DevOps - Group 14 Assignments

Document Version: Unknown

by

Andre Santiago-Neyra
Badri Narayanan Rajendran

Stevens.edu

October 2, 2025

DevOps - Group 14 Assignments

Andre Santiago-Neyra, Badri Narayanan Rajendran
Stevens.edu

This document provides the group assignments completed by Group 14 for the DevOps course. The following table (Table 1) should be updated by authors whenever major changes are made to the document or new assignments 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**
11/11/2025	Badri: • Assignment 10: Load Balancer and Virtual Hosts (Chapter 9).
11/04/2025	Andre: • Assignment 9: Jenkins with Pytest (Chapter 8).
10/27/2025	Andre, Badri: • Assignment 8: Setting up Prometheus And Grafana Dashboards and Node Exporter in DigitalOcean (Chapter 7).
10/20/2025	Andre, Badri: • Assignment 6: Overleaf CI with GitHub Actions (Chapter 6).
10/20/2025	Andre, Badri: • Assignment 5: Overleaf deployment and verification (Chapter 5).
10/05/2025	Andre: • Assignment 4: Bugzilla and Overleaf Dockers on Digital Ocean (Chapter 4).
10/03/2025	Andre, Badri: • Assignment 3: Docker on AWS created (Chapter 3).
09/24/2025	Andre: • Assignment 2: Project Proposal created (Chapter 2).
09/17/2025	Badri: • Assignment 1: Linux Commands created (Chapter 1).
09/12/2025	Andre, Badri, Nozanin: • Initial document creation with template structure.

Passwords and Service Hints

This page contains password hints for various services used throughout the assignments. These hints are for reference only and should be updated as credentials change.

Service	**Password Hint**
Bugzilla Admin	Default password set during deployment (check deployment notes)
MariaDB Root	Password specified in docker-compose environment variables
MariaDB Bugzilla User	Password specified in docker-compose environment variables
Overleaf Admin	Created via command line during setup
DigitalOcean Droplet	SSH key authentication (root user)
AWS Account	SSO credentials via Stevens portal
GitHub	Personal access token for repository access
Docker Hub	Optional - for private image repositories

Table 2: Service Password Hints

Contents

List of Tables

List of Figures

Chapter 1

Linux Commands

This chapter is about understanding and executing Linux commands and some exercises.

1.1 Navigation & File Ops

1. Show your present working directory path only.
   - Input: `pwd`
   - Output: `/Users/badrinarayanan/lx-test`
2. List all entries in the current directory, one per line, including dotfiles.
   - Input: `ls -A1`
   - Output:

     ```
     archive
     blob.bin
     cs.txt
     link-to-file1
     old.txt
     people.csv
     src
     sys.log
     tmp
     words.txt
     ```
3. Copy `src/file1.txt` to `tmp/` only if tmp exists; do it verbosely.
   - Input: `[ -d tmp ] && cp -v src/file1.txt tmp/`
   - Output: `src/file1.txt -> tmp/file1.txt`
4. Move `old.txt` into `archive/` and keep its original timestamp.
   - Input: `mv -v old.txt archive/`
   - Output: `old.txt -> archive/old.txt`
5. Create a new empty file `notes.md` only if it doesn't already exist.

- Input: `[ ! -e notes.md ] && touch notes.md`
- Output: `notes.md` file created in lx-test folder.

6. Show disk usage (human-readable) for the src directory only (not total FS).
   - Input: `du -sh src`
   - Output: `8.0K src`

1.2 Viewing & Searching

1. Print line numbers while displaying `sys.log`.
   - Input: `nl sys.log`
   - Output:

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

2. Show only the lines in `sys.log` that contain ERROR (case-sensitive).
   - Input: `grep 'ERROR' sys.log`
   - Output: `ERROR fan fail`
3. Count how many distinct words appear in `words.txt` (case-insensitive).
   - Input: `tr '[:upper:]' '[:lower:]' < words.txt | uniq | wc -w`
   - Output: `4`
4. From `words.txt`, show lines that start with g or G.
   - Input: `grep -E '^(g|G)' words.txt`
   - Output:

```
Gamma
gamma
```

5. Display the first 2 lines of `people.csv` without using an editor.
   - Input: `head -n 2 people.csv`
   - Output:

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

6. Show the last 3 lines of `sys.log` and keep following if the file grows.
   - Input: `tail -n 3 -f sys.log`
   - Output:

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

1.3 Text Processing

1. From `people.csv`, print only the name column (2nd), excluding the header.
   - Input: `awk -F',' 'NR>1 {print $2}' people.csv`
   - Output:

     ```
     Ada
     Linus
     Grace
     Dennis
     ```

2. Sort `words.txt` case-insensitively and remove duplicates.
   - Input: `sort -f words.txt | uniq`
   - Output:

     ```
     alpha
     beta
     gamma
     Gamma
     ```

3. Replace every three with 3 in all files under `src/` in-place, creating .bak backups.
   - Input: `find src -type f -exec sed -i.bak 's/three/3/g' {} +`
   - Output: one two 3 four in file1.txt
     two 3 four five in file2.txt
     file1.txt.bak and file2.txt.bak created with original data.

4. Print the number of lines, words, and bytes 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
     ```

1.4 Permissions & Ownership

1. Make `tmp/` readable, writable, and searchable only by the owner.
   - Input: `chmod 700 tmp`
   - Output: directory permission changed.

2. Give group execute permission to `src/lib` recursively without touching others/owner bits.
   - Input: `find src/lib -type d -exec chmod g+x {} +`
   - Output: folder permissions changed.

3. Show the numeric (octal) permissions of `src/file2.txt`.
   - Input: `stat -f '%Lp' src/file2.txt`
   - Output: `644`
4. Make `notes.md` append-only for the owner via file attributes (if supported).
   - Input: `sudo chflags uchg notes.md`
   - Output: file permissions changed

1.5 Links & Find

1. Verify whether `link-to-file1` is a symlink and show its target path.
   - Input: `readlink -f link-to-file1`
   - Output: `/Users/badrinarayanan/lx-test/src/file1.txt`
2. Find all regular files under the current tree larger than 40 KiB.
   - Input: `find . -type f -size +40k`
   - Output: `./blob.bin`
3. Find files modified in the last 10 minutes under `tmp/` and print their sizes.
   - Input: `find tmp -type f -mmin -10 -exec ls -lh {} + | awk '{print $5, $9}'`
   - Output: `24B tmp/file1.txt`

1.6 Processes & Job Control

1. Show your processes in a tree view.
   - Input: `ps -ejh`
   - Output: printed all the processes in a tree manner.
2. Start `sleep 120` in the background and show its PID.
   - Input: `sleep 120 & echo $!`
   - Output:

     ```
     12425
     12425
     ```
3. Send a TERM signal to all sleep processes owned by you (don't use kill -9).
   - Input: `pkill -u $(whoami) -x sleep`
   - Output: `[1] + terminated sleep 120`
4. Show the top 5 processes by memory usage (one-shot, not interactive).
   - Input: `ps aux | sort -nrk 4 | head -n 6`
   - Output: top 5 processes shown.

1.7 Archiving & Compression

1. Create a gzipped tar archive `src.tgz` from `src/` with relative paths.
   - Input: `tar czf src.tgz -C src .`
   - Output: tar file created.
2. List the contents of `src.tgz` without extracting.
   - Input: `tar tzf src.tgz`
   - Output:

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

3. Extract only `file2.txt` from `src.tgz` into `tmp/`.
   - Input: `tar xzf src.tgz -C tmp file2.txt`
   - Output: Extracted only file2.txt into tmp.

1.8 Networking & System Info

1. Show all listening TCP sockets with associated PIDs (no root assumptions).
   - Input: `netstat -anp tcp | grep LISTEN`
   - Output:

```
tcp6 0 0 *.51050 *.* LISTEN
tcp4 0 0 *.51050 *.* LISTEN
tcp6 0 0 *.5000 *.* LISTEN
tcp4 0 0 *.5000 *.* LISTEN
tcp6 0 0 *.7000 *.* LISTEN
tcp4 0 0 *.7000 *.* LISTEN
```

2. Print your default route (gateway) in a concise form.
   - Input: `netstat -rn | grep '^default' | awk '{print $2}'`
   - Output:

```
10.0.0.1
fe80::12e1:77ff:fe1c:3e48fe80::fe80::fe80::fe80::
```

3. Display kernel name, release, and machine architecture.
   - Input: `uname -srm`
   - Output: `Darwin 24.6.0 arm64`

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

   - Input: `last -f head -n 5`
   - Output:

     ```
     badrinarayanan ttys000 Tue Sep 16 20:56 still logged in
     badrinarayanan console Tue Sep 16 11:55 still logged in
     ```

1.9 Package & Services (Debian/Ubuntu)

1. Show the installed version of package coreutils.

   - Input: `brew list --versions coreutils`
   - Output: lists the versions of coreutils installed.

2. Search available packages whose names contain ripgrep.

   - Input: `brew search ripgrep`
   - Output:

     ```
     ==> Formulae
     ripgrep
     ripgrep-all
     ==> Casks
     ripme
     ```

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

   - Input: `ps aux | grep [c]ron`
   - Output: `no matches found: [c]ron`

1.10 Bash & Scripting

1. Write a one-liner that loops over *.txt in `src/` and prints: `<filename>:<number of lines>:<number of words>`.

   - Input: `for f in src/*.txt; do echo "$f:$(wc -l < "$f"):$(wc -w < "$f")"; done`
   - Output:

     ```
     src/file1.txt: 1: 4
     src/file2.txt: 1: 4
     ```

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

   - Input: `awk -F',' 'NR>1 && $3=="CS"' people.csv > cs.txt`
   - Output:

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

3. 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

Project Proposal – Group 14

2.1 Project Title

Automated Blog Deployment Pipeline

2.2 Project Description

This project will develop a simple, containerized blog application and automate its build, test, and deployment processes using two CI/CD tools: Jenkins and Github Actions. The application will be hosted on AWS EC2 using Docker. An operational dashboard, built with monitoring tools, like Grafana, Datadog or Dynatrace, for effectively monitoring of application health and logs.

2.3 Simple Tasks

- Develop the blog app using Node.js or Python
- Containerize the app using Docker
- Integrate source control with GitHub
- Setup CI/CD with Jenkins and Github Actions for code build, test, and deploy
- Host on AWS using Docker and ECS
- Design a dashboard for build and deployment status

2.4 DevSecOps Tools

- Programming Language: Python or Node.js
- Back-End Framework: Flask or Express
- Database: PostgreSQL or MongoDB
- Source Control: GitHub

- CI/CD: Jenkins, Github Actions
- Deployment: Docker, AWS EC2, ECS
- Monitoring: Grafana or Datadog or Dynatrace or similar monitoring tool

Chapter 3

AWS Deployment

3.1 Overview

This chapter documents the deployment of the two-button color website to AWS using Docker and Amazon Elastic Container Registry (ECR). The deployment follows the steps outlined in Chapter 3 of the AWS guide.

