UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Industrialization of a multi-server software solution

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Víctor Martínez Bevià

Tutor: Andrés Terrasa Barrena
Jaume Jordán Prunera

Course 2016-2017

Acknowledgements

To my tutors, Andrés and Jaume, for their care, attention and continued help through the execution of this Degree Final Work.

To Encarna Maria, whose support knows no end.

Resum

L'objectiu del Treball de Fi de Grau és reescriure el procés de desplegament per a una solució de programari multiservidor per ser conforme amb el programari d'orquestració i automatització i redissenyar un flux de treball seguint pràctiques d'integració contínua i proves automàtiques. El projecte també inclou la implementació dels instal·ladors de cada component per a les arquitectures Windows i Linux emprant la infraestructura pròpia de l'empresa.

Paraules clau: Industrialització, integració contínua, instal·ladors, testing, vmware orchestrator, jenkins, robot, gitlab

Resumen

El objetivo del Trabajo de Fin de Grado es reescribir el proceso de despliegue para una solución software multiservidor para estar en conformidad con el software de orquestación y automatización y rediseñar un flujo de trabajo siguiendo prácticas de integración continua y pruebas automáticas. El proyecto también incluye la implementación de los instaladores de cada componente para las arquitecturas Windows y Linux usando la infraestructura propia de la empresa.

Palabras clave: Industrialización, integración continua, instaladores, testing, vmware orchestrator, jenkins, robot, gitlab

Abstract

The goal of the Final Degree Work is to rewrite the deployment process for a multiserver software solution in order to be compliant with the company's orchestration and automation software while redesigning a development workflow following continuous integration and continuous automated testing principles. The project also includes the implementation of each component installer for Linux and Windows architectures using the company's in-house framework.

Key words: Industrialization, continuous integration, installers, testing, vmware orchestrator, jenkins, robot, gitlab

Contents

List of Figures

List of Tables

List of Algorithms

Listings

CHAPTER 1
Introduction

This Degree Final Work has been made while working in a real company. The content of the dissertation has been agreed between the co-tutors and the company, but the name of said company and the components that are referenced in this document have been changed in order to preserve anonymity. The scope of this dissertation is a redesign of the deployment process of an existing software solution with an automation goal in mind.

Figure 1.1: Fielding logo

As far as this dissertation is concerned, the work will be done for *Fielding*, a fictitious company working in the field of trend analysis, the effectiveness of commercial campaigns and brand awareness.

1.1 Motivation

With the advent of virtualization and cloud computing, great advances have been made so the process of installing a new server and deploying a service is as fast and errorless as possible. A lot of business are embracing *Infrastructure as Code*[1] methodology and cloud and automation tools using, however, the same principles as when it took days or weeks to provision a new server. It is easy then to get carried away and deploy more servers than it is possibly able to manage.

One of the problems of cloud and virtualization automation is this possibility of not being able to keep up with the management of the virtual machines as they grow up in number. This is commonly known as *server sprawl*[1]. This term describes a scenario where it is needed to apply a patch to a set of servers but the patch is not applied to all the servers that could be affected in the solution. This inconsistency due to *server sprawl* is called *configuration drift*[1]. This *configuration drift* leads to servers with unique configuration that is not possible to ascertain how will respond to the automation tools, leading to fear of using the automation tools.

When designing an automated deployment scenario, in order to avoid these problems, the systems should be:

- **Easily deployable**

  The deployment step should not be a challenge. It should be a tested process that requires a minimum of configuration input and, if possible, no interaction until the process has ended.

- **Disposable**

  Instead of limit and control change to avoid breaking something, the servers should be liable to be destroyed, replaced or resized if changes in configuration demand it.

- **Consistent**

  Two servers that offer the same service and were deployed following the same template should be nearly identical. They should be dimensioned equally, and have the same setup, except for the configuration parameters that differentiate them (host name, IP address, etc.)

- **Reproducible**

  If the server deployed contains a bug, the redeployment of that server will still contain that bug. With a reliable, tested, deployment process we eliminate deployment-related bugs.

To manage all this server movement, the usage of a configuration manager (SaltStack[1], Ansible[2], Chef[3], Puppet[4]) is commonplace in automation nowadays. This migration to a centralized, automated deployment system not only brings advantages in the time factor, it also increases traceability of changes (be it new installations or updates) reducing the aforementioned *configuration drift*.

Nevertheless, to reach a fully automated scenario where it can be defined in a script what, when, and how is a server to be deployed, the starting point is the installer level. Automating the software deployment in a big company, where each product is done by a different department, requires a great deal of organization and synchronization.

In this context, this Degree Final Work tackles the deployment automation of one of the Fielding software solutions called Maptics, which purpose is to track and analyze the effectiveness of marketing campaigns, improving the existing deployment technologies, which are now described in the following section.

1.2 Initial State

The *Maptics software solution* has been developed by an isolated branch of the *Fielding* company. That is why since the beginning files and folders conventions, software compatibility and development guidelines were not taken into account. All of this resulted

[1] *SaltStack Enterprise software helps IT organizations, systems administrators, site reliability engineers and developers configure and manage modern data center infrastructure, applications and code. SaltStack takes a modern approach to systems management and is specifically built for the speed, scale and complexity of cloud and enterprise IT at scale.* For more information, consult `https://saltstack.com/enterprise/`

[2] `https://www.ansible.com/`

[3] `https://www.chef.io/`

[4] `https://puppet.com/`

in a software solution inside *Fielding* with its own way of deployment and configuration separate from the standards of the company.

1.2.1. Deployment solution

Windows

The Windows installers were made with Powershell scripts. All the Windows components were installed by the same script, which had a graphical interface made with the *system.windows.forms* library. Running the graphical interface from Powershell requires the execution mode to be STA[5], wich requires to handle background jobs manually if you want to execute installation jobs and mantain responsiveness in the Graphical User Interface (GUI). This resulted in a maintenance overhead while there are frameworks out there to create installers.

There was no uninstall functionality nor use of the Registry. It was not possible to install via command-line and the choice of folders and parameters was not compliant with Fielding guidelines.

Linux

The components were installed via a Bash script. It was not possible to upgrade automatically, there was no uninstall functionality and the choice of folders and parameters was not compliant with Fielding guidelines.

1.2.2. Release Generation

For the *Maptics Software Solution*, a release is generated at least once a week. Before the work associated to this report, the workflow was as follows:

1. The *Development* team generates a release with Team Foundation Server.
2. If the release is generated successfully, an e-mail is sent to the *System Integration* team.
3. The *System Integration team* executes a sanity check to test the release.
4. If the sanity check passes, the *System Integration* packages the release adding deployment scripts.
5. The *Quality Assurance* environment is upgraded to the new version.
6. An e-mail is sent to the *Quality Assurance* team to notify of the update.
7. The *Quality Assurance* team begins the testing process.

[5] *Single-Threaded Apartment* https://msdn.microsoft.com/en-us/library/windows/desktop/ms680112(v=vs.85).aspx

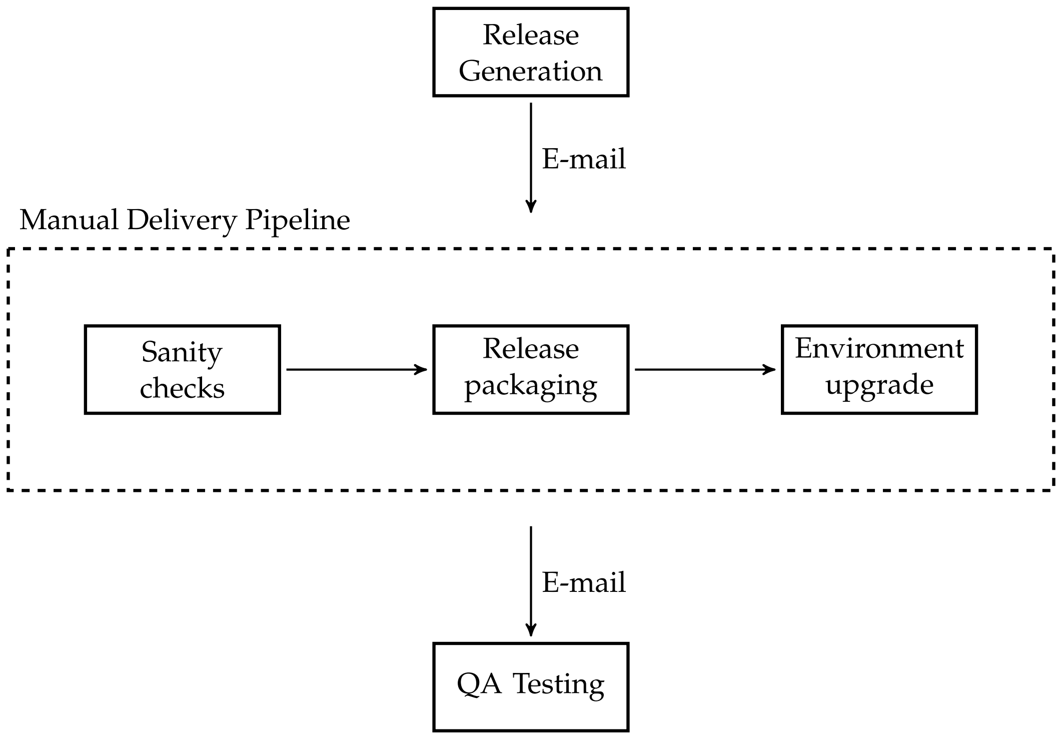


Figure 1.2: Manual Delivery Pipeline

This scenario requires the *Development*, *System Integration* and *Quality Assurance* teams to be present at release generation time, and each step to be executed manually after checking the results. This workflow lengthens the process unnecessarily, with no real advantage over an automated workflow where the release pipeline progresses as long as the defined tests pass.

1.2.3. Testing

There was no automation in regards to integration or build verification tests. There was a definition of test cases to pass and each one had to be checked manually. This limits the amount of testing to be done, leaving out any sort of exploratory testing.

1.3 Goals

To service its customers, *Fielding* provides multiple software solutions that fit a lot of use cases. *Maptics* is one of these solutions. It is a geolocation data analysis suite comprised of an array of servers, each one providing its own service.

The management of these different software solutions is done through the *Field Marshall* component, an application based on SaltStack. The use of this tool provides automation, but demands, in turn, a standard to comunicate with the installers, a predefined log directory, and common interfaces to manage the components services. The goal of this Degree Final Work is to redesign and rewrite the deployment process for *Maptics* in order to be compliant with the company's orchestration and automation software while designing a development workflow following continuous integration and continuous automated testing principles.

This main goal can be refined by stating the following particular objectives to be fulfilled by this project:

1. The new deployment system must be compatible with the current company standards, in particular, with in-house installation *Fielding* frameworks, standard paths and installation/upgrade procedures.
2. The technical debt of the deployment development environment must be reduced implementing *continuous integration* practices.
3. Each deployer must susceptible to be installed unattendedly, in order to be able to be automated.
4. A suite test should be developed for each component, in order to test automatically the outcome of a deployment and its ability to integrate with other components.

1.4 Structure of the dissertation

This dissertation is organized into six chapters:

- **Chapter 1, Introduction**
- **Chapter 2, Development Environment**
  Where the tools used during the dissertation are explained.
- **Chapter 3, Maptics Architecture**
  Where the architecture is explained, the version scheme is defined and the requirements are detailed.
- **Chapter 4, Deployment Solution**
  Where each component is analyzed, its requirements outlined and the corresponding actions to be carried out are specified.
- **Chapter 5, Testing**
  Where the desired final configuration is ensured by a battery of tests designed around its system requirements.
- **Chapter 6, Conclusions**
  Where the problems found during the dissertation are explained, improvements are proposed and conclusions are extracted.

CHAPTER 2

Development Environment

One of the main goals of this Degree Final Work was to reduce the technical debt in the systems department. Before the work portrayed in this dissertation, there was packaging of the software solution, and the released versions were just the folders copied over with a Powershell script. The installation scripts the Operations team used were just modified on the fly in and copied over with the rest of the solution. Any testing would entail deploy manually a virtual machine and install the software, then manually check the issue at hand.

After deliberation, the choice for this Degree Final Work was to set up a development environment where the needed files to compile the installers were version controlled using *Git*, and the compilation was automated whenever a successful Maptics release is generated. In order to do this, the tools chosen were Gitlab[1] and Jenkins[2]. The first acting as a code repository with *continuous integration capabilities*, and the latter being an orchestrator able to execute a pipeline to deploy automatically the solution in a testing environment created from scratch and test it using VMware Orchestrator[3] and Robot Framework[4]. The following sections explain the use of these tools and the design of the delivery pipeline developed during this dissertation.

[1] *Provides Git repository management, code reviews, issue tracking, activity feeds and wikis.* `https://about.gitlab.com/`

[2] *The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.* `https://jenkins.io/`

[3] *VMware vRealize Orchestrator is a development- and process-automation platform that provides a library of extensible workflows to allow you to create and run automated, configurable processes to manage VMware products as well as other third-party technologies.* `http://www.vmware.com/products/vrealize-orchestrator.html`

[4] *Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD).* `http://robotframework.org/`

2.1 GitLab

Figure 2.1: GitLab logo

It is a requirement of the company that all code repositories are hosted internally, and as the chosen version control tool is git, GitLab is a natural step. Other than repositories management, GitLab presents us with (amongst other features):

1. *Active Directory* integration.
2. Issue tracking.
3. Time tracking.
4. Wikis.
5. GitLab *Continuous Integration*[5] and GitLab Runners[6].

These features are used intensively through the development lifecycle.

2.1.1. Development Workflow

The development workflow follows best practices from a continuous integration perspective, ensuring:

1. The main branch (*master*) will always contain a working version of the software.
2. Every commit to the main branch is built, tested, deployed to a testing environment and tested again.
3. The state of the last commit is reflected on the repository page.

Number one allows the automation of the Maptics release generation, as the only thing needed is to download the last commit to the main branch for each installer and build them.

Number two ensures that the compiled installers have passed a defined testing suite.

Number three helps the development team to quickly identify if the changes introduced in a commit have broken any feature.

Taking into account these best practices, a typical git workflow is as follows:

[5] *Integrated* Continuous Integration *and* Continuous Deployment *to test, build and deploy your code* https://about.gitlab.com/gitlab-ci/

[6] *GitLab Runner is the open source project that is used to run your jobs and send the results back to GitLab.* https://docs.gitlab.com/runner/

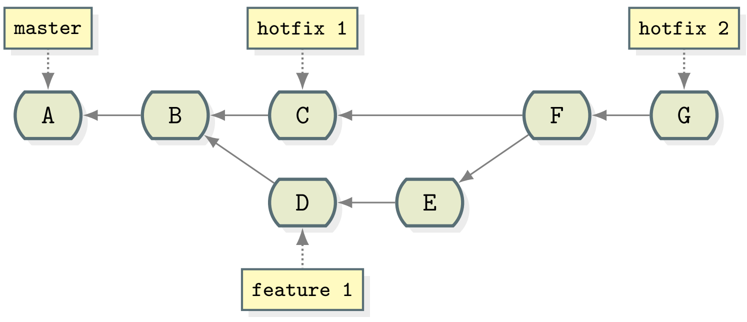


Figure 2.2: Git worklow

2.1.2. *Continuous Integration* Pipeline

The following pipeline is executed with every commit to the main branch:

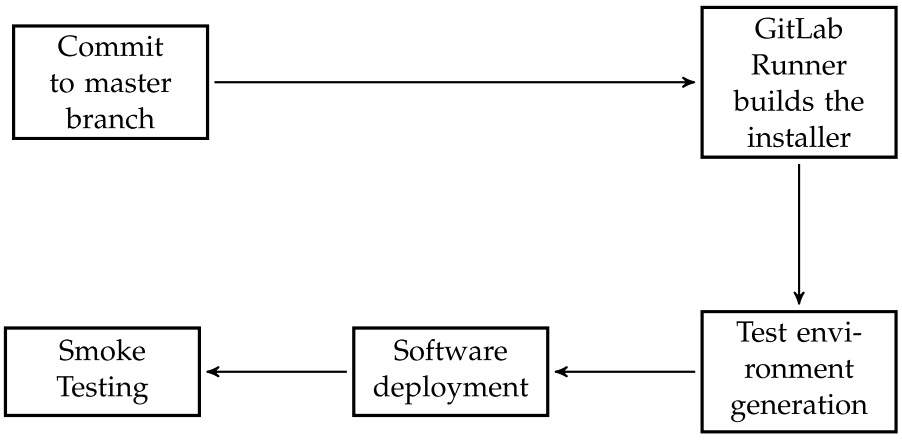

Commit to master branch
GitLab Runner builds the installer
Test environment generation
Software deployment
Smoke Testing

Figure 2.3: CI Pipeline

In order to support this pipeline:

- GitLab has Windows Runners and CentOS Runners to be able to build the solution.
- Jenkins has Jenkins slave nodes to be able to create the needed virtual machines and deploy the software.

2.2 Robot Framework

Figure 2.4: Robot Framework logo

Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD)[7]. In the scope of this dissertation, *Robot Framework* is used for writing tests for:

1. *Sanity check* of a generated release.
2. *Build verification testing* on a deployed component.
3. *Integration testing* on a deployed environment.

A test suite in robot looks like this:

```
*** Settings ***
Library    SSHLibrary    360 seconds
Library    OperatingSystem
Library    String
Resource    LinuxCommon
Suite Setup        Open Connection And Log In
Suite Teardown     Close All Connections

*** Variables ***
${HOST}            1.2.3.4
${USERNAME}        user
${PASSWORD}        password
${PORT}            22

*** Test Cases ***
PostgreSQL service file exists
    SSHLibrary.File Should Exist    /etc/systemd/system/postgresql-9.6.service

PostgreSQL service is up
    ${output}    ${rc}=    SU And CMD    systemctl status postgresql-9.6.service
    Should Be Equal    ${rc}    0

PgBouncer service is up
    ${output}    ${rc}=    SU And CMD    systemctl status pgbouncer.service
    Should Be Equal    ${rc}    0

Port 5432 (PostgreSQL) is open
    ${output}    ${rc}=    SU And CMD    firewall-cmd --list-ports | grep 5432/tcp
    Should Be Equal    ${rc}    0

Port 6432 (PgBouncer) is open
    ${output}    ${rc}=    SU And CMD    firewall-cmd --list-ports | grep 6432/tcp
    Should Be Equal    ${rc}    0
```

The robot test suite almost resembles a tabular test definition. The file is divided into:

[7] http://robotframework.org/

2.2.1. Settings

In the *Settings* section we load any robot library we need with the keyword *Library* if it is an installed library, or *Resource* if it is a *robot* file with variables or *keywords* of our own. A *keyword* is equivalent to a function.

We can also define steps to be executed at the before executing any test (*Suite Setup*) and/or at the end of the execution (*Suite Teardown*).

2.2.2. Variables

Here can be defined the needed variables instead of loading them in the *Settings* section.

2.2.3. Test Cases

The *Test Cases* section is where all the tests are defined. The indentation is important as each test has to begin with no indentation, and more than one space signifies a column delimitation. Another way of delimiting columns is with the character '|'. For example, in the following test case:

```
PostgreSQL service is up
    ${output}    ${rc}=    SU And CMD    systemctl status
   postgresql-9.6.service
    Should Be Equal    ${rc}    0
```

Line 1

`PostgreSQL service is up` is the title of the test case.

Line 2

`${output}    ${rc}=` is the first column. It stores the output and return code of the second's column keyword.

`SU And CMD` is the second column. In this case is a custom keyword that changes into root user and executes the command provided in the third column.

`systemctl status postgresql-9.6.service` is the parameter provided to the keyword at the second column.

Line 3

`Should Be Equal` is a standard keyword that compares its two arguments (second and third column).

`${rc}` is a variable containing the return code of the command executed at the first line.

`0` is what we are comparing against.

2.3 VMware Realize Orchestrator

Figure 2.5: VMware Realize Orchestrator logo

VMware vRealize^{TM} Orchestrator^{TM} [...] is a drag-and- drop workflow software that simplifies the automation of complex IT tasks. [...] It enables administrators to develop complex automation tasks, then quickly access and launch workflows from the VMware vSphere client, various components of VMware vCloud Suite, or other triggering mechanisms.[8]

As all Maptics solutions are deployed with a VMware Sphere license, VMware Realize Orchestrator is a natural choice. It is used to deploy the official Fielding Production OVAs and provision them as needed.

2.4 Jenkins

Figure 2.6: Jenkins logo

Jenkins is an open source orchestrator tool that can be used to automate tasks. It has support for plugins allowing it to interact with a great range of services and applications. Specifically, in the Maptics scenario is used to interact with:

1. *Team Foundation Server*, to download the *Robot Framework* tests.
2. *GitLab*, to download the latest installers.
3. *VMware Realize vOrchestrator*, to deploy and configure new virtual machines.
4. *Robot Framework*, to execute the tests.

[8] http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vrealize/vmware-vrealize-orchestrator.pdf

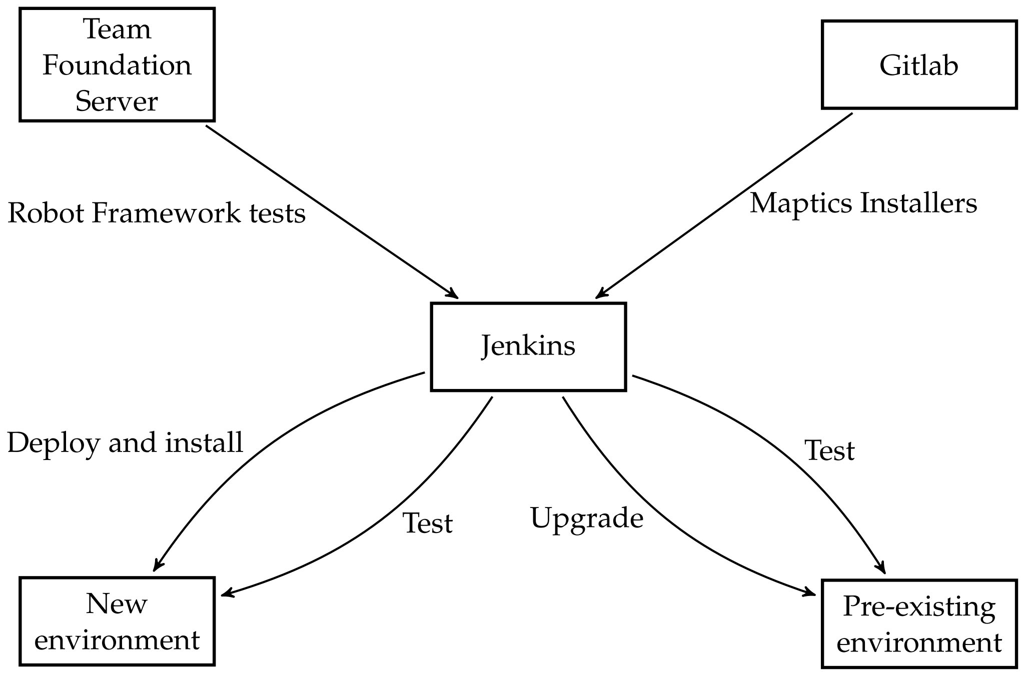


Figure 2.7: Jenkins interactions

CHAPTER 3

Maptics Architecture

One of the services *Fielding* provides is geolocation data analysis through its *Maptics* solution. Tailoring the social media input (Twitter, Facebook, Instagram) to the client's needs, the system is capable of capture, analyze and geolocate the effect of trends, hashtags and brand presence.

Maptics makes use of an array of components to achieve its goal. The solution architecture is as follows:

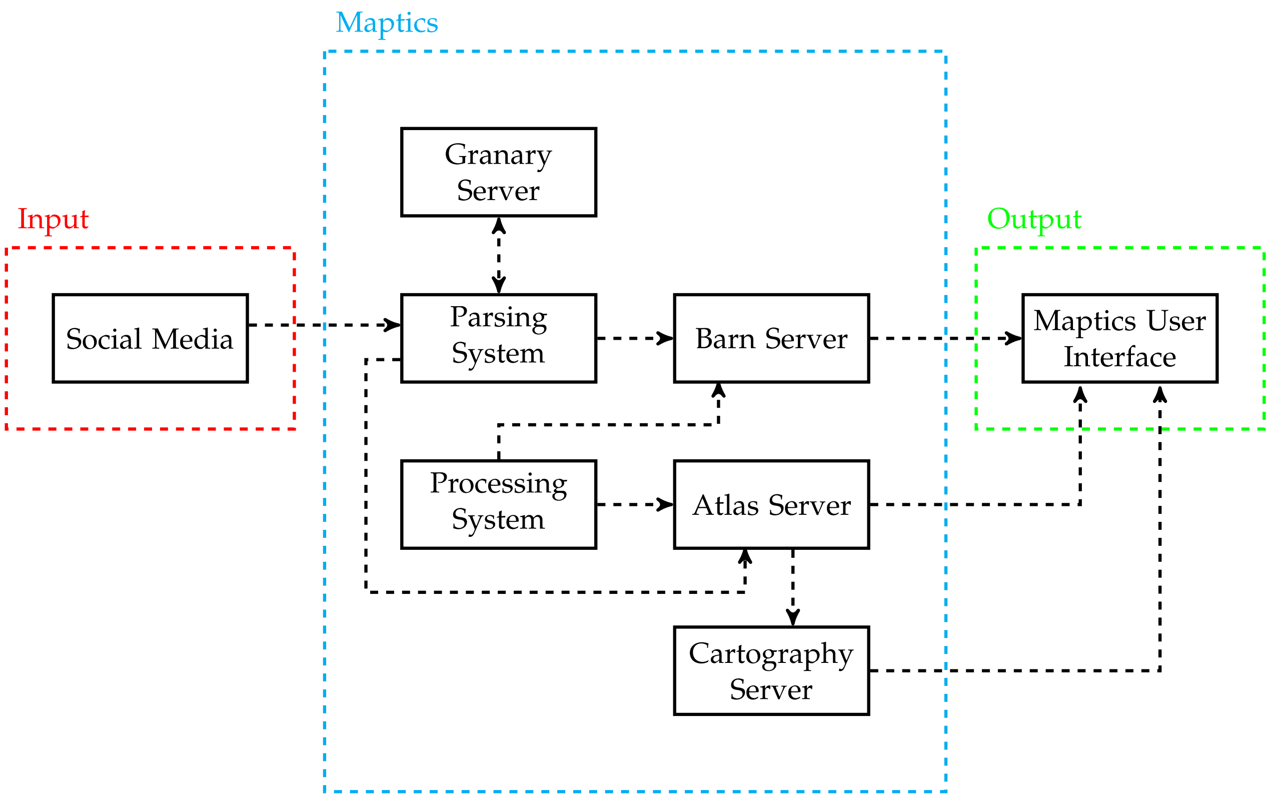


Figure 3.1: Maptics Solution Architecture

The idea of this processing pipeline is to capture all relevant social media interactions, be it via hashtags or keywords, measure and geolocalize it in order to be able to analyze it later. The *Maptics Parsing System* is the one to parse all this input, inserting the relevant data into the *Barn* server (raw data) and into the *Atlas* server (geolocalization data), using the *Granary* server as an intermediate repository. Later on, the *Maptics Processing System* applies algorithms to extract information from the raw data, and processes the geolocalization data for it to be visualized later via the *Cartography* server. An end user only has to have installed the *Maptics User Interface* to access the processed data.

