Fully automating website deployment

Utilizing a continuous deployment pipeline for a blog

Insinööri (AMK)
2022
Joona “Joodari”

Tiivistelmä

Tekijä(t) Joona ”Joodari”	Julkaisun laji Tutkimusseminaarityö	Valmistumisaika 2022
	Sivumäärä 12	
Työn nimi **Webbi-sivuston julkaisun täysin automatisointi** CD putken hyödyntäminen blogisivustolla.		
Tutkinto ja koulutusala Insinööri (AMK)		
Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)		
Tiivistelmä Työ kuvaa itse hallituilla palvelimilla blogisivuston automaattisen julkaisun toteutusta. Tavoitteena on siis rakentaa ja siirtää sivusto versionhallinnasta tuotantopalvelimelle automaattisesti, niin että sivusto tulee suoraan automaattisesti näkyville sivuston lähdekoodin päivityksen jälkeen. Sivuston moniaskelisen julkaisun automaatiolla säästetään aikaa, jolloin blogin kirjoitukseen jää enemmän aikaa. Avoimen lähdekoodin projekteja käyttämällä säästämme myös rahaa, sekä vähennämme tarvittavan ylläpidon määrää. Tämä myös helpottaa olemassa olevaan infrastruktuuriin integrointia, ja auttaa välttämään riippuvuuksia kolmansiin osapuoliin, jolloin datan käsittelyn voidaan varmistaa hoituvan yksityisesti ja turvallisesti. Käytettyihin teknologioihin sisältyy Gitea versionhallintaa varten, Drone CD putkea varten, Docker konttien käyttöön, OpenSSH webbi-sivun tiedostojen siirtoa varten SFTP protokollaa käyttäen, sekä nginx web serveriä.		
Asiasanat Automaatio, Avoin lähdekoodi, Ohjelmisto infrastruktuuri, Staattisen webbi-sivuston rakennus, Jatkuva Integraatio		

Abstract

Author(s) Joona “Joodari”	Type of Publication Thesis seminar	Published 2022
	Number of Pages 12	
Title of Publication **Fully automating website deployment** Utilizing a continuous deployment pipeline for a blog		
Degree, Field of Study Engineer (UAS)		
Organisation of the client (if the thesis work is commissioned by another party)		

Abstract

This thesis describes an automated website publishing pipeline for a blog. The goal is to have the resulting updated website be visible to the public from a production server automatically after any modifications have been pushed to it’s source code repository.

Automating away the tedious steps of publishing a blog site saves time, allowing the writer to focus on the actual writing. Utilizing Open Source technologies for the pipeline reduces costs and required maintenance for the system. It is also easier to integrate into preexisting infrastructure, and since it avoids reliance on third parties, allowing the data to be managed in a private and secure way.

The technologies presented here include Gitea for versioning control, Drone for Continuous Deployment, Docker for containerization, OpenSSH for accepting files in the deployment server via SFTP, and nginx for the production web server.

Keywords

Automation, Open Source, Software Infrastructure, Building a static website, Continuous Integration

CONTENTS

Attachments

1 Introduction

There is clear demand for static websites. There are many services that offer a build pipeline from a writer friendly source format to automate publishing the website to production (Cloudflare). Though taking care of all the infrastructure is advertised as a benefit of these services, it’s also their downside, since they are heavily tied to their specific vendors’ infrastructure and services (Github). The goal of this paper is designing and implementing a more vendor neutral publishing pipeline that’s secure, uncomplicated, performant, and scalable.

Having the whole publishing process automated reduces manual labor, and reduces risks of problems appearing. It also offers additional benefits, like the publishing process being a lot faster than if it was to be done manually, allowing changes to be deployed faster. (Sullivan 2014)

The pipeline shouldn’t rely on third parties, so that it can be modified and adapted even for customers with strict data security and privacy requirements. This can be helpful as the communications and technology industry has to take privacy and security matters more seriously due to legislation like the European Union’s General Data Protection Regulation for example.

The plan is for the pipeline to start at a git repository, which is an industry standard for versioned source control. It shall contain the blog site in a form that’s suitable for writers, but which will require some processing to turn into the actual static website. At the end of the planned pipeline, the goal is for the generated static website to have ended up the production server, where the web server will simply serve the site’s static files to visitors over HTTPS.

