Factoring out code into artifact repositories

Table of Contents

This guide provides a "recipe" for extracting a module (typically a single `.csproj` project) out of the "all-in-one" repo, to be built and tested in its own BOMi artifact repo.

Objectives

In the short-to-medium term there will be two "types" of git repositories:

- BOMi "artifact" repos: those that contain the source code for a single packaged artifact (usually a nuget component, but could in principle be anything)
- everything else, hosted in $/INTEG_MIRROR/Trunk.

The plan is to steadily factor out the existing codebase from the latter into the former; eventually there will be 100s of the former. However, this is an "eat the elephant" type of undertaking, not something that can be done overnight. See the transition plan for a more detailed view as to how we'll get there.

What we want to end up with is:

- a new git "BOMi artifact" repo holding the code and tests of the artifact
- a corresponding CI pipeline to automatically build/test/package the artifact.

  In most cases this will be as a nuget package made available via the nuget server).

- existing code in $/INTEG_MIRROR/Trunk updated to use the packaged artifact

  These changes will be sync'd over to `$/INTEG`.

- the original source code moved in TFVC from $/INTEG/Trunk to $/SDM_Services/MigratedToGit

  This will ensure no file change history will be lost.

Most of the work is done on git repos, in a separate branch. This reduces the chance of a mistake impacting other development. The last step - moving the source code from `$/INTEG` - necessarily must be done on TFVC rather than in git, because that is the only way of preserving file history. If moving the code causes a problem (ie the existing CI starts to break), it is very easy to revert.

Before you start

Prerequisites:

- Identify the code to be moved out from `$/INTEG_MIRROR`, and which TFS team project it should reside in (eg `BOMi`, `BOMi_Infrastructure` etc)

  > Ensure that there are no substantive changes for the code in any of the current project branches (still to be merged), and make sure this code has minimal coupling/is fully decoupled. Obviously, for some modules this may be a significant task.

- create a user story work item on `$/INTEG_MIRROR` to track the work

  The work item number should be used in commit messages. For example, if work item '12345' was created, then commits should be in the form:

  ```
  #12345 - extracts the Sdm.Foo.Bar component from INTEG_MIRROR.
  ```

  Note the use of present continuous tense: "this commit, if applied,

- Let all teams know about the change in advance:

  Add the component you are extracting to the Nugetification Schedule

Create GIT branches

Create git repo, create branch

Create an empty git repo via TFS in the appropriate team project; this will house the extracted BOMi artifact.

Next, we need `.gitignore` and `.gitattributes`.

- The `.gitignore` file is used to specify which files should be checked into a git repository
  - A sample file can be downloaded here (rename to `.gitignore`).
  - For further info, see .gitignore reference.
- We also need an initial `.gitattributes` file. The primary purpose of this file is to describe whether different file types are text or binary (and so, whether they can be diff'd and merged).
  - A sample file can be downloaded here (rename to `.gitattributes`).
  - For further info, see .gitattributes reference.
- Commit and push these changes back to `origin`.

Then, create a branch in the new git repo; all work will be done on this branch:

```
git pull
git checkout -b 12345-extracts-sdm-foo-bar
git push origin 12345-extracts-sdm-foo-bar -u
```