3.1 Maptics Components

The full suite of applications follow a common versioning scheme as the whole range of components is generated at the same time. Through this dissertation we will talk about:

3.1.1. Maptics Parsing System

The *Maptics Parsing System* is the component tasked with parsing the raw data extracted from the tailored social media feed (twitter, facebook, instagram...) and inserting it in the corresponding database: *Barn* server for the raw data and *Atlas* server for the geolocalized data.

3.1.2. Granary

The *Granary* server consists of a Redis[1] database acting as an in-memory data structure store supporting the parsers. It usually stores intermediate results that the *Maptics Parsing System* can retrieve faster than retrieving it from the *Barn* or *Atlas* server.

3.1.3. Maptics Processing System

The *Maptics Processing System* executes maintenance tasks and algorithms on the available data. The algorithm executions result on what the final user can see in the end, like geolocalized data and trend analysis of defined keywords and hashtags.

3.1.4. Barn

The *Barn* server hosts a MSSQL server database which holds the majority of the system's data (all but map generation data). Most of the space is ocuppied by raw data, later to be transformed by the *Maptics Processing System*. This server hosts also a web-based administration interface of the solution (*Maptics Admin GUI*) and a monitoring tool (*Maptics Monitor*).

3.1.5. Atlas

The *Atlas* server hosts a PostgreSQL database which holds the map generation data. As with the *Barn* server, the *Maptics Parsing System* first processes the data and inserts it in the database, later to be transformed by the *Maptics Processing System*.

[1] *"Redis is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker"*. `http://redis.io/`

3.1.6. Cartography

The *Cartography* server publishes spatial information from the *Atlas* server data. It acts as a map provider for the final user, acting as an intermediary between the user and the *Atlas* server.

3.2 Requirements

Designing the deployment workflow of a software solution that is integrated in an larger array of components means that you have to follow guidelines in order to coexist. From path directories to deployment software frameworks, not only makes working with different components easier, it also creates a company image derived from the style and procedures.

This requirements can also extend to inside the software solution. Establishing and enforcing standards and guidelines is a way to ensure that all components follow good practices while avoiding miscommunication.

This section explains the requirements of the *Maptics* solution deployment scheme that has been developed for this Final Degree Work. In particular, the deployment solution implemented has to fulfill several guidelines for the storage of binaries and log files, installer structure and appearance and connectivity. The requirements have been structured in the following five subsections: Windows packaging, Linux packaging, deploy management, log management, and network interfaces.

3.2.1. Windows packaging

Files directories

By default the installation directory should be in `C:\Fielding\<product_name>`. This directory is named **%INSTDIR%**.

Directory	Description
%INSTDIR%/bin	Contains the set of binary product
%INSTDIR%/config	Contains the configuration
%INSTDIR%/data	Contains the data (if needed)
%INSTDIR%/doc	Contains the documentation (if needed)
%INSTDIR%/log	Contains LOG files

Table 3.1: Windows installation folders

If several products must share information, then files must be installed in **shared directory**. This directory is by default `C:\Fielding\shared`.

Product version and information

All information about installation must be written in an INI file, in the section [Install]. Other sections can be created to reference configuration of third party product (e.g. database connection information).

Section	Parameter Name	Description
Install	Version	Product version
Install	Date	Installation date
Install	Username	Name of the user who has installed the product
<thirdparty_software>	Version	Third party software version
<thirdparty_software>	Port	Third party software access port
<thirdparty_software>	AdminLogin	Login
<thirdparty_software>	AdminPwd	Encrypted password

Table 3.2: INI file sections and parameters

The INI file path should be `%INSTDIR%/install.ini`.

GUI

The order of the setup screens should be:

1. Welcome
2. License agreement
3. Install location
4. Choose components
5. Other parameters (choose IP address, additional folders...)
6. Installation progression
7. Finish

3.2.2. Linux packaging

Files directories

On CentOS Linux, the product files are distributed in multiple directories, respecting the Linux philosophy:

Directory	Description
/home/backup	Contains product backup
/var/log/fielding/<product_name>	Contains the configuration
/etc/systemd/system/	Contains service management scripts
/opt/fielding/<product_name>	Contains the binaries
/opt/fielding/<product_name>/scripts	Contains all the product scripts (path must be added to environment path)
/etc/fielding/<product_name>	Contains the configuration
/usr/local/fielding/data/<product_name>	Contains the data (if needed, it is often a mount point on a dedicated partition)

Table 3.3: Linux installation folders

Exceptions for log file:

- Log for third party product included:
  `/var/log/<ThirdParty_Product_name>/`
- Log for installation and uninstallation setup:
  `/var/log/fielding/<product_name>_install.log`

Product version and information

All information about installation must be written in an shared INI file, each component in its own section. Other sections can be created to reference configuration of third party product (e.g. database connection information).

Section	Parameter Name	Description
<Component_name>	Version	Product version
<Component_name>	Date	Installation date
<Component_name>	Version_<thirdparty_software>	Third party software version
<Component_name>	Port_<thirdparty_software>	Third party software access port

Table 3.4: Linux INI file sections and parameters

The INI file path should be: `/etc/fielding/fielding.ini`.

GUI

The order of the setup screens should be:

1. Welcome
2. License agreement
3. Install location
4. Choose components
5. Other parameters (choose IP address, additional folders...)
6. Installation progression
7. Finish

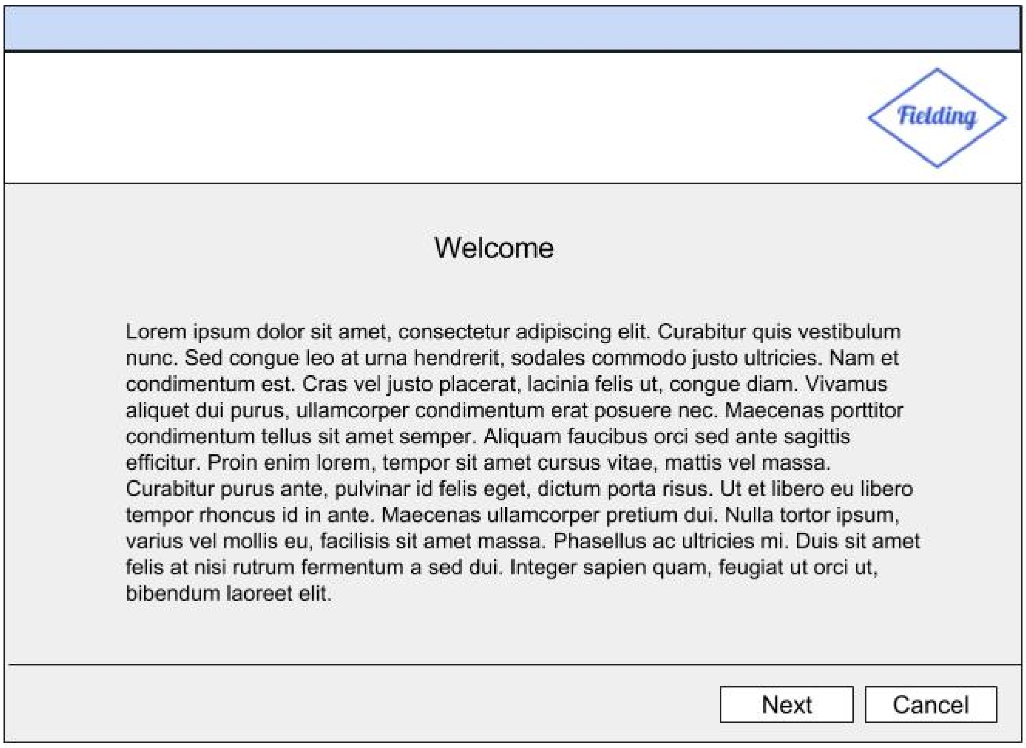
3.2.3. Deployment management

Graphical installer

Each component must present the operator with a graphical installer that asks all the information needed for the component to be installed correctly. How the installer will be built depends on the operating system:

- **Windows:** FIELD_SETUP is to be used. FIELD_SETUP consists of a modification on top of NSIS[2].

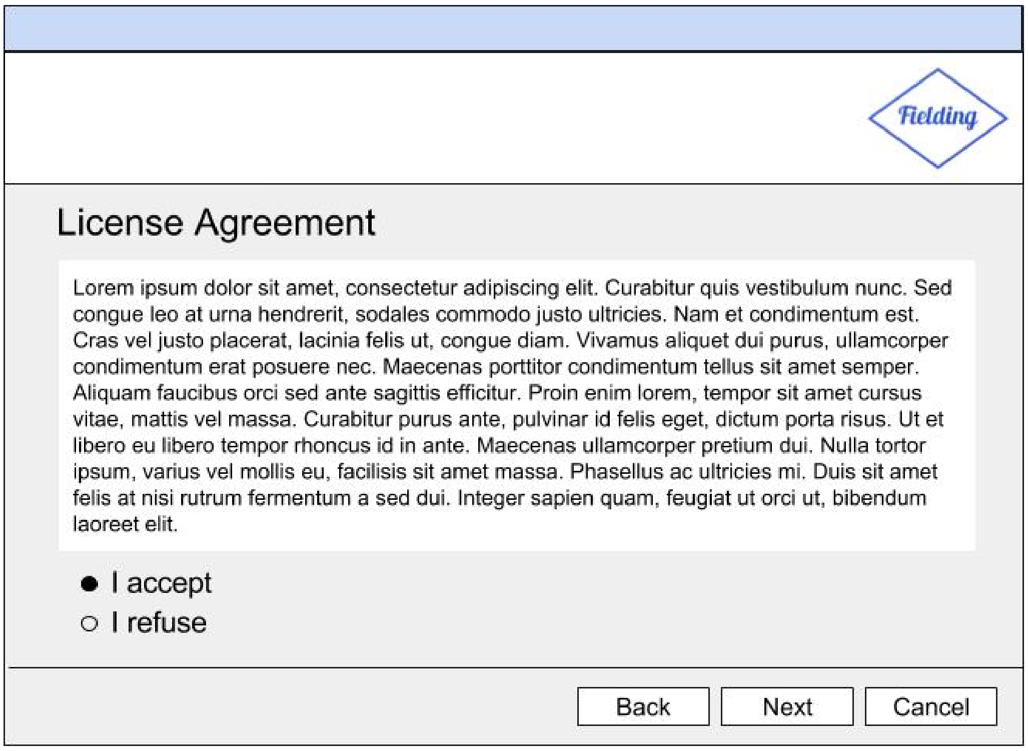
  As an example, the *Maptics Parsing System* installer is as follows:



Welcome

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur quis vestibulum nunc. Sed congue leo at urna hendrerit, sodales commodo justo ultricies. Nam et condimentum est. Cras vel justo placerat, lacinia felis ut, congue diam. Vivamus aliquet dui purus, ullamcorper condimentum erat posuere nec. Maecenas porttitor condimentum tellus sit amet semper. Aliquam faucibus orci sed ante sagittis efficitur. Proin enim lorem, tempor sit amet cursus vitae, mattis vel massa. Curabitur purus ante, pulvinar id felis eget, dictum porta risus. Ut et libero eu libero tempor rhoncus id in ante. Maecenas ullamcorper pretium dui. Nulla tortor ipsum, varius vel mollis eu, facilisis sit amet massa. Phasellus ac ultricies mi. Duis sit amet felis at nisi rutrum fermentum a sed dui. Integer sapien quam, feugiat ut orci ut, bibendum laoreet elit.

Figure 3.2: *Maptics Parsing System* installer: welcome screen



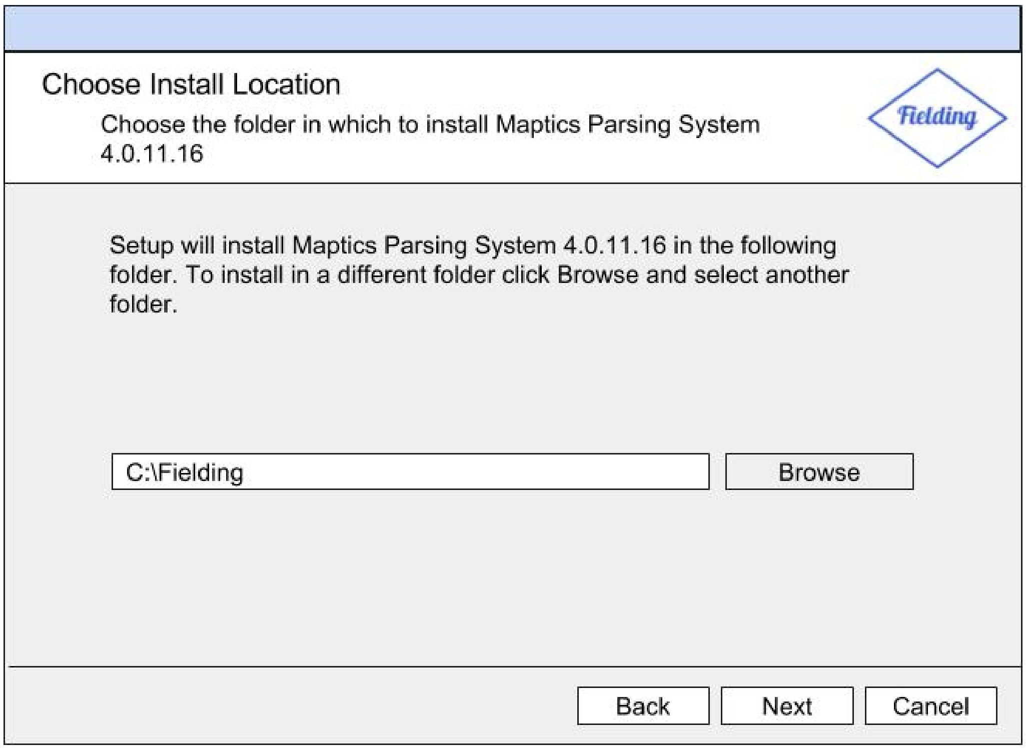
License Agreement

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur quis vestibulum nunc. Sed congue leo at urna hendrerit, sodales commodo justo ultricies. Nam et condimentum est. Cras vel justo placerat, lacinia felis ut, congue diam. Vivamus aliquet dui purus, ullamcorper condimentum erat posuere nec. Maecenas porttitor condimentum tellus sit amet semper. Aliquam faucibus orci sed ante sagittis efficitur. Proin enim lorem, tempor sit amet cursus vitae, mattis vel massa. Curabitur purus ante, pulvinar id felis eget, dictum porta risus. Ut et libero eu libero tempor rhoncus id in ante. Maecenas ullamcorper pretium dui. Nulla tortor ipsum, varius vel mollis eu, facilisis sit amet massa. Phasellus ac ultricies mi. Duis sit amet felis at nisi rutrum fermentum a sed dui. Integer sapien quam, feugiat ut orci ut, bibendum laoreet elit.

[✓ I accept] I refuse

Figure 3.3: *Maptics Parsing System* installer: license agreement screen

[2]NSIS, or Nullsoft Scriptable Install System is an open-source scriptable system to create Windows installers. More information on its website: `http://nsis.sourceforge.net/Main_Page`.

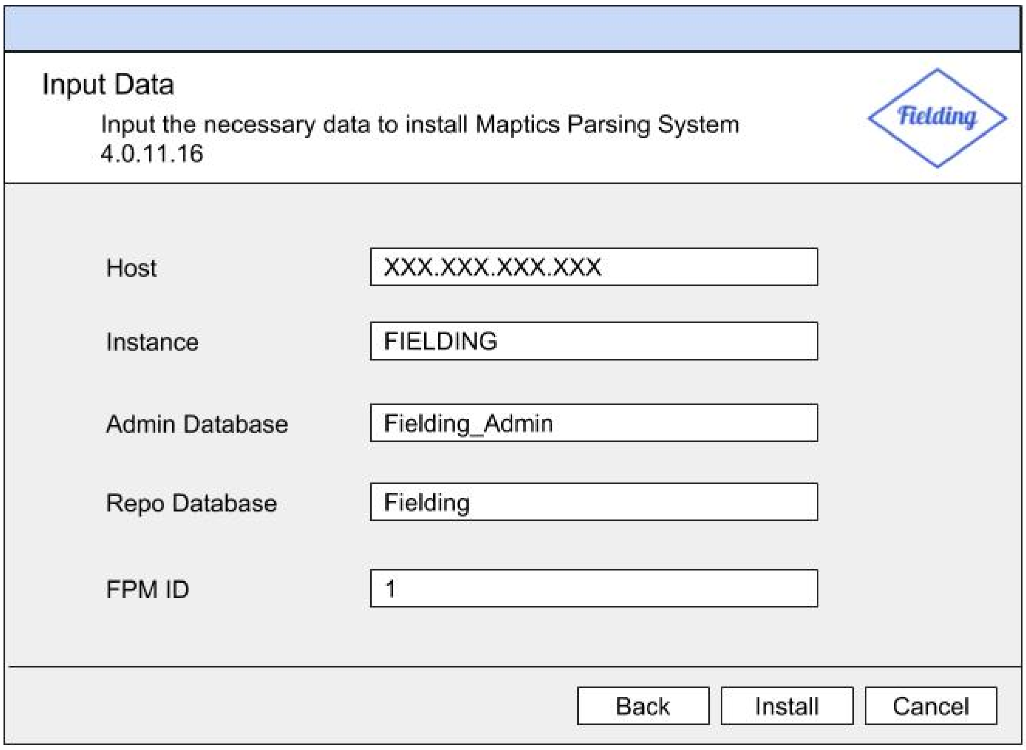

Choose Install Location

Choose the folder in which to install Maptics Parsing System 4.0.11.16

Setup will install Maptics Parsing System 4.0.11.16 in the following folder. To install in a different folder click Browse and select another folder.

C:\Fielding

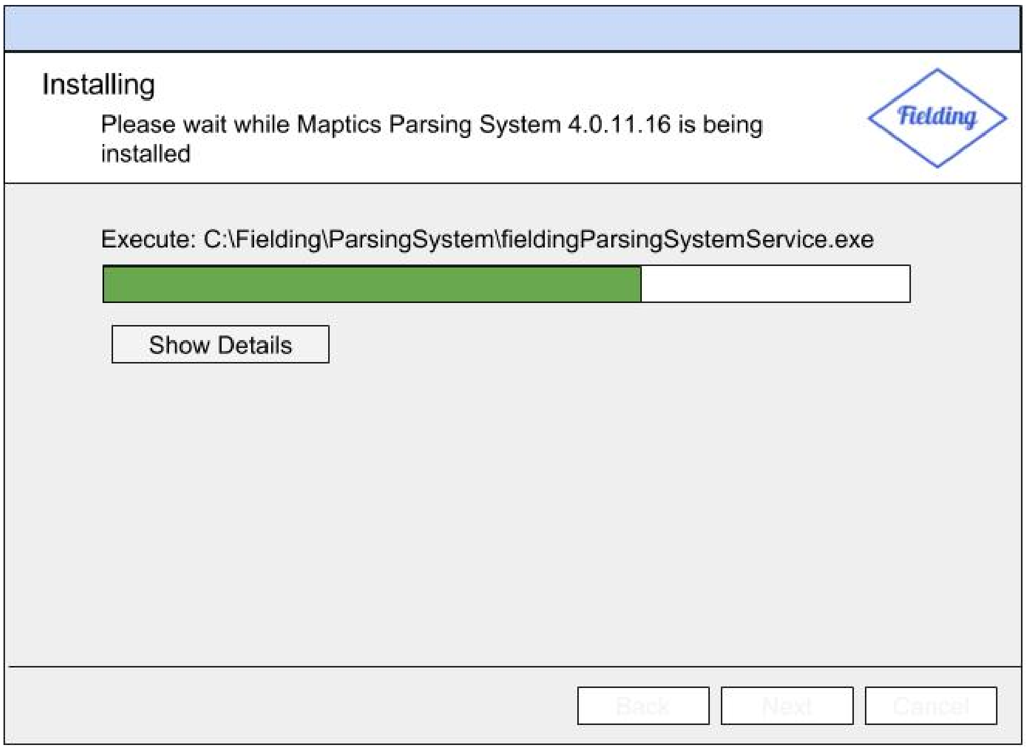
Figure 3.4: *Maptics Parsing System* installer: directory screen


Input Data

Input the necessary data to install Maptics Parsing System 4.0.11.16

Host	XXX.XXX.XXX.XXX
Instance	FIELDING
Admin Database	Fielding_Admin
Repo Database	Fielding
FPM ID	1

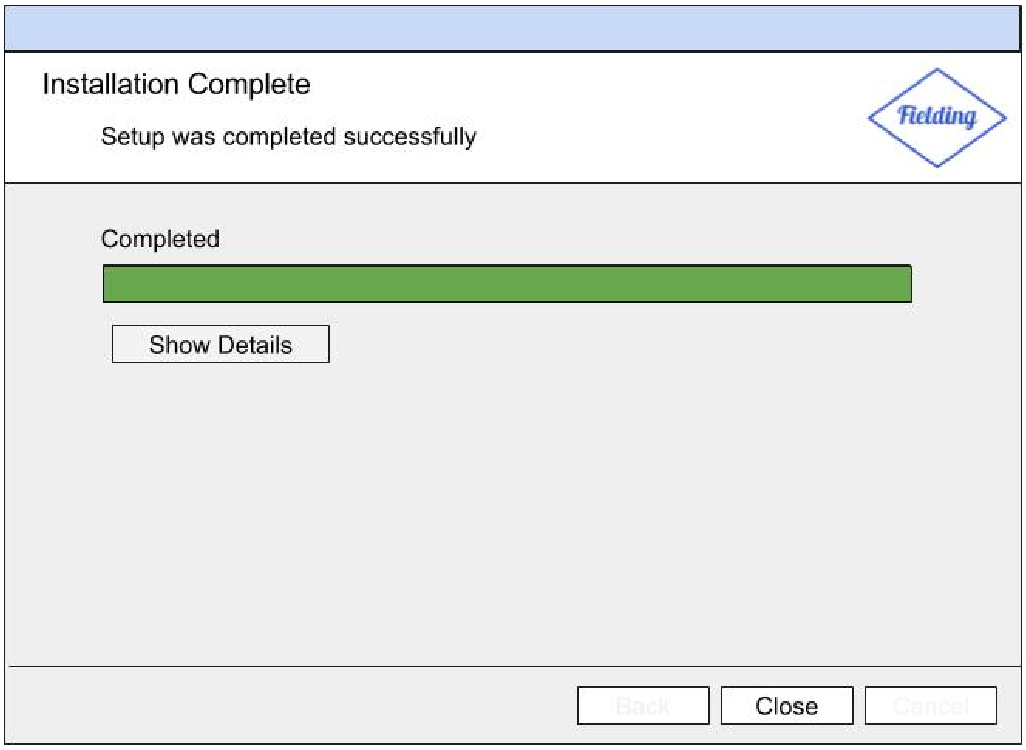
Figure 3.5: *Maptics Parsing System* installer: data screen


Installing

Please wait while Maptics Parsing System 4.0.11.16 is being installed

Execute: C:\Fielding\ParsingSystem\fieldingParsingSystemService.exe

Figure 3.6: *Maptics Parsing System* installer: installing screen


Installation Complete

Setup was completed successfully

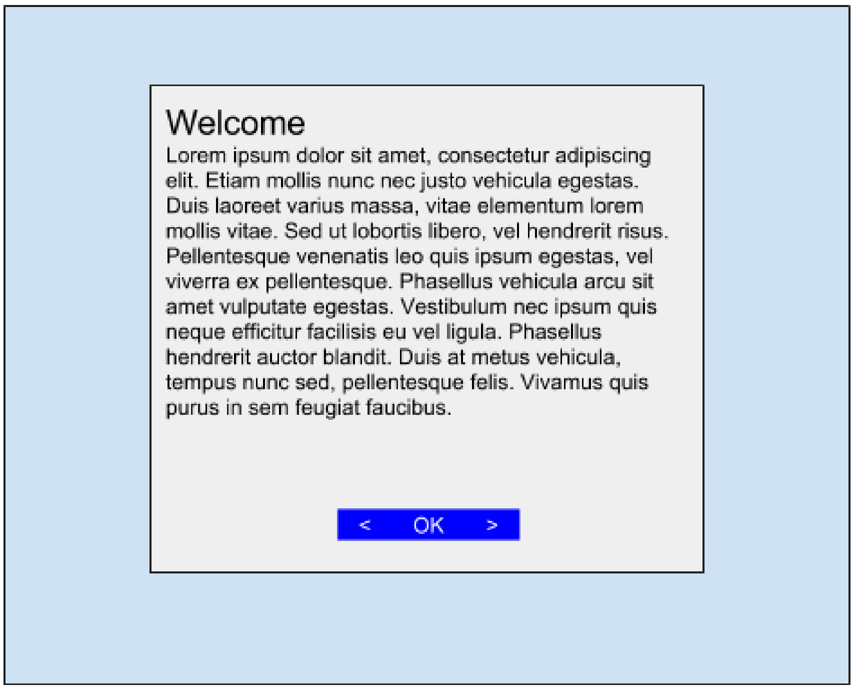
Completed

Figure 3.7: *Maptics Parsing System* installer: completed screen

A similar installer is provided for *Maptics Barn* server, *Maptics Processing System*, *Maptics Admin GUI* and *Maptics Monitor*.

