Automatic deployment of a PrestaShop web shop

Mattias Berg

Master of Science in Technology Thesis

Supervisor: Annamari Soini

Advisors: Tim Wallin, Erik Nylund

Software Engineering Laboratory

Department of Information Technologies

Åbo Akademi University

2024

Abstract

The objective of this thesis is to create a solution for automatic deployment of a PrestaShop web shop to allow for a faster, more streamlined, and less error-prone way of working. Based on the wishes of the stakeholders, the solution should introduce as few new tools as possible. After some investigation of what tools are already installed when developing PrestaShop-based web shops, a solution based on the Python programming language was chosen as it was able to integrate smoothly with most frameworks for automatic deployment. The solution consists of three features, of which the first and main feature is to create a new web shop. This is the most programmatically complex feature and with the modular approach in the design of the solution it will allow for a high level of code reuse. The other two features are to delete a web shop and modify a web shop. Modifying a web shop was initially scoped to be able to change multiple parts of the web shop but was eventually reduced to only the adding of new PrestaShop modules. The solution is successfully able to automatically create and deploy a new web shop to a server, delete it, and modify it in limited ways. It unfortunately falls short when it comes to the continued way of working after a deployment, as it is not able to keep changes made through the web browser synchronized with what it is trying to deploy from. A key lesson learned is that DevOps practices will need to take the full lifecycle into account or risk falling short and creating new problems where there previously were none.

Keywords: automatic deployment, prestashop, web shop, rocketeer

Table of Contents

Table of Figures

Abbreviations

PHP	PHP: Hypertext pre-processor
PHP/FI	Personal Home Page Tools Forms Interpreter
SSH	Secure Shell
CLI	Command Line Interface
DevOps	Cooperation between development and IT operations
CI	Continuous integration
SMTP	Simple Mail Transfer Protocol

1 Introduction

Today's web shops run on several different technologies. It is not uncommon to find web shops written in programming languages such as C#, PHP, or even JavaScript. Looking at just PHP, we find multiple web shop frameworks, the biggest ones being WordPress using the WooCommerce module, Magento, and PrestaShop which is the one we will use in this thesis.

The goal of this thesis is to create a solution for deploying a PrestaShop web shop. The solution should be as automatic as possible and avoid requiring the user to connect manually to the server. The solution should also be suitable for the server environment used by the stakeholder.

Throughout this thesis, I will refer to different aspects which may easily be confused with each other. To make things clearer I will define them shortly here.

The stakeholder refers to the company or interested party who has an invested interest in the solution. These individuals are those who set the broader non-technical requirements for the solution and expect to see how feasible a solution such as this would be.

The client is the component of the solution that runs locally on the user's machine. It serves as the interface between the user and the broader set of features of the solution.

The user represents the individual actively using the solution. He or she interacts with the client to perform operations against the deployment of a web shop. The user is expected to be a developer with experience of the PrestaShop web shop and is expected to be able to take manual intervention, if needed.

1.1 Problem statement

The stakeholder's current deployment process of a PrestaShop web shop requires a long series of manual steps to be performed and no part of the process is automated. To give insight into what is required when a new web shop is to be deployed, here

are some of the major steps that need to be performed: connecting to the server to copy over files, setting up the database for storing stock and customer information, configuring the http server for serving the web shop, configuring the web shop to run in the production environment, and several other smaller steps. These are unnecessary and risk-filled steps for a developer to undertake manually.

Reducing the amount of work required to deploy a web shop would free up development resources in terms of time. The time saved in the deployment phase can be spent on other tasks. If the other task is billable, a direct financial benefit can be gained.

Fewer manual steps required in the deployment process will also reduce the likelihood of human error. When connected directly to a production environment a misconfiguration may result in several different problems. The web shop being deployed may see a delay in deployment, requiring more time or even resulting in a missed deadline. The misconfiguration may even be server-wide, resulting in downtime for other applications running on the same server.

1.2 Thesis structure

The introduction is primarily focused on the stakeholder, what the stakeholder wants and what the stakeholder expects to benefit from a solution to the problem. The introduction will also contain a thesis structure overview.

The requirements section describes the system development process, including development tools and environments. The quality criteria for the final solution will be presented and discussed. The requirements section will also describe the different requirements on the client and server, how error handling should be done, and other considerations such as the user interface and the security aspects of the solution.

The background section examines the different tools and frameworks used in the stakeholder's standard PrestaShop web shop setup. This section will also examine the different deployment solutions considered to solve the problem described in the introduction section.

Following the background section, the implementation section will go into details of the implementation of the solution, starting with the implementation of the requirements on the server and on the client, and then describe how the different features are implemented, and finally how the error handling and testing were done.

The evaluation section will examine if the solution works and how well it meets the quality criteria set by the stakeholder. This section is followed by potential future improvements to the solution such as continuous delivery, smarter configuration, and integration to other solutions solving different problems related to automated deployment.

2 Requirements

The requirements of the solution are set by both the stakeholder and me. This is to ensure that the resulting product will fulfill the functionality the stakeholder expects and that the project is suitable for a Master's thesis.

The solution should be modular to be able to be extended easily in the future for other applications that follow a similar deployment process as PrestaShop does. These would generally be other types of PHP-based web sites or web applications. Modules should be as atomic and general in their operations as practically possible to allow for greater reuse.

A set of workflows should also be created which describe the modules and in what order they should be executed to perform a set task. A workflow here is specialized for a single larger purpose, for example, to create a new shop or update an existing shop.

2.1 The server

In theory, the server can be any server that supports running an SSH-server and Git. In this case, however, it is only required to be able to run on Debian-based servers as this is what the stakeholder uses for all PrestaShop deployments, which allows us to limit the design to work on that specific type of system.

The server needs to meet or preferably exceed the minimum required specifications for running a PrestaShop web shop. While there are no fixed requirements for CPU and memory, a minimum of 256 MB of memory per script execution is recommended. [1]

In addition to being fast enough, the server needs to have PHP installed as this is a requirement to be able to run PrestaShop. The PHP version requirement for the PrestaShop version used needs to be at least version 5.2. [1] Newer versions of PrestaShop will require a newer version of PHP. The PrestaShop CLI installer will be used by the solution to configure the web shop and the CLI installer is executed using PHP.

The server also requires a web server to host the web shop. This is ideally Apache, since PrestaShop comes with configuration files out of the box. However, an Nginx-based web server will also work but its configuration will need to be manually set up. The ability to automatically configure the web server for hosting the new web shop is seen as an optional feature rather than an essential requirement.

Additionally, the server will need to be able to connect to a database. This database will store all the essential information that PrestaShop requires to function. This includes all products, prices, shipping options, and payment options to name a few. The database must be SQL-based, as PrestaShop only supports this type of database. It can be installed on the same server or a different one, if PrestaShop can connect to it. [1]

In addition to the requirements of PrestaShop itself, the server must run an SSH daemon service. A daemon service is a service that runs as a background process. This is required for the development computer to communicate and run commands on the remote server. The server must also have a Git client installed and access to a version control repository, for the solution to be able to request the latest version of the web shop being deployed.

For the rollback functionality to work, the server must also have enough disk space available to be able to store multiple versions of the same web shop. While this is not a strict requirement, not having it would compromise the safety of the deployment.

2.2 The client

From the stakeholder only basic requirements are given. The solution should be possible to run from the type of computers used by the developers. These computers are laptops running either Microsoft Windows or Apple macOS. The interface for a user to interact with the client should be limited to a command-line interface. A command-line interface is simply a text-based way for a user to interact with a computer. This is to allow work to be spent on the parts of the solution which bring value.

Some method for the client to communicate with the server will be required for the deployment to work. While there are multiple ways of establishing a connection to transfer files to a server, the stakeholder currently uses SSH for all server communication. To avoid having to set up a different method of communication SSH will be used as the method of communication. Both Microsoft Windows and Apple macOS support SSH, although only Apple macOS comes with it pre-installed. On Microsoft Windows, it will need to be installed before the solution can be used.