3.2 Set Environment Variables

Edit these to match your setup (region, repo name, etc.):

```
# >>> EDIT THESE <
export AWS_REGION=us-east-1
export ECR_REPO=myapp
export IMAGE_TAG=v1
export CONTAINER_PORT=3000 # Port your app listens on

# Discover your AWS Account ID from your SSO session:
export AWS_ACCOUNT_ID="$(aws sts get-caller-identity --query Account --output text --
```

3.3 Create (or Reuse) an ECR Repository

Idempotently create the repo in your chosen region:

```
aws ecr describe-repositories \
  --repository-names "$ECR_REPO" \
  --region "$AWS_REGION" --profile default >/dev/null 2>&1 || \
aws ecr create-repository \
  --repository-name "$ECR_REPO" \
  --image-scanning-configuration scanOnPush=true \
  --region "$AWS_REGION" --profile default
```

3.4 Authenticate Docker to ECR

Use the AWS CLI to obtain a short-lived registry token and log 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"
```

3.5 Build, Tag, and Push Your Local Image

Run these from the directory containing your Dockerfile:

```
# Build your local image
docker build --platform linux/amd64 -t "$ECR_REPO:$IMAGE_TAG" .

# Tag it for ECR
docker tag "$ECR_REPO:$IMAGE_TAG" \
  "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPO:$IMAGE_TAG"

# Push to ECR
docker push "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPO:$IMAGE_TAG"
```

3.6 Verify the Image in ECR

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

3.7 Quick Deploy with App Runner

Spin up a managed HTTPS service directly from your ECR image:

```
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/$ECR_R
      \"ImageRepositoryType\": \"ECR\",
      \"ImageConfiguration\": {\"Port\": \"$CONTAINER_PORT\"}
    },
    \"AutoDeploymentsEnabled\": true
  }" \
  --instance-configuration "{\"Cpu\":\"1 vCPU\",\"Memory\":\"2 GB\"}"
```

Set the following variables for your own configuration:

```
AWS_REGION=us-east-2
PROFILE=AdministratorAccess-539272219501
SERVICE_ARN=arn:aws:apprunner:us-east-2:539272219501:service/my-apprunner-app/04c50b9
```

Test that the service status changes to running:

```
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
```

When status changes to running, get the service URL:

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

3.8 Deployment Results

The application was successfully deployed to AWS App Runner. Screenshots of the deployed website are shown below.

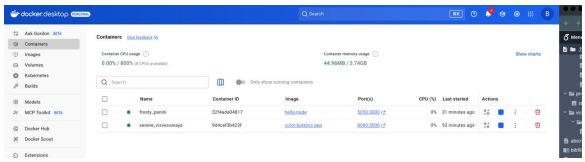

Figure 3.1: AWS Image 1

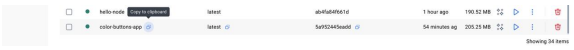

Figure 3.2: AWS Image 2

Figure 3.3: AWS Image 3

Figure 3.4: AWS Image 4

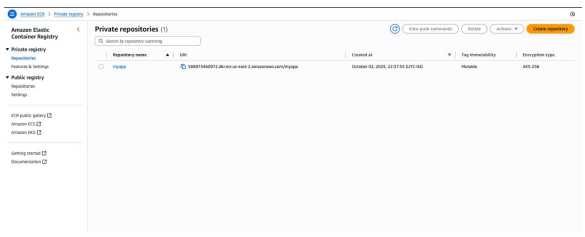

Figure 3.5: AWS Image 5

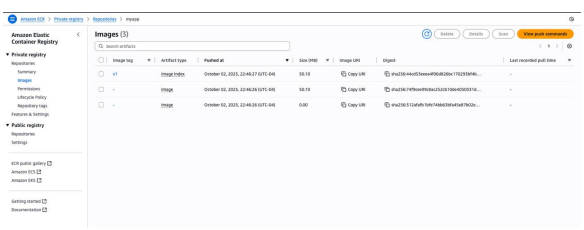

Figure 3.6: AWS Image 6

3.9 Class-Based JavaScript Implementation

The website was updated to use a class-based approach for better code organization.

3.9.1 Updated HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Color Buttons App</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin-top: 50px;
      transition: background-color 0.3s ease;
    }
    button {
      padding: 12px 24px;
      font-size: 18px;
      margin: 10px;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <h1>Click a Button to Change Background</h1>
  <button id="blueBtn">Blue</button>
  <button id="redBtn">Red</button>
  <script src="ColorChanger.js"></script>
  <script>
    const colorChanger = new ColorChanger();
    colorChanger.initialize();
  </script>
</body>
</html>
```

3.9.2 ColorChanger Class

```
class ColorChanger {
  constructor() {
    this.body = document.body;
  }

  changeColor(color) {
    this.body.style.backgroundColor = color;
  }
```

```
  attachEventListeners() {
    document.getElementById("blueBtn").addEventListener("click", () => {
      this.changeColor("blue");
    });

    document.getElementById("redBtn").addEventListener("click", () => {
      this.changeColor("red");
    });
  }

  initialize() {
    this.attachEventListeners();
  }
}
```

3.9.3 UML Class Diagram

```
+-------------------+
|   ColorChanger    |
+-------------------+
| - body: Element   |
+-------------------+
| + constructor()   |
| + changeColor(    |
|     color: string)|
| + attachEvent     |
|     Listeners()   |
| + initialize()    |
+-------------------+
```

3.10 Latex Docker

This chapter demonstrates creating a Docker container to compile LaTeX documents using TeX Live, replicating the functionality of Overleaf locally.

3.11 Project Structure

```
latex-docker/
  Dockerfile
  sample.tex
  compile.sh
```

3.12 Sample LaTeX Document

Create a simple LaTeX document to test compilation:

```
% sample.tex
```

```
\documentclass{article}
\usepackage[utf8]{inputenc}

\title{Docker LaTeX Test}
\author{Group 14}
\date{\today}

\begin{document}
\maketitle

\section{Introduction}
This document was compiled using Docker and TeX Live.

\section{Conclusion}
LaTeX compilation in Docker works successfully.

\end{document}
```

3.13 Dockerfile

Create the Dockerfile for the TeX Live environment:

```
FROM texlive/texlive:latest

WORKDIR /workspace

# Copy compilation script
COPY compile.sh /usr/local/bin/compile.sh
RUN chmod +x /usr/local/bin/compile.sh

# Set default command
CMD ["/bin/bash"]
```

3.14 Compilation Script

Create a helper script to compile LaTeX files:

```
#!/bin/bash
# compile.sh

if [ -z "$1" ]; then
  echo "Usage: compile.sh <filename.tex>"
  exit 1
fi

pdflatex -interaction=nonstopmode "$1"
echo "Compilation complete: ${1%.tex}.pdf"
```

3.15 Build the Docker Image

Build the Docker image with TeX Live:

```
docker build -t latex-compiler .
```

3.16 Compile a LaTeX Document

Run the container to compile a LaTeX file:

```
docker run --rm -v $(pwd):/workspace latex-compiler \
  compile.sh sample.tex
```

3.17 Alternative: Interactive Mode

For interactive compilation and debugging:

```
docker run -it --rm -v $(pwd):/workspace latex-compiler
# Inside container:
pdflatex sample.tex
```

3.18 Results

The Docker container successfully compiles LaTeX documents, producing PDF output files. This provides a portable, consistent LaTeX environment without requiring local TeX Live installation.

Chapter 4

DigitalOcean Setup

4.1 Bugzilla Details

For this assignment, I deployed Bugzilla on a DigitalOcean Droplet using Docker containers. The setup required two separate containers: one for the MariaDB database and one for the Bugzilla application itself.

4.1.1 Docker Images Used

- **Bugzilla:** `nasqueron/bugzilla`
- **Database:** `mariadb:10.6`

4.1.2 Setup Process

Step 1: Create DigitalOcean Droplet

I created a new Droplet on DigitalOcean with the following specifications:

- Operating System: Ubuntu 22.04 LTS
- Plan: Basic ($4–$6/month)
- SSH access enabled

Step 2: Install Docker

After connecting to the Droplet via SSH, I installed Docker and Docker Compose:

```
apt update && apt install -y docker.io docker-compose
systemctl enable docker && systemctl start docker
```

Step 3: Create Docker Network

I created a dedicated Docker network to allow the containers to communicate:

```
docker network create bugzilla-net
```

Step 4: Deploy MariaDB Container

I started the MariaDB database container with the following configuration:

```
docker run -d \
  --name mariadb-bugzilla \
  --network bugzilla-net \
  -e MYSQL_ROOT_PASSWORD=rootpass \
  -e MYSQL_USER=bugzilla \
  -e MYSQL_PASSWORD=bugpass \
  -e MYSQL_DATABASE=bugs \
  mariadb:10.6
```

Step 5: Deploy Bugzilla Container

After the database was running, I deployed the Bugzilla container:

```
docker run -d \
  --name bugzilla \
  --network bugzilla-net \
  -p 80:80 \
  -e DB_HOST=mariadb-bugzilla \
  -e DB_DATABASE=bugs \
  -e DB_USER=bugzilla \
  -e DB_PASSWORD=bugpass \
  -e BUGZILLA_URL=http://159.89.43.12 \
  nasqueron/bugzilla
```

4.1.3 Configuration Parameters

- **Port Mapping:** 80:80 (host:container)
- **Network:** Custom bridge network (`bugzilla-net`)
- **Database Connection:** MariaDB via Docker network
- **Environment Variables:** Database credentials and Bugzilla URL configured via `-e` flags

4.1.4 Troubleshooting

During setup, I encountered an issue where the Bugzilla container kept exiting. The logs revealed missing environment variables (`DB_PASSWORD` and `DB_DATABASE`). This was resolved by ensuring all required environment variables were properly specified in the `docker run` command.

4.1.5 Access URL

The Bugzilla application is accessible at:
http://159.89.43.12

4.1.6 Verification

To verify both containers were running correctly, I used:

```
docker ps
```

Both `mariadb-bugzilla` and `bugzilla` containers showed status "Up" after successful deployment.

4.2 Overleaf Details

I deployed an Overleaf (ShareLaTeX) instance on a dedicated DigitalOcean Droplet using Docker Compose. Overleaf is a collaborative online LaTeX editor that allows real-time document editing and requires MongoDB and Redis as dependencies.