To implement the pipeline, popular Open Source technologies will be utilized, such as OpenSSH, Gitea, Docker, Drone and nginx, which implement standard protocols such as HTTPS, Git, SSH, and SFTP.

2 Technical details

2.1 Infrastructure

Since one of the goals is to not depend on external vendors, the servers can't be tied to be required to be ran by a third party. Having services tied to specific machines also isn't optimal, as it makes scaling the whole system harder. Docker containers can be used to abstract away which machine something is running on to allow for better scalability of the services, while also not giving up performance, which would be the case with virtual machines (Vase 2015, 15).

An assumption that the services have network connections to reach each other will be made, as the specifics of the networking isn't too relevant to this paper. In this setup it is achieved via the servers being publicly reachable over the internet, and DNS names that point to each service that needs to be resolvable and communicating with from another service. Additionally, some services are defined in the same docker-compose file. This allows them to be able to connect to each other more easily using the services names which Docker Compose sets up for their networks (Docker).

Ultimately though, the goal is for the physical infrastructure to not matter much, and be easy to change and scale. Although in this setup only a few real servers are used, there's no reason it couldn't be scaled up to hundreds of servers with minor tweaks. Though the uploading step would need to then take into account multiple possible production services which is outside the scope of a simple blog site. Instead, if further scaling for a static blog site was required, the deployment and web server could be hidden behind HTTP caching services and reverse proxies, like nginx or Varnish for example (Pyhäranta 2019, 14).

2.2 Source control

The source for the website has to be stored somewhere so that writers can access and edit it. Git is a common tool used for storing software source code, and it allows for multiple collaborators to edit the contents at the same time. Although git is primarily aimed towards developers, it nowadays has simple to use graphical user interfaces, making it suitable enough for blog editors to use as well. Using git provides the added benefit that the full edit history of the files will be available, making for easier tracking of revision changes of posts and edit dates.

There are more than a few ways to synchronize the git repository's state. Gitea is one such way, and it offers a web view, user accounts to limit access, and webhooks for the Continuous Deployment to use.

The source code of the website will all be stored in the same repository. This includes the sass style and HTML template files, and the actual contents of the blog posts, which are the markdown files, as is standard practice (Hugo a). It's assumed that writers know to navigate to the markdown files, and that additional pull request checks can be enforced in Gitea in case unwanted modifications to the rest of the style needs to be prevented.

2.3 Continuous Deployment and Integration

Continuous Deployment is the process of automating publishing changes. It's often also referenced to as Continuous Delivery or Continuous Integration, though those are often more specifically about testing the code with integration tests and checking that it builds, instead of delivering the result of the changes (Pittet).

The goal of the Continuous Deployment pipeline is to take the source of the blog site, build the web browser viewable version of it to static files, and then move that result to the destination web server. As the blog site uses Hugo, building the site is as simple as running the build command in the Hugo docker container, which will generate the static website's files (Hugo b).

Drone can be used to manage and run the Continuous Deployment pipelines, and it works well with Docker. It also is the only Continuous Integration and Deployment software which both supports Gitea and is supported by Gitea officially in their documentations. Although the setup could be done with other similar software as well, like Jenkins for example.

