HAAGA-HELIA
University of Applied Sciences

Implementing Continuous Integration and Cloud Server Infrastructure for an International Enterprise Based In Finland

06 October 2015

Author(s)
Rudy Joel Lopez Reyes

Degree programme
Business Information Technology

Report/thesis title
Implementing Continuous Integration and Cloud Server Infrastructure for an International Enterprise Based In Finland

Number of pages and appendix pages
73

This project consists of 3 parts. The first one is the migration of the company's website project's repository from SVN to Git, which involves the migration of the project itself plus the transition of the developer's team from working with an SVN workflow and guidelines designed for SVN to Git workflow and guidelines for working with Git.

The second part is the migration of the project from a set of on-premise hosted servers that had expired their recommended life span of 3 years to a brand-new set of servers provided by a third-party cloud hosting service.

This part involves the decision process of selecting a cloud hosting server provider, the migration of the project from on-premise hosting to cloud hosting and a discussion on the advantage of cloud hosting over on-premise hosting.

The third parts connects the first with the second part consisting of the built automation and continuous integration that now clones the repository from Git to a build sever that builds the project and then the project is transferred to the new servers to make it available over the internet.

Keywords
SVN to Git migration, SVN, Git, Continuous Integration, Cloud Server Hosting, Build Automation

Table of contents

1 Introduction

The project consisted of three parts, the migration of the project's repository from SVN to Git, the migration of the server environment from on-premise to the cloud, and setting the continuous integration and build automation that would clone the repository, build it and serve it on the new cloud server environment.

The main purpose of the project was setting up an improved continuous integration, build automation and deployment to the cloud process by actualizing the process to the newest technology available.

Before the project, the technologies and processes used were presenting challenges that could be overcome by more recent ones.

On the side of version control systems, SVN was presenting challenges regarding the integration of new development into the used branches causing visibility and stability issues.

On the side of the server environment, the servers were already too old and their recommended life span had already been over utilized, meanwhile the build process was done mostly manually which compromised the efficiency and reliability of the process.

On order to solve these presented challenges the project was designed so that a migration to Git had to be done including the redefinition of version control workflows, processes, guidelines and habits, incorporation of feature branches and pull request to increase the visibility and stability of the project.

Also a new server environment infrastructure had to be designed and implemented so that the project could be served from the cloud and server resources would be scalable.

Lastly, the build was planned to be automated enough so that time spending building and deploying the latest code would be reduced, increasing the efficiency of the deploying process.

2 Source Code Migration from SVN to GIT

2.1 Background

Up to November of 2014 Subversion (SVN) was the version control system utilized in for the company's project. (Kiril Jovchev 15 May 2015)

In November of 2014 the decision was made to move the source code from SVN to a Git repository, namely Bitbucket.

Before taking the decision for the migration most other projects handled by the company were already using Git, although the repository of choice was Gitlab.

One of the main reasons for making the migration to Git and specifically to Bitbucket was that there was a need for having a multivendor environment so that partners outside the company's network could easily access the project's source code. (Kiril Jovchev 15 May 2015)

Bitbucket is a cloud based repository that makes making customers and partners administrators possible in comparison to Gitlab or SVN which were on-site repositories inside the company's firewall to which customers and partners are denied access.

Another strong reason to move to Git was that SVN is a centralized version control system while Git is a distributed version control system.

Having a distributed version control system would give the developers' team many advantages (mentioned below) over SVN. (Giancarlo Lionetti February 14, 2012)

This was important because there were at the time many paths of development including new features development and Sitefinity upgrades.

2.2 SVN Structure

The structure of the SVN repository (Figure 1) was the following (Avaus Consulting Oy, 2014):

- **Branches:** 36 different branches
- **DatabaseBackups:** SQL binary backups
- **Design:** HTML & PSD designs
- **Sitefinity:** Sitefinity source code
- **Tags**: tags created by CCnet builds
- **Trunk:** project source files that are compiled into releases

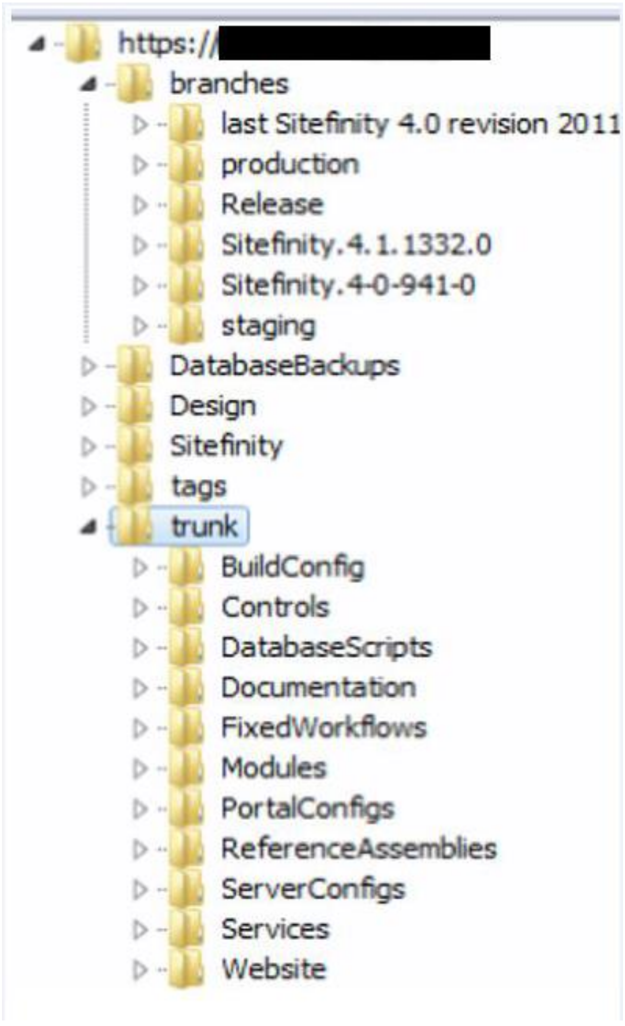
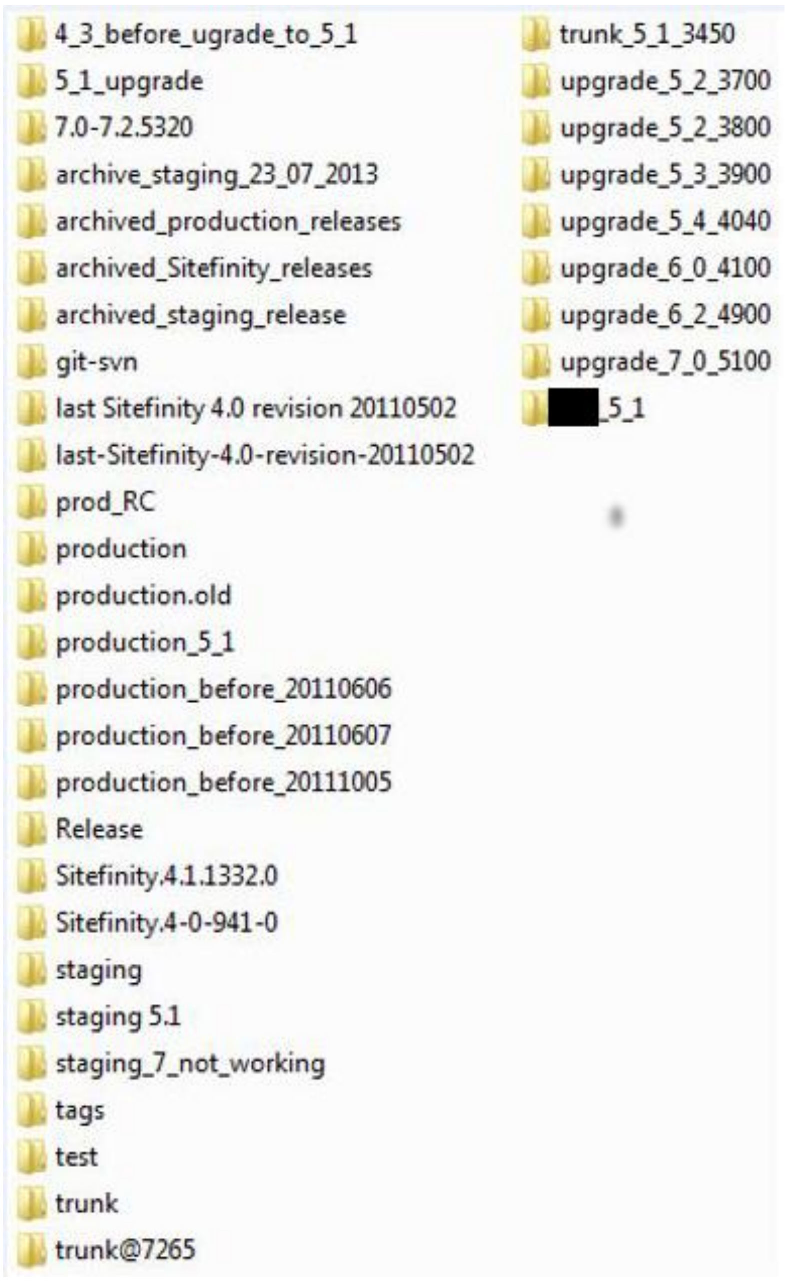

```
https://[redacted]
  branches
    last Sitefinity 4.0 revision 2011
    production
    Release
    Sitefinity.4.1.1332.0
    Sitefinity.4-0-941-0
    staging
  DatabaseBackups
  Design
  Sitefinity
  tags
  trunk
    BuildConfig
    Controls
    DatabaseScripts
    Documentation
    FixedWorkflows
    Modules
    PortalConfigs
    ReferenceAssemblies
    ServerConfigs
    Services
    Website
```

Figure 1. Project's SVN Structure

2.3 SVN Branches

The SVN repository had about 36 branches (Figure 2) including branches with the source code for deploying to the staging and production environments, different versions of the Sitefinity source code, Sitefinity upgrades, tags and the trunk folder which was used to deploy to the development environment. (Avaus Consulting Oy, 2014)



```
4_3_before_ugrade_to_5_1
5_1_upgrade
7.0-7.2.5320
archive_staging_23_07_2013
archived_production_releases
archived_Sitefinity_releases
archived_staging_release
git-svn
last Sitefinity 4.0 revision 20110502
last-Sitefinity-4.0-revision-20110502
prod_RC
production
production.old
production_5_1
production_before_20110606
production_before_20110607
production_before_20111005
Release
Sitefinity.4.1.1332.0
Sitefinity.4-0-941-0
staging
staging 5.1
staging_7_not_working
tags
test
trunk
trunk@7265
trunk_5_1_3450
upgrade_5_2_3700
upgrade_5_2_3800
upgrade_5_3_3900
upgrade_5_4_4040
upgrade_6_0_4100
upgrade_6_2_4900
upgrade_7_0_5100
[redacted]_5_1
```

Figure 2. Project's SVN Branches

2.4 The SVN Workflow

There was a well stablished workflow procedure to get the work done from the development environment to the production environment.

The workflow had to go first from the development environment to the staging environment and after this finally to the production environment.

The protocol dictated the following rules:

2.4.1 From development to staging

- make sure that you have both **trunk** (development) and **staging branch** checked out in your local workspace
- do merging from **trunk** into **staging branch** from the desired revisions or revision range(s)
- do **staging build** using **CCnet** console
- copy and unzip the compressed build package from the development server into both staging servers at the location **D:\temp**
- stop all web sites on the IIS instance on the current server and then stop the whole IIS as well
- remove / copy the original **D:\staging\Inetpub**
- copy the folders from the uncompressed folder into **D:\staging\Inetpub**, usually only the **PortalConfigs** and **Website** folders should be overwritten in case **WAS** module is being left intact
- **NOTE!** Some folders like **Website\AppData** must be shared amongst the cluster deployments so copy also **Website\AppData** folders from the deployment into folder **D:\root\shared\Website\App_Data**
- start IIS and the desired websites and do the QA testing

(Avaus Consulting Oy, 2014)

2.4.2 From staging to production

Once the staging build has passed the QA process successfully in the production environment, follow these steps:

- rename and move **production** branch as **production.before.<YYYYMMDD>** under **archive** branch

- make a copy of the **staging** branch into **production** branch so that CCnet build integration has access for that
- do **production build** using **CCnet** console
- copy and unzip the compressed build package from development server to both production servers into folder **D:\temp**
- stop all web sites on the IIS instance on the current server and then stop the whole IIS as well
- remove / copy the original **D:\production\Inetpub**
- copy the folders from the uncompressed folder into **D:\production\Inetpub**, usually only the **PortalConfigs** and **Website** folders should be overwritten in case **WAS** module is being left intact
- **NOTE!** Some folders like **Website\AppData** must be shared amongst the cluster deployments so copy also **Website\AppData** folders from the deployment into folder **D:\root\shared\Website\App_Data**
- start IIS and the desired websites and do the QA testing

(Avaus Consulting Oy, 2014)

2.5 General Rules to work with SVN

One of the issues encounter when using SVN in contrast with using a source control management system like TFS is that due to the fact the .proj file is changing constantly and there are multiple developers making changes, the developer is very likely to encounter conflicts when committing the file. (Avaus Consulting Oy, 2014)

In general conflicts like the above mentioned were avoided by committing changes as often as possible, and making commits as atomic as possible.

This means that each commit should contained only one fix or feature as a change.

Another important rule is to give proper and descriptive messages to each commit, this plus atomicity in each commit will make picking which fixes and features go from the development to the staging and finally to the production environment. (Avaus Consulting Oy, 2014)

When dealing with commits that fix an issue reported using the issue tracking system, the ticket id most be provided in the commit message.

Example of good commit message:

FIX JIRA-123: Example issue. Provide also a brief description about the fix for the problem.

It should be taken into account that each environment has different Sitefinity configuration files and these have to be branched out per environment basis and merged from the trunk when needed.

One of the configuration files which is the one containing the connection string should be put in the SVN ignore list.

All the right configuration files will be deployed to the right environment and servers, this will be ensured at the time of copying the files from the development server to the staging and production environments. (Avaus Consulting Oy, 2014)

2.6 Issues with original plan

The plan described above worked until continuous development was introduced to the workflow.

One problem that continuous development posed was that at the end of a spring only a subset of features in the development branch was ready therefore the whole development branch could not be merge to the staging branch. (Kiril Jovchev 15 May 2015)

Before the introduction of continuous development the development branch was used for all new development and the stage branch started to be used for user acceptance testing.

At this point in time the features were very well specified and defined so that all features had to pass the user acceptance testing if they followed the specifications, this meant that the whole staging branch could be merged into production. (Kiril Jovchev 15 May 2015)

When continuous integration testing was introduced the definitions and requirements of new features was not very well specified and therefore many features developed were not accepted so therefore only a subset of the features could go to production and so the staging branch could not be completely merged into the production branch.

To solve this problem, at the time of merging branches changes were cherry picked and merged into next branched in the workflow.

This cherry picking method raised the issue that different changes may depend on other changes that may not be picked and therefore features may got broken.

To tackle this issue regression testing was introduced. One issue was there was not a new environment for a release candidate where regression testing could be done, because of this staging environment became also release candidate. (Kiril Jovchev 15 May 2015)

2.7 Migrating to Git

At the time the source code was decided to be migrated to Git, other projects had already been migrated the Gitlab, but this time the repository of choice was Bitbucket.

Atlassian offers a tutorial to migrate a repository from SVN to Git that was used to do the migration. (Atlassian 2014)

The tutorial is divided into 5 steps:

- Prepare your environment for the migration.
- Convert the SVN repository to a local Git repository.
- Synchronizing the local Git repository when the SVN repository changes.
- Share the Git repository with your developers via Bitbucket.
- Migrating to Git

When doing the migration it is highly recommended to do a "one-way synchronization", meaning that developers would commit only to the SVN and not to Git until the migration is overs. (Atlassian 2014)

In order for the migration to go smoothly without any major complications it was decided that developers would stop making commits until the migration was done and afterwards they would be pushing to Bitbucket.