- **Linux:** FIELD_SETUP_LINUX is to be used. FIELD_SETUP_LINUX consists of a library of shell scripts wrapping among others the *dialog*[3] utility (linux-based servers are always installed without graphical interface).

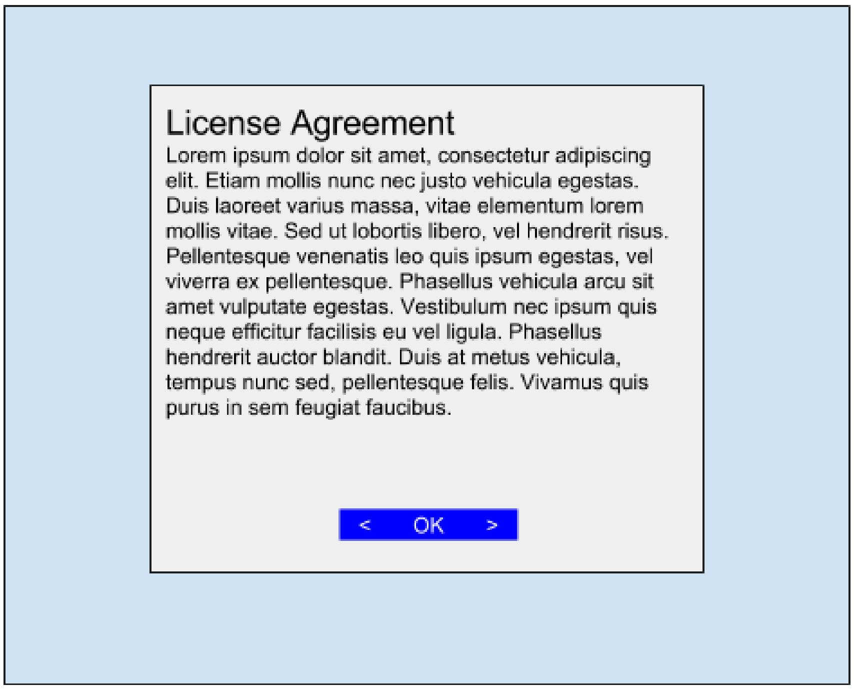
As an example, the *Maptics Cartography System* installer is as follows:



Welcome

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mollis nunc nec justo vehicula egestas. Duis laoreet varius massa, vitae elementum lorem mollis vitae. Sed ut lobortis libero, vel hendrerit risus. Pellentesque venenatis leo quis ipsum egestas, vel viverra ex pellentesque. Phasellus vehicula arcu sit amet vulputate egestas. Vestibulum nec ipsum quis neque efficitur facilisis eu vel ligula. Phasellus hendrerit auctor blandit. Duis at metus vehicula, tempus nunc sed, pellentesque felis. Vivamus quis purus in sem feugiat faucibus.

Figure 3.8: *Cartography* installer: welcome screen



License Agreement

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mollis nunc nec justo vehicula egestas. Duis laoreet varius massa, vitae elementum lorem mollis vitae. Sed ut lobortis libero, vel hendrerit risus. Pellentesque venenatis leo quis ipsum egestas, vel viverra ex pellentesque. Phasellus vehicula arcu sit amet vulputate egestas. Vestibulum nec ipsum quis neque efficitur facilisis eu vel ligula. Phasellus hendrerit auctor blandit. Duis at metus vehicula, tempus nunc sed, pellentesque felis. Vivamus quis purus in sem feugiat faucibus.

Figure 3.9: *Cartography* installer: license agreement screen

[3]More information about *dialog* in its man page: `http://man.he.net/?topic=dialog&section=all`.

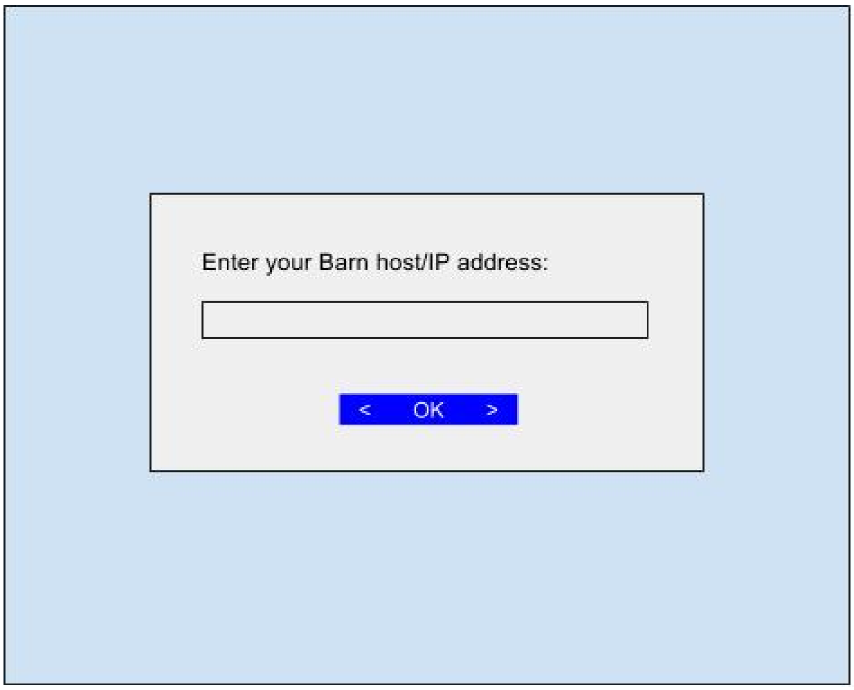

Enter your Barn host/IP address:

Figure 3.10: *Cartography* installer: Barn host screen

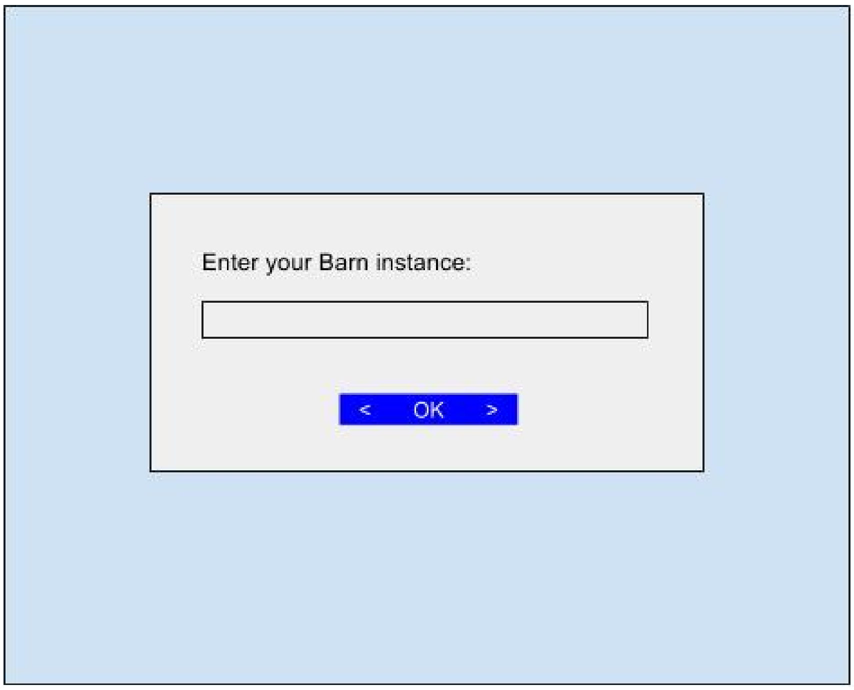

Enter your Barn instance:

Figure 3.11: *Cartography* installer: Barn instance screen

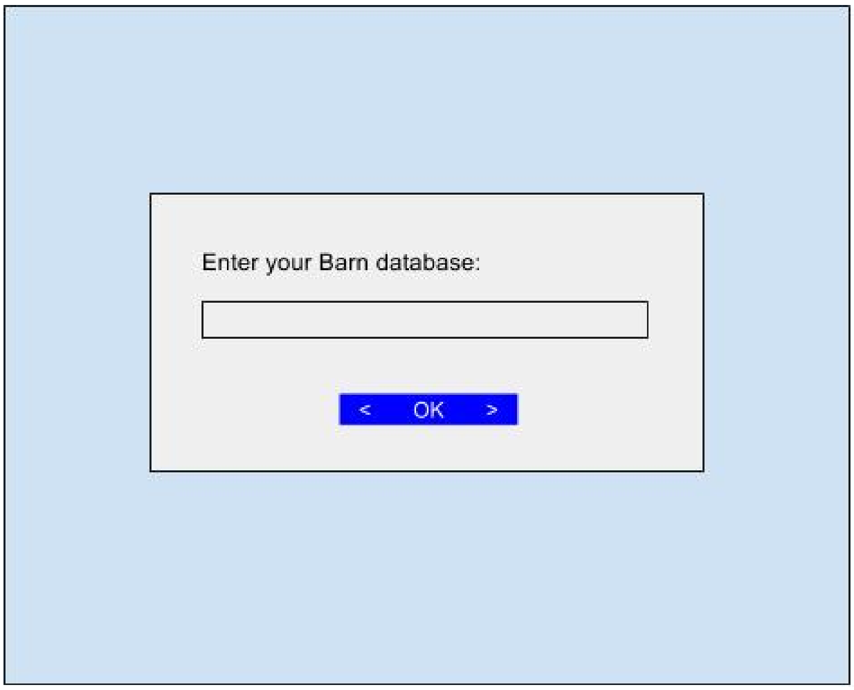

Enter your Barn database:

Figure 3.12: *Cartography* installer: Barn database screen

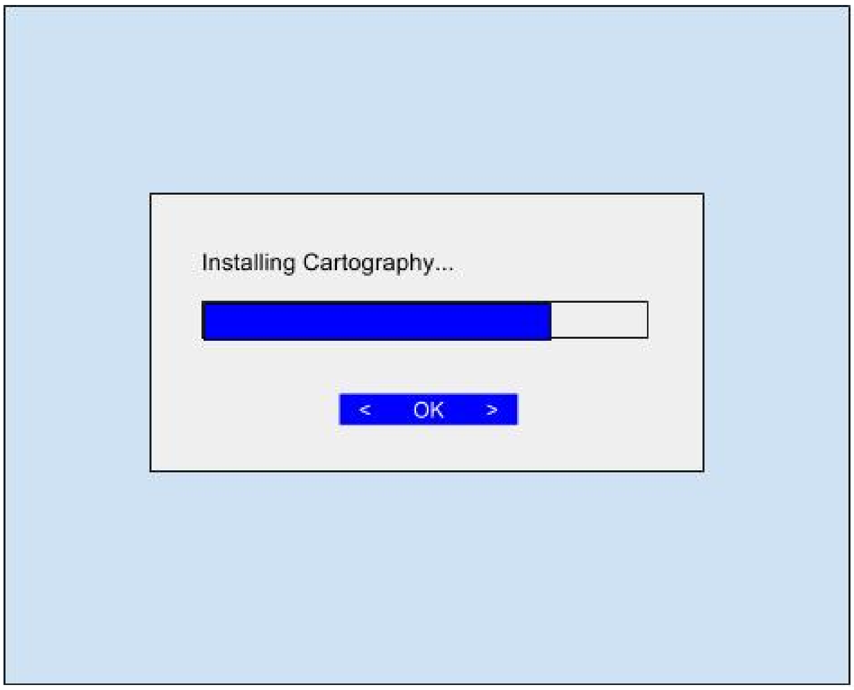

Installing Cartography...

Figure 3.13: *Cartography* installer: installation screen

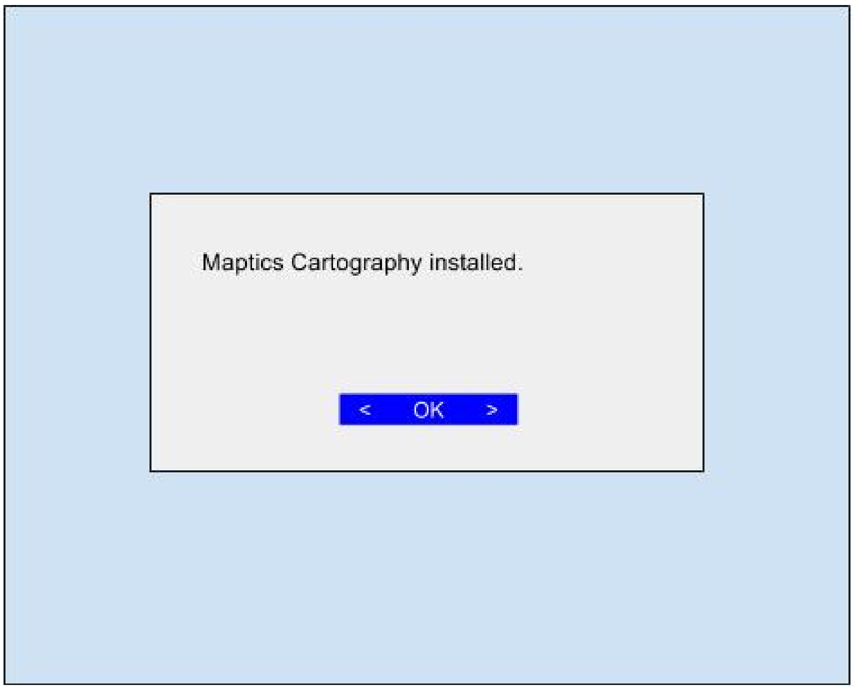

Maptics Cartography installed.

Figure 3.14: *Cartography* installer: completed screen

A similar installer is provided for *Maptics Atlas* server and *Maptics Granary* server.

Command-line parameters

Each installer must include a silent installation mode, denoted by the parameter /S in Windows installers and *-s* in Linux installers. Each component must provide the necessary extra parameters to install the product correctly without a GUI.

Update

The installer's behavior when the component is already installed should be as follows:

- The version installed is inferior or equal: an update is proposed.
- The version installed is superior: the installation is interrupted.

When executing the installer in silent mode, instead of asking the user if he wants to proceed with the update (first case), the update will be forced.

3.2.4. Log management

Each component, once it is installed is required to redirect the output of the day to day operations to a file log. The installer should generate a log with the same format. The log should be precise enough to be able to preserve the application's history. The mandatory fields are:

- **TIMESTAMP:** the date of the event to register
- **SERVERNAME:** the host executing the application
- **USER:** the user executing the application

- **APPLICATION:** name of the application generating the message
- **SEVERITY:** the severity of the event
- **MESSAGE:** the information being registered

The format should be as follows:

<TIMESTAMP> <SERVERNAME> <USER> <APPLICATION>: [<SEVERITY>] <MESSAGE>

The location of the installer's log file should be:

- **Windows:** `C:\Fielding\<Product_Name>\log`
- **Linux:** `/var/log/fielding`

3.2.5. Network Interfaces

In a *Maptics Solutions* deployment, three networks are defined: *Admin*, *Intra* and *User*. The *Admin* network is defined to carry administration tasks. The *Intra* network carries the internal server to server traffic. The *User* network is defined to provide external access to the servers that need it. By definition, all servers are connected to *Admin* and *Intra*, *User* network access is provided to each server that is accessed by the *Maptics User Interface*.

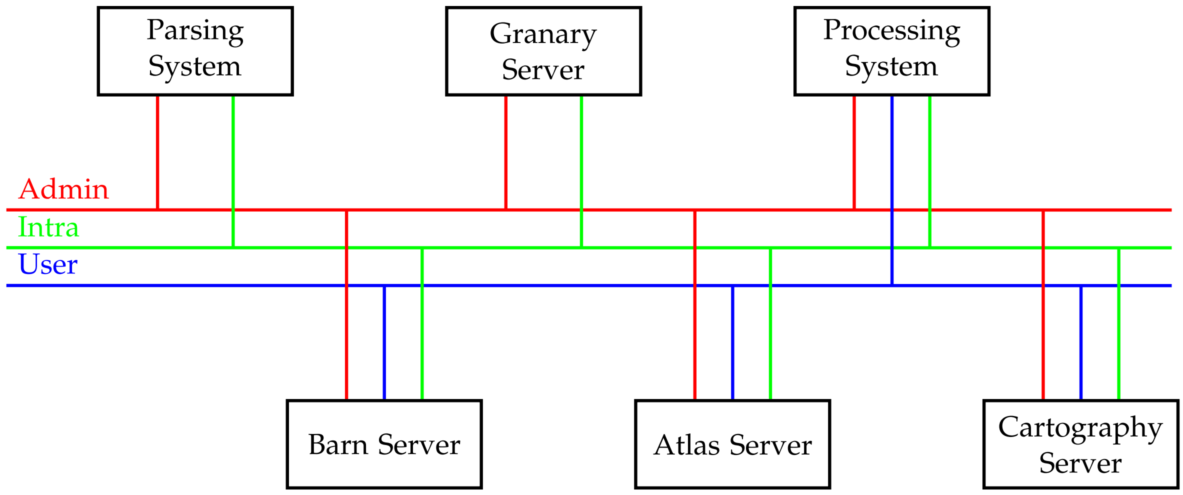


Figure 3.15: Network connectivity

CHAPTER 4

Deployment Solution

With all the requirements specified, the task was to rewrite the deployment process for each component while adhering to said requirements and making use of *continuous integration* practices including the development of testing suites which will be detailed on chapter 5, *Testing*.

While sometimes some of the steps performed in a component deployment corresponds to the installation of existing, third-party software, each procedure has been adapted to follow the requirements specified on section 3.2, *Requirements*. For every Linux component (*Atlas* server, *Cartography* server, *Granary* server), new service files have been created for third-party software in order to comply with the *Maptics* requirements. The service files were written for the init system of choice in *Fielding*, *systemd*. For the rest of the components no modifications were done to third-party software outside of what a custom unattended installation allows.

Following standards means that common practices will be applied to every component deployment. One of these practices is the use of an expressly created user (*geo_fielding*) for the execution of each service installed. This allows us to configure the user with the needed permissions. To this end, the creation of the *Maptics* users is the first step for each component. Another practices are: the use of shared paths for the storage of binaries, data and logs, a *Maptics* firewall zone for the Linux components, and a shared folder containing the registry keys for the Windows components.

It is important to note that while the deployment framework of choice for Windows components was *Nullsoft Scriptable Install System* (NSIS), some parts of the code were written in *Powershell* due to the advantages of the *cmdlets* available in the scripting language, the rest has been done in the *NSIS* scripting language. Powershell also allows us to leverage *.NET* if needed. For the Linux deployers, all the code was written in *Bash*.

The next sections explain what is the purpose of each component and its general structure. It is also specified the steps developed during this Degree Final Work to deploy the component. After all the deployment procedures have been explained, each section closes with the needed flags to be able to install the component unattendedly.

4.1 Maptics Parsing System

4.1.1. Introduction

The *Maptics Parsing System* parses the pre-tailored social media feed and stores the useful data into the *Barn* server or the *Atlas* server. It also uses the *Granary* server for short-term, ongoing calculations.

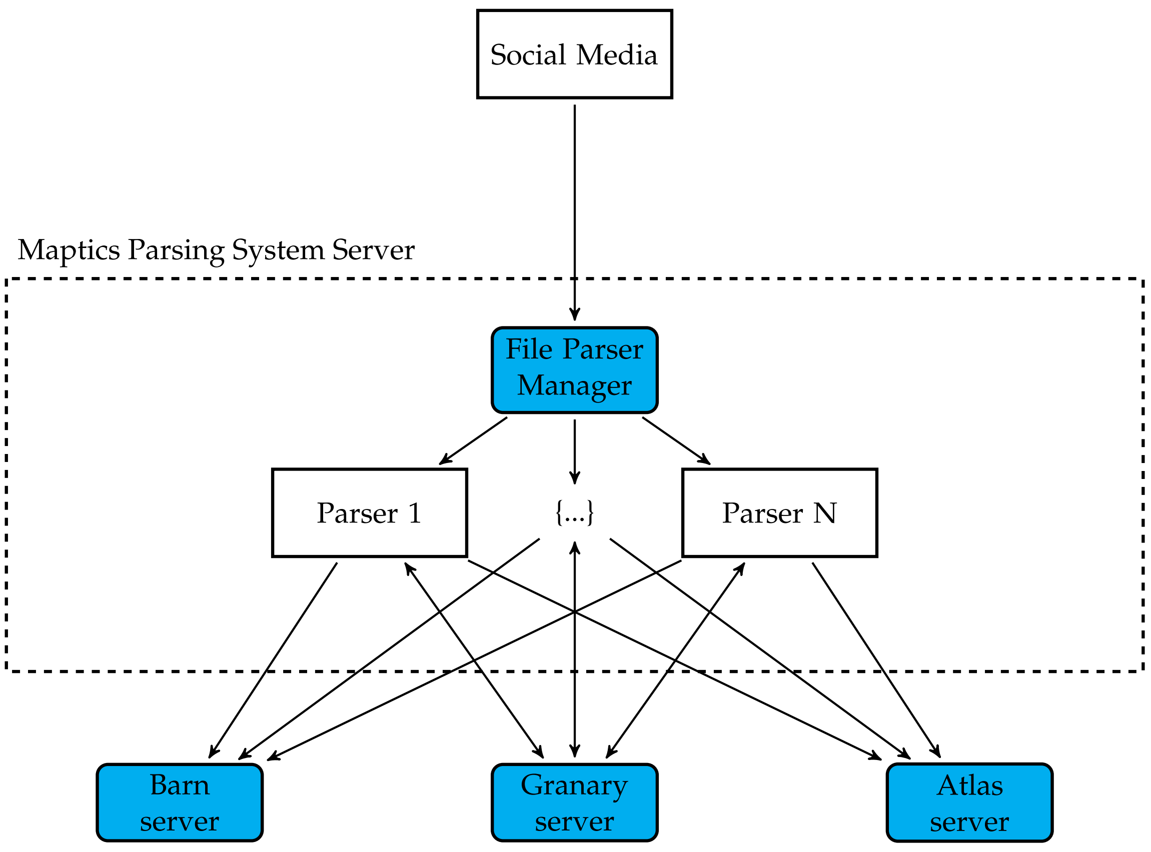


Figure 4.1: *Maptics Parser System* server diagram

The raw data of the messages goes into the *Barn* server, the geolocalization data goes into the *Atlas* server and the *Granary* server is used to store intermediate results while aggregating data.

4.1.2. Installation procedure

Algorithm 4.1: Maptics Parsing System installation

```
if Maptics Parsing System is not installed then
    Create needed Users and Groups
    Install MSMQ
    Deploy and configure Maptics Parsing System
else if Maptics Parsing System is installed and installed version is older or equal then
    Update Maptics Parsing System binaries
    Update Maptics Parsing System configuration
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *Configure-MapticsUsers* in listing A.1.

Install MSMQ

The installation of MSMQ (Message Queuing) is done with Powershell via the *Server Manager* module[8].

```
Import-Module ServerManager
Add-WindowsFeature MSMQ -WarningAction SilentlyContinue
```

Listing 4.1: *Maptics Parsing System* user creation

Deploy and configure Maptics Parsing System

To deploy the *Maptics Parsing System*:

1. Binary files have to be copied to C:\Fielding\Release\Monitor.
2. Connection strings have to be configured to point to the correct *Barn* MSSQL instance.

