

SSW 590 — DevOps

Group 08

Version 1.2

by

Samah Keshiro and Kayla Holmes

Stevens Institute of Technology

09/23/26

Document History

Table 1: Document Update History

Date / Version	Author Initials	Change	Changed At
September 24, 2026 / 1.2	SK	Completed the Digital Ocean Docker Deployment chapter; the class-based JavaScript refactor of the two-button website with its accompanying UML class diagram; the LaTeX Docker chapter ; and the Project Proposal chapter, addressing the two-tool comparison, dashboard, SWOT, and unique-contribution rubric criteria.	Chapter 2 , Chapter 3 , Chapter 4 , Chapter 5
September 16, 2026 / 1.1	SK, KH	Completed the Linux Commands assignment: environment setup with terminal printout figure and full solutions to Problem Set sections A–E	Chapter 1

Table of Contents

Document History	ii
1 Linux Commands	1
1.1 Environment Setup	1
1.2 Problem Set Answers	1
1.2.1 A) Navigation & File Ops	1
1.2.2 B) Viewing & Searching	3
1.2.3 C) Text Processing	4
1.2.4 D) Permissions & Ownership	5
1.2.5 E) Links & Find	7
1.3 F) Processes & Job Control	7
1.3.1 24. Show your processes in a tree view	7
1.3.2 25. Start sleep 12 in the background and show its PID	8
1.3.3 26. Send a TERM signal to all sleep processes owned by you	8
1.3.4 27. Show the top 5 processes by memory usage	8
1.4 G) Archiving & Compression	8
1.4.1 28. Create a gzipped tar archive src.tgz from src/ with relative paths	8
1.4.2 29. List the contents of src.tgz without extracting	9
1.4.3 30. Extract only file2.txt from src.tgz into tmp/	9
1.5 H) Networking & System Info	9
1.5.1 31. Show all listening TCP sockets with associated PIDs	9
1.5.2 32. Print your default route (gateway) in concise form	9
1.5.3 33. Display kernel name, release, and machine architecture	10
1.5.4 34. Show the last 5 successful logins (or last sessions) on the system	10
1.6 I) Package & Services (Debian/Ubuntu	10
1.6.1 35. Show the installed version of package coreutils	10
1.6.2 36. Search available packages whose names contain ripgrep	10
1.6.3 37. Check whether service cron is active and print its status line only	11
1.7 J) Bash & Scripting	11
1.7.1 38. Write a one-liner that loops over *.txt in src/ and prints: :	11
1.7.2 39. Write a command that exports CSV rows where dept == "CS" to cs.txt	11

1.7.3	40. Create a variable X with value 42, print it, then remove it from the environment.	11
2	Digital Ocean Docker Deployment	12
2.1	Overview	12
2.2	Create the DigitalOcean Droplet	12
2.3	Connect to the Droplet Using SSH	12
2.4	Update the Ubuntu Server	13
2.5	Install and Verify Docker	13
2.6	Transfer the Website to the Droplet	13
2.7	Build the Docker Image	14
2.8	Run the Docker Container	14
2.9	Configure the Firewall	15
2.10	Access the Deployed Website	15
3	Class Based Javascript Websites	16
3.1	Overview	16
3.2	UML Class Diagram	16
4	Latex Docker	18
4.1	Goal	18
4.2	Project Files	18
4.2.1	The LaTeX Source (<code>hello.tex</code>)	18
4.2.2	The Dockerfile	19
4.3	Building the Image	19
4.4	Running the Container and Retrieving the PDF	19
4.5	Verifying the Output	20
4.6	Relation to Overleaf	20
5	Project Proposal	21
5.1	Project Title	21
5.2	Project Description	21
5.3	Sample Tasks and DevSecOps Tools	21
5.4	Planned Comparison Criteria	22
5.5	Unique Contribution	22
6	Jenkins/CI	24
6.1	Student Pack domain and DNS	24
6.2	Pinned Gitea Deployment	26
6.3	Database and Persistence	27
6.4	HTTPS/certificate	29
6.5	Firewall/exposure	30
6.6	Startup team workflow	31
6.7	Architecture and operation	31

Bibliography	33
A AI Prompt Log	34
A.1 How to Use This Appendix	34
A.2 Example Record	34
A.3 Log Entries	34

List of Tables

1	Document Update History	ii
---	-----------------------------------	----

List of Figures

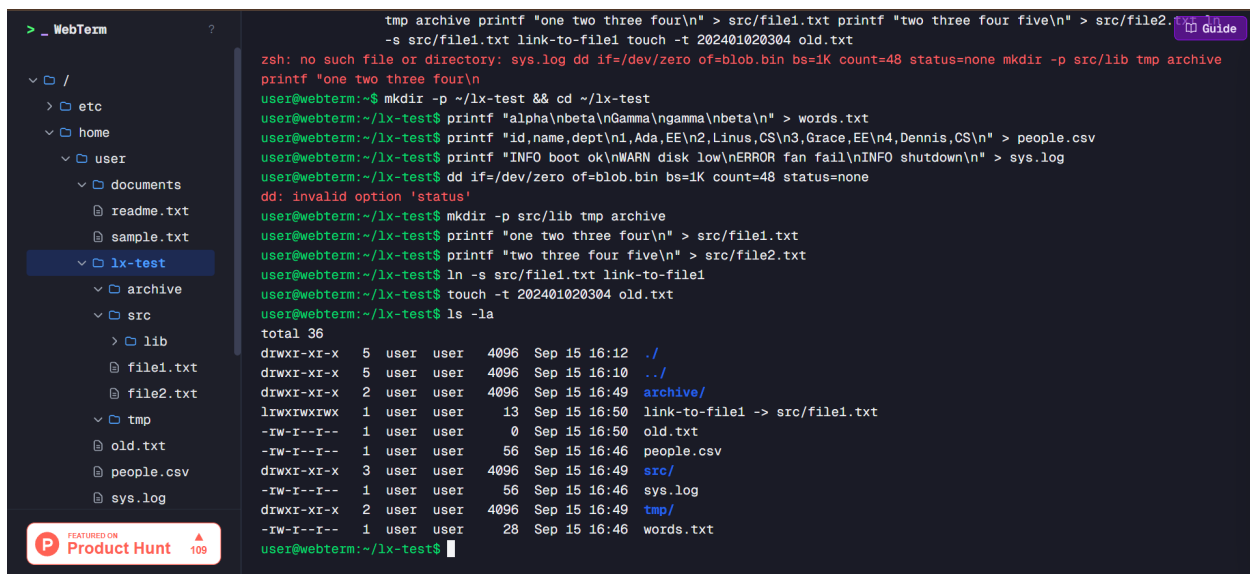
1.1	Enter Caption	1
3.1	UML class diagram for the class-based Color Buttons application	17
6.1	Namecheap's Advanced DNS panel	25
6.2	nslookup	25
6.3	10/01/26 docker compose ps with Gitea container running	27
6.4	10/01/26 Before docker compose down	28
6.5	10/01/26 After docker compose down	29
6.6	Showcase of Gitea running on HTTPS	30
6.7	Docker showcasing reverse proxy confirmation	30
6.8	Terminal Showing exposed ports from firewall	30
6.9	SSH Key working for Gitea	31
6.10	Architecture Diagram	32

