-backups-server\prod-backups\last-release\latest\DB.bak'** WITH NOFORMAT, INIT, COMPRESSION".

The \\prod-backups-server\prod-backups\last-release\latest\DB.bak is a default value. In real environment SQL server most likely won’t have network access to a build server, so you need to specify a path on SQL server and then put the backup to the E:\prod-backups\latest folder at the build server.

# Development with CRT



## Development processes and standards



### Development process overview

1. Internal Development Process

* Aras internal development follows SAFe (Scaled Agile Framework) for many years
  + Includes organizations beyond Development and QA – e.g. Product Management; Documentation; etc.
* SAFe core values:
  + Alignment
  + Code Quality
    - Agile Architecture
    - Continuous Integration
    - Test-First
    - Refactoring
    - Pair Work
    - Collective Ownership
  + Transparency
  + Program Execution
* More information - http://www.scaledagileframework.com

1. Example of Branch & Commit Workflow



1. Main components

* Scalable development process must have:
  + Source control
  + Automated Build and Deployment procedure
  + Automated testing
* Must be setup at project inception
* Term “Continuous Integration” is often used as synonym which is not 100% correct
* “Integration” is the process
  + “Continuous” is just how often it runs

1. Why Use Source Control?

* Single version of truth
* Versioning (Properly)
  + Change Management
  + Sharing/Isolation
  + Restoring previous versions
* Understanding what happened
  + Visibility
  + History
  + Traceability/Auditability
* Collaboration
  + Coding Guidelines
  + Code Review/Approval
  + Secure access to code/Permissions
* Backup

1. Source Control Essentials

* GIT as default Source Control System
  + Both at Aras and Customers
* Frequent Commits
  + Work is not lost
  + More efficient integration with others
* Individual is responsible for merging own code
  + No such thing as dedicated Code Integrator
  + On regular basis!
    - Not at the end of the Sprint
* Atomic commits (logically grouped)
  + Easier to understand
  + Easy to “cherry pick”/revert

1. Automated Build/Deployment

* Based on information in Source Control (GIT)
* Repeatable procedure that is debugged and tested
* More efficient and error-proof
  + People make mistakes using manual methods
* No need to reinvent the wheel
  + Template already exists
  + Based on best practices
* Customizable per customer needs/processes

1. Automated Testing

* Developing comprehensive set of Automated tests takes time
  + Could be considered outside of initial “barebones” development process implementation
  + Can be done gradually in multiple stages
  + Preferred approach is to include Automation Tests implementation into project Sprints
* Possible types of Automation Tests:
  + Unit Tests (part of this training)
  + Integration Tests (part of this training)
  + UI Automation Tests
  + Load/Stress Tests
* Executed as part of Automated Build

1. Benefits

* Enables frequent releases and reduces cycle time for fixes and enhancements
* Greatly improves reliability of Sprint releases
  + Reduces wasted effort caused by bugs and regressions
* Facilitates distributed development teams
  + Helps get new people on the project up to speed
  + Applicable to any size team!
* Ability to reproduce Production system from Source Control
* Allows for seamless continuity of further custom development (save money on reverse engineering of production system)
* Scales up/down to any project size.
* Promotes and enforces standards

1. Sprints and planning

* You need to have a general plan/dashboard for all your projects
* Project plan
* Milestones/Phases
* Everything in place before you start
* Do not forget to include time for QA
* Developers must estimate

1. Sprints

* 1-2-3 weeks
* QA in parallel/final testing
* Daily status meetings
* Sprint planning
* +Task/-Task

1. Development planning

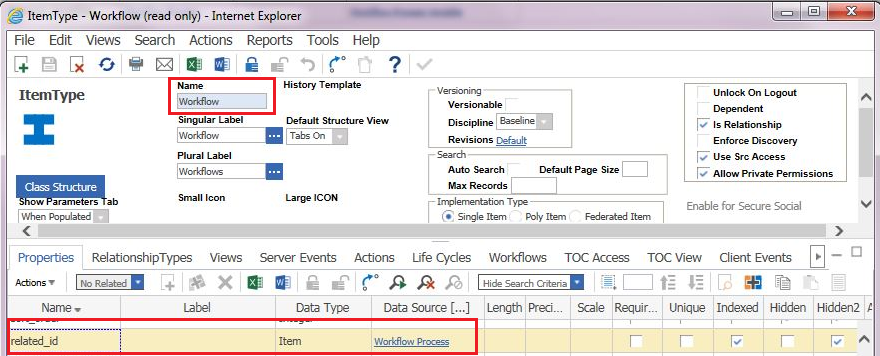
* Before you start you need to create an architecture of your project.
* Create tasks and estimations. Tasks should be no more than 8 hours.
* Plan your team time.
* Track your time.
* Do not forget to create tests.

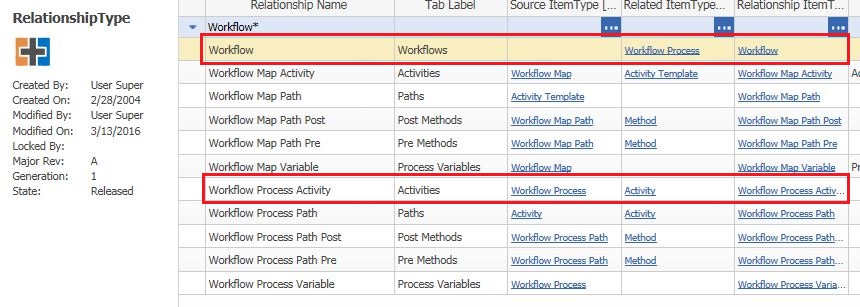
1. Documentation

* Specifications/user stories
* Always create documents for your development. How to setup/use it. Do not copy spec.
* Sprint planning document
* QA test plans/ outlines/ check lists
* Project plan

### Development standards

Innovator data model with relationships structure.