4.2.1 Docker Images Used

- **Overleaf:** `sharelatex/sharelatex:latest`
- **Database:** `mongo:6.0`
- **Cache:** `redis:6.2`

4.2.2 Setup Process

Step 1: Create Dedicated DigitalOcean Droplet

I created a new dedicated Droplet on DigitalOcean with the following specifications:

- Operating System: Ubuntu 22.04 LTS
- RAM: At least 2GB (recommended for Overleaf)
- Plan: Basic ($12/month)
- SSH access enabled

Step 2: Install Docker and Docker Compose

After connecting to the Droplet via SSH, I installed Docker and Docker Compose:

```
apt update && apt install -y docker.io docker-compose
systemctl enable docker && systemctl start docker
```

Step 3: Create Docker Compose Configuration

I created a directory for Overleaf and set up a Docker Compose file:

```
mkdir ~/overleaf
cd ~/overleaf
nano docker-compose.yml
```

Step 4: Docker Compose Configuration

The `docker-compose.yml` file contains three services: MongoDB (with replica set), Redis, and Overleaf (ShareLaTeX):

```
version: '3.8'
services:
  mongo:
    image: mongo:6.0
    container_name: mongo
    command: ["--replSet", "overleaf"]
    restart: always
    volumes:
      - /root/mongo_data:/data/db
    expose:
      - 27017
    healthcheck:
      test: ["CMD", "mongosh", "--eval",
             "db.adminCommand('ping')"]
      interval: 10s
      timeout: 5s
      retries: 10
    networks:
      - overleaf-net

  redis:
    image: redis:6.2
    container_name: redis
    restart: always
    volumes:
      - /root/redis_data:/data
    expose:
      - 6379
    networks:
      - overleaf-net

  sharelatex:
    image: sharelatex/sharelatex
    container_name: sharelatex
    restart: always
    depends_on:
      mongo:
        condition: service_healthy
      redis:
        condition: service_started
    ports:
      - "80:80"
    volumes:
      - /root/sharelatex_data:/var/lib/overleaf
```

```
    environment:
      OVERLEAF_APP_NAME: Overleaf Community Edition
      OVERLEAF_MONGO_URL: mongodb://mongo:27017/sharelatex?replicaSet=overleaf
      OVERLEAF_REDIS_HOST: redis
      ENABLED_LINKED_FILE_TYPES: project_file,project_output_file
      ENABLE_CONVERSIONS: 'true'
      EMAIL_CONFIRMATION_DISABLED: 'true'
      OVERLEAF_SITE_URL: http://<droplet_ip>
      OVERLEAF_NAV_TITLE: Overleaf CE
    networks:
      - overleaf-net

networks:
  overleaf-net:
```

Step 5: Initialize MongoDB Replica Set

After starting the containers, I initialized the MongoDB replica set manually:

```
docker exec -it mongo mongosh --eval "rs.initiate({
  _id: 'overleaf',
  members: [{ _id: 0, host: 'mongo:27017' }]
})"
```

Step 6: Deploy Overleaf

I started all three containers using Docker Compose:

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

Step 7: Configure Firewall

I ensured that port 80 was open in the firewall to allow external access:

```
ufw allow 80/tcp
ufw allow OpenSSH
ufw enable
```

4.2.3 Configuration Parameters

- **Port Mapping:** 80:80 (host:container)
- **Container Names:** `sharelatex`, `mongo`, `redis`
- **Persistent Storage:** Three bind mounts for Overleaf data (`/root/sharelatex_data`), MongoDB data (`/root/mongo_data`), and Redis data (`/root/redis_data`)
- **Network:** Custom bridge network (`overleaf-net`) for inter-container communication
- **Dependencies:** MongoDB 6.0 configured as a replica set for transaction support, and Redis 6.2 for session management

- **MongoDB Replica Set:** Required by Overleaf version 5.0+ for database transaction support

4.2.4 Troubleshooting

During setup, I encountered several critical issues that were resolved:

MongoDB Version Requirement

The initial deployment attempted to use MongoDB 5.0, but Overleaf required MongoDB 6.0. This was resolved by updating the Docker image to `mongo:6.0`.

MongoDB Replica Set Requirement

Overleaf version 5.0+ requires MongoDB to run in replica set mode to support database transactions. The error message "Transaction numbers are only allowed on a replica set member or mongos" indicated this requirement. This was resolved by:

1. Configuring MongoDB with the `--replSet overleaf` command
2. Updating the connection string to include `?replicaSet=overleaf`
3. Manually initializing the replica set using `mongosh`

Environment Variable Rebranding

Overleaf 5.0+ deprecated `SHARELATEX_` prefixed environment variables in favor of `OVERLEAF_` prefixed variables. All environment variables were updated to use the new naming convention.

Volume Path Updates

The path `/var/lib/sharelatex` was deprecated in favor of `/var/lib/overleaf` for the main application data directory.

4.2.5 Initialization

The Overleaf container takes approximately 2–3 minutes to fully initialize on first startup. The `depends_on` configuration with health checks ensures that MongoDB is ready before Overleaf attempts to connect. During this time, the internal services connect to MongoDB and Redis, and the web application becomes available.

4.2.6 Verification

To verify all containers were running correctly, I used:

```
docker-compose ps
```

All three containers (`sharelatex`, `mongo`, and `redis`) showed status "Up" after successful deployment. MongoDB also showed "healthy" status after passing its health check.

4.2.7 Access URL

The Overleaf application is accessible at:

http://142.93.207.133

Note: Overleaf is deployed on a dedicated droplet on port 80, eliminating the need to specify a port number in the URL. This provides a cleaner access experience compared to port-based deployments.

Chapter 5

Overleaf Deployment

5.1 Introduction

This chapter documents the deployment and configuration of Overleaf Community Edition (CE) on DigitalOcean. The deployment includes custom domain integration, SSL certificate setup with Let's Encrypt, GitHub synchronization, and command-line compilation capabilities for LaTeX projects.

5.2 Domain Registration and DNS Configuration

5.2.1 Domain Acquisition

A domain name was acquired through Name.com for the Overleaf deployment:

Domain: `devops-group14.app`

This domain was obtained with a one-year free registration through the GitHub Student Developer Pack.

5.2.2 DigitalOcean Droplet Creation

A DigitalOcean Droplet was created with the following specifications:

Droplet Details:

- **IPv4 Address:** `142.93.207.133`
- **Operating System:** Ubuntu 22.04 LTS
- **Region:** Configured based on proximity requirements

5.2.3 DNS Configuration in DigitalOcean

After creating the Droplet, configure DNS records in the DigitalOcean control panel:

Step 1: Add Domain to DigitalOcean

1. Navigate to Networking → Domains in the DigitalOcean dashboard
2. Enter `devops-group14.app` and click "Add Domain"
3. Select your Droplet from the list

Step 2: Configure DNS Records
The following DNS records are automatically created:

- **A Record:**
  - **Hostname:** @
  - **Will Direct To:** `142.93.207.133`
  - **TTL:** 3600

Step 3: Update Name.com Nameservers
In Name.com DNS settings, update nameservers to point to DigitalOcean:

- `ns1.digitalocean.com`
- `ns2.digitalocean.com`
- `ns3.digitalocean.com`

5.2.4 DNS Propagation Verification

Verify DNS propagation using the following commands from your local machine:

```
# Check A record
dig devops-group14.app

# Verify IP address resolution
ping devops-group14.app

# Check nameservers
dig NS devops-group14.app
```

Expected output should show the IP address `142.93.207.133`. DNS propagation typically takes 15 minutes to 48 hours.

5.3 SSL Certificate Configuration with Let's Encrypt

5.3.1 SSL Certificate Overview

SSL certificates encrypt data transmitted between the client and server, ensuring secure communication. Let's Encrypt provides free SSL certificates that automatically renew every 90 days.

5.3.2 Prerequisites for SSL Setup

Ensure the following requirements are met on your DigitalOcean Droplet (IP: 142.93.207.133):

1. Domain name properly configured and pointing to `142.93.207.133`
2. Ports 80 (HTTP) and 443 (HTTPS) are open in the firewall
3. SSH access to the Droplet

5.3.3 SSL Certificate Installation Process

Install Certbot and Nginx

SSH into your DigitalOcean Droplet and install the required packages:

```
# Update package list
sudo apt update

# Install Nginx
sudo apt install nginx -y

# Install Certbot and Nginx plugin
sudo apt install certbot python3-certbot-nginx -y
```

Configure Nginx as Reverse Proxy

Create an Nginx configuration file for Overleaf:

```
sudo nano /etc/nginx/sites-available/overleaf
```

Add the following configuration:

```
server {
    listen 80;
    server_name devops-group14.app;

    location / {
        proxy_pass http://localhost:80;
        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;

        # WebSocket support
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Timeout settings
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
}
```

Enable the configuration:

```
# Create symbolic link
sudo ln -s /etc/nginx/sites-available/overleaf /etc/nginx/sites-enabled/

# Remove default configuration
sudo rm /etc/nginx/sites-enabled/default
```

```
# Test Nginx configuration
sudo nginx -t

# Restart Nginx
sudo systemctl restart nginx
```

Obtain SSL Certificate

Request an SSL certificate from Let's Encrypt:

```
sudo certbot --nginx -d devops-group14.app
```

During the certificate request:

1. Enter an email address for urgent renewal notices
2. Agree to the Terms of Service
3. Choose whether to redirect HTTP to HTTPS (Select option 2: Redirect)

Certbot automatically modifies the Nginx configuration to include SSL settings.

Verify SSL Configuration

After installation, the Nginx configuration is automatically updated to:

```
server {
    listen 443 ssl;
    server_name devops-group14.app;

    ssl_certificate /etc/letsencrypt/live/devops-group14.app/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/devops-group14.app/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://localhost:80;
        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;

        # WebSocket support
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Timeout settings
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }
}
```

```
server {
    listen 80;
    server_name devops-group14.app;
    return 301 https://$server_name$request_uri;
}
```

5.3.4 SSL Certificate Automatic Renewal

Test Renewal Process

Verify that automatic renewal is configured correctly:

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

Let's Encrypt certificates are valid for 90 days. Certbot automatically creates a systemd timer that checks and renews certificates when they are within 30 days of expiration.

5.3.5 Update Docker Compose for SSL

Since Nginx now handles SSL termination, the docker-compose.yml uses standard HTTP internally. The Overleaf container listens on port 80, and Nginx proxies HTTPS traffic to it.

5.4 Overleaf Instance Deployment

5.4.1 Repository Setup

Fork and Clone Repository

SSH into your DigitalOcean Droplet and clone the forked Overleaf repository:

```
git clone https://github.com/Badri-Narayanan/overleaf-group14.git
cd overleaf-group14
```

Navigate to Configuration Directory

```
cd overleaf
```

5.4.2 Docker Compose Configuration

Create docker-compose.yml

Create the `docker-compose.yml` file with the following configuration:

```
nano docker-compose.yml
```

Add the following content:

```
version: '3.8'
services:
  mongo:
    image: mongo:6.0
    container_name: mongo
    command: ["--replSet", "overleaf"]
    restart: always
    volumes:
      - /root/mongo_data:/data/db
    expose:
      - 27017
    healthcheck:
      test: ["CMD", "mongosh", "--eval", "db.adminCommand('ping')"]
      interval: 10s
      timeout: 5s
      retries: 10
    networks:
      - overleaf-net
  redis:
    image: redis:6.2
    container_name: redis
    restart: always
    volumes:
      - /root/redis_data:/data
    expose:
      - 6379
    networks:
      - overleaf-net
  sharelatex:
    image: sharelatex/sharelatex
    container_name: sharelatex
    restart: always
    depends_on:
      mongo:
        condition: service_healthy
      redis:
        condition: service_started
    ports:
      - "80:80"
    volumes:
      - /root/sharelatex_data:/var/lib/overleaf
    environment:
      OVERLEAF_APP_NAME: Overleaf Community Edition
      OVERLEAF_MONGO_URL: mongodb://mongo:27017/sharelatex?replicaSet=overleaf
      OVERLEAF_REDIS_HOST: redis
      ENABLED_LINKED_FILE_TYPES: project_file,project_output_file
      ENABLE_CONVERSIONS: 'true'
      EMAIL_CONFIRMATION_DISABLED: 'true'
      OVERLEAF_SITE_URL: http://devops-group14.app
      OVERLEAF_NAV_TITLE: Overleaf CE
    networks:
      - overleaf-net
```

```
networks:
  overleaf-net:
```

Key Configuration Points:

- MongoDB runs with replica set support
- Data volumes are stored in `/root/` directory
- Containers communicate through a dedicated network
- Overleaf is accessible on port 80

5.4.3 Build and Deploy Containers

Initialize MongoDB Replica Set

Before starting the containers, ensure MongoDB replica set is properly initialized:

```
# Start containers
docker compose up -d

# Wait for MongoDB to be healthy (check with docker ps)
# Then initialize the replica set
docker exec mongo mongosh --eval "rs.initiate({_id: 'overleaf', members:
↪  [{_id: 0, host: 'mongo:27017'}]})"
```