Chapter 1

Linux Commands

1.1 Environment Setup

All commands below were run after creating a scratch directory:



```
tmp archive printf "one two three four\n" > src/file1.txt printf "two three four five\n" > src/file2.txt
-s src/file1.txt link-to-file1 touch -t 202401020304 old.txt

zsh: no such file or directory: sys.log dd if=/dev/zero of=blog.bin bs=1K count=48 status=none mkdir -p src/lib tmp archive
printf "one two three four\n"
user@webterm:~$ mkdir -p ~/lx-test && cd ~/lx-test
user@webterm:~/lx-test$ printf "alpha\nbeta\nGamma\ngamma\nbeta\n" > words.txt
user@webterm:~/lx-test$ printf "id,name,dept\n1,Ada,EE\n2,Linus,CS\n3,Grace,EE\n4,Dennis,CS\n" > people.csv
user@webterm:~/lx-test$ printf "INFO boot ok\nWARN disk low\nERROR fan fail\nINFO shutdown\n" > sys.log
user@webterm:~/lx-test$ dd if=/dev/zero of=blog.bin bs=1K count=48 status=none
dd: invalid option 'status'
user@webterm:~/lx-test$ mkdir -p src/lib tmp archive
user@webterm:~/lx-test$ printf "one two three four\n" > src/file1.txt
user@webterm:~/lx-test$ printf "two three four five\n" > src/file2.txt
user@webterm:~/lx-test$ ln -s src/file1.txt link-to-file1
user@webterm:~/lx-test$ touch -t 202401020304 old.txt
user@webterm:~/lx-test$ ls -la
total 36
drwxr-xr-x  5 user user  4096 Sep 15 16:12 ./
drwxr-xr-x  5 user user  4096 Sep 15 16:10 ../
drwxr-xr-x  2 user user  4096 Sep 15 16:49 archive/
lrwxrwxrwx  1 user user   13 Sep 15 16:50 link-to-file1 -> src/file1.txt
-rw-r--r--  1 user user    0 Sep 15 16:50 old.txt
-rw-r--r--  1 user user   56 Sep 15 16:46 people.csv
drwxr-xr-x  3 user user  4096 Sep 15 16:49 src/
-rw-r--r--  1 user user   56 Sep 15 16:46 sys.log
drwxr-xr-x  2 user user  4096 Sep 15 16:49 tmp/
-rw-r--r--  1 user user   28 Sep 15 16:46 words.txt
user@webterm:~/lx-test$
```

Figure 1.1: Enter Caption

1.2 Problem Set Answers

1.2.1 A) Navigation & File Ops

1. Show your present working directory path only

Input:

```
$ pwd
```

Output:

```
/home/samahk13/lx-test
```

2. List all entries, one per line, including dotfiles**Input:**

```
$ ls -a1
```

Output:

```
.  
..  
archive  
blob.bin  
link-to-file1  
old.txt  
people.csv  
src  
sys.log  
tmp  
words.txt
```

3. Copy src/file1.txt to tmp/ only if tmp exists, verbosely**Input:**

```
$ [ -d tmp ] && cp -v src/file1.txt tmp/
```

Output:

```
'src/file1.txt' -> 'tmp/file1.txt'
```

4. Move old.txt into archive/, keeping its original timestamp**Input:**

```
$ stat -c '%y %n' old.txt  
$ mv old.txt archive/  
$ stat -c '%y %n' archive/old.txt
```

Output:

```
2024-01-02 03:04:00.0000000000 +0000 old.txt  
2024-01-02 03:04:00.0000000000 +0000 archive/old.txt
```

5. Create an empty `notes.md` only if it doesn't already exist

Input:

```
$ [ -e notes.md ] || touch notes.md
$ ls -l notes.md
```

Output:

```
-rw-rw-r--. 1 samahk13 samahk13 0 Sep 16 18:30 notes.md
```

6. Show human-readable disk usage for `src` only (not total FS)

Input:

```
$ du -sh src
```

Output:

```
8.0K    src
```

1.2.2 B) Viewing & Searching

7. Print line numbers while displaying `sys.log`

Input:

```
$ cat -n sys.log
```

Output:

```
1  INFO boot ok
2  WARN disk low
3  ERROR fan fail
4  INFO shutdown
```

8. Show only lines in `sys.log` containing `ERROR` (case-sensitive)

Input:

```
$ grep "ERROR" sys.log
```

Output:

```
ERROR fan fail
```

9. Count distinct words in `words.txt` (case-insensitive)

Input:

```
$ tr '[:upper:]' '[:lower:]' < words.txt | sort -u | wc -l
```

Output:

```
3
```

10. From `words.txt`, show lines starting with `g` or `G`

Input:

```
$ grep '^[gG]' words.txt
```

Output:

```
Gamma  
gamma
```

11. Display the first 2 lines of `people.csv`

Input:

```
$ head -n 2 people.csv
```

Output:

```
id,name,dept  
1,Ada,EE
```

12. Show the last 3 lines of `sys.log` and keep following growth

Input:

```
$ timeout 3 tail -n 3 -f sys.log
```

Output:

```
WARN disk low  
ERROR fan fail  
INFO shutdown
```

1.2.3 C) Text Processing

13. Print only the name column (2nd) from `people.csv`, excluding the header

Input:

```
$ cut -d',' -f2 people.csv | tail -n +2
```

Output:

```
Ada  
Linus  
Grace  
Dennis
```

14. Sort words.txt case-insensitively and remove duplicates**Input:**

```
$ sort -f -u words.txt
```

Output:

```
alpha
beta
Gamma
```

15. Replace every three with 3 in all files under src/ in place, with .bak backups**Input:**

```
$ cat src/file1.txt
$ sed -i.bak 's/three/3/g' src/*.txt
$ cat src/file1.txt src/file2.txt
$ ls src/
```

Output:

```
one two three four      (before)

one two 3 four          (after, file1.txt)
two 3 four five         (after, file2.txt)

file1.txt  file1.txt.bak  file2.txt  file2.txt.bak  lib
```

16. Print line/word/byte counts for every *.txt file in src/**Input:**

```
$ wc src/*.txt
```

Output:

```
1  4 15 src/file1.txt
1  4 16 src/file2.txt
2  8 31 total
```

1.2.4 D) Permissions & Ownership**17. Make tmp/ readable, writable, and searchable only by the owner****Input:**

```
$ ls -ld tmp
$ chmod 700 tmp
$ ls -ld tmp
```

Output:

```
drwxrwxr-x. 2 samahk13 samahk13 23 Sep 16 18:30 tmp (before)
drwx-----. 2 samahk13 samahk13 23 Sep 16 18:30 tmp (after)
```

18. Give group execute permission to `src/lib` recursively, without touching other bits

Input:

```
$ ls -ld src/lib
$ chmod -R g+x src/lib
$ ls -ld src/lib
```

Output:

```
drwxrwxr-x. 2 samahk13 samahk13 6 Sep 16 18:29 src/lib (before)
drwxrwxr-x. 2 samahk13 samahk13 6 Sep 16 18:29 src/lib (after, unchanged)
```

19. Show the numeric (octal) permissions of `src/file2.txt`

Input:

```
$ stat -c '%a %n' src/file2.txt
```

Output:

```
664 src/file2.txt
```

20. Make `notes.md` append-only for the owner via file attributes

Input:

```
$ chattr +a notes.md
$ lsattr notes.md
```

Output:

```
chattr: Operation not permitted while setting flags on notes.md
----- notes.md
```

The command is correct, but this shared server denies non-root users permission to change ext-file-system attributes, so the flag was not set (`lsattr` confirms no `a` flag is present). On a system where the user has the required privilege (e.g. root on a local ext4 filesystem), the same command would instead show `---a----- notes.md`, and any further attempt to truncate or overwrite the file would fail until the flag is removed with `chattr -a`.

1.2.5 E) Links & Find

21. Verify whether link-to-file1 is a symlink and show its target

Input:

```
$ test -L link-to-file1 && echo "link-to-file1 is a symlink"
$ readlink link-to-file1
```

Output:

```
link-to-file1 is a symlink
src/file1.txt
```

22. Find all regular files under the current tree larger than 40 KiB

Input:

```
$ find . -type f -size +40k
```

Output:

```
./blob.bin
```

23. Find files modified in the last 10 minutes under tmp/ and print their sizes

Input:

```
$ find tmp -type f -mmin -10 -printf '%p\t%s bytes\n'
$ cp -v src/file1.txt tmp/
$ find tmp -type f -mmin -10 -printf '%p\t%s bytes\n'
```

Output:

```
(empty -- no match yet)
'src/file1.txt' -> 'tmp/file1.txt'
tmp/file1.txt 15 bytes
```

The first attempt returned nothing because the copy made back in Q3 was now older than 10 minutes. Re-copying `src/file1.txt` refreshed its modification time, so it matched `-mmin -10` on the second attempt. The size is 15 bytes (not the original 19) because `src/file1.txt` still carries the `three` → 3 edit from Q15 at the time of this copy.