Make as less requests as possible.

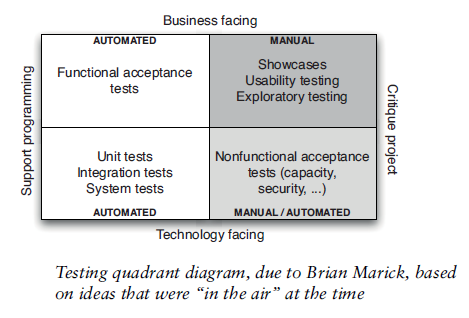
|  |  |
| --- | --- |
| Wrong way | Correct way |
| class BadLogic  {  string myFunc(string id)  {  Item temp = innovator  .getItemById("Type", id);  myFunc2(id);  ...  }  string myFunc2(string id)  {  // No need to get the same data from the  // server again  Item temp = innovator  .getItemById("Type", id);  ...  }  } | class GoodLogic  {  string myFunc(string id)  {  Item temp = innovator  .getItemById("Type", id);  myFunc2(temp);  ...  }  string myFunc2(Item temp)  {  //Item temp = innovator  // .getItemById("Type", id);  // use temp variable  ...  }  } |

Reducing number of requests due to present relationships structure.

|  |
| --- |
| Wrong way |
| '1st query  Dim workflowProcess As Item = Me.newItem("Workflow Process","get")  Dim workflowProcessActivity As Item = workflowProcess.createRelationship("Workflow Process Activity","get")  Dim activity As Item = workflowProcessActivity.createRelatedItem("Activity","get")  activity.setID('activityId')  workflowProcess = workflowProcess.apply()  Dim workflowProcessId As String = workflowProcess.getID()  '2nd query    Dim workflow As Item = Me.newItem("Workflow","get")  workflowProcess = workflow.createRelatedItem("Workflow Process","get")  workflowProcess.setID(workflowProcessId)  workflow = workflow.apply() |
|  |
| Correct way |
| Dim workflow As Item = Me.newItem("Workflow","get")  Dim workflowProcess As Item = workflow.createRelatedItem("Workflow Process","get")  Dim workflowProcessActivity As Item = workflowProcess.createRelationship("Workflow Process Activity","get")  Dim activity As Item = workflowProcessActivity.createRelatedItem("Activity", "get")  activity.setID('activityId')  workflow = workflow.apply() |

### Tests overview

Many kinds of testing exist. Brian Marick came up with table, which is widely used to model the various types of tests that you should have in place to ensure the delivery of a high-quality application.



1. Unit tests:

Unit tests are written to test the behavior of small pieces of your application in isolation (say, a method, or a function, or the interactions between a small group of them).

They can usually be run without starting the whole application. They do not hit the database (if your application has one), the filesystem, or the network.

They don’t require your application to be running in a production-like environment.

Unit tests should run very fast—your whole suite, even for a large application, should be able to run in under ten minutes.

1. Acceptance tests:

Acceptance tests test that the application meets the acceptance criteria decided by the business, including both the functionality provided by the application and its characteristics such as capacity, availability, security, and so on. Acceptance tests are best written in such a way that they run against the whole application in a production-like environment. Acceptance tests can take a long time to run—it’s not unheard of for an acceptance test suite to take more than a day to run sequentially.

1. Integration tests:

If your application is conversing with a variety of external systems through a series of different protocols, or if your application itself consists of a series of loosely coupled modules with complex interactions between them, then integration tests become very important. The line between integration testing and component testing is blurry (not least because integration testing is a somewhat overloaded term). We use the term *integration testing* to refer to tests which ensure that each independent part of your application works correctly with the services it depends on.

Integration tests can be written in the same way as you write normal acceptance tests. Normally, integration tests should run in two contexts: firstly, with the system under test running against the real external systems it depends on, or against their replicas controlled by the service provider, and secondly against a test harness which you create as part of your codebase.

1. Tests in a CRT-based repository:

All tests in repository stored in Tests folder. There are 3 solutions:

1. Tests\AllTests.sln – solution that combine both, unit and integration tests. Used mainly from automatic build procedures.
2. Tests\UnitTests.sln – lightweight solution, that contains tests project and CSharpMethods project.
3. Tests\IntegrationTests.sln – contains single project with integration tests.

There are also several bat files in the repository root to simplify developer’s process: RunUnitTests.bat, RunIntegrationTests.bat, RunAllTests.bat.

1. IntegrationTests framework

Aras provides Aras.IntegrationTests.Framework as part of sample repository. Typical integration test is an XML file of special structure.

<TestCase author="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../../libraries/Aras.IntegrationTests.Framework/TestCase.xsd">  
 <Description/>  
 <Request>  
 <Comments//>  
 <Query>  
 <AML/>  
 </Query>  
 <Baseline/>  
 </Request>  
 <TestCaseCleanUp>  
 <Query>  
 <AML/>  
 </Query>  
 </TestCaseCleanUp>  
</TestCase>

## Developer guides



### Setup machine and branch specific settings

When you are working with several branches and environments, and even repositories it important to have a possibility to store different sets of settings. For this reason, automated scripts use special ‘**\_machine\_specific\_includes**’ folder. By default, this folder located in C:\\_machine\_specific\_includes\. In order to customize this path, you can set system environment variable MACHINE\_SPECIFIC\_INCLUDES\_DIR with custom path.