As the solution will likely require some programming, the language used should be one suitable for command-line interfaces and the chosen automatic deployment framework. The stakeholder has recommended Python, as it is a language that they use for similar projects to this solution. It is also suitable for command-line interfaces.

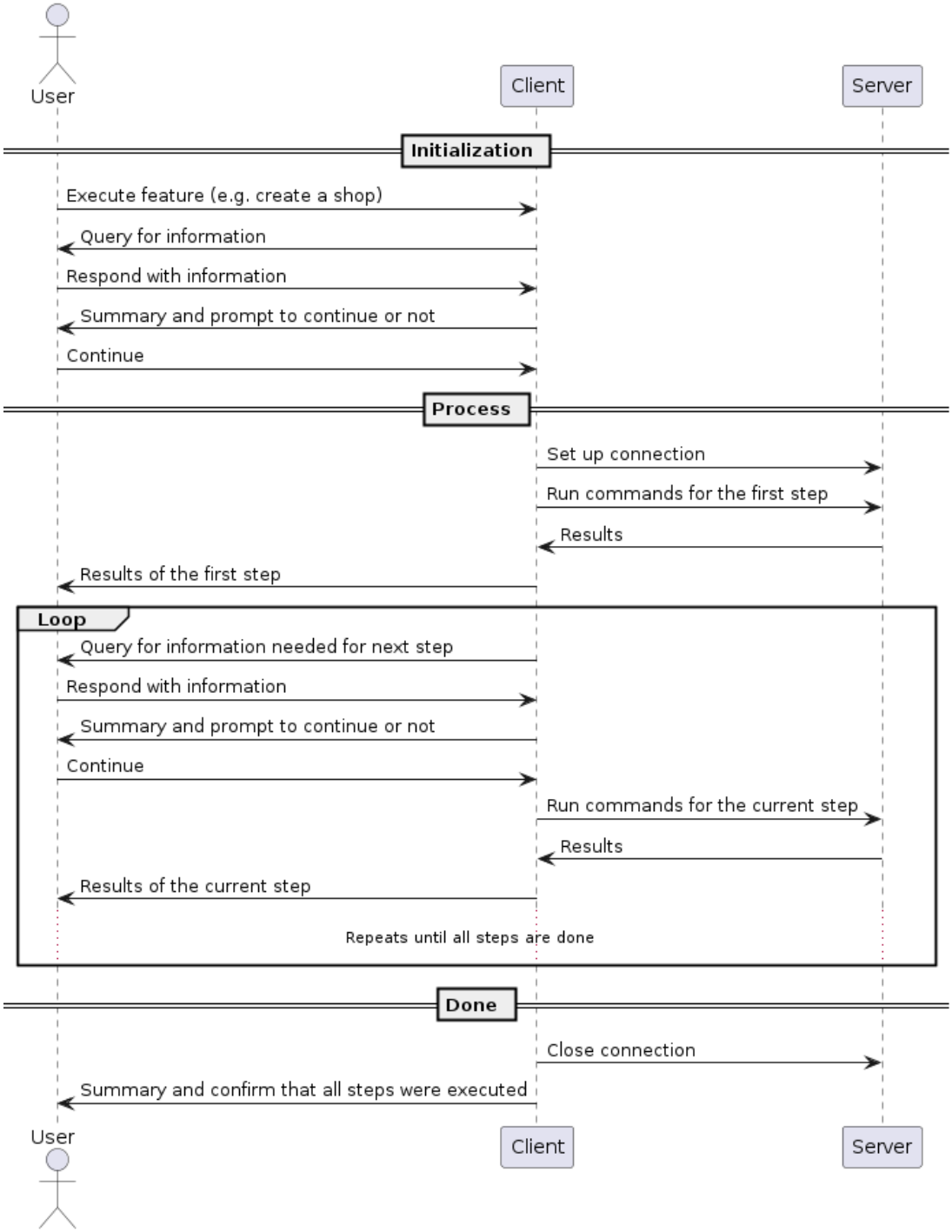

Figure 1 Client workflow

The primary workflow will consist of the user executing the program with the wanted feature, for example, to create a shop. The client will then query the user for any required information and give the user a final summary and ask if it should continue. When the user accepts the summary and lets the client continue, it will open an

encrypted connection to the server where a series of commands will set up the web shop according to the specifications given earlier. During this process, the client will query the user if more information is needed or before performing any potentially destructive operation for confirmation.