```
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Maptics FPM..." ${APP_NAME}
SetOutPath $INSTDIR
CreateDirectory $INSTDIR\FPM
SetOutPath "$INSTDIR\FPM"
File /a /r "FPM\"
File /a /r "Config Files Templates\FPM\"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Maptics FPM
    copied." ${APP_NAME}
#Configure FPM
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Maptics FPM..." ${APP_NAME}
!insertmacro ReplaceInFile "$INSTDIR\FPM\Maptics.ParserSystem.
    FileParserManager.exe.config" "<add key=$\"DataBaseServer$\"
    value=$\"XXX.XXX.XXX.XXX$\"/>" "<add key=$\"DataBaseServer$\"
    value=$\"$v_DataBaseServer$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\FPM\Maptics.ParserSystem.
    FileParserManager.exe.config" "<add key=$\"DataBaseInstance$\"
    value=$\"Maptics$\"/>" "<add key=$\"DataBaseInstance$\" value=$\
    "$v_DataBaseInstance$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\FPM\Maptics.ParserSystem.
    FileParserManager.exe.config" "<add key=$\"DataBaseName$\" value
    =$\"Maptics_Admin$\"/>" "<add key=$\"DataBaseName$\" value=$\"
    $v_DataBaseName$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\FPM\Maptics.ParserSystem.
    FileParserManager.exe.config" "<add key=$\"FPM_ID$\" value=$\"
    X$\"/>" "<add key=$\"FPM_ID$\" value=$\"$v_FPM_ID$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\FPM\NLog.config" "<variable
    name=$\"syslogserver$\" value=$\"[SYSLOG_SUBDOMAIN_OR_IP_ADDRESS
    ]$\"/>" "<variable name=$\"syslogserver$\" value=$\"
    $v_SyslogHost$\"/>"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Maptics FPM
    configured." ${APP_NAME}
```

```

!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Maptics Parser..." ${APP_NAME}
CreateDirectory $INSTDIR\Parser
SetOutPath "$INSTDIR\Parser"
File /a /r "Parser\"
File /a /r "Config Files Templates\Parser\"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Maptics Parser
    copied." ${APP_NAME}
#Configure Parser
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Maptics Parser..." ${APP_NAME}
!insertmacro ReplaceInFile "$INSTDIR\Parser\Maptics.ParserSystem.
    ParserContainer.exe.config" "<add key=$\"DataBaseServer$\" value
    =$\"XXX.XXX.XXX.XXX$\"/>" "<add key=$\"DataBaseServer$\" value=
    $\"$v_DataBaseServer$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\Parser\Maptics.ParserSystem.
    ParserContainer.exe.config" "<add key=$\"DataBaseInstance$\"
    value=$\"Maptics$\"/>" "<add key=$\"DataBaseInstance$\" value=$\
    "$v_DataBaseInstance$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\Parser\Maptics.ParserSystem.
    ParserContainer.exe.config" "<add key=$\"DataBaseName$\" value=
    $\"Maptics_Admin$\"/>" "<add key=$\"DataBaseName$\" value=$\"
    $v_DataBaseName$\"/>"
!insertmacro ReplaceInFile "$INSTDIR\Parser\NLog.config" "<variable
     name=$\"syslogserver$\" value=$\"[
    SYSLOG_SUBDOMAIN_OR_IP_ADDRESS]$\"/>" "<variable name=$\"
    syslogserver$\" value=$\"$v_SyslogHost$\"/>"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Maptics Parser
    configured." ${APP_NAME}
```

Listing 4.2: Deploy and configure *Maptics Parsing System*

4.1.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- **S:** Silent installation flag.
- **HOST:** The hostname or IP address of the MSSQL Server.
- **INSTANCE:** The name of the MSSQL instance.
- **ADMINDB:** The name of the MSSQL administration database.
- **REPODB:** The name of the MSSQL repository database.
- **FPM_ID:** The identifier of the File Parser Manager.

An example of an unattended install is as follows:

```
setupMapticsParsingSystem_v4.0.11.16.exe \S  \HOST=XXX.XXX.XXX.XXX
\INSTANCE=FIELDING \ADMINDB=Fielding_Admin \REPODB=Fielding \FPM_ID=1
\D=C:\Fielding
```

4.2 Granary Server

4.2.1. Introduction

The *granary* server, via *Redis*, provides an in-memory data structure store, used as a database cache. The *Maptics Parsing System* will use this server as an intermediate cache for short-term, ongoing calculations.

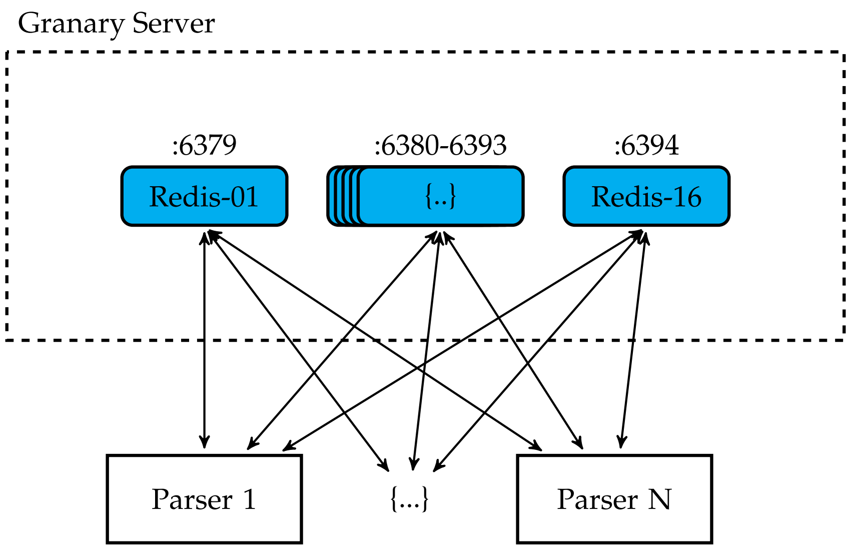


Figure 4.2: *Granary* server diagram

The number of *Redis* listeners is based on available CPU cores and memory:

```
RedisListenersCPU := (VirtualCores) / 3
RedisListenersMemory := (AvailableMemoryInGB / 5)
RedisListeners := MIN (RedisListenersCPU, RedisListenersMemory)
```

Each Redis listener should have available 3 virtual cores and 5GB of memory. The listeners per CPU and Memory are computed and the lesser number of the two (ideally they are the same number if the dimensioning of the machine is correct) is chosen.

4.2.2. Installation procedure

Algorithm 4.2: Granary server installation

```
if Granary is not installed then
    Create needed Users and Groups
    Install Redis
    Configure Redis listeners according to the system hardware
    Configure firewall
else if Granary is installed and installed version is older or equal then
    Update Redis configuration
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *maptics_setup_create_geo_fielding_user* in listing A.7.

Install Redis

To deploy the *Granary* server:

1. Binary files have to be copied to `/opt/fielding/granary`.
2. Data folder has to be configured to be `/usr/local/fielding/data/granary`.
3. Configuration files have to be copied to `/etc/redis`.
4. Service files have to be copied to `/etc/systemd/system`.

```
granary_setup_create_binaries_folders() {
  field_tolog_section "** Creating binaries folders **"
  mkdir -p /etc/redis
  if [ -d "/etc/redis" ]; then
    field_tolog "/etc/redis created successfully."
  else
    field_tolog_error "Error while creating /etc/redis folder."
  fi
  mkdir -p /opt/fielding/granary/bin
  if [ -d "/opt/fielding/granary/bin" ]; then
    field_tolog "/opt/fielding/granary/bin created successfully."
  else
    field_tolog_error "Error while creating /opt/fielding/granary/
     bin."
  fi
}

granary_setup_create_data_folders() {
  field_tolog_section "** Creating data folders  **"
  for IDPROC in `seq 01 16`; do
    IDPROCVar=$IDPROC
    if [ "$IDPROCVar" -lt 10 ]; then
      IDPROCVar='0'$IDPROCVar
    fi
    mkdir -p /usr/local/fielding/data/granary/redis-$IDPROCVar
  done
  chown -R geo_fielding:FIELD_GEO /usr/local/fielding/data/granary
}

granary_setup_copy_binaries() {
  field_tolog_section "** Copying binaries  **"
  cp $FIELD_SCRIPT_DIR/c72/01_redis/redis-2.8.19/src/redis-server /
   opt/fielding/granary/bin/redis-server
  if [ -e "/opt/fielding/granary/bin/redis-server" ]; then
    chmod +x /opt/fielding/granary/bin/redis-server
    field_tolog "/opt/fielding/granary/bin/redis-server copied
     successfully."
  else
    field_tolog_error "Error while copying /opt/fielding/granary/
     bin/redis-server file."
  fi
  cp $FIELD_SCRIPT_DIR/c72/01_redis/redis-2.8.19/src/redis-cli /opt
   /fielding/granary/bin/redis-cli
  if [ -e "/opt/fielding/granary/bin/redis-cli" ]; then
    field_tolog "/opt/fielding/granary/bin/redis-cli copied
   successfully."
    chmod +x /opt/fielding/granary/bin/redis-cli
  else
    field_tolog_error "Error while copying /opt/fielding/granary/
   bin/redis-cli file."
  fi
}

granary_setup_copy_conf_files(){
  field_tolog_section "** Copying configuration files  **"
  cp $FIELD_SCRIPT_DIR/c72/01_redis/redis-2.8.19/deployment/conf/
   redis-??.conf /etc/redis/
  if [ -e "/etc/redis/redis-01.conf"  ]; then
    field_tolog "Configuration files copied successfully to /etc/
   redis."
  else
    field_tolog_error "Error while copying configuration files to /
   etc/redis."
  fi
  chown -R geo_fielding:FIELD_GEO /etc/redis
}

granary_setup_copy_conf_files(){
  field_tolog_section "** Copying configuration files  **"
  cp $FIELD_SCRIPT_DIR/c72/01_redis/redis-2.8.19/deployment/conf/
   redis-??.conf /etc/redis/
  if [ -e "/etc/redis/redis-01.conf"  ]; then
    field_tolog "Configuration files copied successfully to /etc/
   redis."
  else
    field_tolog_error "Error while copying configuration files to /
   etc/redis."
  fi
  chown -R geo_fielding:FIELD_GEO /etc/redis
}

granary_setup_copy_systemd_files() {
  field_tolog_section "** Copying systemd unit files  **"
  cp $FIELD_SCRIPT_DIR/c72/01_redis/redis-2.8.19/deployment/systemd
   /redis-??.service /etc/systemd/system/
  if [ -e "/etc/systemd/system/redis-01.service" ]; then
    field_tolog "Systemd unit files copied success"
  else
    field_tolog_error "Error while copying systemd unit files to /
   etc/systemd/system."
  fi
}

granary_setup_copy_script_files() {
  field_tolog_section "** Copying script files **"
  cp $FIELD_SCRIPT_DIR/c72/01_redis/redis-2.8.19/deployment/
   test_redis.sh /opt/fielding/granary/bin/test_redis.sh
  if [ -e "/opt/fielding/granary/bin/test_redis.sh" ]; then
    field_tolog "Script files copied successfully to /opt/fielding/
   granary/bin."
  else
    field_tolog_error "Error while copying script files to /opt/fielding/granary/bin."
  fi
  chown -R geo_fielding:FIELD_GEO /opt/fielding/granary
}


granary_setup_configure_system() {
  field_tolog_section "** Configuring system  **"
  if ! grep -q "overcommit_memory" /etc/sysctl.conf
  then
    cp /etc/sysctl.conf /etc/sysctl.conf.backup
    echo "vm.overcommit_memory = 1" >> /etc/sysctl.conf
  fi
  echo never > /sys/kernel/mm/transparent_hugepage/enabled
  mkdir -p /var/log/fielding/granary
  chown -R geo_fielding:FIELD_GEO /var/log/fielding/granary
}
```

Listing 4.3: Install Redis

Configure Redis listeners according to the system hardware

The services to be enabled (out of sixteen) are configured automatically following the guidelines already explained:

```
granary_setup_enable_pertinent_listeners() {
  field_tolog_section "** Enable pertinent redis listeners **"
  # Calculate number of REDIS threads based on available CPU cores and RAM
  NumberCores=`cat /proc/cpuinfo | grep processor | wc -l`
  AvailMemory=`cat /proc/meminfo | awk '/^MemTotal/{print $2}'`
  NumberThreadsC=$(expr $NumberCores \* 2 / 3)
  NumberThreadsM=$(expr $AvailMemory / 10000000)
  NumberThreads=$NumberThreadsC
  if [ "$NumberThreadsM" -lt "$NumberThreads" ]; then
    NumberThreads=$NumberThreadsM
  fi
  if [ "$NumberThreads" -gt 16 ]; then
    NumberThreads=16
  fi
  if [ "$NumberThreads" -lt 1 ]; then
   echo '################################################' 2>&1 | field_tolog_pipeline
   echo 2>&1 | field_tolog_pipeline
   echo '     Number Cores available =' $NumberCores 2>&1 | field_tolog_pipeline
   echo '     Memory available =' $AvailMemory 'kB' 2>&1 | field_tolog_pipeline
   echo 2>&1 | field_tolog_pipeline
   echo '     This server doesn't have enough resources to run granary 2>&1 | field_tolog_pipeline
   echo 2>&1 | field_tolog_pipeline
   echo '################################################' 2>&1 | field_tolog_pipeline
  else
   REDIS_LISTENERS=$NumberThreads
```

```
    for IDPROC in `seq 01 $NumberThreads`; do
     IDPROCVar=$IDPROC
     if [ "$IDPROCVar" -lt 10 ]; then
       IDPROCVar='0'$IDPROCVar
     fi
       systemctl enable redis-$IDPROCVar 2>&1 | field_tolog_pipeline
       systemctl start redis-$IDPROCVar 2>&1 | field_tolog_pipeline
    done
    echo '##################################################' 2>&1 |
    field_tolog_pipeline
    echo '###                  IMPORTANT                ###' 2>&1 |
    field_tolog_pipeline
    echo '##################################################' 2>&1 |
    field_tolog_pipeline
    echo 2>&1 | field_tolog_pipeline
    echo '     Number Cores available =' $NumberCores 2>&1 |
    field_tolog_pipeline
    echo '     Memory available =' $AvailMemory 'kB' 2>&1 |
    field_tolog_pipeline
    echo '     Number of REDIS servers =' $NumberThreads 2>&1 |
    field_tolog_pipeline
    echo 2>&1 | field_tolog_pipeline
    echo '     Make sure that you configure exactly' $NumberThreads
    'granary servers in your Maptics XML config file.' 2>&1 |
    field_tolog_pipeline
    echo 2>&1 | field_tolog_pipeline
    echo '##################################################' 2>&1 |
    field_tolog_pipeline
  fi

}
```

Listing 4.4: Configure Redis listeners according to the system hardware

Configure firewall

The ports 6379 to 6394 are open to allow access to the *Redis* listeners.

```
granary_setup_configure_firewall() {
  field_tolog_section "** Configuring firewall **"
  for PORT in `seq 6379 6394`; do
    firewall-cmd --permanent --zone=fielding --add-port=$PORT/tcp
   2>&1 | field_tolog_pipeline
  done
  firewall-cmd --reload 2>&1 | field_tolog_pipeline
}
```

Listing 4.5: Configure *Maptics Granary* firewall

4.2.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- **-s:** Silent installation flag.

An example of an unattended install is as follows:

```
setupGranary_v4.0.11.16.bin -s
```

4.3 Maptics Processing System

4.3.1. Introduction

The *Maptics Processing System* processes the already parsed and stored data in the *Barn* and *Atlas* servers. The processor executes a set of algorithms over the existing data, be it daily or on-demand. The results are later accessible through the *Maptics GUI*.

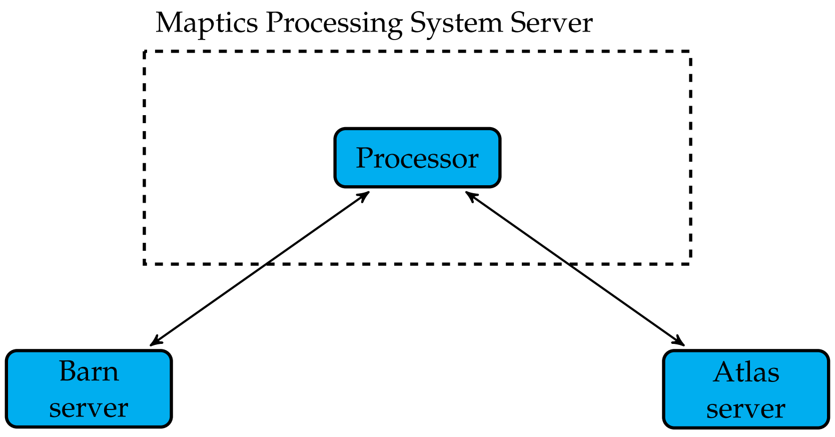


Figure 4.3: *Maptics Processing System* server diagram

4.3.2. Installation procedure

Algorithm 4.3: Maptics Processing System installation

```
if Maptics Processing System is not installed then
    Create needed Users and Groups
    Install GDAL libraries
    Deploy Maptics Processing System binaries
    Deploy SRTM Files
else if Maptics Processing System is installed and installed version is older or equal then
    Update Maptics Processing System binaries
    Update Maptics Processing System configuration
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *Configure-MapticsUsers* in listing A.1.

Install GDAL libraries

GDAL is a translator library for raster and vector geospatial data formats that is released under an X/MIT style Open Source license by the Open Source Geospatial Foundation[11]. The *GDAL* libraries can be installed silently:

```
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Installing GDAL
    libraries..." ${APP_NAME}
SetOutPath $INSTDIR
```

```
File gdal\gdal-111-1600-x64-core.msi
ExecWait '"msiexec" /i "$INSTDIR\gdal-111-1600-x64-core.msi" /qn
    IACCEPTSQLNCLILICENSETERMS=YES'
Delete $INSTDIR\gdal-111-1600-x64-core.msi
${EnvVarUpdate} $0 "PATH" "A" "HKLM" "C:\Program Files\GDAL"
${EnvVarUpdate} $0 "GDAL_DATA" "A" "HKLM" "C:\Program
    Files\GDAL\gdal-data"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "GDAL-DATA
    environment variable created." ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "GDAL libraries
    installed." ${APP_NAME}
```

Listing 4.6: Install GDAL libraries

They are used to draw overlays on the *Maptics UI* with the results of the algorithms.

Deploy *Maptics Processing System* binaries

To deploy the *Maptics Processing System*:

1. Binary files have to be copied to `C:\Fielding\Release\Processor`.
2. Connection strings have to be configured to point to the correct *Barn* MSSQL instance.

```
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Copying
    Processor binaries..." ${APP_NAME}
SetOutPath $INSTDIR
CreateDirectory $INSTDIR\Processor
SetOutPath "$INSTDIR\Processor"
File /a /r "Processor\"
File /a /r "Config Files Templates\Processor\"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Binaries copied.
    " ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Processor..." ${APP_NAME}
#Configure Processor
!insertmacro ReplaceInFile "$INSTDIR\Processor\Maptics.Processor.
    exe.config" "<add name=$\"fieldingDatabaseConnectionString$\"
    connectionString=$\"Data Source=XXX.XXX.XXX.XXX\Maptics;Initial
    Catalog=Maptics;Persist Security Info=True;User ID=gui;Password=
    password;Connect Timeout=30$\" providerName=$\"System.Data.
    SqlClient$\" />" "<add name=$\"
    fieldingDatabaseConnectionString$\" connectionString=$\"Data
    Source=$v_DataBaseServer\$v_DataBaseInstance;Initial Catalog=
    $v_RepoDataBaseName;Persist Security Info=True;User ID=gui;
    Password=password;Connect Timeout=30$\" providerName=$\"System.
    Data.SqlClient$\" />"
!insertmacro ReplaceInFile "$INSTDIR\Processor\Maptics.Processor.
    exe.config" "<add name=$\"
    fieldingAdminDatabaseConnectionString$\" connectionString=$\"
    Data Source=XXX.XXX.XXX.XXX\Maptics;Initial Catalog=
    Maptics_Admin;Persist Security Info=True;User ID=gui;Password=
    password;Connect Timeout=30;$\" providerName=$\"System.Data.
    SqlClient$\" />" "<add name=$\"
    fieldingAdminDatabaseConnectionString$\" connectionString=$\"
    Data Source=$v_DataBaseServer\$v_DataBaseInstance;Initial
    Catalog=$v_DataBaseName;Persist Security Info=True;User ID=gui;
    Password=password;Connect Timeout=30;$\" providerName=$\"System.
    Data.SqlClient$\" />"
!insertmacro ReplaceInFile "$INSTDIR\Processor\Maptics.Processor.
    exe.config" "<add name=$\"EntitiesAdmin$\" connectionString=$\"
    metadata=res://Maptics.Model/fieldingadmin.csdl|res://Maptics.
    Model/fieldingadmin.ssdl|res://Maptics.Model/fieldingadmin.msl;
    provider=System.Data.SqlClient;provider connection string=&quot;
    data source=XXX.XXX.XXX.XXX\Maptics;initial catalog=
    Maptics_Admin;Persist Security Info=True;User ID=gui;Password=
    password;Connect Timeout=30;multipleactiveresultsets=True;App=
    EntityFramework&quot;$\" providerName=$\"System.Data.
    EntityClient$\" />" "<add name=$\"EntitiesAdmin$\"
    connectionString=$\"metadata=res://Maptics.Model/fieldingadmin.
    csdl|res://Maptics.Model/fieldingadmin.ssdl|res://Maptics.Model/
    fieldingadmin.msl;provider=System.Data.SqlClient;provider
    connection string=&quot;data source=
    $v_DataBaseServer\$v_DataBaseInstance;initial catalog=
    $v_DataBaseName;Persist Security Info=True;User ID=gui;Password=
    password;Connect Timeout=30;multipleactiveresultsets=True;App=
    EntityFramework&quot;$\" providerName=$\"System.Data.
    EntityClient$\" />"
${If} $v_SyslogHost != ""
  !insertmacro ReplaceInFile "$INSTDIR\Processor\Nlog.config" "<
    variable name=$\"syslogserver$\" value=$\"[
    SYSLOG_SUBDOMAIN_OR_IP_ADDRESS]$\"/>" "<variable name=$\"
    syslogserver$\" value=$\"$v_SyslogHost$\"/>"
${EndIf}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Processor
    configured." ${APP_NAME}
```

Listing 4.7: Deploy *Maptics Processing System* binaries

Deploy SRTM Files

The SRTM (Shuttle Radar Topography Mission)[12] files are used by the processor geolocalization algorithms. The folder structure is created through NSIS scripting:

```
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Checking SRTM
    folders..." ${APP_NAME}
CreateDirectory D:\Fielding\Maptics\SRTM
CreateDirectory D:\Fielding\Maptics\SRTM\Africa
CreateDirectory D:\Fielding\Maptics\SRTM\Australia
CreateDirectory D:\Fielding\Maptics\SRTM\Eurasia
CreateDirectory D:\Fielding\Maptics\SRTM\Islands
CreateDirectory D:\Fielding\Maptics\SRTM\North_America
CreateDirectory D:\Fielding\Maptics\SRTM\South_America
${if} $v_SRTMSource != ""
  !insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Deploying SRTM
     files..." ${APP_NAME}
  SetOutPath $INSTDIR
  File ExtractSRTM.ps1
  ${DisableX64FSRedirection}
  ExecWait 'C:\Windows\System32\WindowsPowershell\v1.0\powershell.
    exe -inputformat none -File "$INSTDIR\ExtractSRTM.ps1"  "
    $v_SRTMSource"'
  ${EnableX64FSRedirection}
  Delete ExtractSRTM.ps1
  !insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "SRTM files
    deployed." ${APP_NAME}
${endif}
```

Listing 4.8: Prepare SRTM folders

To extract each SRTM file the *System.IO.Compression.Filesystem* .Net library is leveraged from *Powershell* to reduce by half the extraction time needed by the NSIS compression libraries.

```
param(
    [string]$SRTMSource
)

Add-Type -A 'System.IO.Compression.FileSystem'
Write-Host "SRTM Source: $SRTMSource"
foreach ($region in (get-ChildItem $SRTMSource | where { $_.
    PSIsContainer } )){
  Remove-Item "D:\Fielding\Maptics\SRTM\$region\*" -Force
  Write-Host "Extracting $region..."
  foreach ($file in get-ChildItem (""+$SRTMSource+"\"+$region+"\*.
    zip")){
      [IO.Compression.ZipFile]::ExtractToDirectory($file, "D:\
    Fielding\Maptics\SRTM\$region")
        Write-Host -NoNewLine "."
  }
  Write-Host ""
}
```

Listing 4.9: Extract SRTM files

4.3.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- \S: Silent installation flag.
- \HOST: The hostname or IP address of the MSSQL Server.
- \INSTANCE: The name of the MSSQL instance.
- \ADMINDB: The name of the MSSQL administration database.

An example of an unattended install is as follows:

```
setupMapticsProcessingSystem_v4.0.11.16.exe \S  \HOST=XXX.XXX.XXX.XXX
\INSTANCE=FIELDING \ADMINDB=Fielding_Admin \D=C:\Fielding
```

4.4 Barn Server

4.4.1. Introduction

The *Barn* server hosts a MSSQL database which holds both the general data extracted from the social media input and the results of the algorithms applied to said data. The data is inserted by the *Maptics Parsing System* and processed by the *Maptics Processing System*.

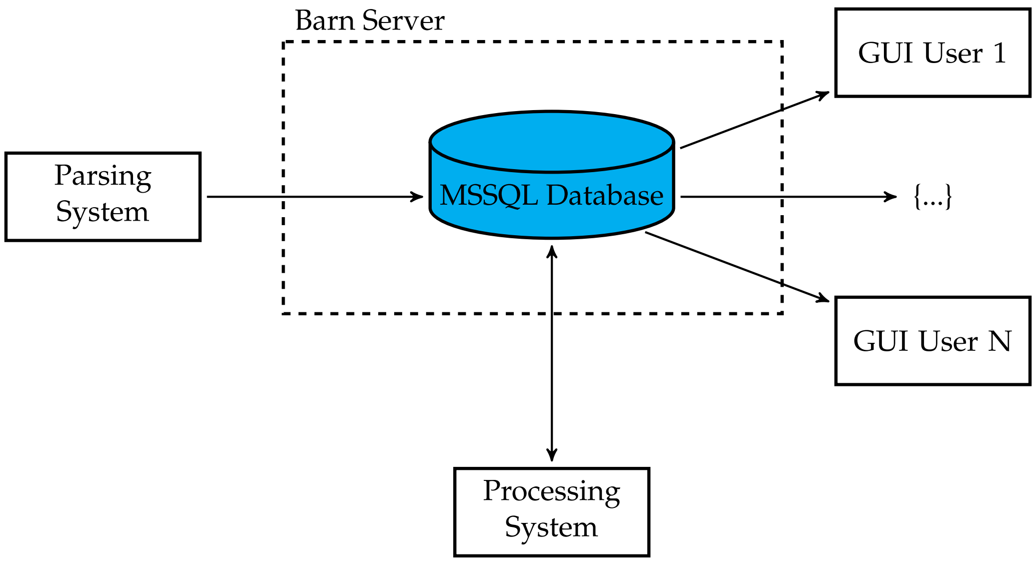


Figure 4.4: *Barn* server diagram

The *Maptics Parsing System* will be continuously inserting information into the MSSQL database and the *Maptics Processing System* will be executing algorithms and transformation over said data, storing the results in the same MSSQL Database. The final user accesses all this information through the *Maptics GUI*.

4.4.2. Installation procedure

Algorithm 4.4 Barn server installation

```
if Barn server is not installed then
    Create needed Users and Groups
    Create Folder Structure
    Install SQL Server
    Configure SQL Server Parameters
    Configure Firewall
    Deploy Barn Database
else if Barn server is installed and installed version is older or equal then
    Upgrade Barn Database
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *Configure-MapticsUsers* in listing A.1.