Build Docker Images

Execute the following command from the repository root:

```
docker compose up -d --build
```

This command:

- Builds Docker images for ShareLaTeX, MongoDB, and Redis
- Creates and starts containers in detached mode
- Establishes networking between containers

Verify Container Status

Check that all containers are running:

```
docker ps
```

Expected output shows three running containers:

- `sharelatex`
- `mongo`
- `redis`

5.4.4 LaTeX Package Installation

Install Essential LaTeX Packages

Install comprehensive LaTeX packages inside the ShareLaTeX container:

```
docker exec -it sharelatex bash -lc "apt-get update && apt-get install -y
↪  --no-install-recommends latexmk texlive-latex-recommended
↪  texlive-latex-extra texlive-fonts-recommended texlive-science
↪  texlive-bibtex-extra texlive-publishers && apt-get clean"
```

This installation includes:

- `latexmk`: Build automation tool
- `texlive-latex-recommended`: Core LaTeX packages
- `texlive-latex-extra`: Extended LaTeX packages
- `texlive-fonts-recommended`: Standard font packages
- `texlive-science`: Scientific and mathematical packages
- `texlive-bibtex-extra`: Bibliography management
- `texlive-publishers`: Journal and publisher templates

5.4.5 Administrator Account Creation

Create Admin Account

Create an administrator account for Overleaf:

```
docker exec sharelatex /bin/bash -c "cd /var/www/sharelatex; grunt
↪  user:create-admin --email=group14@stevens.com"
```

Set Administrator Password

The command outputs a password reset link. Example:

```
Generated password reset token for group14@stevens.com:
http://localhost/user/password/set?passwordResetToken=abc123def456...
```

Access the link by replacing `localhost` with your domain:

```
http://devops-group14.app/user/password/set?passwordResetToken=abc123def456...
```

Set the administrator password as: `somePassword123#`
Administrator Credentials:

- **Email:** `group14@stevens.com`
- **Password:** `somePassword123#`

5.4.6 Access Overleaf Instance

Navigate to https://devops-group14.app in a web browser and log in using the administrator credentials created above.

5.5 GitHub Repository Integration with DigitalOcean App Platform

5.5.1 DigitalOcean App Platform Overview

DigitalOcean App Platform enables automated deployment directly from GitHub repositories. When integrated, any push to the main branch automatically triggers a new deployment, ensuring the application stays synchronized with the latest code changes.

5.5.2 Create DigitalOcean App from GitHub Repository

Connect GitHub Repository

1. Navigate to the DigitalOcean dashboard
2. Click on "Apps" in the left sidebar
3. Click "Create App"
4. Select "GitHub" as the source
5. Authorize DigitalOcean to access your GitHub account
6. Select the repository: `Badri-Narayanan/overleaf-group14`
7. Choose the branch: `main`

Configure Auto-Deploy Settings

Enable automatic deployment on code changes:

1. In the app configuration, locate "Source" settings
2. Check the option **"Autodeploy: code changes"**
3. This ensures that any push to the `main` branch triggers automatic redeployment

Configuration Details:

- **Repository:** https://github.com/Badri-Narayanan/overleaf-group14
- **Branch:** main
- **Autodeploy:** Enabled
- **Deploy on Push:** Yes

Configure App Resources

Set up the application resources:

1. **Resource Type:** Docker Compose
2. **Source Directory:** `/overleaf` (where docker-compose.yml is located)
3. **Environment Variables:** Already defined in docker-compose.yml
4. **HTTP Port:** 80

Deploy the App

1. Review the app configuration
2. Click "Create Resources"
3. Wait for the initial deployment to complete
4. DigitalOcean will build and deploy the containers

5.5.3 Verify Automatic Deployment

Test Auto-Deploy Functionality

To verify automatic deployment works:

```
# Clone the repository locally
git clone https://github.com/Badri-Narayanan/overleaf-group14.git
cd overleaf-group14

# Make a test change
echo "# Test deployment" >> README.md

# Commit and push to main branch
git add README.md
git commit -m "Test automatic deployment"
git push origin main
```

Monitor Deployment

1. Navigate to the DigitalOcean App Platform dashboard
2. Select your app
3. View the "Activity" tab to see the automatic deployment triggered
4. Wait for deployment to complete (typically 5-10 minutes)

5.6 Command-Line Compilation

5.6.1 Compilation Overview

Command-line compilation enables automated builds and integration with CI/CD pipelines. All compilation commands are executed via SSH on the DigitalOcean Droplet (142.93.207.133).

5.6.2 Access DigitalOcean Droplet

Connect to the Droplet via SSH:

```
ssh root@142.93.207.133
```

5.6.3 Direct Container Compilation

Access the ShareLaTeX Container

```
docker exec -it sharelatex bash
```

Navigate to Project Directory

Projects are stored in the compiles directory:

```
cd /var/lib/overleaf/data/compiles
ls -la
```

Navigate to your specific project:

```
cd /var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc
ls -la
```

Locate Main LaTeX File

Identify the main `.tex` file (commonly `main.tex`):

```
ls -la *.tex
```

Compile Using latexmk

Use latexmk for automated compilation with dependency resolution:

```
latexmk -pdf -interaction=nonstopmode main.tex
```

Command Options:

- `-pdf`: Generate PDF output
- `-interaction=nonstopmode`: Continue compilation without stopping for errors

Alternative Compilation Methods

Using pdfLaTeX:

```
pdflatex -interaction=nonstopmode main.tex
```

5.6.4 Extract Compiled PDF

Copy PDF from Container to Droplet

Exit the container and execute from the Droplet:

```
docker cp
↪ sharelatex:/var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc/main.pdf
↪ ./output.pdf
```

Download PDF to Local Machine

From your local machine, download the PDF using SCP:

```
scp root@142.93.207.133:~/output.pdf ./local-output.pdf
```

5.6.5 Compilation from GitHub Repository

Clone Repository on Droplet

SSH into the Droplet and clone the GitHub repository:

```
cd ~
git clone https://github.com/Badri-Narayanan/overleaf-group14.git
cd overleaf-group14
```

Copy Files to Overleaf Container

```
docker cp ./
↪ sharelatex:/var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc/
```

Compile Inside Container

```
docker exec sharelatex bash -c "cd
↪ /var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc && latexmk -pdf
↪ -interaction=nonstopmode main.tex"
```

Extract Compiled Output

```
docker cp
↪ sharelatex:/var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc/main.pdf
↪ ./compiled-output.pdf
```

5.6.6 Troubleshooting Compilation Issues

View Compilation Logs

If compilation fails, check the log file:

```
docker exec sharelatex bash -c "cd
↪ /var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc && cat main.log"
```

Check LaTeX Package Installation

Verify required packages are installed:

```
docker exec sharelatex bash -c "latex --version"
docker exec sharelatex bash -c "pdflatex --version"
```

Install Missing Packages

If specific packages are missing:

```
docker exec -it sharelatex bash
apt-get update
apt-get install -y texlive-PACKAGE-NAME
```

5.7 Version Control Integration with Document Title

5.7.1 Purpose and Implementation

To establish traceability between compiled documents and their source code versions in GitHub, the document title should include the Git commit hash. This creates an audit trail that maps each generated PDF to its exact source code state.

5.7.2 Automated Version Insertion Strategy

Modify LaTeX Document for Dynamic Versioning

Update the main LaTeX file to include a version placeholder:

```
\documentclass{article}
\usepackage{fancyhdr}

% Define version command
\newcommand{\documentversion}{VERSION_PLACEHOLDER}

\title{Project Report - Version \documentversion}
\author{Group 14}
\date{\today}

\begin{document}
\maketitle

% Document content here

\end{document}
```

Create Version Injection Script

Create a script on the DigitalOcean Droplet to automatically inject the Git commit hash:

```
#!/bin/bash
# inject-version.sh

REPO_PATH="$1"
TEX_FILE="$2"

if [ -z "$REPO_PATH" ] || [ -z "$TEX_FILE" ]; then
    echo "Usage: ./inject-version.sh REPO_PATH TEX_FILE"
    exit 1
fi

cd "$REPO_PATH"

# Get the latest commit hash
COMMIT_HASH=$(git rev-parse --short HEAD)
COMMIT_DATE=$(git log -1 --format=%cd --date=short)

echo "Injecting version: $COMMIT_HASH (Date: $COMMIT_DATE)"

# Replace version placeholder with actual commit hash
sed -i "s/VERSION_PLACEHOLDER/$COMMIT_HASH/g" "$TEX_FILE"

echo "Version injection completed."
```

Make the script executable:

```
chmod +x inject-version.sh
```

Usage Example

SSH into the DigitalOcean Droplet (142.93.207.133) and execute:

```
# Clone the repository
git clone https://github.com/Badri-Narayanan/overleaf-group14.git
cd overleaf-group14

# Inject version into LaTeX document
./inject-version.sh . main.tex

# Copy to Overleaf container
docker cp ./
↪  sharelatex:/var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc/

# Compile the document
docker exec sharelatex bash -c "cd
↪  /var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc && latexmk -pdf
↪  -interaction=nonstopmode main.tex"

# Extract the compiled PDF
docker cp
↪  sharelatex:/var/lib/overleaf/data/compiles/68f14824c7c1f423d5bbe7bc/main.pdf
↪  ./output-versioned.pdf
```

5.7.3 Manual Version Addition

If the LaTeX document doesn't contain a VERSION_PLACEHOLDER, manually add the version to the title:

```
# Get the current commit hash
COMMIT_HASH=$(git rev-parse --short HEAD)

# Modify the title in the LaTeX file
sed -i "s/\\title{\([^}]*\)}/\\title{\1 - Version $COMMIT_HASH}/g" main.tex
```

5.7.4 Verification

After compilation, verify that the PDF contains the version information:

1. Open the generated PDF
2. Check the title page for the commit hash
3. Verify the commit hash matches the latest Git commit:

```
git rev-parse --short HEAD
```

Chapter 6

Overleaf Compilation and Sync with GitHub

6.1 System Overview

- **Overleaf Instance:** DigitalOcean Droplet (IP: 142.93.207.133)
- **Project Path:** `/root/sharelatex_data/data/compiles/68f9818f6cbab3ad67b265f7-68f147`
- **Backup Directory:** `/root/overleaf_backups`
- **GitHub Repository:** Connected via SSH
- **Automation:** Cron job (every 5 seconds) + GitHub Actions

6.2 Step 1: SSH into DigitalOcean Droplet

```
ssh root@142.93.207.133
```

6.3 Step 2: Create Backup Directory and Initialize Git

```
# Create backup directory
mkdir -p /root/overleaf_backups
cd /root/overleaf_backups

# Initialize git repository
git init
git branch -M main

# Configure git credentials
git config user.name "Badri-Narayanan"
git config user.email "badhrirajen@gmail.com"

# Generate SSH key for GitHub
ssh-keygen -t ed25519 -C "badhrirajen@gmail.com" -f /root/.ssh/id_ed25519 -N ""
```

```
# Display public key (add this to GitHub)
cat /root/.ssh/id_ed25519.pub
```

Add SSH key to GitHub:

1. Copy the public key output
2. Go to GitHub → Settings → SSH and GPG keys → New SSH key
3. Paste and save

```
# Add GitHub remote (replace with your repo URL)
git remote add origin git@github.com:YOUR_USERNAME/YOUR_REPO.git

# Test connection
ssh -T git@github.com
```

