

SSW 590

Version (CL , SHA ,)

by

Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu

glam1@Stevens.edu, lxu41@stevens.edu, ajain72@stevens.edu

October 25, 2025

© Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu
glam1@Stevens.edu, lxu41@stevens.edu, ajain72@stevens.edu
ALL RIGHTS RESERVED

SSW 590
Version (CL , SHA ,)

Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu
glam1@Stevens.edu, lxu41@stevens.edu, ajain72@stevens.edu

This document provides the requirements and design details of the PROJECT. The following table (Table 1) should be updated by authors whenever major changes are made to the architecture design or new components are added. Add updates to the top of the table. Most recent changes to the document should be seen first and the oldest last.

Table 1: Document Update History

Date	Updates
10/23/2025	Annanya: <ul style="list-style-type: none">• Configured and Documented setup of Prometheus with Grafana(11)
10/22/2025	Gavin and Luo: <ul style="list-style-type: none">• Updated and documented the Overleaf Chapter with information on github actions(10)
10/16/2025	Annanya: <ul style="list-style-type: none">• Documented the creation of custom domain and obtained SSL Certificate for overleaf(10)
10/11/2025	Luo: <ul style="list-style-type: none">• Updated Overleaf Chapter to include all packages Overleaf requires (10)
10/05/2025	Annanya: <ul style="list-style-type: none">• Updated and documented the Overleaf Chapter(10)
10/05/2025	Gavin: <ul style="list-style-type: none">• Updated the Hosts and Password tables (2)(1)
10/03/2025	Luo: <ul style="list-style-type: none">• Updated and documented the Bugzilla Chapter (8)
10/02/2025	Luo: <ul style="list-style-type: none">• Created the Bugzilla (8)
10/01/2025	Annanya: <ul style="list-style-type: none">• Added to chapter LaTeX Docker regarding the steps taken to create Docker container which compiles a simple latex file. (7)
09/29/2025	Gavin: <ul style="list-style-type: none">• Added to chapter AWS the steps we took to get our website deployed. (6)

Table 1: Document Update History

Date	Updates
09/24/2025	Luo, Ananaya, Gavin: <ul style="list-style-type: none"> Improved and added more to the Project Proposal in terms of tools, devop tools, and description. (5) Created the AWS Deployment file. (6)
09/21/2025	Luo: <ul style="list-style-type: none"> Created the Project Proposal (5) Spurthi: <ul style="list-style-type: none"> The Betrayal (she switched teams)
09/15/2025	Gavin, Spurthi, Annanya, Luo: <ul style="list-style-type: none"> Completed remaining Linux Problem Sets in Linux Commands chapter (4)
09/14/2025	Gavin, Spurthi, Annanya, Luo: <ul style="list-style-type: none"> Added new chapter, Linux Commands. (4) Completed terminal session and solved A-D in the problem set (4)
09/9/2025	Gavin, Spurthi, Annanya, Luo: <ul style="list-style-type: none"> Updated Passwords chapter (Chapter 1) with long table with user/password/server rules. Created Hosts chapter (Chapter 2) and Kanban Setup (Chapter 3).

Table of Contents

1	Passwords	
	– <i>Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu</i>	1
2	Hosts	
	– <i>Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu</i>	2
3	Kanban Setup	
	– <i>Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu</i>	3
4	Linux Commands	
	– <i>Spurthi Setty, Gavin Lam, Annanya Jain, Luo Xu</i>	7
4.1	Terminal Session	7
4.2	Problem-Set Commands and Outputs	7
4.2.1	Navigation & File Ops	7
4.2.2	Viewing & Searching	8
4.2.3	Text Processing	9
4.2.4	Permissions & Ownership	10
4.2.5	Links & Find	10
4.2.6	Processes & Job Control	10
4.2.7	Archiving & Compression	11
4.2.8	Networking & System Info	12
4.2.9	Package & Services (Debian/Ubuntu)	12
4.2.10	Bash & Scripting	13
5	Project Proposal	
	– <i>Luo Xu, Annanya Jain, Gavin Lam</i>	15
6	AWS	
	– <i>Gavin Lam, Luo Xu, Annanya Jain</i>	16
7	LaTeX Docker	
	– <i>Annanya Jain, Gavin Lam, Luo Xu</i>	24
7.1	Steps taken for creating the docker container to compile a LaTeX File	24

7.1.1	main.tex	24
7.1.2	Dockerfile	25
7.1.3	Building the Docker image	25
8	Bugzilla	
	– <i>Luo Xu, Gavin Lam, Annanya Jain</i>	28
8.1	DigitalOcean Setup	28
8.2	Bugzilla	28
9	Overleaf	
	– <i>Annanya Jain, Luo Xu, Gavin Lam</i>	33
9.1	Overview	33
9.2	Initial Access Issue	33
9.3	Overleaf Setup Process	34
9.3.1	Step 1: Create Project Directory	34
9.3.2	Step 2: Create Environment File	34
9.3.3	Step 3: Write the Docker Compose File	34
9.3.4	Step 4: Launch the Containers	35
9.4	Debugging the Overleaf Restart Loop	35
9.4.1	Step 1: Check Logs	35
9.4.2	Step 2: Error Observed	35
9.5	Fixing MongoDB Configuration	35
9.5.1	Step 1: Enter Mongo Shell	35
9.5.2	Step 2: Initialize Replica Set	37
9.5.3	Step 3: Update Feature Compatibility Version	37
9.5.4	Step 4: Restart Overleaf	37
9.6	Making Overleaf Publicly Accessible	37
9.7	Access and Verification	37
9.8	Conclusion	38
10	Overleaf	
	– <i>Annanya Jain, Luo Xu, Gavin Lam</i>	41
10.1	Overview	41
10.2	Initial Access Issue	41
10.3	Overleaf Setup Process	42
10.3.1	Step 1: Create Project Directory	42
10.3.2	Step 2: Create Environment File	42
10.3.3	Step 3: Write the Docker Compose File	42
10.3.4	Step 4: Launch the Containers	43
10.4	Debugging the Overleaf Restart Loop	43
10.4.1	Step 1: Check Logs	43
10.4.2	Step 2: Error Observed	43
10.5	Fixing MongoDB Configuration	43
10.5.1	Step 1: Enter Mongo Shell	43

10.5.2	Step 2: Initialize Replica Set	45
10.5.3	Step 3: Update Feature Compatibility Version	45
10.5.4	Step 4: Restart Overleaf	45
10.6	Making Overleaf Publicly Accessible	45
10.7	Access and Verification	45
10.8	Conclusion on Setting up Overleaf	46
10.9	Setting up Overleaf Packages	46
10.10	Domain Names, SSL	49
10.11	Domain and Reverse Proxy Setup	49
10.11.1	DNS Configuration	49
10.11.2	Overleaf Docker Setup	49
10.11.3	Nginx Reverse Proxy	50
10.12	SSL Certificates, Firewall, and Debugging	50
10.12.1	Firewall Configuration	50
10.12.2	Obtaining an SSL Certificate	51
10.12.3	Verification	51
10.12.4	Debugging HTTPS Access	51
10.12.5	Renewal	52
10.12.6	Final Verification	52
10.13	Connecting Our Overleaf Instance to GitHub	53
11	Prometheus and Grafana Monitoring Setup	
	– <i>Annanya Jain, Luo Xu, Gavin Lam</i>	58
11.1	Introduction	58
11.2	The role of Prometheus, Grafana, and Node Exporter	59
11.3	Project Structure	60
11.4	Docker Compose Configuration	60
11.5	Prometheus Configuration	61
11.6	Alerting Rules	61
11.7	Grafana Data Source Configuration	62
11.8	Launching the Monitoring Stack	62
11.9	Grafana Dashboard Setup	63
11.10	Outcome and Observations	63
A	Appendix	
	– <i>Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu</i>	64
	Bibliography	66

List of Tables

1	Document Update History	iii
1	Document Update History	iv
1.1	Password Rules	1

List of Figures

8.1	bugzilla website on http://174.138.68.199/	31
9.1	overleaf website on http://174.138.68.199:8090	39
10.1	overleaf website on http://174.138.68.199:8090	47

Chapter 1

Passwords

– Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu

Table 1.1: Password Rules

User	Password Rule / Hint
OVERLEAF_ADMIN_EMAIL	Password should include a mix of upper and lower case letters, numbers, and a special character. Hint: key + specialcharacters
MONGO_INITDB_ROOT_USERNAME	Strong password required — must not contain common words or personal information. Hint: Secure passphrase based on your project title.
MONGO_INITDB_ROOT_PASSWORD	At least 10 characters, must include at least one uppercase, lowercase, number, and symbol. Hint: Prefix "Mongo" + symbol + random digits
bugzillauser (BugzillaDB)	Must include a combination of regular characters, numbers, and special characters. Hint: key + numbers + specialcharacters
bugzilla	Must include a combination of regular characters, numbers, and special characters. Hint: key + numbers + specialcharacters
devuser	At least 8 characters, include uppercase, lowercase, a number, and a symbol. Hint: First pet.
admin	Must change passwords every 90 days. Hint: Favorite City.
tester	Must include word banana Hint: Popular desert item.

Chapter 2

Hosts

– Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu

Host Name	Description
bugzilla	Bugzilla web application container running on port 80:80 to manage and track software bugs.
overleaf	Overleaf collaborative LaTeX editor container, accessible via port 8090. Configured to use MongoDB and Redis.
mongo	MongoDB service used by Overleaf for document and project data storage. Runs internally on port 27017.
redis	Redis in-memory cache used by Overleaf for session management and performance optimization.
digitalocean droplet	Shared host (Ubuntu) where both Bugzilla and Overleaf Docker environments are deployed. Accessible via SSH key linked to GitHub for secure root access.
bugzillaDB	Bugzilla database
bugzilla	Bugzilla server to catch bugs
dev-server	Primary development server.
test-server	Server for automated testing. scripts.
prod-server	Final server for working product.

Chapter 3

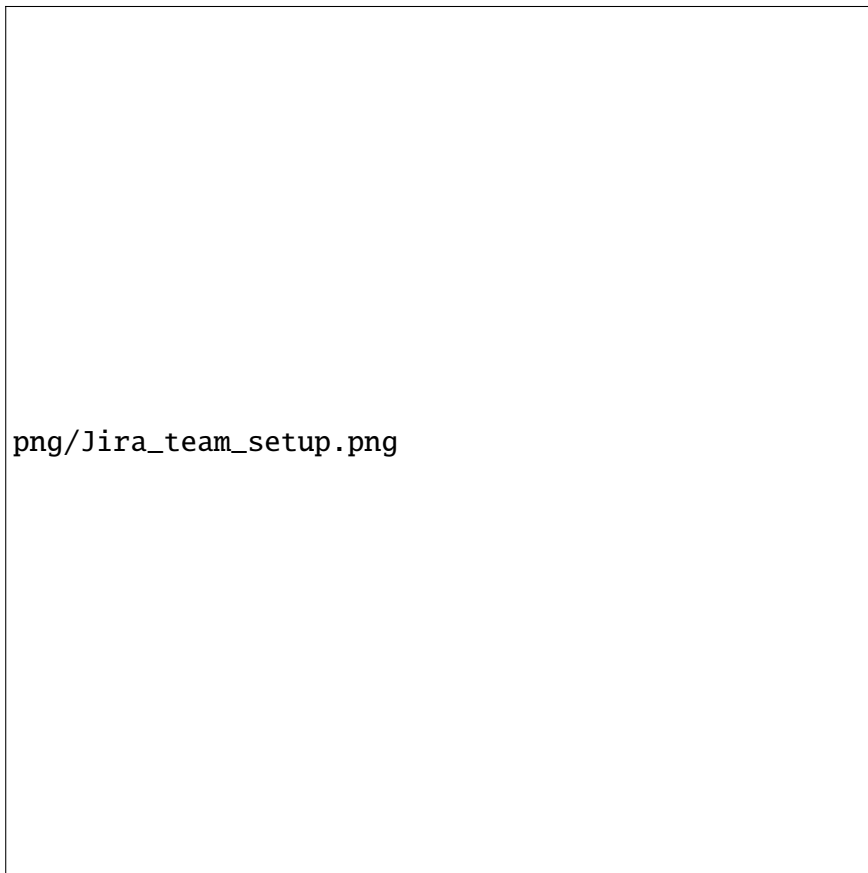
Kanban Setup

– Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu

For my DevOps project, I chose to use Atlassian JIRA to set up my Kanban board because I have some familiarity with it having used it once before in another class.