Additionally, the following function is called:

```
#Add 'Lock pages' and 'Volume management' rights to geo_maptics and
    Administrators group
Configure-LocalSecurityPolicies
```

Listing 4.10: Barn user creation

The content of the function can be seen in listing A.1.

Create Folder Structure

A *Maptics Barn* database deployment expects a set of folders. In order to optimize IOPS (Input/Output Operations Per Second), the data, logs and temp files will be on different partitions. For the temp files case, we will use four different partitions:

```
Function Create-DBFolders{
  #Data
    Create-DataDBFolder "Data"
  #Logs
    Create-DataDBFolder "Logs"
  #Temp1
    Create-TempDBFolder 0
  #Temp2
    Create-TempDBFolder 1
  #Temp3
    Create-TempDBFolder 2
  #Temp4
    Create-TempDBFolder 3
  #Backups
    Create-BackupFolder "Backups"
  icacls "C:\Fielding" /T /grant Administrators:F
  icacls "D:\Fielding" /T /grant Administrators:F
  icacls "E:\Fielding" /T /grant Administrators:F
  icacls "F:\Fielding" /T /grant Administrators:F
  icacls "G:\Fielding" /T /grant Administrators:F
  icacls "H:\Fielding" /T /grant Administrators:F
  icacls "J:\Fielding" /T /grant Administrators:F
}
```

Listing 4.11: Barn folders creation

Create-DataDBFolder, *Create-TempDBFolder* and *Create-BackupFolder* are helper functions specified in listing A.2.

Install SQL Server

The installation of the Microsoft SQL server is done providing a bootstrap file that defines the necessary parameters to ensure the installation follows the *Maptics* specification[7]. This configuration file can be found in listing A.5.

```
Function Install-SQLServerUnattended{
    if(-not(Test-Path "HKLM:\SOFTWARE\Microsoft\Microsoft SQL
   Server\MAPTICS\MSSQLServer")){
        Print-Log "Installing SQL Server..."
        &$MSSQLPath /SQLSVCPASSWORD="Password" /AGTSVCPASSWORD="
    Password" /SAPWD="sapassword" /IACCEPTSQLSERVERLICENSETERMS /
    SQLSYSADMINACCOUNTS="BUILTIN\Administrators" /ConfigurationFile=
    C:\Fielding\MapticsMSSQLEnvironment\ConfigurationFile.ini
    }else{
        Print-Log "SQL Server already installed"
    }
}
```

Listing 4.12: MSSQL server installation

Configure SQL Server Parameters

After the installation, the Microsoft SQL Server is set to listen on port 1433 and is configured taking into account the server memory and location of the temporary files.

```
Set-SQLPort
Configure-SQLServer
```

Listing 4.13: MSSQL server configuration

More details about these two functions can be found in listing A.6.

Configure Firewall

The firewall is configured to allow traffic through TCP port 1433 and UDP port 1434.

```
Function Configure-Firewall{
    if(IsFirewallEnabled){
      Print-Log "Firewall is enabled, adding rules..."
    #Remove Firewall Rules
    netsh advfirewall firewall delete rule name="Fielding - SQL
    Server UDP 1434 Starting Connections"
    netsh advfirewall firewall delete rule name="Fielding - SQL
    Server TCP 1433 Port"
    #Add Firewall Rules
    netsh advfirewall firewall add rule name="Fielding - SQL Server
     UDP 1434 Starting Connections" dir=in action=allow protocol=UDP
     localport=1434
    netsh advfirewall firewall add rule name="Fielding - SQL Server
     TCP 1433 Port" dir=in action=allow protocol=TCP localport=1433
  }else{
      Print-Log "Firewall is not enabled, skipping configuration...
    "
  }

}
```

Listing 4.14: MSSQL server firewall configuration

The function makes use of the *IsFirewallEnabled* function defined in listing A.4.

4.4.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- \S: Silent installation flag.
- \HOST: The hostname or IP address of the MSSQL Server.
- \INSTANCE: The name of the MSSQL instance to create.
- \ADMINDB: The name of the MSSQL administration database to create.
- \REPODB: The name of the MSSQL repository database to create.

An example of an unattended install is as follows:

```
setupBarnServer_v4.0.11.16.exe \S  \HOST=XXX.XXX.XXX.XXX \INSTANCE=FIELDING
\ADMINDB=Fielding_Admin \REPODB=Fielding \D=C:\Fielding
```

4.5 Admin Gui

4.5.1. Introduction

The Admin GUI is an IIS hosted .NET web application that allows to control the *Maptics Parsing System* without manipulating the database directly.

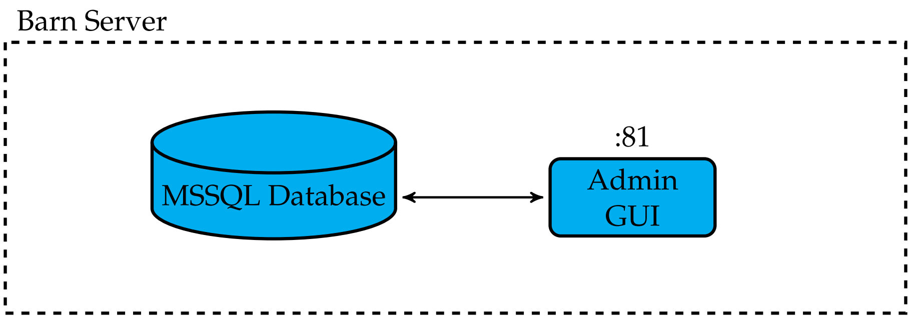


Figure 4.5: *Admin GUI* diagram

4.5.2. Installation procedure

Algorithm 4.5: Admin GUI installation

```
if Admin GUI is not installed then
    Create neede Users and Groups
    Install .NET 4.5 ASPNET
    Install IIS 7
    Create an IIS application and configure it
    Deploy Admin GUI application
else if Admin GUI is installed and installed version is older or equal then
    Upgrade Admin GUI application
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *Configure-MapticsUsers* in listing A.1.

Install .NET 4.5 ASPNET

The installation of the ASPNET 4.5 is done with Powershell via the *Server Manager* module[8].

```
Import-Module ServerManager
Add-WindowsFeature NET-Framework-45-ASPNET -WarningAction
   SilentlyContinue
```

Listing 4.15: Install .NET 4.5 ASPNET

Install IIS 7

The installation of the IIS (Internet Information Services) is also done with Powershell via the *Server Manager* module[8].

```
Import-Module ServerManager
Add-WindowsFeature Web-Server -WarningAction SilentlyContinue
Add-WindowsFeature Web-Net-Ext45, Web-ASP, Web-Asp-Net45,
    Web-ISAPI-Ext, Web-ISAPI-Filter, Web-Http-Redirect,
    Web-Filtering, Web-Security, Web-Client-Auth, Web-Digest-Auth,
    Web-Cert-Auth, Web-Url-Auth, Web-Windows-Auth -WarningAction
    SilentlyContinue
Add-WindowsFeature Web-Mgmt-Console -WarningAction SilentlyContinue
```

Listing 4.16: Install IIS

Create and configure an IIS application

The creation and configuration of the IIS application is done in Powershell via the IIS Administration Cmdlets[9].

```
$AdminGUIExists=Get-WebApplication -Name AdminGUI
if($AdminGUIExists -eq $null){
  New-WebApplication -Name AdminGUI -Site 'Default Web Site'
    -PhysicalPath "C:\Fielding\Maptics\Release\AdminGui"
    -ApplicationPool DefaultAppPool
  #Configure Web Application
  Set-WebConfigurationProperty -Filter "system.webServer/security/
    authentication/anonymousAuthentication" -PSPath MACHINE/WEBROOT/
    APPHOST -Location 'Default Web Site/AdminGUI' -Name Enabled
    -Value $true
  Set-WebConfigurationProperty -Filter "system.webServer/security/
    authentication/anonymousAuthentication" -PSPath MACHINE/WEBROOT/
    APPHOST -Location 'Default Web Site/AdminGUI' -Name username
    -Value ""
  Set-WebConfigurationProperty -filter system.Applicationhost/Sites
    /SiteDefaults/logfile -PSPath MACHINE/WEBROOT/APPHOST -Name
    logFormat -Value NCSA
  Set-WebConfigurationProperty -filter system.Applicationhost/Sites
    /SiteDefaults/logfile -PSPath MACHINE/WEBROOT/APPHOST -Name
    localTimeRollover -Value true
  Set-WebBinding -Name 'Default Web Site' -PropertyName Port -Value
     81
  Start-WebSite -Name "Default Web Site"
  #Open firewall port 81
  if(IsFirewallEnabled){
    if([bool](Get-NetFirewallRule -DisplayName "Admin Gui Port
    Access" -ErrorAction SilentlyContinue)){
      Write-Output "`"Admin Gui Port Access`" firewall rule already
     present."
    }else{
      Write-Output "Adding rule to the firewall to allow access
    to port 81..."
      netsh advfirewall firewall add rule name='Admin Gui Port
    Access' dir=in protocol=tcp localport=81 action=allow profile=
    any
    }
  }else{
    Write-Host "Firewall is disabled, skipping rule addition..."
  }
}
```

Listing 4.17: Create and configure an IIS application

The function makes use of the *IsFirewallEnabled* function defined in listing A.4.

Deploy and configure *Maptics Admin GUI* application

To finally deploy the application, the binary files have to be copied and the connection strings have to be configured to point to the correct MSSQL instance. This part can be done through NSIS scripting.

```
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Copying Admin
   GUI binaries..." ${APP_NAME}
CreateDirectory "${FIELD_CUSTOM_PATH_DEFAULT}\AdminGUI"
SetOutPath "${FIELD_CUSTOM_PATH_DEFAULT}\AdminGUI"
File /a /r "AdminGUI\"
File /a /r "Config Files Templates\AdminGUI\"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Admin GUI
   Binaries copied." ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
   Admin GUI..." ${APP_NAME}
#Configure AdminGUI
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
   \AdminGUI\Web.config" ";data source=XXX.XXX.XXX.XXX\MAPTICS;
   initial catalog=MAPTICS_Admin;" ";data source=
   $v_DataBaseServer\$v_DataBaseInstance;initial catalog=
   $v_DataBaseName;"
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
   \AdminGUI\Web.config" "Data Source=XXX.XXX.XXX.XXX\MAPTICS;
   Initial Catalog=MAPTICS;" "Data Source=
   $v_DataBaseServer\$v_DataBaseInstance;Initial Catalog=
   $v_RepoDataBaseName;"
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
   \AdminGUI\Web.config" "Data Source=XXX.XXX.XXX.XXX\MAPTICS;
   Initial Catalog=MAPTICS_Admin;" "Data Source=
   $v_DataBaseServer\$v_DataBaseInstance;Initial Catalog=
   $v_DataBaseName;"
ExecWait 'cacls "C:\Fielding\Maptics\Release\AdminGui" /T /E /G
   IIS_IUSRS:f'
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Admin GUI
   configured." ${APP_NAME}
```

Listing 4.18: Deploy and configure Admin GUI application

4.5.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- **S:** Silent installation flag.
- **HOST:** The hostname or IP address of the MSSQL Server.
- **INSTANCE:** The name of the MSSQL instance.
- **ADMINDB:** The name of the MSSQL administration database.

- \REPODB: The name of the MSSQL repository database.

An example of an unattended install is as follows:

```
setupAdminGUI_v4.0.11.16.exe \S  \HOST=XXX.XXX.XXX.XXX \INSTANCE=FIELDING
\ADMINDB=Fielding_Admin \REPODB=Fielding \D=C:\Fielding
```

4.6 Maptics Monitor

4.6.1. Introduction

The *Maptics Monitor* is a Tomcat hosted Java web application that exposes information about the whole Maptics solution in *json* form through port 8009.

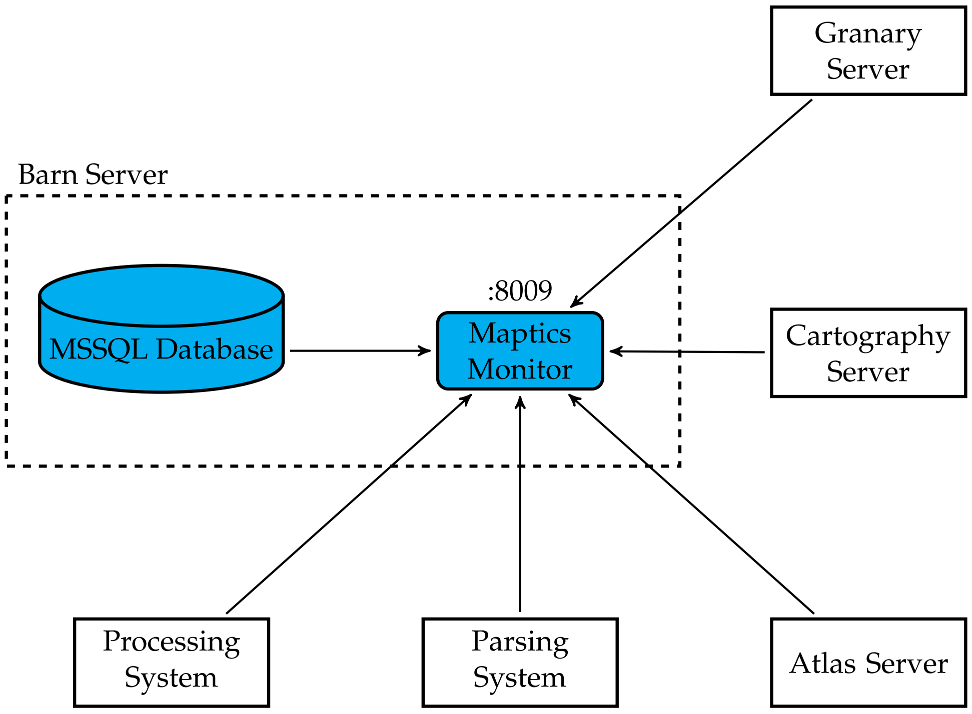


Figure 4.6: *Maptics Monitor* diagram

The application extracts the parameters to monitor from a set of tables in the *Barn* database and matches them with the information arriving from each component.

4.6.2. Installation procedure

Algorithm 4.6: Maptics Monitor installation

```
if Maptics Monitor is not installed then
    Create needed Users and Groups
    Install Apache and Tomcat 6
    Install MSMQ
    Deploy Maptics Monitor application
    Configure Maptics Monitor application
    Configure Firewall
else if Maptics Monitor is installed and installed version is older or equal then
    Upgrade Maptics Monitor
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *Configure-MapticsUsers* in listing A.1.

Install Apache and Tomcat 6

Apache and Tomcat 6 are installed with via the Fielding Web Environment. This is a transversal tool that installs Apache and Tomcat 6 at `C:\Fielding\WebEnvironment`.

Install MSMQ

The installation of MSMQ (Message Queuing) is done with Powershell via the *Server Manager* module[8].

```
Import-Module ServerManager
Add-WindowsFeature MSMQ -WarningAction SilentlyContinue
```

Listing 4.19: Install MSMQ

Deploy and configure *Maptics Monitor* application

To deploy the *Maptics Monitor* application:

1. Binary files have to be copied to `C:\Fielding\Release\Monitor`.
2. Web application file has to be copied to Tomcat web application folder at `C:\Fielding\WebEnvironment`.
3. Connection strings have to be configured to point to the correct *Barn* MSSQL instance.

```
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Copying Monitor
    binaries..." ${APP_NAME}
#Install/update Monitor
CreateDirectory "${FIELD_CUSTOM_PATH_DEFAULT}\Monitor"
CreateDirectory "${FIELD_CUSTOM_PATH_DEFAULT}
    \Monitor\WindowsService"
SetOutPath "${FIELD_CUSTOM_PATH_DEFAULT}\Monitor\WindowsService"
File /a /r "Monitor\Windows Service\"
SetOutPath "${FIELD_CUSTOM_PATH_DEFAULT}\Monitor\WindowsService"
File "Config Files Templates\Monitor\monitor.config"
File "Config Files Templates\Monitor\NLog.config"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Binaries copied.
    " ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Monitor..." ${APP_NAME}
#Configure Monitor
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
    \Monitor\WindowsService\monitor.config" "<databaseServer>XXX.XXX
    .XXX.XXX</databaseServer>" "<databaseServer>$v_DataBaseServer</
    databaseServer>"
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
    \Monitor\WindowsService\monitor.config" "<databaseInstance>
    MAPTICS</databaseInstance>" "<databaseInstance>
    $v_DataBaseInstance</databaseInstance>"
```

```
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
   \Monitor\WindowsService\monitor.config" "<databaseName>
   MAPTICS_Admin</databaseName>" "<databaseName>$v_DataBaseName</
   databaseName>"
!insertmacro ReplaceInFile "${FIELD_CUSTOM_PATH_DEFAULT}
   \Monitor\WindowsService\monitor.config" "<databaseName>MAPTICS</
   databaseName>" "<databaseName>$v_RepoDataBaseName</databaseName>
   "
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Monitor
   configured." ${APP_NAME}

!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Copying web app
   ..." ${APP_NAME}
#Install/update java web application
SetOutPath "C:\Fielding\WebEnvironment\webapps"
File /a /r "Monitor\WebService\com.Fielding.maptics.monitor.servlet
   "
File "Monitor\WebService\com.Fielding.maptics.monitor.servlet.war"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Web app copied."
    ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring web
   app..." ${APP_NAME}
#Configure webapp
SetOutPath "C:\Fielding\WebEnvironment\webapps\com.Fielding.maptics
   .monitor.servlet\WEB-INF"
File "Config Files Templates\Monitor\json_servlet_config.xml"
!insertmacro ReplaceInFile "C:\Fielding\WebEnvironment\webapps\com.
   Fielding.maptics.monitor.servlet\WEB-INF\json_servlet_config.xml
   " "<databaseServer>XXX.XXX.XXX.XXX</databaseServer>" "<
   databaseServer>$v_DataBaseServer</databaseServer>"
!insertmacro ReplaceInFile "C:\Fielding\WebEnvironment\webapps\com.
   Fielding.maptics.monitor.servlet\WEB-INF\json_servlet_config.xml
   " "<databaseInstance>MAPTICS</databaseInstance>" "<
   databaseInstance>$v_DataBaseInstance</databaseInstance>"
!insertmacro ReplaceInFile "C:\Fielding\WebEnvironment\webapps\com.
   Fielding.maptics.monitor.servlet\WEB-INF\json_servlet_config.xml
   " "<databaseName>MAPTICS_Admin</databaseName>" "<databaseName>
   $v_DataBaseName</databaseName>"
!insertmacro ReplaceInFile "C:\Fielding\WebEnvironment\webapps\com.
   Fielding.maptics.monitor.servlet\WEB-INF\json_servlet_config.xml
   " "<databaseName>MAPTICS</databaseName>" "<databaseName>
   $v_RepoDataBaseName</databaseName>"
#Configure server.xml
!insertmacro ReplaceInFile "C:
   \Fielding\WebEnvironment\bin\tomcat\conf\server.xml" "<Connector
    port=$\"8009$\" protocol=$\"AJP/1.3$\" redirectPort=$\"8443$\"
   />" "<Connector port=$\"8009$\" protocol=$\"HTTP/1.1$\"
   redirectPort=$\"8443$\" connectionTimeout=$\"20000$\" maxThreads
   =$\"1$\" />"
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring web
   app..." ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Installing
   Monitor service..." ${APP_NAME}
#Install Monitor service
${If} $v_UPDATE == "0"
  ExecWait '"${FIELD_CUSTOM_PATH_DEFAULT}
   \Monitor\WindowsService\Maptics.MonitorService.exe" -install'
  ExecWait '"sc.exe" failure "fielding.MonitorService" reset= 86400
    actions= restart/60000/restart/60000//60000'
```

```
${EndIf}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Monitor service.
    " ${APP_NAME}
!insertmacro FIELD_TOLOG $INSTDIR\${APP_NAME}.log "Configuring
    Monitor..." ${APP_NAME}
```

Listing 4.20: Deploy *Maptics Monitor*

After that, it has to be ensured that the service runs under *LocalSystem* user.

```
#Change monitor service account to "LocalSystem"
$MonitorSrv = Get-WmiObject Win32_service -filter "name='maptics.
    MonitorService'"
$MonitorSrv.Change($null,$null,$null,$null,$null,$null,".\
    LocalSystem","")
```

Listing 4.21: Configure *Maptics Monitor*

4.6.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- **S:** Silent installation flag.
- **HOST:** The hostname or IP address of the MSSQL Server.
- **INSTANCE:** The name of the MSSQL instance.
- **ADMINDB:** The name of the MSSQL administration database.
- **REPODB:** The name of the MSSQL repository database.

An example of an unattended install is as follows:

```
setupMapticsMonitor_v4.0.11.16.exe \S  \HOST=XXX.XXX.XXX.XXX
\INSTANCE=FIELDING \ADMINDB=Fielding_Admin \REPODB=Fielding \D=C:\Fielding
```

4.7 Atlas Server

4.7.1. Introduction

The *Atlas* server hosts a *PostgreSQL*[13] database with the *PostGIS*[14] which holds the necessary map generation data. The parser will leave packages of data in a shared *samba* folder for the *Atlas* server to process. The concurrency of the parser access needs to be controlled because too many parsers accessing the database could hinder the performance. Thus, we will deploy and configure *PgBouncer*[15] to control and redirect the parser access through port 6432.

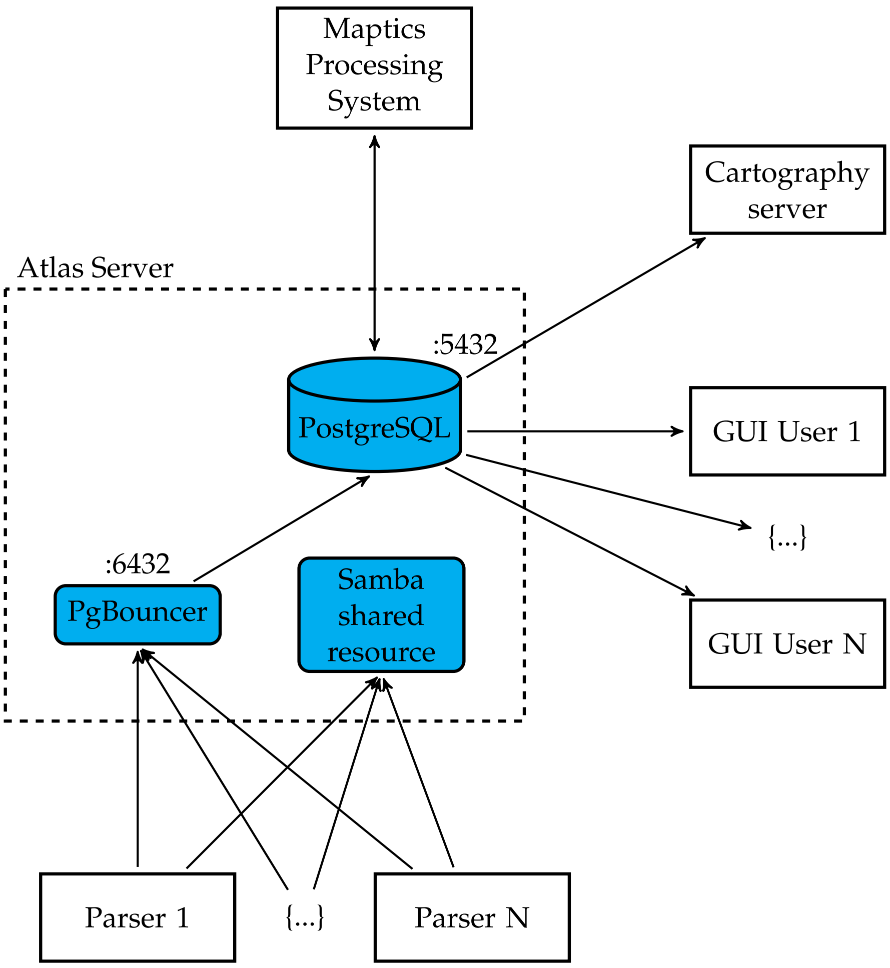


Figure 4.7: *Atlas* server diagram

The rest of the connections will go through the standard port (5432) and will be managed directly by the PostgreSQL server. *Cartography* will access *Atlas* data through *GeoServer* layers. The *Maptics Processing system* will process and aggregate its data, storing the results in other tables. The *Maptics GUI* will leverage this preprocessing to show the result of algorithms in map form.