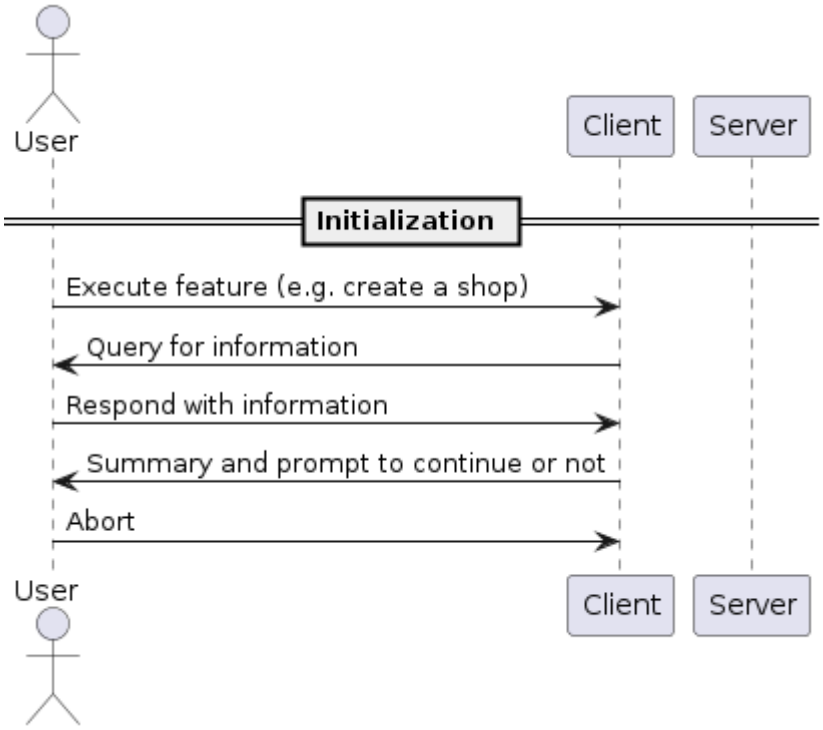

Figure 2 User ends the feature execution early

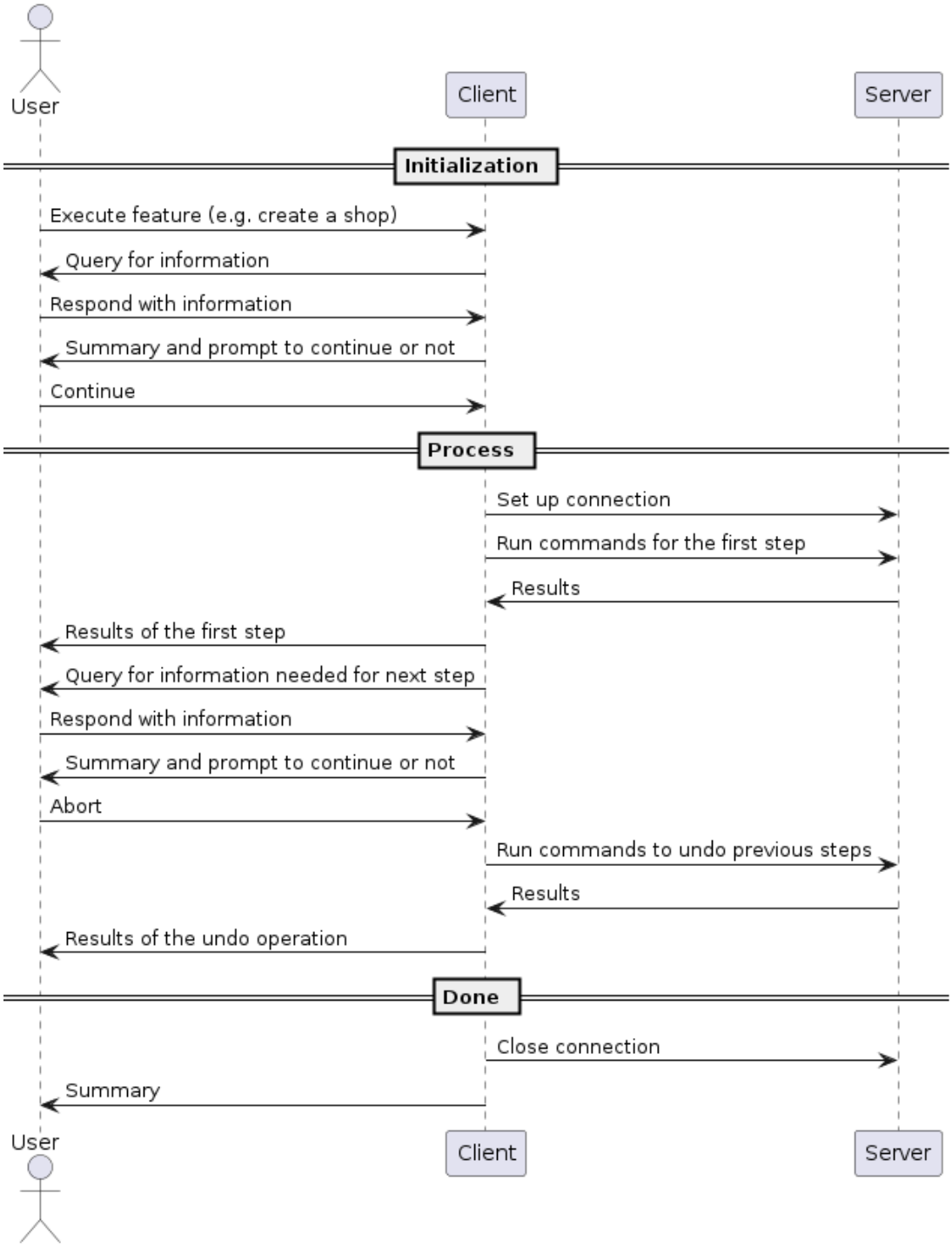

Figure 3 User ends the feature execution in the middle

If the process is aborted by the user before any operation is performed by the client, the client should simply close without contacting the server. If the process is aborted by the user in the middle of a feature execution, the client should attempt to undo all previous steps performed during the feature execution, then close the connection to the server and give a summary to the user before closing.

2.3 Quality Criteria

The solution should be easy to use for novice or senior developers given that they are familiar with a command-line user interface. This is because a graphical user interface is explicitly not in scope due to project size constraints. The solution should be designed so that a command-line user interface will be sufficient for the intended users. Even though the user interface is not graphical, it should always be clear what the next step the solution will perform is. Before any action is taken by the solution, the user should know what will happen and ideally be able to stop or revert execution at any step.

Each step in the process of running the solution should be clear as to what it does. If the step involves modifying data in a non-reversible manner, the step should clearly state it. Before executing a non-reversible step, a yes or no question should be asked before proceeding. If the user answers no to the question on whether or not to execute the non-reversible step, the solution should revert all changes it has made so far, or at a minimum notify the user about what has been left for manual cleanup. It should notify the user regarding what needs manual cleanup, if the execution has stopped after a non-reversible step has been executed.

The solution should take into account that both passwords and SSH keys will be used to configure the web shop, fetch source code from the version control, and communicate securely with the server. The solution should at no point show passwords while executing the solution. The solution should also ensure that there are no passwords or SSH keys stored in a manner where another user would be able to read them once execution is done.

Human error reduction is a key point in the decision to create this solution. One of the goals of the solution is to remove several manual and error-prone steps which are often poorly documented. The process of using the solution should require as few manual steps as practically possible. This should increase efficiency as well as minimize the risk of human error occuring.

For security and practical reasons, the solution should ideally not require the user to install any extra programs or tools on the development computer. For the same reasons, the server should not require any extra programs or tools. Some minimal number of additional tools or programs is still expected. This requirement criterion should be regarded as the fewer additional programs or tools needed the better.

The result of running the solution should be one of the following based on which feature has been executed:

1. A new web shop running on the server based on the configuration given at the start of execution.
2. A new version of an existing web shop running on the server with a new or updated module.
3. An updated version of PrestaShop itself.
4. The removal of a chosen existing web shop that was previously created using the solution.

2.4 Features

Several features were identified and prioritized for the solution. Being able to create a new shop is ranked highest, as this is what brings the most value and enables the other features. Being able to modify a shop is ranked lowest due to its complexity. Deleting a shop is then left in between priority-wise, as it should be a very simple feature to implement after the creation of a shop feature, even though its value is considered less than that of the other two.

Being able to create a shop is the main use case for the solution, which means this feature is seen as a must have. When creating a shop, the solution should set up all the necessary infrastructure such as the database, folder structure, logging, and any other part that is required by PrestaShop. Once the infrastructure is set up, the solution should set up the shop based on a template source code repository.

Deleting a shop is second on the priority list and while it will not give any large amount of value, it will ease the cleanup when a shop is to be decomissioned. Cleaning up the

server after a web shop has been closed down is a slow and manual task where it is easy to forget to remove parts of the web shop. These parts, which can be a database, log files, or even parts of the web shop itself, are usually harmless but can be the cause of confusion when another web shop is to be deployed. This confusion will generally lead to delays that should rather be avoided.

Modifying a shop is quite useful, as it will allow changes to be made to the shop and have those changes recorded or tracked with the solution. There are multiple parts that could be modified in a shop, but the most useful would be to handle modules. Being able to add, remove, and upgrade a module could open up for a standard set of modules to be maintained across all web shops. A less important part to be able to modify through this feature would be the theme of the web shop. A theme in PrestaShop refers to the overall design, such as layout, typography, and color scheme. It is not expected that this feature will be able to modify shops not created using this solution.

3 Technologies and concepts

The technologies used in the solution are all close to what the developer would already have installed on their computer or what the developer would be familiar with for PrestaShop development. The only exception are the technologies which are directly related to the new concept being introduced to the development workflow. The concept of automatic deployment is not something that has been utilized in PrestaShop development by the stakeholder.

3.1 PHP

PHP: Hypertext preprocessor as it is known today is seen as a successor of PHP/FI that was written by Rasmus Lerdorf in 1994. PHP was then short for Personal Home Page Tools and the FI part was short for Forms Interpreter. PHP was originally created as a tool to track visits to Lerdorf's online resume. [2]

PHP 3 development began in 1997 and was completed in 1998 by a group of three developers, the original creator Rasmus Lerdorf being one of them. This release was also where PHP received its new name that it still uses today, PHP: Hypertext preprocessor. [2]

The next big release of PHP was PHP 5. Released in 2004, it introduced the Zend engine 2 and an object model making PHP a true object-oriented programming language. [3] Even to this day, PHP 5 is still in use by many websites around the world and is the language version used by the client's PrestaShop deployments.