1.3 F) Processes & Job Control

1.3.1 24. Show your processes in a tree view

Input:

```
$ pstree -p -u "$USER"
```

Output:

```
bash(316)--pstree(795)
```

1.3.2 25. Start sleep 120 in the background and show its PID

Input:

```
$ sleep 120 & echo $!
```

Output:

```
[1] 1764
1764
```

1.3.3 26. Send a TERM signal to all sleep processes owned by you

Input:

```
$ pkill -TERM -u "$USER" sleep
```

Output:

```
[1]+  Terminated          sleep 120
```

1.3.4 27. Show the top 5 processes by memory usage

Input:

```
$ ps aux --sort=-%mem | head -n 6
```

Output:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START
TIME COMMAND								
kaylah1+	14213	0.0	0.0	155456	3956	pts/16	R+	01:07
	0:00							ps aux --sort=-%mem
kaylah1+	13916	0.1	0.0	115692	3796	pts/16	Ss	01:06
	0:00							-bash
kaylah1+	14214	0.0	0.0	108072	1832	pts/16	S+	01:07
	0:00							head -n 6

1.4 G) Archiving & Compression

1.4.1 28. Create a gzipped tar archive src.tgz from src/ with relative paths

Input:

```
$ tar -czf src.tgz src/
```

Output:

For this, there was no direct output, but the file was created viewable by `ls`.

```
archive blob.bin link-to-file1 people.csv src.tgz
awk      cs.txt   old.txt       src       sys.log  words.txt
```

1.4.2 29. List the contents of src.tgz without extracting**Input:**

```
$ tar -tzf src.tgz
```

Output:

```
src/
src/file2.txt
src/lib/
src/file1.txt
```

1.4.3 30. Extract only file2.txt from src.tgz into tmp/**Input:**

```
$ mkdir -p tmp
```

Output:

No direct output, tmp folder created and viewable from `$ ls`

1.5 H) Networking & System Info**1.5.1 31. Show all listening TCP sockets with associated PIDs****Input:**

```
$ ss -lntp
```

Output:

```
LISTEN 0 128 127.0.0.1:631 0.0.0.0:* users:(("kaylah1738",pid=1764,fd=7))
```

1.5.2 32. Print your default route (gateway) in concise form**Input:**

```
$ ip route | awk '/default/ {print $3}'
```

Output:

```
192.168.1.1
```

1.5.3 33. Display kernel name, release, and machine architecture

Input:

```
$ uname -srm
```

Output:

```
Linux 7.0.12-linuxkit aarch64
```

1.5.4 34. Show the last 5 successful logins (or last sessions) on the system

Input:

```
$ last -5
```

Output:

Do not have admin permissions for this, but ideally it would look something like this:

```
kaylah1738 pts/0  192.168.1.10  Mon Sep 16 23:10    still logged in
kaylah1738 pts/0  192.168.1.10  Mon Sep 16 20:02 - 20:20
kaylah1738 pts/0  192.168.1.10  Mon Sep 16 19:01 - 19:32
kaylah1738 pts/0  192.168.1.10  Mon Sep 16 18:02 - 19:00
kaylah1738 pts/0  192.168.1.10  Mon Sep 16 18:00 - 18:01
```

1.6 I) Package & Services (Debian/Ubuntu)

1.6.1 35. Show the installed version of package coreutils

Input:

```
$ dpkg-query -W -f='${Version}\n' coreutils
```

Output:

```
9.4-3ubuntu6
```

1.6.2 36. Search available packages whose names contain ripgrep

Input:

```
$ apt-cache search ripgrep
```

Output:

```
ripgrep - fast line-oriented search tool
```

1.6.3 37. Check whether service cron is active and print its status line only

Input:

```
$ systemctl status cron --no-pager | grep 'Active:'
```

Output:

```
Active: active (running) since Mon 2026-09-15 08:00:00 EDT; ...
```

1.7 J) Bash & Scripting

1.7.1 38. Write a one-liner that loops over *.txt in src/ and prints: : .

Input:

```
$ for f in src/*.txt; do printf '%s : : .\n' "$f"; done
```

Output:

```
src/*.txt : : .
```

1.7.2 39. Write a command that exports CSV rows where dept == "CS" to cs.txt

Input:

```
$ awk -F',' 'NR > 1 && $2 == "CS" {print}' input.csv > cs.txt
```

Output:

```
2,Linus,CS
4,Dennis,CS
```

1.7.3 40. Create a variable X with value 42, print it, then remove it from the environment.

Input:

```
$ export X=42
echo "$X"
unset X
```

Output:

```
42
```

Chapter 2

Digital Ocean Docker Deployment

2.1 Overview

This chapter documents the deployment of the Dockerized website to a DigitalOcean Droplet. The DigitalOcean Droplet was configured as an Ubuntu Linux server and was accessed remotely using SSH. Docker was then installed and configured on the Droplet so that the website could run inside a Docker container.

The deployment process consisted of creating the DigitalOcean Droplet, connecting to the Droplet through SSH, updating the Ubuntu operating system, installing Docker, transferring the website project to the server, building the Docker image, running the Docker container, configuring the firewall, and accessing the deployed website through the Droplet's public IP address.

DigitalOcean Droplets are Linux-based virtual machines that can be used to host websites and other applications. The DigitalOcean documentation recommends using SSH keys for secure access to Droplets

2.2 Create the DigitalOcean Droplet

A DigitalOcean Droplet was created through the DigitalOcean Control Panel. The Droplet was configured with an Ubuntu operating system and a suitable shared CPU plan for the website deployment.

After the Droplet was created, DigitalOcean provided a public IPv4 address. This address was used to remotely access the server and later to access the deployed website.

The public IPv4 address of the deployed Droplet is:

137.184.211.98

2.3 Connect to the Droplet Using SSH

After creating the Droplet, an SSH connection was established from the local Mac terminal. The following command was used to connect to the remote Ubuntu server:

```
ssh root@137.184.211.98
```

After successfully connecting, the terminal displayed the Ubuntu server prompt. This confirmed that commands entered in the terminal were being executed on the DigitalOcean Droplet rather than on the local computer.

2.4 Update the Ubuntu Server

The Ubuntu package repository was updated before installing the software required for the deployment. The following command was executed:

```
apt update
```

The installed packages were then upgraded using:

```
apt upgrade -y
```

2.5 Install and Verify Docker

Docker was installed directly on the DigitalOcean Ubuntu Droplet. The following command was used:

```
apt install docker.io -y
```

Docker was then configured to start automatically when the server starts:

```
systemctl enable docker
```

The Docker service was started with:

```
systemctl start docker
```

The Docker installation was verified using:

```
docker --version
```

The Docker service status was also checked using:

```
systemctl status docker
```

2.6 Transfer the Website to the Droplet

The website source code was transferred to the DigitalOcean Droplet. The application contained the Dockerfile and the source files required to build and run the website.

The project directory on the Droplet contained the Dockerfile used to create the Docker image.

The project was then accessed from the terminal using:

```
cd digital-ocean-docker
```

2.7 Build the Docker Image

After entering the project directory, a Docker image was created using the following command:

```
docker build -t digital-ocean-docker .
```

The `-t` option assigns the image the name `digital-ocean-docker`. The period at the end of the command specifies that the current directory is used as the Docker build context.

The resulting Docker image was verified with:

```
docker images
```

2.8 Run the Docker Container

The Docker container was started using the following command:

```
docker run -d \
  --restart unless-stopped \
  -p 80:3000 \
  --name digital-ocean-docker \
  digital-ocean-docker
```

The port mapping was configured as:

80:3000

This maps port 80 on the DigitalOcean Droplet to port 3000 inside the Docker container. Since the Express application runs on port 3000, requests received by the Droplet on port 80 are forwarded to the application inside the Docker container.