2.7.1 Preparing the environment

Preparing the environment consists of 3 steps:

- Downloading the migration script
- Mounting a case-sensitive disk image (if necessary)
- Extracting the author information

There are several elements that must be installed before we can do the migration, namely:

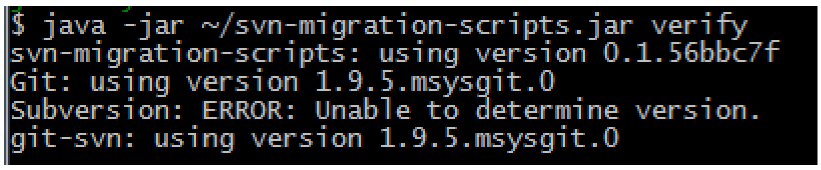
- Java Runtime Environment
- Git
- Subversion
- Git-Svn utility

2.7.2 Downloading the migration script

The migration script is a JAR file created by Atlassian that contains functionality that is not found in Git and it is necessary to make the migration.

The script can be found at https://bitbucket.org/atlassian/svn-migration-scripts/downloads. It is recommended to download this script to the home directory. (Atlassian 2014)

The script contains a command that is able to verify if we are missing any of the tools needed for the migration. In case we have the tool installed the version will appear, and if we don't have the tool installed we will get an error "unable to determine version" (Figure 3).



```
$ java -jar ~/svn-migration-scripts.jar verify
svn-migration-scripts: using version 0.1.56bbc7f
Git: using version 1.9.5.msysgit.0
Subversion: ERROR: Unable to determine version.
git-svn: using version 1.9.5.msysgit.0
```

Figure 3. Git-SVN migration error

2.7.3 Mounting a case-sensitive disk image

The tutorial says that the migration must be done on a case-sensitive file system in order to avoid the corruption of the repository. (Atlassian 2014)

The tutorial warns that this will be a problem when making the migration on an OS X computer because it is not case-sensitive and they provide a workaround with a script that will turn a directory into a virtual case-sensitive file system where the migration can be performed.
In our case we tried doing the migration on a Windows 7 OS and we found some issues that made the migration very difficult.

In later steps when trying to convert the SVN repository into a Git one we found the following error:

Git - fatal: Unable to create '/path/my_project/.git/index.lock': No such file or directory read tree xxxx command returned error: 128

This error which was not descriptive enough ended up being a case sensitivity issue because the reason of this error was that some branches' name had spaces and windows was unable to read the whole branch's name but just the first word and therefore didn't find the branch which in turn may the whole conversion process stop.

Changing the name of the branch proved to be very difficult so instead the migration was tried again on a virtual machine (using virtual box) with Ubuntu where we didn't find the issue.

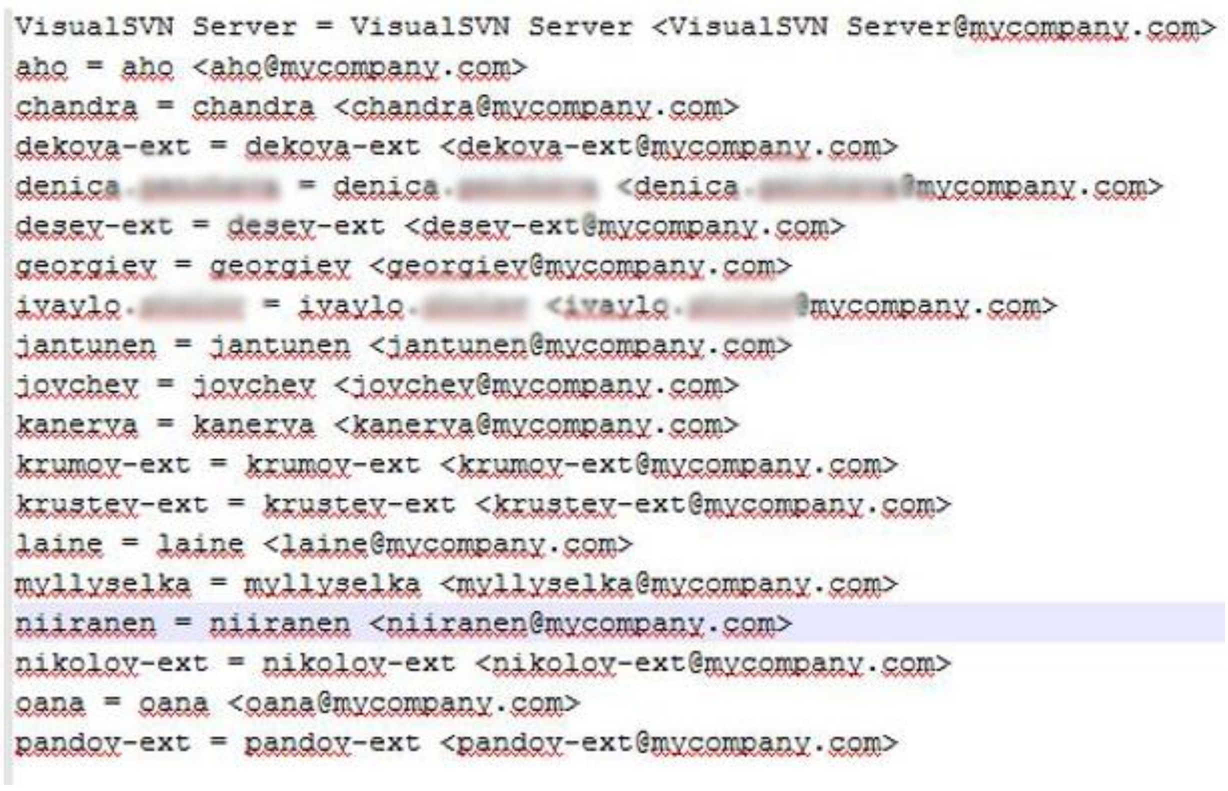
2.7.4 Extracting the author information

While SVN only uses the username of the person who makes the commits (the author) Git uses the name and email address, therefore we need to extract this information from SVN.

The migration script comes with the following command that allow us to generate a text file with all the authors:

```
cd ~/GitMigration
java -jar ~/svn-migration-scripts.jar authors <svn-repo> > authors.txt
```

After running this command we will have a text file called authors looking like the following (Figure 4):



```
VisualSVN Server = VisualSVN Server <VisualSVN Server@mycompany.com>
aho = aho <aho@mycompany.com>
chandra = chandra <chandra@mycompany.com>
dekova-ext = dekova-ext <dekova-ext@mycompany.com>
denica.[redacted] = denica.[redacted] <denica.[redacted]@mycompany.com>
desev-ext = desev-ext <desev-ext@mycompany.com>
georgiev = georgiev <georgiev@mycompany.com>
ivaylo.[redacted] = ivaylo.[redacted] <ivaylo.[redacted]@mycompany.com>
jantunen = jantunen <jantunen@mycompany.com>
jovchev = jovchev <jovchev@mycompany.com>
kanerva = kanerva <kanerva@mycompany.com>
krumov-ext = krumov-ext <krumov-ext@mycompany.com>
krustev-ext = krustev-ext <krustev-ext@mycompany.com>
laine = laine <laine@mycompany.com>
myllyselka = myllyselka <myllyselka@mycompany.com>
niiranen = niiranen <niiranen@mycompany.com>
nikolov-ext = nikolov-ext <nikolov-ext@mycompany.com>
oana = oana <oana@mycompany.com>
pandov-ext = pandov-ext <pandov-ext@mycompany.com>
```

Figure 4. Project's SVN Structure

We then can edit this file and input the name and email address of the authors to later use them in Git.

2.7.5 Convert the SVN repository to a local Git repository

After preparing the environment we had to convert our SVN repository into a local Git repository. For this we used the git svn utility we had acquired earlier.

The tutorial mentioned 2 ways of using this command depending on whether we had a standard SVN layout or not. (Atlassian 2014)

The standard SVN layout is:

- Repository
  - Branches
  - Tags
  - Trunk

In our case we didn't have a standard SVN lay out so we used a command where we had the option to specify the location of all these.

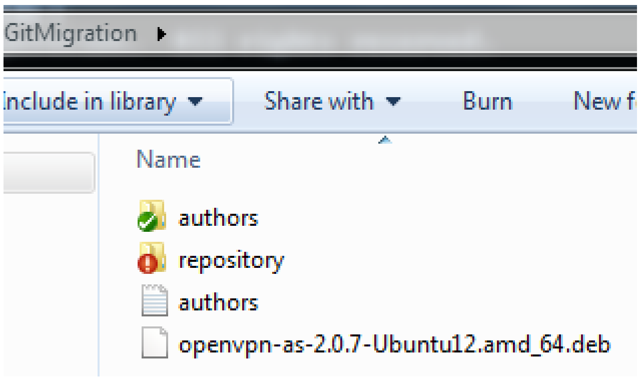
The command we used was the following:

```
git svn clone --trunk=<trunk location> --branches=/branches --tags=<tags location> --authors-file=authors.txt
<svn-repo>/<project> <git-repo-name>
```

This commands goes through each branch and each commit and transforms this into a Git repository with its history. In our case we had branches with as many as 5000 commits, this made the converting process last for about 6 hours.

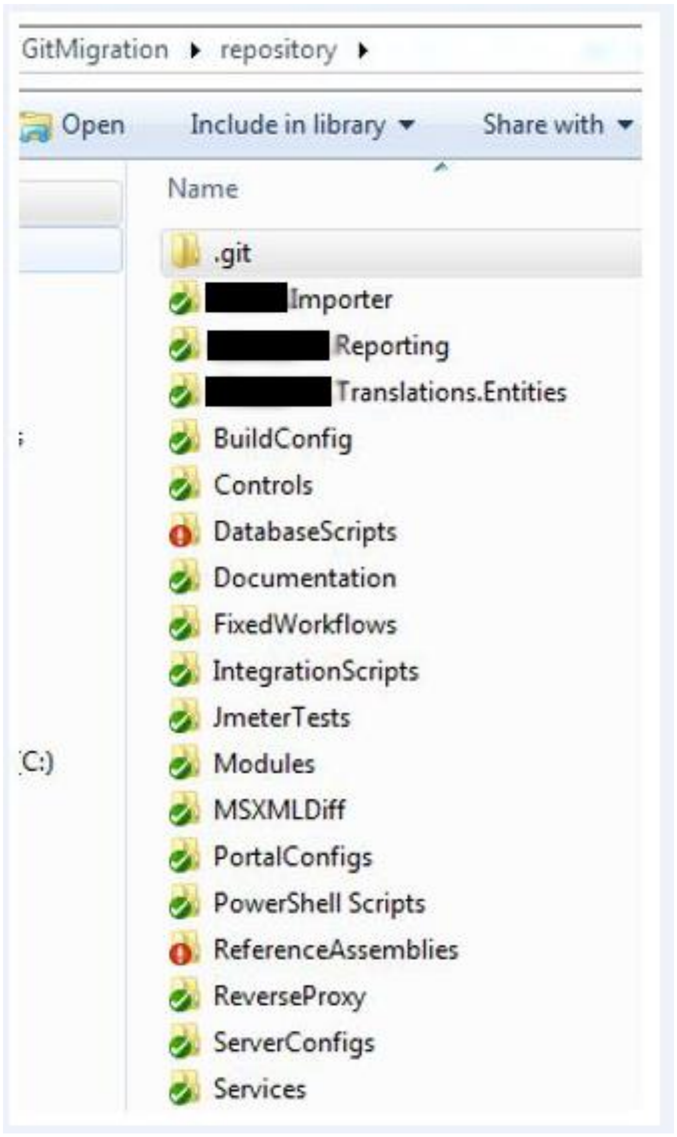
As mentioned above we tried this process on a windows machine but we had difficulties due to some branches having spaces in their names, this in turn cost us some hours of trial and error.

When we ran this command on Ubuntu we didn't have any difficulties and the process was finished in about 6 hours after which we had a Git repository in the local computer.



authors
repository
authors
openvpn-as-2.0.7-Ubuntu12.amd_64.deb

Figure 5. Git migration folder contents



.git
[redacted]Importer
[redacted]Reporting
[redacted]Translations.Entities
BuildConfig
Controls
DatabaseScripts
Documentation
FixedWorkflows
IntegrationScripts
JmeterTests
Modules
MSXMLDiff
PortalConfigs
PowerShell Scripts
ReferenceAssemblies
ReverseProxy
ServerConfigs
Services

Figure 6. Git migration repository contents

At first our local Git repository had all branches as remote SVN branches. The process is designed this way to ensure one-way synchronization.

We had to run the following script to convert the SVN branches into local Git branches:

```
java -Dfile.encoding=utf-8 -jar ~/svn-migration-scripts.jar clean-git –force
```

2.7.6 Synchronizing the local Git repository when the SVN repository changes

Despite there was an SVN freeze we noticed that there were about 70 commits missing in our local Git repository. This were not new changes but rather some commits that for some reason didn't get through the conversion process.

In this case we decided to synchronize the SVN repository with our local Git one.

First we fetch the SVN commits missing using the command: `git svn` fetch and after we integrated the commits into our local Git repository with the command: `java -Dfile.encoding=utf-8 -jar ~/svn-migration-scripts.jar sync-rebase`.

There was the option of "Merging" instead of "Rebasing" the commits. The difference between the two is that Merging integrates the commits without its history so the sequential order of what committed was before or after what commit is lost. Rebasing integrates the commits in a sequential order so the history is preserved.

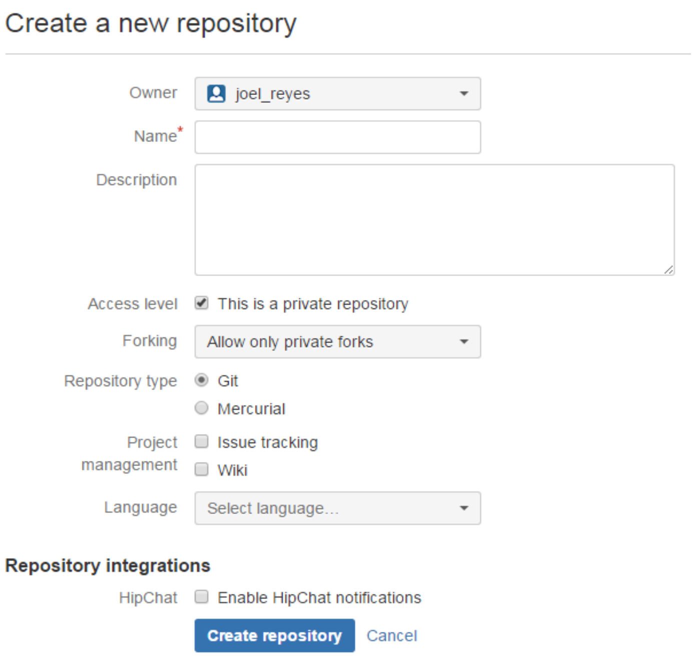
At last we re-ran the command: `java -Dfile.encoding=utf-8 -jar ~/svn-migration-scripts.jar clean-git –force` to delete the obsolete remote SVN branches and tags that could be still on the SVN repository.

2.7.7 Sharing the Git repository with your developers via Bitbucket

Now that we had a Git repository with our project on our local machine we created a Bitbucket repository where we were going to push our local one.

Creating a Bitbucket repository was really easy and we had the options between creating a Git repository or a Mercurial one, making it private or public one and selecting the programming language.

We created a private Git repository with C# programming language.



Create a new repository

Owner	joel_reyes
Name*	
Description	
Access level	[✓ This is a private repository]
Forking	Allow only private forks
Repository type	[✓ Git] Mercurial
Project management	Issue tracking, Wiki
Language	Select language…

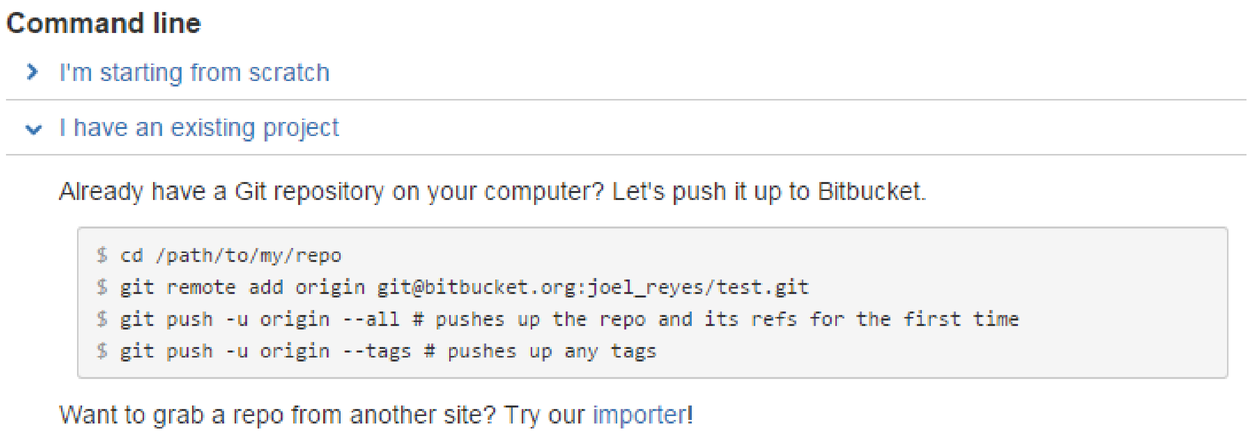
Repository integrations

HipChat	Enable HipChat notifications

Figure 7. Bitbucket new repository creation menu

After creating the repository there was an option between starting the repository from scratch or starting with an existing repository.

