Cloud Server Project

Global IP address: 54.172.7.187

DNS: www.samonshahzad.com

Name: Samon Shahzad

Student ID: 35562778

Introduction

This document outlines the complete setup and configuration of a cloud-based web server using AWS EC2, with Nginx as a reverse proxy for a Flask application. The server is accessible via a custom domain name (www.samonshahzad.com) with secure HTTPS access. This documentation is designed to allow replication of the server environment from scratch if needed, and includes all necessary commands, configurations, and explanations.

Project Overview

The deployed application is an event planning and management platform called “samoonEvents”. It allows users to create and manage events, handle guest lists and invitations, coordinate with vendors, track tasks, and manage budgets. The application demonstrates a complete web stack with a Flask backend, SQLAlchemy for database operations, and a responsive frontend using Tailwind CSS, all hosted on AWS infrastructure.

Setting up a Web Server

Launching an AWS EC2 Instance

1. Sign in to AWS Management Console and navigate to EC2 service
2. Click “Launch Instance”
3. Select Ubuntu Server 22.04 LTS
4. Choose an instance type (t2.micro is eligible for free tier)
5. Configure security groups to allow traffic:
   - SSH (Port 22) from your IP
   - HTTP (Port 80) from anywhere
   - HTTPS (Port 443) from anywhere
   - Custom TCP (Port 5000) from anywhere (for Flask development)

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
-	sgr-055c67e34d6d6081a	IPv4	SSH	TCP	22	0.0.0.0/0
-	sgr-04fcda45fbdc384f7	IPv4	Custom TCP	TCP	5000	0.0.0.0/0
-	sgr-0870d336d0ef9d250	IPv4	HTTPS	TCP	443	0.0.0.0/0
-	sgr-0e562171a34f16b03	IPv4	HTTP	TCP	80	0.0.0.0/0

Setting up SSH Access

Generate an SSH key pair during instance creation or use an existing one. After downloading the `.pem` file, set the correct permissions:

```
mv samon-key.pem ~/.ssh/
chmod 600 ~/.ssh/samon-key.pem
```

Create an SSH config file for easier access:

```
vim ~/.ssh/config
```

Add the following configuration:

```
Host samoon-server
    HostName 54.172.7.187
    User ubuntu
    IdentityFile ~/.ssh/samon-key.pem
```

Now you can connect with:

```
ssh samoon-server
```

Initial Server Setup

Update the system packages:

```
sudo apt update
sudo apt upgrade -y
```

For AWS instances, they come with some essential packages installed such as python, git, mysql, etc.

Setting up Nginx

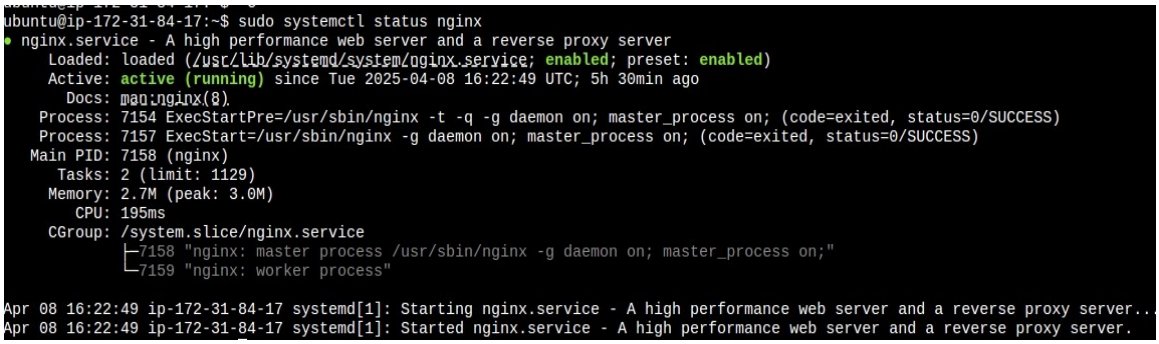
Install Nginx:

```
sudo apt install nginx -y
```

Verify Nginx is running:

```
sudo systemctl status nginx
```

You should see output indicating that Nginx is active (running):



Configuring the Flask Application

Cloning the Application Repository

```
cd ~
git clone https://github.com/samonShahzad/samoonEvents.git
cd samoonEvents
```

