Chapter 11: Practical Exercises and Labs

Overview

This chapter provides hands-on exercises designed to reinforce the concepts learned throughout this guide. Each exercise builds upon previous knowledge and includes step-by-step instructions, expected outcomes, and troubleshooting tips.

Exercise Categories

🎯 Beginner Exercises (Chapters 1-4)

- Basic installation and setup
- First code review workflow
- Understanding core concepts

🎯 Intermediate Exercises (Chapters 5-7)

- Advanced features and workflows
- Integration with external systems
- Automation and scripting

🎯 Advanced Exercises (Chapters 8-10)

- Enterprise deployment
- Security hardening
- Performance optimization
- Troubleshooting scenarios

Lab Environment Setup

Virtual Lab Environment

```
#!/bin/bash
# setup-lab-environment.sh
# Creates a complete virtual lab for Gerrit exercises

# VM Requirements:
# - 4 GB RAM minimum
# - 20 GB disk space
# - Ubuntu 20.04 LTS or CentOS 8

LAB_DIR="/opt/gerrit-lab"
GERRIT_VERSION="3.8.4"

setup_lab_users() {
    echo "Setting up lab users..."
```

```
    # Create lab users
    for user in alice bob charlie admin; do
        if ! id "$user" &>/dev/null; then
            useradd -m -s /bin/bash "$user"
            echo "$user:password123" | chpasswd
            echo "Created user: $user"
        fi
    done

    # Create development teams
    groupadd -f developers
    groupadd -f reviewers
    groupadd -f admins

    usermod -aG developers alice
    usermod -aG developers bob
    usermod -aG reviewers charlie
    usermod -aG admins admin
}

setup_git_repositories() {
    echo "Setting up sample repositories..."

    mkdir -p "$LAB_DIR/sample-repos"
    cd "$LAB_DIR/sample-repos"

    # Sample Java project
    git init java-calculator
    cd java-calculator

    cat > Calculator.java << 'EOF'
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Division by zero");
        }
        return a / b;
    }
}
EOF
```

```
    cat > CalculatorTest.java << 'EOF'
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {
    private Calculator calc = new Calculator();

    @Test
    public void testAdd() {
        assertEquals(5, calc.add(2, 3));
    }

    @Test
    public void testSubtract() {
        assertEquals(1, calc.subtract(3, 2));
    }

    @Test
    public void testMultiply() {
        assertEquals(6, calc.multiply(2, 3));
    }

    @Test
    public void testDivide() {
        assertEquals(2, calc.divide(6, 3));
    }

    @Test(expected = IllegalArgumentException.class)
    public void testDivideByZero() {
        calc.divide(5, 0);
    }
}
EOF

    cat > pom.xml << 'EOF'
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>calculator</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>
```

```
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
EOF

    cat > README.md << 'EOF'
# Calculator Project

A simple calculator implementation for Gerrit exercises.

## Building

```bash
mvn clean compile
```

Testing

```bash
mvn test
```

Features

- Basic arithmetic operations
- Error handling for division by zero
- Comprehensive unit tests
EOF

git add .
git commit -m "Initial calculator implementation"

cd ..
```

Sample Python project

```
git init python-webapp
cd python-webapp

cat > app.py << 'EOF'
from flask import Flask, request, jsonify
```

app = Flask(**name**)

In-memory storage for demo

users = []

@app.route('/health', methods=['GET'])
def health_check():
return jsonify({"status": "healthy"}), 200

@app.route('/users', methods=['GET'])
def get_users():
return jsonify(users), 200

@app.route('/users', methods=['POST'])
def create_user():
data = request.get_json()

```
if not data or 'name' not in data or 'email' not in data:
    return jsonify({"error": "Name and email required"}), 400

user = {
    "id": len(users) + 1,
    "name": data['name'],
    "email": data['email']
}

users.append(user)
return jsonify(user), 201
```

@app.route('/users/int:user_id', methods=['GET'])
def get_user(user_id):
user = next((u for u in users if u['id'] == user_id), None)

```
if not user:
    return jsonify({"error": "User not found"}), 404

return jsonify(user), 200
```

if **name** == '**main**':
app.run(debug=True, host='0.0.0.0', port=5000)
EOF

```
cat > requirements.txt << 'EOF'
```