Since we had our converted Git repository in our local machine we selected the latter option.



Command line

Already have a Git repository on your computer? Let's push it up to Bitbucket.

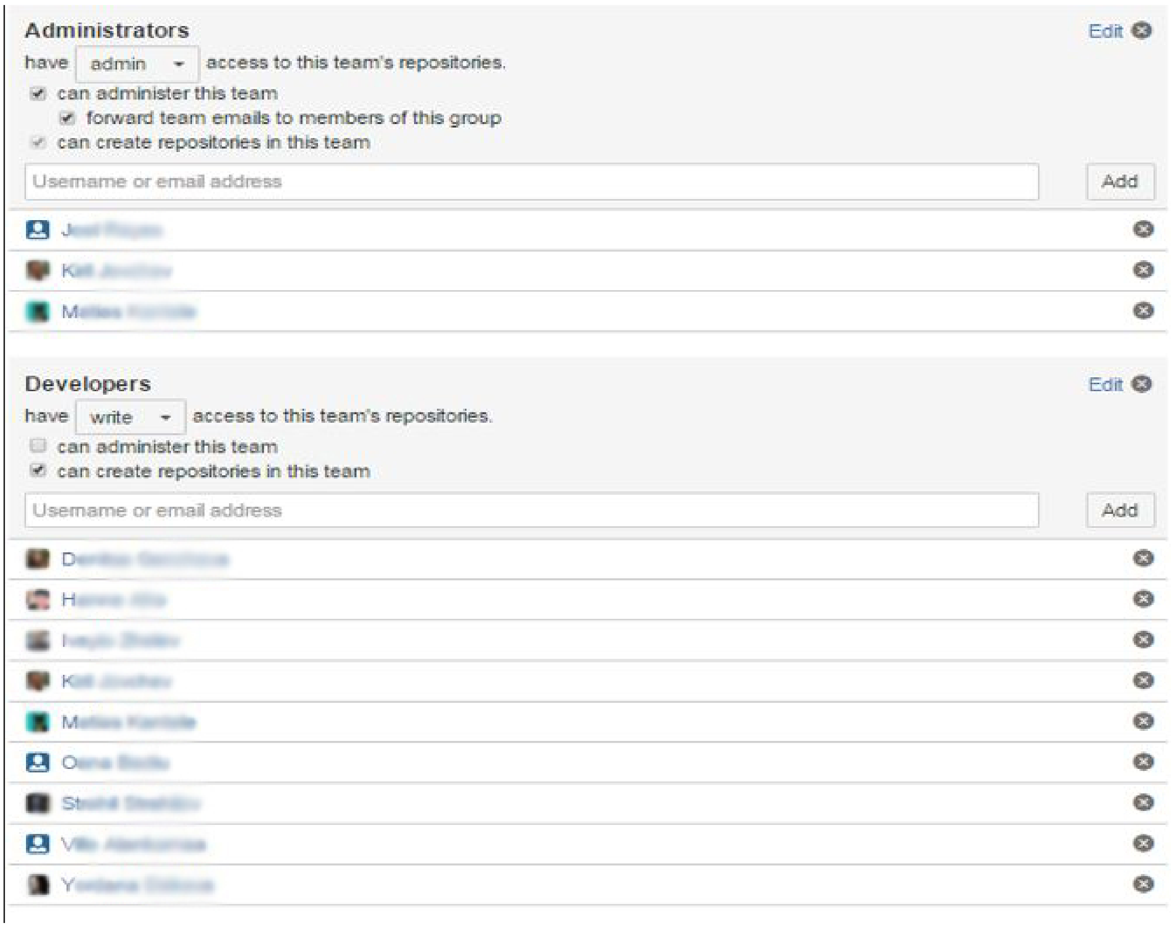
```
$ cd /path/to/my/repo
$ git remote add origin git@bitbucket.org:joel_reyes/test.git
$ git push -u origin --all # pushes up the repo and its refs for the first time
$ git push -u origin --tags # pushes up any tags
```

Want to grab a repo from another site? Try our importer!

Figure 8. Git repository migration to Bitbucket instructions

In order to push our local Git repository to Bitbucket we used the Git Bash tool to navigate to our repository, add the http address of our Bitbucket repository as origin with the command: git remote add origin <http address> and use the git push –u origin -- all,and git push –u origin —-tags, to push the repository and its tags for the first time.

After we had our repository on Bitbucket every developer created a Bitbucket account so that we could add them to our private Git repository and give them access to it and share the http address with them.


Administrators Edit

have admin access to this team's repositories.

- ✓ can administer this team
  - ✓ forward team emails to members of this group
- ✓ can create repositories in this team

Username or email address — Add

- [redacted]
- [redacted]
- [redacted]

Developers Edit

have write access to this team's repositories.

- can administer this team
- ✓ can create repositories in this team

Username or email address — Add

- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]
- [redacted]

Figure 9. Git migration repository contents

2.7.8 Migrating to Git

After the Bitbucket contained the local Git repository there was no need to synchronize it with the SVN one therefore the only steps left was to deactivate the SVN account and start committing to Git.

There was also a small Git training for all developers involved in the project.

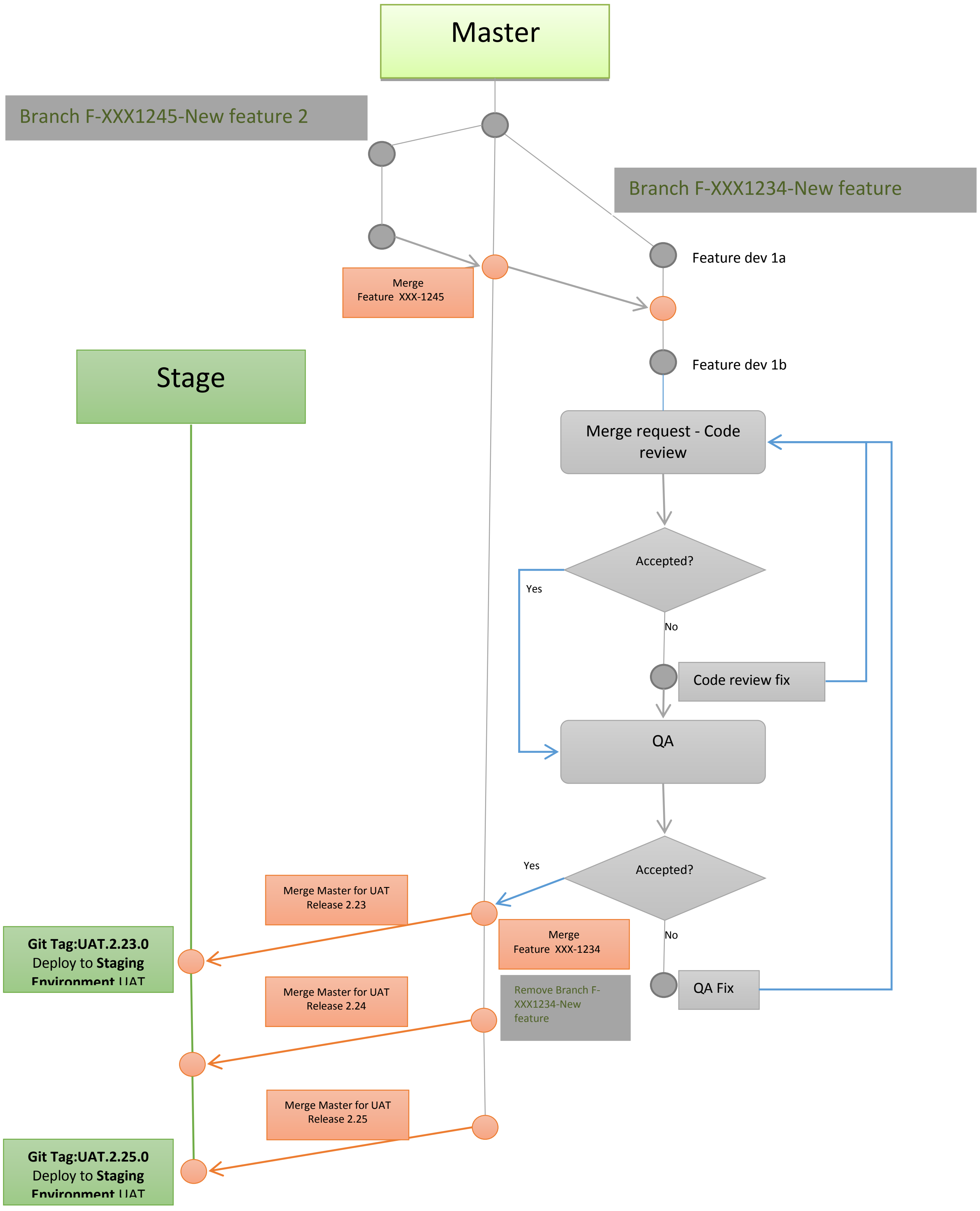
2.8 Git Workflow

After working with SVN for several years some working habits were kept after the migration to Git that needed to be change because of differences in the way SVN and Git are designed.

One such habit was to use the trunk branch in SVN as development branch or in case of Git the branch Master as development branch, this was later change to using Master branch as the production branch.

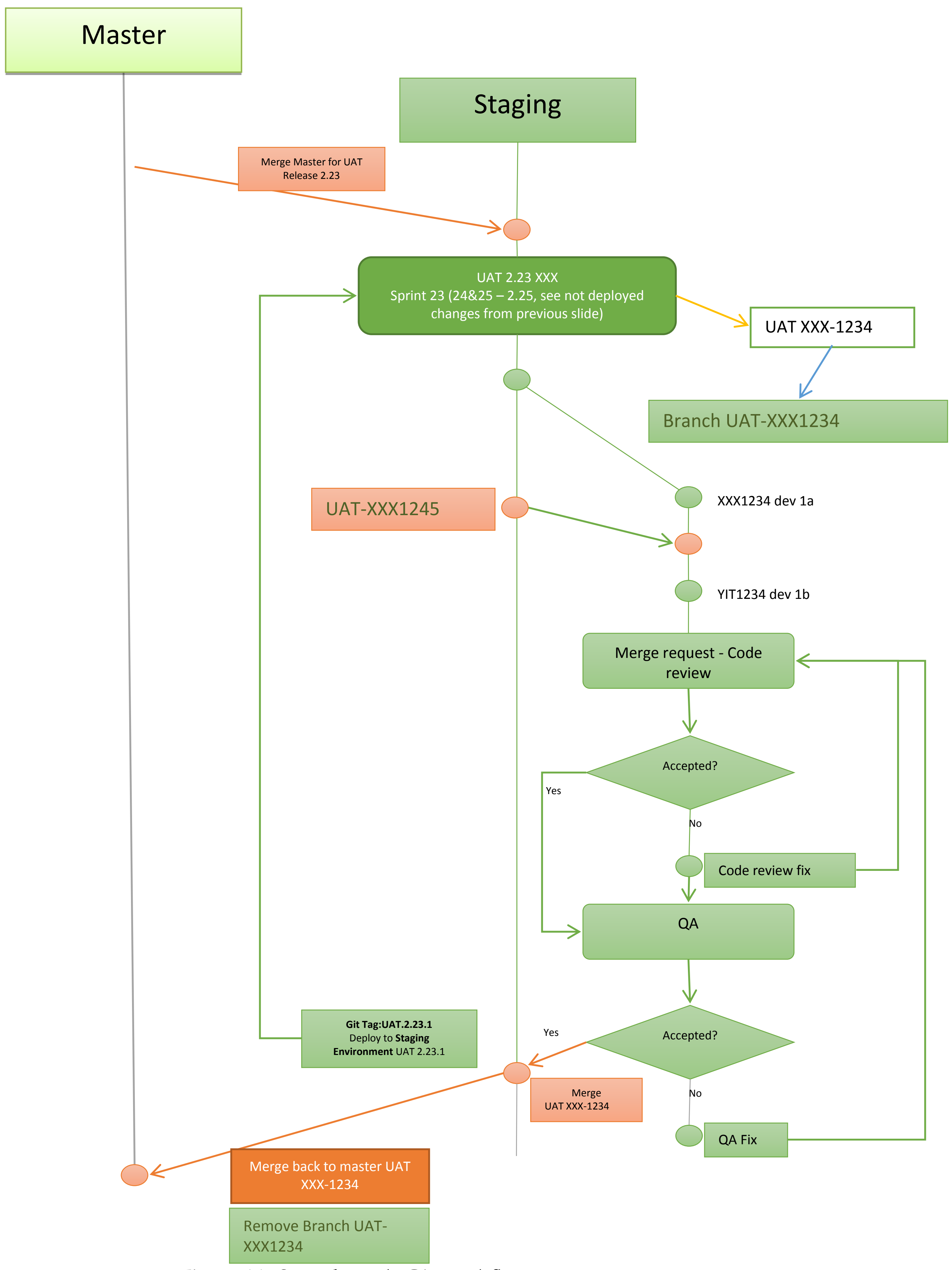
At first the plan was to direct the merging workflow from development to production, the general idea was that new feature branches would be created from the development branch then merged into stage and lastly into production. (Kiril Jovchev 15 May 2015)

The following diagrams described the original plan for the Git workflow:



Master
Branch F-XXX1245-New feature 2
Branch F-XXX1234-New feature
Feature dev 1a
Merge Feature XXX-1245
Feature dev 1b
Stage
Merge request - Code review
Accepted?
Yes
No
Code review fix
QA
Yes
Accepted?
No
Merge Master for UAT Release 2.23
Git Tag:UAT.2.23.0 Deploy to Staging Environment UAT
Merge Feature XXX-1234
QA Fix
Remove Branch F-XXX1234-New feature
Merge Master for UAT Release 2.24
Merge Master for UAT Release 2.25
Git Tag:UAT.2.25.0 Deploy to Staging Environment UAT

Figure 10. Git workflow from Master branch to Stage branch



Master
Staging
Merge Master for UAT Release 2.23
UAT 2.23 XXX Sprint 23 (24&25 – 2.25, see not deployed changes from previous slide)
UAT XXX-1234
Branch UAT-XXX1234
UAT-XXX1245
XXX1234 dev 1a
YIT1234 dev 1b
Merge request - Code review
Accepted?
Yes
No
Code review fix
QA
Git Tag:UAT.2.23.1 Deploy to Staging Environment UAT 2.23.1
Yes
Accepted?
No
Merge UAT XXX-1234
QA Fix
Merge back to master UAT XXX-1234
Remove Branch UAT-XXX1234