3.2 Web shops

A web shop refers to a website or online platform that offers functionalities to facilitate the purchase of goods or services over the Internet. A web shop is also known as an e-commerce website, it is a subset of the broader e-commerce term which encompasses more than just the selling of goods and services. E-commerce refers to all types of commercial transactions done over the internet, such as

electronic payment and digital marketing. A web shop is the component of e-commerce that allows businesses to engage in online sale of goods and services. [4]

A core aspect of a successful web shop is being able to efficiently handle product listings and stock management. This involves keeping track of inventory, updating products, and ensuring that the information is accurate as far as product availability is concerned. Web shops often use highlighted products in a prominent place in the home page of the website to attract new customers. Building customer profiles and tracking purchase history is used widely by web shops to enhance the experience and keep customers returning with targeted sales coupons or advertisement campaigns.

PrestaShop is an open-source e-commerce platform built using the PHP programming language. It is designed in a modular way to allow for a high degree of customizability. This design principle makes it easy for the end user to make changes to the appearance or functionality of the store. [5] These modules allow PrestaShop to fulfill the key aspects of a successful web shop described earlier.

3.3 Automatic deployment

Automatic deployment, often referred to as continuous deployment or continuous delivery, is a software development practice that involves automating the process of deploying new code or configuration to a server. While these terms are often used interchangeably, they are not the same. Continuous deployment or continuous delivery is about automatically deploying code or configuration changes to at least a test environment without any human interaction. Automatic deployment, in turn, is about automating the deployment process while still requiring a human to start it. Automatic deployment should not be confused with continuous integration, which is the process of continuously integrating code changes to a centralized branch to be built and tested but does not include deployment of those changes.

Automatic deployment relies on having one's code in an already deployable state. This means it does not take into account any type of testing that might be required to ensure the code works correctly and it requires a final build of the software. This

final build would be, for example, an executable file, minified and packaged javascript, or in the case of interpreted languages such as PHP the code might be ready to deploy as it is. In the solution described in this thesis, the PHP code only requires configuration to work correctly and that configuration is already stored on the server it is being deployed to. This means that the code is already in a deployable state.

Automatic deployment and its further developed continuous integration and continuous deployment are crucial practices for DevOps and agile development teams. They speed up the deployment process while also reducing the risk of human error and increase the consistency and reliability of deployments.

4 Design

The scope of the solution is limited to creating, modifying, and deleting a web shop and only through a command line interface. This allows the solution design to focus on the core functionality to fulfill the requirements and tailor the architecture and framework decisions to the limited scope. The biggest decision to make will be which framework to base the solution on, as that will impact the workflow of the solution. Choosing the wrong framework will potentially make the solution unable to fulfill the requirements or even prevent wanted features from being implemented. The system development process is the only part that does not need any specific decisions to be made, as it will have to follow the stakeholder's standard practices.

4.1 System development process

Standard development practices of the stakeholder should be followed during the system development process. This includes but is not limited to storing all code in a version control system, generally following the code style for the language used and including a readme file in the version control system.

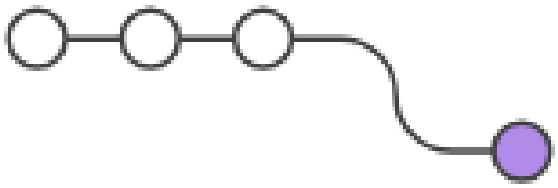

Figure 4 Creating a new feature branch.

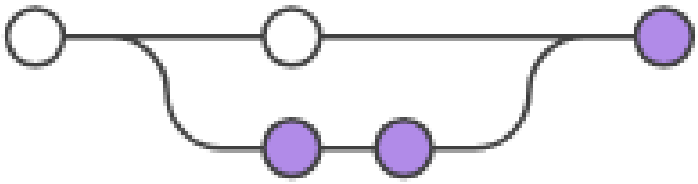

Figure 5 Changes made on the feature branch are merged back to the main branch.

The version control system should be of the same type that the company otherwise uses, even though the version control service supports other types. The version

control type to use is called Git. Git supports a functionality called branching which allows a developer or team of developers to work on an isolated version of the source code without affecting other developers. A basic workflow to use is called Feature Branch Workflow [6] and is the recommended workflow by the stakeholder. It starts by the developer creating a new branch, known as a feature branch, based on the main branch, illustrated in Figure 5. The developer can then make changes and commit those changes to the version control system without affecting the main branch. Once the feature branch is ready to be merged back into the main branch the developer opens a pull request which allows other developers to review the code before it is merged into the main branch [7]. When the pull request is approved and merged the resulting version control history will look similar to what is illustrated in Figure 6.

The code style of the language chosen should be followed unless otherwise agreed on unless it makes sense to go against the style guide under specific circumstances. While this is not a strict rule the idea of it should be followed, which is to maintain human readability of the code.

The code repository should contain a file called "Readme.md". This file should contain all information a developer requires for getting started with the project. This information can be in the form of a brief introduction, a step-by-step guide on what needs to be done to get started with either developing the tool further or using the tool, and preferably a list of known issues. In this case, information needs to include using the tool as well as continuing the development of the solution.

4.2 Architecture

The solution uses a modular approach to allow for new functionality to be easily added without affecting current functionality. Modules are loaded on demand based on the selected use case.

Python supports dynamic imports of new source code files during runtime [8]. This allows the solution to only import the modules which are needed for the current use

case. While this was a positive side effect for resource usage, the original intent was to not have to worry about which modules should be executed and which should not be for a given use case.

If, for example, the *create shop* use case is selected the solution will read the list and order of modules to be loaded from a configuration file linked to that use case. The modules are then executed in the order they are in the configuration file.

```
1   class HelloWorld:
2
3       def __init__(self):
4           self.name = "HelloWorld"
5
6       def getName(self):
7           return self.name
8
9       def execute(self, config):
10          print "Hello world!"
11          return config
12
13  def initModule():
14      return HelloWorld()
15
```

Figure 6 Hello world written as a module.

For this to work, each module must follow a specified interface which takes inspiration from the command pattern [9]. Each module must implement at least one class and an initialization method which returns an instance of the class. The class must implement two methods, one which returns the module name in human readable form and one to start the execution of the module. The figure above, see Figure 7, shows an example of what a “Hello world” type application would look like when written as a module to the client.

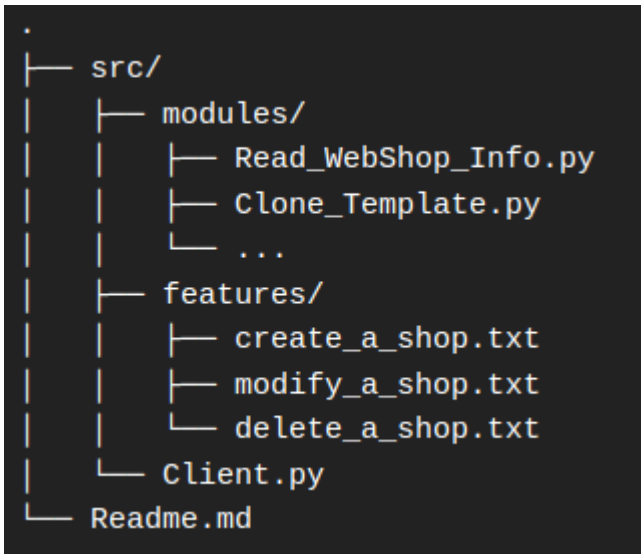

Figure 7 Source code file tree example.

In the source code file tree example figure, see Figure 8, the *Client.py* would be the main entry point to execute the client. When executed with a flag to use the *create shop* feature it would first read the *create_a_shop.txt* file to fetch a list of modules to execute and to know in which order they should be executed. All modules are already set up to follow a specific interface which allows *Client.py* to know how to run them.

4.3 Testing and error handling

In case of any error during the process, the solution should report the error to the user and ideally roll back any changes it has made, if possible. If it notices any changes that might be destructive, the user should have to decide whether to continue or not.