4.7.2. Installation procedure

Algorithm 4.7: Atlas server installation

```
if Atlas is not installed then
   Create needed Users and Groups
   Create atlas local repository
   Install postgresql, postgis, pgbouncer
   Configure Atlas
   Configure pgbouncer
   Install and configure samba
   Configure firewall
else if Atlas is installed and installed version is older or equal then
   Update Atlas configuration
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *maptics_setup_create_geo_fielding_user* in listing A.7.

Create *Atlas* local repository

Even though *Fielding* mantains an internal repository, some applications need more advanced versions than the ones used transversally. This is the case for *Atlas* regarding *PostgreSQL*, *PostGIS* and *Pgbouncer*. The rpm have to be provided with the installer and to help resolve the dependencies while installing, a local repository is created with *createrepo*[16][17]. The repository contains PostgreSQL, Pgbouncer and all the needed dependencies.

```
atlas_setup_create_repository() {
  field_tolog_section "** Creating local repository **"
  yum --disablerepo="*" --enablerepo="atlasrepo,Fielding*" -y
   install createrepo
  createrepo $FIELD_SCRIPT_DIR/c72/atlasrepo >> $FIELD_LOGFILE 2>&1
  mv $FIELD_SCRIPT_DIR/c72/conf/atlasrepo.repo /etc/yum.repos.d/
}
```

Listing 4.22: Create *Atlas* local repository

Install postgresql, postgis, pgbouncer

Once the local repository is created, *PostgreSQL*, *PostGIS* and *PgBouncer* can be installed via *yum*.

```
atlas_setup_install_rpms() {
  field_tolog_section "** Installing atlas **"
  # Install postgres and postgis from repository
  yum --disablerepo="*" --enablerepo="Fielding*,atlasrepo" -y
   install postgresql96 postgresql96-server postgresql96-libs
   postgresql96-contrib postgresql96-devel >> $FIELD_LOGFILE 2>&1
  yum --disablerepo="*" --enablerepo="Fielding*,atlasrepo" -y
   install postgis2_96 >> $FIELD_LOGFILE 2>&1
  # Install pgbouncer from repository
  yum --disablerepo="*" --enablerepo="Fielding*,atlasrepo" -y
   install pgbouncer >> $FIELD_LOGFILE 2>&1
}
```

Listing 4.23: Install postgresql postgis pgbouncer

Configure *Atlas*

To configure the *Atlas* server we will move the *PostgreSQL* data folder from `/var/lib/pgsql/9.6/data` to `/user/local/fielding/data/atlas/data`. The new data location has to be set also in the *systemd* service file at `/etc/systemd/system/postgresql-9.6.service`. As the *pg_xlog* folder is in another partition, it has to be moved to the right place and set a symbolic link to it.

```
atlas_setup_configure() {
  # Make sure necessary folders exist with appropriated permissions
  mkdir -p /usr/local/fielding/data/atlas/data
  mkdir -p /usr/local/fielding/data/atlas/xlog
  # Create a new systemd service unit file pointing to our db
   folder
  sed 's@/var/lib/pgsql/9.6/data/@/usr/local/fielding/data/atlas/
   data/@' /usr/lib/systemd/system/postgresql-9.6.service > /etc/
   systemd/system/postgresql-9.6.service
  field_tolog_section "** Initializing DB  **"
  /usr/pgsql-9.6/bin/postgresql96-setup initdb >> $FIELD_LOGFILE
   2>&1
  field_tolog_section "** Configuring atlas  **"
  # Move postgres transaction log to another partition and symlink
   it
  mv /usr/local/fielding/data/atlas/data/pg_xlog /usr/local/
   fielding/data/atlas/xlog
  ln -s /usr/local/fielding/data/atlas/xlog/pg_xlog /usr/local/
   fielding/data/atlas/data/pg_xlog
  # Copy postgres configuration files
  mv $FIELD_SCRIPT_DIR/c72/conf/pg_hba.conf /usr/local/fielding/
   data/atlas/data/pg_hba.conf
  cp -f  $FIELD_SCRIPT_DIR/c72/conf/postgresql64GB.conf /usr/local/
   fielding/data/atlas/data/postgresql.conf
  TOTALMEM=$(free -h | awk '/Mem\:/ { print $2 }' | sed 's/[A-Za-z
   ]//')
  SHAREDMEM=$(echo "($TOTALMEM * 0.33) / 1" | bc)"GB"
  sed -i "/shared_buffers/c\ shared_buffers = $SHAREDMEM" /usr/
   local/fielding/data/atlas/data/postgresql.conf
  EFFECTIVECACHESIZE=$(echo "($TOTALMEM * 0.75) / 1" | bc)"GB"
  sed -i "/effective_cache_size/c\ effective_cache_size =
   $EFFECTIVECACHESIZE" /usr/local/fielding/data/atlas/data/
   postgresql.conf
  chown postgres:postgres /usr/local/fielding/data/atlas/data/
   postgresql.conf
  chmod 700 /usr/local/fielding/data/atlas/data
  chmod 700 /usr/local/fielding/data/atlas/xlog
  chown -R postgres:postgres /usr/local/fielding/data/atlas/data
  chown -R postgres:postgres /usr/local/fielding/data/atlas/xlog
}
```

Listing 4.24: Configure *Atlas*

The default configuration files for an *Atlas* server have to be copied and be modified taking into account the memory of the server. Finally, we have to ensure all the files have the correct permissions.

Configure *pgbouncer*

Like with *PostgreSQL*, a new *pgbouncer* service file has to be created at `/etc/systemd/system` copied from `/usr/lib/systemd/system`. This is done because the service files located at `/etc/systemd/system` take precedence over the ones at `/usr/lib/systemd/system` which is where the standard installation leaves the service file.

```
atlas_setup_configure_pgbouncer() {
  # Create a new systemd service
  sed 's/User=pgbouncer/User=geo_fielding/' /usr/lib/systemd/system
   /pgbouncer.service > /etc/systemd/system/pgbouncer.service
  sed -i 's/Group=pgbouncer/Group=FIELD_GEO/' /etc/systemd/system/
   pgbouncer.service
  # pgbouncer configuration
  yum --disablerepo="*" --enablerepo="fielding*" -y install python-
   psycopg2 >> $FIELD_LOGFILE 2>&1
  chown -R geo_fielding:FIELD_GEO /var/log/pgbouncer
  chown -R geo_fielding:FIELD_GEO /var/run/pgbouncer
  sed -i 's@/home/pgbouncer:/bin/bash@/home/pgbouncer:/sbin/
   nologin@' /etc/passwd
  mv $FIELD_SCRIPT_DIR/c72/conf/pgbouncer.ini /etc/pgbouncer/
  mv $FIELD_SCRIPT_DIR/c72/conf/userlist.txt /etc/pgbouncer/
  cp -f $FIELD_SCRIPT_DIR/c72/conf/pgbouncer.service /etc/systemd/
   system/pgbouncer.service
  chown -R geo_fielding:FIELD_GEO /etc/pgbouncer/
  systemctl restart pgbouncer >> $FIELD_LOGFILE 2>&1
}
```

Listing 4.25: Configure *pgbouncer*

Psycopg2[18], the *PostgreSQL* database adapter for *python* is installed to connect to the database and generate properly a *userlist* file with the users allowed to connect to the database. Then, it is ensured that all the users and files have the correct permissions.

Install and Configure *samba*

Samba is installed via yum.

```
atlas_setup_install_samba() {
  field_tolog_section "** Installing samba  **"
  yum --disablerepo="*" --enablerepo="fielding*,atlasrepo" -y
   install dos2unix >> $FIELD_LOGFILE 2>&1
  yum --disablerepo="*" --enablerepo="fielding*,atlasrepo" -y
   install samba samba-client samba-common  >> $FIELD_LOGFILE 2>&1
  mv /etc/samba/smb.conf /etc/samba/smb.conf.bak
  mv $FIELD_SCRIPT_DIR/c72/conf/smb.conf /etc/samba/
  dos2unix /etc/samba/smb.conf >> $FIELD_LOGFILE 2>&1
  systemctl restart smb
}
```

Listing 4.26: Install and Configure *samba*

After the installation, the *Atlas* default configuration is copied over.

Configure firewall

The firewall is configured to allow traffic for *samba*, *PgBouncer* and *PostgreSQL*.

```
atlas_setup_configure_firewall(){
  firewall-cmd --permanent --zone=fielding --add-service=samba >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --permanent --zone=fielding --add-port=137/tcp >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --permanent --zone=fielding --add-port=138/tcp >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --permanent --zone=fielding --add-port=139/tcp >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --permanent --zone=fielding --add-port=445/tcp >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --permanent --zone=fielding --add-port=5432/tcp >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --permanent --zone=fielding --add-port=6432/tcp >>
   $FIELD_LOGFILE 2>&1
  firewall-cmd --reload >> $FIELD_LOGFILE 2>&1
}
```

Listing 4.27: Configure firewall

4.7.3. Unattended Installation

To execute an unattended installation the following parameters have to be provided:

- **-s:** Silent installation flag.

An example of an unattended install is as follows:

```
setupAtlas_v4.0.11.16.bin -s
```

4.8 Cartography Server

4.8.1. Introduction

The *Cartography* server provides geolocalized layers over available map engines like Carto[1], OpenStreetMap[2] or Stamen[3]. It allows the visualization of the algorithm's results executed by the Maptics Processor component.

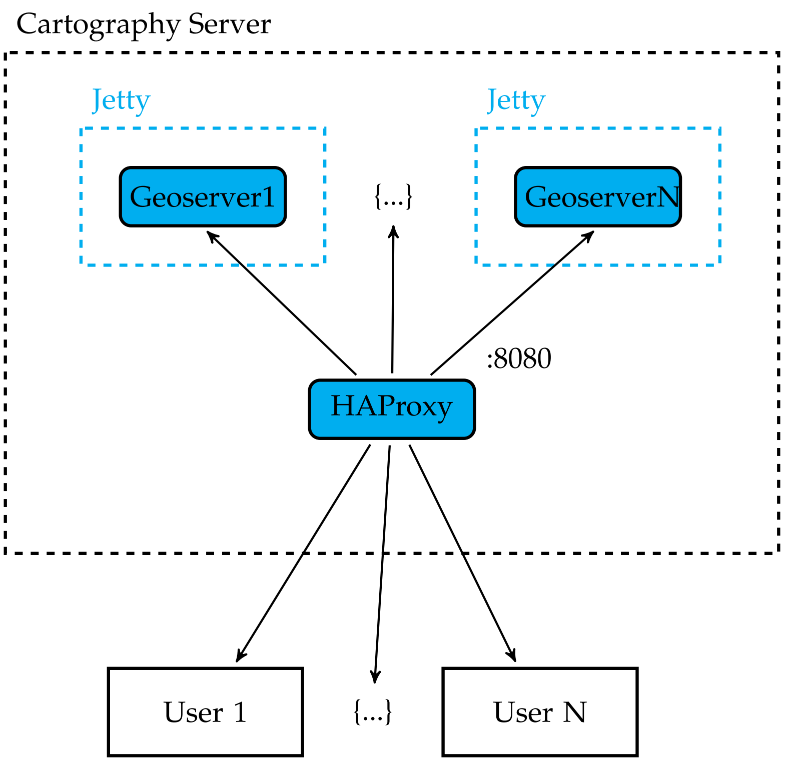


Figure 4.8: *Cartography* server diagram

There are multiple *GeoServer*[19] instances behing a load balancer (HAProxy[20]). The more resources the server has, the more instances we can enable, thus being able to server more users concurrently.

[1]Carto: https://carto.com/
[2]OpenStreetMap: https://www.openstreetmap.org/
[3]Stamen: http://maps.stamen.com/

4.8.2. Installation procedure

Algorithm 4.8: Cartography server installation

```
if Cartography is not installed then
    Create needed Users and Group
    Deploy GeoServer binaries
    Install Java
    Install and configure JAI and ImageIO libraries
    Create and configure geoserver data folder
    Install and configure HAProxy
    Instance the appropriate number of geoserver services
    Install prerequisites to create styles and layers
    Create appropriate styles and layers through geoserver REST configuration API
    Configure firewall
else
    if Cartography is installed and installed version is older or equal then
        Update configuration files
    end if
    Update styles an layers through geoserver REST configuration API
end if
```

Create needed Users and Groups

Standard *Maptics* group and user is created through the helper function *maptics_setup_create_geo_fielding_user* in listing A.7.

Deploy GeoServer binaries

Copy the GeoServer binaries to */usr/share/geoserver*.

Install Java

As a requirement, the *Java Runtime Environment* is installed.

```
cartography_install_java() {
  field_tolog_section "** Install java **"
  rpm -Uvh $FIELD_SCRIPT_DIR/c72/java/jre-8u102-linux-x64.rpm 2>&1
   | field_tolog_pipeline
  rpm -qi jre1.8.0_102-1.8.0_102-fcs.x86_64 2>&1 |
   field_tolog_pipeline
  if [ "$?" -eq "0" ]; then
    field_tolog "Java Runtime Environment 1.8u102 installed."
  else
    field_tolog_error "Error installing Java Runtime Environment."
  fi
}
```

Listing 4.28: Install Java prerequisite

Install and configure JAI and ImageIO libraries

To improve performance, the *JAI ImageIO*[22] libraries will be installed natively to replace the ones in *GeoServer*.

```
cartography_setup_install_jai_imageio() {
  field_tolog_section "** Install JAI/ImageIO libraries **"
  if [ ! -e /usr/java/jre1.8.0_102/LICENSE-jai.txt ]; then
    cp $FIELD_SCRIPT_DIR/c72/jai_imageio/jai-1_1_3-lib-linux-amd64-
   jre.bin /usr/java/jre1.8.0_102
    pushd /usr/java/jre1.8.0_102 > /dev/null
    yes | sh jai-1_1_3-lib-linux-amd64-jre.bin > /dev/null
    JAI_INSTALLED=`echo $?`
    if [ "$JAI_INSTALLED" -eq "0" ]; then
      field_tolog "JAI library installed."
    else
      field_tolog_error "Error installing JAI library."
    fi
    rm -f jai-1_1_3-lib-linux-amd64-jre.bin
    popd > /dev/null
  fi
  if [ ! -e /usr/java/jre1.8.0_102/LICENSE-jai_imageio.txt ]; then
    cp $FIELD_SCRIPT_DIR/c72/jai_imageio/jai_imageio-1_1-lib-linux-
   amd64-jre.bin /usr/java/jre1.8.0_102
    export _POSIX2_VERSION=199209
    pushd /usr/java/jre1.8.0_102 > /dev/null
    yes | sh jai_imageio-1_1-lib-linux-amd64-jre.bin > /dev/null
    IMAGEIO_INSTALLED=`echo $?`
    if [ "$IMAGEIO_INSTALLED" -eq "0" ]; then
      field_tolog "IMAGEIO library installed."
    else
      field_tolog_error "Error installing IMAGEIO library."
    fi
    rm -f jai_imageio-1_1-lib-linux-amd64-jre.bin
    popd > /dev/null
  fi

  #Delete geoserver jai libraries as we have native installation
  rm -f /usr/share/geoserver/webapps/geoserver/WEB-INF/lib/
   jai_codec-1.1.3.jar 2>&1 | field_tolog_pipeline
  rm -f /usr/share/geoserver/webapps/geoserver/WEB-INF/lib/jai_core
   -1.1.3.jar 2>&1 | field_tolog_pipeline
  rm -f /usr/share/geoserver/webapps/geoserver/WEB-INF/lib/
   jai_imageio-1.1.jar 2>&1 | field_tolog_pipeline
}
```

Listing 4.29: Install JAI and ImageIO libraries

Create and configure geoserver data folder

A common data folder is configured for all *GeoServer* instances.

```
cartography_setup_data_folders(){
  field_tolog_section "** Setup data folders **"
  cp -r /usr/share/geoserver/data_dir/* /usr/local/fielding/data/
   cartography/data_dir/
  if [ -d "/usr/local/fielding/data/cartography/data_dir/data" ];
   then
```

```
    field_tolog "Data folders moved to /usr/local/fielding/
   cartography/data_dir."
  else
    field_tolog_error "Error moving binary folders."
  fi
  chown -R geo_fielding:FIELD_GEO /usr/local/fielding/data/
   cartography/
  field_tolog_section "** Configure geoserver **"
  # Overwrite GeoWebCache conf file
  cp $FIELD_SCRIPT_DIR/c72/conf/CONFgwc-gs.xml /usr/local/fielding/
   data/cartography/data_dir/gwc-gs.xml 2>&1 | field_tolog_pipeline
  if [ -e "/usr/local/fielding/data/cartography/data_dir/gwc-gs.xml
   " ]; then
    field_tolog "Configured GeoWebCache."
  else
    field_tolog_error "Error configuring GeoWebCache."
  fi
  # Overwrite global conf file
  cp $FIELD_SCRIPT_DIR/c72/conf/CONFglobal.xml /usr/local/fielding/
   data/cartography/data_dir/global.xml 2>&1 | field_tolog_pipeline
  if [ -e "/usr/local/fielding/data/cartography/data_dir/global.xml
   " ]; then
    field_tolog "Configured global options."
  else
    field_tolog_error "Error configuring global options."
  fi
 # Overwrite wfs conf file
  cp $FIELD_SCRIPT_DIR/c72/conf/CONFwfs.xml /usr/local/fielding/
   data/cartography/data_dir/wfs.xml 2>&1 | field_tolog_pipeline
  if [ -e "/usr/local/fielding/data/cartography/data_dir/wfs.xml"
   ]; then
    field_tolog "Configured wfs options."
  else
    field_tolog_error "Error configuring global options."
  fi
 # Overwrite logging conf file
  cp $FIELD_SCRIPT_DIR/c72/conf/CONFlogging.xml /usr/local/fielding
   /data/cartography/data_dir/logging.xml 2>&1 |
   field_tolog_pipeline
  if [ -e "/usr/local/fielding/data/cartography/data_dir/logging.
   xml" ]; then
    field_tolog "Configured logging options."
  else
    field_tolog_error "Error configuring logging options."
  fi
  chown -R geo_fielding:FIELD_GEO /usr/local/fielding/data/
   cartography/
}
```

Listing 4.30: Create and configure geoserver data folder

It is ensured that all the files have the needed permissions.

Install and configure HAProxy

HAProxy is built an configured using a default template.

```
cartography_setup_install_haproxy(){
  field_tolog_section "** Install HAProxy  **"
  #install gcc
  if [ ! -e /etc/systemd/system/haproxy.service ]; then
    mkdir -p /run/haproxy
    chown -R geo_fielding:FIELD_GEO /run/haproxy
    yum --disablerepo="*" --enablerepo="fielding*" -y install gcc
   >> $FIELD_LOGFILE 2>&1
    pushd $FIELD_SCRIPT_DIR/c72/haproxy/ > /dev/null
    tar xzfv haproxy-1.6.9.tar.gz > /dev/null
    pushd $FIELD_SCRIPT_DIR/c72/haproxy/haproxy-1.6.9/ > /dev/null
    make TARGET=linux2628 ARCH=$(uname -m) 2>&1 |
   field_tolog_pipeline
    make install 2>&1 | field_tolog_pipeline
    cp haproxy-systemd-wrapper /usr/local/sbin 2>&1 |
   field_tolog_pipeline
    ln -s /usr/local/sbin/haproxy /usr/sbin/ 2>&1 |
   field_tolog_pipeline
    ln -s /usr/local/sbin/haproxy-systemd-wrapper /usr/sbin/ 2>&1 |
    field_tolog_pipeline
    useradd --system haproxy 2>&1 | field_tolog_pipeline
    popd > /dev/null
    popd > /dev/null
    mkdir -p /etc/haproxy
  fi
  cp $FIELD_SCRIPT_DIR/c72/conf/haproxy.cfg /etc/haproxy/haproxy.
   cfg 2>&1 | field_tolog_pipeline
  chown -R geo_fielding:FIELD_GEO /etc/haproxy
  cp $FIELD_SCRIPT_DIR/c72/conf/haproxy.service /etc/systemd/system
   /haproxy.service 2>&1 | field_tolog_pipeline
  sed -i 's@/home/haproxy:/bin/bash@/home/haproxy:/sbin/nologin@' /
   etc/passwd
  mkdir -p /run/haproxy
  chown -R geo_fielding:FIELD_GEO /run/haproxy
}
```

Listing 4.31: Install and configure HAProxy

Instance the appropriate number of *GeoServer* services

The number of *GeoServer* instances depends upon the virtual cores available to the server. The specification allows for a *GeoServer* instance per three cores, each instance being able to service six concurrent *Maptics GUI* users.

```
cartography_setup_instance_geoservers(){
  NumberCores=`cat /proc/cpuinfo | grep processor | wc -l`
  NGEOSELECTION=$(($NumberCores/3))
  if [ "$NGEOSELECTION" -lt "1" ];then
    NGEOSELECTION=1
  fi
  NGEOUSERS=$(($NGEOSELECTION*6))
  if [ "$SILENT" == "0" ]; then
    show_msg "Your server is capable of $NGEOSELECTION concurrent
   geoserver instances ($NGEOUSERS concurrent map users)."
  fi
  field_tolog "Your server is capable of $NGEOSELECTION concurrent
   geoserver instances ($NGEOUSERS concurrent map users)."
  field_tolog_section "** Instance geoserver **"
  for (( i=1; i<=$NGEOSELECTION; i++))
  do
```

```
    field_tolog "Instancing geoserver$i"
    #make a symbolic link of /usr/share/geoserver
    GEOPATH="/opt/fielding/cartography/geoserver"$i
    JETTYPORT=$((8080 + $i))
    STOPJETTYPORT=$((8080 - $i))
    mkdir -p $GEOPATH
    ln -s /usr/share/geoserver/* $GEOPATH
    unlink $GEOPATH"/start.ini"
    cp /usr/share/geoserver/start.ini $GEOPATH
    sed -i "s/jetty.port=8080/jetty.port=$JETTYPORT/" $GEOPATH"/start.ini"
    #create the systemd service files
    SERVICEPATH="/etc/systemd/system/geoserver"$i".service"
    ENVSERVICEPATH="/etc/systemd/system/geoserver"$i".service.d/environment.env"
    cp $FIELD_SCRIPT_DIR/c72/conf/geoserver.service $SERVICEPATH
    mkdir -p $SERVICEPATH".d/"
    cp $FIELD_SCRIPT_DIR/c72/conf/environment.env $ENVSERVICEPATH
    #configure the systemd service files
    sed -i "s@/etc/systemd/system/geoserver.service.d/environment.env@$ENVSERVICEPATH@" $SERVICEPATH
    sed -i "s@/opt/fielding/cartography/geoserver@$GEOPATH@" $SERVICEPATH
    sed -i "s@/opt/fielding/cartography/geoserver@$GEOPATH@" $ENVSERVICEPATH
    sed -i "s@PORT=8080@PORT=$JETTYPORT@" $ENVSERVICEPATH
    sed -i "s@STOPPORT=8079@STOPPORT=$STOPJETTYPORT@" $ENVSERVICEPATH
    sed -i "s@STOPKEY=geoserver@STOPKEY=geoserver$i@" $ENVSERVICEPATH
    #uncomment geoserver in haproxy.cfg
    sed -i "s/#\tserver geoserver$i /\tserver geoserver$i /" /etc/haproxy/haproxy.cfg
    #enable and start service
    systemctl enable geoserver$i.service 2>&1 | field_tolog_pipeline
    systemctl start geoserver$i.service 2>&1 | field_tolog_pipeline
    sleep 10
  done
  chown -R geo_fielding:FIELD_GEO /opt/fielding/cartography/
  chown -R geo_fielding:FIELD_GEO /usr/share/geoserver/
}
```

Listing 4.32: Instance the appropriate number of geoserver services

Each *GeoServer* instance will listen through a port beggining at 8081 and onwards. For each instance we will also create a *systemd* service file at `/etc/systemd/system`. *HAProxy* will be configured to be able to balance the load over all *GeoServer* instances available.

Install prerequisites to create styles and layers

To update the *GeoServer* layers, the *Cartography* server needs to be able to connect to *MSSQL server* through a *python* script. This is accomplished installing *FreeTDS*[23] and *pyodbc*[24] requirements.

```
cartography_install_layers_configurator_prerequisites(){
  field_tolog_section "** Install layer configurator prerequisites **"
  yum --disablerepo="*" --enablerepo="fielding*" -y install gcc gcc
   -c++ python-devel unixODBC unixODBC-devel 2>&1 |
   field_tolog_pipeline
  rpm -Uvh $FIELD_SCRIPT_DIR/c72/freetds/freetds-0.95.81-1.el7.
   x86_64.rpm 2>&1 | field_tolog_pipeline
  rpm -qi freetds-0.95.81-1.el7.x86_64 2>&1 | field_tolog_pipeline
  if [ "$?" -eq "0" ]; then
    field_tolog "freeTDS installed."
  else
    field_tolog "Error installing freeTDS"
  fi
  pushd $FIELD_SCRIPT_DIR/c72/pyodbc/ > /dev/null
  tar xzf pyodbc-3.0.10.tar.gz > /dev/null
  pushd $FIELD_SCRIPT_DIR/c72/pyodbc/pyodbc-3.0.10/ > /dev/null
  python setup.py build > /dev/null
  python setup.py install > /dev/null
  popd > /dev/null
  popd > /dev/null
  cp -f $FIELD_SCRIPT_DIR/c72/conf/odbcinst.ini /etc/odbcinst.ini

}
```

Listing 4.33: Install prerequisites to create styles and layers

Create appropriate styles and layers through geoserver REST configuration APIs

A *python* script is executed with the *GEOCONFIGXML* and *MAXCONNECTIONS* parameters. The *GEOCONFIGXML* contains the connection strings to the *Barn MSSQL* database, that has all the data needed to access the *Atlas* server. The *MAXCONNECTIONS* defines how the *GeoServer* layers will be created.