Figure 11. Stage branch Git workflow

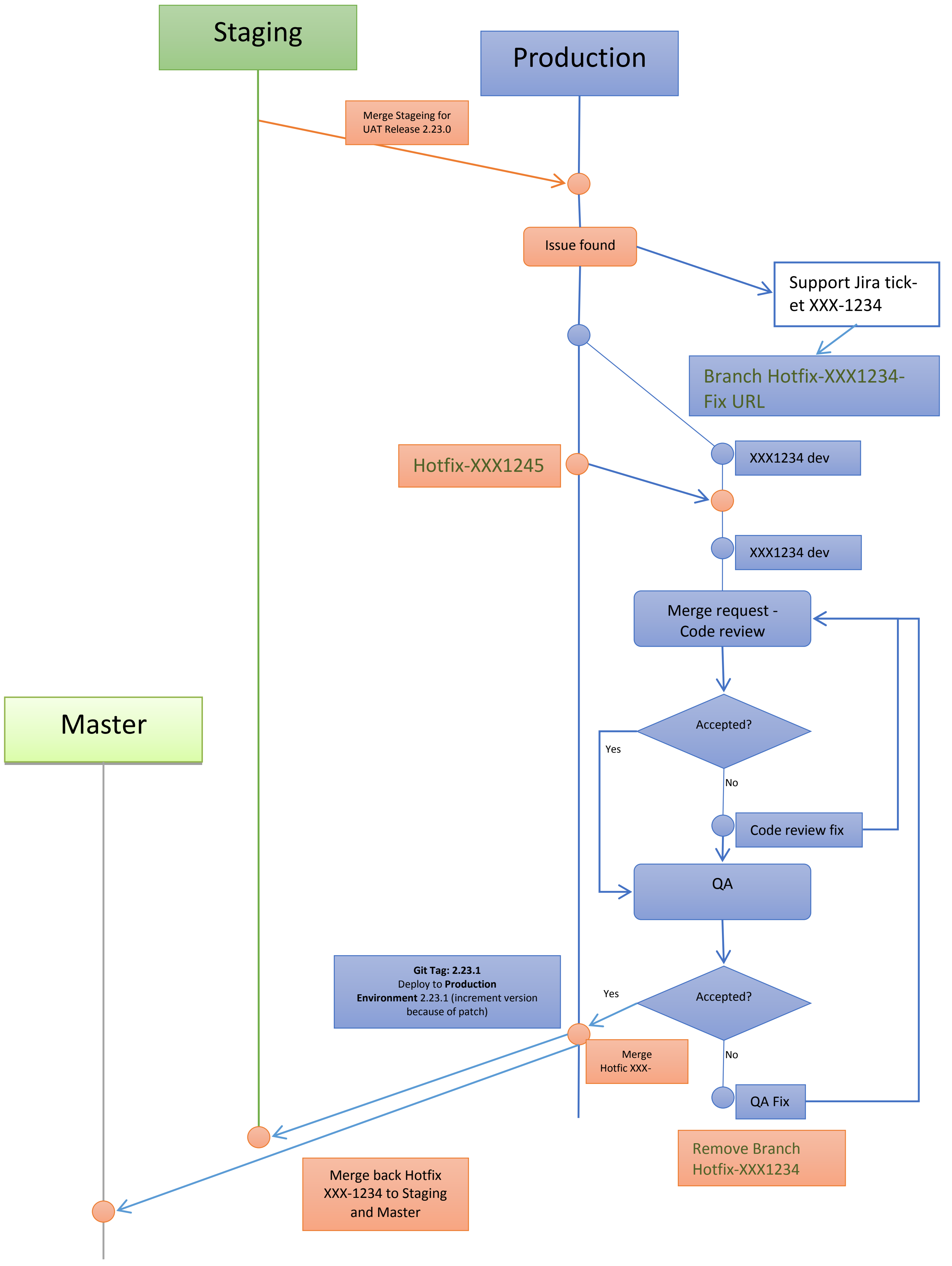

Staging
Production
Merge Stageing for UAT Release 2.23.0
Issue found
Support Jira ticket XXX-1234
Branch Hotfix-XXX1234-Fix URL
Hotfix-XXX1245
XXX1234 dev
XXX1234 dev
Merge request - Code review
Master
Accepted?
Yes
No
Code review fix
QA
Git Tag: 2.23.1 Deploy to Production Environment 2.23.1 (increment version because of patch)
Yes
Accepted?
No
Merge Hotfic XXX-
QA Fix
Remove Branch Hotfix-XXX1234
Merge back Hotfix XXX-1234 to Staging and Master

Figure 12. Git workflow from Stage branch to Production branch

2.9 Git workflow guidelines (original plan)

New feature development:

- Master is the development branch
- Feature branches are branched from master
- After the feature is developed on the feature branch it is submitted for a code review
- If the code review is not passed it has to be fixed
- When the code review is passed it is submitted for QA testing
- If QA testing is not passed then it needs to be fixed
- If QA testing is accepted the feature branch is merged into master branch (development)
- development is merged into stage

- In stage there is user acceptance testing
- If an issue is found a branch is created from staging for fixing the issue
- The issue is resolved in that branch
- After the issue is fixed on the branch it is submitted for a code review
- . If the code review is not passed it has to be fixed
- When the code review is passed it is submitted for QA testing
- If QA testing is not passed then it needs to be fixed
- If QA testing is accepted the feature branch is merged into staging branch
- staging is merged back into master
- After staging branch has passed the user acceptance testing it is merged into production

Hot fixes:

- If there is a need to perform hot fixes a branch is created from production
- After the hot fix is done it is submitted for a code review
- If the code review is not passed it has to be fixed
- When the code review is passed it is submitted for QA testing
- If QA testing is not passed then it needs to be fixed
- If QA testing is accepted the branch is merged into production
- production is merged into staging and master

2.10 Issues with original plan

After the creation of the above described plan, many of the same issues previously encounter when working with SVN workflow started to re-appear.

It was noticed that there were many difficulties when working from development towards the production branch.

The main issue was that in many cases the whole branch could not be merged into the next branch in the workflow but rather only some features were ready or accepted to be merged.

Merging only some but not all features into the next branch in the workflow very likely breaks other features dependent on the features not merged.

In order to solve this issue the workflow was reverted from production to development, and instead of merging new branches with fixes to production, the branches themselves become the production branch and the older production branch become history. (Kiril Jovchev 15 May 2015)

A new plan and workflow guidelines were designed.

2.11 New and improved Git workflow guidelines

When working with Git we will use three main branches, namely: Production, Development, and Release candidate (Stage).

The following guidelines must be followed when working with branches:

- Production will be the master branch
- Development and Release candidate will be branched from production
- Whenever a new feature will be developed or a fix will be made a feature branch will be created from production
- After the feature is developed or a Fix is done the feature branch will be merged into development branch and a UAT functional test will be performed
- If the test is not passed then work will continue to be done in the feature branch and merge again into Development until the functional test is passed
- After the test is passed the Feature branch will be merged into the Release candidate branch and a Regression testing will be performed
- If the Regression testing is not passed this will have to be fixed in the feature branch again and merge back into the Release candidate branch and development branch

- After the regression testing is passed production will be branched into a branch named as production plus the date and hour of creation, this branch will be used as history in case we need to go back to a previous state of the production branch and the original production branch will be deleted
- After the original production branch is deleted a new production branch will be created from the Release candidate branch
- The feature branch is deleted

When doing hot fixes the following guidelines must be followed:

- A hot fix branch will be created from production for each hot fix to be made
- The hot fix branch will be merged into Release Candidate for regression testing
- If the Regression testing is not passed this will have to be fixed in the hot fix branch again and merge back into the Release candidate branch
- After the regression testing is passed a copy of the production branch will be created and named as production plus the date and hour of creation, this branch will be used as history in case we need to go back to a previous state of the production branch and the original production branch will be deleted
- After the original production branch is deleted a new production branch will be created from the Release candidate branch
- After the hot fix is in production, the hot fix branch will be merged into the development branch
- The hot fix branch is deleted

2.12 Discussion

In this section I discuss the advantages of centralized vs distributed version control system, how our developers team beneficiate from switching from SVN to Git, I also expose four different popular Git workflows and how we evolved in our team from popular Git workflow to our own unique workflow that works the best for us.

2.12.1 Centralized vs Distributed Version Control System

Centralized version control systems like SVN follow a model in which there is a single “central” repository to which all developers commit changes to.

Other developers can then “pull” the latest version of the central repository that will contain all changesets committed by all developers.

Although the latest version of the repository can be pulled to the developers’ hard drive, the full history of the repository will remain in the central repository. (Giancarlo Lionetti February 14, 2012)

In the case of distributed version control systems, the developer can clone a copy of the repository containing its full history. (Giancarlo Lionetti February 14, 2012)

The advantages of having a distributed version control system are:

- The system only needs to access the hard drive and not the remote repository to perform actions which makes them very fast to perform
- Changes can be committed to the local copy of the repository first and pushed to the remote repository when ready
- There is no need for internet connection when committing changes to the local repository which makes developers not to be forced to pushed changes directly to the remote repository
- Changes committed to local repository can be shared with a some developers for feedback before showing them to the whole team

The disadvantages of having a distributed version control system are:

- If there repository has many large files that cannot be compressed easily having many different versions of such files would take a lot of space
- If the repository has a long history it may consume an unacceptable time to download it and too much space.

2.12.2 Experiences in our developers team

In the case of our company’s customer, the project has more than 5000 thousand revisions which means that we encounter challenges at the time of cloning since it may take a long time and it occupy a lot of space.

Nevertheless the advantages are greater than the downsides since it only takes too much time the first time the repository is downloaded but afterwards the process of

committing and performing actions become so much faster because the repository is now on the local environment.
The fact that the repository takes a lot of space is a fact that has been decided to be acceptable and manageable by spending on disk space resources.

After allocating enough disk space on local environments and allocating enough time for cloning the repository locally, the team is satisfied with the advantages of been able to commit to the local repository before pushing to the server-side one and the faster time to perform actions locally.

2.12.3 Popular and suggested Git workflows

Atlassian, a company that produces IT services that we as a company have greatly utilized, suggests in an article four different Git workflows that our developers' team took into consideration:

- Centralized workflow
- Feature Branch workflow
- Gitflow workflow
- Fork workflow

Our team considered these 4 workflows and developed rules and behaviors based on the Feature branch workflow that evolved into a unique workflow. Here I expose the four considered workflows and our evolution to a unique workflow that works the best for us.

2.12.4 Centralized Workflow

The normal workflow in Git starts by initializing a repository on the server-side. Each developer now can clone the repository to their local environment and work independently from each other by committing changes only to their local repository.

When a developer finishes his feature he can commit to the server-side repository by “pushing” his local commitments.

Now that a developer has integrated his changes into the server-side repository, if a second developer tries to push his changes to it he won’t be able to do it unless he first pulls the updated version of the server-side repository to his local repository.

If there are conflicts between the first and second developer then the second developer must resolve the conflicts in his own local environment before he can push into the server-side.

2.12.5 Feature Branch Workflow

The idea of the feature branch workflow is that there must be a branch created from master for every single feature in which the feature will be developed in.

Every developer must create a new branch (“feature branch”) and make changes concerning a feature only to this branch. The developer can work independently on this branch without the need to push it to the server-side repository before the feature is complete, but instead the developer can commit to his local repository.

When a feature is ready the developer can make a pull request that lets the team know that the feature is ready. The feature is examined and the developer in charge of the master branch may request some changes done to the feature branch before it is merge into master.

If everything is ok then the feature is merged into master, while other developers are working in the same way for new features.

2.12.6 Gitflow Workflow

The main idea behind Gitflow workflow is to have a branch that contains all of the release versions as a history and a branch from which feature branches can be created from.

In contrast with other workflows feature branches are created from the “develop” branch while the “master” branch is the one that will contain the releases history.

The develop branch is first created from the master branch. Feature branches are created from and merged back into develop. When develop has enough features for a release, a release branch is created from develop to use for production.

The release branch is then merged back into develop and into master to mark a release version.

2.12.7 Fork workflow

The fork workflow is fundamentally different from the above mentioned workflows in that instead of each developer cloning the main repository on their local environments, each developer get their own server side repository which it is later cloned to their local environments.

Developers work on their local environments and commit and push to their own server-side repositories.

When and update is ready to be integrated into the project, the official repositories’ maintainer pulls from the developer’s server-side repository and merges into his own local environment, checks that everything is ok and pushes to the official repository.

After changes are integrated into the main repository developers can pull from it to their local can be shared changes.

One important thing to take into account is that although issues are been isolated into different server-side repositories it is still important to isolate different features into different branches.

2.12.8 Developers Team Experience

The experience of our developers' team starts when the customer's source code was in SVN. The workflow used at this point in time was similar to the centralized workflow described above.

Features were worked on development branch, then when development contained all features for a sprint development branch was merged into staging branch where tests were performed and later in theory when features would passed the test, staging branch would be merged into production.

Several problems where discovered with this approach:

1. Not all the features passed the test which left development branch containing some features ready for implementation and some features not ready.
2. Development branch could not be merged into production as a whole but the features ready for implementation had to be cherry picked and merged into production.
3. Cherry picking features cannot assure the stability of production because some features may depend on others from proper functioning

One solution was to force all of the features to pass the customer's test by starting with higher levels of the quality of definitions of customer's requirements, so that by adhering to the well-defined requirements the customer would be compelled to accept the features produced, this in turn would allow to merge the entire development branch into production.

After the migration to Git, another proposed solution was to start implementing the feature branch workflow that meant that feature branches would be spawned from the production branch and later merged into development and staging branch for testing.

This solved part of the problem because features developed on feature branches would only merge into development branch after the feature is finished, meaning that the development branch would always be ready to merge into staging.

The problem that persisted was again that some features would pass the test and others would not pass the test making it impossible to merge the whole development branch into production.

This problem was solved by merging and testing one feature branch at a time into master and then into a release candidate branch for regression testing and after all testing is performed, the branch that has passed all the testing would become the new production branch instead of been merge into the already existing production branch.

This is the current model of workflow which has proven to be effective and stable.

2.12.9 Git benefits according to a developer on the project

"The biggest benefit of the new workflow is that the entire process is now much more transparent and controlled than earlier.

This is mainly because we are now using pull requests - no code will end up in the master branch before someone else has reviewed it. There is no doubt that this has positive effects on the code quality, because all the developers have to critically evaluate their own and each other's code, and the most obvious bugs and problems are likely to be caught early on. This is also a very good way to learn and discuss the good coding practices with the other developers.

In the old workflow, everything was directly committed to the master branch and some code review was maybe done when features were cherry-picked for the deployment probably not always even then." (Project's developer, 2015)

3 Part 2: Migration from onsite hosting to the cloud

3.1 Background

For the last 4 years the company has been hosting their websites' applications on-site. The architecture started with:

- 2 Active Directory Servers
- 4 application servers (2 for staging environment and 2 for production environment)
- 1 SQL cluster

Active directories and applications servers were virtualized on top of VMware where the SQL cluster runs directly on the hardware. (Kiril Jovchev 15 May 2015)

VMware was running on our server cluster, there were 4 hardware machines.

The server environment supported 14 sites from which 12 were corporate sites and 2 were business sites.

In the last 4 years since the project started the number of sites increased to 28 from which 15 were business sites. This expansion put extra load on the servers and not only because the mere increase in the number of sites but also because an increase in the percentage of business sites which consume 3 times more resources than corporate sites. (Avaus Consulting Oy, 2015)

The increased demand for resources on the servers represented a big problem since there was a continuous lack of disk space that had to be expanded and high CPU usage. (Kiril Jovchev 15 May 2015)

The company decided to migrate to the cloud due to the fact that the severs' life time had expired already. The company decided to run a one year long analysis to decide which cloud hosting service to use. (Kiril Jovchev 15 May 2015)

The life span of the servers was passed (over 3 years).

There were 3 major candidates with their advantages:

- Amazon Web Services – The advantage was that they are the market leader.
- Azure- The advantage was that they had an enterprise agreement that would make the hosting very cheap.
- Enterprise Cloud – The advantage was that they felt more secure if their websites were hosted in Finland.

The decision was taken in favor of a enterprise cloud service based in Finland.

The agreement was that the enterprise cloud service company would copy the old environment and create a new environment with the same number of servers according to the architectural specification and that Avaus consulting was in charge of the migration of the sites from the current servers to the new servers. (Kiril Jovchev 15 May 2015)

The architecture is designed so that for each of the production and staging environments there is a load balancer that keeps the servers belonging to the same environment in sync. (Avaus Consulting Oy, 2014)

Each server contains multiple websites which have the same web root but different configuration files and virtual directories. Each of the sites belonging to the same environment connect to the same database. (Avaus Consulting Oy, 2014)

3.2 Migration Tasks

In order to successfully migrate the company's websites to the new servers the following tasks must be performed:

- Installing the necessary IIS modules
- Migrating all necessary files from the old servers to the new ones
- Exporting all sites
- Creating virtual directories

- Directing logs to new locations
- Configure FTP server
- DNS Resolution
- Testing
- Migration

3.3 Setting up the necessary IIS modules

IIS comes with a set of core functionalities and the option of installing additional sets of functionalities called modules.