If you did not create the folder it will be created automatically by automated scripts. For the first run of a script a new setting file created if that script is dependent from any parameter from machine and branch specific settings. The file’s name has the following template:

[project\_prefix]-[branch\_name].Settings.include

project\_prefix identified by the ‘*Project.Prefix*’ property specified in the *.\AutomatedProcedures\Default.Settings.include*.

branch\_name is equal to the name of a currently checked out branch.

When file created by the automated scripts the script gives you a message like the following:

File [path\_to\_machine\_and\_branch\_specific\_settings\_file] does not exists!

Copying default setting from the repository to create new Machine And Branch Specific Settings

Please modify the following values according to your environment and re-run the script:

Path.To.DB.Bak - absolute path to the DB.bak with production DB backup.

Path.To.CodeTree.Zip - absolute path to the CodeTree.zip archive with code tree of production Innovator instance. (Required for ContinuousIntegration task)

Innovator.License.Key - license key that should be obtained from [link\_to\_license\_key\_service]

Innovator.Activation.Key - activation key that should be obtained from [link\_to\_license\_key\_service]

MSSQL.Server - MS SQL server name. Correct value is required for cases when DB is restored from backup.

MSSQL.SA.Password - password for login 'sa' on a SQL Server

MSSQL.Innovator.Password - password for login 'innovator' on a SQL Server.

Path to the settings file is: [path\_to\_machine\_and\_branch\_specific\_settings\_file]

In order to be able to setup Innovator you have to specify named properties in the created machine and branch specific settings file, according to the message, provided above.

### Setup local Innovator instance from the repository

Requirements:

* IIS.
* MSSQL Server.
* Administrative privileges.
* [Machine and branch specific settings file](#_Setup_machine_and).

Deployment procedure consists of the following steps:

1. Prepare Innovator configuration files
2. Setup IIS application
3. Restore baseline database backup
4. Prepare deployment package based on a diff between current repository state and baseline commit state
5. Apply database upgrade

NOTE: Code Tree upgrade is skipped due to the fact that IIS setup on this repository which already has latest state of the Code Tree.

In order to deploy changes to a local instance of Innovator run .\SetupInnovatorHere.bat as administrator.

If you perform Innovator setup for the first time you have to setup Machine and branch specific includes first.

In order to remove an instance previously installed by SetupInnovatorHere.bat run as administrator the .\AutomatedProcedures\RemoveThisInnovator.bat

### Run Continuous Integration

Continuous integration is a required step that checks your local changes.

Requirements:

* IIS
* MSSQL Server
* Code Tree backup
* Database backup
* [Machine and branch specific settings file](#_Setup_machine_and).

Run the AutomatedProcedures\ContiniousIntegration.bat. This will create temporary innovator instance from release backups and restore database. The folder will be created in the root of a disk where your repository is placed. The the procedure will validate encoding of files, validate your code, build and run tests.

The same procedure is used in regular automated CI procedure in your CI manager (i.e. Jenkins). Use ContiniousIntegration.bat to reduce possibility that your modifications will cause an error during official build.

### Converting Innovator server methods

There is a tool called MethodConverter, located in Tests\PackageMethods\MethodConverter\PackageMethods.MethodConverter.exe, which purpose is to convert AML representation of Innovator server methods (stored as xml files) to .cs files, attached to a unit tests project, and vice versa.

There are 2 batch files for conversion of Innovator server methods available in folder AutomatedProcedures:

**ConvertPackageMethodsFromXmlToCs.bat** used in case when you added server methods to the repository. Steps to use:

1. Export package, that contains server methods.
2. Merge exported package with packages contained in the AML-packages folder.
3. Run ConvertPackageMethodsFromXmlToCs.bat
4. Run CI in order to check modifications
5. Commit and push modifications from AML-packages and Tests modification.

**ConvertPackageMethodsFromCsToXml.bat** for cases when there are modifications in server methods in unit tests, that have to be applied to AML representation (most likely during refactoring). Steps to use:

1. Modify C# method in a unit tests project.
2. Run ConvertPackageMethodsFromCsToXml.bat
3. Run CI in order to check modifications
4. Commit and push modifications from AML-packages and/or Tests modification.

NOTE: From xml to C# conversion is called automatically by RunUnitTests.bat

### Run Tests

Requirements:

* MSBuild Tools (for C# unit and integration tests)
* NPM (for client tests)
* Installed innovator and [machine and branch specific settings file](#_Setup_machine_and) (for integration tests)

There is a bunch of automated batch scripts in the repository root with ‘Run’ and ‘Tests’ in their names. These scripts are intended to run corresponding tests (i.e. unit, integration, client, all). In order to run tests simply run according batch file.

The scripts are helpful for cases when you want to check that tests are passing, without running whole [CI procedure](#_Run_Continuous_Integration).

### Add or edit an item/method

This section describes how to put or edit an item using CRT.

Steps:

1. [Setup local Innovator instance from the repository](#_Setup_local_Innovator).
2. If you…
   1. …add new item to the Repository:
      1. Create new item/method using the instance.
      2. Add the item/method to an existing or new package.
   2. …edit an item that is already exists in the Repository
      1. Find the item/method and modify it using the instance.
      2. Find package that contains the item/method.
3. Export the package using Export Tool to a temporary directory. Path to Export Tool: .\AutomatedProcedures\tools\PackageImportExportUtilities\Export\export.exe
4. Merge package changes to the Repository to the AML-packages folder.
5. Commit and push the changes to the repository.
6. Run [Continuous Integration](#_Continuous_Integration_or) in order to check your modifications.