Flask==2.3.3
pytest==7.4.2
requests==2.31.0
EOF

```
cat > test_app.py << 'EOF'
```

import pytest
import json
from app import app

@pytest.fixture
def client():
app.config['TESTING'] = True
with app.test_client() as client:
yield client

def test_health_check(client):
response = client.get('/health')
assert response.status_code == 200
data = json.loads(response.data)
assert data['status'] == 'healthy'

def test_create_user(client):
user_data = {
"name": "John Doe",
"email": "john@example.com"
}

```
response = client.post('/users',
                       data=json.dumps(user_data),
                       content_type='application/json')

assert response.status_code == 201
data = json.loads(response.data)
assert data['name'] == user_data['name']
assert data['email'] == user_data['email']
assert 'id' in data
```

def test_get_users(client):
response = client.get('/users')
assert response.status_code == 200
EOF

```
cat > README.md << 'EOF'
```

Python Web Application

A simple Flask web application for Gerrit exercises.

Setup

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

Running

```
python app.py
```

Testing

```
pytest test_app.py
```

API Endpoints

- `GET /health` - Health check
- `GET /users` - List all users
- `POST /users` - Create a new user
- `GET /users/<id>` - Get user by ID
  EOF

  git add .
  git commit -m "Initial Flask web application"

  cd "$LAB_DIR"
  echo "Sample repositories created in $LAB_DIR/sample-repos"
  }

setup_docker_environment() {
echo "Setting up Docker environment..."

```
cat > "$LAB_DIR/docker-compose.yml" << 'EOF'
```

```
version: '3.8'

services:
gerrit:
image: gerritcodereview/gerrit:3.8.4
container_name: gerrit-lab
ports:
- "8080:8080"
- "29418:29418"
volumes:
- gerrit-data:/var/gerrit/review_site
environment:
- CANONICAL_WEB_URL=http://localhost:8080
restart: unless-stopped

postgresql:
image: postgres:15
container_name: postgres-lab
environment:
- POSTGRES_DB=reviewdb
- POSTGRES_USER=gerrit
- POSTGRES_PASSWORD=secret
volumes:
- postgres-data:/var/lib/postgresql/data
restart: unless-stopped

ldap:
image: osixia/openldap:1.5.0
container_name: ldap-lab
ports:
- "389:389"
- "636:636"
environment:
- LDAP_ORGANISATION=Example Corp
- LDAP_DOMAIN=example.com
- LDAP_ADMIN_PASSWORD=admin
volumes:
- ldap-data:/var/lib/ldap
- ldap-config:/etc/ldap/slapd.d
restart: unless-stopped

jenkins:
image: jenkins/jenkins:lts
container_name: jenkins-lab
ports:
```

- "8090:8080"
- "50000:50000"
volumes:
- jenkins-data:/var/jenkins_home
restart: unless-stopped

volumes:
gerrit-data:
postgres-data:
ldap-data:
ldap-config:
jenkins-data:
EOF

```
  echo "Docker environment configured"
  echo "Start with: docker-compose -f $LAB_DIR/docker-compose.yml up -d"
```

}

create_exercise_scripts() {
echo "Creating exercise helper scripts..."

```
  mkdir -p "$LAB_DIR/scripts"

  # Reset environment script
  cat > "$LAB_DIR/scripts/reset-environment.sh" << 'EOF'
```

#!/bin/bash

Reset lab environment for fresh exercises

echo "Resetting Gerrit lab environment..."

Stop Gerrit if running

sudo systemctl stop gerrit 2>/dev/null || true

Clean data directories

sudo rm -rf /opt/gerrit/review_site/git/*
sudo rm -rf /opt/gerrit/review_site/cache/*
sudo rm -rf /opt/gerrit/review_site/index/*

Reinitialize Gerrit

sudo -u gerrit java -jar /opt/gerrit/gerrit.war init -d /opt/gerrit/review_site --batch --no-auto-start

Start Gerrit

sudo systemctl start gerrit

echo "Environment reset complete"
echo "Gerrit will be available at http://localhost:8080 in a few moments"
EOF

```
# Quick setup script
cat > "$LAB_DIR/scripts/quick-setup.sh" << 'EOF'
```