Setting up a Virtual Environment

```
python3 -m venv .venv
source .venv/bin/activate
```

Install dependencies:

```
pip install -r requirements.txt
pip install gunicorn
```

Creating a WSGI Entry Point

The application already includes a `wsgi.py` file with the following code:

```
from samoonEvents.app import app

if __name__ == "__main__":
    app.run()
```

Configuring Nginx as a Reverse Proxy

Create a new Nginx configuration file:

```
sudo vim /etc/nginx/sites-available/samoonevents
```

Add the following configuration:

```
server {
    listen 80;
    server_name 54.172.7.187 www.samonshahzad.com;

    location /static {
        alias /home/ubuntu/samoonEvents/samoonEvents/static;
    }

    location / {
        proxy_pass http://localhost:5000;
        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;
    }
}
```

Enable the configuration by creating a symbolic link:

```
sudo ln -s /etc/nginx/sites-available/samoonevents /etc/nginx/sites-enabled
```

Test Nginx configuration:

```
sudo nginx -t
```

If the test is successful, restart Nginx:

```
sudo systemctl restart nginx
```

Creating a Systemd Service File

Create a service file to automatically start the Flask application with Gunicorn:

```
sudo vim /etc/systemd/system/samoonevents.service
```

Add the following configuration:

```
[Unit]
Description=Gunicorn instance to serve samoonEvents Flask application
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/samoonEvents
Environment="PATH=/home/ubuntu/samoonEvents/.venv/bin"
ExecStart=/home/ubuntu/samoonEvents/.venv/bin/gunicorn --workers 3 --bind 0.0.0.0:5000 wsgi:app
Restart=always

[Install]
WantedBy=multi-user.target
```

Start and enable the service:

```
sudo systemctl start samoonevents
sudo systemctl enable samoonevents
sudo systemctl status samoonevents
```

Setting up the Database

Initialize and create the database:

```
cd ~/samoonEvents
source .venv/bin/activate
export FLASK_APP=samoonEvents.app
flask db init
```

```
flask db migrate -m "Initial migration"
flask db upgrade
```

Populate the database with sample data:

```
python -c samoonEvents.generate_data
```

Linking with a DNS Entry

Domain Registration with Namecheap

1. Register a domain with Namecheap (www.samonshahzad.com)
2. Navigate to the Domain List and select “Manage”
3. Go to the “Advanced DNS” tab

Setting up DNS Records

Create an A Record that points to your EC2 instance:

- Type: A Record
- Host: @ (for root domain)
- Value: 54.172.7.187 (your EC2 IP address)
- TTL: 30 minutes

Create an A Record for www subdomain:

- Type: A Record
- Host: www
- Value: 54.172.7.187 (your EC2 IP address)
- TTL: 30 minutes

Allow time for DNS propagation (typically 30 minutes to 48 hours).

SSL/TLS Configuration

Install Certbot:

```
sudo apt install certbot python3-certbot-nginx -y
```

Obtain and install SSL certificate:

```
sudo certbot --nginx -d www.samonshahzad.com -d samonshahzad.com
```

Follow the prompts, providing your email and accepting the terms of service.

Certbot will automatically update your Nginx configuration to redirect HTTP traffic to HTTPS and set up the SSL certificate.

Verify the automatic renewal:

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

Custom Scripts

Application Monitoring and Maintenance Script

Below is a custom Bash script that performs regular monitoring and maintenance of the application, including backups, log rotation, and health checks.

Create a file named `maintain_samoonevents.sh` in your home directory:

```
vim ~/maintain_samoonevents.sh
```