First, I created a new project in Jira and selected the **Kanban template**. I decided to use the team-managed project option and gave it the name SSW 590 after the class. JIRA automatically provided the columns **To Do**, **In Progress**, and **Done**.

Step 1: Set up Jira Team




I have chosen the name 'stevens-ssw590-fall2025' as the site name.

Step 2: Set up a Kanban Project



Step 3: Kanban



png/kanban_example.png

Chapter 4

Linux Commands

– Spurthi Setty, Gavin Lam, Annanya Jain, Luo Xu

4.1 Terminal Session

The following commands were run from ~/Documents/Devops/Assignment2 and create several test files and directories.

```
ubuntu@Ubuntu:~/Documents/Devops/Assignment2$ mkdir -p ~/lx-test && cd ~/lx-test
ubuntu@Ubuntu:~/lx-test$ printf "alpha\nbeta\nGamma\ngamma\nbeta\n" > words.txt
ubuntu@Ubuntu:~/lx-test$ printf
  ↪ "id,name,dept\n1,Ada,EE\n2,Linus,CS\n3,Grace,EE\n4,Dennis,CS\n" > people.csv
ubuntu@Ubuntu:~/lx-test$ printf "INFO boot ok\nWARN disk low\nERROR fan
  ↪ fail\nINFO shutdown\n" > sys.log
ubuntu@Ubuntu:~/lx-test$ dd if=/dev/zero of=blob.bin bs=1K count=48 status=none
ubuntu@Ubuntu:~/lx-test$ mkdir -p src/lib tmp archive
ubuntu@Ubuntu:~/lx-test$ printf "one two three four\n" > src/file1.txt
ubuntu@Ubuntu:~/lx-test$ printf "two three four five\n" > src/file2.txt
ubuntu@Ubuntu:~/lx-test$ ln -s src/file1.txt link-to-file1
ubuntu@Ubuntu:~/lx-test$ touch -t 202401020304 old.txt
```

4.2 Problem-Set Commands and Outputs

4.2.1 Navigation & File Ops

1. Present working directory

```
ubuntu@Ubuntu:~/lx-test$ pwd
/home/ubuntu/lx-test
```

2. List all entries, including dotfiles

```
ubuntu@Ubuntu:~/lx-test$ ls -A1
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 (verbose)**

```
ubuntu@Ubuntu:~/lx-test$ test -d tmp && cp -v src/file1.txt tmp/
'src/file1.txt' -> 'tmp/file1.txt'
```

4. **Move old.txt into archive/ and keep timestamp**

```
ubuntu@Ubuntu:~/lx-test$ mv -v old.txt archive/
renamed 'old.txt' -> 'archive/old.txt'
```

5. **Create an empty notes.md only if it does not exist**

```
ubuntu@Ubuntu:~/lx-test$ [ -e notes.md ] || : > notes.md
```

6. **Show disk usage (human-readable) for src directory**

```
ubuntu@Ubuntu:~/lx-test$ du -sh src
16K    src
```

4.2.2 Viewing & Searching

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

```
ubuntu@Ubuntu:~/lx-test$ nl sys.log
 1  INFO boot ok
 2  WARN disk low
 3  ERROR fan fail
 4  INFO shutdown
```

8. **Show only the lines in sys.log that contain ERROR (case-sensitive)**

```
ubuntu@Ubuntu:~/lx-test$ grep 'ERROR' sys.log
ERROR fan fail
```

9. **Count how many distinct words appear in words.txt (case-insensitive)**

```
ubuntu@Ubuntu:~/lx-test$ tr '[:upper:]' '[:lower:]' < words.txt | tr -s
↪ '[:space:]' '\n' | sort -u | wc -l
3
```

10. From words.txt, show lines that start with g or G

```
ubuntu@Ubuntu:~/lx-test$ grep -E '^[gG]' words.txt
Gamma
gamma
```

11. Display the first 2 lines of people.csv without using an editor

```
ubuntu@Ubuntu:~/lx-test$ head -n 2 people.csv
id,name,dept
1,Ada,EE
```

12. Show the last 3 lines of sys.log and keep following if the file grows

```
ubuntu@Ubuntu:~/lx-test$ tail -n 3 -f sys.log
WARN disk low
ERROR fan fail
INFO shutdown
```

4.2.3 Text Processing

13. From people.csv, print only the name column (2nd), excluding the header.

```
ubuntu@Ubuntu:~/lx-test$ tail -n +2 people.csv | cut -d',' -f2
Ada
Linus
Grace
Dennis
```

14. Sort words.txt case-insensitively and remove duplicates

```
ubuntu@Ubuntu:~/lx-test$ sort -f words.txt | uniq -i
alpha
beta
Gamma
```

15. Replace every three with 3 in all files under src/ in-place, creating .bak backups.

```
ubuntu@Ubuntu:~/lx-test$ find src -type f -exec sed -i.bak 's/three/3/g' {}
↪ +
```

16. Print the number of lines, words, and bytes for every *.txt file in src/.

```
ubuntu@Ubuntu:~/lx-test$ wc src/*.txt
 1  4 15 src/file1.txt
 1  4 16 src/file2.txt
 2  8 31 total
```

4.2.4 Permissions & Ownership

17. Make tmp/ readable, writable, and searchable only by the owner.

```
ubuntu@Ubuntu:~/lx-test$ chmod 700 tmp/
```

18. Give group execute permission to src/lib recursively without touching others/owner bits.

```
ubuntu@Ubuntu:~/lx-test$ chmod -R g+x src/lib
```

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

```
ubuntu@Ubuntu:~/lx-test$ stat -c "%a" src/file2.txt  
664
```

20. Make notes.md append-only for the owner via file attributes (if supported).

```
ubuntu@Ubuntu:~/lx-test$ sudo chattr +a notes.md
```

4.2.5 Links & Find

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

```
luo@ubuntuluo:~/lx-test$ ls -l link-to-file1  
lrwxrwxrwx 1 luo luo 13 Sep 16 18:56 link-to-file1 -> src/file1.txt
```

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

```
luo@ubuntuluo:~/lx-test$ find . -type f -size +40k
```

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

```
luo@ubuntuluo:~/lx-test$ find tmp/ -type f -mmin -10 -exec ls -lh {} +
```

4.2.6 Processes & Job Control

24. Show your processes in a tree view.

```
luo@ubuntuluo:~/lx-test$ pstree -p
```

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

```
luo@ubuntuluo:~/lx-test$ sleep 120 &
echo $!
[1] 4474
4474
```

26. Send a TERM signal to all sleep processes owned by you (don't use kill -9).

```
luo@ubuntuluo:~/lx-test$ pkill -TERM -u "$USER" sleep

[1]  + Terminated  sleep 120
```

27. Show the top 5 processes by memory usage (one-shot, not interactive).

```
luo@ubuntuluo:~/lx-test$ ps -eo pid,ppid,user,%mem,%cpu,comm --sort=-%mem |
→ head -n 5
PID      PPID USER      %MEM %CPU COMMAND
  1925    1711 luo         9.8  4.0 gnome-shell
  2451    1925 luo         2.4  0.0 mutter-x11-fram
  2328    1711 luo         2.0  0.0 gsd-xsettings
  2245    1925 luo         1.7  0.0 Xwayland
```

4.2.7 Archiving & Compression

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

```
luo@ubuntuluo:~/lx-test$ tar -czf src.tgz -C src .
```

29. List the contents of src.tgz without extracting.

```
luo@ubuntuluo:~/lx-test$ tar -tzf src.tgz
./
./file2.txt
./lib/
./file1.txt.bak
./file2.txt.bak
./file1.txt
```

30. Extract only file2.txt from src.tgz into tmp/.

```
luo@ubuntuluo:~/lx-test$ tar -xvzf src.tgz -C tmp ./file2.txt
./file2.txt
```

4.2.8 Networking & System Info

31. Show all listening TCP sockets with associated PIDs (no root assumptions).

```
annanyajain@ubuntu:~/lx-test$ ss -tlnp
State  Recv-Q  Send-Q      Local Address:Port      Peer Address:Port  Process
```

32. Print your default route (gateway) in a concise form.

```
annanyajain@ubuntu:~/lx-test$ ip route show default
default via 198.19.249.1 dev eth0 proto dhcp src 198.19.249.228 metric 100
```

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

```
annanyajain@ubuntu:~/lx-test$ uname -srm
Linux 6.12.10-orbstack-00297-gf8f6e015b993 aarch64
```

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

```
annanyajain@ubuntu:~/lx-test$ last -n 5
reboot    system boot  6.12.10-orbstack Tue Sep 16 01:01    still running
reboot    system boot  6.12.10-orbstack Tue Jan 21 13:27 - 13:40    (00:12)
reboot    system boot  6.12.10-orbstack Tue Jan 21 10:51 - 10:56    (00:04)
reboot    system boot  6.12.9-orbstack- Mon Jan 20 19:48 - 19:48    (00:00)
reboot    system boot  6.12.9-orbstack- Mon Jan 20 17:30 - 19:48    (02:18)

wtmp begins Mon Jan 20 15:07:29 2025
```

4.2.9 Package & Services (Debian/Ubuntu)

35. Show the installed version of package coreutils.

```
annanyajain@ubuntu:~/lx-test$ dpkg -s coreutils | grep '^Version:'
Version: 8.32-4.1ubuntu1.2
```

36. Search available packages whose names contain ripgrep.

```
annanyajain@ubuntu:~/lx-test$ apt search ripgrep
Sorting... Done
Full Text Search... Done
elpa-dumb-jump/jammy 0.5.3-1 all
  jump to definition for multiple languages without configuration

ripgrep/jammy-updates,jammy-security 13.0.0-2ubuntu0.1 arm64
  Recursively searches directories for a regex pattern
```

```
ugrep/jammy 3.7.2+dfsg-1 arm64  
faster grep with an interactive query UI
```

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

```
annanyajain@ubuntu:~/lx-test$ systemctl status cron | grep 'Active:'  
Active: active (running) since Tue 2025-09-16 01:01:18 EDT; 31min ago
```

4.2.10 Bash & Scripting

38. Write a one-liner that loops over *.txt in src/ and prints: : (Let's print number of words in the files)

```
annanyajain@ubuntu:~/lx-test$ for f in src/*.txt; do echo "$f: $(wc -w <  
→ "$f")";done  
src/file1.txt: 4  
src/file2.txt: 4
```

39. Write a command that exports CSV rows where dept == "CS" to cs.txt (exclude header).

So here, let me first see the structure of a csv file in src directory:

```
annanyajain@ubuntu:~/lx-test$ cat people.csv  
id,name,dept  
1,Ada,EE  
2,Linus,CS  
3,Grace,EE  
4,Dennis,CS
```

Now I know that third column is the dept:

```
annanyajain@ubuntu:~/lx-test$ awk -F, 'NR>1 && $3=="CS" {print}' people.csv >  
→ cs.txt
```

The output is redirected into the cs.txt file. We can now verify the results:

```
annanyajain@ubuntu:~/lx-test$ cat cs.txt  
2,Linus,CS  
4,Dennis,CS
```

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

```
annanyajain@ubuntu:~/lx-test$ export X=42; echo $X; unset X  
42
```

The variable `X` was created with value 42, and printed. The bash command (`unset X`) was used to remove it from the environment. Now, let's verify whether the variable is still there:

```
annanyajain@ubuntu:~/lx-test$ echo $X;
```

Nothing got printed. Hence, it is removed from the environment.

Chapter 5

Project Proposal

– Luo Xu, Annanya Jain, Gavin Lam

Our project is an AI Health Voice Assistant named AVA, short for Artificial Voice Assistant. Users will be able to log in and create an account and chat with AVA. This project aims to support users in tracking emotions, managing reminders, and accessing mental wellness resources. Unlike traditional chatbots, our assistant goes beyond simple question–answer interaction by integrating:

- Agentic AI workflows, enabling the assistant to interpret user intent, plan actions, and decide between generating responses, retrieving wellness exercises, or scheduling reminders.
- DevSecOps practices to make it scalable, and secure from development to deployment. The end goal is a functional prototype that not only demonstrates AI-driven health support but also serves as a practical use of modern DevOps pipelines and monitoring

This project will be an enhancement of our previously created project for CS555 course. Since then we have grown a lot in terms of knowledge and skill sets and we would like to improve it to have agentic powers to better aid users. Here is the link for the repository for the project: <https://github.com/cascadingluo/SSW590-team-7-project>

Tool	Usage
Flask	Web framework.
MongoDB	Database for user information and chat history storage.
Gemini	AI API used for the chatbot.
GitHub	Source Control and Collaboration
CI/CD	Github Actions for automated testing
Jira	Issue tracking and Agile project management tool.
AWS	Pushing local Docker with AWS and deploying with App Runner.
TikZ	Vizualizing Architecture of our assistant.

Chapter 6

AWS

– Gavin Lam, Luo Xu, Annanya Jain

To get our website deployed, we followed the steps in the DockerLocalAndAWS.pdf document with some changes to the code as parts in the document did not work on our machines.

First we created an AWS root account at aws.amazon.com. MFA was enabled on the root account and our region was set to us-east-2. A cost budget was set up as well to monitor costs. The SSO start URL was also recorded to be used later

We then enabled the IAM Identity center and created a new user with an alternate email. At first this user wasn't able to access any dashboards as it had no roles or permissions. This user was given the permission set AdministratorAccess by the root account and after a relog was able to access a dashboard. From this dashboard the AWS Access Key ID, AWS Secret Access Key, and AWS session ID were written down.

After the AWS accounts were correctly set up, we ran the commands in the document on the terminal.