```
cartography_update_layers(){
  field_tolog_section "** Update layers **"
  NumberCores=`cat /proc/cpuinfo | grep processor | wc -l`
  NGEOSELECTION=$(($NumberCores/3))
  if [ "$NGEOSELECTION" -lt "1" ];then
    NGEOSELECTION=1
  fi
  NGEOUSERS=$(($NGEOSELECTION*6))
  pushd $FIELD_SCRIPT_DIR/c72/CartographyConfigurator > /dev/null
  MAXCONNECTIONS=$(awk -v geou="$NGEOUSERS" 'BEGIN {print 2.5 *
   geou}' | awk '{$1=int($1)}1')
  if [ "$MAXCONNECTIONS" -lt "10" ];then
        MAXCONNECTIONS=10
  fi
  python geoserverDeploy.py $GEOCONFIGXML $MAXCONNECTIONS 2>&1 |
   field_tolog_pipeline
  popd > /dev/null
  systemctl stop haproxy 2>&1 | field_tolog_pipeline
  GEOTORESTART=$(ls /etc/systemd/system/geoserver*.service | wc -l)
  for (( i=1; i<=$GEOTORESTART; i++))
  do
    field_tolog "Restarting geoserver$i..."
    systemctl restart geoserver$i.service 2>&1 |
   field_tolog_pipeline
    sleep 5
  done
  systemctl start haproxy 2>&1 | field_tolog_pipeline
}
```

Listing 4.34: Create appropriate styles and layers through geoserver REST configuration API

After the configuration, all the *GeoServer* instances are restarted to refresh the changes made to the common data folder.

Configure firewall

The firewall is configured to allow traffic through the port 8080, where *HAProxy* is listening.

```
cartography_setup_configure_firewalld() {
  field_tolog_section "** Configure firewall **"
  firewall-cmd --permanent --zone=fielding --add-port=8080/tcp 2>&1
     | field_tolog_pipeline
  firewall-cmd --reload 2>&1 | field_tolog_pipeline
}
```

Listing 4.35: Configure firewall

4.8.3. Unattended installation

To execute an unattended installation the following parameters have to be provided:

- **-s:** Silent installation flag.
- **-h:** Barn host/IP address.
- **-i:** Barn instance name.
- **-d:** Barn database name.

An example of an unattended install is as follows:

```
setupCartography_v4.0.11.16.bin -s -h XXX.XXX.XXX.XXX -i FIELDING -i Fielding
```

CHAPTER 5

Testing

When tackling a change of procedures and workflows in a software solution with so many moving parts, one thing that can be seen clearly is that something is going to fail sooner or later. The lack of standardized testing was one of the main concerns when writing this dissertation and from the beginning was a key factor to decide the platform and workflow to use. The decission was to use during development *Gitlab* paired with *Robot*, with the first providing the *continuous integration* pipeline and the latter the test definition and execution. During a release generation *Jenkins* would be paired with *Robot*, being *Jenkins* the choice because the *Maptics* codebase resides in *Team Foundation Server* and was easier to integrate, pending the *Maptics* codebase migration to *Gitlab*.

All the tests defined during this *Degree Final Work* for the *Maptics* components can be broken down to *build verification* tests and *integration* tests. As all the testing before this dissertation was manual and without a clear procedure, every *build verification* test suite has been defined and developed alongside the development of the installer itself, executing the tests with every commit to the code repository. After the whole deployment solution was developed and tested, a set of *integration* test suites was defined to ensure that the servers were properly configured to be work with each other.

During the development phase, only the *build verification* tests are applied, whereas in during the generation of a release, both types of tests are executed. In this chapter, both types of testing (build verification and integration) are detailed. After that, both scenarios and their implications are examined.

5.1 Build Verification Tests

The *Build verification tests* are tests that ensure that a component is installed correctly and the operative system has been configured properly.

The typical testing contexts and their examples are:

1. **Services:** Service exist, service is running, service is executed by, and so on.
2. **Firewall:** Specific ports are open.
3. **Application:** Application is installed, requirement is installed.
4. **Files and folders:** Files and folders that should be present exist.

A comprehensive specification of the *build verification tests* defined and automated for each component can be found in appendix B.

5.2 Integration Tests

The *Integration tests* are tests that ensure that a component reaches the necessary resources to be able to function normally. The typical testing contexts and their examples are:

1. **Connectivity:** the component is capable to reach the resources it needs.

2. **Monitoring:** the component is capable to reach the monitoring central servers.

A comprehensive specification of the *integration tests* defined and automated for each component can be found in appendix C.

5.3 Testing Scenario: Continuous Integration

As explained in chapter 2, the workflow approach to develop and mantain the installers is trunk-based development. In order to mantain the sanity of the code, a *continuous integration pipeline* is defined for each project. This pipeline will be executed each time someone commits to the main branch and has the following structure, divided in stages:

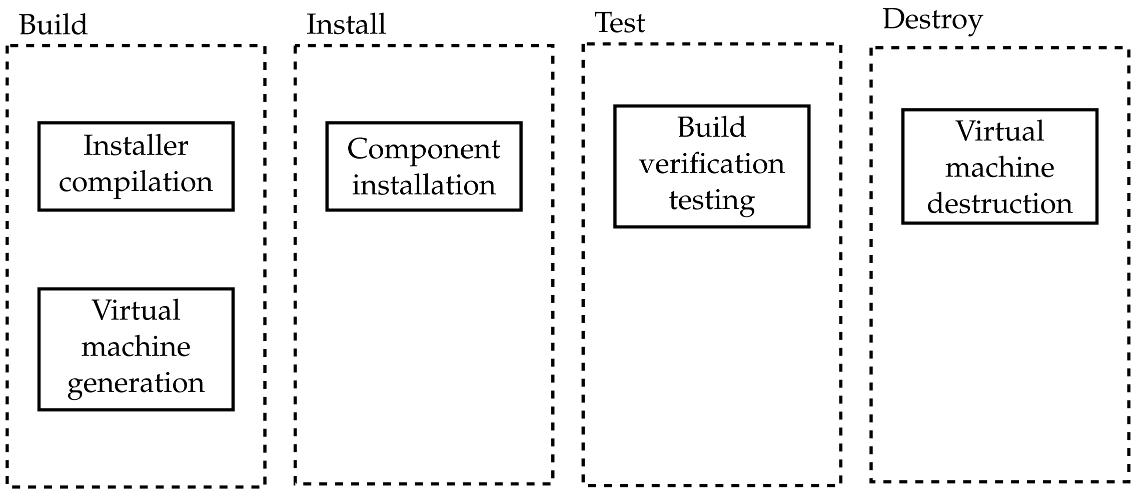

Build
Install
Test
Destroy
Installer compilation
Virtual machine generation
Component installation
Build verification testing
Virtual machine destruction

Figure 5.1: Continuous integration pipeline

In order to keep the execution time under ten minutes, the more intensive steps (*Installer compilation* and *Virtual machine generation*) are executed concurrently. When all steps in a stage are completed, the next stage is executed. It is important to remark that the created virtual machine is identical to the machine that will be deployed in a production environment for that component.

If all the stages pass successfully we can be sure that the commit did not break the installer and that the component can be installed without problems in a production environment.

5.4 Testing Scenario: Release

Using the new development workflow and the tools at our disposal, the following pipeline has been developed with an automation of processes and testing in mind:

1. The *Development* team generates a release with *Team Foundation Server* (TFS).

2. If the release is generated successfully, TFS sends a web hook to begin a job in Jenkins.

3. A sanitiy check is executed on the generated release.

4. The installers' code from the GitLab repository is downloaded and coupled with the generated binary files. The installers are compiled.

5. A release package is generated grouping the compiled installers and the current version documentation.

6. A complete environment is created from scratch duplicating the parameters of a production environment.

7. The generated release is installed in the new environment. *Build verification tests* are executed

8. *Integration tests* are executed.

9. If the *integration tests* pass, the *QA* testing suite begins execution.

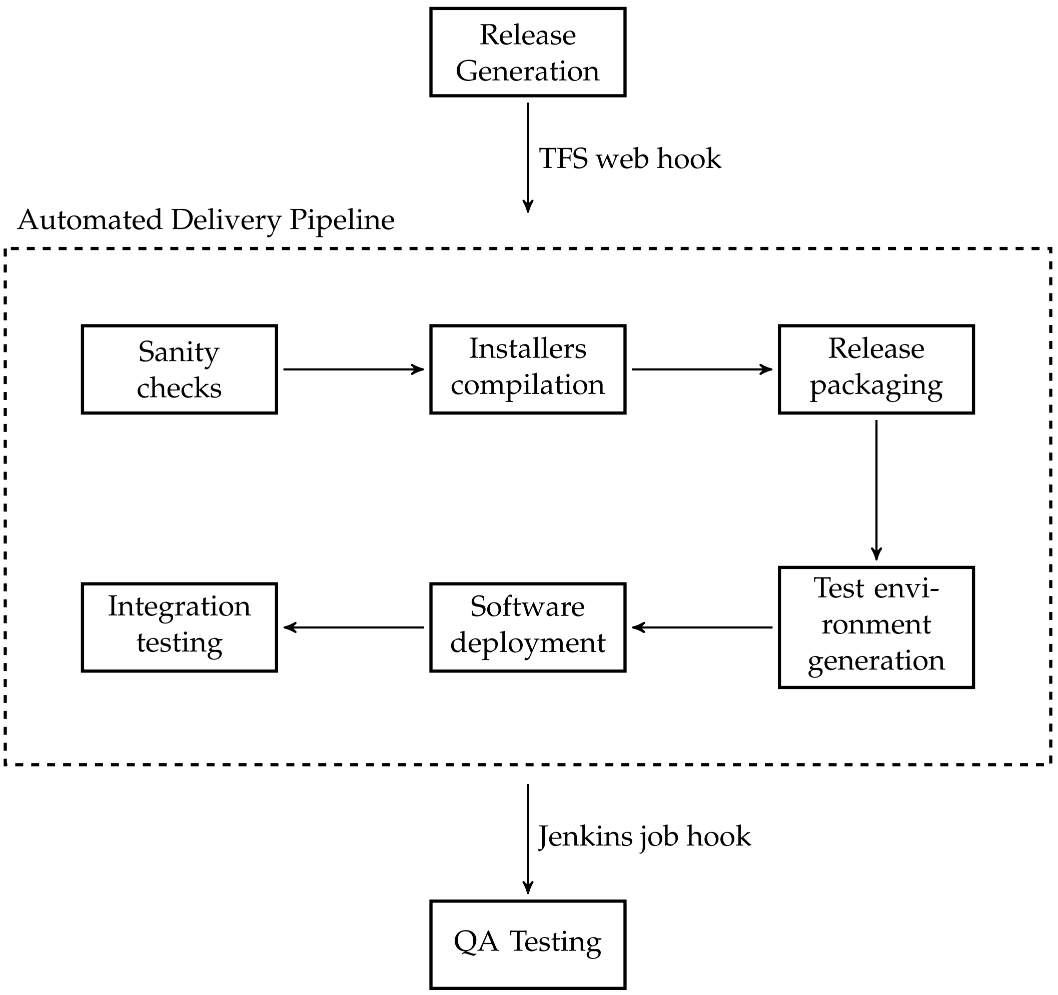


Figure 5.2: Automated Delivery Pipeline

This scenario allows to anyone with permissions in the TFS server to generate a release and execute the defined testing suite. Moreover, the *continuous integration* methodology allows us to upgrade any of our existing environments without fear of breaking the installation. In comparison with the old pipeline (figure 1.2), the new pipeline allows to release a version implicating less people, executing more tests and performing the whole process faster.

CHAPTER 6

Conclusions

When working in a big company with a highly complex software architecture, its important to take into account that changes made can affect other components. From changing the way the events are logged, thus making centralized logging unusable, to storing data slightly differently denying retrieval to a middleware application or even not considering security best practices that will be applied in a client's production environment, incurring in malfunctions.

Unfortunately, making all departments follow guidelines that will result in a more robust software solution but imply extra work can be challenging, more so if said departments are already overworked and advantages are difficult to see as they will be only perceptible on other teams or even directly in production environments. Nevertheless, with a plan and KPIs (*Key Performance Indicator*) defined, the convenience of adopting certain standards and procedures should be clear enough.

Redesigning how a whole software solution is deployed is not a simple task, specially when you have to take into account multiple database engines, load balancers, connection poolers, web applications and application monitors. Nevertheless, having a standard implemented pays off. The possibility of automating the installation and having the assurance of being able of consistently deploy systems in the exact same way benefits the exploration of bugs and malfunctions in the software solution. The completion of this Degree Final Work allows the deployment of all the *Maptics Solution* following the same practices as the other *Fielding* software solutions.

Following standards for installation paths, shared resources and installation and update procedures is key when working in a multi-solution software company. Neglecting this aspect (the origin of this Degree Final Work) closes doors for automation and integration with other systems within the company. Other advantage of the unification of common practices is the reduction of the training overhead for *Operations* teams in order to be proficient in more than one standard.

The application of a *continuous integration* to the installers' code has made also possible to improve the code quality, with each commit triggering a pipeline consisting of compiling the installer, spinning up an ephemeral virtual machine, installing the component and passing a test suite. All automatically and under ten minutes. Putting the installers' code into version control has also made possible to retrieve the exact version of the installer's code when a bug is detected.

The results down the line still have to be measured properly through *Key Performance Indicators*, but as per the immediate results it is now possible to deploy ephemeral predefined environments with all our components in under two hours without any need for human interaction until the process is complete. Previously, a member of the *System Integration* team had to install manually everything, from virtual machines to software components, taking at least one day of work. These changes introduced with this Degree Final Work result in a more efficient use of the department's time allowing to devote time to deeply test the software solution instead of just building it.

Through the duration of this Degree Final Work obstacles were found, so were some aspects of the work that could benefit from some improvements that were not pursued be it for lack of resources or for falling out of the scope of the dissertation. In the next sections, there is an enumeration of the problems and improvements found along the consecution of this Degree Final Work.

6.1 Problems

Standard practices not being enforced

On its early stages of the *Degree Final Work*, a lot of time was devoted to clearly define what were the standards in the sphere of installers and configuration, as studying other software solutions within the company made clear that everyone was doing things slightly differently.

The correct specifications have been followed to industrialize the *Maptics* solution, but while this falls out of the scope of the project, it is important to note that this issue produces bugs and malfunctions when interfacing with other software solutions within the company. To the day of this dissertation this problem persists, making necessary bespoke solutions for each issue.

Disconnection between Development teams and the Operation teams

Being part of the *System Integration* team, makes regular communication with the *Operation* and *Development* necessary in order to obtain feedback. This has brought to light that a lot of issues that arise in *production* could have been avoided if the *Development team's* knowledge about the constraints of the environments the solution is being deployed was better.

6.2 Improvements

Expose the development team to support

Making available the development teams as technical expert support would expose them to the production environments and make them aware of the conditions their software have to operate within.

Better in-house logging and monitoring

The cross-solution monitoring software chosen in *Fielding* is based on Nagios. It works well when deployed on the field as all we want to monitor is defined. Modern monitoring tools like Kibana[1] or Grafana[2] let you analyze freely your logs when you want to explore raw data.

[1] *Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack.* `https://www.elastic.co/products/kibana`

[2] *The open platform for beautiful analytics and monitoring.* `https://grafana.com/`

Provisioning and change management

Provisioning and change management is key to being able to pinpoint causes of systems failure. Currently, the provisioning and configuration is being specified in VMware Realize Orchestrator workflows that are packaged in a non-text format. This means that if we would want to see the change that broke a build, we have to rely on third-party converters instead of seeing just the change in our version control manager, as we would with the rest of our code.

For this matter the *Fielding Field Marshall* software, based on Saltstack, could be leveraged to deploy and configure the virtual machines, as the configuration files in Salstack are written in *YAML*(*Yet Another Markup Language*) format.

Standard testing environments for development teams

In the staging and testing environments a higher number of issues than desired are caused by development testing in non-standard environments, mainly their own machines where they develop their code. Making available ephemeral virtual machines with production-like configuration would greatly reduce this errors, being able to fail earlier in the development cycle.

Bibliography

[1] Kief Morris. *Infrastructure as Code. Managing servers in the cloud*. O'Reilly Media Inc., Sebastopol, first edition, 2016.

[2] Gene Kim, Jez Humble, Patrick Debois, John Willis, John Allspaw. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, October 6, 2016.

[3] Jez Humble, David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, first edition July 27, 2010.

[4] Scott Chacon, Ben Straub. *Pro Git* Apress, Second edition.

[5] Nullsoft Scriptable Install System, 12 August 2016. Retrieved 15 May 2017 from `http://nsis.sourceforge.net/Main_Page`.

[6] *dialog* man page, 17 October 2011. Retrieved 23 May 2017 from `http://man.he.net/?topic=dialog&section=all`.

[7] Install SQL Server 2012 Using a Configuration File, 13 August 2017. Retrieved 20 August 2017 from `https://technet.microsoft.com/en-us/library/dd239405(v=sql.110).aspx`.

[8] Server Manager Cmdlets in Windows PowerShell, 13 August 2017. Retrieved 20 August 2017 from `https://technet.microsoft.com/en-us/library/jj205465(v=wps.630).aspx`.

[9] Web Server (IIS) Administration Cmdlets in Windows PowerShell, 13 August 2017. Retrieved 20 August 2017 from `https://technet.microsoft.com/en-us/library/ee790599.aspx`.

[10] Redis Documentation. Retrieved 12 July 2017 from `https://redis.io/documentation`.

[11] GDAL - Geospatial Data Abstraction Library. Retrieved 20 July 2017 from `http://www.gdal.org/`.

[12] Shuttle Radar Topography Mission. Retrieved 8 July 2017 from `https://www2.jpl.nasa.gov/srtm/`.

[13] PostgreSQL: The world's most advanced open source database. Retrieved 15 June 2017 from `https://www.postgresql.org/`.

[14] PostGIS - Spatial and Geographic objects for PostgreSQL. Retrieved 15 June 2017 from `http://postgis.net/`.

[15] PgBouncer - lightweight connection pooler for PostgreSQL. Retrieved 15 June 2017 from https://pgbouncer.github.io/.

[16] Createrepo. Retrieved 18 July 2017 from http://createrepo.baseurl.org/.

[17] createrepo - Linux man page. Retrieved 18 July 2017 from https://linux.die.net/man/8/createrepo.

[18] pyscopg2 2.7.3: Python Package Index. Retrieved 18 July 2017 from https://pypi.python.org/pypi/psycopg2.

[19] GeoServer. Retrieved 10 June 2017 from http://geoserver.org/.

[20] HAProxy -The Reliable, High Performance TCP/HTTP Load Balancer. Retrieved 10 June 2017 from http://www.haproxy.org/.

[21] How do I arrange a silent (unattended) Java installation?, 01 June 2016. Retrieved 14 July 2017 from https://www.java.com/en/download/help/silent_install.xml.

[22] Jai ImageIO core. Retrieved 18 August 2017 from https://github.com/jai-imageio/jai-imageio-core.

[23] FreeTDS, making the leap to SQL Server. Retrieved 18 August 2017 from http://www.freetds.org/.

[24] Pyodbc, DB API Module for ODBC. Retrieved 18 August 2017 from https://pypi.python.org/pypi/pyodbc.

APPENDIX A

Installation libraries

A.1 Windows Installation Libraries

A.1.1. User creation helper functions

```
Function IsMemberOf{
  param(
    [String]$localGroup,
    [String]$localUser
  )
  #Get the group
  $localGroupObject = [ADSI]"WinNT://localhost/$localGroup"
  $Members = @($localGroupObject.psbase.Invoke("Members"))
  #Populate the $MemberNames array with all the user ID s
  $MemberNames = @()
  $Members | ForEach-Object {$MemberNames += $_.GetType().
   InvokeMember("Name", 'GetProperty', $null, $_, $null);}

  #See if your user ID is in there
  if (-Not $MemberNames.Contains($localUser)) {
      return $false
  }
  return $true
}

Function Create-User{
    param(
        [string]$UserToCreate,
        [string]$PasswordToSet
    )
    net user $UserToCreate $PasswordToSet /add /PASSWORDCHG:no

}

Function Add-UserToGroup{
    param(
        [string]$GroupToBelong,
        [string]$UserToAdd
    )
    net localgroup $GroupToBelong $UserToAdd /add
}

Function Configure-LocalSecurityPolicies {
    &secedit /configure /cfg C:\Fielding\MapticsMSSQLEnvironment\Security\Templates\MapticsSQLTemplate.inf /db secedit.sdb
  &secedit /configure /cfg C:\Fielding\MapticsMSSQLEnvironment\Security\Templates\DenyLogonGeo.inf /db secedit.sdb
}

Function Configure-MapticsUsers{

        #Create MAP_GEO group
      Print-Log "Creating MAP_GEO group..."
      if(-not([ADSI]::Exists('WinNT://./MAP_GEO'))){
        net localgroup MAP_GEO /add /comment:"Maptics Installation group"
        Print-Log "Group created."
      }else{
        Print-Log "Group already exists."
      }
        #Create geo_maptics user
      Print-Log "Creating geo_maptics user..."
      if(-not([ADSI]::Exists('WinNT://./geo_maptics'))){
        net user geo_maptics Password /add /PASSWORDCHG:no /USERCOMMENT:" maptics  User for shared folders" /Y
      WMIC USERACCOUNT WHERE "Name='geo_maptics'" SET PasswordExpires=FALSE
        Print-Log "User created."
      }else{
        Print-Log "User already exists."
      }
        #Add geo_maptics to MAP_GEO
      Print-Log "Adding geo_maptics to MAP_GEO..."
      if(-not(IsMemberOf MAP_GEO geo_maptics)){
            Add-UserToGroup "MAP_GEO" "geo_maptics"
        Print-Log "User added."
      }else{
        Print-Log "User already belongs to the group."
      }
        #Add geo_maptics to Administrators
      Print-Log "Adding geo_maptics to Administrators..."
      if(-not(IsMemberOf Administrators geo_maptics)){
            Add-UserToGroup "Administrators" "geo_maptics"
        Print-Log "User added."
      }else{
        Print-Log "User already belongs to the group."
      }
}
```

Listing A.1: User creation helper functions

A.1.2. Folder creation helper functions

```

Function Create-DataDBFolder{
    param(
        [string]$drive
    )
    if($InstallData["DeploymentType"] -eq "Standard"){
        $pathToTest=($InstallData["DBDrives"][$drive]+"\Fielding\Maptics\DatabaseFiles")
```

```
      }else{
        $pathToTest=($InstallData["DBDrives"][$drive]+"\Fielding\Maptics\$($InstallData["OperatorName"])\DatabaseFiles")
      }
      Print-Log "Creating $drive folders($pathToTest)..."
      if(-not(Test-Path $pathToTest)){
        New-Item -Path $pathToTest -ItemType directory -Force | Out-Null
        Print-Log "Folder created."
      }else{
        Print-Log "Folder already exists."
  }
}

Function Create-TempDBFolder{
    param(
        [int]$tempdrive
    )
     if($InstallData["DeploymentType"] -eq "Standard"){
        $pathToTest=($InstallData["DBDrives"]["Temp"][$tempdrive]+"\Fielding\Maptics\DatabaseFiles")
   }else{
        $pathToTest=($InstallData["DBDrives"]["Temp"][$tempdrive]+"\Fielding\Maptics\$($InstallData["OperatorName"])\DatabaseFiles")
   }
   Print-Log "Creating Temp$tempdrive folders($pathToTest)..."
   if(-not(Test-Path $pathToTest)){
      New-Item -Path $pathToTest -ItemType directory -Force | Out-Null
    Print-Log "Folder created."
   }else{
      Print-Log "Folder already exists."
   }
}

Function Create-BackupFolder{
    param(
        [string]$drive
    )
    if($InstallData["DeploymentType"] -eq "Standard"){
        $pathToTest=($InstallData["DBDrives"][$drive]+"\Fielding\Maptics\DatabaseFiles\Backups")
      }else{
        $pathToTest=($InstallData["DBDrives"][$drive]+"\Fielding\Maptics\$($InstallData["OperatorName"])\DatabaseFiles\Backups")
      }
      Print-Log "Creating $drive folders($pathToTest)..."
      if(-not(Test-Path $pathToTest)){
        New-Item -Path $pathToTest -ItemType directory -Force | Out-Null
        Print-Log "Folder created."
      }else{
        Print-Log "Folder already exists."
  }
}
```

Listing A.2: Folder creation helper functions

A.1.3. Log printing helper functions

```
Function Print-LineToLog{
    Param(
        [string]$Line,
        [string]$PathToLog = "$InstallLogFileName"
    )
  $FormatedDate=$(Get-Date -format "d/M/yyyy HH:mm:ss")
    Add-Content $PathToLog "$FormatedDate$ - MapticsMSSQLEnvironment - [info] $Line "
}

Function Print-Log{
  Param(
    [string]$Line
  )
  Write-Host $Line
  Print-LineToLog $Line
}
```

Listing A.3: Log printing helper functions

A.1.4. Firewall helper functions

```
Function IsFirewallEnabled{
  $HKLM = 2147483650
  $reg = get-wmiobject -list -namespace root\default | where-object { $_.name -eq "StdRegProv" }
  $firewallEnabled = $reg.GetDwordValue($HKLM, "System\ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\DomainProfile","EnableFirewall")
  return ([bool]($firewallEnabled.uValue))
}
```

Listing A.4: Firewall helper functions

A.1.5. MSSQL bootstrap configuration file

```
;SQL Server 2012 Configuration File
[OPTIONS]

IACCEPTSQLSERVERLICENSETERMS="True"
; Specifies a Setup work flow, like INSTALL, UNINSTALL, or UPGRADE. This is a required parameter.