Any type of error logging and traceability will not be included as part of this first version of the solution. If the solution is changed to run in a continuous integration environment, then this would need to be added since no person would be actively following along the process. A continuous integration, or CI for short, environment is most often a build service where the software is built, tested, and sometimes integrated with other components in an automated way to provide quick feedback to the developers. [10]

4.4 Frameworks for automatic deployment

Several different solutions for automatic deployment were evaluated for this solution. The three main contenders are described in more detail below. With a goal of minimum extra dependencies on the development computer Rocketeer was eventually chosen as the solution to use. This is due to it being able to handle Git code repositories, its runtime being PHP, and because there is built-in support for a shared folder between releases. [11]

4.4.1 Rocketeer

Rocketeer is built using PHP and runs using PHP, which is an ideal scenario since that is what the development computer is required to have as well when developing PrestaShop applications.

In its core Rocketeer is a basic SSH (secure shell) task runner, which means after configuring the setup it will be able to connect to a server using SSH and perform different tasks on that server. However, it is the built-in functions for deployment, which make it a contender and ultimately the chosen automatic deployment tool for this project.

Figure 8 Rocketeer folder structure.

When using the built-in functions for deployment, Rocketeer is given a root folder to work in on the server. Any operations it will perform will be performed within this folder. Inside this folder three different folders will be created, see Figure 9. The first folder is called Current. This is the folder which the web server should serve. The second folder is called Releases. This folder is used to store a predefined number of releases of the application. The Current folder will always point to one of these release folders using a symbolic link. This allows Rocketeer to perform rollbacks to an earlier release, if the latest release is experiencing problems. The final folder is called Shared and it contains shared files or folders between releases, such as images, log files, or configurations. [12]

4.4.2 Capistrano

Capistrano is a task-based remote server automation and deployment tool written in Ruby. Even though it is written in Ruby, it can deploy any language due to its extendibility. This also means that the development computer also requires Ruby to be installed in addition to the normal development tools.

Like Rocketeer, Capistrano is based on using SSH to manage one's deployments on the server. It is also based on the same idea of having a Current folder, which is a symbolic link to a folder inside another folder called Releases. It also utilizes a folder called Shared for any files that should be shared between releases.

What sets Capistrano more apart from Rocketeer is that it also has the code repository stored in a folder called Repo. If the code repository is based on Git, then the raw Git repository will be stored there. [13]

4.4.3 Ansible

Ansible is a much larger tool than the others mentioned above and application deployment is just a part of what it offers. Ansible is task based and uses SSH to perform those tasks on the server just like the other tools.

What sets Ansible apart from these other tools is that instead of triggering predefined scripts it allows you to describe the state one wants, and it executes different commands to get one to that state based on the current state and the environment. These on-demand created scripts are sent to the server as a module, executed and then removed from the server. [14]

4.5 Further considerations

While the solution will be a console application only, it should be self-explanatory to use. Using it should be like following a step-by-step guide. Each step should explain what it does and what it requires as input.

Security will also be an important factor to consider. The solution will be connecting directly to a server, interfacing with the code repository, and managing application credentials.

As far as connecting directly to a server is concerned the solution will need to manage SSH keys and/or passwords. These should not be stored in plain text or even be shown in the console window while using the solution. Passwords should ideally be hidden from view and only stored in memory for the duration they are needed and then forgotten.

5 Implementation

Rocketeer was the framework chosen to base this solution on. The solution itself was built using the Python programming language. It wraps the Rocketeer commands as well as makes any code modifications required for the deployment. Implementing the solution in a satisfactory way to meet most of the requirements turned out to be the simple part. The challenge was how PrestaShop handles its configuration files.

5.1 Handling stateful files

PrestaShop stores modules and configuration of modules as files on disk. During the design and requirements gathering, this was not seen as a problem as the designed solution supports shared files and folders between releases. What was noted quite early in the implementation phase is that PrestaShop not only modifies files on disk as the user makes configuration changes to the installed modules. It can also remove and create new files as a module is installed, upgraded, or removed through the back office.

The PrestaShop back office is the management part of the web shop. This is where basic web shop functionality, such as products, stock, shipping, and prices, is handled. It is also where PrestaShop handles its modules. A module can be for example a shipping or payment solution. The PrestaShop back office allows an administrator of the web shop to install new modules, configure or upgrade existing ones, and remove modules no longer needed.

The initial thought was to simply store the entire modules folder as a shared folder. This would unfortunately not work, as it would break the rollback functionality. Modules are able to store data in the database, which is not something that can be rolled back safely due to the risk of losing order data.

In the end, a web shop deployment will have to choose between one of three options. The first option is to abandon rollback functionality and store all modules as shared folders between releases. The second option is to only allow modules to be managed through the *modify a shop* feature. The last and least practical option is to manually and continuously maintain the list of files to be shared between releases. This last option would allow rollback functionality to work, as well as allow modules to be handled through the back office. This is a choice that can be made per web shop deployed using this solution, most of them will likely choose the first option as it maintains the full functionality of the shop.

5.2 Features

The solution consists of three features, the first and main feature is to be able to create a new web shop. This feature will need to query the user for all the necessary information needed for installing a new web shop and running the installer while keeping the code in the repository up to date with what will be running on the server. The second feature is to delete a web shop that has been deployed by this solution. As a requirement here is that the web shop was deployed using this solution, it will know the location of all files and other types of configuration and infrastructure by asking the user a few questions, such as the name of the web shop. The third and final feature is to modify a web shop. This feature is able to reuse several parts of the first feature but was eventually limited in scope due to the complexities of deleting and modifying modules.

5.2.1 Create a shop

Running the *create a shop* functionality loads a configuration and modules required to deploy a new web shop. The major steps are as follows:

1. Gather information from the user
2. Set up an empty shop from a template
3. Copy over any chosen modules or themes
4. Set up a project repository
5. Deploy an initial version with Rocketeer
6. Set up a database
7. Run the CLI installer
8. Commit finalized shop changes to repository
9. Redeploy the shop with Rocketeer

The first step involves querying the user for information about the shop, such as the name of the web shop and which domain name it will run on, the server, such as which folder the shop will be deployed to, and also credentials to be able to access the server and version control repositories.

Steps two through five are all about setting up the base of the shop in a version-controlled manner. Rocketeer deployments work by fetching a new version from a version control repository and to be able to proceed with the next steps the web shop will need to be located on the server.

Steps six through nine take the base shop and configure it to the specifications given earlier in the process.

5.2.2 Delete a shop

The *delete a shop* functionality is by far the simplest but also the most dangerous, if given the wrong information. It will completely delete all source code, configurations and databases related to the shop that the other functionalities have created. It does this by first querying the user for all required information about the shop. It then checks the data for any obvious errors, prints the information out and asks if the user wants to continue. Only after being prompted to continue after displaying the information will it proceed with the deletion.

5.2.3 Modify a shop

This functionality was initially planned to be able to add or remove both modules and themes. However, due to difficulties mentioned when discussing the handling of stateful files, the functionality was limited to only adding new modules. The main issue here was how PrestaShop manages configuration files for modules that need to be shared between releases.

Adding a new module ended up following much the same steps as when creating a new shop, so many parts could be reused. In the end, it mainly involved modifying the project repository to include the new module and then triggering a new deployment through Rocketeer. The enabling and configuring of the module must be performed manually in the PrestaShop back office.

5.3 Testing and error handling

Due to the limitations in the initial architecture of the solution, no automatic testing was created. All testing has been done manually and this negatively impacted on the result, since not all cases could be practically tested. The solution has been manually tested by executing each functionality. The “happy path”, the path through the process where no issues are expected to be found, has been the testing case through most of the development process. Towards the end of the development process all functionality has been tested by attempting to cancel execution at each step to ensure the solution is able to revert changes and any data input fields have been tested with both valid and invalid inputs. Invalid inputs include writing only letters where numbers are expected, leaving an input field empty, or inserting symbols or non-standard characters.