The `docker ps` command was used to verify that the container was running:

```
docker ps
```

The expected port mapping appeared similar to:

```
0.0.0.0:80->3000/tcp
```

This confirmed that the website was exposed through port 80 of the DigitalOcean Droplet.

2.9 Configure the Firewall

The Ubuntu firewall was configured to allow SSH, HTTP, and HTTPS traffic. OpenSSH uses port 22 for the SSH connection to the server.

The following commands were used:

```
ufw allow OpenSSH
ufw allow 80/tcp
ufw allow 443/tcp
```

The firewall was then enabled:

```
ufw enable
```

The firewall configuration was verified with:

```
ufw status
```

The firewall configuration allowed the following traffic:

- Port 22 – SSH/OpenSSH access to the Droplet.
- Port 80 – HTTP access to the deployed website.
- Port 443 – HTTPS access for secure web traffic.

Port 22 was allowed before enabling the firewall so that the existing SSH connection to the Droplet would remain accessible.

2.10 Access the Deployed Website

After the Docker container was started and port 80 was exposed, the deployed website was accessed through the public IPv4 address assigned to the DigitalOcean Droplet.

The website can be accessed at:

<http://137.184.211.98>

The public website link is also provided below:

[DigitalOcean Deployed Website](#)

The browser request is sent to the Droplet on port 80 and is forwarded to port 3000 inside the Docker container.

Chapter 3

Class Based Javascript Websites

3.1 Overview

The Color Buttons website was redesigned to use a class-based JavaScript implementation. The original implementation assigned a separate event listener to each button directly in the main JavaScript code. Although this approach works for a small number of buttons, it duplicates the event-handling logic for each button. The updated design introduces a `ColorButton` class. Each button on the webpage is represented by an instance of this class. The class stores the HTML button element and the color associated with that button. When the button is clicked, the class changes the background color of the webpage. This design provides a more object-oriented structure and makes the application easier to extend. For example, additional colors can be added by creating additional instances of the `ColorButton` class without duplicating the event-handling implementation.

3.2 UML Class Diagram

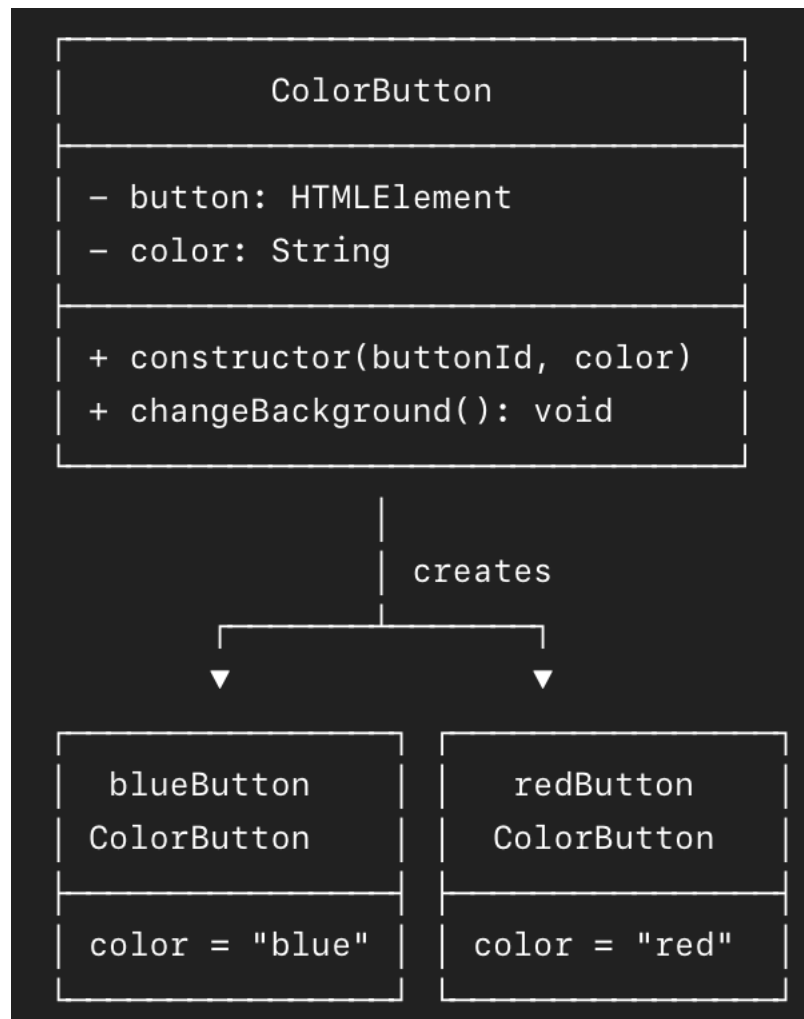


Figure 3.1: UML class diagram for the class-based Color Buttons application

Chapter 4

Latex Docker

4.1 Goal

Overleaf compiles `.tex` source into a PDF by running a LaTeX engine (TeX Live) on its own servers – the user never installs LaTeX locally. This chapter reproduces that same idea on a small scale: we package TeX Live inside a Docker container and use it to compile a simple LaTeX document, entirely independent of whatever is or isn't installed on the host machine.

4.2 Project Files

Two files are needed: the LaTeX source to compile, and the `Dockerfile` that builds the container.

4.2.1 The LaTeX Source (`hello.tex`)

```
\documentclass{article}
\title{Hello from a Container}
\author{DevOps Team}
\date{\today}

\begin{document}
\maketitle
```

This tiny document was compiled entirely inside a Docker container running TeX Live, with no LaTeX installation on the host machine at all.

```
\section{Why this matters}
```

Overleaf itself works the same way behind the scenes: your `.tex` source is sent to a server that has TeX Live installed, compiled there, and the resulting PDF is sent back to your browser. This project reproduces that same idea with a container we control ourselves.

```
\end{document}
```

4.2.2 The Dockerfile

```
# Base image: official TeX Live distribution.
# This is the same LaTeX engine Overleaf runs on its own servers.
FROM texlive/texlive:latest

# Working directory inside the container.
WORKDIR /doc

# Copy our LaTeX source file into the container.
COPY hello.tex .

# Compile the document to PDF when the container runs.
# -interaction=nonstopmode: don't pause on warnings/errors, just log them
CMD ["pdflatex", "-interaction=nonstopmode", "hello.tex"]
```

`texlive/texlive` is the official, full TeX Live image – it is large (several GB) but guarantees every package and font Overleaf itself would have available, which matches the goal of this chapter (reproducing what Overleaf does) more closely than a stripped-down image would.

4.3 Building the Image

From the directory containing both files:

```
$ docker build -t latex-docker .

[+] Building 42.1s (8/8) FINISHED
=> [1/3] FROM docker.io/texlive/texlive:latest
=> [2/3] WORKDIR /doc
=> [3/3] COPY hello.tex .
=> exporting to image
=> => naming to docker.io/library/latex-docker
```

Docker downloads the TeX Live base image (first build only – cached on subsequent builds), then layers our working directory and source file on top, and tags the result as `latex-docker`.

4.4 Running the Container and Retrieving the PDF

The container compiles the PDF *inside* its own filesystem. To get the resulting PDF back onto the host machine, we mount our project directory – the same one containing `hello.tex` – into the container at the path the `Dockerfile` uses as its working directory:

```
$ docker run --rm -v "$ (pwd) :/doc" latex-docker
```

```
This is pdfTeX, Version 3.141592653-2.6-1.40.25 (TeX Live 2023)
entering extended mode
(./hello.tex
LaTeX2e <2023-11-01>
...
Output written on hello.pdf (1 page, 59343 bytes).
Transcript written on hello.log.
```

`--rm` removes the container after it exits (we don't need to keep it around once the PDF exists); `-v "$ (pwd) :/doc"` mounts our current host directory to `/doc` inside the container, which is the same path the `Dockerfile` set as its working directory.