Create INTEG_MIRROR branch

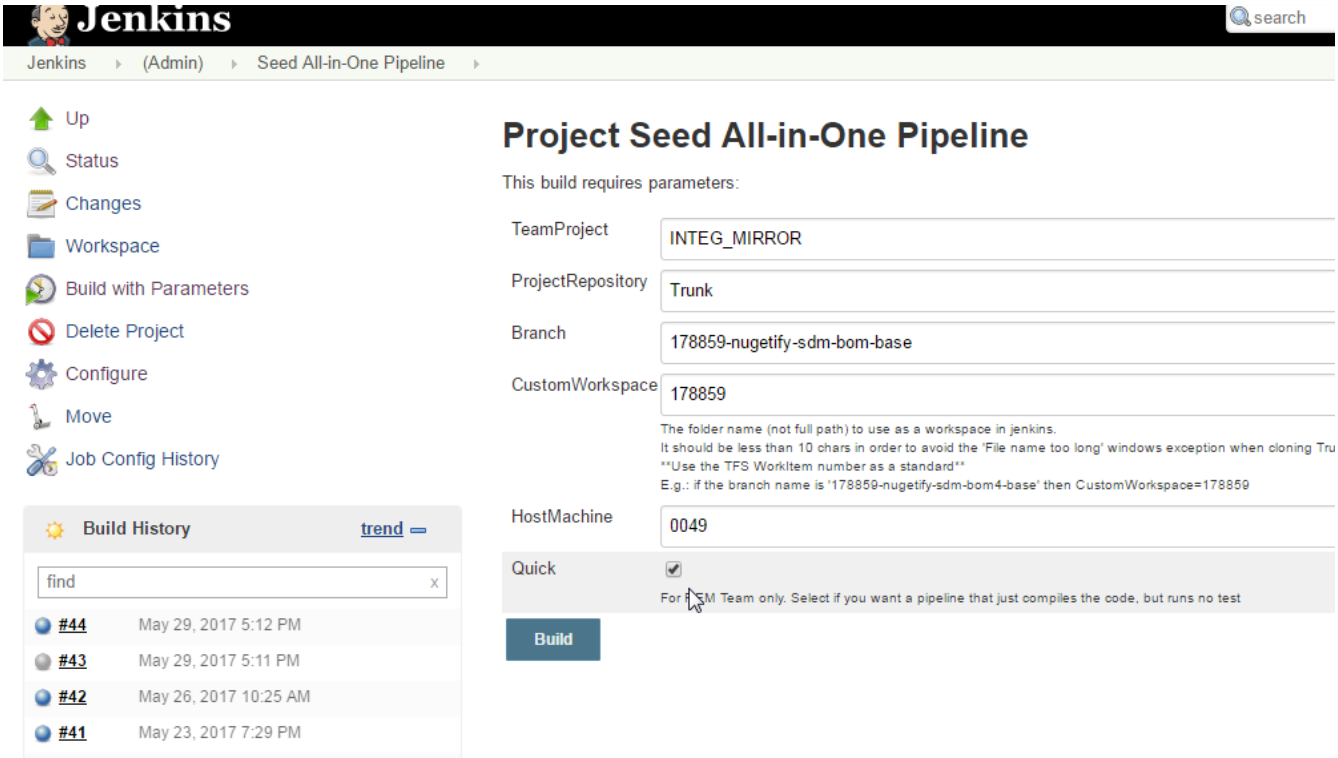
Èxtracting the code out of `$/INTEG_MIRROR` will result in changes to existing projects (they will end up using a nuget package rather than a DLL directly). So again, rather than working directly on the `master` branch of `$/INTEG_MIRROR/Trunk`, we work on a branch instead.

Assuming you've already cloned `$/INTEG_MIRROR/Trunk` git repo (see here if not), create a branch both locally and on the remote:

```
git checkout master
git pull
git checkout -b 12345-extracts-sdm-foo-bar
git push origin 12345-extracts-sdm-foo-bar -u
```

> It's important to use the exact same branch name in all repos.
>
> The name is used later on to ensure that the respective CI pipelines are able to resolve the extracted nuget packages (prior to merging back into `master`).

Then, set up a CI pipeline for your new branch of `$/INTEG_MIRROR/Trunk`:



Extract the Component

Copy code out of $/INTEG_MIRROR/Trunk

Copy the files out of `$/INTEG_MIRROR/Trunk` and into the new git "artifact" repo created previously.

New solution

In the git artifact repo:

- run the `Prepare-NugetProject` command on the extracted projects.
- create a new Visual Studio solution for the code that's been copied in.

It should reference both the production `.csproj` (or `.vbproj`) project as well as any unit tests.

Confirm that Visual Studio can build the projects ok.

Create the artifact.yml file

Every BOMi artifact repository requires a `artifact.yml` file, residing at the root of the repository. This is used:

- by the `Build-NugetSolution` script, to know how to build the artifact, and
- by the CI system (described below), +Build locally, do not push just yet.

Use an existing artifact.yml as a template.

