DevOps - Group 14 Assignments

Document Version: e14649b

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**
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.

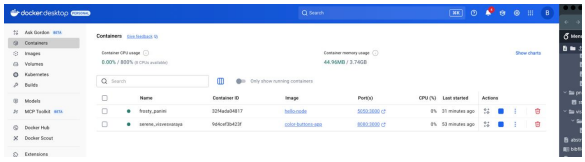

Figure 3.1: AWS Image 1

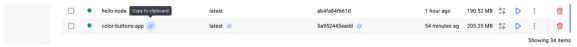

Figure 3.2: AWS Image 2

Figure 3.3: AWS Image 3

Figure 3.4: AWS Image 4

3.9 Class-Based JavaScript Implementation

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

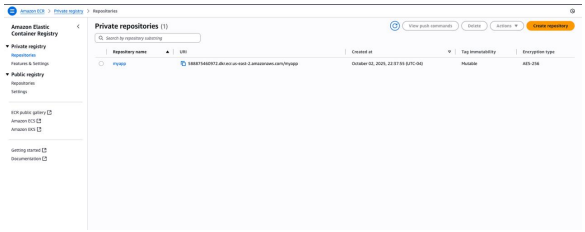

Figure 3.5: AWS Image 5

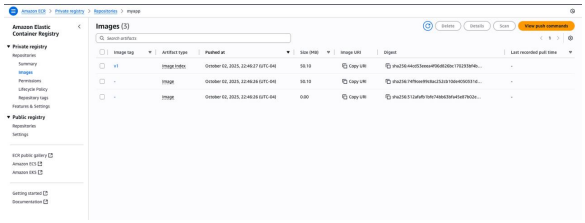

Figure 3.6: AWS Image 6

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:

```yaml
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 CI with GitHub Action

6.1 Introduction

This chapter documents the implementation of a CI/CD pipeline using GitHub Actions to automatically compile LaTeX documents, inject version information, and archive compiled PDFs.

6.2 Project Setup from Overleaf

6.2.1 Exporting Overleaf Project

The collaborative LaTeX project was initially developed on Overleaf. The project was exported and prepared for Git version control:

```
# Download project from Overleaf as ZIP file
# File downloaded: overleaf-project.zip

# Extract the ZIP file
unzip overleaf-project.zip -d latex-project
cd latex-project

# Initialize Git repository
git init

# Create initial commit
git add .
git commit -m "Initial commit: Import from Overleaf"

# Create GitHub repository and link
git remote add origin
↪  https://github.com/Badri-Narayanan/overleaf-local-grp14.git
git branch -M main
git push -u origin main
```

6.2.2 Project Structure

After extraction, the project contains:

```
latex-project/
 groupAssignments.tex        # Main document
 itGlossary.tex              # Glossary definitions
 bibfile.bib                 # Bibliography
 history.tex                 # Revision history
 itPasswords.tex             # Service credentials
 linuxCommands.tex           # Chapter 1
 projectProposal.tex         # Chapter 2
 awsDeployment.tex           # Chapter 3
 digitalOceanSetup.tex       # Chapter 4
 overleafDeployment.tex      # Chapter 5
 png/                        # Images directory
```

6.3 GitHub Actions Workflow Setup

6.3.1 Workflow File Creation

Create `.github/workflows/label-project.yml`:

```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{\\\\gitversion}{${GI
      ↪ groupAssignments.tex
      echo "Updated version to: ${GIT_HASH}"
      grep "gitversion" groupAssignments.tex

  - name: Compile LaTeX document (Pass 1)
    run: |
      pdflatex -shell-escape -interaction=nonstopmode groupAssignments.tex
      ↪ || true

  - name: Run BibTeX
    run: |
      bibtex groupAssignments || true

  - name: Run MakeIndex
    run: |
      makeindex groupAssignments.idx || true

  - name: Compile LaTeX document (Pass 2)
    run: |
      pdflatex -shell-escape -interaction=nonstopmode groupAssignments.tex
      ↪ || true

  - name: Compile LaTeX document (Pass 3 - Final)
    run: |
      pdflatex -shell-escape -interaction=nonstopmode groupAssignments.tex
      ↪ || true

  - name: Check if PDF was generated
    run: |
      if [ -f "groupAssignments.pdf" ]; then
        echo "PDF generated successfully!"
        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 to commit"
git push origin HEAD:main || echo "Push failed"
```

6.3.2 Key Workflow Components

Critical Flags:

- `-shell-escape`: Required for minted package
- `-interaction=nonstopmode`: Non-interactive compilation
- `|| true`: Continue on warnings

Dependencies:

- `texlive-full`: Complete LaTeX distribution
- `python3-pygments`: Required for minted syntax highlighting

6.4 LaTeX Document Configuration

6.4.1 Preamble Setup

Modify `groupAssignments.tex` preamble:

```
\documentclass[12pt,a4paper]{report}
\usepackage[utf8]{inputenc}
\usepackage[margin=1in]{geometry}
\usepackage{titling}
\usepackage{array}
\usepackage{booktabs}
\usepackage{fancyhdr}
\usepackage{minted}
\usepackage{graphicx}
\usepackage{url}
\usepackage{longtable}
\usepackage{enumitem}
\usepackage{makeidx}

% Hyperref configuration (removed pdfmark option)
\usepackage[
breaklinks=true,
colorlinks=true,
citecolor=blue,
linkcolor=blue,
menucolor=black,
urlcolor=blue
```

```
]{hyperref}

% Glossaries configuration (removed unsupported options)
\usepackage[toc]{glossaries}

% Load glossary definitions
\input{itGlossary.tex}
\makenoidxglossaries

% Make index
\makeindex

% Git version command - updated by GitHub Actions
\newcommand{\gitversion}{Unknown}

% Header/Footer setup
\pagestyle{fancy}
\fancyhf{}
\fancyfoot[C]{\thepage}
\fancyhead[L]{v\gitversion}
\fancyhead[C]{Chapter \thechapter \hspace{1em} \copyright Stevens -- \today
↪  \hspace{1em} -- Do Not Distribute!}
\setlength{\headheight}{15pt}
\renewcommand{\headrulewidth}{0pt}
```

6.4.2 Title Page with Version

```
\begin{titlepage}
    \centering
    \vspace*{5cm}

    {\Huge\bfseries DevOps - Group 14 Assignments\par}

    \vspace{0.5cm}
    {\large Document Version: \gitversion\par}

    \vspace{1cm}

    {\Large by\par}

    \vspace{0.5cm}

    {\Large
    Andre Santiago-Neyra\\
    Badri Narayanan Rajendran\\
    }

    \vspace{1cm}

    {\large Stevens.edu\par}
```

```
    \vfill

    {\large October 2, 2025\par}

\end{titlepage}
```

6.5 Glossary Configuration

6.5.1 Adding Glossary Entries

In `itGlossary.tex`, add required entries:

```
\newglossaryentry{cicd}{
    name={CI/CD},
    description={Continuous Integration and Continuous Deployment - automated
    ↪ software development practices for frequent code integration and
    ↪ deployment}
}
```

6.5.2 Using Glossary Entries

In document text:

```
... deployment processes using two \gls{cicd} tools: Jenkins and GitHub
↪ Actions...
```

6.6 Implementation Steps

6.6.1 Step 1: Export and Setup from Overleaf

```
# Download ZIP from Overleaf
# Extract and initialize repository
unzip overleaf-project.zip -d latex-project
cd latex-project
git init
git add .
git commit -m "Initial commit from Overleaf"
```

6.6.2 Step 2: Create GitHub Repository

```
# Create repository on GitHub (via web interface)
# Link local repository to GitHub
git remote add origin
↪ https://github.com/Badri-Narayanan/overleaf-local-grp14.git
git branch -M main
git push -u origin main
```

6.6.3 Step 3: Create Workflow File

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

# Create workflow file
touch .github/workflows/label-project.yml

# Add workflow YAML content (as shown in Section 3.1)
```

6.6.4 Step 4: Update LaTeX Preamble

Modifications to `groupAssignments.tex`:

1. Remove `pdfmark` from hyperref options
2. Change glossaries package: `\usepackage[toc]{glossaries}`
3. Add version command: `\newcommand{\gitversion}{Unknown}`
4. Set header height: `\setlength{\headheight}{15pt}`
5. Add version to title page and header

6.6.5 Step 5: Fix Glossary Issues

```
# Edit itGlossary.tex
# Add missing glossary entry for 'cicd'
```

Add to `itGlossary.tex`:

```
\newglossaryentry{cicd}{
    name={CI/CD},
    description={Continuous Integration and Continuous Deployment}
}
```

6.6.6 Step 6: Commit and Push Changes

```
# Stage all changes
git add .github/workflows/label-project.yml
git add groupAssignments.tex
git add itGlossary.tex

# Commit changes
git commit -m "Add GitHub Actions CI/CD for LaTeX compilation"

# Push to trigger workflow
git push origin main
```

6.6.7 Step 7: Monitor Workflow Execution

1. Navigate to GitHub repository
2. Click "Actions" tab
3. View real-time workflow execution
4. Wait for green checkmark (3-5 minutes)

6.6.8 Step 8: Access Compiled PDF

Method 1 - Workflow Artifacts:

```
# Via GitHub web interface:
# Actions → Select workflow run → Artifacts → Download "Compiled-PDF"
```

Method 2 - Repository Builds Directory:

```
# Navigate to builds/ directory in repository
# Access: builds/document_[commit-hash].pdf
# Example: builds/document_f3f37c0.pdf
```

Glossary

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

Bibliography

Index