### Remove an item/method that exists in a baseline

This section describes how to correctly remove metadata from production instance using this Repository.

Steps:

1. [Setup local Innovator instance from the repository](#_Setup_local_Innovator).
2. Create new AML script in the AmlDeploymentScripts\1-BeforeAmlPackagesImport.
3. Remove item/method using AML like this:

<AML>

<Item type=”some\_type” id=”some\_id” action=”delete” />

</AML

1. Remove an item/method using the AML script applied via nash (default url is http://localhost/*InnovatorServerAlias*/Client/scripts/nash.aspx).
2. Find package that contained the item/method.
3. Export the package using Export Tool to a temporary directory (Export Tool is available public tool)
4. Merge package changes to the Repository to the AML-packages folder (Note that you will have to manually remove item/method from the repository. You will need to find out which file is absent in the package, that you obtained during step 4. The same file has to be removed from the repository. You can use a diff tool in order to find removed item/method in the repository).
5. Commit and push the changes to the repository.
6. [Run Continuous Integration](#_Continuous_Integration_or) in order to check your modifications.

### Update imports.mf

To upgrade Innovator instance - you need to import core and solutions packages modified in a period of time between your current version and target version. To do so you need to:

1. Get list of modified packages:

Considering you have 2 points in your repository: base commit for clean innovator 11.0 SP9 – CleanInnovator11SP9 and latest sync commit CleanInnovator11SP12. In this case you need to understand what packages was modified between those commits. To do this you need to run this command:

git diff --name-only CleanInnovator11SP12 CleanInnovator11SP9 -- AML-packages/ | sed -r 's/\/[^\/]\*\/[^\/]\*$/ /' | uniq

The command gets names of all modified core and solution packages. Output example for diff between 11.0 SP12 and 11.0 SP5 (not SP9 because there is no diff in solution packages):

AML-packages/DatabaseIdentifier/Import

AML-packages/PLM/Import

AML-packages/Project/Import

AML-packages/com/aras/defaults/filerepresentations

AML-packages/com/aras/innovator/ACExtentions

AML-packages/com/aras/innovator/QueryBuilder

….

// Omitted due to similarity of results

….

AML-packages/com/aras/innovator/tdf

AML-packages/com/aras/innovator/tgv/configuration

AML-packages/com/aras/innovator/tgv/viewer

AML-packages/com/aras/innovator/viewers

AML-packages/com/aras/innovator/wm

AML-packages/core\_imports.mf

AML-packages/samples

AML-packages/solutions\_imports.mf

1. Put updated packages to imports.mf:

imports.mf file located in the .\AML-packages folder. The file has the following structure:

<imports>

<package name="MyPackage1" path="MyPackage1\Import" />

<package name="MyPackage2" path="MyPackage2\Import" >

<dependson name="MyPackage1" />

</package>

<package name="MyPackage3" path="MyPackage3\Import" />

<package name="MyPackage4" path="MyPackage4\Import" >

<dependson name="MyPackage2" />

<dependson name="MyPackage3" />

</package>

</imports>

To put modified packages you need to add them to your imports.mf file from template import files core\_imports.mf and solutions\_imports.mf like shown below (added lines marked in green):

<imports>

<package name="MyPackage1" path="MyPackage1\Import" />

<package name="MyPackage2" path="MyPackage2\Import" >

<dependson name="MyPackage1" />

</package>

<package name="MyPackage3" path="MyPackage3\Import" />

<package name="MyPackage4" path="MyPackage4\Import" >

<dependson name="MyPackage2" />

<dependson name="MyPackage3" />

</package>

<package name="com.aras.innovator.wm" path=".\" />

<package name="com.aras.innovator.admin" path=".\" />

<package name="com.aras.innovator.core" path=".\" >

<dependson name="com.aras.innovator.admin" />

</package>

<package name="com.aras.innovator.QueryBuilder" path=".\" />

<package name="com.aras.innovator.extendedclassification" path=".\">

<dependson name="com.aras.innovator.core" />

</package>

…

<!--Omitted due to similarity-->

…

<package name="com.aras.innovator.solution.PLM" path="PLM\Import" />

<package name="com.aras.innovator.solution.Project" path="Project\Import">

<dependson name="com.aras.innovator.solution.PLM" />

</package>

</imports>

NOTE: make sure that you take only those packages that was modified. We got a list of modifications during step 1.

### Setup solution: generic steps

The following steps describe how do you usually start development of a solution in the Customer Repository Template. Each solution can have some specific action. Each solution can have individual structure. The steps below have detailed description for each solution.

1. Get solution package, unzip its content to a temporary folder (Temp Folder).
2. Locate the following package components:
   1. Folder with Client modifications (Package Client Folder);
   2. Folder with pre/post AML fixes (Package Fixes Folder);
   3. Folder with AML-package imports (Package Imports Folder).
3. Put listed above package components to destination folders in the repository:
   1. AmlDeploymentScripts;
   2. AML-packages;
   3. Innovator\Client.
4. Put solution under source control.

### Setup solution: Product Engineering

**Prerequisites:**

* Ensure that a solution is supported by version of Innovator stored in CRT at <http://www.aras.com/subscriberportal/support.aspx>.

**Steps**:

1. Get Product Engineering package from Aras ftp by this relative path: Builds\PE\Aras.PE-Build-1008-Tag-RELS11-0-R3-PE.zip. Copy the package archive and unzip it to the Temp Folder.
2. Put Solution components under source control.