Create a top-level README.md

Every git repo should have a top-level `README.md` explaining its content. As a minimum it should contain:

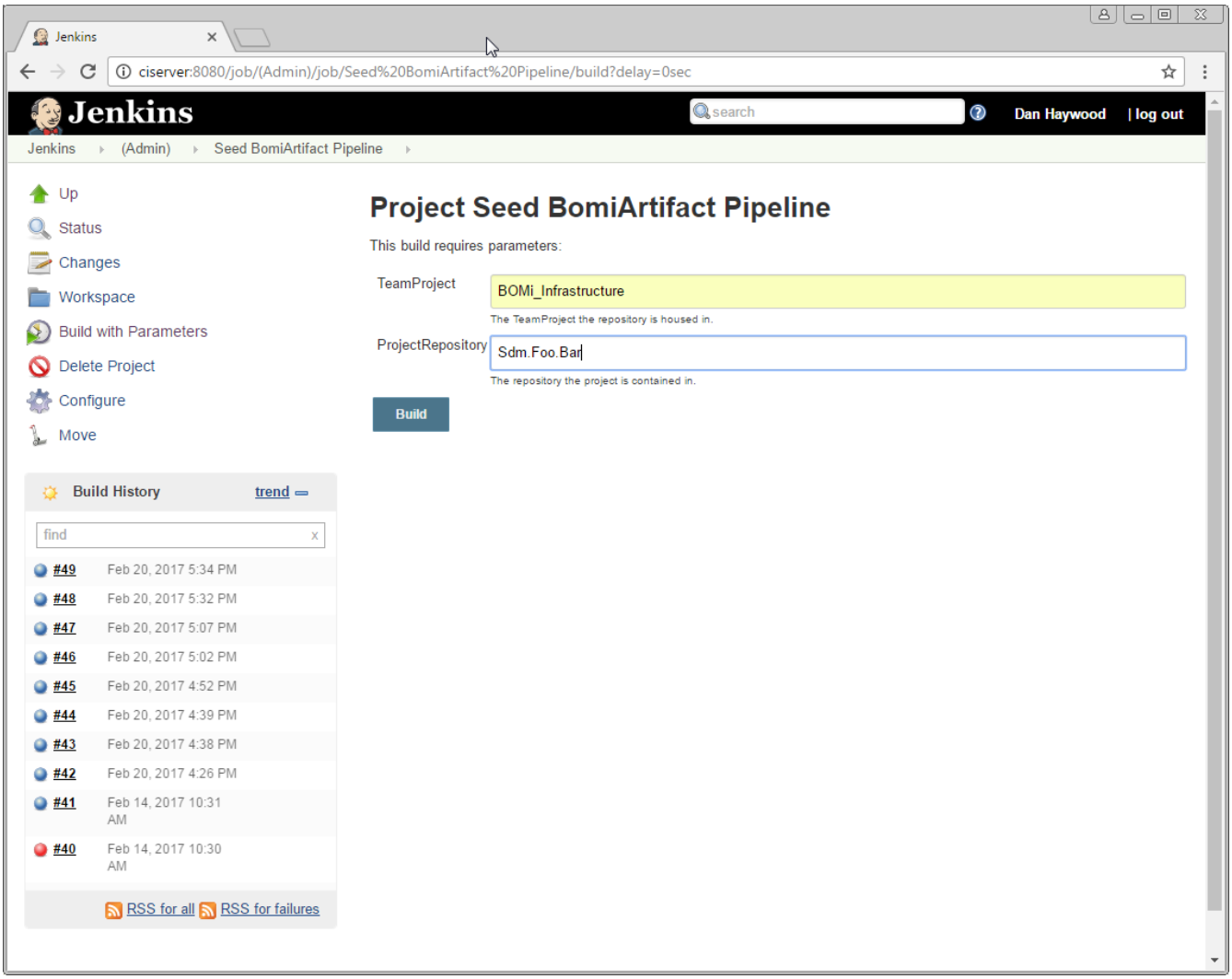
- short description
- a `artifact.yml` file
- the CI pipeline jobs
- the nuget package (if any)
- Change Log
- Notes (if any)

An example `README.md` can be seen here.