#!/bin/bash

Quick setup for specific exercises

usage() {
echo "Usage: $0 [basic|advanced|enterprise]"
exit 1
}

if [$# -ne 1]; then
usage
fi

SETUP_TYPE=$1

case $SETUP_TYPE in
basic)
echo "Setting up basic Gerrit environment..."
Basic setup commands here
;;
advanced)
echo "Setting up advanced environment with integrations..."
Advanced setup commands here
;;
enterprise)
echo "Setting up enterprise environment..."
Enterprise setup commands here
;;
*)

usage
;;
esac
EOF

```
chmod +x "$LAB_DIR/scripts"/*.sh
```

}

main() {
echo "Setting up Gerrit Lab Environment"
echo "================================="

```
mkdir -p "$LAB_DIR"

setup_lab_users
setup_git_repositories
setup_docker_environment
create_exercise_scripts

echo ""
echo "Lab environment setup complete!"
echo "Lab directory: $LAB_DIR"
echo ""
echo "Next steps:"
echo "1. Start the Docker environment: docker-compose -f $LAB_DIR/docker-
compose.yml up -d"
echo "2. Wait for services to start (about 2 minutes)"
echo "3. Access Gerrit at http://localhost:8080"
echo "4. Begin with Exercise 1.1"
```

}

main "$@"

```
## Beginner Exercises

### Exercise 1.1: Basic Installation and Setup

**Objective:** Install and configure a basic Gerrit instance

**Prerequisites:** Linux/macOS system or Windows with WSL

**Time Required:** 30 minutes
```

```
**Steps:**

1. **Install Java and Dependencies**
   ```bash
   # Ubuntu/Debian
   sudo apt update
   sudo apt install openjdk-11-jdk git maven

   # CentOS/RHEL
   sudo yum install java-11-openjdk git maven
```

2. **Download and Initialize Gerrit**

```
wget https://gerrit-releases.storage.googleapis.com/gerrit-3.8.4.war
java -jar gerrit-3.8.4.war init -d ~/gerrit_testsite
```

3. **Configuration Exercise**
   - Modify `gerrit.config` to change the web port to 8081
   - Enable email notifications
   - Configure Git repository path

4. **Start Gerrit and Test**

```
~/gerrit_testsite/bin/gerrit.sh start
curl http://localhost:8081
```

Expected Outcome: Working Gerrit instance accessible via web browser

Troubleshooting Tips:

- Check Java version: `java -version`
- Verify port availability: `netstat -tulpn | grep 8081`
- Check logs: `tail -f ~/gerrit_testsite/logs/error_log`

Validation Script:

```
#!/bin/bash
# validate-exercise-1-1.sh

echo "Validating Exercise 1.1..."

# Check if Gerrit is running
if pgrep -f "GerritCodeReview" >/dev/null; then
    echo "✅ Gerrit process is running"
else
```
```

```
    echo "❌ Gerrit process not found"
    exit 1
fi

# Check web interface
if curl -s http://localhost:8081 >/dev/null; then
    echo "☑ Web interface accessible"
else
    echo "❌ Web interface not accessible"
    exit 1
fi

# Check configuration
if grep -q "httpd.listenUrl.*8081" ~/gerrit_testsite/etc/gerrit.config; then
    echo "☑ Port configuration correct"
else
    echo "❌ Port configuration incorrect"
    exit 1
fi

echo "🎉 Exercise 1.1 completed successfully!"
```

Exercise 1.2: Creating Your First Review

Objective: Create and submit your first code review

Prerequisites: Completed Exercise 1.1

Time Required: 45 minutes

Steps:

1. **Setup Git Identity**

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

2. **Clone Test Repository**

```
git clone http://admin@localhost:8081/All-Projects
cd All-Projects
```

3. **Install Commit Hook**

```
curl -Lo .git/hooks/commit-msg
http://admin@localhost:8081/tools/hooks/commit-msg
```

```
chmod +x .git/hooks/commit-msg
```

4. **Create and Submit Change**

```
# Create a test file
echo "Hello Gerrit World!" > hello.txt
git add hello.txt
git commit -m "Add hello world file

This is my first Gerrit review.

Bug: 123
Test: Manual verification"

# Push for review
git push origin HEAD:refs/for/master
```