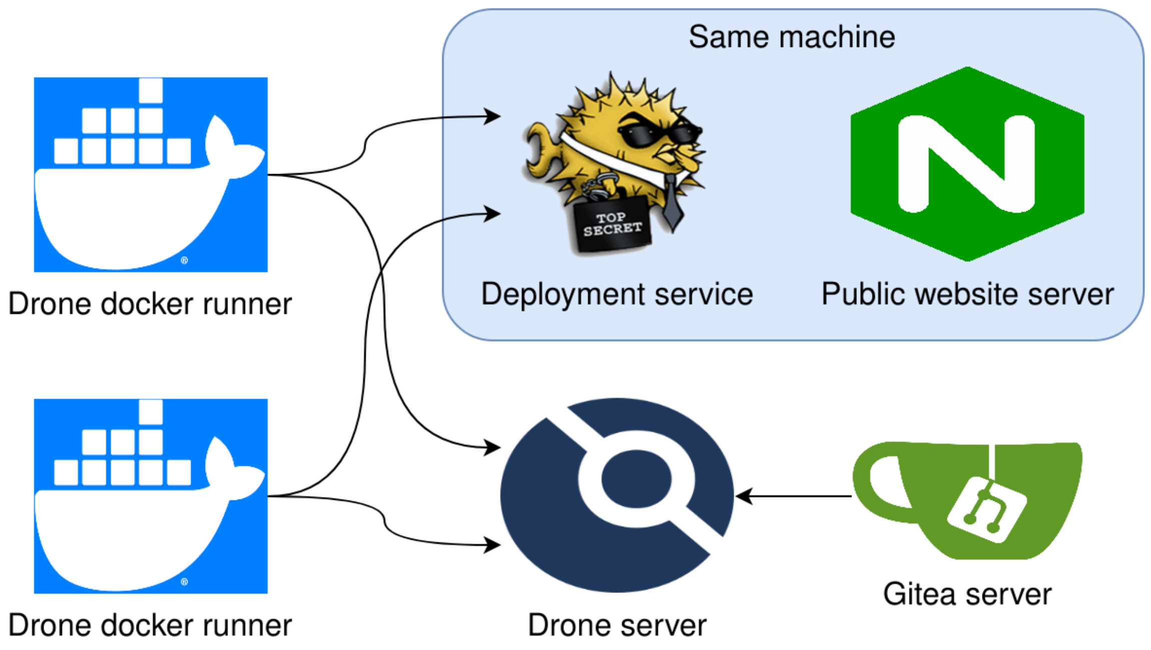


Figure 1. Container connections with runner scaling

Drone is split into the actual Drone management server and the runners, as can be seen in the above illustration. The Drone server is the part that communicates with the source control service, and manages the tasks. Runners as their name suggests, are responsible for running the Drone pipelines, and communicating back the the Drone server. The Drone server will take care of distributing tasks across runners if there’s multiple available, so scaling the system is easy if there’s ever need for more runners. A single Drone docker runner instance will be enough for this setup though, as only running a single blog building and publishing pipeline is required at a time.

2.4 Deployment service

The CD pipeline needs to be able to push the static website files to the publicly facing web server. However, doing so securely across different docker containers and possibly different physical machines complicates things. Some kind of a service needs to be setup on the same machine as the web server, like seen from the last section’s diagram. It will handle accepting the files from the Continuous Deployment service. It can possibly be defined in the same docker-compose file as the web server for simplicity.

There are multiple suitable file transfer protocols. SFTP has the broadest support and simplicity. It’s also very reliable and not as prone to possibly being abused if an attacker gains access as scp/ssh/rsync possibly could be. The standard server used for it is OpenSSH, which can be configured to only allow SFTP to keep the system secure and not allow arbitrary command execution (OpenBSD).

Additional care still needs to be taken care of with file permissions and access though. Tools such as chroot can be used to further restrict the SFTP user account, so that even in the case of a Continuous Deployment pipeline compromise, only the resulting static website can be modified whilst the rest of the production system will be able to resist modification attempts.

2.5 Deployment web server

A web server capable of servicing static files is required, for example nginx or Apache2. Note that unlike most other services described in this thesis, the web server has to be on the same physical host machine as the deployment service, since it needs to able to access the files from the deployment service via a shared docker volume.

Using a standard web server and a static website has the benefit that the web request responses can be aggressively cached. Compared to WordPress and most other blogging solutions which require running more code to respond to each request, the approach also improves security, as static sites minimize the attack surface to only the web server, And since the most popular web servers are so widely used, and thus battle tested, that they are not likely to have critical security flaws. (Petersen 2016, 16.)

2.6 Overview of the pipeline

The flow of the planned pipeline starts at the source code repository service, Gitea. When an update gets pushed to the repository by a writer, Gitea will inform the Drone server with a webhook, which is just a standard HTTPS REST JSON request.