Seed the CI jobs for the repo

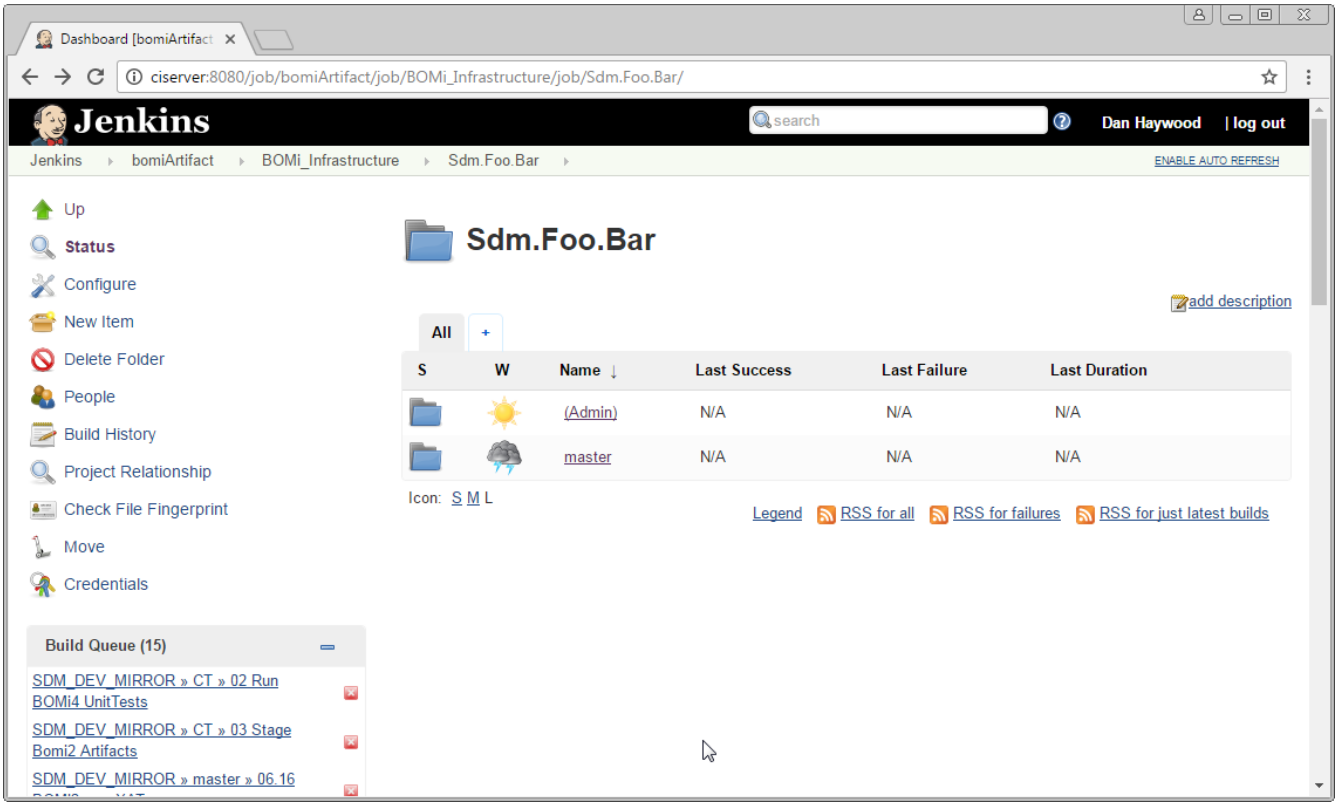
Each project that has been copied out of `$/INTEG_MIRROR` requires a new CI pipeline. This is set up

using the "seed BOMi Artifact Pipeline job":



This uses various templates.

The screenshot below shows the jobs and views that it creates:



This creates jobs that:

- build the pipeline whenever changes are pushed to `master`
- will automatically create and execute the pipeline on any pull request

> TODO: not properly implemented; currently monitors only a `ready/*` branch rather than from a pull request tag.
>
> See https://trello.com/c/mt0I588U

- allows the developer to create a pipeline for any arbitrary branch

Seed Pipeline for the Component's branch

You've created a Git repository for your component. You've also created a Jenkins pipeline for it.