Add the following script:

```
#!/bin/bash

# samoonEvents Maintenance Script
# This script performs routine maintenance tasks for the samoonEvents application
# Author: Samon Shahzad
# Created: April 2025

# Configuration
APP_DIR="/home/ubuntu/samoonEvents"
BACKUP_DIR="/home/ubuntu/backups"
LOG_DIR="/home/ubuntu/logs"
DATE=$(date +%Y-%m-%d_%H-%M-%S)
BACKUP_FILENAME="samoonevents_backup_$DATE.tar.gz"
MAIN_LOG="$LOG_DIR/maintenance_log.txt"
APP_DB="$APP_DIR/samoonEvents/app.db"
EMAIL="35562778@student.murdoch.edu.au"

# Create directories if they don't exist
mkdir -p "$BACKUP_DIR" "$LOG_DIR"

# Log function
log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$MAIN_LOG"
}

log "Starting maintenance tasks for samoonEvents"

# Check if application service is running
if systemctl is-active --quiet samoonevents; then
    log "Service check: samoonevents service is running"
else
     log "WARNING: samoonevents service is not running! Attempting to restart..."
    sudo systemctl restart samoonevents
```

```
    # Check if restart was successful
    if systemctl is-active --quiet samoonevents; then
            log "Service recovery: Successfully restarted samoonevents
service"
    else
        log "ERROR: Failed to restart samoonevents service!"
        echo "samoonEvents service is down and could not be restarted!"
| mail -s "ALERT: samoonEvents Service Down" "$EMAIL"
    fi
fi

# Perform database backup
log "Starting database backup..."
if [ -f "$APP_DB" ]; then
    # Create backup directory for this date if it doesn't exist
    mkdir -p "$BACKUP_DIR/db_backups"

    # Copy the SQLite database file
    cp "$APP_DB" "$BACKUP_DIR/db_backups/app_db_$DATE.sqlite"

    if [ $? -eq 0 ]; then
        log "Database backup successful: app_db_$DATE.sqlite"
    else
        log "ERROR: Database backup failed!"
    fi
else
    log "ERROR: Database file not found at $APP_DB"
fi

# Create application backup
log "Creating application backup..."
tar -czf "$BACKUP_DIR/$BACKUP_FILENAME" "$APP_DIR" 2>/dev/null

if [ $? -eq 0 ]; then
    log "Application backup successful: $BACKUP_FILENAME"

    # Clean up old backups (keep only the 7 most recent)
    log "Cleaning up old backups..."
    ls -t "$BACKUP_DIR"/*.tar.gz | tail -n +8 | xargs -r rm
    log "Retained most recent 7 backups"
else
    log "ERROR: Application backup failed!"
fi

# Check disk space
DISK_USAGE=$(df -h / | awk 'NR==2 {print $5}' | sed 's/%//')
if [ "$DISK_USAGE" -gt 85 ]; then
    log "WARNING: Disk space is critical: ${DISK_USAGE}%"
     echo "Disk space on samoonEvents server is at ${DISK_USAGE}%" |
```

```
mail -s "ALERT: Low Disk Space on samoonEvents Server" "$EMAIL"
else
    log "Disk space check: ${DISK_USAGE}% used (OK)"
fi

# Check for system updates
log "Checking for system updates..."
sudo apt update &>/dev/null
UPDATES=$(apt list --upgradable 2>/dev/null | grep -c "upgradable")
SECURITY_UPDATES=$(apt list --upgradable 2>/dev/null | grep -c
"security")

if [ "$SECURITY_UPDATES" -gt 0 ]; then
    log "WARNING: $SECURITY_UPDATES security updates available!"
      echo "$SECURITY_UPDATES security updates are available on the
samoonEvents server!" | mail -s "ALERT: Security Updates Available"
"$EMAIL"
else
    log "Security check: No security updates required"
fi

if [ "$UPDATES" -gt 0 ]; then
    log "System has $UPDATES packages that can be upgraded"
else
    log "System is up to date"
fi

# Rotate logs if they're getting too large
find "$LOG_DIR" -type f -name "*.log" -size +100M | while read
log_file; do
    log "Rotating large log file: $log_file"
    mv "$log_file" "${log_file}.1"
    touch "$log_file"
done

# Perform HTTP health check
HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}"
https://www.samonshahzad.com)
if [ "$HTTP_STATUS" -eq 200 ]; then
    log "Website health check: HTTP Status $HTTP_STATUS (OK)"
else
        log "WARNING: Website health check failed! HTTP Status:
$HTTP_STATUS"
    echo "Website health check failed with HTTP status $HTTP_STATUS" |
mail -s "ALERT: samoonEvents Website Down" "$EMAIL"
fi

log "Maintenance tasks completed for samoonEvents"
log "-------------------------------------------"
```