5. **Review Process Exercise**
    - Access the web interface
    - Find your change
    - Add yourself as a reviewer
    - Leave comments
    - Give scores (+1/+2, -1/-2)
    - Submit the change

Expected Outcome: Successfully submitted and merged change

Learning Points:

- Understanding Change-Id generation
- Push to `refs/for/branch` syntax
- Review process workflow
- Commit message formatting

Exercise 1.3: Handling Review Feedback

Objective: Learn to address review comments and update changes

Prerequisites: Completed Exercise 1.2

Time Required: 30 minutes

Steps:

1. **Create Change with Issues**

```
echo "print('hello world')" > buggy_script.py
git add buggy_script.py
git commit -m "Add Python script with intentional issues"
git push origin HEAD:refs/for/master
```

2. **Simulate Review Comments**
    - Access the web interface
    - Add comments about code style
    - Request changes (-1 score)
3. **Address Feedback**

```
# Fix the issues
cat > buggy_script.py << 'EOF'
#!/usr/bin/env python3
"""
Hello World script for Gerrit exercise.
"""

def main():
    """Main function."""
    print("Hello, Gerrit World!")

if __name__ == "__main__":
    main()
EOF

# Amend the commit
git add buggy_script.py
git commit --amend
git push origin HEAD:refs/for/master
```

4. **Re-review and Submit**
    - Verify the new patch set
    - Approve and submit

Expected Outcome: Understanding of patch set workflow and change amendments

Intermediate Exercises

Exercise 2.1: Branch Management and Policies

Objective: Configure branch permissions and submission policies

Prerequisites: Administrative access to Gerrit

Time Required: 60 minutes

Steps:

1. **Create Development Branches**

```
# In your repository
git checkout -b develop
git push origin develop

git checkout -b release/1.0
git push origin release/1.0
```

2. **Configure Branch Permissions**
   - Navigate to Project Settings → Access
   - Configure different permissions for:
     - `refs/heads/master` (restricted)
     - `refs/heads/develop` (developers can push)
     - `refs/heads/release/*` (release managers only)

3. **Setup Submit Requirements**

```
# Add submit requirements to project.config
[submit-requirement "Code-Review"]
description = At least one maximum vote for Code-Review
submittableIf = label:Code-Review=MAX AND -label:Code-Review=MIN
canOverrideInChildProjects = true

[submit-requirement "Verified"]
description = Build and test verification
submittableIf = label:Verified=+1
canOverrideInChildProjects = true
```

4. **Test Branch Policies**
   - Create changes targeting different branches
   - Verify permission enforcement
   - Test submit requirements

Expected Outcome: Properly configured branch access control

Exercise 2.2: Plugin Configuration

Objective: Install and configure Gerrit plugins

Prerequisites: Gerrit administrative access

Time Required: 45 minutes

Steps:

1. **Install Download Commands Plugin**

```
# Download plugin
wget https://gerrit-ci.gerritforge.com/job/plugin-download-commands-
stable-3.8/lastSuccessfulBuild/artifact/bazel-bin/plugins/download-
commands/download-commands.jar

# Copy to plugins directory
cp download-commands.jar ~/gerrit_testsite/plugins/

# Restart Gerrit
~/gerrit_testsite/bin/gerrit.sh restart
```

2. **Configure Plugin Settings**

```
# Add to gerrit.config
[plugin "download-commands"]
command = checkout
command = cherry_pick
command = pull
command = format_patch
```

3. **Install and Configure Webhooks Plugin**

```
# Install webhooks plugin
wget -O webhooks.jar "https://example.com/webhooks.jar"
cp webhooks.jar ~/gerrit_testsite/plugins/

# Configure webhook
git config -f ~/gerrit_testsite/etc/gerrit.config \
  plugin.webhooks.url "http://localhost:3000/gerrit-webhook"
```

4. **Test Plugin Functionality**

    - Verify download commands appear in web UI
    - Test webhook delivery
    - Check plugin logs

Expected Outcome: Functional plugins enhancing Gerrit capabilities

Exercise 2.3: CI/CD Integration

Objective: Integrate Gerrit with Jenkins for automated verification

Prerequisites: Jenkins instance available

Time Required: 90 minutes

Detailed Setup:

1. **Jenkins Configuration**

```
// Jenkinsfile for Gerrit integration
pipeline {
    agent any

    triggers {
        gerrit customUrl: '',
                gerritProjects: [
                    [branches: [
                        [compareType: 'PLAIN', pattern: 'master']
                    ],
                    compareType: 'PLAIN',
                    disableStrictForbiddenFileVerification: false,
                    pattern: 'test-project']
                ],
                triggerOnEvents: [
                    patchsetCreated(),
                    changeMerged()
                ]
    }

    stages {
        stage('Checkout') {
            steps {
                git url:
"${GERRIT_SCHEME}://${GERRIT_HOST}:${GERRIT_PORT}/${GERRIT_PROJECT}",
                    branch: "${GERRIT_BRANCH}"
            }
        }

        stage('Build') {
            steps {
                script {
                    if (fileExists('pom.xml')) {
                        sh 'mvn clean compile'
                    } else if (fileExists('requirements.txt')) {
                        sh '''
                            python3 -m venv venv
                            source venv/bin/activate
                            pip install -r requirements.txt
                        '''
                    }
                }
```

```
            }
        }

        stage('Test') {
            steps {
                script {
                    if (fileExists('pom.xml')) {
                        sh 'mvn test'
                    } else if (fileExists('test_*.py')) {
                        sh '''
                            source venv/bin/activate
                            python -m pytest
                        '''
                    }
                }
            }
        }

        stage('Code Quality') {
            steps {
                script {
                    if (fileExists('pom.xml')) {
                        sh 'mvn checkstyle:check'
                    } else if (fileExists('*.py')) {
                        sh '''
                            source venv/bin/activate
                            pip install flake8
                            flake8 .
                        '''
                    }
                }
            }
        }
    }

    post {
        success {
            script {
                if (env.GERRIT_CHANGE_NUMBER) {
                    gerritReview labels: [Verified: +1],
                                 message: 'Build and tests passed
successfully'
                }
            }
        }
        failure {
            script {
                if (env.GERRIT_CHANGE_NUMBER) {
                    gerritReview labels: [Verified: -1],
                                 message: 'Build or tests failed'
                }
            }
        }
```

```
    }
}
```

2. **Gerrit Trigger Configuration**

```
# Install Gerrit Trigger plugin in Jenkins
# Configure connection:
# - Hostname: localhost
# - Frontend URL: http://localhost:8080
# - SSH Port: 29418
# - Username: jenkins
# - SSH Key: [generated key]
```

3. **Test Integration**
    - Create a change in Gerrit
    - Verify Jenkins build triggers
    - Check verification votes