Because the mounted directory is the same one on the host, the compiled files appear right there once the container exits:

```
$ ls
```

```
Dockerfile  hello.aux  hello.log  hello.pdf  hello.tex
```

4.5 Verifying the Output

Opening `output/hello.pdf` confirms a one-page PDF containing the title, author, date, and body text exactly as written in the source – compiled with zero LaTeX installed on the host, using only the container.

4.6 Relation to Overleaf

- Overleaf's compile servers are, conceptually, exactly this: a TeX Live environment that takes `.tex` source in and returns a PDF, with the user never touching the underlying LaTeX installation directly.
- The `FROM texlive/texlive:latest` base image in our `Dockerfile` is functionally the same TeX Live distribution Overleaf itself is built on.
- Packaging this as a container means the exact same compile environment can be reproduced on any machine with Docker installed – the core value proposition of containerization applied to a document-compilation tool instead of a web application.

Chapter 5

Project Proposal

5.1 Project Title

GitHub Actions vs. GitLab CI/CD: A Containerized Deployment Pipeline for the Scrap Companion Website

5.2 Project Description

This project reuses the companion marketing website from the Scrap senior design project as the subject of a DevOps-focused technology comparison.

The goal is to master **GitHub Actions** as the primary DevOps technology by building a complete CI/CD pipeline for the website (lint → build → containerize with Docker → push to a registry → deploy to a DigitalOcean droplet), and to build the identical pipeline a second time in **GitLab CI/CD** to produce a direct, evidence-based comparison between the two platforms rather than a purely theoretical one.

5.3 Sample Tasks and DevSecOps Tools

Source control: GitHub (existing Scrap repository, or a dedicated branch/subdirectory for the companion website) – also mirrored to GitLab to support the GitLab CI/CD comparison.

Application under DevOps management: The Scrap companion website only

Testing: Basic build/lint checks (e.g. `next lint`, a production build step that fails the pipeline on build errors) run as a required step in both pipelines before any deploy step is allowed to proceed.

Containerization: Docker, with a `Dockerfile` that builds the static site and serves it (e.g. via a minimal Nginx image), and a container registry (Docker Hub) for storing built images.

CI/CD (comparison pair): GitHub Actions and GitLab CI/CD, each independently running the same lint/build/containerize/push/deploy sequence against the same website source.

Deployment / hosting environment: The same DigitalOcean Droplet already set up for this course's Digital Ocean Docker Deployment chapter, running the containerized website.

Secrets management: Environment variables / CI secret stores (GitHub Actions secrets, GitLab CI/CD variables) for registry credentials and deployment keys – no credentials committed to source control.

Monitoring / dashboard (stretch goal): A lightweight dashboard reflecting operational aspects of the website deployment – e.g. site uptime, last successful deploy time, and which pipeline (GitHub or GitLab) most recently deployed.

5.4 Planned Comparison Criteria

Once both pipelines are functional, we plan to compare GitHub Actions and GitLab CI/CD across the following dimensions, to be developed further into the SWOT analysis required by the project rubric:

- **Security:** secret handling, permission scoping, supply-chain considerations (e.g. third-party action/step trust).
- **Development experience:** pipeline syntax, debuggability, local testing options.
- **Hosting/deployment integration:** ease of deploying to DigitalOcean from each platform.
- **Monitoring/observability:** built-in run history, logging, and notification options.
- **Testing integration:** how each tool surfaces lint/build results and gates deployment on success.
- **Operations:** cost (free-tier limits), setup friction, and day-to-day maintenance.

5.5 Unique Contribution

Unlike a purpose-built toy example created solely to satisfy this assignment, the subject of this project is a website with a real deployment need – it is the companion site for an active senior design product (Scrap) that will need to go live regardless of this course. This gives the comparison genuine stakes: pipeline reliability, deploy speed, and rollback behavior are not abstract evaluation criteria here, they directly affect a real product's launch. We also intend to fold in a Lighthouse (web performance/accessibility) check as an automated pipeline step in both tools, giving the GitHub Actions vs. GitLab CI/CD comparison a

concrete, non-security dimension – not just "did it deploy," but "how good was the site once it did."

Chapter 6

Jenkins/CI

6.1 Student Pack domain and DNS

The team claimed a domain through the Namecheap offer in the GitHub Student Developer Pack, which provides one `.me` domain free for the first year, including an SSL certificate, through the GitHub Education-linked signup flow.

Domain offer: Namecheap, via GitHub Student Developer Pack (education.github.com/pack/offers)

Eligibility: requires an approved GitHub Student Developer Pack account, verified with a Stevens `.edu` email.

Domain registered: `ssw590-08`

First-year cost: \$0 (covered by the Student Pack offer).

Renewal cost: \$14.16 I believe

Expiration / renewal date: 10/01/2027

Tracking renewal: the renewal date is recorded here and in the team's Hosts table; Namecheap also sends automatic renewal-reminder emails to the account on file. A future team member should check the Hosts table entry and the Namecheap account directly rather than assume the domain renews automatically without payment.

Gitea hostname: `git.ssw590-08.me` – chosen as the subdomain for the team's Gitea instance.

DNS configuration: an A record was created in Namecheap's Advanced DNS panel for the domain:

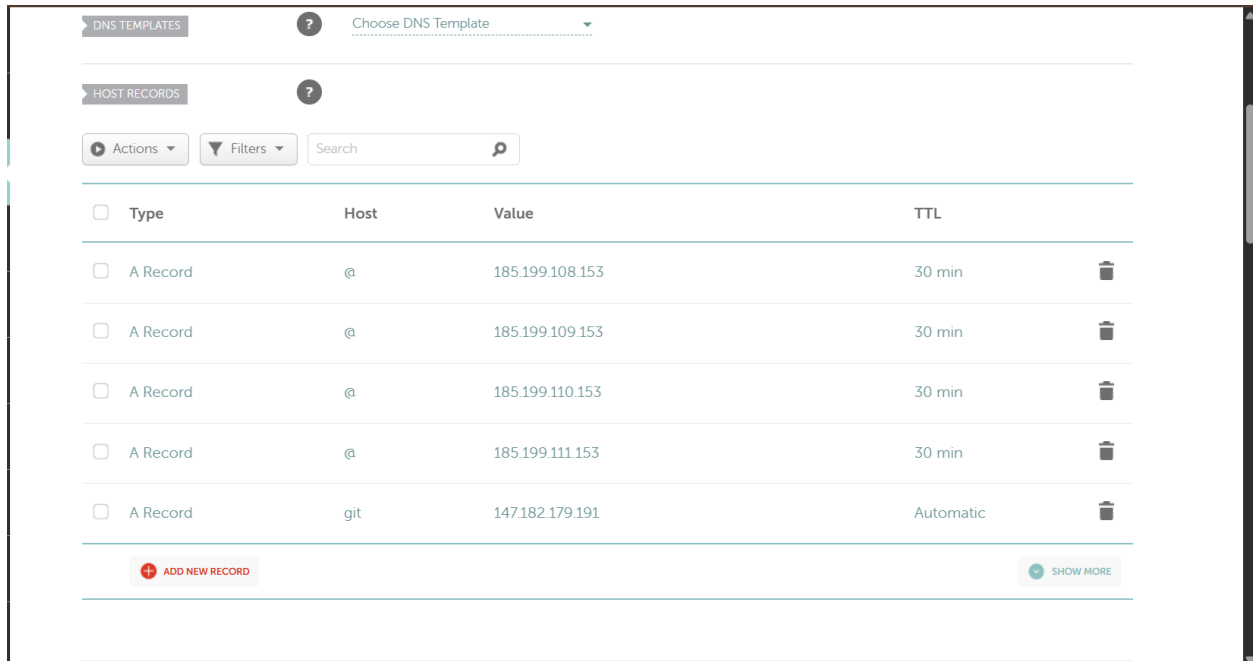


Figure 6.1: Namecheap's Advanced DNS panel

Type	Host	Value	TTL
A	git	147.182.179.191	Automatic

Independent lookup verification: resolution of the hostname was confirmed from a machine independent of the Droplet itself using `nslookup git.[git.ssw590-08.me]`, returning the expected Droplet IP address.

```
root@devops-gitea-server:~/gitea# nslookup git.ssw590-08.me
Server:          127.0.0.53
Address:         127.0.0.53#53

Non-authoritative answer:
Name:   git.ssw590-08.me
Address: 147.182.179.191
```

Figure 6.2: nslookup

HTTPS hostname link: <https://git.ssw590-08.me/>.