Make the script executable:

```
chmod +x ~/maintain_samoonevents.sh
```

Set up a cron job to run the script daily:

```
crontab -e
```

Add the following line to run the maintenance script at 3 AM every day:

```
0 3 * * * /home/ubuntu/maintain_samoonevents.sh
```

Functions of the script

This maintenance script performs several key functions:

1. Checks if the application service is running and attempts to restart it if not
2. Creates timestamped backups of both the application files and database
3. Implements a retention policy by keeping only the 7 most recent backups
4. Monitors disk space and sends email alerts if space is low
5. Checks for available system updates, particularly security updates
6. Performs log rotation to prevent logs from consuming too much disk space
7. Conducts a health check of the website by verifying HTTP status
8. Sends email alerts for critical issues
9. Maintains a detailed log of all maintenance operations

This comprehensive approach ensures the application remains healthy, secure, and backed up, with automated notifications when issues arise.

Server Testing and Validation

Verifying Nginx Configuration

Check if Nginx is properly configured and running:

```
sudo systemctl status nginx
```

Test the Nginx configuration for syntax errors:

```
sudo nginx -t
```

Verifying SSL Configuration

Check the SSL certificate information:

```
openssl s_client -connect www.samonshahzad.com:443 -servername www.samonshahzad.com
```

Verify the certificate is valid and properly configured.

Testing the Flask Application

Check if the Flask application service is running:

```
sudo systemctl status samoonevents
```

Verify the application is accessible through the domain:

```
curl -I https://www.samonshahzad.com
```

This should return a 200 OK status code.

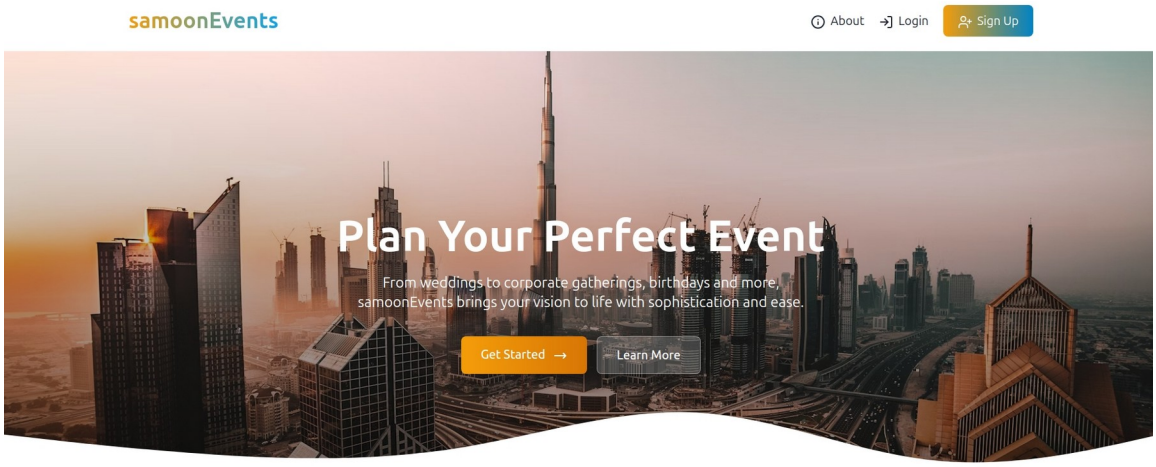
Testing Database Connection

Verify the application can connect to the database by checking for successful queries in the logs:

```
sudo journalctl -u samoonevents | grep -i "database"
```

Screenshots of Working Application

Homepage Screenshot



Maintenance Documentation

Regular Updates

To keep the server secure and up-to-date, run the following commands periodically:

```
sudo apt update
sudo apt upgrade -y
sudo apt autoremove -y
```

Application Updates

To update the samoonEvents application:

```
cd ~/samoonEvents
source .venv/bin/activate

# Backup the database first
cp samoonEvents/app.db samoonEvents/app.db.backup

# Pull latest code and update
git pull
pip install -r requirements.txt

# Apply any database migrations
export FLASK_APP=samoonEvents.app
flask db migrate -m "Update migration"
flask db upgrade

# Restart the service
sudo systemctl restart samoonevents
```

Database Maintenance

For SQLite database maintenance:

```
cd ~/samoonEvents
source .venv/bin/activate
sqlite3 samoonEvents/app.db "VACUUM;"
```

This command optimizes the database by rebuilding it to reclaim unused space.