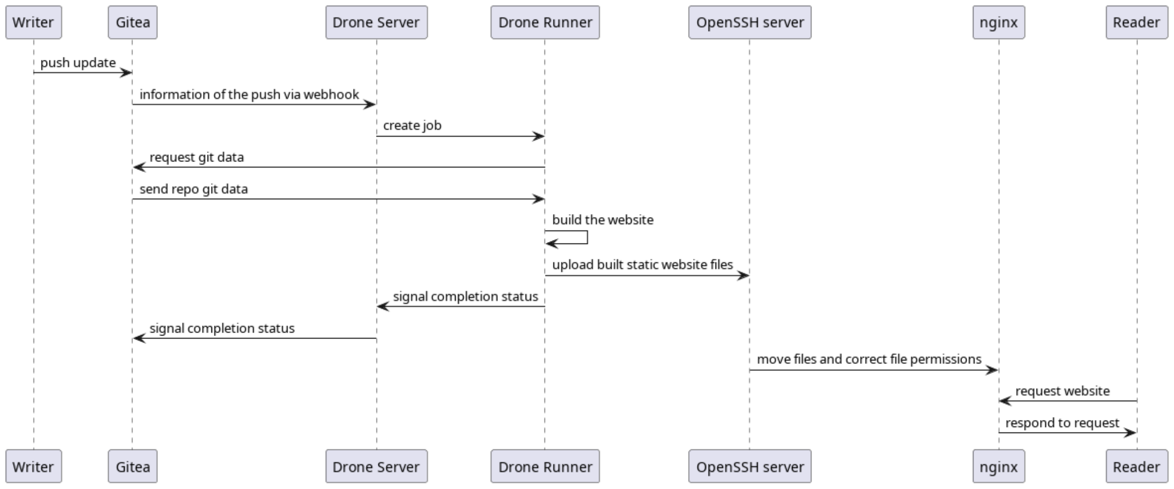


Figure 2. Pipeline UML sequence diagram

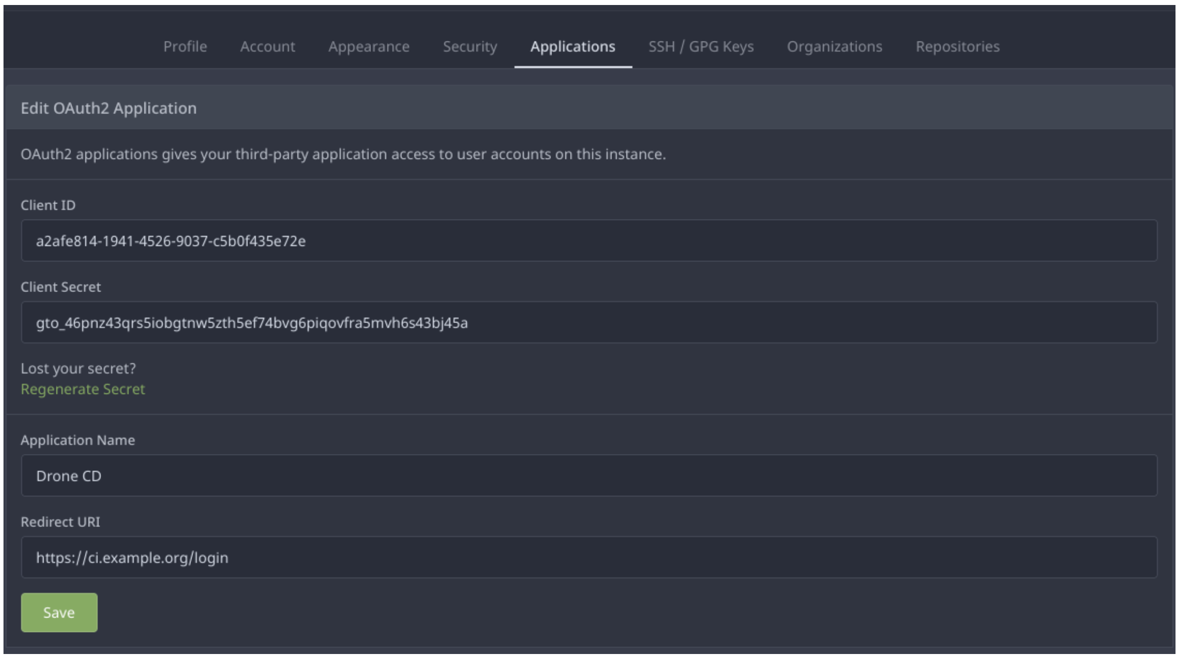
As the pipeline diagram illustrates, the Drone server will then wait for a suitable runner to be available to run the job, and send a message to it to start processing the pipeline once it’s available. The runner will then pull in the code from Gitea using the authentication that the Drone server provided in the pipeline start message, build the website, and then contact the deployment service. Jumping forward a bit in the diagram, after the upload to the deployment service is done, the runner will signal to the Drone server that the pipeline has reached it’s completion, which will send the signal to Gitea so that Gitea can display the pipeline status on the repository.