Expected Outcome: Automated build verification on code changes

Advanced Exercises

Exercise 3.1: High Availability Setup

Objective: Configure Gerrit in HA mode with load balancing

Prerequisites: Multiple servers or VMs

Time Required: 3 hours

Architecture Setup:

```
#!/bin/bash
# ha-setup.sh - High Availability Setup

# Server Layout:
# - gerrit-01: Primary Gerrit instance
# - gerrit-02: Secondary Gerrit instance
# - postgres-01: Primary database
# - postgres-02: Database replica
# - haproxy-01: Load balancer

setup_database_replication() {
    echo "Setting up PostgreSQL replication..."

    # Primary database configuration
    cat > /etc/postgresql/15/main/postgresql.conf << 'EOF'
```

```
# Replication settings
wal_level = replica
max_wal_senders = 3
max_replication_slots = 3
archive_mode = on
archive_command = 'test ! -f /var/lib/postgresql/15/main/archive/%f && cp %p
/var/lib/postgresql/15/main/archive/%f'

# Performance settings
shared_buffers = 256MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
EOF

    # Setup replication user
    sudo -u postgres psql << 'EOF'
CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD 'replica_password';
EOF

    # Configure pg_hba.conf for replication
    echo "host replication replicator postgres-02/32 md5" >>
/etc/postgresql/15/main/pg_hba.conf
}

setup_haproxy() {
    echo "Configuring HAProxy..."

    cat > /etc/haproxy/haproxy.cfg << 'EOF'
global
    daemon
    maxconn 4096
    log stdout local0

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
    option httplog

frontend gerrit_web
    bind *:80
    bind *:443 ssl crt /etc/ssl/certs/gerrit.pem
    redirect scheme https if !{ ssl_fc }
    default_backend gerrit_web_servers

backend gerrit_web_servers
    balance roundrobin
    option httpchk GET /config/server/healthcheck~status
    server gerrit-01 gerrit-01:8080 check
    server gerrit-02 gerrit-02:8080 check backup
```

```
frontend gerrit_ssh
    mode tcp
    bind *:29418
    default_backend gerrit_ssh_servers

backend gerrit_ssh_servers
    mode tcp
    balance roundrobin
    server gerrit-01 gerrit-01:29418 check
    server gerrit-02 gerrit-02:29418 check backup

listen stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 30s
EOF
}

setup_gerrit_ha() {
    echo "Configuring Gerrit for HA..."

    # Shared configuration
    cat > /opt/gerrit/review_site/etc/gerrit.config << 'EOF'
[gerrit]
    serverId = gerrit-01
    canonicalWebUrl = https://gerrit.company.com/

[database]
    type = postgresql
    hostname = postgres-01
    port = 5432
    database = reviewdb
    username = gerrit

[auth]
    type = LDAP

[cache]
    directory = cache

[container]
    javaOptions = -Xms2g -Xmx4g
    javaOptions = -XX:+UseG1GC

[httpd]
    listenUrl = http://*:8080/

[sshd]
    listenAddress = *:29418

[receive]
    enableSignedPush = false
```

```
[sendemail]
    smtpServer = smtp.company.com
    smtpServerPort = 587
EOF

    # Configure shared Git directory
    echo "Setting up shared Git repositories..."
    # Use NFS or distributed filesystem for git directory
}

deploy_monitoring() {
    echo "Setting up monitoring..."

    # Prometheus configuration
    cat > /etc/prometheus/prometheus.yml << 'EOF'
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'gerrit'
    static_configs:
      - targets: ['gerrit-01:8080', 'gerrit-02:8080']
    metrics_path: '/config/server/metrics'

  - job_name: 'haproxy'
    static_configs:
      - targets: ['haproxy-01:8404']
    metrics_path: '/stats;csv'

  - job_name: 'postgres'
    static_configs:
      - targets: ['postgres-01:9187', 'postgres-02:9187']
EOF
}

main() {
    setup_database_replication
    setup_haproxy
    setup_gerrit_ha
    deploy_monitoring

    echo "HA setup completed"