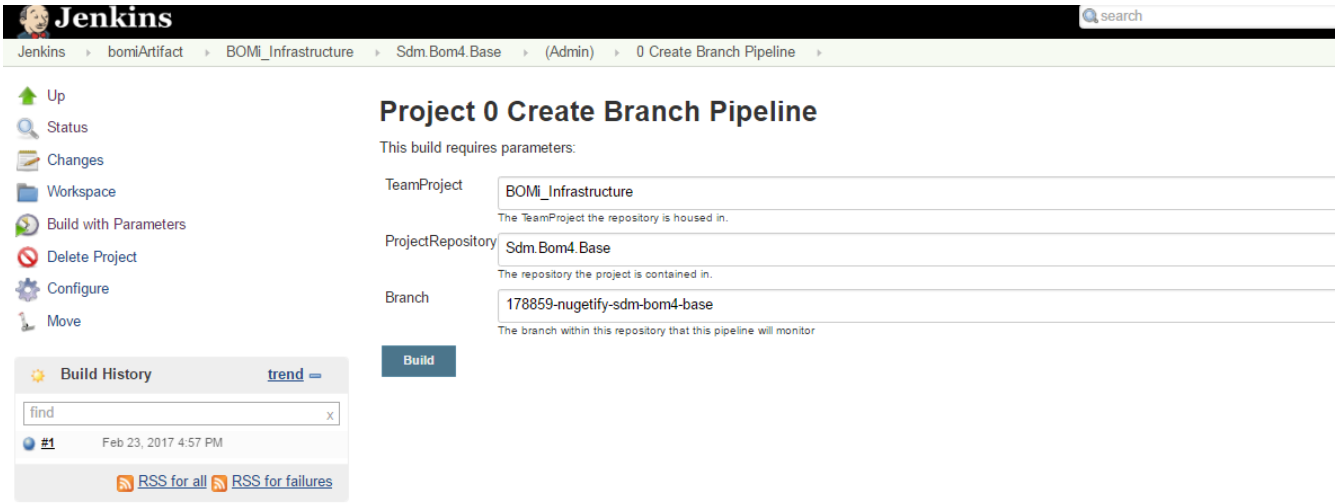
This means that at the moment Jenkins has the following jobs created:

```
Initial Jenkins Jobs:
* bomiArtifact/%ProjectTeam%/%ProjectRepo%/%component%/(Admin)/Create Branch Pipeline
* bomiArtifact/%ProjectTeam%/%ProjectRepo%/%component%/(Admin)/Delete Branch Pipeline
* bomiArtifact/%ProjectTeam%/%ProjectRepo%/%component%/master/Get %component%
```

When the last job completes execution, it will complete the creation of the pipeline for the master branch. In other words %myBranch%/Get %component% is the seeding job for each branch pipeline. Since we are working off a non-master branch, we will need to create a version of Get %component% for our branch. We do this by executing the first job in the list:

```
bomiArtifact/%ProjectTeam%/%ProjectRepo%/%component%/(Admin)/Create Branch Pipeline.
```

For example, here is how you would build the branch 178859-nugetify-sdm-bom4-base in the Sdm.Bom4.Base repository:



Check CI Scripts

Look at the configuration of the CI pipeline jobs (for the `master` branch), and run the corresponding commands locally:

- check that all the expected steps are present.

  For example, that any unit- or integration test jobs have been created correctly.

- run the build.

  For example, `Build-NugetSolution`

- run tests.

  For example, `Run-NugetUnitTest`

> The jobs when run from CI are called with a "-CI" flag. This should be replaced with "-Local" if run from the developer's PC.

Fix any issues, and then commit the changes to your local branch and push to `origin`.

Check CI Pipeline

Assuming that you created a pipeline for your branch (see above), then the CI server will automatically publish to a NugetWorkInProgress nuget source.

> TODO: not yet implemented.
>
> See https://trello.com/c/sxuqwQM7

Fix up INTEG_MIRROR

With a version of the new nuget package available via `NugetWorkInProgress`, the next step is to fix up project references for existing code in `$/INTEG_MIRROR`.

Change references from other projects

To do this, the `Update-NugetRefs` command can be used. For example:

```
Update-NugetRefs –Package Sdm.Cluster.Root.Api –Version 1.2.0 -Git
```

Remove the project from DefinedSolutionProjects

Inspect the DefinedSolutionProjects.ps1 file.

If the project that has been extracted is explicitly listed, then remove it.

Remove dll from lib folder

Old references to the dll exist in the lib folder. They must be removed.

For example this command shows a search in Trunk for the corejava dll.

```
gci -Recurse "sdm.corejava.dll"
```

For your component it would be the name of the dll. Any of the references to that dll must be removed.

We only want the reference to the dll in the packages folder.

Remove the project from the Visual Studio Solution

In Visual Studio, open the solution file containing this. Locate the project you are extracting to Git, right click on it, and select "Remove" (or just press Del) Save the solution.

To see the list of solutions referencing the project, you can run this command (replacing Sdm.Test.Common.csproj with the name of your project). Make sure you execute this command from the root folder of your INTEG_MIRROR clone.

```
gci -rec *.sln|sls "Sdm.Test.Common.csproj"|group path|select name
```

Remove from Trunk\Code\build.xml

If the solution is the only one that exists in trunk then it must also be removed from build.xml then delete it.