6.4 Step 3: Create Backup Script

```
nano /root/backup-overleaf-projects.sh
```

Script Content:

```
#!/bin/bash

# Configuration
SOURCE_DIR="/root/sharelatex_data/data/compiles/68f9818f6cbab3ad67b265f7-68f14753eda1
BACKUP_DIR="/root/overleaf_backups"
LOG_FILE="/var/log/overleaf-backup.log"
GIT_BRANCH="main"
MAX_LOG_SIZE=10485760  # 10MB

# Function to rotate log if too large
rotate_log() {
    if [ -f "$LOG_FILE" ]; then
        LOG_SIZE=$(stat -f%z "$LOG_FILE" 2>/dev/null || stat -c%s "$LOG_FILE" 2>/dev/
        if [ "$LOG_SIZE" -gt "$MAX_LOG_SIZE" ]; then
            mv "$LOG_FILE" "$LOG_FILE.old"
            echo "Log rotated at $(date '+%Y-%m-%d %H:%M:%S')" > "$LOG_FILE"
        fi
    fi
}

# Function to log messages with timestamp
log_message() {
    local TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')
    echo "[$TIMESTAMP] $1" >> "$LOG_FILE"
}
```

```
# Function to log errors
log_error() {
    local TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')
    echo "[$TIMESTAMP] ERROR: $1" >> "$LOG_FILE"
}

# Function to log success
log_success() {
    local TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')
    echo "[$TIMESTAMP] SUCCESS: $1" >> "$LOG_FILE"
}

# Rotate log if needed
rotate_log

# Start backup process
log_message "========== Starting backup process =========="

# Check if source directory exists
if [ ! -d "$SOURCE_DIR" ]; then
    log_error "Source directory does not exist: $SOURCE_DIR"
    exit 1
fi

log_message "Source directory exists: $SOURCE_DIR"

# Check if backup directory exists
if [ ! -d "$BACKUP_DIR" ]; then
    log_error "Backup directory does not exist: $BACKUP_DIR"
    exit 1
fi

log_message "Backup directory exists: $BACKUP_DIR"

# Check if backup directory is a git repository
if [ ! -d "$BACKUP_DIR/.git" ]; then
    log_error "Backup directory is not a Git repository: $BACKUP_DIR"
    exit 1
fi

log_message "Git repository detected in backup directory"

# Change to backup directory
cd "$BACKUP_DIR" || {
    log_error "Cannot change to backup directory: $BACKUP_DIR"
    exit 1
}

log_message "Changed to backup directory"
```

```
# Check if we're on the correct branch
CURRENT_BRANCH=$(git rev-parse --abbrev-ref HEAD 2>/dev/null)
if [ "$CURRENT_BRANCH" != "$GIT_BRANCH" ]; then
    log_message "Current branch is $CURRENT_BRANCH, switching to $GIT_BRANCH"
    git checkout "$GIT_BRANCH" >> "$LOG_FILE" 2>&1
    if [ $? -ne 0 ]; then
        log_error "Failed to checkout branch $GIT_BRANCH"
        exit 1
    fi
fi

# Stash any local changes to protected files before pulling
log_message "Checking for local changes to protected files"
PROTECTED_FILES=".github .gitignore README.md builds"
HAS_PROTECTED_CHANGES=false

for file in $PROTECTED_FILES; do
    if [ -e "$file" ] && git diff --quiet "$file" 2>/dev/null; then
        :
    elif [ -e "$file" ]; then
        HAS_PROTECTED_CHANGES=true
        break
    fi
done

if [ "$HAS_PROTECTED_CHANGES" = true ]; then
    log_message "Stashing changes to protected files"
    git stash push -m "Auto-stash before pull $(date '+%Y-%m-%d %H:%M:%S')" $PROTECTE
fi

# Pull latest changes from remote
log_message "Pulling latest changes from remote"
git pull origin "$GIT_BRANCH" >> "$LOG_FILE" 2>&1
PULL_EXIT_CODE=$?

if [ $PULL_EXIT_CODE -eq 0 ]; then
    log_success "Successfully pulled from remote"
else
    log_error "Git pull failed with exit code $PULL_EXIT_CODE"
    log_message "Attempting to continue anyway..."
fi

# Pop stashed changes if any
if [ "$HAS_PROTECTED_CHANGES" = true ]; then
    log_message "Restoring stashed changes"
    git stash pop >> "$LOG_FILE" 2>&1
fi
```

```
# Ensure .github directory exists
if [ ! -d "$BACKUP_DIR/.github" ]; then
    log_message ".github directory not found, creating it"
    mkdir -p "$BACKUP_DIR/.github/workflows"
fi

# Sync the compiles directory to backup location
log_message "Starting rsync from $SOURCE_DIR to $BACKUP_DIR"

rsync -av --delete \
    --exclude='.git' \
    --exclude='.git/' \
    --exclude='.github' \
    --exclude='.github/' \
    --exclude='.gitignore' \
    --exclude='README.md' \
    --exclude='LICENSE' \
    --exclude='builds' \
    --exclude='builds/' \
    "$SOURCE_DIR/" "$BACKUP_DIR/" >> "$LOG_FILE" 2>&1

RSYNC_EXIT_CODE=$?

if [ $RSYNC_EXIT_CODE -eq 0 ]; then
    log_success "Rsync completed successfully"
else
    log_error "Rsync failed with exit code $RSYNC_EXIT_CODE"
    exit 1
fi

# Check Git status before staging
log_message "Checking for changes in Git repository"
git status --porcelain >> "$LOG_FILE" 2>&1

# Stage all changes
log_message "Staging all changes"
git add -A >> "$LOG_FILE" 2>&1

if [ $? -ne 0 ]; then
    log_error "Failed to stage changes"
    exit 1
fi

# Check if there are any changes to commit
if git diff --cached --quiet; then
    log_message "No changes detected - nothing to commit"
    log_message "========== Backup process completed (no changes) =========="
    exit 0
fi
```

```
# Count the changes
ADDED=$(git diff --cached --numstat | wc -l)
log_message "Changes detected: $ADDED file(s) modified/added/deleted"

# Show what changed for debugging
log_message "Files changed:"
git diff --cached --name-status >> "$LOG_FILE" 2>&1

# Create commit message with details
COMMIT_TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')
COMMIT_MESSAGE="Auto-backup: $COMMIT_TIMESTAMP

Changes: $ADDED file(s) modified/added/deleted
Source: $SOURCE_DIR"

# Commit the changes
log_message "Committing changes"
git commit -m "$COMMIT_MESSAGE" >> "$LOG_FILE" 2>&1
COMMIT_EXIT_CODE=$?

if [ $COMMIT_EXIT_CODE -ne 0 ]; then
    log_error "Git commit failed with exit code $COMMIT_EXIT_CODE"
    exit 1
fi

log_success "Changes committed successfully"

# Push to remote repository
log_message "Pushing to remote repository (branch: $GIT_BRANCH)"
git push origin "$GIT_BRANCH" >> "$LOG_FILE" 2>&1
PUSH_EXIT_CODE=$?

if [ $PUSH_EXIT_CODE -eq 0 ]; then
    log_success "Changes pushed to remote repository successfully"
    COMMIT_HASH=$(git rev-parse --short HEAD)
    log_message "Commit hash: $COMMIT_HASH"
else
    log_error "Git push failed with exit code $PUSH_EXIT_CODE"
    log_message "Checking if local is behind remote..."
    git fetch origin "$GIT_BRANCH" >> "$LOG_FILE" 2>&1
    LOCAL=$(git rev-parse @)
    REMOTE=$(git rev-parse @{u})
    if [ "$LOCAL" != "$REMOTE" ]; then
        log_message "Local is out of sync with remote, will retry on next run"
    fi
    exit 1
fi
```

```
log_message "========== Backup process completed successfully =========="
exit 0

# Make script executable
chmod +x /root/backup-overleaf-projects.sh
```

6.5 Step 4: Setup Cron Job (Every 5 Seconds)

```
crontab -e
```

Add these lines:

```
* * * * * /root/backup-overleaf-projects.sh
* * * * * sleep 5; /root/backup-overleaf-projects.sh
* * * * * sleep 10; /root/backup-overleaf-projects.sh
* * * * * sleep 15; /root/backup-overleaf-projects.sh
* * * * * sleep 20; /root/backup-overleaf-projects.sh
* * * * * sleep 25; /root/backup-overleaf-projects.sh
* * * * * sleep 30; /root/backup-overleaf-projects.sh
* * * * * sleep 35; /root/backup-overleaf-projects.sh
* * * * * sleep 40; /root/backup-overleaf-projects.sh
* * * * * sleep 45; /root/backup-overleaf-projects.sh
* * * * * sleep 50; /root/backup-overleaf-projects.sh
* * * * * sleep 55; /root/backup-overleaf-projects.sh
```

Verify cron is running:

```
sudo systemctl status cron
crontab -l
```

6.6 Step 5: Create GitHub Actions Workflow

```
# Create workflow directory
mkdir -p /root/overleaf_backups/.github/workflows

# Create workflow file
nano /root/overleaf_backups/.github/workflows/build.yml
```

Workflow Content (YAML):

```
name: Build LaTeX PDF
permissions:
  contents: write
on:
  push:
    branches:
      - main
  workflow_dispatch:
jobs:
```

```
build:
  runs-on: ubuntu-latest
  steps:
    - name: Checkout repository
      uses: actions/checkout@v4
      with:
        persist-credentials: true
        fetch-depth: 0

    - name: Install TeX Live and Pygments
      run: |
        sudo apt-get update
        sudo apt-get install -y texlive-full python3-pygments

    - name: Get Git commit hash
      run: echo "GIT_HASH=$(git rev-parse --short HEAD)" >> $GITHUB_ENV

    - name: Update version in groupAssignments.tex
      run: |
        sed -i "s/\\\\newcommand{\\\\gitversion}{Unknown}/\\\\newcommand{\\\\gitver
        echo "Updated version to: ${GIT_HASH}"

    - name: Compile LaTeX document
      run: |
        pdflatex -shell-escape -interaction=nonstopmode groupAssignments.tex || tru
        bibtex groupAssignments || true
        makeindex groupAssignments.idx || true
        pdflatex -shell-escape -interaction=nonstopmode groupAssignments.tex || tru
        pdflatex -shell-escape -interaction=nonstopmode groupAssignments.tex || tru

    - name: Verify PDF was created
      run: |
        if [ -f "groupAssignments.pdf" ]; then
          echo "PDF successfully generated"
          ls -lh groupAssignments.pdf
        else
          echo "ERROR: PDF was not generated!"
          exit 1
        fi

    - name: Upload PDF artifact
      uses: actions/upload-artifact@v4
      with:
        name: Compiled-PDF
        path: groupAssignments.pdf

    - name: Commit PDF to repo
      if: success()
      env:
```

```
      GIT_HASH: ${{ env.GIT_HASH }}
    run: |
      git config user.name "Badri-Narayanan"
      git config user.email "badhrirajen@gmail.com"
      mkdir -p builds
      cp groupAssignments.pdf builds/document_${GIT_HASH}.pdf
      git add builds/document_${GIT_HASH}.pdf
      git commit -m "Auto-build PDF for commit ${GIT_HASH}" || echo "No changes"
      git push origin HEAD:main || echo "Push failed"
```

6.7 Step 6: Initial Commit and Push

```
cd /root/overleaf_backups

# Add .gitignore
cat > .gitignore << 'EOF'
*.log
*.aux
*.out
*.toc
*.synctex.gz
_minted-*
*.idx
*.ilg
*.ind
*.bbl
*.blg
*.fls
*.fdb_latexmk
EOF

# Add all files
git add -A

# Initial commit
git commit -m "Initial commit: Overleaf project setup"

# Push to remote
git push -u origin main
```