    echo "Access Gerrit at: https://gerrit.company.com"
    echo "Monitor at: http://haproxy-01:8404/stats"
}

main "$@"
```

Testing HA Configuration:

```python
#!/usr/bin/env python3
# test-ha.py - Test HA functionality

import requests
import time
import subprocess
import sys

class HATest:
    def __init__(self, gerrit_url, haproxy_stats_url):
        self.gerrit_url = gerrit_url
        self.haproxy_stats_url = haproxy_stats_url

    def test_web_availability(self):
        """Test web interface availability"""
        try:
            response = requests.get(f"{self.gerrit_url}/config/server/version")
            if response.status_code == 200:
                print("☑ Web interface accessible")
                return True
            else:
                print(f"❌ Web interface returned {response.status_code}")
                return False
        except Exception as e:
            print(f"❌ Web interface connection failed: {e}")
            return False

    def test_ssh_availability(self):
        """Test SSH interface availability"""
        try:
            result = subprocess.run(
                ["ssh", "-p", "29418", "admin@gerrit.company.com", "gerrit",
"version"],
                capture_output=True, text=True, timeout=10
            )
            if result.returncode == 0:
                print("☑ SSH interface accessible")
                return True
            else:
                print(f"❌ SSH interface failed: {result.stderr}")
                return False
        except Exception as e:
            print(f"❌ SSH interface connection failed: {e}")
            return False

    def test_failover(self):
        """Test failover functionality"""
        print("Testing failover scenario...")

        # Get current active server
        try:
            response = requests.get(f"{self.haproxy_stats_url}/stats;csv")
```

```
            print("Current HAProxy status:")
            print(response.text)
        except Exception as e:
            print(f"Failed to get HAProxy stats: {e}")
            return False

        # Simulate primary server failure
        print("Simulating primary server failure...")
        # This would require actual server control

        # Test continued availability
        time.sleep(5)
        if self.test_web_availability():
            print("✅ Failover successful - service still available")
            return True
        else:
            print("❌ Failover failed - service unavailable")
            return False

    def test_database_replication(self):
        """Test database replication"""
        # This would require database-specific testing
        print("Testing database replication...")
        # Implementation depends on specific database setup
        return True

    def run_all_tests(self):
        """Run all HA tests"""
        tests = [
            ("Web Availability", self.test_web_availability),
            ("SSH Availability", self.test_ssh_availability),
            ("Database Replication", self.test_database_replication),
            ("Failover", self.test_failover)
        ]

        results = []
        for test_name, test_func in tests:
            print(f"\n--- {test_name} ---")
            result = test_func()
            results.append((test_name, result))

        print("\n--- Test Summary ---")
        for test_name, result in results:
            status = "✅ PASS" if result else "❌ FAIL"
            print(f"{test_name}: {status}")

        return all(result for _, result in results)

if __name__ == "__main__":
    tester = HATest(
        gerrit_url="https://gerrit.company.com",
        haproxy_stats_url="http://haproxy-01:8404"
    )
```

```
    success = tester.run_all_tests()
    sys.exit(0 if success else 1)
```