TODO: Delete solution if no project is left (but also remove it from Build.xml)

Delete the extracted Source Code

Locate the folder or folders that have been moved to the component's Git repository. Delete them from $\Trunk.. and ensure you Stage the deleted files in git (i.e. stage the deletion)

Publish the changes

Commit and push the changes to your branch of `$/INTEG_MIRROR`. This will then kick off the pipeline created earlier, using the branch name to resolve the extracted nuget package from the NugetNugetWorkInProgress source.

> Review each file you commit. There may be files modified unintentionally (e.g. DLLs if you builded locally)

> TODO: having the "dependent" CI pipeline (eg INTEG_MIRROR) automatically use the correct nuget source for "work-in-progress" packages is not yet implemented.
>
> See trello: WIP nuget source (bomi artifact) and trello: WIP nuget source (all-in-one)

Raise PRs to merge into `master`

To actually publish the nuget package, raise a pull request for the "artifact" repo, to merge the branch into `master`.

Once approved, this will kick off a new pipeline for the (tag representing the) pull request.

If the pipeline succeeds, a new nuget package will be a published to the nuget server.

Confirm that the CI pipeline has passed successfully, and that any artifacts have been published, eg:

```
c:\nuget\nuget.exe list -source http://nugetserver/nuget/nuget/ |`
                    select-string sdm.foo.bar
```

Similarly, raise a PR to merge the changes in `$/INTEG_MIRROR` from your branch and into `master`. Keep an eye on the "master" pipeline to ensure that it builds correctly.

These changes will then be synced automatically over to `$/INTEG`.

> Previously there was a step to archive the code to $/SDMServices.
>
> However, attempting to do this will lead to conflicts with the files deleted by way of the sync from `$/INTEG_MIRROR`.
>
> In any case, there is no need: the history of deleted files can still be viewed in TFS (or indeed in the `$INTEG_MIRROR` git repo, if checkout just prior to the module being deleted as a result of merging in the branch, as described above).

Perform After-Merge tasks

The following tasks should be performed once a development branch is merged to master and deleted. Merging and deleting the branch will be performed by the Pull Request Reviewer once he/she accepted and completed the changes.

Delete Jenkins pipeline

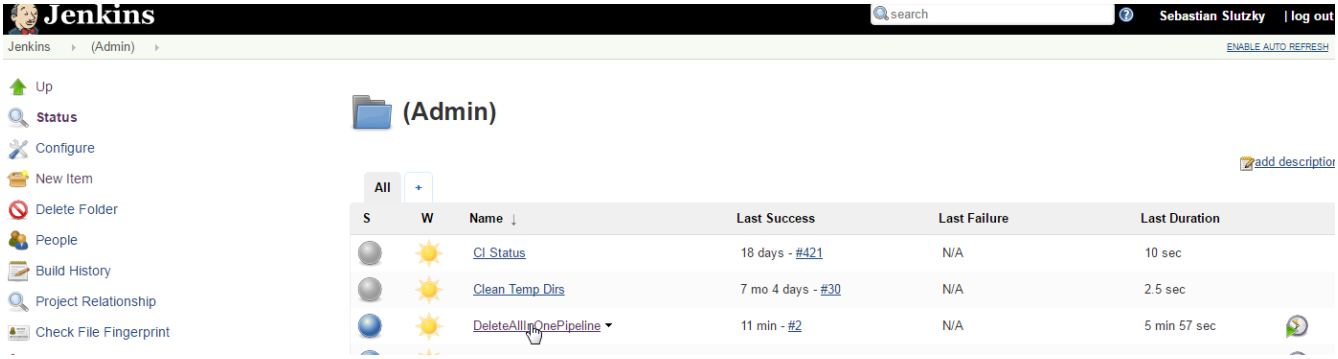
The administrative Jenkins job "DeleteAllInOnePipeline" is used to delete an all-in-one pipeline once the associated Git branch has been deleted on the remote server.