On the deployment end, the deployment service will receive the files via OpenSSH’s SFTP, and it will handle replacing the files in the directory that the web server has been configured to serve static files from once the transfer is complete. Note that in the diagram the only displayed request happens after the site has been swapped to the newly uploaded one, though that it’s assumed that readers may be constantly requesting the files from nginx.

3 Pipeline data source

Installing Gitea with docker-compose is relatively simple, and can be done just by copying the Docker compose configuration file and starting the containers (Gitea). The rest of the standard setup can be done after connecting to it’s web server from a web browser.

Next the OAuth2 application for the CD service needs to be created from Gitea’s settings menu. The application name can be set to whatever, but the redirect URI should resolve to the login page of the Drone server instance.



Edit OAuth2 Application

OAuth2 applications gives your third-party application access to user accounts on this instance.

Client ID

a2afe814-1941-4526-9037-c5b0f435e72e

Client Secret

gto_46pnz43qrs5iobgtnw5zth5ef74bvg6piqovfra5mvh6s43bj45a

Lost your secret?
Regenerate Secret

Application Name

Drone CD

Redirect URI

https://ci.example.org/login

Figure 3. Gitea OAuth2 configuration

In the above image it’s set to https://ci.example.org/login, which would mean that the CI service is planned to be resolvable from the ci.example.org domain via a DNS lookup. The Client ID and Client Secret need to be stored for later when the CD service is being set up.

The OAuth2 application will allow the Drone server to use the Gitea instance for user authentication. When an user authenticates and grants the Drone server access to their account, Drone can then proceed to add the required webhooks to the repositories that the user enables processing CD pipelines for. The webhooks are configured by Drone so that it gets notified whenever a change happens in the repository that may trigger a pipeline run.

4 Pipeline Target

4.1 Deployment web server

Support for docker is included by the nginx team, and setting it up requires running only a few commands (Nelson and Garcia 2021). Though as per usual, in this setup the the ports and volumes will be defined in the docker compose configuration file for easier management.

Thought needs to be put into the user and group that nginx will run as, as the files will need to be accessible to it. Any extra privileges should be dropped though for security reasons, though nginx does quite a lot of it by default already. However at least basic things such as maximum connection timeout limits should likely be set to lower values, as not overloading the origin server is a priority.

4.2 Deployment listener service

OpenSSH can be easily installed into most base system images. To avoid reinventing the wheel, the `atmoz/sftp` docker image can be used as it already includes a strict configuration with chroot enabled for the users. It also already contains handy scripts for automatically creating the users, and starting more custom scripts. The user creation script reads a simple configuration file, and runs basic Linux commands to create the users with the ssh keys added to them.

The script in the attachment 1 file will be the part that’s actually responsible for awaiting for completed file transfers, calling the attachment 2 script file which will handle switching over the website’s files and while also correcting the file permissions. This enables being able to give very minimal permissions to the SFTP uploader users and the web server user. It also avoids issues with partial file transfers, and always cleans up any leftover files, as the folder is completely replaced whenever a new build job indicates that it’s upload is done by writing it’s version to a specific file.

The deployment directory should be mounted from the host system to the same folder that was setup in the web server container image. This is required so that the web server can access the files.

5 Continuous deployment

5.1 Source control connection

The drone server installation is quite simple, and well documented (Drone a). After creating random shared secret for the Drone server and the runners, the OAuth2 application client id and secret from earlier when Gitea was setup need to be defined using docker environment variable definitions.

Additionally the Gitea server's URI needs to be defined, as well as the drone server's own host. The URIs should be resolvable from the other services that need them. After having defined all the options in the compose file, the Drone server can be started. Then in a web browser, it can be logged into, and the pipeline can be enabled for the target repository.

5.2 Build steps

The docker Drone runner setup is easier than the main Drone management service, as it only requires the shared secret and the host of the Drone server (Drone b). The runner will automatically connect to the server, and run any jobs that the server assigns to it.

The build step will use the official Hugo docker image as planned. After the building step is done, any container with SFTP can be used to upload the site's files. After the sites files have been successfully uploaded, the last uploaded file should be written to a specific location with the upload folder name, so that the deployment service knows to deploy the site from there. The configuration for the Drone pipeline is stored in the repository in a .drone.yml file. An example of this configuration can be found from the attachment 3.