```
aws configure sso
```

This command prompted us to give answers

```
#SSO session name:
#SSO start URL:
#SSO region:
#Account:
#Role:
#Default region:
#Output:
```

After entering all the details the login was saved.

The command ran next was

```
aws configure
```

This command prompted us to give answers

```
#AWS Access key ID:
#AWS Secret Access key:
#AWS Session ID:
#Default region:
#Output:
```

After entering all the details we successfully logged in and ran another command to confirm.

```
aws sts get-caller-identity
```

The next steps were to create an ECR repository as stated in the document. We first set some environment variables through these commands.

```
export AWS_REGION=us-east-2
export ECR_REPO=myapp
export IMAGE_TAG=v1
export CONTAINER_PORT=3000
export AWS_ACCOUNT_ID="$(aws sts get-caller-identity --query Account --output
↪ text --profile (my account here))"
```

After setting the variables we created a new ECR repository through these commands.

```
aws ecr create-repository \
  --repository-name "$ECR_REPO" \
  --image-scanning-configuration scanOnPush=true \
  --region "AWS_REGION" --profile (my account here)
```

AWS CLI was then used to obtain a short-lived registry token and logged in Docker.

```
aws ecr get-login-password --region "AWS_REGION" --profile default \
| docker login --username AWS --password-stdin \
  "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com"
```

Our local image was then built, tagged, and pushed. These commands were run from the directory containing the Dockerfile.

```
docker build --platform linux/amd64 -t "$ECR_REPO:$IMAGE_TAG" .

docker tag "$ECR_REPO:$IMAGE_TAG" \
"$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPO:$IMAGE_TAG"

docker push
↪ "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPO:$IMAGE_TAG"
```

Verifying the image in ECR was next

```
aws ecr describe-images \
  --repository-name "$ECR_REPO" \
  --region "AWS_REGION" --profile default \
  --query 'imageDetails[].imageTags'
```

The next step of quick deploying with App Runner was where we ran into a roadblock. The code shown when inputted resulted in the error below.

```
export APP_NAME=my-apprunner-app

aws apprunner create-service \
  --service-name "$APP_NAME" \
  --region "$AWS_REGION" --profile default \
  --source-configuration "{
    \"ImageRepository\": {
      \"ImageIdentifier\": \"$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPO:$IMAGE_TAG\",
      \"ImageRepositoryType\": \"ECR\",
      \"ImageConfiguration\": {\"Port\": \"$CONTAINER_PORT\"}
    },
    \"AutoDeploymentsEnabled\": true
  }" \
  --instance-configuration "{\"Cpu\": \"1 vCPU\", \"Memory\": \"2 GB\"}"
```

An error occurred (InvalidRequestException) when calling the CreateService
 ↳ operation: Authentication configuration is invalid.

After looking at online resources we assumed that the permission set AdministratorAccess was not working as we hoped and didn't give us the permissions we needed to use CreateService. In order to fix this we created a trust policy named AppRunnerECRAccessRole in IAM roles. The json used is:

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::(our account id
        ↳ here):role/AppRunnerECRAccessRole"
    },
    {
      "Effect": "Allow",
      "Action": [ "apprunner:CreateService", "apprunner:UpdateService",
        "apprunner:DeleteService" ],
      "Resource": "*" }
  ]
}
```

After relogging the SSO session and rerunning early commands to confirm we are logged in properly everywhere we retried the original deploying command again. We did some more research and found out more commands we could try. First we had to update the trust policy to replace the second bracket of code.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "build.apprunner.amazonaws.com" },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The final inline policy we used is

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::(our account id
        ↪ here):role/AppRunnerECRAccessRole"
    },
    {
      "Effect": "Allow",
      "Principal": { "Service": "build.apprunner.amazonaws.com" },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

We then had to attach permissions to the role. The AWS managed policy `AWSAppRunnerServicePolicyForECRAccess` was added. After confirming the role we retried the command again. The error persisted. We were stuck at this point so we prompted chatGPT multiple times before it recommended us to correct the create-service command with safe quoting, i.e. use single quotes around the JSON blocks so the shell does not break the quoting. The new create-service command is

```
aws apprunner create-service \
  --service-name "$APP_NAME" \
  --region "$AWS_REGION" --profile default \
  --source-configuration '{
    "ImageRepository": {
      "ImageIdentifier": "'"$AWS_ACCOUNT_ID"'.dkr.ecr.'"$AWS_REGION"'.amazonaws.'
        ↪ com/'"$ECR_REPO"':'"$IMAGE_TAG"'",
      "ImageRepositoryType": "ECR",
    }
  }
```

```

    "ImageConfiguration": { "Port": ""$CONTAINER_PORT"" }
  },
  "AuthenticationConfiguration": {
    "AccessRoleArn":
      ↪ "arn:aws:iam::"$AWS_ACCOUNT_ID":role/AppRunnerECRAccessRole"
  },
  "AutoDeploymentsEnabled": true
}' \
--instance-configuration '{
  "Cpu": "1 vCPU",
  "Memory": "2 GB"
}'

```

This miraculously went through, and we thought we had successfully created the service. We continued following the steps in the document creating more environment variables.

```

AWS_REGION=us-east-2
PROFILE=(Our Profile here)
SERVICE_ARN=(Our Service ARN here)
}

```

We then tried to run the service status changes to running command but this resulted in another error:

```

while true; do
STATUS=$(aws apprunner describe-service \
--service-arn "$SERVICE_ARN" \
--region "$AWS_REGION" --profile "$PROFILE" \
--query 'Service.Status' --output text)
echo "Service status: $STATUS"
case "$STATUS" in RUNNING|CREATE_FAILED|OPERATION_FAILED) break ;; esac
sleep 4
done

```

An error occurred (InvalidRequestException) when calling the DescribeService
 ↪ operation: Authentication configuration is invalid.

We then read the document again and saw that the next line was to grab the service URL and just hoped that the service was running properly.

```
aws apprunner list-services \  
--region "$AWS_REGION" --profile default \  
--query "ServiceSummaryList[?ServiceName=='$APP_NAME'].ServiceUrl" --output text
```

Unfortunately, it did not create successfully and we ran into an error website. We were stuck at this point and did not know how to proceed so we decided to leave it for a little bit and come back at a later time to retry.

After coming back at a later time we reran through the steps of, aws configure sso, aws configure (with new details this time), sanity test, resetting the environment variables (they did not change from above we just reentered them), authenticated Docker to ECR, built, tagged, and pushed the local image, verified the image in ECR, deployed with App Runner. This is where we ran into the same error, but we found a service on AWS called CloudWatch that essentially replaced the need for the "test the service status changes to running" command. CloudWatch enabled us to look at the logs of services we have tried to deploy. After checking CloudWatch we saw the previous instance of our service had failed due to the ECR repository not being recognized, and saw that our most recent creation of the service had successfully deployed. We then ran the command to get the service URL.

```
aws apprunner list-services \  
--region "$AWS_REGION" --profile default \  
--query "ServiceSummaryList[?ServiceName=='$APP_NAME'].ServiceUrl" --output text
```

Which resulted in

```
mu3fbjbbv2.us-east-2.awsapprunner.com
```

This URL works, which means our deployment with AWS was successful.

After the successful deployment, we change the code for the original website to use a class-based javascript code instead of methods. The code largely remain the same, we just moved parts of code in the html file into a javascript file and had the html file run the script instead of the code directly. We have the new ColorController class that has DOM references, event binding and the color-changing behaviors. ColorController.init() connects the app on DOMContentLoaded. Here is the code in our new javascript file:

```
class ColorController { //class  
  constructor({ blueBtnId = "blueBtn", redBtnId = "redBtn", target =  
    ↪ document.body } = {}) {
```

```

    this.target = target;
    this.blueBtn = document.getElementById(blueBtnId);
    this.redBtn = document.getElementById(redBtnId);
    this._bind();
  }

  // private method to bind event listeners, the same exact code as the ones
  // originally in html
  _bind() {
    if (this.blueBtn) {
      this.blueBtn.addEventListener("click", () => this.setColor("blue"));
    }
    if (this.redBtn) {
      this.redBtn.addEventListener("click", () => this.setColor("red"));
    }
  }

  setColor(color) {
    this.target.style.backgroundColor = color;
  }

  static init(opts) {
    return new ColorController(opts);
  }
}

document.addEventListener("DOMContentLoaded", () => {
  ColorController.init();
});

```

Here is the UML diagram for the new code (this was created with the help of GPT-5):

```

+-----+
|  ColorController  |
+-----+
| - blueBtn: Button |
| - redBtn:  Button |
| - target: Element |
+-----+
| + constructor(...) |
| - _bind(): void    |
| + setColor(c):void |
| + init(...): CC    |
+-----+

```

Notes:

- constructor wires DOM elements and calls _bind()

- `_bind` attaches event listeners to blue/red buttons
- `setColor` updates the background color of the target
- `init` is a convenience factory method

Chapter 7

LaTeX Docker

– Annanya Jain, Gavin Lam, Luo Xu

In this chapter, we created a Docker container to compile a simple LaTeX document using TeX Live, which is basically what Overleaf does.