In order to serve or websites successfully I needed to install the following IIS modules which are required to run a Sitefinity application (Telerik, 2015):

1. **Dynamic Content Compression**: Enables dynamic compression which makes for a more efficient use of bandwidth, although it may increase the load on CPU. (Microsoft. TechNet. 2015)

2. **Request Filtering:** Filters incoming requests using defined sets of rules. (Microsoft. TechNet. 2015)

3. **Basic Authentication:** It provides a basic security level by requiring a user ID and a password to use as credentials. (Microsoft. TechNet. 2015)

4. **IP and Domain Restrictions:** It makes IIS restrict the access to defined IPs and/or domains. (Microsoft. TechNet. 2015)

5. **Windows Authentication:** It provides a high level of security by transferring user authentication over a network as a Kerberos ticket. (Microsoft. TechNet. 2015)

6. **.NET Extensibility 4.5:** Infrastructure that enables developers to modify and add functionality to the Web server and support ASP.NET (Microsoft. IIS. 2015)

7. **Application Initialization:** Allows for the performance of Web application initialization tasks previous to serving pages. (Microsoft. IIS. 2015)

8. **ASP.NET 4.5:** Supports communication for server applications that were created with IIS 8.x and ASP.NET 4.5. (Microsoft. IIS. 2015)

9. **ISAPI Extensions:** Supports dynamic Web content development. (Microsoft. IIS. 2015)

10. **ISAPI Filter:** It allows for the modification and addition of IIS functionality by reviewing and processing each request sent to IIS. (Microsoft. IIS. 2015)

11. **Web Sockets:** Provides communication channels for server applications created with IIS 8.x and ASP.NET 4.5. (Microsoft. IIS. 2015)

12. **HTTP Activation 4.5:** Allows WFC services to run on IIS/ASP.net over HTTP (Microsoft. MSDN. 2015)

13. **URL Rewrite:** Allows the creation of rules to define URL rewriting behavior like redirections and custom responses. (Microsoft. IIS. 2015)

14. **ARR Module:** It forwards HTTP requests to content servers (Microsoft. IIS. 2015)

Most above mentioned modules with the exception of the URL rewrite module can be installed using the following powershell code (Microsoft. MSDN. 2013):

```
add-windowsfeature web-webserver, Web-Dyn-Compression, Web-ODBC-Logging, Web-Filtering, Web-Basic-Auth, Web-IP-Security, Web-Windows-Auth, Web-Net-Ext45, Web-AppInit, Web-Asp-Net45, Web-ISAPI-Ext, Web-ISAPI-Filter, Web-WebSockets, RSAT-SMTP, SMTP-SERVER, Web-WMI, NET-WCF-HTTP-Activation45
```

The url Rewrite module has to be downloaded from: http://www.iis.net/downloads/microsoft/url-rewrite and installed. (Microsoft. IIS. 2015)

In addition to the URL Rewrite module, The URL Rewrite Extensibility Samples which can be downloaded from http://www.iis.net/learn/extensions/url-rewrite-module/using-custom-rewrite-providers-with-url-rewrite-module has to be installed. (Microsoft. IIS. 2015)

3.4 Migrating all necessary files from the old servers to the new ones

The file system was improved and designed to avoid running out of space which is a constant problem in the current environments. (Kiril Jovchev 15 May 2015)

The new servers' architecture was designed so that there would be a better organization for the important files and folders needed to run all aspects of the websites and for improved scalability of the resources. (Kiril Jovchev 15 May 2015)

The important files and folders of the websites are:

- **Web root folder:** This is the folder containing the application to which the all company's sites direct on IIS server (all of these sites use the same web root folder but different configuration files).
- **Configuration files:** Contains the configuration files for each of the different websites. There is a different configuration folder for each of all sites.
- **WAS folder:** Contains web services application that support the website, like POI and image recognition services
- **Integrations site:** The importer site is a duplicate of the web root folder against which the imports are ran.
- **Logs:** The logs folder contains the IIS logs, Sitefinity logs and HTTP error logs.
- **ASP.NET temporary files**
- **Shared files**
- **Importer dump folders:** These folders receives the XML files sent by the importer
- **File System:** This folder contains the images used on the site
- **Scripts folder:** Folder containing various utility scripts like scripts to start and stop all sites, deployment scripts, monitoring scripts etc.
- **Reverse proxy sites folder:** containing the web.config of 4 of the sites which are reverse proxy ones.

The old servers consisted of 3 drives. The C: drive was used for the operating system and the D: drive contained the application root folder, configuration files, WAS folder Logs and share folder. (Avaus Consulting Oy, 2014)

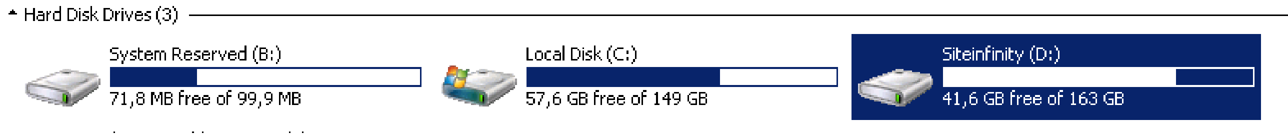


Figure 13. Old servers' drives

In the old environment the D: drive contained most of the application folders. The new environment was designed to compartmentalize the different files needed to run the project.

The new servers contain 6 drives each:

- C: drive for the Operating system
- D: drive to store the application
- E: drive for different kinds of logs
- F: drive for temporary ASP.NET files
- G: drive for temporary deployment files
- H: drive for DFS

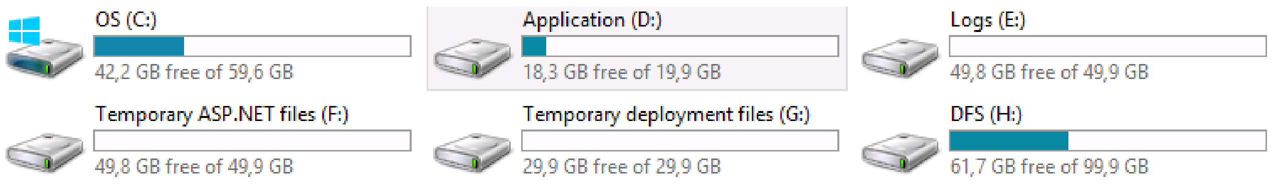


Figure 14. New servers' drives

The Web root folder, Configuration files, WAS folder, Importer site where moved from the old server to the new one under a folder named IIS on the D: drive.

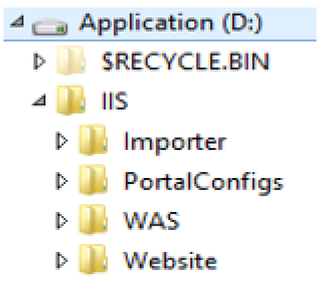


Figure 15. D drive contents

All the files that were to be shared between the two nodes were moved to the H: drive including:

XML Importer dump folder, the storage file system and different folders containing PowerShell scripts and services.

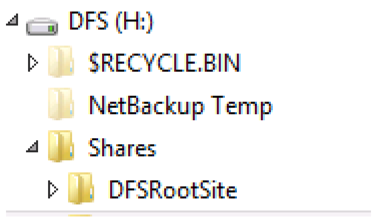


Figure 16. DFS drive contents

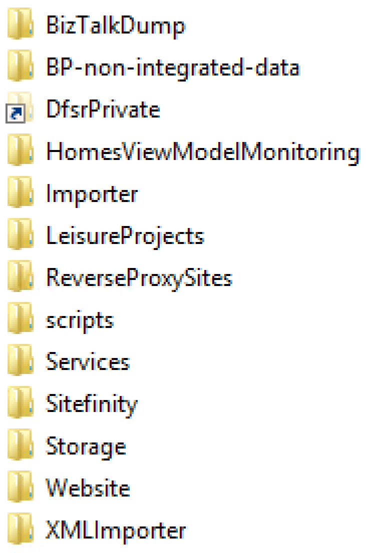


Figure 17. DFS drive contents

There were 4 folders created on the E: drive to save different kinds of logs, namely:

- HTTP Error logs

- IIS logs
- Importer logs
- Sitefinity logs

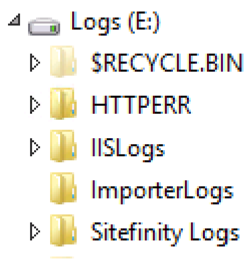
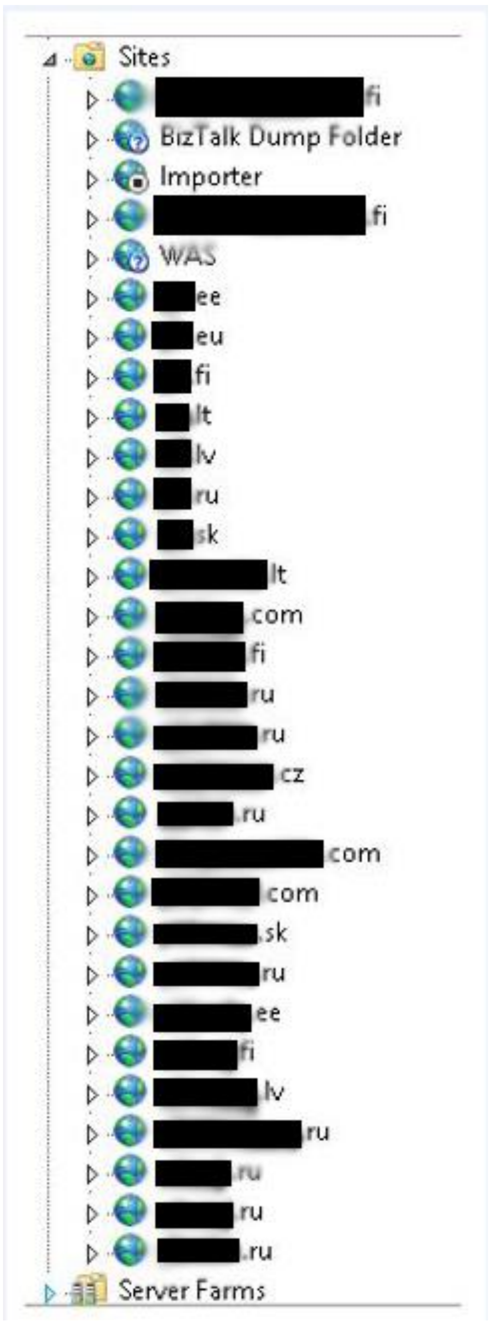


Figure 18. Logs drive contents

3.5 Exporting all sites

After having the necessary files and folders in place, the next step was to create all the sites on the IIS. To make this task simple and effective the application pools and sites nodes where copied from the applicationhostconfig file on the old server to the one in the new server.



```
Sites
  [redacted]fi
  BizTalk Dump Folder
  Importer
  [redacted]fi
  WAS
  [redacted]ee
  [redacted]eu
  [redacted]fi
  [redacted]lt
  [redacted]lv
  [redacted]ru
  [redacted]sk
  [redacted]lt
  [redacted].com
  [redacted]fi
  [redacted]ru
  [redacted]ru
  [redacted]cz
  [redacted].ru
  [redacted]com
  [redacted].com
  [redacted].sk
  [redacted]ru
  [redacted]ee
  [redacted]fi
  [redacted]lv
  [redacted]ru
  [redacted]ru
  [redacted]ru
  [redacted].ru
Server Farms
```

Figure 20. Project sites on IIS

After copying the sites to the new servers we had created 30 sites in the new servers' IIS:

- 1 FTP server (BizTalk dump folder)
- 1 site for the importer
- 1 Site to direct to the WAS folder
- 27 Company sites from which 4 where proxy servers

3.6 Creating virtual directories

The application pools and sites' nodes copied from the old servers to the new ones had virtual directories with locations that do not applied any longer in the new server's environment.

New virtual directories' locations needed to be created. (Microsoft 2015)
The virtual directories needed to be created were:

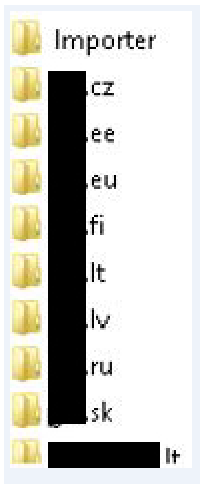

Importer
[redacted].cz
.ee
.eu
.fi
.lt
.lv
.ru
.sk
[redacted] lt

Figure 19. Virtual directories

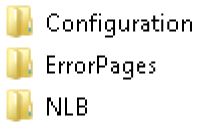

Configuration
ErrorPages
NLB

Figure 20. Virtual directories

- **Sitefinty Configuration Files –** Each site has different Sitefinity configurations, thus we had to create a configuration files folder for each of the different sites to keep them organized. Inside each folder we had 3 different folders to server as virtual directories, namely: Configurations, ErrorPages, and NLB.

The Configuration folder contains all the Sitefinity configuration files needed by each site. (Telerik, 2015)

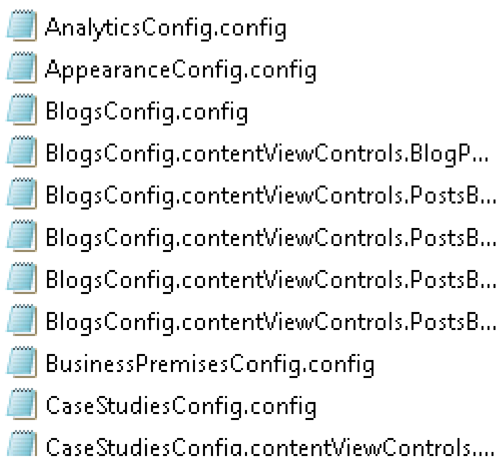

AnalyticsConfig.config
AppearanceConfig.config
BlogsConfig.config
BlogsConfig.contentViewControls.BlogP...
BlogsConfig.contentViewControls.PostsB...
BlogsConfig.contentViewControls.PostsB...
BlogsConfig.contentViewControls.PostsB...
BlogsConfig.contentViewControls.PostsB...
BusinessPremisesConfig.config
CaseStudiesConfig.config
CaseStudiesConfig.contentViewControls....

Figure 21. Sitefinity configuration files

- **Error Pages –** The error pages directory contains custom error pages for 404 and 500 error messages.

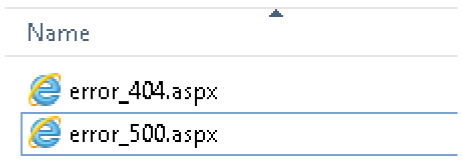
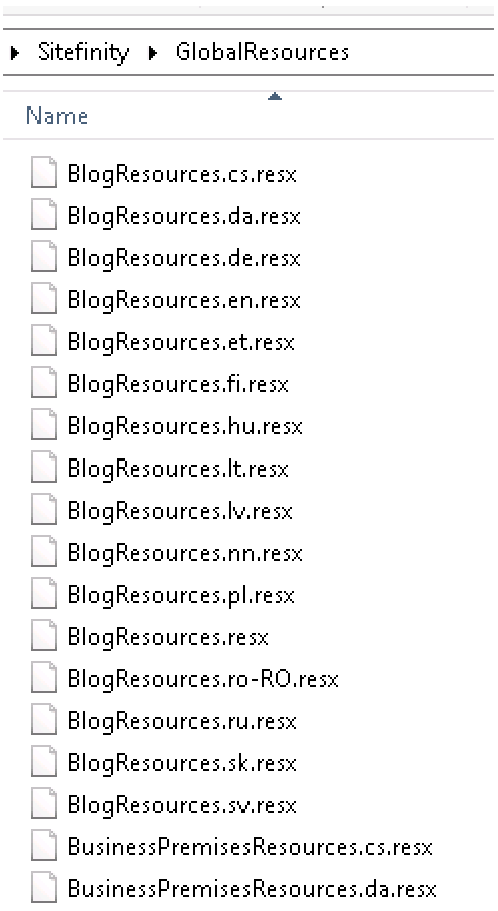

Name
error_404.aspx
error_500.aspx

Figure 22. Error pages files

- **Global Resources –** The global resources directory contains Sitefinity resources used by each and site. In this case global resources are not separated by different folders per site but they are all in the same virtual directory. (Telerik, 2015)

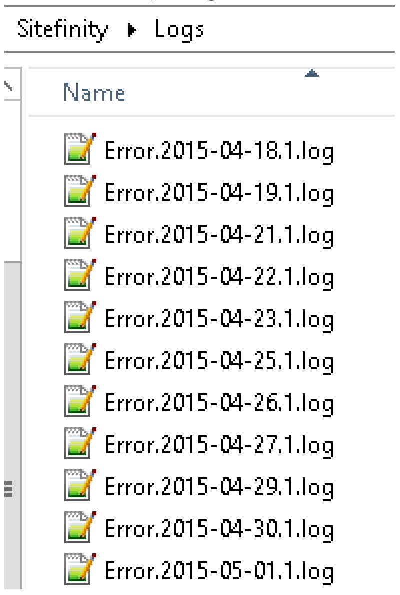

Sitefinity ▸ GlobalResources

Name

BlogResources.cs.resx
BlogResources.da.resx
BlogResources.de.resx
BlogResources.en.resx
BlogResources.et.resx
BlogResources.fi.resx
BlogResources.hu.resx
BlogResources.lt.resx
BlogResources.lv.resx
BlogResources.nn.resx
BlogResources.pl.resx
BlogResources.resx
BlogResources.ro-RO.resx
BlogResources.ru.resx
BlogResources.sk.resx
BlogResources.sv.resx
BusinessPremisesResources.cs.resx
BusinessPremisesResources.da.resx

Figure 23. Global Resources Files

- **Sitefinity Logs –** This folder contains the sitefinity error logs.