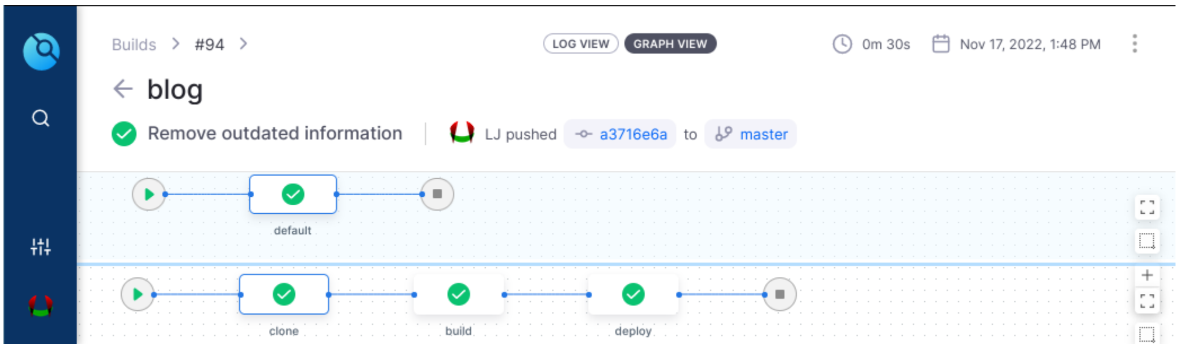


Figure 4. Drone graph view of a simple automation pipeline

Now that everything is setup, doing a push to the repository this all was setup for, the result is that the above pipeline is ran. After a slight delay after the push the resulting website shows up on the production site. The slight delay can be explained by the data being moved between the different servers and containers, and the build and deployment processes taking a short while to initialize.

6 Summary

With the implemented automated deployment pipeline, a fully automated system that reduces manual labor and speeds up updating of a statically built site has been achieved, where just pushing code to the repository leads to the live production website being automatically updated. And it's been fully done with open source technologies and protocols, in a fashion which allows easily tailoring the pipeline for any current or possible future requirements.

The implementation also avoids any vendor lock-in. The container approach with proven technologies also ensures that the pipeline implementation stays relatively low maintenance to keep running, and can be scaled on the go as needed.

It's still also worth comparing the implementation to GitHub pages and Cloudflare pages, which were mentioned in the introduction. Generally this deployment pipeline is as easy to use, but it does involve the covered more complicated first time manual setup. Additionally there's a trade-off between the flexibility and configuration of the implemented pipeline, versus the simplicity of the aforementioned solutions.

Another big advantage of the vendor specific automation solutions for organizations might be the support and documentation, and not having to think about the maintenance at all. What this pipeline offers in exchange though is full control of the data, which enables potential use in domains with potentially private information which can't legally be shared to most third parties, like in healthcare for example.

Overall, the goals of the pipeline implementation can be considered to have been met. The automation pipeline provides the benefits of the simpler vendor specific solutions, while avoiding their downsides, with just a bit more initial setup and minimal maintenance over time.

References

Cloudflare. Cloudflare Pages. Website. Referenced on 8.10.2022. Available at https://pages.cloudflare.com/

Docker. Networking in Compose. Documentation. Referenced on 8.10.2022. Available at https://docs.docker.com/compose/networking/

Drone a. Gitea. Documentation. Referenced on 9.10.2022. Available at https://docs.drone.io/server/provider/gitea/

Drone b. Install on Linux. Documentation. Referenced on 9.10.2022. Available at https://docs.drone.io/runner/docker/installation/linux/

Gitea. Installation with Docker. Documentation. Referenced on 8.10.2022. Available at https://docs.gitea.io/en-us/install-with-docker-rootless/

GitHub. About GitHub Pages. Documentation. Referenced on 8.10.2022. Available at https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages

Hugo a. Directory Structure. Documentation. Referenced on 8.10.2022. Available at https://gohugo.io/getting-started/directory-structure/

Hugo b. Installing Hugo. Documentation. Referenced on 8.10.2022. Available at https://gohugo.io/getting-started/installing/