Product Engineering consists of the following components:

* Client Folder
* Pre-Import Fixes Folder
* Application Core Imports
* PLM Imports
* imports.mf file

Next table describes where specified components stored and where they should be stored in the repository:

|  |  |  |
| --- | --- | --- |
| Component name | Source relative path | Target relative path |
| Client Folder | .\Files\Innovator\Client\ | .\Innovator\Client\ |
| Pre-Import Fixes Folder | .\AML\_Pre-Import\Fixes\ | .\AmlDeploymentScripts\1-BeforeAmlPackagesImport\ |
| Application Core Imports | .\AML\_Imports\ApplicationCore\ | .\AML-packages\ApplicationCore\ |
| PLM Imports | .\AML\_Imports\Import\ | .\AML-packages\PLM\Import\ |
| Imports.mf file | .\AML\_Imports\imports.mf | .\AML-packages\imports.mf |

Note: source path is relative to the Temp Folder; target path is relative to the Repository Root.

1. Copy Client Folder, Pre-Import Fixes Folder, Application Core Imports, and PLM Imports from the Temp Folder to the Repository root using relative paths from the table above.
2. Do not copy imports.mf file itself as well as its content. PE has malformed structure, which is not compatible with CRT Automated Scripts. Instead merge the following package definition to imports.mf file at the Repository (relative path specified above):

<package name="com.aras.innovator.solution.ApplicationCore" path="ApplicationCore\Import" />

<package name="com.aras.innovator.solution.PLM" path="PLM\Import">

<dependson name="com.aras.innovator.solution.ApplicationCore" />

</package>

1. Run the following git commands to add modifications:

git add Innovator/\*

git add AML-packages/\*

git add AmlDeploymentScripts/\*

NOTE: the solution was developed for Innovator 11.0.SP8. Since Innovator 11.0.SP9 Client Folder from PE is in Innovator code tree, so there is no need to copy and add Innovator\Client folder.

1. Create commit with comment like the next one:

*Put solution PE 11.0.R3 (build 1008)*

*- Put Innovator Client files.*

*- Put Application Core imports.*

*- Put PLM imports.*

*- Ensure that imports.mf has package definition.*

*Source:*

*Aras ftp mirror: Builds\PE\Aras.PE-Build-1008-Tag-RELS11-0-R3-PE.zip*

1. Run ContinuousIntegration.bat as administrator in order to ensure that validation, deploy, build and tests passed.
2. Run SetupInnovatorHere.bat as administrator and ensure that solution imported successfully and works.

More information regarding run of automated scripts can be found in [Developer Tools Overview](../For%20Developers/Developer%20Tools%20Overview.docx#SetupInnovatorHere)

1. If there were no errors and everything works as expected push the commit to a central repository using **git push** command.

### Setup solution: Program Management

**Prerequisites**

* Ensure that a solution is supported by version of Innovator stored in CRT at <http://www.aras.com/subscriberportal/support.aspx>.

**Steps:**

1. Get Program Managementpackage from Aras ftp by this relative path: Builds\PM\Aras.PM-Build-1024-Tag-RELS11-0-R2-PM.zip. Copy the package archive and unzip it to the Temp Folder.
2. Put Solution components under source control

Product Engineering consists of the following components:

* Client Folder
* PM Imports
* imports.mf file

Next table describes where specified components stored and where they should be stored in the repository:

| Component name | Source relative path | Target relative path |
| --- | --- | --- |
| Client Folder | .\Files\Innovator\Client\ | .\Innovator\Client\ |
| PM Imports | .\AML\_Imports\Import\ | .\AML-packages\Project\Import\ |
| Imports.mf file | .\AML\_Imports\imports.mf | .\AML-packages\imports.mf |

Note: source path is relative to the Temp Folder; target path is relative to the Repository Root.

1. Copy Client Folder and PM Imports from the Temp Folder to the Repository root using relative paths from the table above.
2. Do not copy imports.mf file itself as well as its content. PM has malformed structure, which is not compatible with CRT Automated Scripts. Instead merge the following package definition to imports.mf file at the Repository (relative path specified above):

<package name="com.aras.innovator.solution.Project" path="Project\Import"/>

Please NOTE that the PM solution is dependent from the following packages:

* com.aras.innovator.solution.PLM

The reason why specified packages are not described in the package definition is the fact, that they are not modified for this solution and already exists in an Innovator database.

1. Run the following git commands to add modifications:

git add Innovator/\*

git add AML-packages/\*

1. Create commit with comment like the next one:

*Put solution PM 11.0.R2 (build 1024)*

*- Put Innovator Client files.*

*- Put PM imports.*

*- Ensure that imports.mf has package definition.*

*Source:*

*Aras ftp mirror: Builds\PM\Aras.PM-Build-1024-Tag-RELS11-0-R2-PM.zip*

1. Run ContinuousIntegration.bat as administrator in order to ensure that validation, deploy, build and tests passed.
2. Run SetupInnovatorHere.bat as administrator and ensure that solution imported successfully and works.

More information regarding run of automated scripts can be found in [Developer Tools Overview](../For%20Developers/Developer%20Tools%20Overview.docx#SetupInnovatorHere)

1. If there were no errors and everything works as suspected push the commit to a central repository using **git push** command.

### Setup solution: Component Engineering

**Prerequisites:**

* Ensure that a solution is supported by version of Innovator stored in CRT at <http://www.aras.com/subscriberportal/support.aspx>.
* Component Engineering license

**Steps:**

1. Get Product Engineeringpackage from Aras ftp by one of these relative paths: **11.0.R3**: Builds\CE\CE 11.0R3 1293 for AU1.1.zip; **11.0.R4**: Builds\CE\CE-RELS11-0-R4-11.0R4-1312.zip; **11.0.R5**: Builds\CE\CE-RELS11-0-R5-11.0R5-1321.zip. Copy the package archive and unzip it to the Temp Folder.