If the solution encounters an error an exception will be thrown and the process will in most cases be reverted back to how things were before the execution started. There are a few cases where the process has performed an irreversible step, before each of these the user will be notified. The expectation is that the user of the solution will know how to handle all the steps manually so that an execution can be reverted to prevent any lasting effects on the server. The solution has multiple steps where it checks that all required information has been given and is in the correct format, as well as checking that the connection to the server is working so that in case of a detected error it alerts the user and stops any execution it is doing to prevent non-reversible damage.

6 Evaluation

There is very little information in terms of numbers to gather as far as the use and success of the solution is concerned. The evaluation will be focused on user feedback and to which degree the solution fulfills the different use cases.

6.1 Web shop

The first check is a visual inspection that the web shop is deployed correctly and is working. This visual inspection will ensure that no major hidden errors have occurred.

The web shop deploys correctly with the selected theme and modules, otherwise PrestaShop default values are used. There are no visible errors and normal functionality of the web shop is working. However, the solution is not able to configure the modules added nor is it able to configure default functionality, such as payment or shipping options. Configuring default functionality was not part of the initial requirements, however, it would be beneficial to have.

The steps of setting up a new web shop are the same as before from the perspective of the server, from the perspective of the developer the process is much different. The developer now interacts with a Python program which will query for information about the web shop being created, modified, or deleted and will then in the background perform the previously manually done tasks for setting up a new web shop.

6.2 Quality criteria

Due to the area the tool operates in as well as the requirements set by the customer, the evaluation of the quality criteria is limited. This means that the evaluation of quality criteria will be mostly based on human feedback and descriptions of what does and does not work according to the initial requirements.

6.2.1 Easy to use

The solution guides the user through each step in the process and explains what will happen. It would be beneficial if it were better able to revert changes when the user cancels the process, but it does warn before a change happens that it will not be able to revert.

It was noted from an end user of the solution that they only needed minimal knowledge about the server itself. Previously, the end user had to know a great deal about the configuration of the server. Directories had to be created in the correct places, web servers had to be configured, and databases had to be set up with correct credentials and settings.

Generally, while the feedback concerning the solution has not been comprehensive, it has been seen as a significant improvement compared to before. The initial setup of the solution on the user's computer did cause some confusion due to the differences in Python versions. The user's computer did not have the correct Python version installed by default and had to be updated. This issue stems from the long overlap of Python version 2 and Python version 3. Python version 3 was released in 2008 [15] while Python version 2 was maintained all the way until 2020 [16] and in this case the developer's computer had Python version 2 installed.

6.2.2 Automate as much as possible

Starting by looking at what was not automated, the largest manual work remaining is that entering information by hand is still needed. Moving away from manual information entry would be a large project of its own, requiring a way to parse offers or contracts to be able to extract the needed information.

After the web shop has been created, there are still some configurations that the solution does not yet support automatic configuration of. E-mail settings such as Simple Mail Transfer Protocol, often just known as SMTP, server and authentication, default payment options, shipping, and the web shop slogan have to be manually entered by the user in the web shop back office. Any added module will also need to be manually configured. A potential workaround for this could be to supply

preconfigured modules but since each web shop usually requires its own configuration for the modules, this would not be a good way of managing it.

After gathering information from the user, the solution will automatically set up a new empty web shop with any chosen modules or themes in its own code repository. This allows a developer to continue to make any manual changes, if needed. The solution will then deploy the initial version to a server, set up any necessary infrastructure, such as a database, and run the CLI installer fully automated. The code repository for the web shop will then automatically be updated by the solution to include the installation changes. That these parts were automated was seen as an enormous time saver.

6.2.3 Safe

Each step the solution performs is clearly described to the user and to avoid mistakes it requires a 'Yes' response at certain times in the process. This is generally before it does something it cannot undo itself but also, for example, after it has shown the user a summary of entries. This gives the user a chance to quit the process at multiple steps to minimize the impact of errors, if one should occur.

All communication with the server is done securely over SSH. This ensures that no third party can listen in on the communication and steal credentials to gain unwanted access. SSH is set up on the server to use public and private keys to authenticate. These keys are generally stored securely and encrypted with a password. Unfortunately, due to a limitation with Rocketeer the password for the SSH key is stored in plain text on the user's computer. The solution warns about this, so the user can delete the password after use. The solution does not show any passwords or keys on the screen while it is being used. This means that if the user removes the stored password after using it, it will no longer be visible or potentially visible to anyone.

6.2.4 No extra tools

To run the solution, Python and Pip as well as the Rocketeer binary are required. The Rocketeer binary is supplied with the solution so the only extra tools needed to be

installed on the client side, other than what should already be installed by default, are Python and Pip. Pip is the package installer for Python and is used to fetch Python packages from the Python Package Index [17].

The server requires an SSH daemon to be running to enable access to run commands and copy files from the client. This is because Rocketeer communicates with the server using SSH. Luckily, this is already installed and enabled on the server the solution would be targeting, so no additional programs are required to be installed on the server.

6.3 Features

Not all features were successfully implemented. Most notably, the modification of an existing web shop feature ended up being almost unusable due to its limitations of only functioning correctly, if run directly after a web shop creation. The most successful feature was creating a new web shop. This feature ended up working exactly as required and it was also the feature which was considered the most important. The feature related to deleting a web shop is working as intended, but it was noted by the stakeholder that the ease of running this functionality with the potentially high negative impact if the wrong web shop was deleted was a high risk. Previously, deleting a web shop required much more manual work and was not something that could easily be done accidentally.

6.3.1 Create a shop

This is the main use case of the solution. All information needed for setting up a new web shop is asked from the user and stored in memory for later use. The workflow then creates a new project repository in the source code management software. This repository is where the code for the web shop is stored and where modifications for it can be made. When there is a place to store the code the solution adds the template web shop, any selected modules, and theme to the repository. Using the code repository that has just been created the solution deploys the code to the server

by using the Rocketeer framework. With the code deployed to the server it sets up a database and a database user so that the actual installation of PrestaShop can begin.

The solution uses the PrestaShop CLI installer to set up the web shop based on the information provided at the beginning of running this workflow. This CLI installer normally functions by querying the user for information as it runs. However, by providing all necessary information as parameters to the installer it will run in a non-interactive mode. The solution takes this into account and makes sure not to continue execution during the information input phase, if the solution does not receive the necessary information. When all this setup is completed, the solution updates the code repository with the installed code so that both sides, what is on the server and what is in the code repository, are synchronized. A developer can now make changes to the code in the code repository and deploy those changes to the server, as needed.

This functionality works as initially envisioned and fulfills the requirements. Unfortunately, the expected continued way of working with editing the code in the repository and deploying those changes to the server is not feasible. PrestaShop makes changes to the files of the web shop as it is configured through the web browser interface and as such makes the two codes, what is on the server and what is in the code repository, will be unsynchronized. If a new deployment were to be made, it would likely cause changes to be reverted or, in the worst case, it would cause the web shop to malfunction.

6.3.2 Delete a shop

Running this workflow results in a complete deletion of the database, database user, source code, and project code repository. This does unfortunately not include any type of restore feature, if done by mistake, so care should be taken when running this functionality.

The files on one's local computer are left untouched, which means that a new web shop with the same configurations can be created again. This will, however, be a new and empty web shop since the database and all configurations stored in files on the

server will be gone. Aside from that, this can be a useful way to quite quickly deploy a new web shop, if some mistake were made with the initial web shop.

6.3.3 Modify a shop

Due to the way PrestaShop handles configurations and module source code, when making changes to them through the back office, this workflow only performs as expected as long as the modules have not been updated or no new modules have been added through the back office.

In practice, this feature is only something that can be used immediately after creating a shop. If it is used after the shop has been used or even configured, it may start overriding configurations. This reduces the usefulness of this functionality in a major way and makes it more of a data integrity issue if used. This functionality ended up being disabled in the final version of the solution.