This job will perform the following tasks:

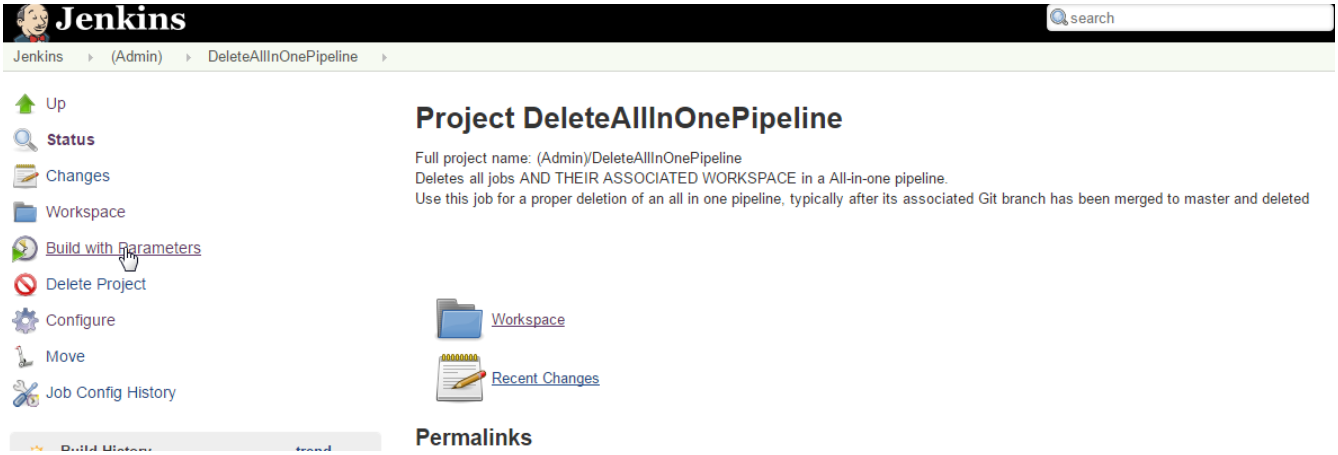
- "Wipe" the workspace of each job in the pipeline (regardless it exists on jenkins master or a slave)
- Delete each job in the pipeline
- Delete the Jenkins folder for the pipeline

> In the future, this step will be automatically performed when a branch is deleted. Follow this ticket on Trello https://trello.com/c/fdkmTIVk to check when is done

To run this job, go to the Jenkins job "(Admin)/DeleteAllInOnePipeline"

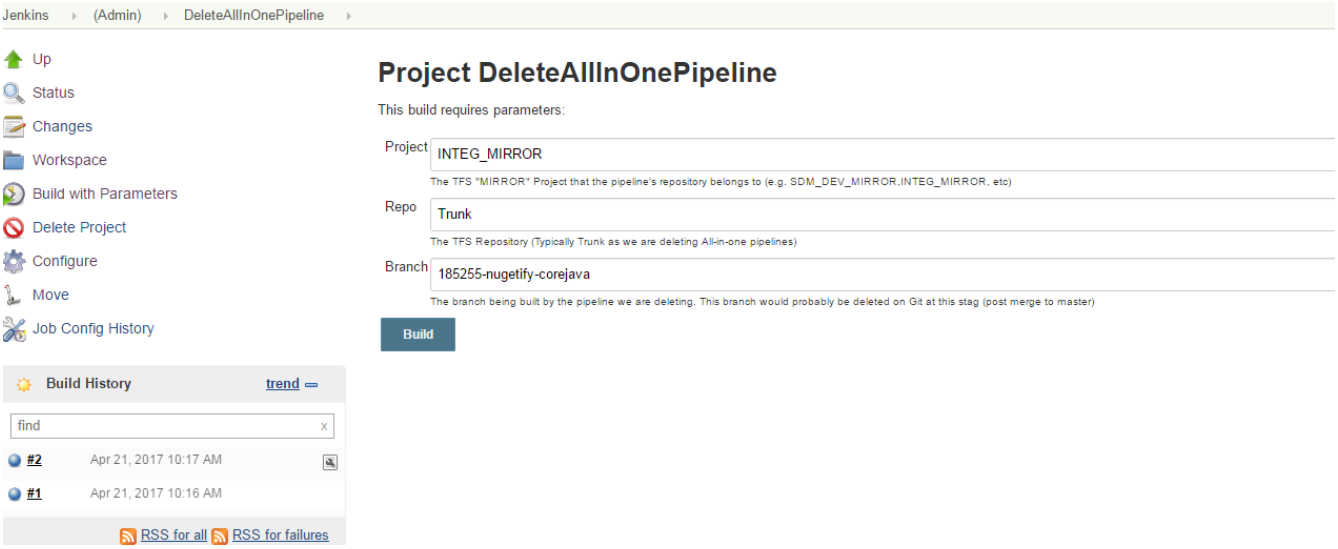


Now build the job by selecting "Build with Parameters"