NOTE: this document describes the latest version - **11.0.R5**. Older versions have the same structure and there are no differences in further steps, except commit message.

1. Put Solution components under source control

Product Engineering consists of the following components:

* Client Folder
* Server Folder
* CE\_core Imports
* CE Imports
* CE\_settings Imports
* CE imports.mf file
* CE\_settings imports.mf file

Next table describes where specified components stored and where they should be stored in the repository:

|  |  |  |
| --- | --- | --- |
| Component name | Source relative path | Target relative path |
| Client Folder | .\Files\Innovator\Client\ | .\Innovator\Client\ |
| Server Folder | .\Files\Innovator\Server\ | .\Innovator\Server\ |
| CE\_Core Imports | .\Imports\CE\com\aras\innovator\CE\_core\ | .\AML-packages\com\aras\innovator\CE\_core\ |
| CE Imports | .\Imports\CE\CE\Import\ | .\AML-packages\CE\Import |
| CE\_settings Imports | .\Imports\CE-Settings\CE\_settings\Import\ | .\AML-packages\CE\_settings\Import\ |
| CE Imports.mf file | .\Imports\CE\imports.mf | .\AML-packages\imports.mf |
| CE\_settings Imports.mf file | .\Imports\CE-Settings\imports.mf | .\AML-packages\imports.mf |

Note: source path is relative to the Temp Folder; target path is relative to the Repository Root.

1. Copy Client Folder, Server Folder, CE\_core Imports, CE Imports, and CE\_settings Imports from the Temp Folder to the Repository root using relative paths from the table above.
2. You can either copy contents of CE imports.mf file and CE\_settings imports.mf file to the imports.mf file in the Repository (relative path specified above); or add the following package definition to it:

<package name="com.aras.innovator.CE\_core" path=".\">

<dependson name="com.aras.innovator.solution.PLM" />

</package>

<package name="com.aras.innovator.solution.CE" path="CE\Import">

<dependson name="com.aras.innovator.solution.PLM" />

<dependson name="com.aras.innovator.CE\_core" />

</package>

<package name="com.aras.innovator.solution.CE\_settings" path="CE\_settings\Import">

<dependson name="com.aras.innovator.CE\_core" />

<dependson name="com.aras.innovator.solution.CE" />

</package>

1. Run the following git commands to add modifications:

git add Innovator/\*

git add AML-packages/\*

1. Create commit with comment like the next one:

*Put solution CE 11.0.R5 (build 1321)*

*- Put Innovator Client files.*

*- Put Innovator Server files.*

*- Put CE\_Core imports.*

*- Put CE imports.*

*- Put CE\_settings imports.*

*- Ensure that imports.mf has package definition.*

*Source:*

*Aras ftp mirror: 11.0.R5: Builds\CE\CE-RELS11-0-R5-11.0R5-1321.zip*

1. Run ContinuousIntegration.bat as administrator in order to ensure that validation, deploy, build and tests passed.
2. Run SetupInnovatorHere.bat as administrator and ensure that solution imported successfully and works.

More information regarding run of automated scripts can be found in [Developer Tools Overview](../For%20Developers/Developer%20Tools%20Overview.docx#SetupInnovatorHere)

1. If there were no errors and everything works as suspected push the commit to a central repository using **git push** command.

### Setup solution: Technical Documentation

**Prerequisites**

* Ensure that a solution is supported by version of Innovator stored in CRT at <http://www.aras.com/subscriberportal/support.aspx>.

**Steps:**

1. Get Technical Documentationpackage from Aras ftp by this relative path: Builds\TechDoc\Aras.TechDoc-Build-1008-Tag-RELS11-0-R1.zip. Copy the package archive and unzip it to the Temp Folder.
2. Put Solution components under source control

Product Engineering consists of the following components:

* TechDoc Imports
* imports.mf file

Next table describes where specified components stored and where they should be stored in the repository:

|  |  |  |
| --- | --- | --- |
| Component name | Source relative path | Target relative path |
| TechDoc Imports | .\imports\TechDoc\ | .\AML-packages\TechDoc\Import\ |
| Imports.mf file | .\imports\imports.mf | .\AML-packages\imports.mf |

Note: source path is relative to the Temp Folder; target path is relative to the Repository Root.

1. Copy TechDoc Imports from the Temp Folder to the Repository root using relative paths from the table above.
2. Do not copy imports.mf file itself as well as its content. TechDoc has malformed structure, which is not compatible with CRT Automated Scripts. Instead merge the following package definition to imports.mf file at the Repository (relative path specified above):

<package name="com.aras.innovator.solution.TechDoc" path="TechDoc\Import"/>

Please NOTE that the TechDoc solution is dependent from the following packages:

* com.aras.innovator.solution.PLM
* com.aras.innovator.tdf

The reason why specified packages are not described in the package definition is the fact, that they are not modified for this solutions and already exists in an Innovator database.

1. Run the following git commands to add modifications:

git add AML-packages/\*

1. Create commit with comment like the next one:

*Put solution TechDoc 11.0.R1 (build 1008)*

*- Put TechDoc imports.*

*- Ensure that imports.mf has package definition.*

*Source:*

*Aras ftp mirror: Builds\TechDoc\Aras.TechDoc-Build-1008-Tag-RELS11-0-R1.zip*

1. Run ContinuousIntegration.bat as administrator in order to ensure that validation, deploy, build and tests passed.
2. Run SetupInnovatorHere.bat as administrator and ensure that solution imported successfully and works.