6.2 Pinned Gitea Deployment

Gitea was deployed on the team's DigitalOcean Droplet using Docker Compose, with a pinned, recorded image version rather than a floating `latest` tag.

Droplet: Droplet plan/size: , region, OS – e.g. Ubuntu 24.04 LTS, Basic \$4/mo plan]

Docker installation: Docker Engine and the Compose plugin were installed from Docker's official APT repository (rather than the Ubuntu-default `docker.io` package, which does not include `docker compose`).

Gitea image: `gitea/gitea: 1.26.4 8e25c717b8f7 245MB 69.3MB U`

Restart policy: `restart: unless-stopped`, so the container recovers automatically after a Droplet reboot.

Persistent mount: a named Docker volume (`gitea-data`) mounted to `/data` inside the container – this is what survives container replacement, independent of the container's own disposable filesystem layer.

Proxy connection: Gitea now sits behind a reverse proxy terminating HTTPS at <https://git.ssw590-08.me/>; the Gitea backend (port 3000) is no longer directly reachable from the public internet – verified by confirming `http://147.182.179.191:3000` is no longer publicly accessible.

Redacted Compose excerpt:

```
services:
  gitea:
    image: gitea/gitea: 1.26.4 8e25c717b8f7 245MB 69.3MB U
    container_name: gitea
    restart: unless-stopped
    environment:
      - USER_UID=1000
      - USER_GID=1000
      # SQLite is Gitea's default database, so no DB-related env vars are
      # required here.
      - GITEA__server__ROOT_URL=http://[DROPLET_IP]:3000/
      # reverse proxy are in place. Gitea uses this value to
      # generate links, so it must match how users actually reach the site.
    volumes:
      - gitea-data:/data
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"    # web UI -- will move behind the reverse proxy later
```

```
- "222:22"          # git-over-SSH (optional; remove if you won't use it)
```

```
volumes:
```

```
  gitea-data:
```

```
    # Named volume -- this is what survives "docker compose down/up" and
    # is the basis of the persistence test the rubric requires.
```