ACTION="Install"

; Detailed help for command line argument ENU has not been defined yet.

ENU="True"

; Setup will not display any user interface.

QUIET="False"

```

```
; Setup will display progress only, without any user interaction.

QUIETSIMPLE="True"

; Specify whether SQL Server Setup should discover and include
   product updates. The valid values are True and False or 1 and 0.
    By default SQL Server Setup will include updates that are found
   .

UpdateEnabled="False"

; Specifies features to install, uninstall, or upgrade. The list of
    top-level features include SQL, AS, RS, IS, MDS, and Tools. The
    SQL feature will install the Database Engine, Replication,
   Full-Text, and Data Quality Services (DQS) server. The Tools
   feature will install Management Tools, Books online components,
   SQL Server Data Tools, and other shared components.

FEATURES=SQLENGINE,IS,BC,SSMS,ADV_SSMS,SNAC_SDK

; Specify the location where SQL Server Setup will obtain product
   updates. The valid values are "MU" to search Microsoft Update, a
    valid folder path, a relative path such as .\MyUpdates or a UNC
    share. By default SQL Server Setup will search Microsoft Update
    or a Windows Update service through the Window Server Update
   Services.

UpdateSource="MU"

; Displays the command line parameters usage

HELP="False"

; Specifies that the detailed Setup log should be piped to the
   console.

INDICATEPROGRESS="False"

; Specifies that Setup should install into WOW64. This command line
    argument is not supported on an IA64 or a 32-bit system.

X86="False"

; Specify the root installation directory for shared components.
   This directory remains unchanged after shared components are
   already installed.

INSTALLSHAREDDIR="C:\Program Files\Microsoft SQL Server"

; Specify the root installation directory for the WOW64 shared
   components.  This directory remains unchanged after WOW64 shared
    components are already installed.

INSTALLSHAREDWOWDIR="C:\Program Files (x86)\Microsoft SQL Server"

; Specify a default or named instance. MSSQLSERVER is the default
   instance for non-Express editions and SQLExpress for Express
   editions. This parameter is required when installing the SQL
   Server Database Engine (SQL), Analysis Services (AS), or
   Reporting Services (RS).

INSTANCENAME="MAPTICS"

; Specify the Instance ID for the SQL Server features you have
   specified. SQL Server directory structure, registry structure,
   and service names will incorporate the instance ID of the SQL
   Server instance.

INSTANCEID="MAPTICS"

; Specify that SQL Server feature usage data can be collected and
   sent to Microsoft. Specify 1 or True to enable and 0 or False to
    disable this feature.

SQMREPORTING="False"

; Specify if errors can be reported to Microsoft to improve future
   SQL Server releases. Specify 1 or True to enable and 0 or False
   to disable this feature.

ERRORREPORTING="False"

; Specify the installation directory.

INSTANCEDIR="C:\Program Files\Microsoft SQL Server"

; Agent account name

AGTSVCACCOUNT="geo_maptics"

; Auto-start service after installation.

AGTSVCSTARTUPTYPE="Automatic"

; Startup type for Integration Services.

ISSVCSTARTUPTYPE="Automatic"

; Account for Integration Services: Domain\User or system account.

ISSVCACCOUNT="NT Service\MsDtsServer110"

; CM brick TCP communication port

COMMFABRICPORT="0"

; How matrix will use private networks

COMMFABRICNETWORKLEVEL="0"

; How inter brick communication will be protected

COMMFABRICENCRYPTION="0"

; TCP port used by the CM brick

MATRIXCMBRICKCOMMPORT="0"
```

```

; Startup type for the SQL Server service.

SQLSVCSTARTUPTYPE="Automatic"

; Level to enable FILESTREAM feature at (0, 1, 2 or 3).

FILESTREAMLEVEL="0"

; Set to "1" to enable RANU for SQL Server Express.

ENABLERANU="False"

; Specifies a Windows collation or an SQL collation to use for the
    Database Engine.

SQLCOLLATION="Modern_Spanish_CI_AS"

; Account for SQL Server service: Domain\User or system account.

SQLSVCACCOUNT="geo_maptics"

; Windows account(s) to provision as SQL Server system
    administrators.

;SQLSYSADMINACCOUNTS="BUILTIN\Administrators"

; The default is Windows Authentication. Use "SQL" for Mixed Mode
    Authentication.

SECURITYMODE="SQL"

; Default directory for the Database Engine backup files.

SQLBACKUPDIR="J:\Fielding\Maptics\DatabaseFiles\Backups"

; Default directory for the Database Engine user databases.

SQLUSERDBDIR="D:\Fielding\Maptics\DatabaseFiles"

; Default directory for the Database Engine user database logs.

SQLUSERDBLOGDIR="E:\Fielding\Maptics\DatabaseFiles"

; Directory for Database Engine TempDB files.

SQLTEMPDBDIR="F:\Fielding\Maptics\DatabaseFiles"

; Specify 0 to disable or 1 to enable the TCP/IP protocol.

TCPENABLED="1"

; Specify 0 to disable or 1 to enable the Named Pipes protocol.

NPENABLED="0"

; Startup type for Browser Service.

BROWSERSVCSTARTUPTYPE="Automatic"
```

Listing A.5: MSSQL bootstrap configuration file

A.1.6. MSSQL configuration helper functions

```
Function Set-SQLPort{
if($SQLVersion -eq "SQL Server 2012"){
    $SqlObject = get-WMiObject -Namespace "root\microsoft\SQLServer
    \ComputerManagement11" -Class ServerNetworkProtocolProperty
    -filter "ProtocolName='Tcp' AND PropertyName='TcpDynamicPorts'
    AND IpAddressName='IPAll'"
    if($SqlObject.PropertyStrVal -ne ""){
        $SqlObject.SetStringValue('')
    }
    $SqlObject = get-WMiObject -Namespace "root\microsoft\SQLServer
    \ComputerManagement11" -Class ServerNetworkProtocolProperty
    -filter "ProtocolName='Tcp' AND PropertyName='TcpPort' AND
    IpAddressName='IPAll'"
    if($SqlObject.PropertyStrVal -ne "1433"){
        $SqlObject.SetStringValue(1433)
    }
  }else{
  $SqlObject = get-WMiObject -Namespace "root\microsoft\SQLServer\
    ComputerManagement12" -Class ServerNetworkProtocolProperty
    -filter "ProtocolName='Tcp' AND PropertyName='TcpDynamicPorts'
    AND IpAddressName='IPAll'"
    if($SqlObject.PropertyStrVal -ne ""){
        $SqlObject.SetStringValue('')
    }
    $SqlObject = get-WMiObject -Namespace "root\microsoft\SQLServer
    \ComputerManagement12" -Class ServerNetworkProtocolProperty
    -filter "ProtocolName='Tcp' AND PropertyName='TcpPort' AND
    IpAddressName='IPAll'"
    if($SqlObject.PropertyStrVal -ne "1433"){
        $SqlObject.SetStringValue(1433)
    }

  }
    Restart-Service 'MSSQL$MAPTICS' -Force
}


Function Configure-SQLServer{
  $env:PSModulePath = [System.Environment]::GetEnvironmentVariable(
    "PSModulePath","Machine")
  Import-Module SQLPS
  [int]$mem=Get-CimInstance Win32_PhysicalMemory | Measure-Object
    -Property capacity -Sum | Foreach {"{0:N2}" -f ([math]::round((
    $_.Sum / 1MB)))}

  If($mem -lt 256000){
    $mem = $mem*0.8
  }else{
    $mem = $mem*0.9
  }

  $query="use master;
```

```
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE WITH OVERRIDE;
GO
sp_configure 'show advanced options', 1;
GO
RECONFIGURE WITH OVERRIDE;
GO
sp_configure 'max degree of parallelism', 4;
GO
RECONFIGURE WITH OVERRIDE;
GO
sp_configure 'Database Mail XPs', 1;
GO
RECONFIGURE WITH OVERRIDE;
GO
sp_configure 'max server memory', $mem;
RECONFIGURE WITH OVERRIDE;
GO"

Print-Log "Configuring SQL Server"
Push-Location
Invoke-Sqlcmd -ServerInstance localhost\MAPTICS -Query $query
Pop-Location

$testTempDBFiles="select Count (*) from sys.master_files mf
where database_id =2 and type_desc= 'rows'"

Push-Location
$nTempDB=(Invoke-Sqlcmd -ServerInstance localhost\MAPTICS -Query
  $testTempDBFiles)[0]
Pop-Location

If($nTempDB -eq 12){
   Print-Log "TempDB files already configured"
}else{
   Print-Log "Configuring TempDB files"
[int]$disksize=(Get-WmiObject Win32_LogicalDisk -Filter "DeviceID
  ='H:'").Size / 1MB
$disksize=$disksize*0.3

$TempfilesCreatorQuery="/*MAXSIZE Variable is 30% of dedicated
  disk. The size is in MB*/
DECLARE @sizeMax INT = $disksize
/*Drives dedicated to tempdb*/
DECLARE @tempDisk1 varchar(max) = 'H'
DECLARE @tempDisk2 varchar(max) = 'I'
DECLARE @tempDisk3 varchar(max) = 'J'
DECLARE @tempDisk4 varchar(max) = 'K'
DECLARE @sql_temp varchar(max)
DECLARE @sql varchar(max)

/*H:*/
SET @sql_temp = 'alter Database tempdb modify file(
NAME=N''##name##'', NEWNAME=N''##newname##'')'
set @sql = REPLACE(@sql_temp, '##name##', N'tempdev')
set @sql = REPLACE(@sql, '##newname##', N'tempdb1')
EXECUTE(@sql)
```

```

SET @sql_temp = 'alter Database tempdb modify file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb1.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb1')
set @sql = REPLACE(@sql,'##path##',@tempDisk1)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

SET @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb5.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb5')
set @sql = REPLACE(@sql,'##path##',@tempDisk1)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

SET @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb9.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb9')
set @sql = REPLACE(@sql,'##path##',@tempDisk1)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

/*I:*/
set @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb2.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb2')
set @sql = REPLACE(@sql,'##path##',@tempDisk2)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

set @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb6.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb6')
set @sql = REPLACE(@sql,'##path##',@tempDisk2)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

set @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb10.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb10')
set @sql = REPLACE(@sql,'##path##',@tempDisk2)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

/*J:*/
SET @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb3.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb3')
set @sql = REPLACE(@sql,'##path##',@tempDisk3)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

SET @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb7.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb7')
set @sql = REPLACE(@sql,'##path##',@tempDisk3)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

SET @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb11.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb11')
set @sql = REPLACE(@sql,'##path##',@tempDisk3)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

/*K:*/
SET @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb4.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb4')
set @sql = REPLACE(@sql,'##path##',@tempDisk4)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

set @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb8.
 ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
 ##)'
set @sql = REPLACE(@sql_temp,'##name##', N'tempdb8')
set @sql = REPLACE(@sql,'##path##',@tempDisk4)
set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
EXECUTE(@sql)

set @sql_temp = 'alter Database tempdb add file(
NAME = N''##name##'',
  FILENAME = N''##path##:\Fielding\Maptics\DatabaseFiles\tempdb12.
   ndf'' , SIZE = 100MB , FILEGROWTH = 100MB, MAXSIZE = ##sizemax
   ##)'
  set @sql = REPLACE(@sql_temp,'##name##', N'tempdb12')
  set @sql = REPLACE(@sql,'##path##',@tempDisk4)
  set @sql = REPLACE(@sql,'##sizeMax##',@sizeMax)
  EXECUTE(@sql)"

  Push-Location
  Invoke-Sqlcmd -ServerInstance localhost\MAPTICS -Query
   $TempfilesCreatorQuery
  Pop-Location
  }

  Restart-Service 'MSSQL$MAPTICS' -Force
}
```

Listing A.6: MSSQL configuration helper functions

A.2 Linux Installation Libraries

A.2.1. User creation helper function

```
maptics_setup_create_geo_fielding_user(){
  getent group FIELD_GEO >/dev/null || groupadd -g 35106 FIELD_GEO
  getent passwd geo_fielding >/dev/null || \
  useradd -u 35006 -g FIELD_GEO --key UMASK=007 geo_fielding ; \
  echo geo_fielding:pwd | chpasswd
  grep -w geo_fielding /etc/passwd >/dev/null && chage -I -1 -m 0 -
   M 99999 -E -1 geo_fielding > /dev/null 2>&1
}
```

Listing A.7: User creation helper function

APPENDIX B
Build Verification Tests

B.1 Barn Server

The *Barn* server should pass the following tests for each context:

B.1.1. Services

Description	Test
MSSQL Server service exists	"MSSQL$MAPTICS" service should exist
MSSQL Server service is running	"MSSQL$MAPTICS" service should be running
MSSQL Server service is executed by "geo_fielding"	"MSSQL$MAPTICS" service should be executed by "geo_fielding"
MSSQL Agent service exists	"SQLAGENT$MAPTICS" service should exist
MSSQL Agent service is running	"SQLAGENT$MAPTICS" service should be running
MSSQL Agent service is executed by "geo_fielding"	"SQLAGENT$MAPTICS" service should be executed by "geo_fielding"

B.1.2. MSSQL files

Description	Test
Log file for Repo DB exists	"F:\Fielding\Data\Log\Maptics_log.ldf" file should exist
Log file for Admin DB exists	"F:\Fielding\Data\Log\Maptics_Admin_log.ldf" file should exist
One MessageList file exists	At least one file with pattern *.ndf should exist in "D:\Fielding\Data\MessageLists"
One ThreadFlow file exists	At least one file with pattern *.ndf should exist in "D:\Fielding\Data\ThreadFlows"

B.1.3. Firewall

Description	Test
MSSQL server is reachable	Port 1433 should be open in the firewall

B.1.4. Application

Description	Test
Barn is listed as installed	*Barn* section should be present in "C:\Fielding\install.ini"
Barn is listed as installed	*MSSQL* server registry key should exist

B.2 Admin GUI

The *Admin GUI* server should pass the following tests for each context:

B.2.1. Internet Information Services

Description	Test
IIS requirements are installed	"Web-Server","Web-Net-Ext45", "Web-ASP", "Web-Asp-Net45", "Web-ISAPI-Ext", "Web-ISAPI-Filter", "Web-Http-Redirect", "Web-Filtering", "Web-Security", "Web-Client-Auth","Web-Digest-Auth", "Web-Cert-Auth", "Web-Url-Auth", "Web-Windows-Auth" should be installed

B.2.2. Firewall

Description	Test
Admin GUI is reachable	Port 81 should be open in the firewall

B.2.3. Application

Description	Test
Admin GUI is listed as installed	*Admin GUI* section should be present in "C:\Fielding\install.ini"
Admin GUI is listed as installed	*Admin GUI* server registry key should exist

B.3 Maptics Monitor

The *Maptics Monitor* server should pass the following tests for each context:

B.3.1. Monitor

Description	Test
Monitor binaries folder exists	"C:\Fielding\Monitor" should exist
Monitor executable exists	"C:\Fielding\Monitor\maptics.Monitor.exe" should exist

B.3.2. Apache server

Description	Test
Apache server is installed	"httpd" service should be installed

B.3.3. Java

Description	Test
Correct Java version is installed	Java version should be 6u45

B.3.4. Firewall

Description	Test
Monitor is reachable	Port 8009 should be open in the firewall

B.3.5. Application

Description	Test
Monitor is listed as installed	*Monitor* section should be present in "C:\Fielding\install.ini"
Monitor is listed as installed	*Monitor* server registry key should exist

B.4 Maptics Parsing System

The *Maptics Parsing System* server should pass the following tests for each context:

B.4.1. FPM

Description	Test
FPM binaries folder exists	"C:\Fielding\FPM" should exist
FPM executable exists	"C:\Fielding\FPM\maptics.FPM.exe" should exist

B.4.2. FPM monitor

Description	Test
FPM monitor binaries folder exists	"C:\Fielding\FPM Monitor" should exist
FPM monitor executable exists	C:\Fielding\FPM Monitor\maptics.monitor.exe" should exist

B.4.3. Parser

Description	Test
Parser binaries folder exists	"C:\Fielding\Parser" should exist
Parser executable exists	"C:\Fielding\Parser\maptics.Parser.exe" should exist

B.4.4. Java

Description	Test
Correct Java version is installed	Java version should be 7u80
Environment variable JAVA_HOME is set correctly	“JAVA_HOME” should be “C:\java\jre7”

B.4.5. Hadoop

Description	Test
Hadoop folder exists	C:\Fielding\Hadoop should exist
Environment variable HADOOP_HOME exists	“HADOOP_HOME” environment variable should exist
Environment variable HADOOP_HOME is set correctly	“HADOOP_HOME” should be “C:\Fielding\Hadoop\hadoop-2.7.1”
Environment variable CLASSPATH exists	“CLASSPATH” environment variable should exist
Environment variable CLASSPATH is set correctly	“CLASSPATH” should be “C:\Fielding\Hadoop\hadoop-2.7.1”

B.4.6. MSMQ

Description	Test
MSMQ is installed	“MSMQ” Windows Feature state should be “Installed”

B.4.7. Application

Description	Test
Maptics Parsing system is listed as installed	*Maptics Parsing system* section should be present in “C:\Fielding\install.ini”
Maptics Parsing system is listed as installed	*Maptics Parsing system* registry key should exist

B.5 Granary Server

The *Granary* server should pass the following tests for each context:

B.5.1. Redis

Description	Test
All the *redis* servers are listening	The number of *redis* listeners specified in “/etc/fielding/fielding.ini” should be the same as the number of *redis* servers listening
Redis servers are being executed by “geo_fielding” user	“redis*.service” should be executed by “geo_fielding” user

B.5.2. Firewall

Description	Test
Redis servers are reachable	Ports 6379 to 6934 should be open in the firewall

B.5.3. Application

Description	Test
Granary is listed as installed	*Granary* section should be present in "/etc/fielding/fielding.ini"

B.6 Maptics Processing System

The *Maptics Processing System* server should pass the following tests for each context:

B.6.1. Processor

Description	Test
Processor binaries folder exists	"C:\Fielding\Processor" should exist
Processor executable exists	"C:\Fielding\Processor\maptics.Processor.exe" should exist

B.6.2. Processor monitor

Description	Test
Processor monitor binaries folder exists	"C:\Fielding\Processor Monitor" should exist
Processor monitor executable exists	C:\Fielding\Processor Monitor\maptics.ProcessorMonitor.exe" should exist

B.6.3. Shuttle Radar Topology Mission files

Description	Test
SRTM files exists	"D:\Fielding\SRTM" should contain *.srtm files

B.6.4. SFTP server

Description	Test
SFTP service exists	"SolarWinds SFTP Server" service should exist
SFTP service is running	"SolarWinds SFTP Server" service should be running
SFTP local data folder exists	"D:\Fielding\SFTP_Exports" should exist

B.6.5. Firewall

Description	Test
SFTP server is reachable	Port 22 should be open in the firewall

B.6.6. Application

Description	Test
Maptics Processing system is listed as installed	*Maptics Processing system* section should be present in "C:\Fielding\install.ini"
Maptics Processing system is listed as installed	*Maptics Processing system* registry key should exist

B.7 Atlas Server

The *Atlas* server should pass the following tests for each context:

B.7.1. Services

Description	Test
PosgtreSQL custom service file exists	"/etc/systemd/system/postgresql-9.6.service" should exist
PosgtreSQL service is running	"postgresql-9.6.service" should be running
PosgtreSQL version is correct	*PostgreSQL* version should be 9.6.2
Samba service is running	"smb.service" should be running
Pgbouncer service is running	"pgbouncer.service" should be running

B.7.2. Firewall

Description	Test
PostgreSQL is reachable through port 5432	Port 5432 should be open in the firewall
Pgbouncer is reachable through port 6432	Port 6432 should be open in the firewall
Samba is reachable through port 139	Port 139 should be open in the firewall
Samba is reachable through port 445	Port 445 should be open in the firewall

B.7.3. Application

Description	Test
Atlas is listed as installed	*Atlas* section should be present in "/etc/fielding/fielding.ini"

B.8 Cartography Server

The *Cartography* server should pass the following tests for each context:

B.8.1. Services

Description	Test
Apache service is running	"httpd.service" should be running
Apache service is listening through standard port	Port 80 should be "open/listening"
All *Geoserver* instances are running	The number of *geoserver* instances specified in "/etc/fielding/fielding.ini" should be the same as the number of *geoserver* instances running
Geoserver instances are being executed by "geo_fielding" user	"geoserver*.service" should be executed by "geo_fielding" user
HAproxy service is running	"haproxy.service" should be running
HAproxy service is being executed by "geo_fielding" user	"haproxy.service" should be executed by "geo_fielding" user

B.8.2. Firewall

Description	Test
Apache is reachable through port 80	Port 80 should be open in the firewall
HAproxy is reachable through port 8080	Port 8080 should be open in the firewall

APPENDIX C
Integration Tests

C.1 Barn Server

The *Barn* server should pass the following tests for each context:

C.1.1. Maptics connectivity

Description	Test
Barn server reaches *Maptics Monitor* host	*Maptics Monitor* host is reachable through *Lan_Admin* vlan
Barn server reaches *Maptics Admin GUI* host	*Maptics Admin GUI* host is reachable through *Lan_Intra* vlan
Barn server reaches *Maptics Parsing System* host	*Maptics Parsing System* host is reachable through *Lan_Intra* vlan
Barn server reaches *Maptics Processing System* host	*Maptics Processing System* host is reachable through *Lan_Intra* vlan
Barn server reaches *Maptics Cartography* host	*Maptics Cartography* host is reachable through *Lan_Intra* vlan
Barn server reaches *Granary* host	*Granary* host is reachable through *Lan_Intra* vlan

C.1.2. Monitoring

Description	Test
Barn server can send messages to *Maptics Monitor*	*Maptics Monitor* queue exists

C.2 Admin GUI

C.2.1. Maptics connectivity

Description	Test
Maptics Admin GUI server reaches *Barn* host	*Barn* host is reachable through *Lan_Admin* vlan
Maptics Admin GUI server reaches *Maptics Parsing System* host	*Maptics Parsing System* host is reachable through *Lan_Admin* vlan
Maptics Admin GUI server reaches *Maptics Processing System* host	*Maptics Processing System* host is reachable through *Lan_Admin* vlan
Maptics Admin GUI server reaches *Maptics Cartography* host	*Maptics Cartography* host is reachable through *Lan_Admin* vlan
Maptics Admin GUI server reaches *Atlas* host	*Atlas* host is reachable through *Lan_Admin* vlan
Maptics Admin GUI server reaches *Granary* host	*Granary* host is reachable through *Lan_Admin* vlan

C.3 Maptics Monitor

C.3.1. Maptics connectivity

Description	Test
Maptics Monitor server reaches *Barn* host	*Barn* host is reachable through *Lan_Admin* vlan
Maptics Monitor server reaches *Maptics Parsing System* host	*Maptics Parsing System* host is reachable through *Lan_Admin* vlan
Maptics Monitor server reaches *Maptics Processing System* host	*Maptics Processing System* host is reachable through *Lan_Admin* vlan
Maptics Monitor server reaches *Maptics Cartography* host	*Maptics Cartography* host is reachable through *Lan_Admin* vlan
Maptics Monitor server reaches *Granary* host	*Granary* host is reachable through *Lan_Admin* vlan

C.4 Maptics Parsing System

The *Maptics Parsing System* server should pass the following tests for each context:

C.4.1. Maptics connectivity

Description	Test
Maptics Parsing System server reaches *Maptics Monitor* host	*Maptics Monitor* host is reachable through *Lan_Admin* vlan
Maptics Parsing System server reaches *Barn* host	*Barn* host is reachable through *Lan_Intra* vlan
Maptics Parsing System server reaches *Atlas* host	*Atlas* host is reachable through *Lan_Intra* vlan
Maptics Parsing System server reaches *Granary* host	*Granary* host is reachable through *Lan_Intra* vlan

C.4.2. Monitoring

Description	Test
Maptics Parsing System server can send messages to *Maptics Monitor*	*Maptics Monitor* queue exists

C.5 Granary Server

The *Granary* server should pass the following tests for each context:

C.5.1. Maptics connectivity

Description	Test
Granary server reaches *Maptics Parsing System* host	*Maptics Parsing System* host is reachable through *Lan_Intra* vlan
Granary server reaches *Barn* host	*Barn* host is reachable through *Lan_Intra* vlan

C.6 Maptics Processing System

C.6.1. Maptics connectivity

Description	Test
Maptics Processing System server reaches *Maptics Monitor* host	*Maptics Monitor* host is reachable through *Lan_Admin* vlan
Maptics Processing System server reaches *Barn* host	*Barn* host is reachable through *Lan_Intra* vlan
Maptics Processing System server reaches *Atlas* host	*Atlas* host is reachable through *Lan_Intra* vlan
Maptics Processing System server reaches *Granary* host	*Granary* host is reachable through *Lan_Intra* vlan

C.6.2. Monitoring

Description	Test
Maptics Parsing System server can send messages to *Maptics Monitor*	*Maptics Monitor* queue exists

C.7 Atlas Server

C.7.1. Maptics connectivity

Description	Test
Atlas server reaches *Cartography* host	*Cartography* host is reachable through *Lan_Intra* vlan

C.8 Cartography Server

C.8.1. Maptics connectivity

Description	Test
Cartography server reaches *Atlas* host	*Atlas* host is reachable through *Lan_Intra* vlan