More information regarding run of automated scripts can be found in [Developer Tools Overview](../For%20Developers/Developer%20Tools%20Overview.docx#SetupInnovatorHere)

1. If there were no errors and everything works as suspected push the commit to a central repository using **git push** command.

### Synchronize source control with production instance

There are cases when production synchronization is required. For example, customer with existing production instance starts development using CRT based repository or production had issue that was fixed on the production itself. In both cases we have to level the gap between existing production instance and production branch in the repository. To do this follow these steps:

1. **Sync automated procedures** from production instance
2. **Sync Code Tree** from production instance
3. **Sync Metadata AML-packages** from production instance
4. **Update baseline Tag**
5. Sync automated procedures

First you to get latest modifications from CRT regarding automated scripts, AML-packages and code tree.

To synchronize your repository with CRT - go through the following procedure:

Add remote to CRT and checkout to a current production branch:

git remote add crt ssh://git@git.aras.com/devops/customerrepositorytemplate.git

git fetch crt

git checkout production

1. Make sure that your repository root structure looks like this example:  
   

When a CRT based repository created it has 3 commits: empty initial commit, Clean contains of targeted CRT branch (RELS11-0-SP9), updated project prefix. To correctly update automated procedure, you need to run the following command before get updates from CRT:

git checkout -b sync\_auto\_procedures CleanInnovator11SP10~1

This command will create branch from a commit before CleanInnovator11SP9 tag. It is done to not overwrite project prefix with default value, which should be the first customization for any CRT based repository.

1. Synchronize sync\_auto\_procedures branch with crt/RELS11-0-SP10:
   1. Remove all files from repository except the .git folder.
   2. Fetch all files from crt and create a commit:

git checkout crt/RELS11-0-SP10 -- .

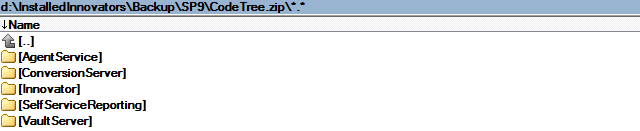
git add .

git commit -m “upgrade automated procedures from crt”

1. Merge sync\_auto\_procedures into your production:

git checkout production  
git merge sync\_auto\_procedures

1. Sync Code Tree

To prepare code tree navigate to the physical location of a production instance of the Innovator (note that it could be located at several nodes) and put code tree files to a temporary folder or archive like this: 

You need to copy the following Innovator components code tree:

* Agent Service
* Conversion Server
* Innovator
* Self Service Reporting
* Vault Server
* (starting from 11.0 SP12) OAuth Server

Once you have copied all code tree components from production, you need to remove configuration files from it. List of configuration files to remove is:

* AgentService\Aras.Server.Agent.Service.exe.config
* ConversionServer\ConversionServer.xml
* Innovator\Client\Innovator.xml
* Innovator\Server\Innovator.xml
* SelfServiceReporting\SelfServiceReport.xml
* VaultServer\VaultServer.xml
* ConversionServerConfig.xml
* InnovatorServerConfig.xml
* SelfServiceReportConfig.xml
* VaultServerConfig.xml

To complete sync in production branch you need to remove code tree from the repository and put the prepared copy of a production code without config files. After it is done commit recent modification to a Git:

git add .

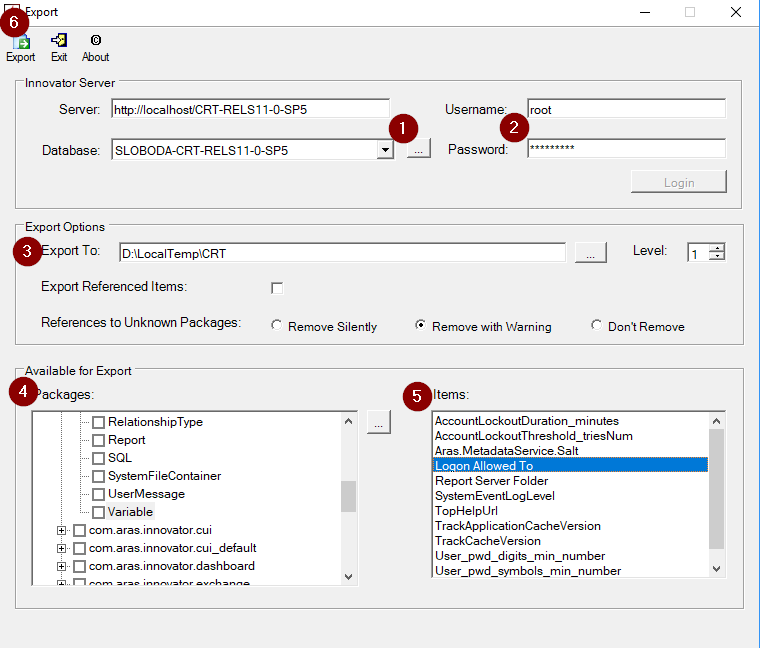
git commit -m “Synchronize code tree with production”

1. Sync Metadata AML-packages

The easiest way to do this is to use Export tool located in the repository at AutomatedProcedures\tools\PackageImportExportUtilities\Export\export.exe

Run the tool (See image below). Specify the following:

1. Production Innovator Server URL and database
2. Login information: user name and password
3. Directory for export packages to. It is good idea to export packages to a new clean temporary directory in case of errors.
4. Specify packages to export, that contain metadata only, do not export actual data
5. Specify items to be exported.