6.4 Resource usage

The only resource of note when using this solution is the disk usage, and by that is meant that the file system is not edited outside of the working directory and that no extra files or folders are left on the system.

6.4.1 The server

Rocketeer, which is used to deploy the web shop, only works in one specific folder. This means that if the given folder is empty, creating a new shop or modifying an existing shop does not impact the rest of the server.

The solution itself does not in its current state perform any operations outside the working directory, but this can change due to its modular design and easy extendibility. To keep the solution as open as possible, there are no blocks in place to keep a developer from creating a module it can use which would make changes outside the working directory.

6.4.2 The client

The solution stores the configuration files of the deployed shop. This enables easy redeployment of new changes. However, part of this configuration has passwords stored in plain text. This is a limitation of Rocketeer, but since it is on the client only and the solution warns about it, it is seen as acceptable.

6.5 Performance

Deploying or deleting a shop requires information input from the user. This causes any performance numbers to become unusable in terms of measurement. Deploying a shop previously could take anything from a few hours to a few days depending on the complexity of the shop. With the solution, deployment of a base shop could be performed in minutes given that all information is known from the start. In a best-case scenario for both manually deploying and using the solution, the speed of deployment has moved from hours to minutes.

7 Future improvements

The solution currently only contains a minimum set of features that generate the most value. If its development continues, several quality-of-life features could be added.

The solution at its current state automates most of the steps required to deploy a new web shop. However, there are a few steps that still require manual work to be finalized. The data entering at the start of each workflow could be reduced and PrestaShop module configuration could be automated.

Only one environment is currently supported, which limits the use of the solution in bigger projects that require larger changes during the lifetime of the shop. Adding support for deploying to different environments, such as test or quality assurance, would greatly help in the reliability of deployments.

Changing the solution to run as a service would also allow for tighter integration to source control systems which, in turn, could enable a form of continuous delivery. This would even further reduce the need for human interaction in the deployment process. The change would, however, require much better logging and handling of errors as there would not be a human in the loop.

Considerations regarding changing the underlying architecture to allow for robust unit testing and integration testing would increase the reliability of the solution. This would help especially when or if further development begins to prevent regression issues.

Additional lower value functionality would be support for SSH keys to reduce the need for passwords. A graphical user interface to ease the use of the solution when used by developers who are not familiar with a command prompt would also be beneficial.

8 Conclusion

The result of this thesis is a solution that can in a largely automated way manage a PrestaShop deployment on a web server. The solution can deploy a new web shop including configuring the web server and setting up a new database in a mostly automated way. It is also able to delete a shop that has earlier been deployed using the solution as well as modify the shop in limited ways.

Creating a good DevOps [18] workflow is more difficult and time consuming than most people realize. It requires careful design and continuous development to be performant and easy to use in a changing business environment. If too little focus is put on DevOps practices, much of it will end up becoming an afterthought and manual work.

While Rocketeer has several good parts to it, it has also been difficult to work with when it comes to a PrestaShop deployment. PrestaShop handles its configurations by modifying files in multiple locations. These configurations would normally be handled by marking them as shared between releases. They can, however, be changed, moved, or created at any time from the back office. This makes multiple releases of the shop nearly impossible in the Rocketeer workflow.

The solution in its current state fulfills the requirements, but during development some of the core aspects of it were not ideal. The solution is designed to be run on one computer, and it stores information about a deployment only on that computer. If the solution instead were running as a service, multiple users and computers could access it at any one time.

Swedish summary – Svensk sammanfattning

Automatisk distribuering av en PrestaShop-webbshop

Introduktion

Målet med denna avhandling är att utveckla ett verktyg och en metod för att distribuera en PrestaShop-webbshop. Metoden ska distribuera en webbshop på ett så automatiserat sätt som möjligt och vara passande för den servermiljö som kunden använder.

En webbshop är en webbsajt där försäljning av varor och tjänster sker på internet. Detta är känt som e-handel. PrestaShop är en e-handelsplattform som är byggd med programmeringsspråket PHP. Dess källkod är öppen och designad att vara modulär för att möjliggöra snabb och enkel modifikation utan behov att redigera existerande kod. Installation och uppdatering av moduler görs med hjälp av PrestaShops inbyggda administrationssida.

Det nuvarande systemet för att distribuera en PrestaShop-webbshop kräver mycket manuellt och tidskrävande arbete. Denna tid kunde istället spenderas på andra debiterbara uppgifter. Det manuella arbetet måste också göras direkt i en produktionsmiljö som kör flera webbsajter och webbshoppar. En felkonfigurering kan lätt orsaka problem för hela miljön, vilket leder till uppehåll i driften.

Krav

Rocketeer är ramverket som valts att basera detta verktyg på. Själva verktyget kommer att vara skrivet i programmeringsspråket Python och fungerar som ett mer användarvänligt skal till Rocketeer. Verktyget ska vara modulärt i sin design för att senare kunna utvecklas vidare för att stödja mer än bara PrestaShop-utgivningar. Modulerna i verktyget kommer att köras i den ordning som ett arbetsflöde säger. Dessa arbetsflöden ska inkludera alla nödvändiga steg för att till exempel skapa en ny webbshop.

Verktyget ska vara enkelt och säkert att använda även för oerfarna utvecklare. Det ska inte finnas osäkerhet i vad nästa steg kommer att göra. En av nyckelpunkterna som verktyget försöker lösa är att reducera antalet manuella steg som behöver göras eftersom det är vid dessa som det sker misstag. Om verktyget behöver utföra ett steg som det inte går att gå tillbaka från så ska verktyget först fråga användaren om det ska fortsätta eller inte. Lösenord som fylls i ska inte synas i användargränssnittet eller i någon fil under verktygets körning eller efter körning.

Användning av verktyget ska inte kräva att en utvecklare installerar extra program på sin dator medan så mycket av processen som möjligt ska automatiseras. Det slutliga resultatet av en körning av verktyget ska vara en nyutgåva av mjukvaran.

Verktyget måste gå att köras på Microsoft Windows och Apple Mac OS X-baserade datorer. PHP, Python, en Python-pakethanterare och en SSH-klient behövs på användarsidan för att köra verktyget. Alla är antingen förinstallerade eller går lätt att installera så detta har blivit godkänt av kunden.

Bakgrund

PHP: Hypertext preprocessor, förkortat PHP, som det är känt som i dag är en uppföljare till PHP/FI som utvecklades 1994 av Rasmus Lerdorf. Det var först i version tre, som färdigställdes 1998, som PHP fick det fulla namn det har i dag. PHP 5 är nästa stora version av språket och färdigställdes 2004. PHP 5 introducerade en objektmodell som möjliggjorde att språket kunde användas med en objektorienterad stil. Detta är också den version som kundens PrestaShop-distributioner använder.

Automatisk distribuering är processen att få mjukvara från versionshantering distribuerad till exempel till en server där en slutanvändare har tillgång till den. Ramverket som valts för att automatisera detta heter Rocketeer. Detta ramverk valdes för att det är byggt att köras på PHP, kan hantera GIT-baserad versionshantering och stöder en delad mapp mellan utgåvor av koden.

Rocketeer är byggt med PHP och kör på PHP. Detta ger en perfekt passform eftersom utveckling av PrestaShop-webbshoppar görs med PHP. Grundidén med Rocketeer är att paketera in SSH-kommandon (secure shell) i ett lättare använt verktyg. Ramverket

innehåller flera inbyggda funktioner som underlättar många manuella steg i distribuering.

Hjälpfunktionerna för distribuering i Rocketeer arbetar i en given mapp på servern. Inne i denna mapp skapar Rocketeer tre mappar: Current, Releases och Shared. Current-mappen är egentligen endast en länk till en mapp i Releases-mappen. Releases-mappen innehåller ett förbestämt antal utgivningar i egna mappar. Shared innehåller filer som är delade mellan olika utgivningar, till exempel bilder och konfigurationsfiler.