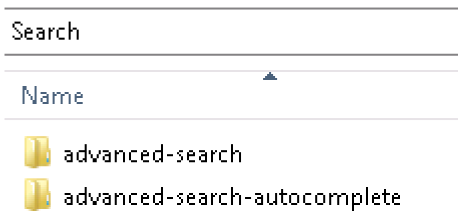

Sitefinity ▸ Logs

Name

Error.2015-04-18.1.log
Error.2015-04-19.1.log
Error.2015-04-21.1.log
Error.2015-04-22.1.log
Error.2015-04-23.1.log
Error.2015-04-25.1.log
Error.2015-04-26.1.log
Error.2015-04-27.1.log
Error.2015-04-29.1.log
Error.2015-04-30.1.log
Error.2015-05-01.1.log

Figure 24. Error log files

- **Search –** Contains Sitefinity search indexes.

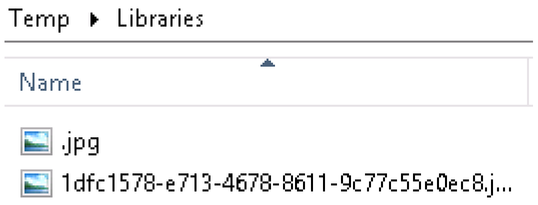

Search

Name

advanced-search
advanced-search-autocomplete

Figure 25. Search folders

- **Sitefinity Temp folder –** Contains Sitefinity temporary files.

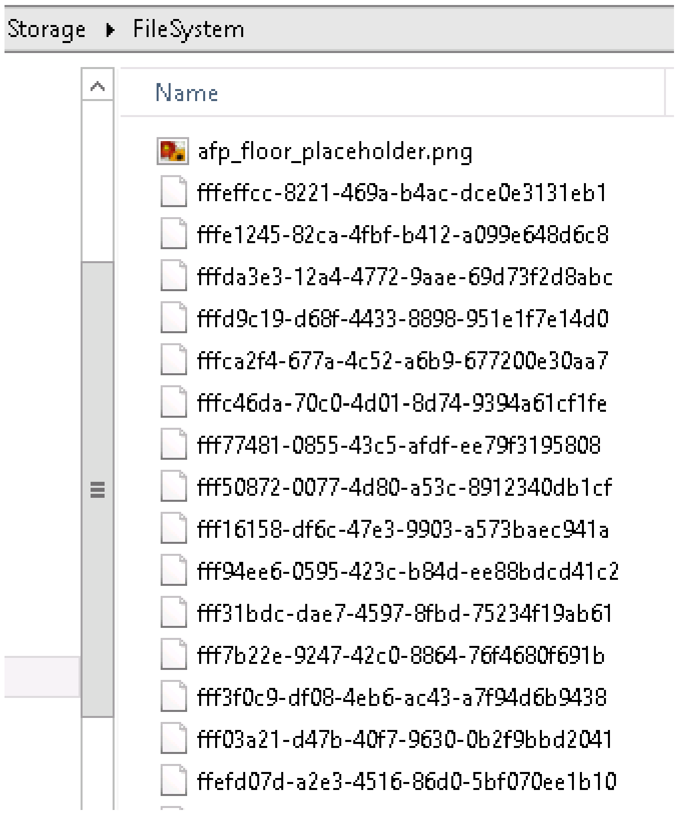

Temp ▸ Libraries

Name

.jpg
1dfc1578-e713-4678-8611-9c77c55e0ec8.j...

Figure 26. Sitefinity temp folder contents

- **Storage / File System –** Contains image files used by Sitefinity.

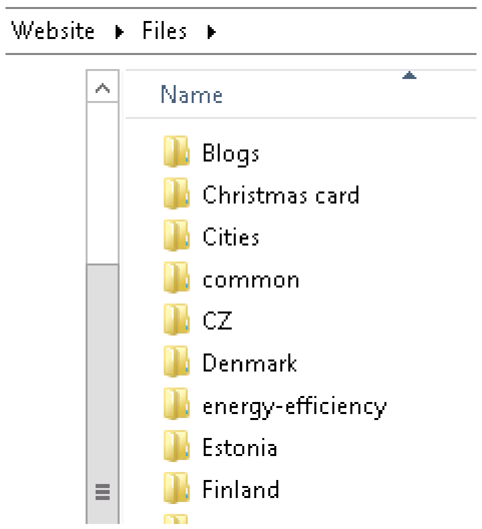

Storage ▸ FileSystem

Name

afp_floor_placeholder.png
fffeffcc-8221-469a-b4ac-dce0e3131eb1
fffe1245-82ca-4fbf-b412-a099e648d6c8
fffda3e3-12a4-4772-9aae-69d73f2d8abc
fffd9c19-d68f-4433-8898-951e1f7e14d0
fffca2f4-677a-4c52-a6b9-677200e30aa7
fffc46da-70c0-4d01-8d74-9394a61cf1fe
fff77481-0855-43c5-afdf-ee79f3195808
fff50872-0077-4d80-a53c-8912340db1cf
fff16158-df6c-47e3-9903-a573baec941a
fff94ee6-0595-423c-b84d-ee88bdcd41c2
fff31bdc-dae7-4597-8fbd-75234f19ab61
fff7b22e-9247-42c0-8864-76f4680f691b
fff3f0c9-df08-4eb6-ac43-a7f94d6b9438
fff03a21-d47b-40f7-9630-0b2f9bbd2041
ffefd07d-a2e3-4516-86d0-5bf070ee1b10

Figure 27. Files system contents

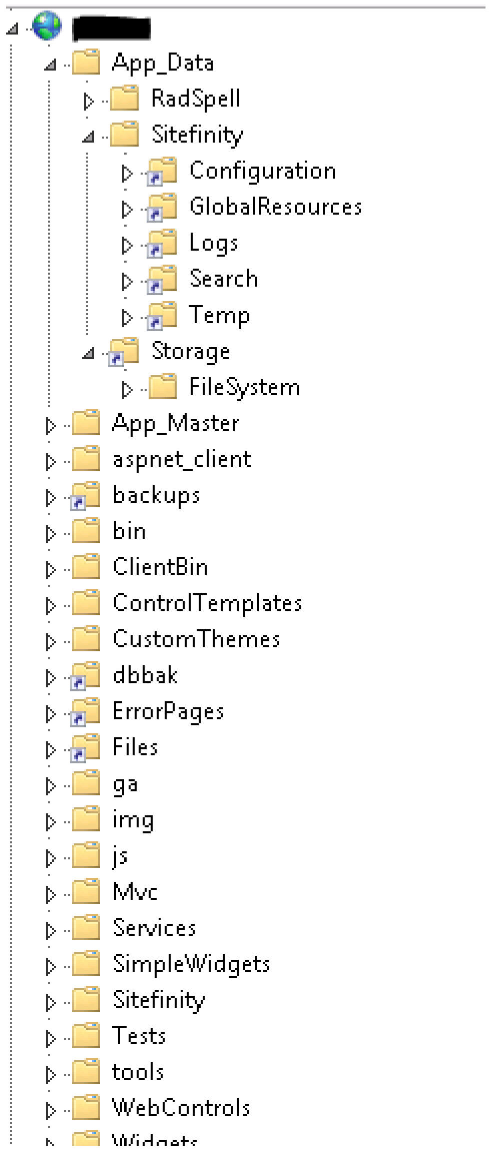
- **Files**



```
Website ▸ Files ▸

Name
Blogs
Christmas card
Cities
common
CZ
Denmark
energy-efficiency
Estonia
Finland
```

Figure 28. Files folder contents



```
◢ [redacted]
   ◢ App_Data
      ▷ RadSpell
      ◢ Sitefinity
         ▷ Configuration
         ▷ GlobalResources
         ▷ Logs
         ▷ Search
         ▷ Temp
      ◢ Storage
         ▷ FileSystem
   ▷ App_Master
   ▷ aspnet_client
   ▷ backups
   ▷ bin
   ▷ ClientBin
   ▷ ControlTemplates
   ▷ CustomThemes
   ▷ dbbak
   ▷ ErrorPages
   ▷ Files
   ▷ ga
   ▷ img
   ▷ js
   ▷ Mvc
   ▷ Services
   ▷ SimpleWidgets
   ▷ Sitefinity
   ▷ Tests
   ▷ tools
   ▷ WebControls
```

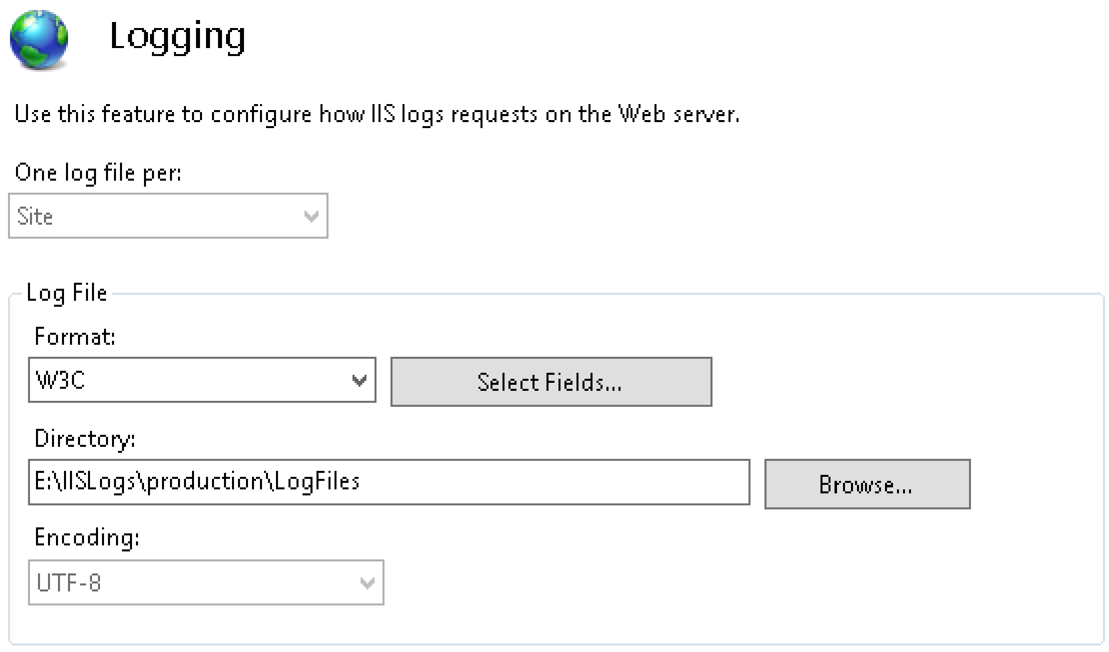
Figure 29. Virtual directories on IIS UI

3.7 Directing logs to new locations

Besides the Sitefinity and the importer logs which are virtual directories and which's locations where configured when other virtual directories' locations where configured, The IIS logs and the HTTP error logs' had to be change from their default location. (Microsoft. Technet. 2015)

3.8 Configuring the IIS logs location

For each of the sites the IIS logs where change to the location E:\IISLogs\production\LogFiles using the IIS console.


Logging

Use this feature to configure how IIS logs requests on the Web server.

One log file per: Site

Log File

Format: W3C

Directory: E:\IISLogs\production\LogFiles

Encoding: UTF-8

Figure 30. IIS's logging menu

3.9 Configuring the HTTP error logs location

HTTP error log is changed to the location E:/HTTPERR/logs with the following PowerShell script:

$myRegKeyBase = "HKLM:\SYSTEM\CurrentControlSet\services\HTTP\Parameters" $myRegKey-Name = "ErrorLoggingDir" $myRegKeyVal = "E:\HTTPERR\logs" # Create

Key property if it doesn't already exist $myRC = New-ItemProperty $myRegKey-Base -Name $myRegKeyName -Value $myRegKeyVal -PropertyType String -ErrorAction SilentlyContinue   # Update Key property to $iisLogFileDir if it already exists   Set-ItemProperty $myRegKeyBase -Name $myRegKeyName -Value $myRegKeyVal -ErrorAction SilentlyContinue $myProperties = Get-ItemProperty $myRegKeyBase -ErrorAction SilentlyContinue   Write-Host "regKeyPath: $($myRegKeyBase) regKeyName: $($myRegKeyName) value: $($myProper-ties.$myRegKeyName)"

For logging we are using 2 toolsets, one is Log4net and the otherone is sys-tem.diagnostics from the .NET framework.

To direct Log4net to Logs drive Log4net.config file was updated and the sys-tem.diagnostics the settings are changes in web.config.

3.10 Configure FTP server

BizTalk generates XML files containing integration data that is retrieved for the pro-duction environment at the cloud service provider's Links via FTP connection periodi-cally.

In order to stablish a FTP connection an FTP server was created on the IIS and a username and password was created and shared with the cloud service provider. (Mi-crosoft. IIS. 2015)

The FTP server contains several virtual directories where the XML files that contain integration data are dumped.

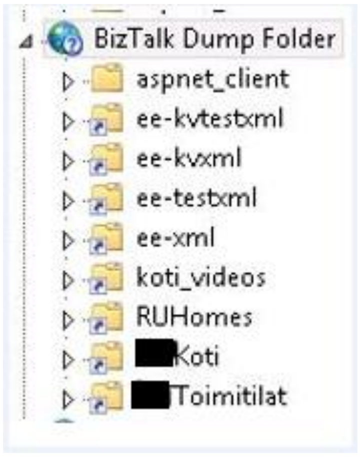
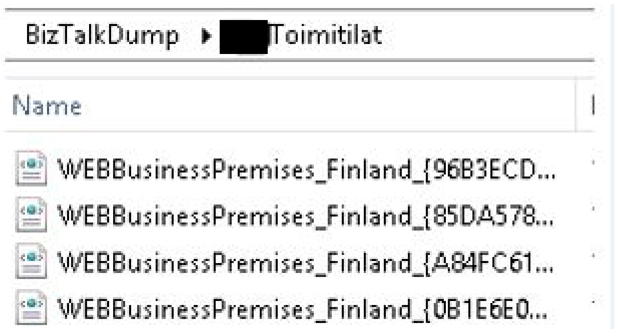

BizTalk Dump Folder
aspnet_client
ee-kvtestxml
ee-kvxml
ee-testxml
ee-xml
koti_videos
RUHomes
[redacted]Koti
[redacted]Toimitilat

Figure 31. FTP server's folders


BizTalkDump ▸ [redacted]Toimitilat

Name
WEBBusinessPremises_Finland_{96B3ECD...
WEBBusinessPremises_Finland_{85DA578...
WEBBusinessPremises_Finland_{A84FC61...
WEBBusinessPremises_Finland_{0B1E6E0...

Figure 32. FTP server folder contents

Transfer process:

1. The Source system generates a transfer data file(s) based on the manual trigger or internal scheduler of the Source.
2. The transfer data file(s) generated by the Source are pushed into FTP directory at the Destination.
3. XML Preprocessor is triggered manually or based on the scheduler at the Destination.
4. Import process is triggered manually or based on the scheduler at the Destination.
5. After a successful import, the transfer data file(s) are moved into archive directory.

XML Preprocessor:

XML Preprocessor is a standalone application that is used to prepare the XML file pushed into FTP in way that will optimize the import process. The XML Preprocessor could be triggered manually or by a scheduler on the Destination machine.