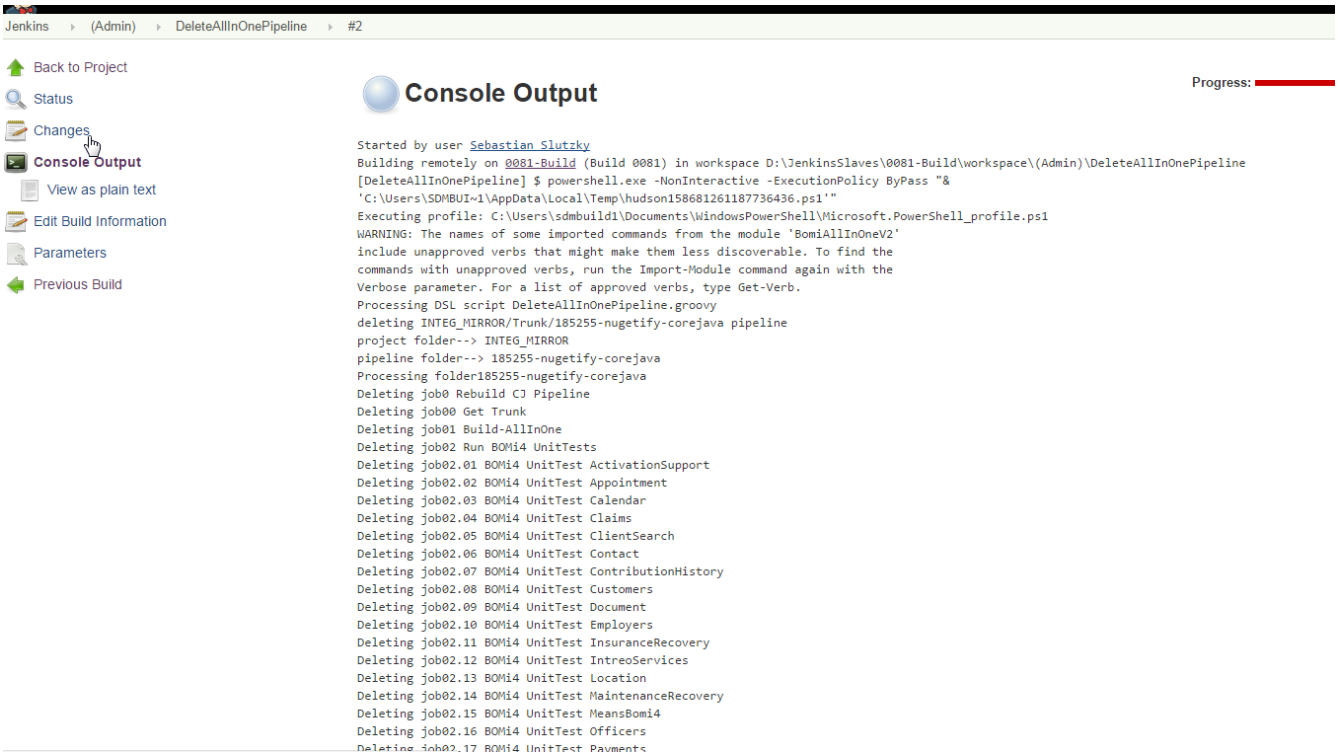


Finally, enter the values for Project and Branch parameters (The repo parameter should be left

untouched for AllInOne repositories) Once the paramters are complted, select "Build" to build the job.



That's it, the job will take a few minutes and produce an output similar to this one:



Update documentation to reflect progress

Update the nugetification schedule: Update the Nugetification Schedule to reflect the progress.

Update the nugetification analysis odc: Update the Nugetification Analysis page to reflect progress and see what is next to nugetify.

Delete local branch (optional)

You probably want to delete the local branch. Do this by running this command from your "Trunk\Code" folder

```
git branch -D <branch name>
```

Alternatively, you can execute this script, which will delete all local branches no longer on the server.

```
$local=git branch -l
$remote=git branch -r
$local|
   %{$_.Trim()}|
   ?{-not ($remote -like '*' + $_) }|
   ?{-not($_ -match "master" )}|
   %{git branch -D $_}
```