7.1 Steps taken for creating the docker container to compile a LaTeX File

I created the following files in a folder named: texlive-app:

```
texlive-app/  
  Dockerfile  
  main
```

7.1.1 main.tex

```
\begin{document}  
  
\title{Password Policy Documentation}  
\date{\today}  
\maketitle  
Let me take an example of LaTeX file with a simple table to compile.  
\section{Password Rules}  
\begin{longtable}{|l|p{9cm}|}  
\caption{Password Rules \label{Table::Passwords}}\\  
\hline  
\textbf{User} & \textbf{Password Rule / Hint} \\  
\hline  
\endhead
```

```

devuser & At least 8 characters, include uppercase,
lowercase, a number, and a symbol. Hint: First pet.
\\
\hline

admin & Must change passwords every 90 days. Hint: Favorite City.
\\
\hline

tester & Must include word banana Hint: Popular dessert item.
\\
\hline

\end{longtable}

\end{document}

```

7.1.2 Dockerfile

```

FROM debian:stable-slim

RUN apt-get update && apt-get install -y \
    texlive-latex-base \
    texlive-latex-recommended \
    texlive-latex-extra \
    texlive-fonts-recommended \
    texlive-fonts-extra \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /data

CMD ["pdflatex", "main.tex"]

```

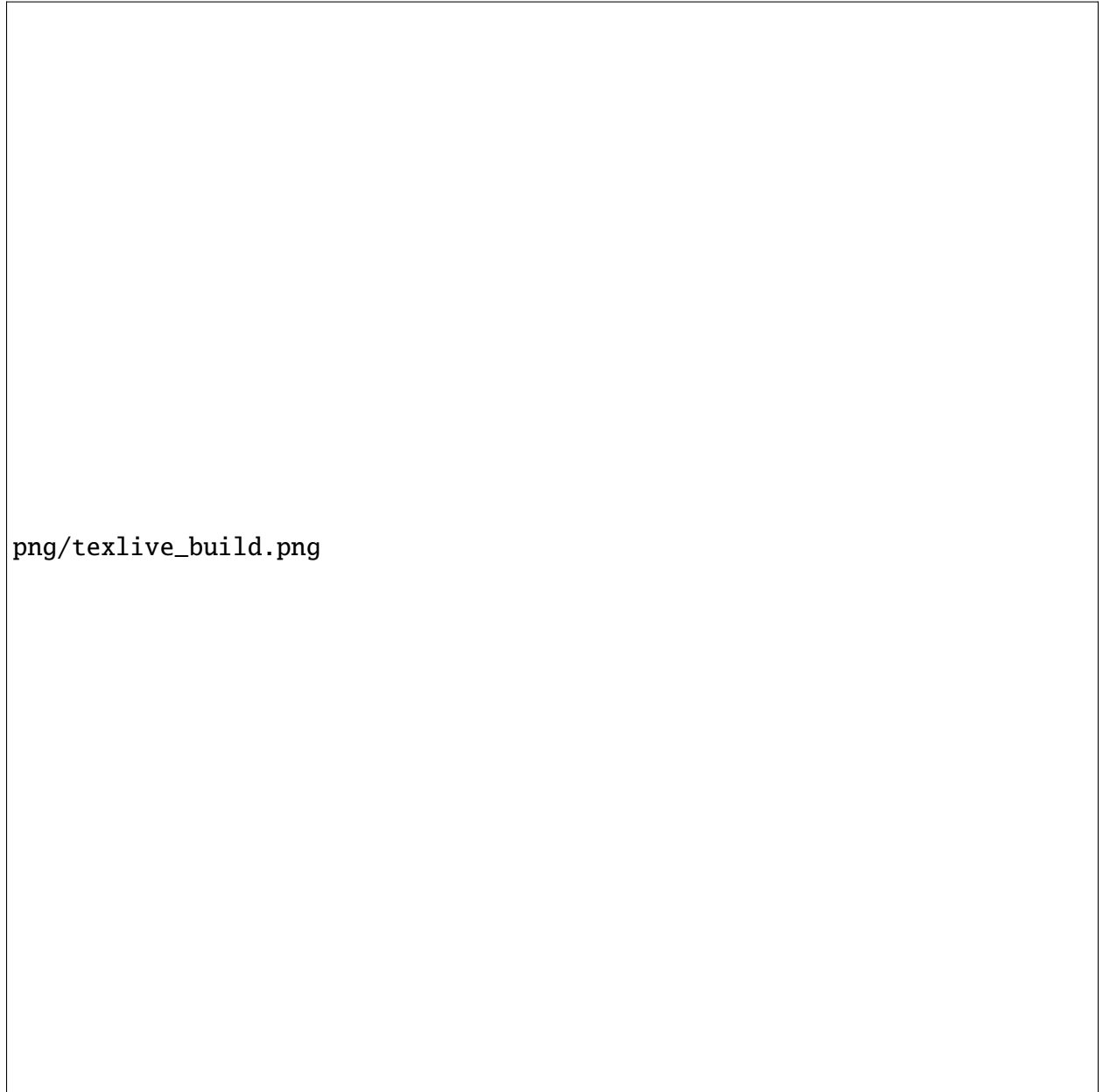
7.1.3 Building the Docker image

To build and run the container:

```


docker build -t texlive-app .
docker run --rm -v $(pwd):/data texlive-app

```



png/texlive_build.png

After running the commands, the output file `main.pdf` gets generated in the project folder on the machine.



png/main_pdf generated.png

Chapter 8

Bugzilla

– Luo Xu, Gavin Lam, Annanya Jain

8.1 DigitalOcean Setup

We made an account in DigitalOcean via GitHub and received 200 credits via student package. We created an droplet with these stats: 2 GB Memory / 60 GB Disk / NYC3 - Ubuntu 24.04 (LTS) x64

8.2 Bugzilla

After setting up the droplet, we ssh into the droplet through the local terminal with the public ip: 174.138.68.199. Then I checked the updates, and installed the essential tools and prerequisites.

```
ssh root@174.138.68.199
apt update
apt upgrade -y
apt install -y git curl wget nano build-essential
apt install -y apache2 libapache2-mod-perl2 \
    mariadb-server mariadb-client \
    libcgi-pm-perl libdbi-perl libdbd-mysql-perl \
    libtemplate-perl libdatetime-perl libdatetime-timezone-perl \
    libemail-sender-perl libemail-mime-perl libxml-twig-perl \
    libgd-perl libjson-xs-perl libauthen-sasl-perl libnet-ldap-perl \
    libsoap-lite-perl libxmlrpc-lite-perl libtest-taint-perl \
    libhtml-scrubber-perl libfile-mimeinfo-perl libcache-memcached-perl \
    perlmagick graphviz lynx python3-sphinx
```

I then configured the database.

```
systemctl start mariadb
systemctl enable mariadb
mysql_secure_installation
```

I logged into the database.

```
mysql -u root -p
```

and then inside the shell i set up a user in the SQL shell.

```
CREATE DATABASE bugzilla;
CREATE USER 'bugzillauser'@'localhost' IDENTIFIED BY '<password here>';
GRANT ALL PRIVILEGES ON bugzilla.* TO 'bugzillauser'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

I downloaded and configured bugzilla.

```
cd /var/www
git clone https://github.com/bugzilla/bugzilla.git
# Or download a tarball, e.g. wget from bugzilla.org, then extract
```

Here was when we realized we might have made an mistake of not running this in docker, so we went to stop the services based on what chat said, and resetted the host services within the droplet.

```
sudo systemctl stop apache2 || true
sudo systemctl disable apache2 || true
sudo systemctl stop mariadb || true
sudo systemctl disable mariadb || true
```

Then we installed docker and all the compose plugins incase we are missing anything.

```
sudo apt update
sudo apt install -y ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
  ↪ \
  https://download.docker.com/linux/ubuntu $(. /etc/os-release; echo
  ↪ $VERSION_CODENAME) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
  ↪ docker-compose-plugin
```

Then I made a project folder and env file.

```
mkdir -p ~/bugzilla-docker
cd ~/bugzilla-docker

cat > .env << 'EOF'
# ---- DB credentials (choose your own secure values) ----
MYSQL_ROOT_PASSWORD=<password>
BUGZ_DB=bugzilla
```

```
BUGZ_USER=bugzillauser
BUGZ_PASS=<password>

# ---- Internal service names / URLs ----
BUGZ_HOST=bugzilla
# If you have a domain now, set https URL. If not, set http://<PUBLIC_IP> for now
→ and change later.
BUGZ_URL=http://<PUBLIC_IP>
EOF
```

I created docker-compose.yml.

```
services:
  db:
    image: mariadb:10.6
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${BUGZ_DB}
      MYSQL_USER: ${BUGZ_USER}
      MYSQL_PASSWORD: ${BUGZ_PASS}
    command:
      → ["--character-set-server=utf8mb4","--collation-server=utf8mb4_unicode"]
    volumes:
      - db_data:/var/lib/mysql
    networks: [bugznet]

  bugzilla:
    image: nasqueron/bugzilla:latest
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    environment:
      DB_HOST: db
      DB_USER: ${BUGZ_USER}
      DB_PASSWORD: ${BUGZ_PASS}
      DB_DATABASE: ${BUGZ_DB}
      BUGZILLA_URL: ${BUGZ_URL}
    ports:
      - "80:80"
    volumes:
      - bug_data:/var/www/html/bugzilla
    networks: [bugznet]

volumes:
  db_data:
  bug_data:
```

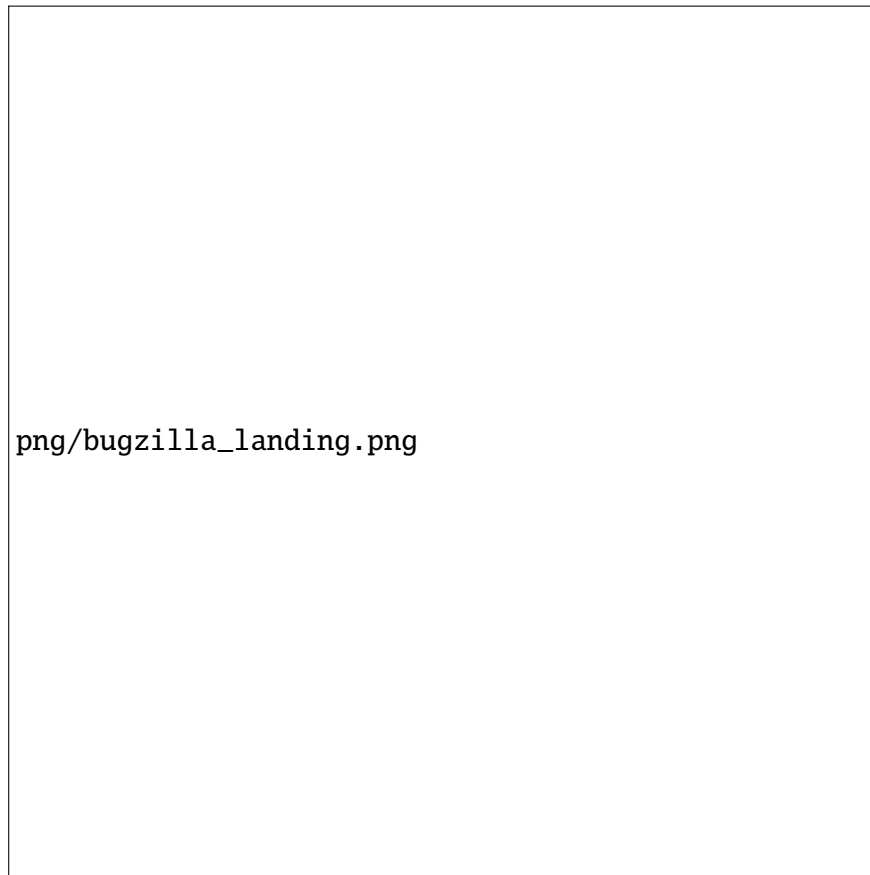


Figure 8.1: bugzilla website on <http://174.138.68.199/>

```
networks:  
  bugznet:
```

I create the stack and checked the logs for the email and password.

```
docker compose down && docker compose up -d  
docker compose logs -f bugzilla  
docker compose exec -it db bash  
docker compose restart bugzilla  
docker compose logs -f bugzilla  
...  
bugzilla-1 | If no admin account is already defined in your database, this one  
→ will be created:  
bugzilla-1 |  
bugzilla-1 |           E-mail ..... admin@domain.tld  
bugzilla-1 |           Password ... OdMW0br43g6VW2uG8  
...
```

I then went to the bugzilla site.

logged in with the given email and password, then head to Administration → Users and re-setted the password and email to real ones.

and now everything is running on <http://174.138.68.199/>. :)

Chapter 9

Overleaf

– Annanya Jain, Luo Xu, Gavin Lam

9.1 Overview

This document explains the complete process of deploying **Overleaf (Community Edition)** on a DigitalOcean droplet that already hosted **Bugzilla** via Docker. It includes the initial access issues, configuration steps, and debugging process that led to a fully functional Overleaf instance running alongside Bugzilla on the same host.

9.2 Initial Access Issue

At the beginning, I could not connect to the droplet from my local machine using its public IP address. Both `ssh root@174.138.68.199` and web requests to `http://174.138.68.199` were failing with timeout errors.

Since I didn't originally create the droplet, I did not have SSH access permissions. To resolve this, my teammate who had created the droplet and had root access, added my GitHub SSH key to the droplet's authorized keys through the DigitalOcean dashboard:

1. Logged into **DigitalOcean**.
2. Opened the droplet's page.
3. Navigated to **Access → Add SSH Keys**.
4. Added my GitHub SSH key (fetched automatically from my GitHub account).

After this, I was able to connect successfully:

```
ssh root@174.138.68.199
```

Once connected, I could verify that Bugzilla was already running:

```
docker ps
```

Bugzilla was occupying port 80, so I decided to host Overleaf on a different port.

9.3 Overleaf Setup Process

9.3.1 Step 1: Create Project Directory