6.8 Step 7: Test the Setup

```
# Test backup script manually
/root/backup-overleaf-projects.sh

# Check log
tail -50 /var/log/overleaf-backup.log
```

```
# Verify files in backup directory
ls -la /root/overleaf_backups/
```

6.9 Step 8: Monitoring Commands

```
# Watch backup log in real-time
tail -f /var/log/overleaf-backup.log

# Monitor cron execution
grep CRON /var/log/syslog | tail -20

# Check git status
cd /root/overleaf_backups && git status

# View recent commits
cd /root/overleaf_backups && git log --oneline -10

# Check if .github folder is protected
ls -la /root/overleaf_backups/.github/workflows/
```

6.10 Workflow Process

1. Edit Overleaf document on DigitalOcean (http://devops-group14.app)
2. Cron job runs every 5 seconds:
    - Pulls latest from GitHub
    - Syncs Overleaf files (excluding .github)
    - Commits changes
    - Pushes to GitHub
3. GitHub Actions triggers:
    - Checks out repository
    - Installs TeX Live
    - Gets commit hash
    - Updates version in document
    - Compiles LaTeX (3 passes)
    - Creates PDF with version number
    - Saves to builds/ directory
    - Commits and pushes PDF back

6.11 Verification Checklist

Cron job running every 5 seconds

Files syncing from Overleaf to GitHub

.github folder protected from deletion

GitHub Actions workflow executing on push

PDF being compiled with commit hash

PDF saved to builds/ directory

All changes committed and pushed

6.12 Troubleshooting Commands

```
# Check cron service
sudo systemctl status cron

# View full backup log
cat /var/log/overleaf-backup.log

# Test rsync without delete
rsync -avn --exclude='.git' --exclude='.github' \
  /root/sharelatex_data/data/compiles/68f9818f6cbab3ad67b265f7-68f14753eda193f7e7fa03
  /root/overleaf_backups/

# Check SSH connection to GitHub
ssh -T git@github.com

# Force manual sync
cd /root/overleaf_backups && git pull && git push
```

6.13 Project Structure

```
/root/overleaf_backups/
|-- .git/
|-- .github/
|   `-- workflows/
|       `-- build.yml
|-- .gitignore
|-- groupAssignments.tex
|-- [other .tex files]
|-- png/
|   `-- [images]
`-- builds/
    `-- document_[hash].pdf
```

Chapter 7

Prometheus And Grafana

7.1 Technologies Used

- **Prometheus** is a monitoring system that collects and stores time-series metrics from various sources by scraping HTTP endpoints at regular intervals.
- **Node Exporter** is an agent that runs on your servers and exposes system-level metrics in a format Prometheus can scrape.
- **Grafana** is a visualization platform that queries Prometheus data and displays it through customizable dashboards with graphs and alerts.
- **Dashboard 1860** It shows real-time and historical data about CPU usage, memory consumption, disk I/O and space, network traffic, and system load.

7.2 Executed Outputs

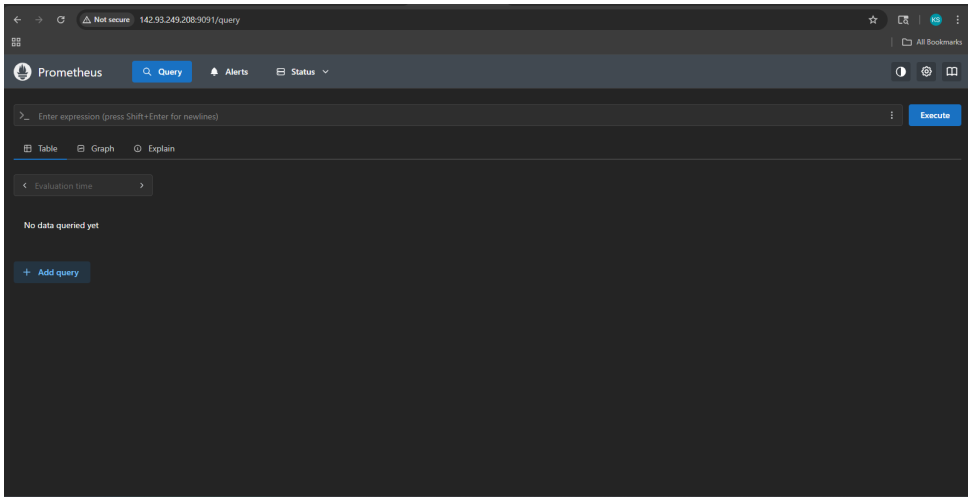

Figure 7.1: Prometheus Dashboard Image - 1

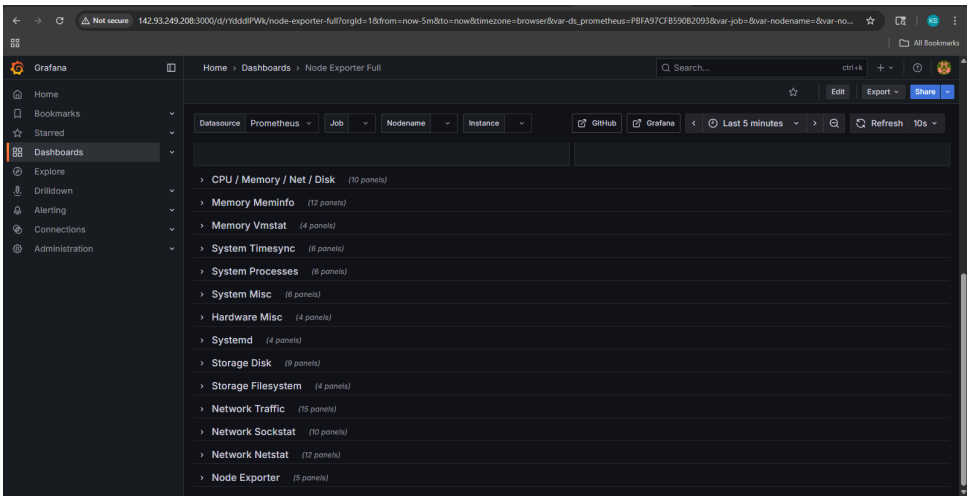

Figure 7.2: Prometheus Dashboard Image - 2

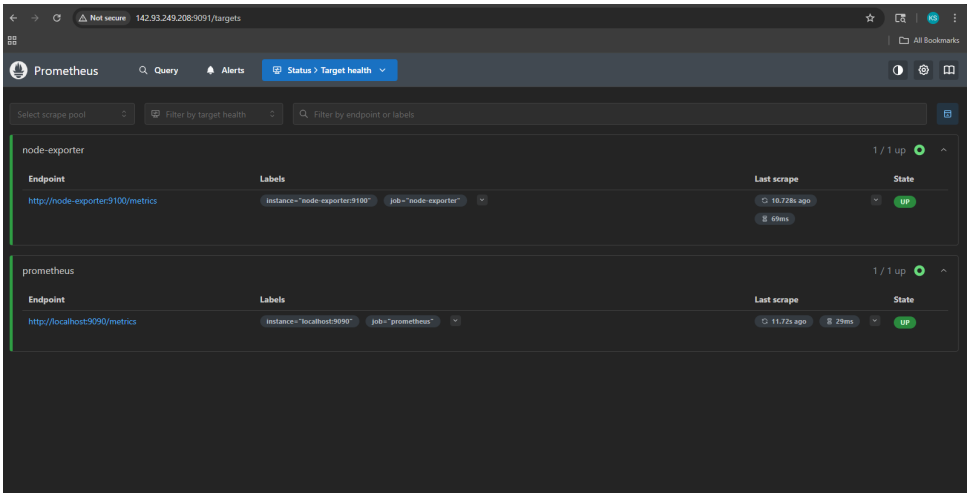

Figure 7.3: Prometheus Status Page

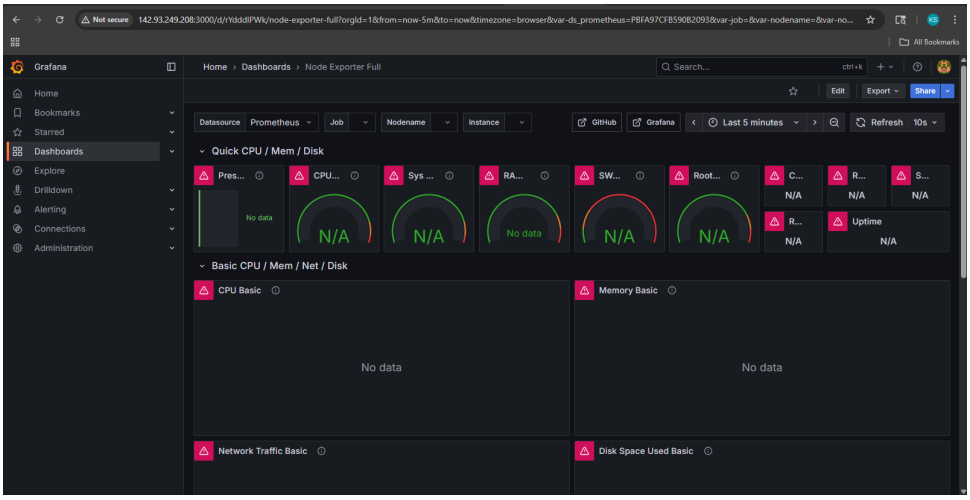

Figure 7.4: Grafana 1860 Dashboard

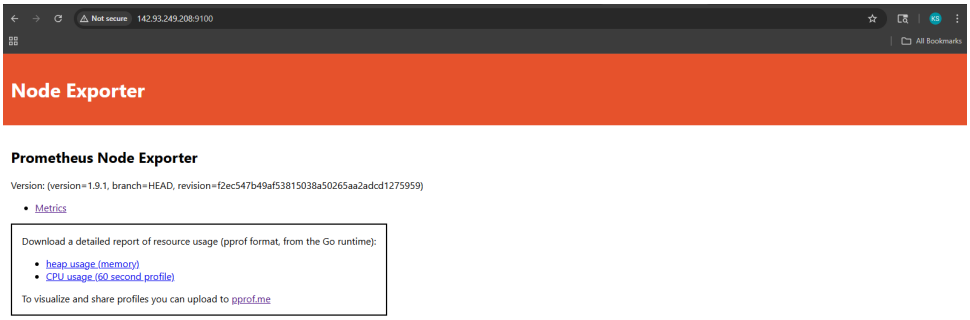

Figure 7.5: Node Exporter Home

Chapter 8

Jenkins with Pytest

8.1 Introduction

This chapter walks through setting up a CI/CD pipeline with Jenkins, Docker, Python, and Pytest. The goal was to create an automated testing system that works with Git and generates test reports automatically.

8.2 Environment Setup

8.2.1 Project Directory Structure

First, I created the project directory and organized the files like this:

```
jenkins-python-pytest-demo/
 docker-compose.yml
 Jenkinsfile
 requirements.txt
 .gitignore
 tests/
     test_sample.py
```

8.2.2 Docker Compose Configuration

I set up a Docker Compose file to run Jenkins in a container. The configuration includes volume mounts so Jenkins data persists between restarts, and also mounts the Docker socket so Jenkins can run Docker commands if needed.