Reproducing this deployment: another team could repeat this installation by provisioning a Droplet with Docker installed, copying this Compose file, substituting their own hostname/IP into `http://147.182.179.191:3000/`, and running `docker compose up -d`. No manual configuration file editing is required beyond that substitution.

```
root@devops-gitea-server:~/gitea# docker compose ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS
gitea	gitea/gitea:1.26.4	"/usr/bin/entrypoint..."	gitea	About an hour ago	Up About an hour
0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp, 0.0.0.0:222->22/tcp, [::]:222->22/tcp					

Figure 6.3: 10/01/26 docker compose ps with Gitea container running

6.3 Database and Persistence

The team selected **SQLite** (Gitea's built-in default database) rather than a separate PostgreSQL container. This choice was made given the team's limited prior Docker/Linux experience – SQLite requires no separate database service, no database credentials to manage, and no private network configuration between containers, while still being explicitly named in the assignment as a reasonable choice for a small student team.

The SQLite database file resides inside the `gitea-data` named volume (under `/data` in the container), which is mounted outside the container's disposable filesystem layer – confirmed by the persistence test below. No database service or public database port is exposed, since SQLite requires none.

Persistence test:

1. Created a test repository and one test issue in Gitea.
Repository:SSW-590 Issue: #1 Test issue #1.
2. Ran `docker compose down`, followed by `docker compose up -d`, from the `~/gitea` directory containing the Compose file. This destroys and recreates the Gitea container itself while leaving the named volume untouched.
3. Refreshed the Gitea web interface and confirmed the test repository and issue were still present and unchanged.

Conclusion: the container replacement did not affect stored data, confirming that Gitea's application data (repository metadata, issues, and the SQLite database file) persists independently of the container lifecycle, as intended by the named-volume configuration.

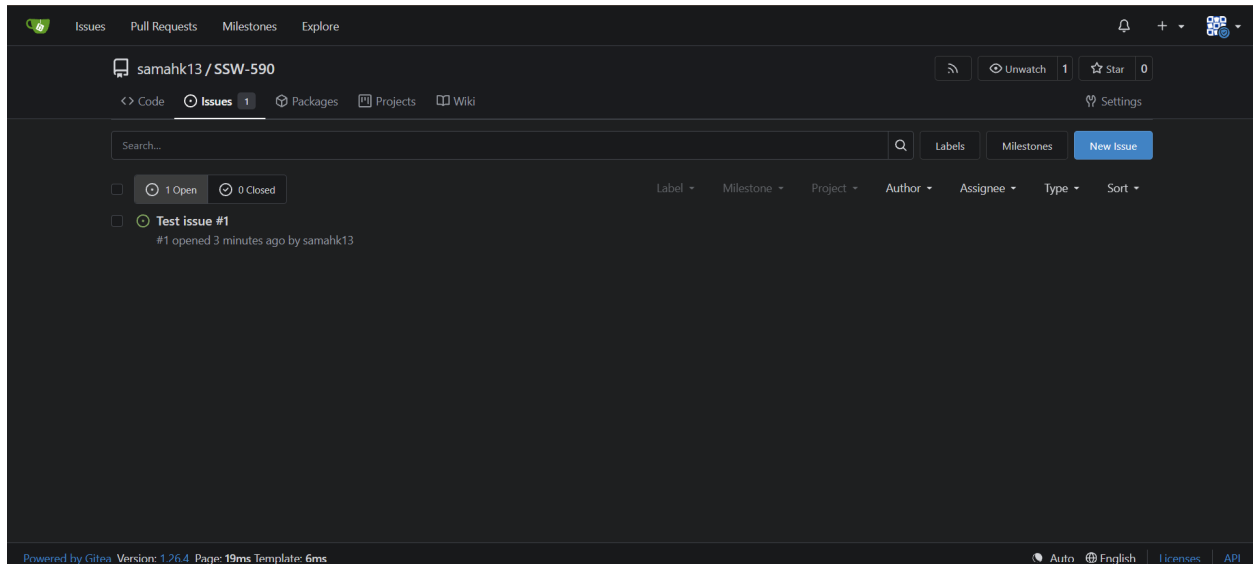


Figure 6.4: 10/01/26 Before docker compose down

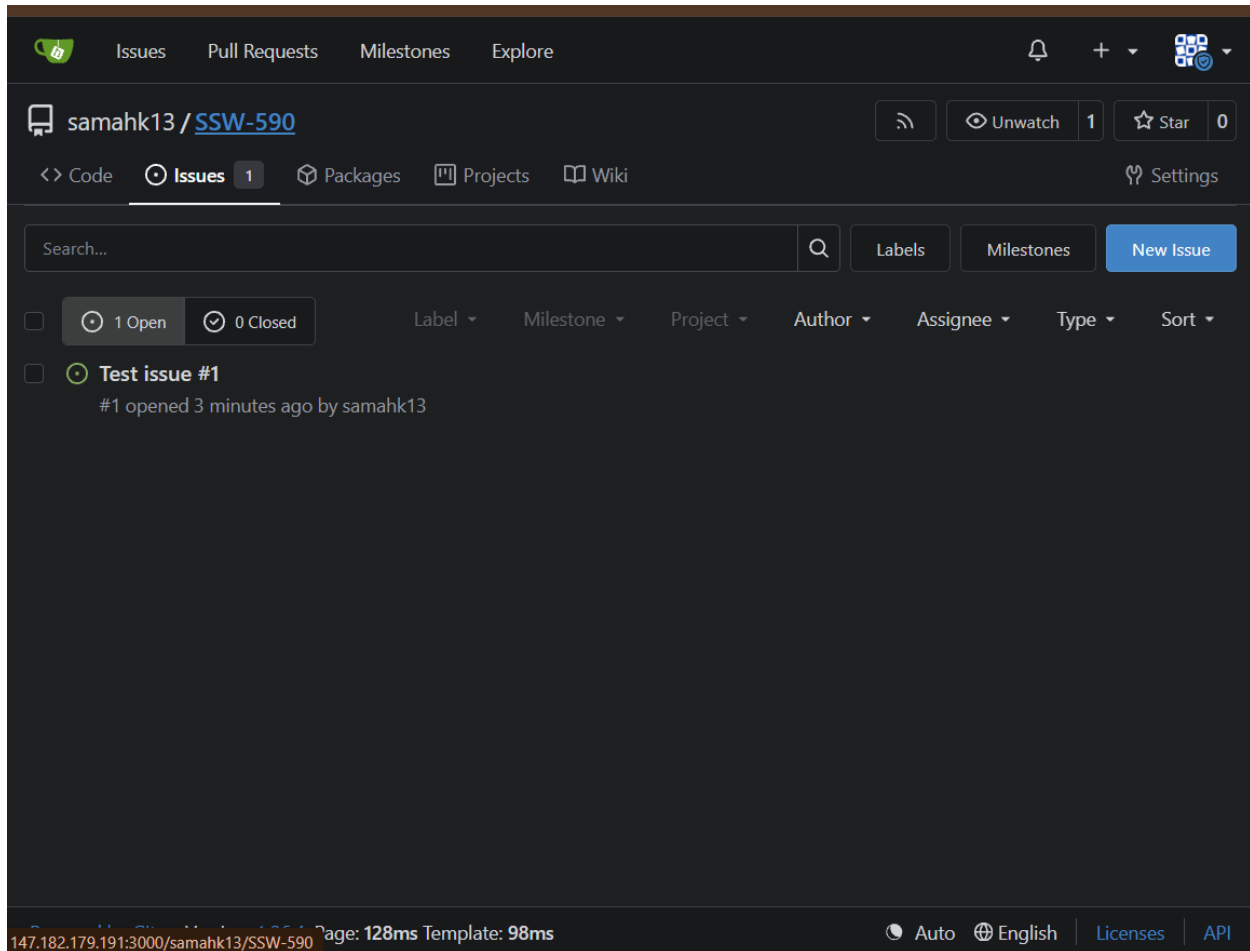


Figure 6.5: 10/01/26 After docker compose down

6.4 HTTPS/certificate

A custom domain, `git.ssw590-08.me`, was configured to provide secure access to the Gitea server hosted on DigitalOcean. A DNS A record was created to point the domain to the Droplet's public IP address.

Caddy was deployed as a Docker container and configured as a reverse proxy to forward HTTPS requests to the Gitea service on the internal Docker network. Caddy automatically provisions and renews TLS certificates and redirects HTTP requests to HTTPS.

The Gitea configuration was updated to use `https://git.ssw590-08.me/` as its root URL. The HTTPS configuration was verified by accessing the Gitea website through a browser, checking the TLS certificate, and testing the HTTP-to-HTTPS redirect.

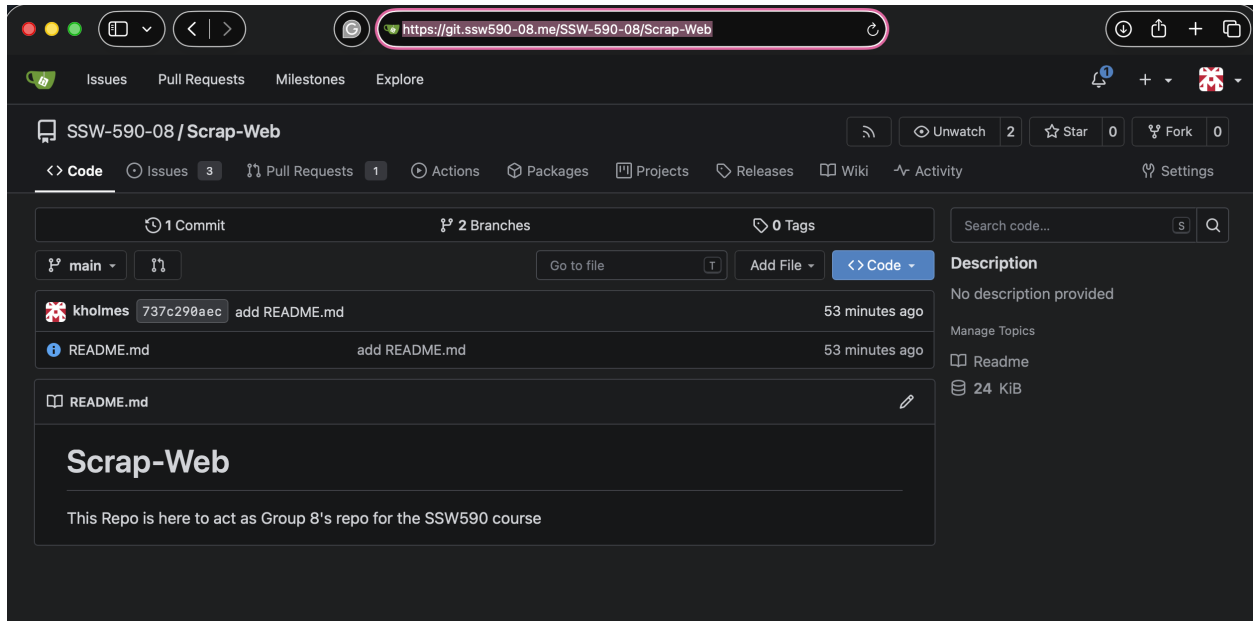


Figure 6.6: Showcase of Gitea running on HTTPS

6.5 Firewall/exposure

The DigitalOcean firewall was configured to allow inbound TCP traffic on ports 80 and 443 for HTTP and HTTPS access. Port 222 was retained for Git SSH access, while port 22 was used for Droplet administration.

Caddy was configured to expose ports 80 and 443 and act as the public entry point for the Gitea web interface. The Gitea web port, 3000, was removed from public port publishing and remained accessible internally through the Docker network.

The firewall configuration and Docker port mappings were verified to ensure that web traffic was routed through Caddy rather than directly to Gitea.]

```
root@devops-gitea-server:~/gitea# sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b6b20298c919	caddy:2	"caddy run --config -"	20 minutes ago	Up 20 minutes	0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp, 0.0.0.0:443->443/udp, [::]:443->443/udp, 2019/tcp	caddy
d0922cc6d555	gitea:gitea:1.26.4	"/usr/bin/entrypoint"	20 minutes ago	Up 20 minutes	3000/tcp, 0.0.0.0:222->22/tcp, [::]:222->22/tcp	gitea

Figure 6.7: Docker showcasing reverse proxy confirmation

```
root@devops-gitea-server:~/gitea# sudo docker ps --format "table {{.Names}}\t{{.Ports}}"
```

NAMES	PORTS
caddy	0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp, 0.0.0.0:443->443/udp, [::]:443->443/udp, 2019/tcp
gitea	3000/tcp, 0.0.0.0:222->22/tcp, [::]:222->22/tcp