Design

All kod i projektet kommer att sparas i versionshanteringssystem för att ge tillgång till historik och enkel delning av koden till andra utvecklare. Kodstilen för språket som används ska följas och en introduktionstext ska finnas tillgänglig i versionshanteringssystemet.

Versionshanteringssystemet som ska användas heter Git. Även om Rocketeer och versionshanteringsleverantören stödjer andra versionshanteringssystem så är det Git som används av Gambit i alla övriga projekt.

Den officiella kodstilen för språket ska användas. Detta är för att behålla läsbarheten och enigheten med annan kod skriven med samma språk. Det är godtaget att gå emot den officiella kodstilen i enskilda fall ifall det förbättrar läsbarheten vid till exempel variabelnamn.

Tillsammans med koden ska det finnas en fil som heter “Readme.md”. Denna fil ska innehålla all information en utvecklare, med liknande kunskapsnivå som skribenten av filen själv har, behöver för att komma igång med vidareutveckling av projektet. Denna fil innehåller vanligtvis en lista över steg för hur man kommer igång och en kort introduktion till vad koden gör.

Implementation

Verktyget använder sig av en modulär uppbyggnad för att ny funktionalitet lätt ska kunna läggas till utan att nuvarande funktionalitet riskerar att ändras. Modulerna läses in endast om de är tillsatta i arbetsflödet som valts. Om till exempel arbetsflödet *skapa webbshop* är valt så kommer verktyget endast att läsa in de moduler som används för det arbetsflödet. Arbetsflöden skapas genom att redigera en konfigurationsfil med samma namn som arbetsflödet.

För att modulerna ska kunna läsas in och köras endast vid behov måste de följa ett specifikt gränssnitt. Detta gränssnitt kräver att modulen förverkligar åtminstone en klass och en initieringsmetod som returnerar en instans av klassen. Klassen måste implementera två metoder, en som returnerar modulens namn i läsbar form och en som startar exekveringen av modulen.

Utvärdering

Resultatet av distribueringsfunktionen i verktyget är att en webbshop körs korrekt på den angivna servern. Detta inkluderar även valt tema för webbshoppen och eventuella moduler. Dessa moduler behöver dock ännu konfigureras manuellt.

Enligt respons från användare av verktyget är det väldigt lätt att använda jämfört med hur processen varit tidigare. Det största problemet från en användarsynvinkel var att versionen av Python-programmet inte var tydligt beskrivet från början och att det resulterande felmeddelandet inte var lätt att tyda.

Användaren behöver fortfarande manuellt fylla i all information om webbshopen men efter det sköter verktyget automatiskt om att utföra de nödvändiga uppgifterna för att få webbshopen att fungera. Dessa uppgifter görs säkert över en krypterad kommunikationskanal till servern.

Verktyget kan korrekt skapa och radera en webbshop enligt den information som användaren fyllt i. Att modifiera en webbshop är dock inte möjligt i praktiken eftersom det endast kan göras om inte någon använt webbshopen ännu. Detta beror på hur Prestashop hanterar sina konfigurationer. Dessa funktioner har en betydande

effekt på hur snabbt hanteringen av webbshopar kan göras. Tiden det tar att skapa en ny webbshop har gått ner från flera timmar till några minuter.

Framtida förbättringar

Verktyget innehåller endast en minimal uppsättning av funktioner som genererar mest värde. Om verktyget vidareutvecklas så skulle jag rekommendera funktioner som ger arbetsflödet mera värde, till exempel möjlighet att endast göra små ändringar i specifika moduler eller att uppdatera PrestaShops version.

Att ändra verktyget så att det körs som en tjänst på en server skulle möjliggöra bättre integration med versionshanteringssystem. Detta kan sedan vidareutvecklas till en form av kontinuerlig distribuering, vilket leder till färre manuella steg i distribueringsprocessen.

Den nuvarande arkitekturen med modulär struktur gör det möjligt att snabbt skapa nya arbetsflöden, men denna arkitektur lämpar sig för tillfället inte bra för enhetstestning och integreringstestning. En vidareutveckling på denna front skulle minska risken att fel uppstår vid ändringar till verktyget.

Slutsats

Resultatet av denna avhandling är ett verktyg som kan till stor del automatiskt distribuera en PrestaShop-webbshop till en webbserver. Verktyget kan distribuera en ny webbshop med konfiguration för webbservern och sätta upp en ny databas och databasanvändare. Verktyget kan också radera en webbshop som skapats med verktyget och redigera en webbshop till viss mån.

Att skapa ett bra DevOps-arbetsflöde är mycket svårare och mera tidskrävande än man kunde tro. Det kräver noggrann design och kontinuerlig vidareutveckling för att behålla sin användbarhet i en värld som förändras snabbt.

Rocketeer har många bra sidor men det har inte fungerat smidigt att använda med PrestaShop-distribueringar. PrestaShop hanterar sina konfigurationer genom att modifiera filer på flera ställen i källkoden. För att Rocketeer ska fungera korrekt

mellan distribueringar måste alla dessa filer finnas i den delade mappen. Detta fungerar bra i början när en webbshop skapas men om PrestaShop-moduler ändras från PrestaShops inbyggda administrationssida så kommer nya konfigurationsfiler att skapas och de finns inte med i den delade mappen.

Bibliography

[1] System requirements for PrestaShop 1.7 :: PrestaShop Developer Documentation. PrestaShop Developer Documentation, https://devdocs.prestashop-project.org/1.7/basics/installation/system-requirements/ (Last read: 7.1.2024)

[2] Lerdorf, Rasmus. History of PHP - Manual. PHP, http://php.net/manual/en/history.php.php (Last read: 7.1.2024)

[3] Pleva, Justin T. PHP: Hypertext Preprocessor. 2013.

[4] Bârsan, Ghiţă, and Oancea Romana. Considerations on e-commerce platforms. 2014.

[5] About PrestaShop, the open source e-commerce software. PrestaShop, https://prestashop.com/about-us/ (Last read: 7.1.2024)

[6] Git Feature Branch Workflow. Atlassian, https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow (Last read: 4.2.2024)

[7] Pull Requests. Atlassian, https://www.atlassian.com/git/tutorials/making-a-pull-request (Last read: 4.2.2024)

[8] imp load_source, https://docs.python.org/2/library/imp.html#imp.load_source (Last read: 7.1.2024)

[9] Sarcar, Vaskaran. Java Design Patterns: A Hands-On Experience with Real-World Examples. Apress, 2022.

[10] What is CI? - Continuous Integration Explained. AWS, https://aws.amazon.com/devops/continuous-integration/ (Last read: 4.2.2024)

[11] Humble, Jez, and David Farley. Continuous Delivery. Addison-Wesley, 2011.

[12] Rocketeer. Home, http://rocketeer.autopergamene.eu/I-Introduction/Whats-Rocketeer.html (Last read: 7.1.2024)

[13] Capistrano. A remote server automation and deployment tool written in Ruby., 1 June 2013, https://capistranorb.com/ (Last read: 7.1.2024)

[14] Ansible. Ansible is Simple IT Automation, https://www.ansible.com/ (Last read: 7.1.2024)

[15] Python 3.0 Release. Python.org, https://www.python.org/download/releases/3.0/ (Last read: 10.2.2024)

[16] Sunsetting Python 2. Python.org, https://www.python.org/doc/sunset-python-2/ (Last read: 10.2.2024)

[17] pip documentation v24.0, https://pip.pypa.io/en/stable/ (Last read: 10.2.2024)

[18] What is DevOps? Atlassian, https://www.atlassian.com/devops (Last read: 7.1.2024)