When you filled mentioned information – run export by pressing “Export” button (6)

After export completed you need to remove AML-packages folder from the repository and put just exported packages to the production branch. After it is done commit recent modification to a Git:

git add .

git commit -m “Synchronize AML-packages with production”

1. Update baseline Tag

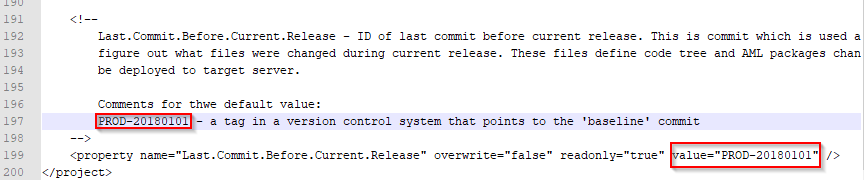
Your repository is almost synchronized. The last but not the least thing to be done is to update the “Last.Commit.Before.Current.Release” property and set a tag.

First you need to set a new tag, which will correspond to a current production state. To do this run these commands:

git tag PROD-20180101

git push origin --tags

The new tag is a base commit to evaluate delta to HEAD commit. It has to be specified in the ‘AutomatedProcedures\Default.Settings.include’ file. The file contains “Last.Commit.Before.Current.Release” property, change its value to the tag name you have just created:



### Fix CI and deployment errors

This section describes how to fix frequent deployment issues

1. Remove circular dependency  
   When you get an error that says one of the following:
   1. statement conflicted with the FOREIGN KEY constraint
   2. Item not found by some criteria

typically means that you have circular between packages. To resolve this you need to put aml of missing items to the .\AmlDeploymentScripts\1-BeforeAmlPackagesImport folder.

Configuration issues  
Sometimes configuration files got new sections but deploy procedure do not do anything with configuration files, in order to prevent loss of user preferences. To update configuration files - use upgrade scripts described at [Writing upgrade scripts](#_Writing_upgrade_scripts) section, particularly add/remove XML element and [NAnt: xmlpoke](http://nant.sourceforge.net/release/0.92/help/tasks/xmlpoke.html), [NAnt: xmlpeek](http://nant.sourceforge.net/release/0.92/help/tasks/xmlpeek.html).

# Reference



## Tools structure

The repository contains tools for automated tasks such as restoration of Innovator instance and deploy upgrade:

1. ContinuousIntegration.bat

Target audience: Development team

Purpose: Run analog of ContinuousIntegration build locally

1. RunIntegrationTests.bat

Target audience: QA team, Development team

Purpose: Run Integration tests.

1. RunTests.bat

Target audience: QA team, Development team

Purpose: Run all tests. This inculdes both unit and integration tests.

1. RunUnitTests.bat

Target audience: QA team, Development team

Purpose: Run Unit tests.

1. SetupInnovatorHere.bat

Target audience: QA team, Development team

Purpose: Setup new Innovator instance in local IIS using this repository as a code tree.

1. .\AutomatedProcedures\ConfigureNpmToWorkWithProxy.bat:

Target audience: Deployment engineer;

Purpose: Setup proxy server for NodeJs for environment without intenet access

1. .\AutomatedProcedures\ConvertPackageMethodsFromCsToXml.bat:

Target audience: Development team;

Purpose: Updates method code of package methods from the directory AML-packages with code from .cs files from the project CSharpMethods.csproj;

1. .\AutomatedProcedures\ConvertPackageMethodsFromXmlToCs.bat:

Target audience: Development team;

Purpose: Converts package methods from the directory AML-packages to .cs files and includes them to the project CSharpMethods.csproj;

1. .\AutomatedProcedures\CreateZipWithDeploymentPackageAndScripts.bat:

Target audience: Deployment engineer;

Purpose: Create Zip archive with deployment script that can be used to upgrade the target server;

1. .\AutomatedProcedures\Deploy.bat:

Target audience: QA team, Deployment engineer;

Purpose: Deploy upgrade to the target server

1. .\AutomatedProcedures\DeployFromZipArchiveWithPackageAndScripts.bat:

Target audience: Deployment engineer, QA team;

Purpose: Deploy upgrade to the target server from zip with diff

1. .\AutomatedProcedures\DeployToCustomEnvironment.bat:

Target audience: QA team, Deployment engineer;

Purpose: Deploy upgrade to a different from Deploy.bat target server

1. .\AutomatedProcedures\NantScript.xml:

Target audience: Deployment engineer;

Purpose: Contains nant scripts with targets used by the mentioned batch files.

1. .\AutomatedProcedures\PublishCodeTreeBackup.bat:

Target audience: Deployment engineer;

Purpose: Create CodeTree.zip based on current innovator state and publish it

1. .\AutomatedProcedures\PublishDatabaseBackup.bat:

Target audience: Deployment engineer;

Purpose: Create DB.bak based on current innovator state and publish it

1. .\AutomatedProcedures\PublishNewBaseline.bat:

Target audience: Deployment engineer;

Purpose: Create CodeTree.zip and DB.bak based on current innovator state and publishes it using Path.To.CodeTree.Zip and Path.To.DB.Bak properties.

1. .\AutomatedProcedures\RemoveThisInnovator.bat:

Target audience: Development team;

Purpose: Remove instance of Innovator, that was configured by SetupInnovatorHere.bat. This action will remove IIS application, database and Agent Service

1. .\AutomatedProcedures\RestoreInnovatorFromBackups.bat:

Target audience: QA team;

Purpose: Restore instance of Innovator from code tree and database backups;

1. .\AutomatedProcedures\SetupInnovatorFromBaseline.bat:

Target audience: Developer team, QA team

Purpose: Setup local instance of Innovator from production backups