trooper.stsgym.com Authentication Platform

CONTENTS

trooper.stsgym.com: Authentication and VM Provisioning Platform

Abstract

This paper documents the design, implementation, and deployment of trooper.stsgym.com, a Flask-based authentication and virtual machine provisioning platform. The system provides secure user authentication with email verification, role-based access control, and automated VM provisioning on remote hosts via SSH. Built with containerization-first principles, the platform leverages Docker for deployment isolation, PostgreSQL for data persistence, and Redis for session management and rate limiting. The platform serves as a gateway for users to request and manage virtual machines on a development infrastructure, with administrative oversight and audit logging.

1. Introduction

1.1 Background

As development environments grow in complexity, the need for automated infrastructure provisioning becomes increasingly important. Traditional VM provisioning requires manual intervention, SSH access management, and coordination between users and administrators. trooper-auth-extended addresses these challenges by providing a self-service portal where authenticated users can request virtual machines on a development host.

1.2 Objectives

- Provide secure user authentication with email verification
- Implement role-based access control (RBAC)
- Automate VM provisioning on remote hosts

- Integrate with existing infrastructure (SMTP, DNS, cloud services)
- Maintain comprehensive audit logging
- Support administrative workflows for user approval

1.3 Scope

This paper covers: - System architecture and design decisions - Authentication and authorization implementation - VM provisioning workflow - Security considerations - Deployment and operational procedures

2. System Architecture

2.1 High-Level Architecture

The system follows a three-tier architecture:

2.2 Component Breakdown

2.2.1 Flask Application

The core application is built on Flask with the following extensions:

Extension	Purpose
`Flask-Login`	User session management
`Flask-SQLAlchemy`	ORM for PostgreSQL
`Flask-Limiter`	Rate limiting
`Flask-Mail`	Email notifications

2.2.2 Database Schema

2.3 Deployment Architecture

The application is deployed on `miner` (the production server) using Docker containers:

```
# docker-compose.yml structure
services:
  web:      # Flask application (port 5004)
  db:       # PostgreSQL 15
  redis:    # Redis 7 for rate limiting
```

Nginx proxies `trooper.stsgym.com` to `127.0.0.1:5004`.

3. Authentication and Authorization

3.1 User Registration Flow

3.2 Role-Based Access Control

Role	Permissions
`user`	Dashboard, VM panel, Chat
`admin`	All user permissions + User management, Approve/deny users, View audit logs

3.3 Security Features

1. **Password Security**: bcrypt hashing via werkzeug
2. **Session Security**: Flask-Login with secure cookies
3. **Rate Limiting**: 200/day, 50/hour per IP
4. **Account Lockout**: 5 failed attempts → 30 minute lockout
5. **Email Verification**: Required before admin approval
6. **Audit Logging**: All actions logged with user, action, timestamp

4. VM Provisioning

4.1 Provisioning Workflow

4.2 VM Configuration Defaults

Setting	Value
Base Image	`/var/lib/libvirt/images/ubuntu-24.04-cloud.img`
Memory	8192 MB (8 GB)
vCPUs	4
Network	`default` (NAT)
Disk Format	qcow2

4.3 SSH Connection

The application uses SSH keys to connect to `trooper2`:

```
# Configuration
VM_HOST = "${VPN_HOST}"  # trooper2 via Tailscale
VM_HOST_USER = "wez"
VM_SSH_KEY = "/app/ssh_key"  # Mounted in container
```

5. Implementation Details

5.1 Directory Structure

5.2 Key Dependencies

```
Flask==3.0.0
Flask-Login==0.6.3
Flask-SQLAlchemy==3.1.1
Flask-Limiter==3.5.0
Flask-Mail==0.9.1
psycopg2-binary==2.9.9
redis==5.0.1
gunicorn==21.2.0
```

5.3 Rate Limiting Implementation

```
limiter = Limiter(
    app=app,
    key_func=get_remote_address,
    default_limits=["200 per day", "50 per hour"],
    storage_uri="redis://redis:6379/0"
)

# Applied to routes
@app.route("/login", methods=["GET", "POST"])
@limiter.limit("10 per minute")
def login():
    # ...
```

6. Security Analysis

6.1 Threat Model

Threat	Mitigation
Brute force login	Account lockout after 5 failures, rate limiting
Session hijacking	Secure cookies, HTTPS enforced
SQL injection	SQLAlchemy parameterized queries