```
mkdir ~/overleaf-docker  
cd ~/overleaf-docker
```

9.3.2 Step 2: Create Environment File

```
nano .env
```

Contents:

```
OVERLEAF_PORT=8090  
OVERLEAF_DOMAIN=http://174.138.68.199:8090  
OVERLEAF_ADMIN_EMAIL=(email here)  
OVERLEAF_ADMIN_PASSWORD=(password here)  
MONGO_INITDB_ROOT_USERNAME=(username here)  
MONGO_INITDB_ROOT_PASSWORD=(password here)
```

9.3.3 Step 3: Write the Docker Compose File

```
1 version: "3.8"  
2  
3 services:  
4   mongo:  
5     image: mongo:6.0  
6     restart: unless-stopped  
7     command: ["--replSet", "rs0", "--bind_ip_all"]  
8     volumes:  
9       - mongo_data:/data/db  
10  
11   redis:  
12     image: redis:7  
13     restart: unless-stopped  
14  
15   overleaf:  
16     image: sharelatex/sharelatex:latest  
17     restart: unless-stopped  
18     depends_on:  
19       - mongo  
20       - redis  
21     ports:  
22       - "8090:80"  
23     environment:  
24       OVERLEAF_MONGO_URL: mongodb://mongo:27017/sharelatex?replicaSet=rs0
```

```
25     OVERLEAF_REDIS_HOST: redis
26     OVERLEAF_SITE_URL: http://174.138.68.199:8090
27     OVERLEAF_ADMIN_EMAIL: admin@example.com
28     OVERLEAF_ADMIN_PASSWORD: password123
29     OVERLEAF_APP_NAME: Overleaf
30     OVERLEAF_ALLOW_PUBLIC_REGISTRATION: "true"
31
32 volumes:
33     mongo_data:
```

9.3.4 Step 4: Launch the Containers

```
docker compose up -d
docker ps
```

At this point, MongoDB, Redis, and Overleaf containers appeared, but Overleaf was continuously restarting.

9.4 Debugging the Overleaf Restart Loop

9.4.1 Step 1: Check Logs

```
docker logs overleaf-docker-overleaf-1 --tail 40
```

9.4.2 Step 2: Error Observed


```
The MongoDB server has featureCompatibilityVersion=5.0,
but Overleaf requires at least version 6.0.
Aborting.
```

This indicated that MongoDB was version 6.0, but its internal `featureCompatibilityVersion` (FCV) was still set to 5.0.

9.5 Fixing MongoDB Configuration

9.5.1 Step 1: Enter Mongo Shell

```
docker exec -it overleaf-docker-mongo-1 mongosh
```



png/restarting_problem_overleaf.png

9.5.2 Step 2: Initialize Replica Set

```
rs.initiate()
```

If you see `MongoServerError[AlreadyInitialized]`, it means it's already set up — that's fine.

9.5.3 Step 3: Update Feature Compatibility Version

```
use admin
db.adminCommand({ setFeatureCompatibilityVersion: "6.0" })
```

Expected output:

```
{ "ok" : 1 }
```

9.5.4 Step 4: Restart Overleaf

```
exit
docker restart overleaf-docker-overleaf-1
```

9.6 Making Overleaf Publicly Accessible

Since Bugzilla was already bound to port 80, Overleaf was assigned to port 8090. I confirmed the port was open:

```
ss -tuln | grep 8090
```

Then enabled it in the firewall:

```
sudo ufw allow 8090/tcp
sudo ufw reload
```

9.7 Access and Verification

Once restarted, Overleaf was reachable at:

```
http://174.138.68.199:8090
```

I verified it via:

```
curl -I http://174.138.68.199:8090
```

Result:

```
HTTP/1.1 200 OK
```

9.8 Conclusion

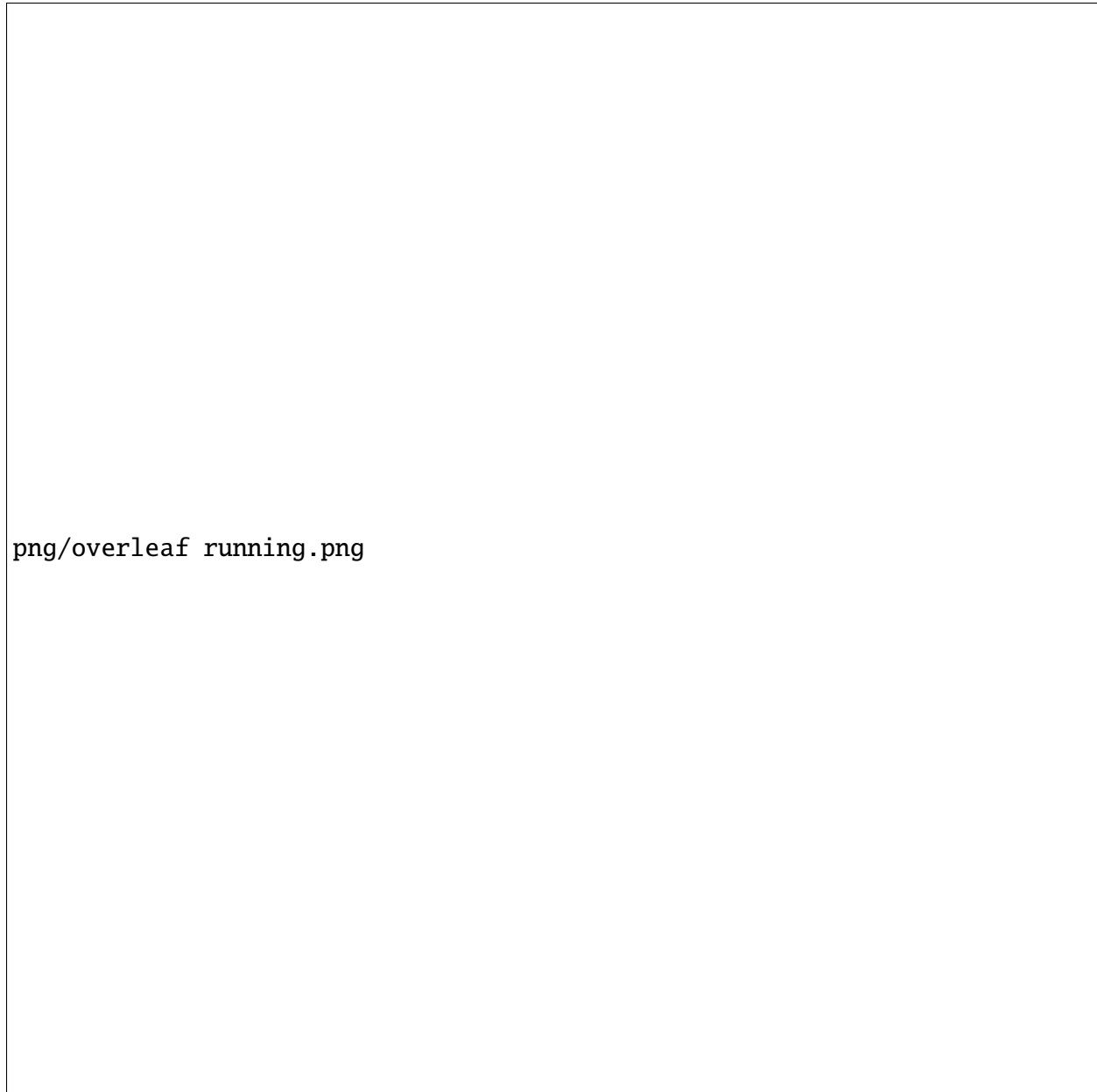
The deployment succeeded after:

1. Gaining SSH access by adding my GitHub SSH key to the droplet.
2. Running Overleaf on port 8090 (to avoid conflict with Bugzilla on port 80).
3. Initializing the MongoDB replica set.
4. Updating the MongoDB feature compatibility version to 6.0.

After these steps, Overleaf was fully functional and accessible publicly via:

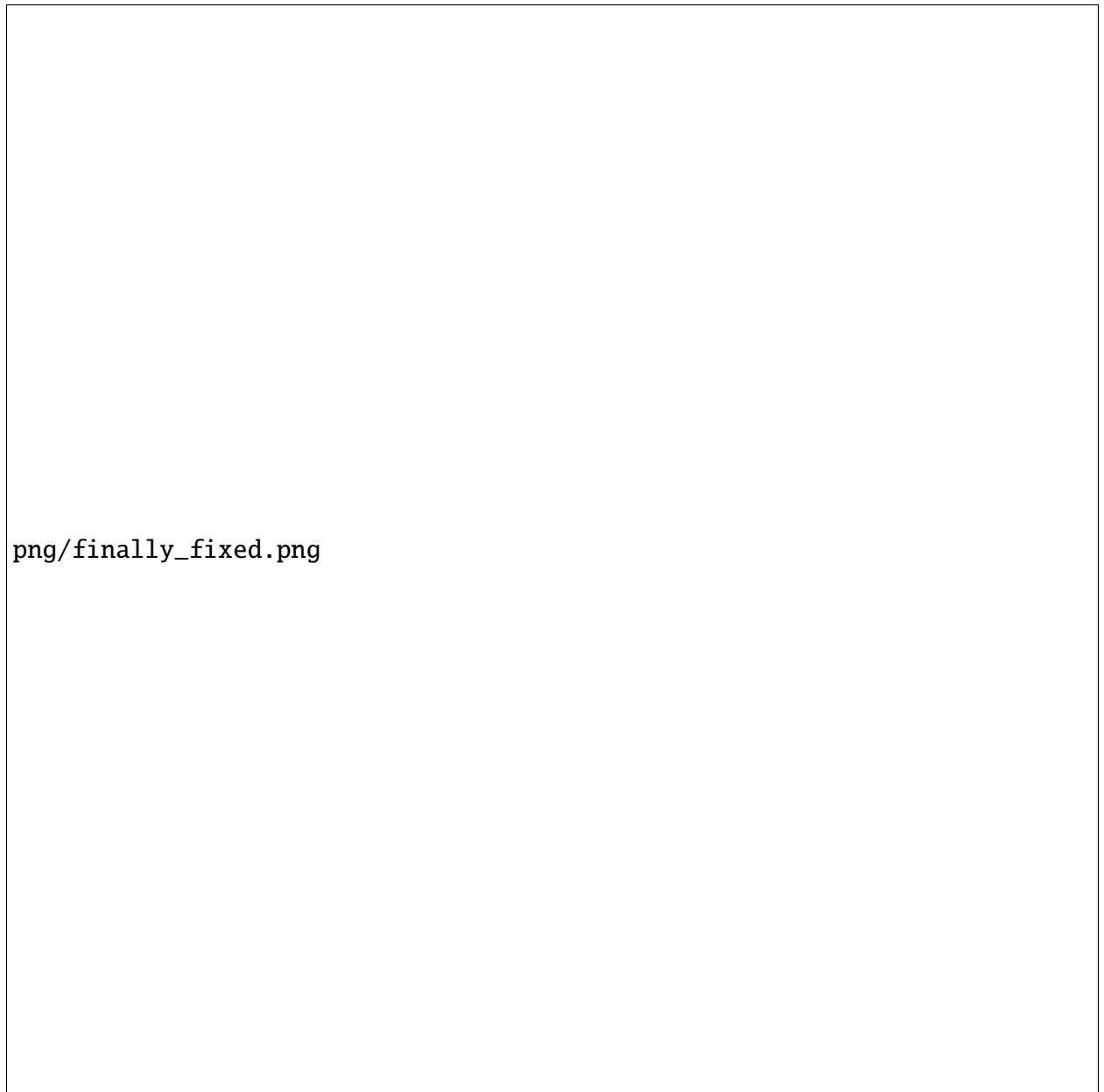
<http://174.138.68.199:8090>

This was what was seen:



png/overleaf running.png

Figure 9.1: overleaf website on <http://174.138.68.199:8090>



png/finally_fixed.png

Chapter 10

Overleaf

– Annanya Jain, Luo Xu, Gavin Lam

10.1 Overview

This document explains the complete process of deploying **Overleaf (Community Edition)** on a DigitalOcean droplet that already hosted **Bugzilla** via Docker. It includes the initial access issues, configuration steps, and debugging process that led to a fully functional Overleaf instance running alongside Bugzilla on the same host.

10.2 Initial Access Issue

At the beginning, I could not connect to the droplet from my local machine using its public IP address. Both `ssh root@174.138.68.199` and web requests to `http://174.138.68.199` were failing with timeout errors.

Since I didn't originally create the droplet, I did not have SSH access permissions. To resolve this, my teammate who had created the droplet and had root access, added my GitHub SSH key to the droplet's authorized keys through the DigitalOcean dashboard:

1. Logged into **DigitalOcean**.
2. Opened the droplet's page.
3. Navigated to **Access → Add SSH Keys**.
4. Added my GitHub SSH key (fetched automatically from my GitHub account).

After this, I was able to connect successfully:

```
ssh root@174.138.68.199
```

Once connected, I could verify that Bugzilla was already running:

```
docker ps
```

Bugzilla was occupying port 80, so I decided to host Overleaf on a different port.

10.3 Overleaf Setup Process

10.3.1 Step 1: Create Project Directory

```
mkdir ~/overleaf-docker  
cd ~/overleaf-docker
```

10.3.2 Step 2: Create Environment File

```
nano .env
```

Contents:

```
OVERLEAF_PORT=8090  
OVERLEAF_DOMAIN=http://174.138.68.199:8090  
OVERLEAF_ADMIN_EMAIL=(email here)  
OVERLEAF_ADMIN_PASSWORD=(password here)  
MONGO_INITDB_ROOT_USERNAME=(username here)  
MONGO_INITDB_ROOT_PASSWORD=(password here)
```

10.3.3 Step 3: Write the Docker Compose File

```
1 version: "3.8"  
2  
3 services:  
4   mongo:  
5     image: mongo:6.0  
6     restart: unless-stopped  
7     command: ["--replSet", "rs0", "--bind_ip_all"]  
8     volumes:  
9       - mongo_data:/data/db  
10  
11   redis:  
12     image: redis:7  
13     restart: unless-stopped  
14  
15   overleaf:  
16     image: sharelatex/sharelatex:latest  
17     restart: unless-stopped  
18     depends_on:  
19       - mongo  
20       - redis  
21     ports:  
22       - "8090:80"  
23     environment:  
24       OVERLEAF_MONGO_URL: mongodb://mongo:27017/sharelatex?replicaSet=rs0
```

```
25     OVERLEAF_REDIS_HOST: redis
26     OVERLEAF_SITE_URL: http://174.138.68.199:8090
27     OVERLEAF_ADMIN_EMAIL: admin@example.com
28     OVERLEAF_ADMIN_PASSWORD: password123
29     OVERLEAF_APP_NAME: Overleaf
30     OVERLEAF_ALLOW_PUBLIC_REGISTRATION: "true"
31
32 volumes:
33     mongo_data:
```

10.3.4 Step 4: Launch the Containers

```
docker compose up -d
docker ps
```

At this point, MongoDB, Redis, and Overleaf containers appeared, but Overleaf was continuously restarting.

10.4 Debugging the Overleaf Restart Loop

10.4.1 Step 1: Check Logs

```
docker logs overleaf-docker-overleaf-1 --tail 40
```

10.4.2 Step 2: Error Observed


```
The MongoDB server has featureCompatibilityVersion=5.0,
but Overleaf requires at least version 6.0.
Aborting.
```

This indicated that MongoDB was version 6.0, but its internal `featureCompatibilityVersion` (FCV) was still set to 5.0.

10.5 Fixing MongoDB Configuration

10.5.1 Step 1: Enter Mongo Shell

```
docker exec -it overleaf-docker-mongo-1 mongosh
```



png/restarting_problem_overleaf.png

10.5.2 Step 2: Initialize Replica Set

```
rs.initiate()
```

If you see `MongoServerError[AlreadyInitialized]`, it means it's already set up — that's fine.

10.5.3 Step 3: Update Feature Compatibility Version

```
use admin
db.adminCommand({ setFeatureCompatibilityVersion: "6.0" })
```

Expected output:

```
{ "ok" : 1 }
```

10.5.4 Step 4: Restart Overleaf

```
exit
docker restart overleaf-docker-overleaf-1
```

10.6 Making Overleaf Publicly Accessible

Since Bugzilla was already bound to port 80, Overleaf was assigned to port 8090. I confirmed the port was open:

```
ss -tuln | grep 8090
```

Then enabled it in the firewall:

```
sudo ufw allow 8090/tcp
sudo ufw reload
```

10.7 Access and Verification

Once restarted, Overleaf was reachable at:

```
http://174.138.68.199:8090
```

I verified it via:

```
curl -I http://174.138.68.199:8090
```

Result:

```
HTTP/1.1 200 OK
```

10.8 Conclusion on Setting up Overleaf

The deployment succeeded after:

1. Gaining SSH access by adding my GitHub SSH key to the droplet.
2. Running Overleaf on port 8090 (to avoid conflict with Bugzilla on port 80).
3. Initializing the MongoDB replica set.
4. Updating the MongoDB feature compatibility version to 6.0.

After these steps, Overleaf was fully functional and accessible publicly via:

<http://174.138.68.199:8090>

This was what was seen:

10.9 Setting up Overleaf Packages

Even though Overleaf is up and running, we weren't able to run our document due to Overleaf not having access to any of the packages we were using. Thus we had to download the packages needed with commands:

```
apt-get update
apt-get install -y texlive-full
mktexlsr
```

This took a long time to run, but afterwards our notebook was able to compile and had no errors in showing the pdf, however minted has refused to work even after installation... to fix the issue of packages corrupted or errors from minted I ran this command:

```
docker compose exec overleaf bash -lc '
set -e
export PATH=/usr/local/texlive/2025/bin/x86_64-linux:$PATH
tlmgr option repository https://mirror.ctan.org/systems/texlive/tlnet
tlmgr update --self
tlmgr install scheme-full
mktexlsr
'
```

this command forced the packages to be correctly installed in the correct path.

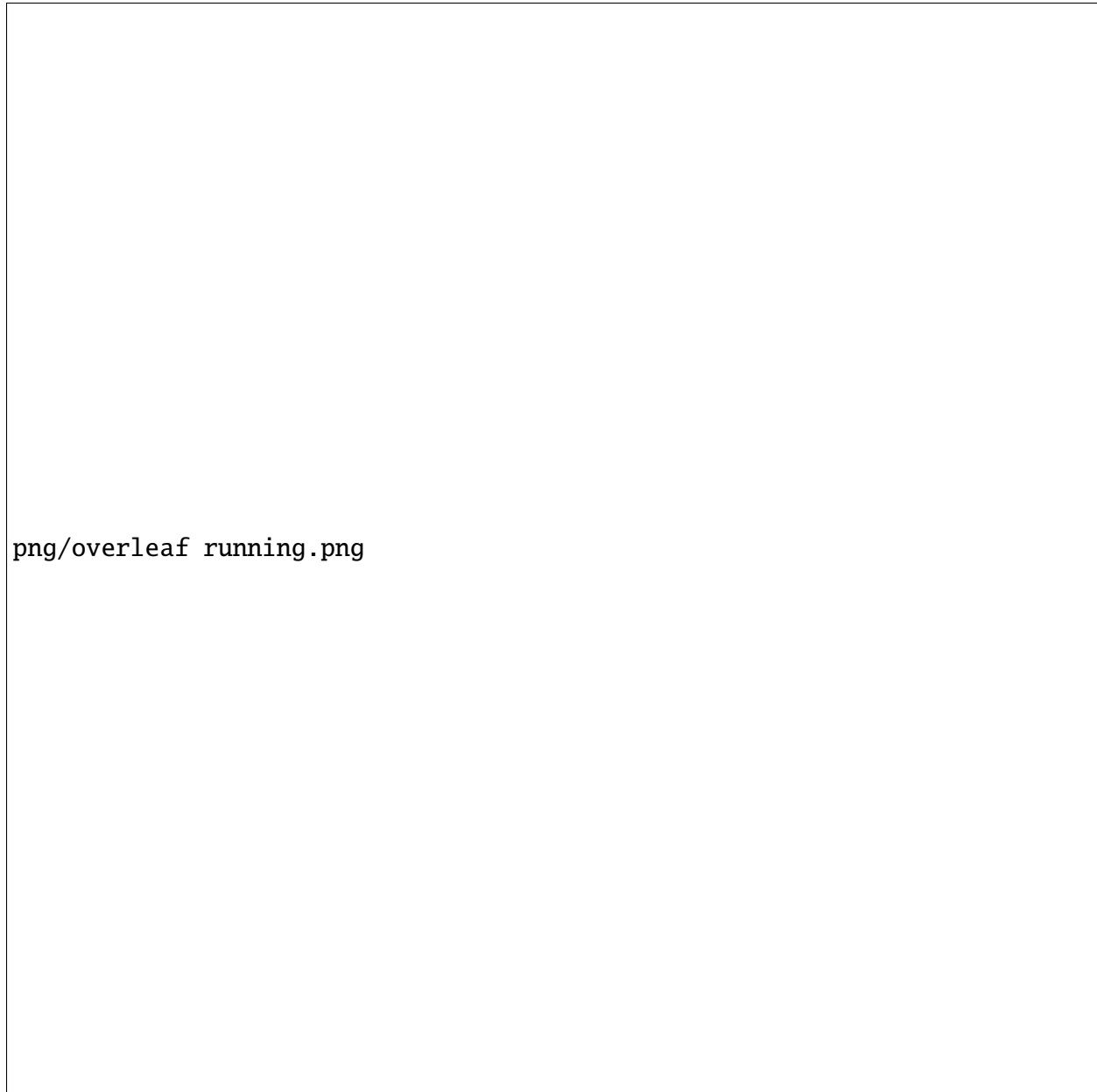
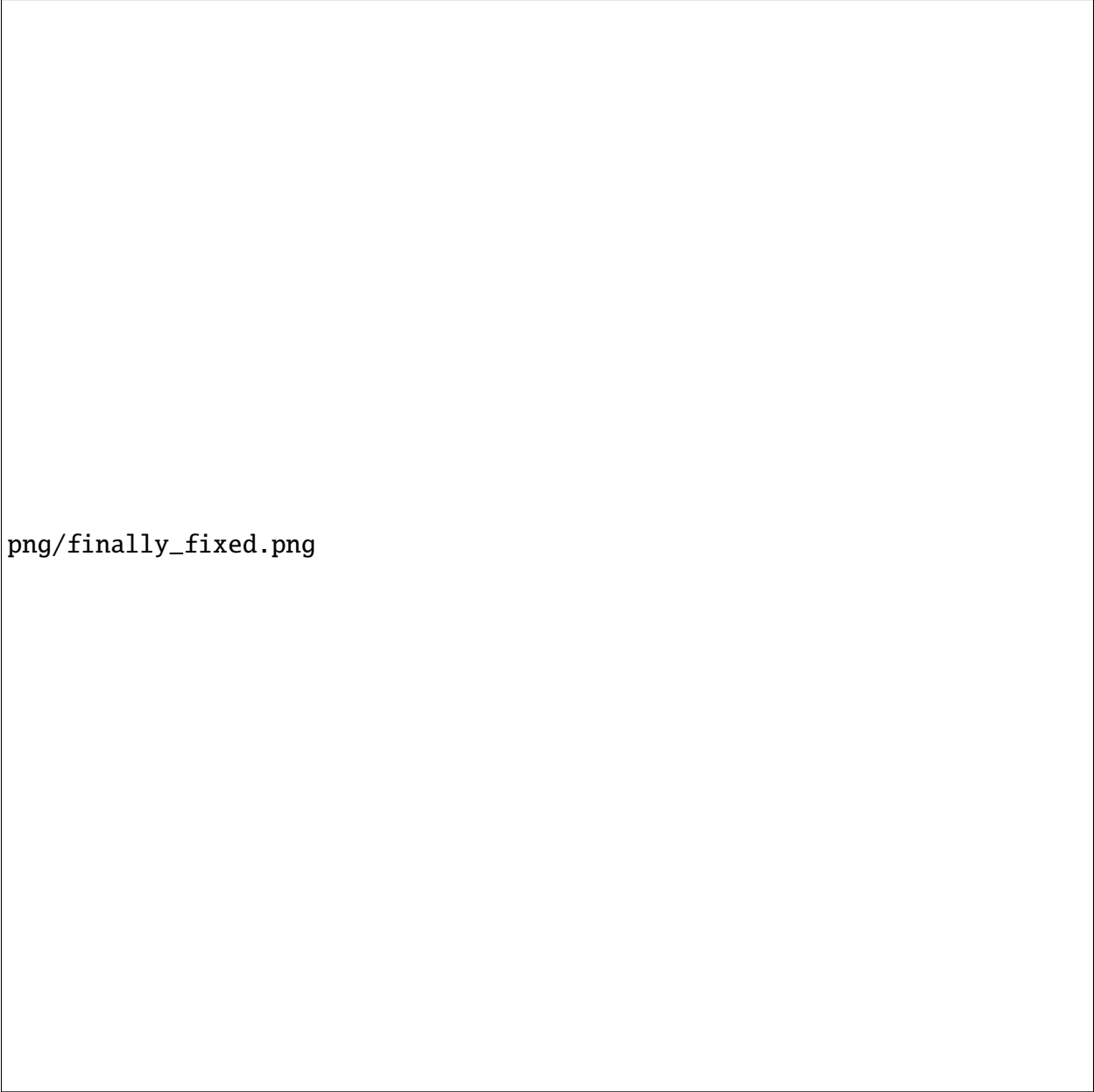


Figure 10.1: overleaf website on <http://174.138.68.199:8090>



png/finally_fixed.png

10.10 Domain Names, SSL

Lets discuss the full process of deploying an Overleaf instance with a custom domain and secure HTTPS access on a DigitalOcean droplet.

It includes:

- Creating and mapping a custom subdomain (overleaf-ssw590-team07.annanyajain.me).
- Configuring Nginx as a reverse proxy for the Overleaf Docker container.
- Obtaining and applying a Let's Encrypt SSL certificate.
- Debugging HTTPS access until the setup functioned securely.

10.11 Domain and Reverse Proxy Setup

10.11.1 DNS Configuration

A subdomain was created in the DNS settings for annanyajain.me with the following record:

```
Type: A
Host: overleaf-ssw590-team07
Value: 174.138.68.199
TTL: 3600
```

We verified DNS propagation:

```
dig overleaf-ssw590-team07.annanyajain.me +short
174.138.68.199
```

10.11.2 Overleaf Docker Setup

The Overleaf container was configured to listen on port 8090:

```
sudo docker ps --format "table {{.Names}}\t{{.Ports}}"
overleaf-docker-overleaf-1    0.0.0.0:8090->80/tcp
```

Verification:

```
curl -I http://localhost:8090/login
HTTP/1.1 302 Found
Location: /login
```

10.11.3 Nginx Reverse Proxy

Nginx was used to route traffic from the public domain to the Overleaf container. Configuration file path:

`/etc/nginx/sites-available/overleaf-https.conf`

File contents:

```
server {
    listen 80;
    server_name overleaf-ssw590-team07.annanyajain.me;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name overleaf-ssw590-team07.annanyajain.me;

    ssl_certificate /etc/letsencrypt/live/overleaf-ssw590-team07.annanyajain.me/
        ↪ fullchain.pem;
    ssl_certificate_key
        ↪ /etc/letsencrypt/live/overleaf-ssw590-team07.annanyajain.me/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
        proxy_pass http://localhost:8090;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Enabled and tested:

```
sudo ln -s /etc/nginx/sites-available/overleaf-https.conf
    ↪ /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

10.12 SSL Certificates, Firewall, and Debugging

10.12.1 Firewall Configuration

The firewall was configured to allow web and SSH access:

```
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw allow 8090/tcp
sudo ufw allow OpenSSH
sudo ufw enable
sudo ufw status
```

10.12.2 Obtaining an SSL Certificate

Once HTTP (port 80) was accessible, a Let's Encrypt certificate was obtained:

```
sudo certbot certonly --nginx -d overleaf-ssw590-team07.annanyajain.me
```

Certbot confirmed:

```
Successfully received certificate.
Certificate is saved at:
/etc/letsencrypt/live/overleaf-ssw590-team07.annanyajain.me/fullchain.pem
Key is saved at:
/etc/letsencrypt/live/overleaf-ssw590-team07.annanyajain.me/privkey.pem
```

10.12.3 Verification

Check certificate validity:

```
sudo openssl x509 -in
↳ /etc/letsencrypt/live/overleaf-ssw590-team07.annanyajain.me/fullchain.pem
↳ -noout -dates
notBefore=Oct 15 23:53:29 2025 GMT
notAfter=Jan 13 23:53:28 2026 GMT
```

Confirm ports are open:

```
sudo ss -tuln | grep -E ':80|:443'
tcp LISTEN 0 511 0.0.0.0:443 0.0.0.0:*
tcp LISTEN 0 4096 0.0.0.0:80 0.0.0.0:*
```

10.12.4 Debugging HTTPS Access

Initially, even with a valid Let's Encrypt certificate, HTTPS access failed. Local tests worked, but external connections hung during the TLS handshake:

```
curl -vk https://overleaf-ssw590-team07.annanyajain.me
# (Hangs during TLS handshake)
```

Diagnosis: Nginx was confirmed to be listening:

```
sudo ss -tuln | grep :443
tcp LISTEN 0 511 0.0.0.0:443 0.0.0.0:*
```

Thus, the issue was a blocked inbound port.

Fix: We explicitly reopened port 443:

```
sudo ufw allow 443/tcp
sudo ufw status
```

Verification: After reopening the port, HTTPS immediately worked:

```
curl -vk https://overleaf-ssw590-team07.annanyajain.me
# SSL connection using TLSv1.3 / AEAD-AES256-GCM-SHA384
# SSL certificate verify ok.
# HTTP/1.1 302 Found -> /login
```

External connection confirmed:

```
nc -vz overleaf-ssw590-team07.annanyajain.me 443
Connection to overleaf-ssw590-team07.annanyajain.me port 443 [tcp/https]
↪ succeeded!
```

10.12.5 Renewal

Automatic renewal is managed by Certbot's system timer. Manual renewal test:

```
sudo certbot renew --dry-run
```

10.12.6 Final Verification

Overleaf is now securely available at:

- **HTTPS (secure):** <https://overleaf-ssw590-team07.annanyajain.me>
- **HTTP (debug only):** <http://overleaf-ssw590-team07.annanyajain.me:81>

Results:

- Fully functional HTTPS via Let's Encrypt.
- Nginx reverse proxy routing traffic to Docker.
- Confirmed automatic certificate renewal.
- Firewall correctly configured and verified.

10.13 Connecting Our Overleaf Instance to GitHub

I first created a new empty GitHub Repository for Overleaf, and then connected to our droplet under where the Overleaf yaml file exists.

The workflow is as follows:

1. Overleaf CE is installed via Docker on a Droplet.
2. Overleaf project is connected to a GitHub repository (SSW590-team-7-Overleaf).
3. A GitHub Action builds the LaTeX document, adds version metadata, and commits the compiled PDF back to the repo automatically.

I first modified the Docker file:

```
1 services:
2   mongo:
3     image: mongo:6.0
4     restart: unless-stopped
5     command: ["--replSet", "rs0", "--bind_ip_all"]
6     volumes:
7       - mongo_data:/data/db
8
9   redis:
10    image: redis:7
11    restart: unless-stopped
12
13  overleaf:
14    image: sharelatex/sharelatex:latest
15    restart: unless-stopped
16    depends_on:
17      - mongo
18      - redis
19    ports:
20      - "8090:80"
21    environment:
22      - OVERLEAF_MONGO_URL=mongodb://mongo/sharelatex
23      - OVERLEAF_REDIS_HOST=redis
24      - OVERLEAF_MAX_UPLOAD_SIZE=50
25      - V1_HISTORY_URL=http://overleaf:3100/api
26    volumes:
27      - ./data:/var/lib/overleaf
28
29 volumes:
30   mongo_data:
```

and then started the Overleaf again with the new file and checked if it was running correctly:

```
docker compose up -d
docker compose ps
```

I created the new GitHub repo via their web interface, and then ran these commands to gain access and configure git identity:

```
git clone git@github.com:cascadingluo/SSW590-team-7-Overleaf.git
git config --global user.email "<email>"
git config --global user.name "<username>"
```

I committed to initialize the project:

```
git add .
git commit -m "Initial LaTeX project"
git push -u origin main
```

Then we created the workflow directory:

```
mkdir -p .github/workflows
```

and then created the workflow file:

```
1 name: Build PDF on push
2
3 on:
4   push:
5     branches: [ "main", "master" ]
6     workflow_dispatch:
7 permissions:
8   contents: write
9
10 jobs:
11   build:
12     runs-on: ubuntu-latest
13
14     steps:
15       - name: Checkout (full history)
16         uses: actions/checkout@v4
17         with:
18           fetch-depth: 0
19
20       - name: Create version.tex
21         id: ver
22         shell: bash
23         run: |
24           SHORT_SHA="$(git rev-parse --short HEAD)"
25           CL_NUM="$(git rev-list --count HEAD)"
26           RUN_NUM="${GITHUB_RUN_NUMBER}"
27           DATE="$(date -u +'%Y-%m-%d')"
28           {
29             echo "short_sha=$SHORT_SHA"
30             echo "cl_num=$CL_NUM"
```

```

31     echo "run_num=$RUN_NUM"
32     echo "date=$DATE"
33 } >> "$GITHUB_OUTPUT"
34
35 printf '%% Auto-generated by GitHub Actions\n' > version.tex
36 printf '\\newcommand{\\bulddate}{%s}\n' "$DATE" >> version.tex
37 printf '\\newcommand{\\changeset}{%s}\n' "$SHORT_SHA" >> version.tex
38 printf '\\newcommand{\\changelist}{%s}\n' "$CL_NUM" >> version.tex
39 printf '\\newcommand{\\buildnum}{%s}\n' "$RUN_NUM" >> version.tex
40 printf '%% Example: Version \\buildnum{} (CL \\changelist{}, SHA
    ↪ \\changeset{}, \\bulddate{})\n' >> version.tex
41
42 - name: Move version.tex next to itManual.tex
43   run: |
44     cp version.tex "2025F_SSW590_07-3/"
45
46 - name: Build LaTeX
47   uses: xu-cheng/latex-action@v3
48   with:
49     working_directory: "2025F_SSW590_07-3"
50     root_file: itManual.tex
51     latexmk_use_xelatex: true
52     args: -norc -file-line-error -halt-on-error -interaction=nonstopmode
    ↪ -output-directory=. itManual.tex
53     latexmk_shell_escape: true
54     # pre_compile: |
55     #   # cp ../version.tex .
56     #   # apt-get update && apt-get install -y python3-pygments
57     #   # Uncomment and edit if you need extra packages
58     pre_compile: |
59       echo "PATH=$PATH"
60       tlmgr option repository https://mirror.ctan.org/systems/texlive/tlnet
61       tlmgr update --self
62       tlmgr install minted fvextra fancyvrb framed upquote lineno xcolor
    ↪ etoolbox siunitx titlesec caption pgfplots
63
64 - name: Save and commit PDF
65   run: |
66     mkdir -p builds
67
68 - name: Save and commit PDF
69   run: |
70     cp "2025F_SSW590_07-3/itManual.pdf" "builds/itManual_${{ github.sha
    ↪ }}.pdf"
71
72 - name: Upload PDF
73   uses: actions/upload-artifact@v4

```



```

74     with:
75         name: itManual
76         path: builds/itManual_${{ github.sha }}.pdf
77
78     - name: Set GitHub user config
79       run: |
80         git config user.name "github-actions[bot]"
81         git config user.email "github-actions[bot]@users.noreply.github.com"
82         git add builds/*.pdf version.tex
83         git commit -m "CI: build PDF CL=${{ steps.ver.outputs.changelist }}" ||
84           ↪ exit 0
85         git push
86
87     - name: Clean LaTeX build files
88       run: |
89         rm -f *.aux *.log *.out *.toc *.fdb_latexmk *.fls

```

After creating this file we linked it with overleaf and allowed it to auto create a new version every time we pushed into github.

In order to add versioning, we went into the itManual.tex file and added:

```
input{version.tex}
```

This allowed us to use the github variables inside the LaTeX document which we used in:

```
Version buildnum{} CL changelist{} SHA changeset{} and builddate{}
```

I tested the workflow by triggering the action:

```

git add .
git commit -m "Test Overleaf GitHub build"
git pull --rebase origin main
git push

```

We created a file for syncing github with our server overleaf:

```

#!/bin/bash
# === Configuration ===
CONTAINER_NAME="overleaf-docker-overleaf-1"
PROJECT_PATH="/var/lib/overleaf/data/compiles/68eabaec5adfde60821a1298-68eab97f5_
↪ 34beb493f28d48a"
GITHUB_REPO_PATH="/root/overleaf-docker/ssw590-overleaf"
TARGET_SUBFOLDER="2025F_SSW590_07-3"

# === Sync Overleaf project into GitHub repo ===
echo "Copying project from container to GitHub repo..."
docker cp "${CONTAINER_NAME}:${PROJECT_PATH}/."
↪ "${GITHUB_REPO_PATH}/${TARGET_SUBFOLDER}/"

```

```
cd "$GITHUB_REPO_PATH" || exit

# === Commit and push ===
git add .
git commit -m "Auto-sync from Overleaf container on $(date -u +%Y-%m-%d
↪ %H:%M:%S')" || true
git push origin main
```

Whenever we update the files on our Overleaf, we run:

```
./sync_overleaf_to_github.sh
```

to sync all of our changes.

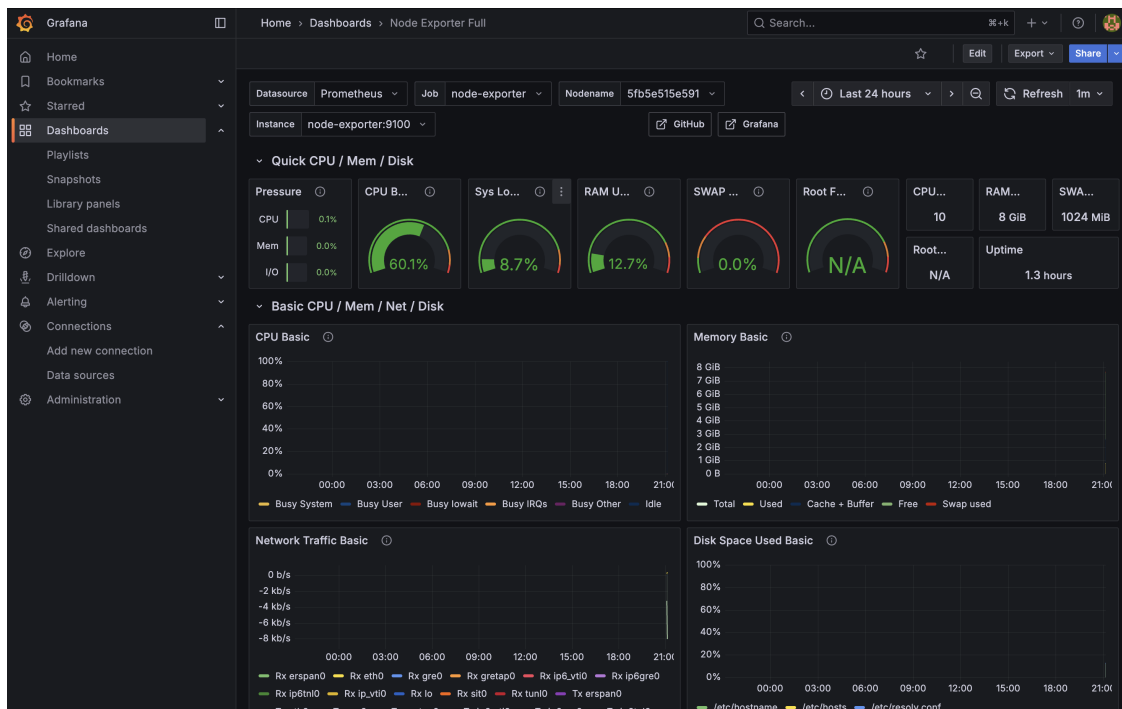
Chapter 11

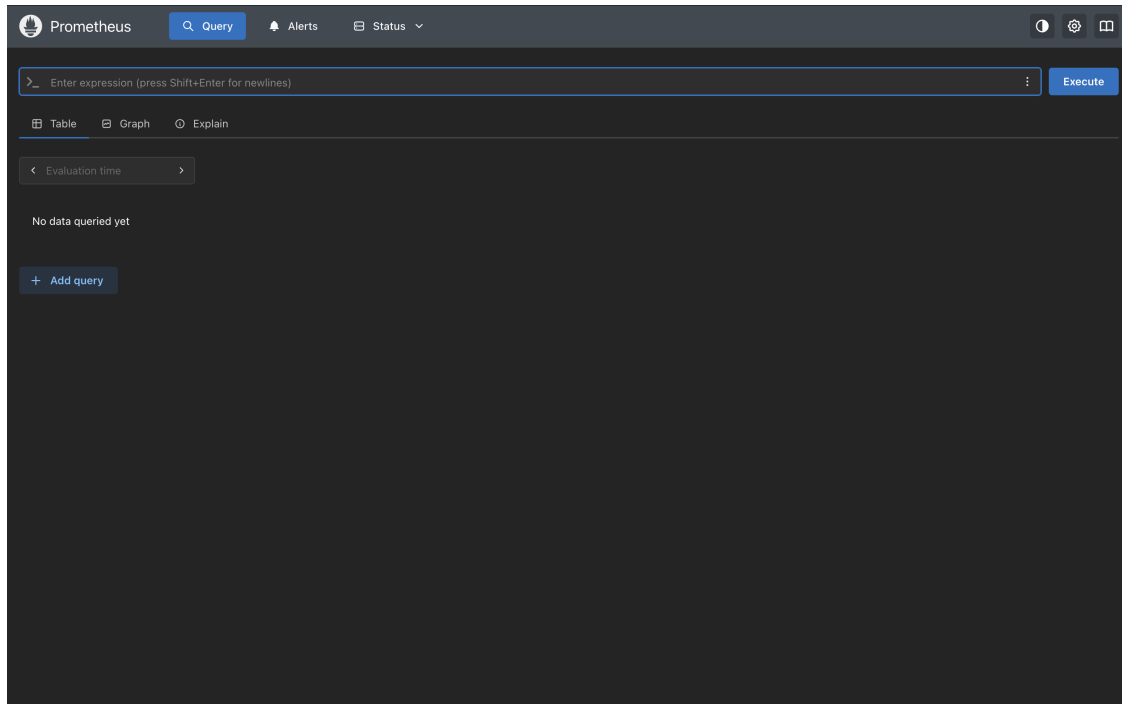
Prometheus and Grafana Monitoring Setup

– Annanya Jain, Luo Xu, Gavin Lam

11.1 Introduction

This chapter documents our setup and configuration of a monitoring stack using **Prometheus**, **Grafana**, and **Node Exporter**. The objective was to collect, visualize, and analyze system-level metrics such as CPU usage, memory utilization, disk I/O, and network performance.





11.2 The role of Prometheus, Grafana, and Node Exporter

Prometheus is an open-source monitoring system that collects and stores time-series data from different sources, such as servers and applications. Node Exporter provides Prometheus with detailed system-level metrics like CPU usage, memory consumption, disk activity, and network performance. The Grafana dashboard visualizes key system health metrics collected by **Node Exporter** and stored by **Prometheus**. It provides a real-time overview of CPU, memory, disk, and network performance, allowing quick assessment of system status.

What Dashboard 1860 tells you about system health?

The top row displays a summary of **CPU**, **Memory**, and **Disk** utilization. In the captured dashboard, CPU usage is approximately **60%**, memory usage around **12.7%**, and swap usage is **0%**, indicating stable performance with no significant resource pressure. Additional panels show detailed metrics:

- **CPU Basic:** Tracks CPU activity over time and highlights user, system, and idle states.
- **Memory Basic:** Displays total, used, and free memory to detect potential memory issues.
- **Network Traffic:** Monitors data transmitted and received per network interface.
- **Disk Space Used:** Shows disk utilization across mounted file systems.
- **System Uptime:** Indicates total runtime of the node (about 1.3 hours in this case).

Overall, this dashboard provides an effective visualization of system performance and helps identify resource bottlenecks or irregular activity.

11.3 Project Structure

The following directory structure was used to organize configuration files and Docker services:

```
PrometheusGrafana/  
  
docker-compose.yml  
prometheus/  
  prometheus.yml  
  rules.yml  
grafana/  
  provisioning/  
    datasources/  
      datasource.yml
```

11.4 Docker Compose Configuration

The monitoring stack consisted of three services: **Prometheus**, **Grafana**, and **Node Exporter**. These were defined in `docker-compose.yml` as follows:

```
version: '3.8'  
  
services:  
  prometheus:  
    image: prom/prometheus:latest  
    container_name: prometheus  
    volumes:  
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml  
      - ./prometheus/rules.yml:/etc/prometheus/rules.yml  
    ports:  
      - "9091:9090"  
  
  grafana:  
    image: grafana/grafana:latest  
    container_name: grafana  
    ports:  
      - "3000:3000"  
    volumes:  
      - ./grafana/provisioning:/etc/grafana/provisioning  
    environment:  
      - GF_SECURITY_ADMIN_USER=admin  
      - GF_SECURITY_ADMIN_PASSWORD=admin  
    depends_on:  
      - prometheus  
  
  node-exporter:
```

```
image: prom/node-exporter:latest
container_name: node-exporter
ports:
  - "9100:9100"
```

In this configuration:

- Prometheus runs on port 9091 (mapped from container port 9090).
- Grafana runs on port 3000.
- Node Exporter exposes system metrics on port 9100.

11.5 Prometheus Configuration

The `prometheus.yml` file defines global settings, scrape intervals, and targets for metric collection:

```
global:
  scrape_interval: 15s

rule_files:
  - "rules.yml"

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node-exporter'
    static_configs:
      - targets: ['node-exporter:9100']
```

This configuration instructs Prometheus to:

- Collect data every 15 seconds.
- Scrape metrics from itself on `localhost:9090`.
- Scrape metrics from Node Exporter at `node-exporter:9100`.

11.6 Alerting Rules

A rules file named `rules.yml` was created to define alerting conditions. This also resolved an earlier configuration error when Prometheus tried to load an empty directory as a rule file.

```
groups:
  - name: example
    rules:
      - alert: InstanceDown
        expr: up == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Instance {{ $labels.instance }} down"
```

This rule triggers an alert if any monitored instance is down for more than one minute.

11.7 Grafana Data Source Configuration

Grafana was configured to use Prometheus as its default data source using the following file: `grafana/provisioning/datasources/datasource.yml`

```
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9091
    isDefault: true
```

11.8 Launching the Monitoring Stack

The monitoring setup was launched using Docker Compose:

```
docker compose up -d
```

Once the containers were running, the following endpoints were verified:

- Prometheus: <http://localhost:9091>
- Grafana: <http://localhost:3000>
- Node Exporter: <http://localhost:9100>

11.9 Grafana Dashboard Setup

After logging into Grafana using the default credentials (admin/admin), the **Node Exporter Full Dashboard (ID 1860)** was imported:

1. Go to + → Import.
2. Enter Dashboard ID 1860 and click Load.
3. Select Prometheus as the data source.
4. Click Import.

Auto-refresh was set to 10 seconds, and the time range was configured to display the last 5 minutes.

11.10 Outcome and Observations

After adding the rule file and restarting the containers, Prometheus began successfully scraping data from Node Exporter. The Grafana dashboard displayed real-time metrics such as:

- CPU usage and load averages
- Memory and swap utilization
- Disk I/O statistics
- Network traffic rates

The system provided clear visibility into server performance and resource utilization.

Appendix A

Appendix

– *Gavin Lam, Spurthi Setty, Annanya Jain, Luo Xu*

Bibliography

Index

appendix, [64](#)

AWS, [16](#)

Bugzilla, [28](#)

Chapter

Appendix, [64](#)

AWS, [16](#)

Bugzilla, [28](#)

Hosts, [2](#)

Kanban Setup, [3](#)

LaTeX Docker, [24](#)

Linux Commands, [7](#)

Overleaf, [33](#), [41](#)

Passwords, [1](#)

Project Proposal, [15](#)

PrometheusGrafana, [58](#)

Hosts, [2](#)

Kanban Setup, [3](#)

LaTeX Docker, [24](#)

Linux Commands, [7](#)

Overleaf, [33](#), [41](#), [58](#)

Passwords, [1](#)

Project Proposal, [15](#)