Nginx and SSL Maintenance

Nginx configuration changes:

```
sudo vim /etc/nginx/sites-available/samoonevents
sudo nginx -t
sudo systemctl restart nginx
```

SSL certificate renewal (happens automatically with cron, but can be manually triggered):

```
sudo certbot renew
```

Cost Analysis

The following table outlines the estimated monthly costs for maintaining this server:

Service	Configuration	Monthly Cost (USD)
AWS EC2	t2.micro	$8.50 (on-demand) or Free Tier eligible
Domain Name	samonshahzad.com	$\sim 10-15/year (\approx 1.25/month)$
SSL Certificate	Let's Encrypt	Free
Data Transfer	100 GB/month	~$9.00
Total Estimated Cost		**~$18.75/month**

Note: Costs may vary based on actual usage and AWS pricing changes. Free tier eligibility can significantly reduce first-year costs.

Cost Optimization Strategies

1. **Use Reserved Instances**: For long-term deployments, reserved instances can reduce EC2 costs by up to 75%
2. **Scale appropriately**: The t2.micro instance is sufficient for development and low traffic, but may need to be upgraded for production
3. **Monitor data transfer**: Set up AWS CloudWatch alarms to monitor and alert on excessive data transfer usage
4. **Use AWS Free Tier**: Take advantage of AWS Free Tier offerings for the first 12 months

Troubleshooting Common Issues

Nginx 502 Bad Gateway Error

This typically means Gunicorn isn't running or accessible.

Check the Flask application service:

```
sudo systemctl status samoonevents
```

If the service is failing, check the logs:

```
sudo journalctl -u samoonevents
```

Common solutions:

1. Check that the `wsgi.py` file is correctly configured
2. Ensure the virtual environment paths in the service file are correct
3. Verify that all dependencies are installed:
   ```
   source .venv/bin/activate
   pip install -r requirements.txt
   ```

Database Connection Issues

If the application cannot connect to the database:

1. Check the database file exists:

   ```
   ls -la ~/samoonEvents/samoonEvents/app.db
   ```

2. Verify the permissions:

   ```
   sudo chown -R ubuntu:ubuntu ~/samoonEvents
   ```

3. Check for database lock issues:

   ```
   find ~/ -name "app.db-*"
   ```

   If lock files exist, you may need to remove them and restart the application.

SSL Certificate Issues

If your certificate isn't renewing or shows as invalid:

Check Certbot certificates:

```
sudo certbot certificates
```

Try manual renewal:

```
sudo certbot renew --force-renewal
```

Check Nginx SSL configuration:

```
grep -r "ssl" /etc/nginx/sites-available/
```

Application Loading Errors

If the application loads but has errors:

1. Check the application logs:

   ```
   sudo journalctl -u samoonevents
   ```

2. Test the application directly with Gunicorn:

   ```
   cd ~/samoonEvents
   source .venv/bin/activate
   gunicorn --bind 0.0.0.0:5000 wsgi:app
   ```

3. Check for static file issues:

   ```
   sudo ls -la /etc/nginx/sites-available/samoonevents
   sudo ls -la ~/samoonEvents/samoonEvents/static
   ```

References

1. AWS Documentation. “Amazon EC2 User Guide.” Retrieved from https://docs.aws.amazon.com/ec2/
2. Nginx Documentation. “Beginner’s Guide.” Retrieved from https://nginx.org/en/docs/beginners_guide.html
3. Flask Documentation. “Deployment Options.” Retrieved from https://flask.palletsprojects.com/en/2.2.x/deploying/
4. SQLAlchemy Documentation. “SQLAlchemy 2.0 Documentation.” Retrieved from https://docs.sqlalchemy.org/en/20/
5. Let’s Encrypt. “Getting Started.” Retrieved from https://letsencrypt.org/getting-started/
6. DigitalOcean Community Tutorials. “How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 20.04.” Retrieved from https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-20-04
7. Certbot Documentation. “User Guide.” Retrieved from https://certbot.eff.org/docs/using.html
8. Python Documentation. “venv — Creation of virtual environments.” Retrieved from https://docs.python.org/3/library/venv.html