[language=yaml, caption=docker-compose.yml] version: '3.8' services: jenkins: image: jenkins/jenkins:lts container$_{n}ame$: $jenkinsports$: $-$"8080 : 8080" $-$ "50000 : 50000"$volumes$: $-jenkins_{h}ome$: $/var/jenkins_{h}ome-/var/run/docker.sock$: $/var/run/docker.sock$u $root$

volumes: jenkins$_{h}ome$:

Configuration Note: I added `user: root` to fix permission issues when accessing the Docker socket. Without this, Jenkins couldn't access Docker properly.

8.2.3 Starting Jenkins

Starting Jenkins was straightforward with Docker Compose:

```
docker compose up -d
```

I waited about 1-2 minutes for the container to fully start up before accessing it.

8.3 Jenkins Initial Configuration

8.3.1 Accessing Jenkins Web Interface

Once the container was running, I opened a browser and went to `http://localhost:8080`. To unlock Jenkins, I needed the initial admin password which I got by running:

```
docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

8.3.2 Plugin Installation

During the setup wizard, I just clicked "Install suggested plugins" which gave me all the basic plugins I needed for Git integration, pipelines, and JUnit test reporting.

8.3.3 Admin User Creation

I created an admin account with my credentials so I wouldn't have to use that long generated password every time. View password hints for details.

8.3.4 Python Installation in Jenkins Container

Here's where I ran into my first issue - the Jenkins Docker image doesn't come with Python installed by default. So I had to install it manually inside the container:

```
docker exec -it jenkins bash
apt-get update
apt-get install -y python3 python3-venv python3-pip
exit
```

This was a critical step - without Python installed, the pipeline wouldn't be able to run any tests.

8.4 Python Project Implementation

8.4.1 Dependencies Configuration

I created a simple `requirements.txt` file with just pytest:
[caption=requirements.txt] pytest

8.4.2 Test Suite Implementation

For the tests, I wrote three simple test functions in `tests/test_sample.py`:

[language=Python, caption=tests/test_sample.py] def test$_addition() : assert 1+1 == 2$

def test$_subtraction() : assert 5-2 == 3$

def test$_failure_example() : assert 2*2 == 5 This will fail$

The third test was intentionally made to fail (2 * 2 doesn't equal 5!) so I could show test failure reporting in Jenkins.

8.5 CI/CD Pipeline Configuration

8.5.1 Jenkinsfile Implementation

The Jenkinsfile is where the magic happens - it tells Jenkins what to do automatically:

[language=Java, caption=Jenkinsfile] pipeline agent any

stages stage('Install dependencies') steps sh 'python3 -m venv venv' sh './venv/bin/pip install -r requirements.txt'

stage('Run tests') steps sh './venv/bin/pytest –junitxml=report.xml'

stage('Publish Report') steps junit 'report.xml'

The pipeline has three main stages:

1. **Install dependencies:** Creates a Python virtual environment and installs pytest
2. **Run tests:** Runs pytest and generates a JUnit XML report
3. **Publish Report:** Makes the test results visible in Jenkins UI

8.6 Git Integration

8.6.1 Local Repository Initialization

I initialized a Git repo to track my code:

```
git init
git add .
git commit -m "Initial commit"
```

8.6.2 Git Ignore Configuration

Created a `.gitignore` file to keep generated files out of version control:

[caption=.gitignore] venv/ *.pyc $_{pycache/report.xml.pytest_cache/}$

8.6.3 GitHub Repository Setup

I pushed everything to GitHub so Jenkins could pull the code from there:

```
git remote add origin https://github.com/[username]/jenkins-python-pytest-demo.git
git branch -M main
git push -u origin main
```

8.7 Jenkins Pipeline Job Configuration

8.7.1 Job Creation

Creating the pipeline job in Jenkins was pretty straightforward:

1. Went to Jenkins Dashboard
2. Clicked "New Item"
3. Named it: `Python-Pytest-Pipeline`
4. Selected "Pipeline" as the project type
5. Clicked "OK"

8.7.2 SCM Configuration

I configured Jenkins to pull the Jenkinsfile from my GitHub repo:

- Definition: Pipeline script from SCM
- SCM: Git
- Repository URL: My GitHub repository URL
- Branch Specifier: `*/main`
- Script Path: `Jenkinsfile`

8.8 Pipeline Execution and Results

8.8.1 Build Execution

I clicked "Build Now" and watched Jenkins do its thing. The pipeline ran through all three stages automatically.

8.8.2 Console Output Analysis

Looking at the console output, I could see each step happening:

- Creating the virtual environment
- Installing pytest
- Running the tests
- Generating the JUnit report

8.8.3 Test Results

As expected, the tests ran with these results:

- Total tests: 3
- Passed: 2 (test_addition, test_subtraction)
- Failed: 1 (test_failure_example)

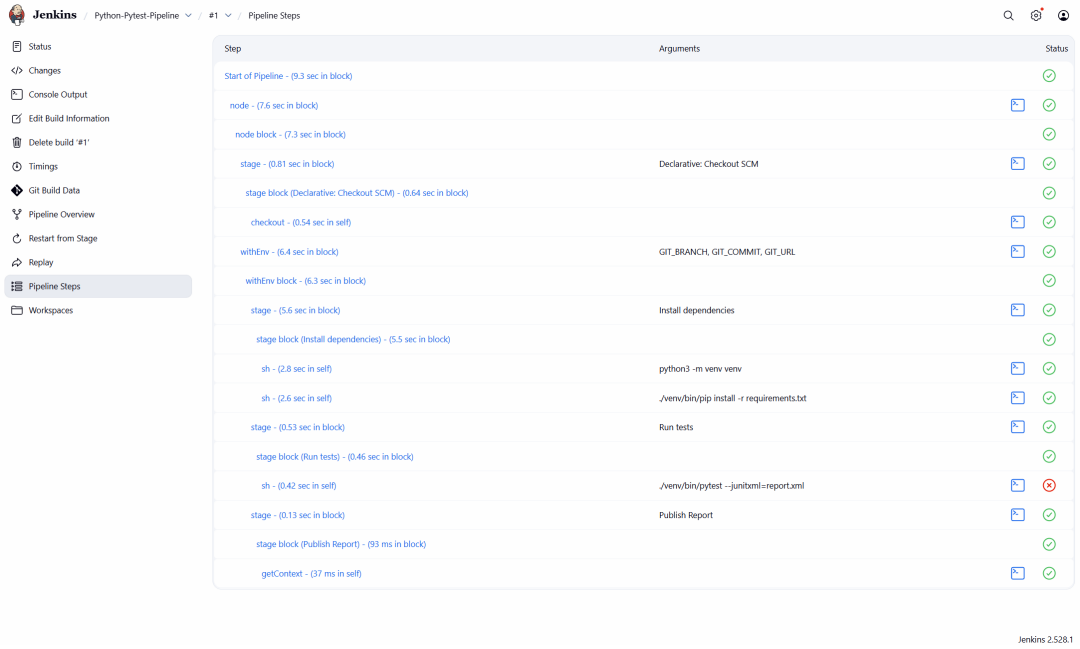

Figure 8.1: Jenkins Pipeline Successful Execution showing all stages completed

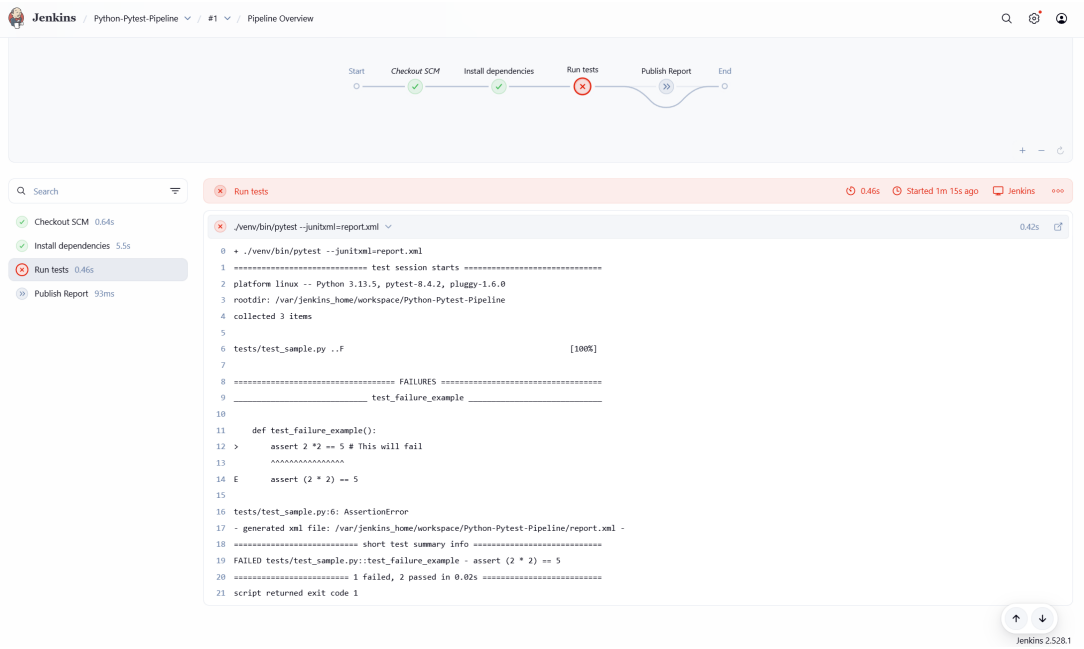

Figure 8.2: Jenkins Test Results showing 2 passed and 1 failed test

8.9 Additional Configuration Changes

8.9.1 Docker Socket Permissions

Early on, I hit a permission error with `/var/run/docker.sock`. Fixed it by adding `user: root` to the Docker Compose file.

8.9.2 Workspace Cleanup

I also made sure the workspace directory had the right permissions:

```
docker exec -it jenkins chmod -R 755 /var/jenkins_home/workspace
```

8.10 Bonus Features Implementation

8.10.1 Code Coverage Tracking

I added code coverage to get more detailed metrics about the tests. First, updated `requirements.txt`:

[caption=Updated requirements.txt] pytest pytest-cov

Then modified the Jenkinsfile to generate coverage reports:

[language=Java, caption=Updated Run tests stage] stage('Run tests') steps sh './venv/bin/pytest –junitxml=report.xml –cov=. –cov-report=html'

8.10.2 Email Notifications on Failure

I set up email notifications so I'd know when builds fail. Added this to the Jenkinsfile:

[language=Java, caption=Post-build actions in Jenkinsfile] post failure mail to: 'admin@example.com', subject: "Failed Pipeline: $currentBuild.fullDisplayName", body : "$Buildfailed.Check$env.BUILD_URL"

Email Server Configuration: To make the emails actually work, I had to configure SMTP settings in Jenkins (Manage Jenkins → Configure System → E-mail Notification):

- SMTP Server: smtp.gmail.com
- SMTP Port: 465
- Use SSL: Enabled
- SMTP Authentication: Added my Gmail app password

Note: Regular Gmail passwords don't work - you need to generate an "App Password" from your Google account settings.

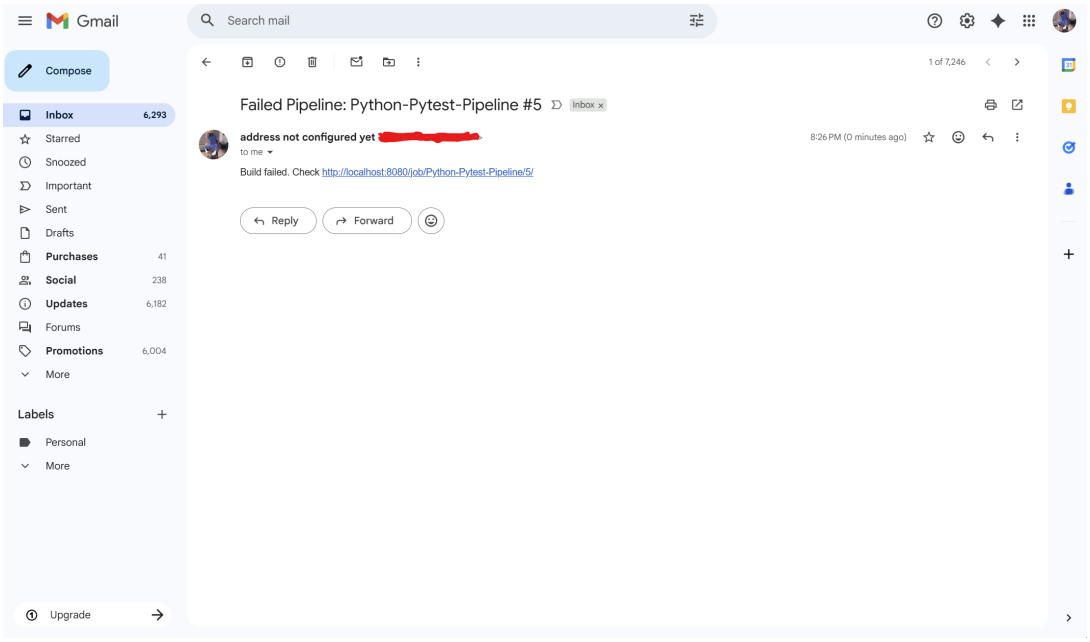

Figure 8.3: Email notification received after test failure

8.11 Deliverables

8.11.1 GitHub Repository

All the project code is available on GitHub at:

`https://github.com/[username]/jenkins-python-pytest-demo`

8.11.2 Screenshots Provided

Here are the screenshots showing everything working:

1. Jenkins pipeline execution with all stages completed (Figure 8.1)
2. Test results showing 2 passed and 1 failed test (Figure 8.2)
3. Email notification when a test fails (Figure 8.3)

8.12 Challenges and Solutions

8.12.1 Challenge: Python Not Available

Problem: Jenkins Docker image doesn't come with Python installed.
Solution: Had to manually install Python 3, pip, and venv inside the container using apt-get.

8.12.2 Challenge: Docker Socket Permissions

Problem: Got permission denied errors when Jenkins tried to access Docker.
Solution: Added `user: root` to the docker-compose.yml file.

8.12.3 Challenge: Virtual Environment Path Issues

Problem: Jenkins couldn't find the pytest executable because the venv wasn't in PATH.

Solution: Used full paths like `./venv/bin/pytest` instead of just `pytest`.

8.12.4 Challenge: Email SMTP Configuration

Problem: Initial email notifications failed with "Connection refused" errors.

Solution: Had to properly configure SMTP settings in Jenkins with Gmail's SMTP server and generate an App Password instead of using my regular password.

8.13 Conclusion

This project was a great hands-on experience setting up a complete CI/CD pipeline. I got Jenkins running in Docker, created a Python testing environment, and made it all work together with Git integration. The pipeline automatically installs dependencies, runs tests, and generates reports - exactly what you'd want in a real development workflow.

The bonus features (email notifications and code coverage) added even more value and made the pipeline feel more production-ready. While I ran into some issues along the way (Python installation, permissions, SMTP config), working through them helped me understand how all these pieces fit together.

Key things I learned:

- How to containerize Jenkins for consistent environments
- Writing declarative Jenkins pipelines
- Integrating automated testing with Python and Pytest
- Setting up Git/GitHub integration
- Configuring notifications and monitoring

This setup could easily be expanded with deployment stages, multiple environments, or integration with cloud services. It's a solid foundation for any CI/CD workflow.

Chapter 9

Load Balancer and Virtual Host Setup

A — Project: Load Balancer

File Structure

```
load-balanced-app/
 docker-compose.yml
 nginx/
    nginx.conf
 web1/
    index.html
 web2/
     index.html
```

File Contents

web1/index.html

```
<h1>Hello from Web 1</h1>
```

web2/index.html

```
<h1>Hello from Web 2</h1>
```

nginx/nginx.conf

```
events {}

http {
    upstream backend {
        server web1:80;
        server web2:80;
    }

    server {
        listen 80;

        location / {
```

```
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }
}
```

docker-compose.yml

```
version: '3'

services:
  web1:
    image: nginx
    container_name: web1
    volumes:
      - ./web1:/usr/share/nginx/html:ro

  web2:
    image: nginx
    container_name: web2
    volumes:
      - ./web2:/usr/share/nginx/html:ro

  loadbalancer:
    image: nginx
    container_name: loadbalancer
    ports:
      - "8080:80"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - web1
      - web2
```

Steps Executed

```
cd /path/to/load-balanced-app
docker-compose up -d --build
docker ps
# verify containers: web1, web2, loadbalancer

# local browser test:
# open http://localhost:8080 and refresh to see alternating servers

docker-compose down
```

B — Project: Virtual Host (Name-Based)

File Structure

```
virtual-hosts/
```

```
docker-compose.yml
nginx/
   nginx.conf
web1/
   index.html
web2/
    index.html
```

File Contents

web1/index.html

```
<h1>This is Web 1 Virtual Host</h1>
```

web2/index.html

```
<h1>This is Web 2 Virtual Host</h1>
```

nginx/nginx.conf

```
events {}

http {
    server {
        listen 80;
        server_name web1.domain.com;

        location / {
            proxy_pass http://web1:80;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }

    server {
        listen 80;
        server_name web2.domain.com;

        location / {
            proxy_pass http://web2:80;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }
}
```

docker-compose.yml

```
version: '3'

services:
  web1:
    image: nginx
    container_name: vh_web1
```

```
    volumes:
      - ./web1:/usr/share/nginx/html:ro

  web2:
    image: nginx
    container_name: vh_web2
    volumes:
      - ./web2:/usr/share/nginx/html:ro

  proxy:
    image: nginx
    container_name: vh_proxy
    ports:
      - "80:80"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - web1
      - web2
```

Steps Executed

```
cd /path/to/virtual-hosts
docker-compose up -d --build
docker ps
# verify containers: vh_web1, vh_web2, vh_proxy

curl -H "Host: web1.domain.com" http://localhost
curl -H "Host: web2.domain.com" http://localhost

docker-compose down
```

C — Local DNS (Simulated Domains)

Windows Hosts File Configuration

```
C:\Windows\System32\drivers\etc\hosts

142.93.249.208 web1.domain.com
142.93.249.208 web2.domain.com
```

Here the IP Address before the domain is the ¡digital-ocean-droplet-ipv4-addr¿

```
ipconfig /flushdns
```

Open in browser:

- http://web1.domain.com
- http://web2.domain.com

Linux / Unix Hosts File Configuration

```
sudo nano /etc/hosts
```

Add:

```
142.93.249.208 web1.domain.com
142.93.249.208 web2.domain.com
```

Here the IP Address before the domain is the ¡digital-ocean-droplet-ipv4-addr¿
Save and apply:

```
sudo systemctl restart NetworkManager
# or
sudo dscacheutil -flushcache  # macOS
```

Test:

```
ping web1.domain.com
ping web2.domain.com
```

Open in browser:

- http://web1.domain.com
- http://web2.domain.com

Executed Outputs

Figure 9.1: Load Balancer Response 1

Figure 9.2: Load Balancer Response 2

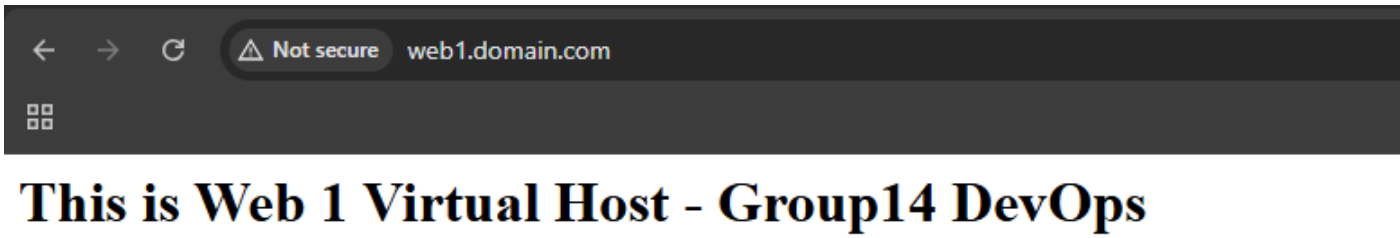

Figure 9.3: Virtual Host Response 1

Figure 9.4: Virtual Host Response 2

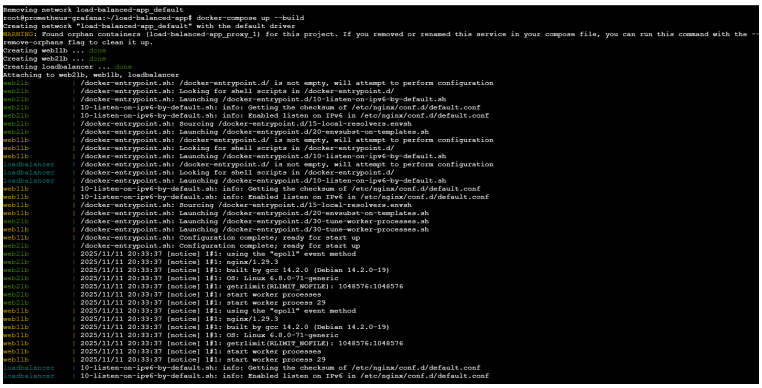

Figure 9.5: Inside Droplet 1

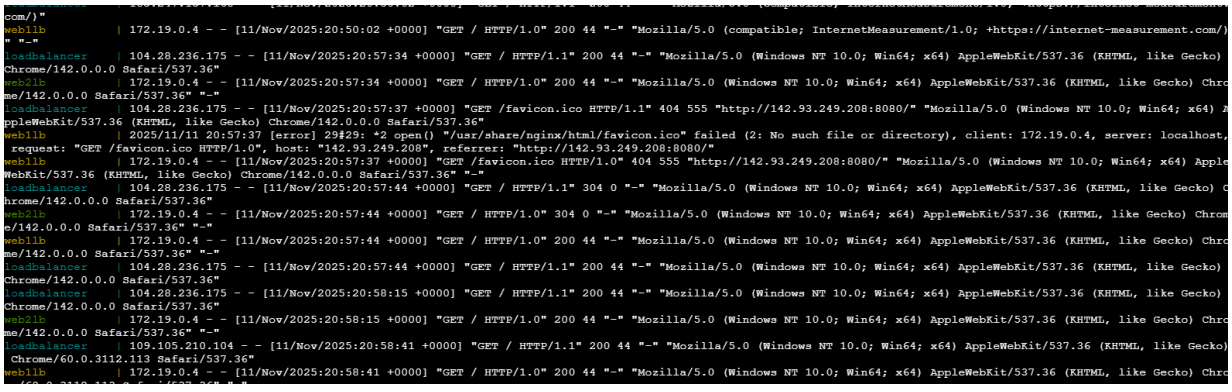

Figure 9.6: Inside Droplet 2

Figure 9.7: Docker Processes running

Glossary

CI/CD Continuous Integration and Continuous Deployment - automated software development practices that enable frequent code integration and deployment. 8

Bibliography

Index