Threat	Mitigation
XSS	Jinja2 auto-escaping
CSRF	Flask-WTF tokens
Unauthorized VM access	Role checks, SSH key isolation

6.2 Security Best Practices

1. **Principle of Least Privilege**: Users only have access to their own VMs
2. **Defense in Depth**: Multiple layers of authentication and authorization
3. **Audit Trail**: All actions logged for forensic analysis
4. **Rate Limiting**: Prevents abuse and DoS attacks
5. **Email Verification**: Ensures valid contact for accountability

7. Operational Procedures

7.1 Deployment

```
        # Initial deployment
        git clone git@idm.wezzel.com:crab-meat-repos/trooper-au
extended.git
        cd trooper-auth-extended
        cp .env.example .env
        # Edit .env with production values
        docker-compose build
        docker-compose up -d

        # Initialize database
        docker exec trooper-auth-extended flask db upgrade

        # Create admin user
        docker exec -it trooper-auth-extended flask create-admi
```

7.2 Monitoring

```
        # Health check
        curl http://trooper.stsgym.com/health

        # Container status
        docker ps --filter "name=trooper-auth"

        # Application logs
```

```
        docker logs -f trooper-auth-extended

        # Database logs
        docker logs -f trooper-auth-extended-db
```

7.3 Backup

```
        # Database backup
        docker exec trooper-auth-extended-db pg_dump -U trooper
trooper > \
          backup_$(date +%Y%m%d).sql

        # Application backup
        tar -czf trooper-backup_$(date +%Y%m%d).tar.gz \
          /home/wez/trooper-auth-extended
```

7.4 Scaling Considerations

The architecture supports horizontal scaling:

1. **Application Layer**: Multiple Flask containers behind load balancer
2. **Database Layer**: PostgreSQL read replicas
3. **Cache Layer**: Redis cluster
4. **VM Provisioning**: Multiple VM hosts in round-robin

8. Future Enhancements

8.1 Planned Features

1. **Two-Factor Authentication**: TOTP-based 2FA
2. **VM Templates**: Multiple OS options for VM creation
3. **Resource Quotas**: Per-user VM limits
4. **Webhook Notifications**: Slack/Discord alerts
5. **API Endpoints**: REST API for programmatic access

8.2 Technical Improvements

1. **Database Migrations**: Flask-Migrate/Alembic
2. **Unit Tests**: pytest coverage
3. **CI/CD Pipeline**: GitLab CI integration
4. **Monitoring**: Prometheus/Grafana dashboards
5. **Log Aggregation**: ELK stack integration

9. Conclusion

trooper.stsgym.com demonstrates a practical implementation of a self-service infrastructure platform with robust authentication, authorization, and automated provisioning. The containerized deployment ensures consistency across environments, while the modular architecture allows for future expansion and integration with additional services.

The platform successfully addresses the challenges of: - Secure user onboarding with email verification - Administrative oversight for security compliance - Automated VM provisioning with minimal manual intervention - Comprehensive audit logging for accountability

10. References

- Flask Documentation: https://flask.palletsprojects.com/
- Flask-Login: https://flask-login.readthedocs.io/
- libvirt Documentation: https://libvirt.org/docs/
- PostgreSQL 15: https://www.postgresql.org/docs/15/
- Redis Documentation: https://redis.io/docs/

Appendix A: Configuration Reference

```
# Required Environment Variables
DATABASE_URL=postgresql://user:pass@host:port/db
SECRET_KEY=<random-secret-key>
SMTP_HOST=mail.example.com
SMTP_PORT=587
SMTP_USER=username
SMTP_PASSWORD=password
BASE_URL=https://trooper.stsgym.com
VM_HOST=${VPN_HOST}
VM_HOST_USER=wez
VM_MEMORY=8192
VM_VCPUS=4
```

Appendix B: API Reference

Endpoint	Method	Auth	Description
/	GET	No	Home page
/login	POST	No	Authenticate user
/register	POST	No	Create account
/dashboard	GET	Yes	User dashboard

Endpoint	Method	Auth	Description
`/vm/create`	POST	Yes	Request new VM
`/admin/users`	GET	Admin	List all users
`/admin/approve/<id>`	POST	Admin	Approve user
`/health`	GET	No	Health check

Last Updated: 2026-03-10
Repository: https://idm.wezzel.com/crab-meat-repos/trooper-auth-extended
Maintainer: Wesley Robbins