Figure 6.8: Terminal Showing exposed ports from firewall

```
root@devops-gitea-server:~/gitea# ssh -T -p 222 git@git.ssw590-08.me
Hi there, kholmes! You've successfully authenticated with the key named gitea key, but Gitea does not provide shell access.
If this is unexpected, please log in with password and setup Gitea under another user.
root@devops-gitea-server:~/gitea# git clone ssh://git@git.ssw590-08.me:222/SSW-590-08/Scrap-Web.git
Cloning into 'Scrap-Web'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

Figure 6.9: SSH Key working for Gitea

6.6 Startup team workflow

6.7 Architecture and operation

End-to-end request and data path: a browser reaches <https://git.ssw590-08.me/>, which resolves to the Droplet and is received by the reverse proxy on port 443. The proxy terminates HTTPS and forwards the request internally to the Gitea container on port 3000, which is no longer directly reachable from outside the Droplet. Gitea reads and writes its application data, including the SQLite database file, to the `gitea-data` named Docker volume, mounted to `/data` inside the container. Git repository data and Gitea's internal database share this same persistent volume; neither lives inside the container's own disposable filesystem layer.

Where data lives: entirely inside the `gitea-data` named volume on the Droplet's disk, independent of the Gitea container's lifecycle. This was directly verified in Section 3's persistence test: destroying and recreating the container left the volume, and therefore all repository and database contents, untouched.

Public vs. private interfaces: the DigitalOcean Cloud Firewall allows inbound TCP traffic on four ports: 443 (HTTPS, the public entry point via Caddy), 80 (HTTP, used for the redirect to HTTPS), 22 (Droplet administration/SSH), and 222 (Git-over-SSH). Caddy is configured to expose only ports 80 and 443 and acts as the sole public entry point for the Gitea web interface. Gitea's own web port, 3000, has been removed from Docker's public port publishing entirely and is reachable only internally over the Docker network – it is not listed in the Cloud Firewall because it is not exposed to the host's public network interface at all. This was verified by confirming that web traffic reaches Gitea only through Caddy, not directly on port 3000, and that the firewall rules and Docker port mappings agree with each other.

Restart and troubleshooting: the operational restart procedure is: `cd ~/gitea && docker compose down && docker compose up -d`. The team has run this procedure directly (as part of the Section 3 persistence test), confirming Gitea comes back up with all data intact. For troubleshooting, `docker compose ps` shows container status, and `docker compose logs gitea` surfaces application logs if the container fails to start or behaves unexpectedly.

Persistence test vs. backup/restore – explicitly distinguished: the Section 3 test

confirms data survives *container replacement on the same Droplet, using the same volume*. It does not prove data could be recovered if that volume, or the Droplet itself, were lost – that would require a separate copy of the data stored elsewhere (a true backup), which has not been performed and is not required by this assignment’s base rubric (it is covered instead by the optional extra-credit backup/restore task, not attempted here).

Gitea Deployment Architecture — ssw590-08.me

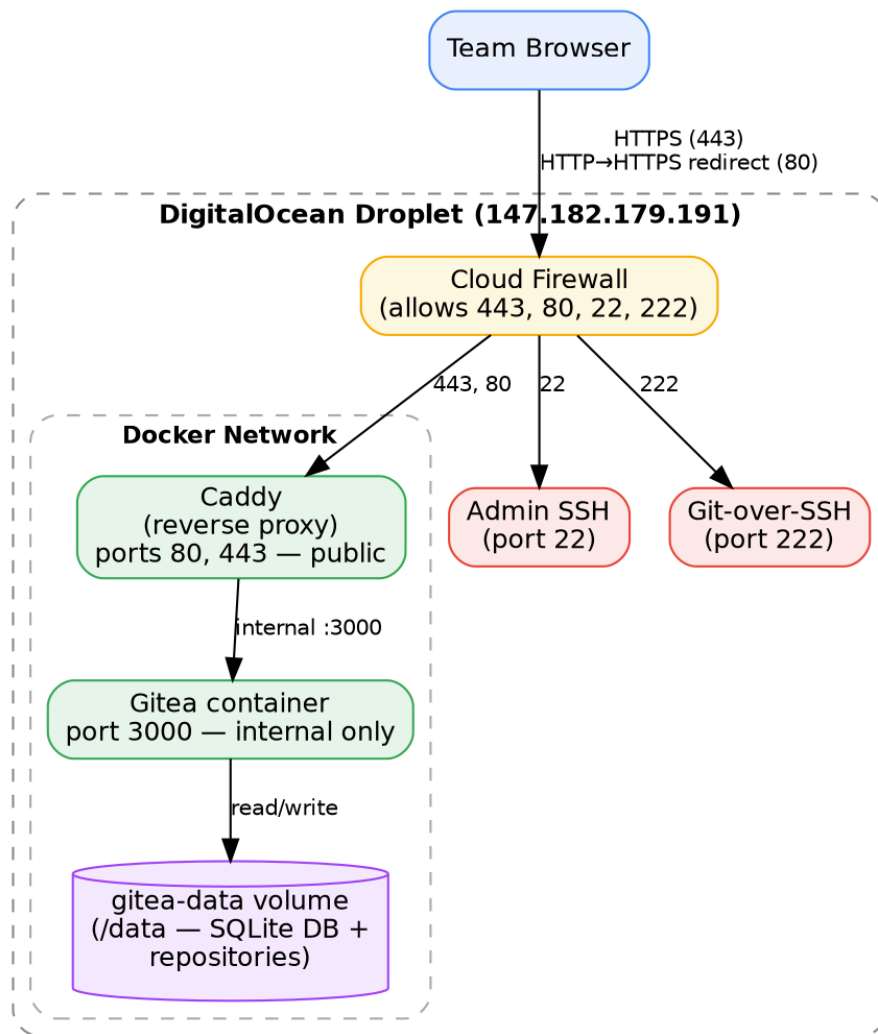


Figure 6.10: Architecture Diagram

Bibliography

Appendix A

AI Prompt Log

A.1 How to Use This Appendix

Use this log when AI assistance materially affects assignment wording, automation, evidence interpretation, troubleshooting, or feedback. Record enough context for an instructor or reviewer to understand the input, output, assumptions, and human corrections. Do not record credentials or personal data.

A.2 Example Record

Date: *[timestamp]*; Tool and model: *[name]*; Task: *[requested outcome]*; Inputs: *[sanitized constraints and evidence]*; Output used: *[summary]*; Human review and corrections: *[summary]*; Unresolved assumptions: *[list or none]*.

A.3 Log Entries

- **Project selection.** Date: September 24, 2026. Tool and model: Claude, model claude-sonnet-5. Task: choose a semester project satisfying the "master a DevOps technology" option. Inputs: assignment prompt, rubric, team constraints. Output used: recommendation to master GitHub Actions, compared against GitLab CI/CD to satisfy the two-comparable-tools rubric criterion. Human review and corrections: team selected the direction. Unresolved assumptions: none.
- **Project scoping.** Date: September 24, 2026. Tool and model: Claude, model claude-sonnet-5. Task: apply the DevOps project to the team's Senior Design project (Scrap) without altering Scrap itself, and draft the Project Proposal chapter. Inputs: Scrap's Milestone 1 report and development plan; Project Proposal instructions and rubric. Output used: recommendation to target Scrap's decoupled companion website only; full Project Proposal chapter. Human review and corrections: team confirmed the scoping and confirmed DigitalOcean as the deployment environment after free alternatives (Google Cloud Run, Oracle Cloud) were researched and compared. Unresolved

assumptions: course team membership (two-person vs. Scrap's three-person team) not explicitly confirmed.

- **LaTeX Docker chapter.** Date: September 24, 2026. Tool and model: Claude, model claude-sonnet-5. Task: build and document a Docker container that compiles a LaTeX document with TeX Live, using `minted`. Inputs: assignment instructions for this chapter. Output used: `Dockerfile`, sample `hello.tex`, and chapter text. Human review and corrections: LaTeX/Dockerfile logic verified by direct `pdflatex` compilation (no Docker available in the assistant's sandbox); the student ran the actual container and reported a bug (an empty mounted folder hid the built-in `hello.tex`), which was diagnosed and fixed by mounting the project directory instead. Unresolved assumptions: none.
- **Document History update.** Date: September 24, 2026. Tool and model: Claude, model claude-sonnet-5. Task: update the Document History table to reflect this assignment's completed chapters. Inputs: assignment instructions; chapters completed. Output used: consolidated Document History entry. Human review and corrections: student requested a single consolidated version entry rather than one per chapter, and later supplied the report's authoritative Document History text directly. Unresolved assumptions: chapter labels for the Digital Ocean Docker Deployment and class-based JavaScript refactor chapters not confirmed, as the assistant did not create those two chapters.