XML Preprocessor workflow:

1. Load the base(archived XML) and the one pushed into FTP;
2. Compare the differences between the two XMLs:
    1. Detect changes over the data;
    2. Insert new data (Projects, Apartments, Attachments, etc.);
    3. Remove missing data.
3. Download all the attachments in the new generated file and create:
    1. AttachmentID – hash of the attachment location;
    2. Hash – hash of the binary content;
    3. UpdatedAt – add the current date in UTC, XML formatted if there is a difference between the base and the new xml.
4. Save the final file that is containing the specific information that need to be imported.

3.11 DNS Resolution

The last step for the websites to be able to be browsed was adding the bindings to the hosts file. This is a simple procedure that requires simply to open the hosts file which can be done with the command: `notepad C:\Windows\System32\drivers\etc\hosts` on the windows command prompt and add the DNS to it.

3.12 Testing

Each of the 27 company sites was smoke tested by browsing the sites inside the server using the IIS console for each binding.

Browse Website

Browse sf-ee[redacted].com on *:80 (http)

Browse [redacted]-ee.local on *:80 (http)

Figure 33. Website URL bindings

After successfully browsing each site with their bindings without getting any errors, the migration was completed and the servers were ready to go live.

3.13 Migration

After the stage and production were ready to be migrated there was a plan stablished to be followed on the migration day.
The migration protocol went as following:

3.13.1 GO LIVE on stage

E.g. starting on Mon 23.3.2015.

1. 16:00, Content freeze starts*, importers shut down
   a. Notifications to ALL when done
2. 16:10, Contents of the production DFS copied to the new servers
   a. Time consuming task (4h)
   b. Notifications to ALL when done
3. 16:00, Add stage3 and stage4 to the load balancer – can be done earlier during the day, if more convenient
   a. Notifications to ALL when done
4. 16:00, Switch BizTalk to send data to stage3 – can be done earlier during the day, if more convenient
   a. Notifications to ALL when done

Following Day:

5. 10:00, Smoke testing (Manual)
   a. For each site, start on front3 and front4
   b. For each site – shut down front1 and front2
   c. Verify each server – front3 and front4
6. 15:00, Smoke testing completed
   a. Notifications to ALL when done, if OK continue to next step, if Not OK, telco (15:00) about next actions
   b. 15:00, Importer smoke testing
   c. test data to be secured
   d. Notifications to ALL when done
7. 15:00, CRF service check from BizTalk/VIP
   a. Check all VIP sites
   b. Notifications to ALL when done
8. 16:00, GO decision from all parties via email
   a. if OK from ALL, continue to next step, if Not OK, telco about next actions
9. 16:00, Content freeze over, importers resumed

*) Content freeze not 100% necessary, stage can be used during the operation.

On the following week (date to be agreed on weekly 1.4) remove stage1 and stage two from the load balancer, see section 4 for the procedure.

3.13.2 GO LIVE on production

10. E.g. starting on Fri 27.3.2015.
11. 19:00, Content freeze starts, importers shut down
12. Notifications to ALL when done
13. 19:10, Contents of the production DFS copied to the new servers (Risk)
14. Time consuming task (4h)
15. Notifications to ALL when done
16. 12:00, Add Front3 and Front4 to the load balancer
17. Notifications to ALL when done
18. 19:00, Switch BizTalk to send data to front3 (Risk)
19. Notifications to ALL when done

Following Day:

20. 10:00, Smoke testing
21. For each site, start on front3 and front4
22. For each site – shut down front1 and front2
23. Verify each server – front3 and front4
24. 15:00, Smoke testing completed
25. Notifications to ALL when done, if OK continue to next step, if Not OK, telco about next actions
26. 15:00, Importer smoke testing
27. test data to be secured
28. Notifications to ALL when done
29. 15:00, CRF service check from BizTalk/VIP
30. Check all VIP sites
31. Notifications to ALL when done
32. 16:00, GO decision from all parties via email (no requirement from cloud service provider but availability yes)
33. if OK from ALL, continue to next step, if Not OK, telco about next actions
34. 16:00 – 18:00 Monitoring the sites, cloud service provider to designate a specific contact for this time in case of issues and/or rapid revert needed
35. 16:00, Content freeze over, importers resumed
36. 18:00 – 21:00 On call for emergencies

3.14 Results

The migration was successful although there were some delays with the schedule. There were some misconfigurations of the load balancing settings from the part of the cloud hosting service provider that didn't allowed the web server to serve some sites which were fixed after the testing.

There was also some firewall setting problems with the FTP server connection that were fixed as well.

At the end the customer was satisfied with the migration. The issues with performance and stability that existed in the old database servers disappeared which was a main concern in the old server environment. (Matti Kiviluoto, 2015)

The team was expecting a performance boost on the front-end servers due to the upgrade of resources provided by the new server but unfortunately there was no significant boost on site performance. (Matti Kiviluoto, 2015)

After the migration was completed there were some downsides identified by our team regarding the service provided by the cloud hosting service.

Mainly, the problem was the delayed response time when something was requested from them. In contrast the availability of resources is immediate in other projects that use Amazon Web Services due to the fact that we have access to the console. (Matti Kiviluoto, 2015)

3.15 Discussion

3.15.1 Hosted vs on premise advantages and disadvantages

When deciding between hosting a project on premise or online the specific needs of the company must be taken into account. There is not one solution that fits all cases and therefore the advantages and disadvantages of both options have to be considered.

On premise advantages:

- **Control over systems and data:** The advantages of hosting a project on promise are first of all, control over the whole system and data. Data can be retrieved on demand and decisions can be taken internally expediting the results.

- **Corporate data is stored and handled internally:** Retaining privacy of sensitive information is another advantage of on premise hosting. Since the data is not given to a third-party to handle the privacy of the data is safer.

- **Optimal utilization of existing hardware:** Hardware owned by the company is can be utilized more optimally when projects are hosted internally. If the company owns the required hardware costs can be saved by avoiding paying for hardware used by third parties.

- **High level of compliance with company's standards:** The level of compliance with the company's criteria is higher when the company itself is handling the project internally.

- **On-demand provisioning of resources**: Resources can be provided on demand around the company since there are not intermediaries between the resources and someone needing them inside the company.

(Salam, Gilani, Ul Haq. 2015)

On premise disadvantages:

- **Extra resource management load:** one of the main disadvantages to on premise project hosting is that the management becomes more complex and more resources are required. Moving the project hosting to a third party service greatly decreases the load of management and the resources required.

- **Need for special skills in cloud solution:** The Company will have to invest in looking and hire people with special skills to handling the project. This can increase the cost of managing the project.

- **Additional costs for infrastructure:** The infrastructure can also increase the cost of running things internally since there is an additional need for hardware, software and work involved in setting up and maintaining the infrastructure. (Salam, Gilani, Ul Haq. 2015)

Taking into account the advantages of on premise hosting it is best suit for large companies with hardware resources under-utilized.

It may be the case for and enterprise to have hardware resources that could be not in use and could supply all the hardware need to host their projects.

In this case using those resources could end up been cheaper than paying extra to a third-party service.

Having the necessary hardware resources is not enough to go for on premise hosting, the company needs to take into consideration that it will have to invest in long-term expertise to run the resources.

Hiring cloud computing and infrastructure experts could be a smart investment in the case the company has many projects or big enough projects so that employees can be utilized at maximum capabilities.

If the project is too small it hiring experts to manage it can be inefficient and outsourcing it to a third party service may be more cost-effective.

In case a company has not the hardware resources, it must have a reason to want invest in infrastructure resources and human resources.

The truth is that the amount of companies that will beneficiate from hosting their own projects is reduced to large corporations with already possession infrastructure resources, having large projects, and with the budget to hire people with special skills in cloud computing and infrastructure.

Virtual Hosting advantage:

- **No software licensing costs:** Usually the costs for software and tools required to manage a project falls under the responsibility of the hosting service provider, therefore the costs of buying software are saved. In addition the workload of choosing the right software is saved as well.

- **No new infrastructure requirements:** Project hosting providers supply all the infrastructure needed to manage a project (hardware and interfaces). Since they specialize in providing this services the cost of infrastructure is cheaper and there is no set up cost.

- **Low cost for services:** Getting the service from a provider can be many times cheaper and more efficient than running all the infrastructure on premise, avoiding the costs not only of infrastructure but also of specialized personnel.

- **Minimum vendor lock-in concerns:** Thanks to a wide array of competitors and options available and the easy technology provides for the migration of a project from one hosting provider to another the risk of locking-in with

- **No management requirements:** The maintenance and management of the project hosting infrastructure is also a responsibility of the service provider allowing the company to rip all the benefits of the service without major specialization in the issue.

- **On-demand provisioning of resources with no cost:** A good service provider will provide resources accessibility around the clock and without extra cost.

- **Strong levels of security, privacy, standards and regulations**: Data may in certain cases be more secure in hands of a service provider and the infrastructure may be more reliable and secure as well. Thanks to the service provider's specialization in the industry there is a higher level of compliance with the industry's standards and regulations.
  (Salam, Gilani, Ul Haq. 2015)

Disadvantages

- Additional compliances maybe required: The third party service provider usually will have its own terms and conditions that have to be followed as well. In some cases these will conflict with the company's standards and regulations and on some others they will give extra considerations and requirements to manage.

- Customers of company may have security concerns: Customers of the company may have concerns about the trustworthiness of the third-party service provider and this in turn can compromise the trustworthiness of the company as well.
  (Salam, Gilani, Ul Haq. 2015)

Taking into account the advantages and disadvantages of virtual hosting, we can say that virtual hosting works best for organizations that don't want to invest in private infrastructure and private human resources or don't want to maintain their own hardware and software.

Another aspect to consider is that a company will beneficiate from virtual hosting if they don't want to manage the compliance with standards, regulations, privacy and security policies.

When a company is handling highly sensitive and confidential information it also may be best to entrust this information to a dedicated company with higher competence in dealing with this kind of data, in this case a disadvantage may be that customers may have trust issues with third-party service providers. (Salam, Gilani, Ul Haq. 2015)

3.15.2 Comparing virtual hosting providers

The Gartner's 2015 Cloud Iaas Magic Quadrant ranks cloud providers by using two criteria: Ability to Execute and Completeness of Vision. (Joseph Tzidulko, 2015)

The Quadrant categorizes providers into four categories:

- Leaders: High on both criteria
- Visionaries: High on vision but low on ability to execute
- Challengers: High ability to execute but low on vision
- Niche Players: Low on both criteria

Out of 15 vendors considered only two are considered leaders, namely Amazon Web Services, and Microsoft Azure. (Joseph Tzidulko, 2015)

AWS strengths are:

- Extensive network of partners providing expertise, managed and professional services
- Greatest variation of IaaS and Platform-as-a-Service capabilities

(Joseph Tzidulko, 2015)

AWS weaknesses

- Customers must manage right level of sales and solution architecture they need
- Amazon new features compete with own partners

(Joseph Tzidulko, 2015)

Microsoft Azure strengths

- IaaS and PaaS are greatly integrated with one another as a whole
- Features are increasing very fast

(Joseph Tzidulko, 2015)

Microsoft Azure Weaknesses

- It lacks necessary security, availability, performance, networking flexibility and user management features for enterprises
- Azure has suffered more downtime than its main competitors which may require a third party disaster recovery solution
- The partner ecosystem is still lacking with many Microsoft partners still lacking the expertise needed to provide the quality required by customers (Joseph Tzidulko, 2015)

3.15.3 Customer's choice

Our customer's choice took about a year of analysis. The main reason they wanted to change servers was that the servers used at the time had expired their recommended life span of three years.

They knew that they needed to take a step from on-premise to virtual hosting. This decision was easy to take as they don't possess a strong IT department able to provide with all the infrastructure and hardware/software maintenance or the human expertise needed to provide with high levels of quality hosting service.

They also knew that they would benefit from costs reductions, service quality improvement and higher adherence to industry standards.

The problem now was to find the right third-party service. Naturally the first services considered were the industry leaders: Amazon Web Services and Microsoft Azure.

Taking into account Gartner's Magic quadrant we can see that Amazon Web Services has a clear advantage margin over Azure (Joseph Tzidulko, 2015), although Azure had the added benefit of significantly lower costs thanks to the partnership between the company and Microsoft.

In addition to all of the previously mentioned consideration the company had issues with relying with non-Finnish companies because they would not be bounded by Finnish legislation.

At the end the company decided that having a Finnish service provider bounded by Finnish legislation would be more beneficial that higher quantity of features services like Microsoft Azure or AWS would provide.

3.15.4 Experience with chosen provider:

At the beginning of the analysis concerning which cloud hosting service to choose, our team suggested Amazon Web Services as a first option and Azure as a second one. The reason behind was our extensive experience working with Amazon Web Services, the amount of features and the first-hand accessibility. (Matti Kiviluoto, 2015)

The team was not able to convince the customer which had a concern with the data not being stored inside Finnish boundaries nor protected by Finnish laws if they chose a global hosting provider. (Matti Kiviluoto, 2015)

After the migration our team experienced the downsides of choosing a local cloud hosting provider, such as the inability for cost optimization, server optimization and fast resource scalability. (Matti Kiviluoto, 2015)

This was all due to the fact that the third-party service provider had the control over the resources and in order to make a change our team had to get in contact with them instead of accessing a console ourselves.

On top of this was the fact that the customer had always been very reluctant to make changes to the environment, and liked to keep the settings fixed with little room for flexibility, so that every development or improvement had to be negotiated with the customer and arranged with the cloud hosting provider. This delayed very much the project's planning and implementation. (Matti Kiviluoto, 2015)

The main disadvantages identified were delayed reaction time to requests and not getting the flexibility that we get from global cloud providers.

On the positive side, having a local cloud hosting provider meant that we had personalized service and a named person present in the meetings involved in the planning and execution of the project. (Matti Kiviluoto, 2015)

4 Part 3: Setting up build automation

After having the customer's source code migrated from SVN to Git and having migrated the project's infrastructure to a new virtual servers hosting environment, the next step was to set up the build automation for the project.

The build automation was set up in a build server owned by our company that is used to build several different projects of different customers using CruiseControl.NET.

CruiseControl.NET is an automated continuous integration server that uses the .NET Framework. In order to set the automated build there are the following steps we took:

- Installation of CruiseControl.NET and other required software
- Installation of Git
- Creating and configuring the project
- Setting the build automation tasks

4.1 Installation of CruiseControl.NET and other required software

CruiseControl.NET requires the .NET framework installed in the build server. The version of the framework must match the visual studio version used to create and develop the project.

CC.NET also requires IIS server with ASP.NET enabled to serve the user interface. (CruiseControl.Net, 2015)

After installing the CruiseControl.NET software we get:

- **CCNet server:** contains the core of the project, configuration files, executables, dependencies and project files
- **WebDashboard:** contains user interface files

After installation we have a site running on IIS containing the CC.NET dashboard.

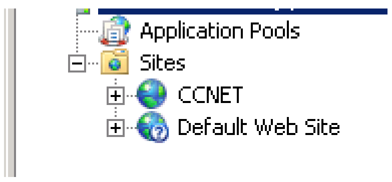
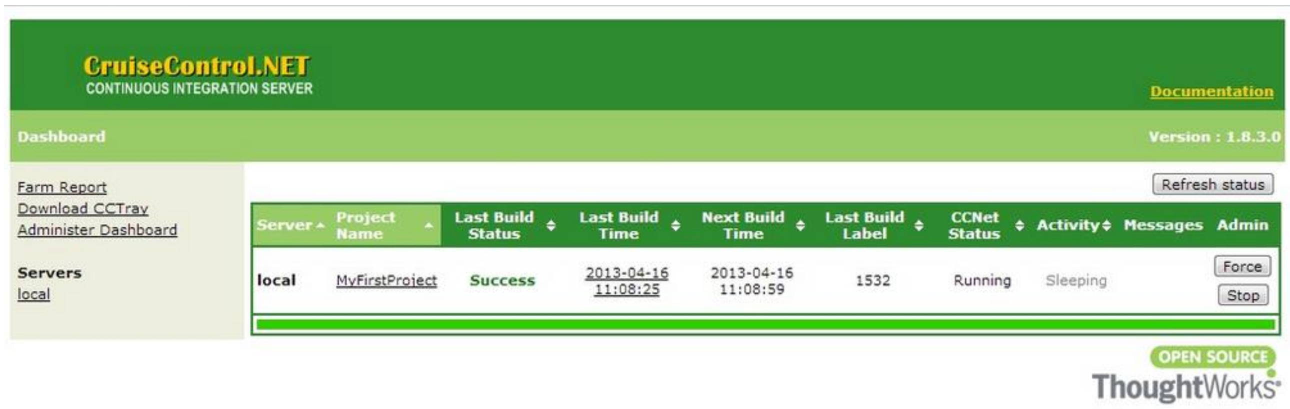


Figure 34. CCNET site on IIS



Server	Project Name	Last Build Status	Last Build Time	Next Build Time	Last Build Label	CCNet Status	Activity	Messages	Admin
local	MyFirstProject	Success	2013-04-16 11:08:25	2013-04-16 11:08:59	1532	Running	Sleeping		Force Stop

Figure 35. CrouseControl.NET Dashboard

4.2 Git client Installation

The Git client will allow us to clone the Git repository from Bitbucket to the build server so that we can build the project.