Nelson, R. Garcia, A. Deploying NGINX and NGINX Plus with Docker. Blog post. nginx. Referenced on 9.10.2022. Available at https://www.nginx.com/blog/deploying-nginx-nginx-plus-docker/

OpenBSD. sshd_config(5). Documentation. Referenced on 8.10.2022. Available at https://man.openbsd.org/sshd_config

Petersen, H. 2016. From Static and Dynamic Websites to Static Site Generators. University of Tartu. Bachelor’s thesis. Available at https://dspace.ut.ee/bitstream/10062/56282/1/thesis.pdf

Pittet, S. Continuous integration vs. delivery vs. deployment. Atlassian. Guide. Referenced on 8.10.2022. Available at https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Pyhäranta, M. 2019. Vapaan ohjelmiston HTTP-kiihdyttimien vaikutus verkkosivujen suorituskykyyn. Haahe-Helia ammattikorkeakoulu Oy. Research report. Available at https://markuspyharanta.com/wp-content/uploads/2019/02/Tutkimusraportti-Markus-Pyh%C3%A4ranta.pdf

Sullivan, J. 2014. Consider Deployment Automation to Add Business Value Faster. Article. Referenced on 7.11.2022. Available at https://www.cmcrossroads.com/article/consider-deployment-automation-add-business-value-faster

Vase T. 2015. Advantages of Docker. University of Jyväskylä. Bachelor’s thesis. Available at http://urn.fi/URN:NBN:fi:jyu-201512093942

Attachment 1. switch-deploy-watch.sh

```
#!/bin/sh
while true
do
        touch  /last-deploy-watch-run
        sleep 30
        find /home/*/upload/build_number.txt -cnewer \
        ./last-deploy-watch-run -exec /usr/local/bin/switch-deploy.sh {} \;
done
```

Attachment 2. switch-deploy.sh

```
#!/bin/sh
set -e

BUILD_VERSION_FILE_PATH="$1"
BUILD_VERSION="$(head --lines=1 "$BUILD_VERSION_FILE_PATH")"

if [ ! -z "${BUILD_VERSION##*[!0-9]*}" ]; then
  echo "Deploying $BUILD_VERSION_FILE_PATH $BUILD_VERSION"
else
  echo "Build version needs to be an int"
  exit 1
fi

BASE_DIR="$(dirname "$BUILD_VERSION_FILE_PATH")"
BASE_DIR="$(realpath "$BASE_DIR"/..)"
UPLOAD_DIR="$BASE_DIR/upload/$BUILD_VERSION"

TMP_DIR="$(mktemp -d)"
mv "$BASE_DIR/deploy"/* "$TMP_DIR/"
mv "$UPLOAD_DIR"/* "$BASE_DIR/deploy/"
rm -r "$TMP_DIR" "$UPLOAD_DIR"
```

Attachment 2. .drone.yml

```
kind: pipeline
type: docker
name: default
steps:
- name: build
  image: klakegg/hugo:ext-alpine-ci
  commands:
  - hugo --minify –enableGitInfo && echo "$DRONE_BUILD_NUMBER" > build_number.txt
- name: deploy
  image: alpine:latest
  commands:
  - apk add openssh && echo "$SSH_KEY" > "priv.key" && chmod 0600 "priv.key"
  - mkdir -p "$HOME/.ssh" && echo "$DEPLOY_IDENTITY" > "$HOME/.ssh/known_hosts"
  - chmod 600 "$HOME/.ssh/known_hosts"
  - echo "mkdir upload/$DRONE_BUILD_NUMBER" > deploy.txt
  - echo "put -R public/* upload/$DRONE_BUILD_NUMBER" >> deploy.txt
  - echo "put build_number.txt upload/build_number.txt" >> deploy.txt
  - echo "bye" >> deploy.txt
  - echo "$DRONE_BUILD_NUMBER" > build_number.txt
   - sftp -i "priv.key" -b deploy.txt -P "$DEPLOY_PORT" "$DEPLOY_USERNAME@$DE-
PLOY_HOST"
  environment:
    SSH_KEY:
      from_secret: deploy_ssh_key
    DEPLOY_USERNAME:
      from_secret: deploy_username
    DEPLOY_HOST:
      from_secret: deploy_host
    DEPLOY_IDENTITY:
      from_secret: deploy_identity
    DEPLOY_PORT:
      from_secret: deploy_port
```