Git can be downloaded from https://git-scm.com/downloads and it is a good idea to install the Git- bash command line so we can test and troubleshoot the cloning manually.

4.3 Creating and configuring the project

To create a project we must open the ccnet.config which is an xml file and add a new <project> tag in the following way:

```
<project name="[PROJECT NAME]">

</project>
```

Every other configuration must be inside the <project> tags.

We can now define the working directory where we have the project files and the artifact directory which is mean for anything we want to save from the results of the build, like build logs, distributables, etc.(CruiseControl.Net, 2015)

```
<project name="[PROJECT NAME]">

  <workingDirectory>D:\Builds\[PROJECT NAME\[BRANCH NAME]</workingDirectory>

  <artifactDirectory>D:\Builds\[PROJECT NAME\[BRANCH NAME]\Artefacts</artifactDirectory>

</project>
```

Next we can add a “source control” section where we can configure values like: source control type, repository URL, branch name, whether to sync the latest changes before the build (autoGetSource), whether to fetch git submodules, where to find the Git executable, set tags, set working directory and timeout limit. (CruiseControl.Net, 2015)

```
<project name="[PROJECT NAME]">

…

  <sourcecontrol type="git">

                    <repository>[REPOSITORY URL]</repository>
                    <branch>[BRANCH NAME]</branch>
                    <autoGetSource>true</autoGetSource>
                    <fetchSubmodules>true</fetchSubmodules>
                    <executable>C:\Program Files (x86)\Git\bin\git</executable>
                    <tagOnSuccess>false</tagOnSuccess>
                    <commitBuildModifications>false</commitBuildModifications>
                    <commitUntrackedFiles>false</commitUntrackedFiles>
                    <tagCommitMessage>CCNet Build {0}</tagCommitMessage>
                    <tagNameFormat>CCNet-Build-{0}</tagNameFormat>                    <working-
                    Directory> D:\Builds\[PROJECT NAME\[BRANCH NAME]</workingDirectory>
                    <timeout>600000</timeout>

   </sourcecontrol>


…

</project>
```

We can also set automated pulls and builds of the project at specific times in the day and specific days of the week as following:

```
<project name="[PROJECT NAME]">

…

 <triggers>

    <scheduleTrigger time="07:30" buildCondition="ForceBuild" name="Scheduled">

                    <weekDays>
                              <weekDay>Sunday</weekDay>
                              <weekDay>Monday</weekDay>
                              <weekDay>Tuesday</weekDay>
                              <weekDay>Wednesday</weekDay>
```

```
                                    <weekDay>Thursday</weekDay>
                      </weekDays>

    </scheduleTrigger>

  </triggers>

…

</project>
```

After setting the source control values and the triggers we can set a task which will execute MS build to build the project.

Here we must set the location of the MS build executable, the location of the ".proj" file which is an XML file containing the MS build task we want to execute, the name of the ".proj" file, the parameters to pass to the file, the timeout limit of the MS build and the logging settings.

```
<project name="[PROJECT NAME]">

…

 <tasks>
      <msbuild>
            <executable>
                                    C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe
                  </executable>
            <workingDirectory>
                                    D:\Builds\[PROJECT NAME\[BRANCH NAME]\
                  </workingDirectory>
            <buildArgs>
                                    UomaYITWebBuild.proj /noconsolelogger /t:Debug
                                    /p:Configuration=Release
                                    /p:Environment=Release
                                    /p:CopyToIIS=$[CopyToIIS|Yes]
                                    /p:BuildZip=$[BuildZip|No] /p:CleanDeployFolder=$[CleanDeployFolder|No]
                                    /p:HostingEnvironment=development_Git
            </buildArgs>
            <timeout>1800</timeout>
            <logger>
                                    C:\Program Files(x86)\CruiseControl.NET\... serv-
                                    er\ThoughtWorks.CruiseControl.MSBuild.dll
            </logger>
```

```
     </msbuild>
 </tasks>

…

</project>
```

The last step is to add a logging location as following:

```
<project name="[PROJECT NAME]">

…

            <publishers>

                        <xmllogger logDir="D:\Builds\[PROJECT NAME]\Logs\[Branch NAME]" />

            </publishers>


…

</project>
```

4.4 Setting the build automation tasks

MS build is a framework to build projects, it tasks that can be called and modify to customize the build process.

The ".proj" file is a file containing XML markup that contains the MS build tasks' specifications. The location and name of the .proj file is referenced in the ccnet.config configuration file to be used during the build of the project.

A collection of tasks is called a target. Our project's .proj file contains the following targets:

NugetExe: Executes the nuget.exe, a program that downloads .Net dependencies over the internet needed to build the project successfully.

The Nuget application is also configured in our project to download sitefinity dependencies.

By downloading all the dependencies at the time of the build instead of having them already in the repository makes the repository lighter and faster to clone.

Clean: Cleans the build which means that all the intermediate and output files are deleted and only the project and component files are left. New intermediate and output files will be created after the build. (Microsoft. MSDN, 2015)

Publish: Builds and publishes the project to a specified system's file location.

CleanDeployFolder: Deletes the contents of the deploy folder which contains a copy of website, services, and configurations.

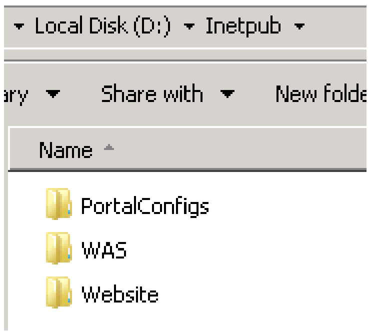


Figure 36. Deploy Folder

CopyAssemblies: Copies DLL and PDB files from the project pulled from Git to the published project.

CopyServicesWebsite: Copies the services according to the environment from the project pulled from Git to the published project.

CopyPortalConfigs: Copies Sitefinity configurations according to the environment from the project pulled from Git to the published project.

CopyTimeZones: Copies time zone configurations according to the environment from the project pulled from Git to the published project.

CopyOnServer: Copies the published project to a location in the server where IIS will be configured to serve it on the localhost.

MergeTimeZoneSettings: Each site in the project has different time zone settings that are merged into a single file by this task.

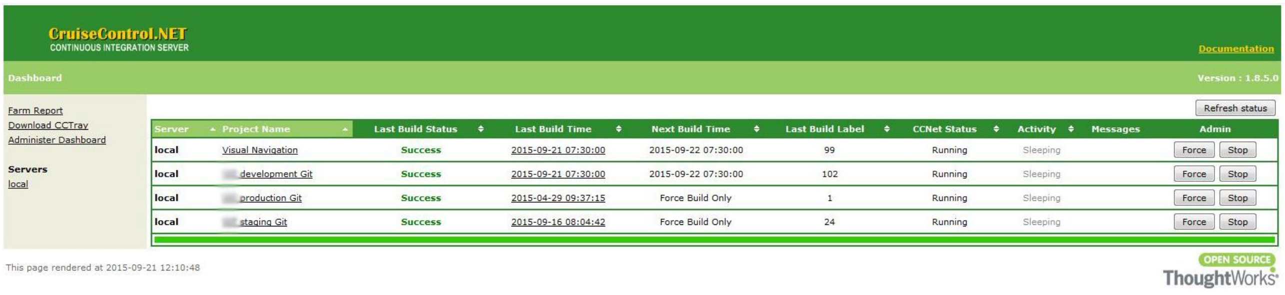
MergeLoadBalancingConfigs: Each site in the project has different load balancing settings settings that are merged into a single file by this task.

```
<Target Name="Debug">

   <CallTarget Targets="NugetExe" />
   <CallTarget Targets="Clean" />
   <CallTarget Targets="Publish" />
   <CallTarget Targets="CleanDeployFolder" Condition="$(CleanDeployFolder)=='Yes'" />
   <CallTarget Targets="CopyAssemblies" />
   <CallTarget Targets="CopyServicesWebsite" />
   <CallTarget Targets="CopyPortalConfigs" />
   <CallTarget Targets="CopyTimeZones"/>
   <CallTarget Targets="CopyOnServer" Condition="$(CopyToIIS)=='Yes'" />
   <CallTarget Targets="MergeTimeZoneSettings" Condition="$(CopyToIIS)=='Yes'" />
   <CallTarget Targets="MergeLoadBalancingConfigs"/>

</Target>
```

The projects can now be built by simply pressing the “force” button on the ccnet dashboard.



Server	Project Name	Last Build Status	Last Build Time	Next Build Time	Last Build Label	CCNet Status	Activity	Messages	Admin
local	Visual Navigation	Success	2015-09-21 07:30:00	2015-09-22 07:30:00	99	Running	Sleeping		Force Stop
local	[redacted] development Git	Success	2015-09-21 07:30:00	2015-09-22 07:30:00	102	Running	Sleeping		Force Stop
local	[redacted] production Git	Success	2015-04-29 09:37:15	Force Build Only	1	Running	Sleeping		Force Stop
local	[redacted] staging Git	Success	2015-09-16 08:04:42	Force Build Only	24	Running	Sleeping		Force Stop

This page rendered at 2015-09-21 12:10:48

Figure 37. CCNET dashboard with projects

After the project is build and published the project can be downloaded from an URL and manually copied into the servers and configured to be served over the internet.

The build automation's infrastructure is scheduled to be developed further so that the project gets to the servers automatically.

5 Results and Conclusions

The SVN to Git migration was a success with all of the revisions and all the important branches transferred to Git. There were some problematic branches that were decided to be left behind because they were not necessary anymore.

After the migration the source code was able to be downloaded and run against a Sitefinity database and served with IIS.

The Git workflow implemented in the beginning proved to be flawed and old problems that the team had while working on SVN recurred while implemented the Git workflow. One of the most persisting problem was that of the team being force to "cherry-pick" features that would go into production which created an instability in the code.

Nevertheless the Git workflow was modified and redesign to prevent the challenges encounter with the first implementation and at the end the Git workflow was successful in providing a reliably and effective way of managing the continuous integration.

The migration of the project from on premise server environment to the cloud hosting environment had challenges of its own. The main challenge was choosing a cloud hosting provider and coordinating the migration with a third party server provider.

One of the main issues was concerns with the quality of the service provided which proved to be challenging in some cases where some aspects of the migration where unsuccessful due to flawed configuration settings from the third-party cloud hosting provider.

Sometimes there were problems with the load balancing settings that made the website unavailable over the internet and some others were network setting problems that didn't allow us to connect to the servers.
We would have preferred to use a service like Amazon Web Services or Microsoft's Azure instead of the chosen by our customer but at the end we could not affect their decision.

The migration was a success nevertheless although we have encounter some problems with the quality of the service provided from time to time.

The build automation has given us no major challenges or issues and has been running successfully from months without any human intervention in the configuration.

The reason behind the smooth success of the built automation is due to a year and a half of design and implementation in virtually all the customer's projects.

The build process has been refined by testing it in many projects and making improvements and fixes along the way, therefore the implementation of the build process on this project didn't present any challenges.

The development team overall has been experienced improvements in the efficiency of integrating developments into the Git repository and faster deployment time with the build automation.

6 References

Abdul Salam, Zafar Gilani, Salman Ul Haq, 2015. Deploying and Managing a Cloud Infrastructure. SYBEX. URL:https://books.google.fi/books?id=f_clBgAAQBAJ&pg=PR23&dq Accessed: June 15 2015

Atlassian 2014. Migrate to Git from SVN. URL: https://www.atlassian.com/git/tutorials/migrating-overview/ Accessed: November 2014

Atlassian 2015.Comparing Workflows. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/ Accessed: September, 05 2015

Avaus Consulting Oy, 2014. Customer's project documentation. Intranet. Project Infrastructure. Accessed: May, 22 2015

CruiseControl.Net, 2015. Documentation. Project Configuration Block. URL: http://cruisecontrolnet.org/projects/ccnet/wiki/Project_Configuration_Block Accessed: September, 15 2015

CruiseControl.Net, 2015. Wiki. Install. URL: Http://www.cruisecontrolnet.org/projects/ccnet/wiki/Install Accessed: September, 15 2015

CruiseControl.Net, 2015. Wiki. Install. URL: Http://www.cruisecontrolnet.org/projects/ccnet/wiki/Install Accessed: September, 15 2015

Giancarlo Lionetti February 14, 2012. URL: http://blogs.atlassian.com/2012/02/version-control-centralized-dvcs/ Accessed: May, 18 2015

Joseph Tzidulko, 2015. Here's Who Made Gartner's 2015 Cloud IaaS Magic Quadrant. URL: http://www.crn.com/slide-shows/cloud/300076877/heres-who-made-gartners-2015-cloud-iaas-magic-quadrant.htm Accessed: June 17 2015

Kiril Jovchev, 15 May 2015. Solutions Architect, Team Leader. Avaus Consulting Oy / Siili Solutions Plc. Interview. Helsinki.

Matti Kiviluoto, 23 September 2015. Team Leader. Avaus Consulting Oy / Siili Solutions Plc. Interview. Helsinki.

Microsoft 2013. MSDN. Add-WindowsFeature. URL https://msdn.microsoft.com/en-us/library/ee662309.aspx Accessed: September, 05 2015

Microsoft 2015. IIS. Binding. URL: https://www.iis.net/configreference/system.applicationhost/sites/site/bindings/binding Access: June 10 2015

Microsoft 2015. IIS. Configure FTP with IIS manager. URL: http://www.iis.net/learn/publish/using-the-ftp-service/configure-ftp-with-iis-manager-authentication-in-iis-7 Accessed: June 10 2015

Microsoft 2015. IIS. URL Rewrite. URL:http://www.iis.net/downloads/microsoft/url-rewrite Accessed: September, 05 2015

Microsoft 2015. IIS. Using the application request routing module. URL: http://www.iis.net/learn/extensions/planning-for-arr/using-the-application-request-routing-module Accessed: September, 05 2015

Microsoft 2015. MSDN. WCF 4.5 HTTP Activation. URL: https://msdn.microsoft.com/en-us/library/jj979362%28v=winembedded.81%29.aspx Accessed: September, 05 2015

Microsoft 2015. TechNet. Checking the HTTP Error Log (IIS 6.0). URL: https://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/bcabdcfb-2421-eab-b5fb-777c791daaa9.mspx?mfr=true Access: June, 10 2015

Microsoft, 2015. IIS. Installing IIS 8.5 on Windows Server 2012 R2. URL: http://www.iis.net/learn/install/installing-iis-85/installing-iis-85-on-windows-server-2012-r2 Accessed: September, 05 2015

Microsoft, 2015. Support. How to Create a Virtual Directory in Internet Information Services (IIS). URL: https://support.microsoft.com/en-us/kb/172138 Accessed: July, 02 2015

Microsoft, 2015. TechNet. Customizing IIS 7.0 Roles and Modules.
Project's Developer, 23 September 2015. Web Developer. Avaus Consulting Oy / Siili Solutions Plc. Interview. Helsinki.

Telerik 2015. Administration settings and configuration. URL: http://docs.sitefinity.com/administration-settings-and-configurations Access: June, 10 2015

Telerik, 2015. Configure the IIS to host Sitefinity projects, URL: http://docs.sitefinity.com/configure-the-iis-to-host-sitefinity-projects Accessed: June, 02 2015

CruiseControl.Net, 2015. Configuration Examples. URL: http://ccnet.sourceforge.net/CCNET/Perforce%20Source%20Control%20Block.html Accessed: September, 15 2015

CruiseControl.Net, 2015. Wiki. Git Source Control Block. URL: http://cruisecontrolnet.org/projects/ccnet/wiki/Git Accessed: September, 15 2015

Microsoft 2013. MSDN. How to Clean a Build. URL https://msdn.microsoft.com/en-us/library/ms171480.aspx Accessed: September, 16 2015