Exercise 3.2: Security Hardening Implementation

Objective: Implement comprehensive security measures

Prerequisites: Working Gerrit installation

Time Required: 2 hours

Security Implementation:

```
#!/bin/bash
# security-hardening.sh

implement_ssl_tls() {
    echo "Implementing SSL/TLS security..."

    # Generate SSL certificates
    openssl req -newkey rsa:4096 -x509 -sha256 -days 365 -nodes \
        -out /etc/ssl/certs/gerrit.crt \
        -keyout /etc/ssl/private/gerrit.key \
        -subj "/C=US/ST=State/L=City/O=Company/OU=IT/CN=gerrit.company.com"

    # Configure Gerrit for HTTPS
    cat >> /opt/gerrit/review_site/etc/gerrit.config << 'EOF'
[httpd]
    listenUrl = https://*:8443/
    sslKeyStore = /etc/ssl/certs/gerrit.jks
    sslKeyStorePassword = changeit

[sshd]
    ciphers = aes128-ctr,aes192-ctr,aes256-ctr
    macs = hmac-sha2-256,hmac-sha2-512
    kexAlgorithms = diffie-hellman-group14-sha256,ecdh-sha2-nistp256,ecdh-sha2-
nistp384,ecdh-sha2-nistp521
EOF

    # Convert to Java keystore
    openssl pkcs12 -export -in /etc/ssl/certs/gerrit.crt \
        -inkey /etc/ssl/private/gerrit.key \
        -out /tmp/gerrit.p12 -name gerrit -password pass:changeit

    keytool -importkeystore -deststorepass changeit \
        -destkeystore /etc/ssl/certs/gerrit.jks \
        -srckeystore /tmp/gerrit.p12 -srcstoretype PKCS12 \
        -srcstorepass changeit
}
```

```
implement_firewall() {
    echo "Configuring firewall rules..."

    # UFW configuration
    ufw --force reset
    ufw default deny incoming
    ufw default allow outgoing

    # Allow SSH
    ufw allow 22/tcp

    # Allow Gerrit HTTPS
    ufw allow 8443/tcp

    # Allow Gerrit SSH
    ufw allow 29418/tcp

    # Allow from specific networks only
    ufw allow from 10.0.0.0/8 to any port 5432  # Database access
    ufw allow from 192.168.0.0/16 to any port 5432

    ufw --force enable
}

implement_access_controls() {
    echo "Implementing access controls..."

    # Configure project.config for strict access
    cat > /tmp/project.config << 'EOF'
[access "refs/*"]
    read = group Administrators
    read = group Anonymous Users

[access "refs/heads/*"]
    label-Code-Review = -2..+2 group Developers
    label-Verified = -1..+1 group CI-Servers
    submit = group Submitters
    create = group Project Owners
    push = group Project Owners
    pushMerge = group Project Owners

[access "refs/for/refs/heads/*"]
    push = group Registered Users

[access "refs/meta/config"]
    read = group Project Owners
    push = group Project Owners

[submit-requirement "Code-Review"]
    description = At least one maximum vote for Code-Review
    submittableIf = label:Code-Review=MAX AND -label:Code-Review=MIN
    canOverrideInChildProjects = false
```

```
[submit-requirement "Verified"]
    description = Requires verification from CI
    submittableIf = label:Verified=+1
    canOverrideInChildProjects = false
EOF
}

implement_audit_logging() {
    echo "Setting up audit logging..."

    # Configure detailed logging
    cat >> /opt/gerrit/review_site/etc/gerrit.config << 'EOF'
[log]
    textLogging = true
    jsonLogging = true

[index]
    maxLimit = 10000

[change]
    maxFiles = 1000
    maxPatchSets = 1000
EOF

    # Setup log rotation
    cat > /etc/logrotate.d/gerrit << 'EOF'
/opt/gerrit/review_site/logs/*.log {
    daily
    missingok
    rotate 30
    compress
    notifempty
    create 644 gerrit gerrit
    postrotate
        systemctl reload gerrit
    endscript
}
EOF

    # Setup audit log monitoring
    cat > /opt/gerrit/audit-monitor.sh << 'EOF'
#!/bin/bash
# Monitor for suspicious activities

LOG_FILE="/opt/gerrit/review_site/logs/gerrit.log"
ALERT_EMAIL="security@company.com"

# Monitor for multiple failed logins
tail -f "$LOG_FILE" | while read line; do
    if echo "$line" | grep -q "Authentication failed"; then
        echo "$(date): Authentication failure detected: $line" >>
/var/log/gerrit-security.log
```

```
        # Count failures in last 5 minutes
        FAILURES=$(grep "Authentication failed" /var/log/gerrit-security.log |
\
                   grep "$(date '+%Y-%m-%d %H:%M')" | wc -l)

        if [ "$FAILURES" -gt 10 ]; then
            echo "ALERT: Multiple authentication failures detected" | \
                mail -s "Gerrit Security Alert" "$ALERT_EMAIL"
        fi
    fi
done &
EOF

    chmod +x /opt/gerrit/audit-monitor.sh
}

main() {
    implement_ssl_tls
    implement_firewall
    implement_access_controls
    implement_audit_logging

    echo "Security hardening completed"
    echo "Remember to:"
    echo "1. Update DNS to point to new HTTPS port"
    echo "2. Update client configurations"
    echo "3. Test all security measures"
    echo "4. Review audit logs regularly"
}

main "$@"
```

Exercise 3.3: Performance Optimization Challenge

Objective: Optimize Gerrit for high-volume enterprise use

Prerequisites: Gerrit with substantial data

Time Required: 4 hours

Performance Analysis and Optimization:

```
#!/usr/bin/env python3
# performance-optimizer.py

import subprocess
import json
import time
import psutil
import requests
from datetime import datetime, timedelta
```

```
class GerritPerformanceOptimizer:
    def __init__(self, gerrit_site, gerrit_url):
        self.gerrit_site = gerrit_site
        self.gerrit_url = gerrit_url
        self.metrics = {}

    def analyze_current_performance(self):
        """Analyze current Gerrit performance"""
        print("Analyzing current performance...")

        # JVM metrics
        gerrit_pid = self.get_gerrit_pid()
        if gerrit_pid:
            self.metrics['jvm'] = self.get_jvm_metrics(gerrit_pid)

        # Database metrics
        self.metrics['database'] = self.get_database_metrics()

        # Git repository metrics
        self.metrics['repositories'] = self.get_repository_metrics()

        # Web interface response times
        self.metrics['web_performance'] = self.get_web_performance()

        return self.metrics

    def get_gerrit_pid(self):
        """Get Gerrit process PID"""
        try:
            result = subprocess.run(['pgrep', '-f', 'GerritCodeReview'],
                                    capture_output=True, text=True)
            return int(result.stdout.strip()) if result.stdout.strip() else
None
        except:
            return None

    def get_jvm_metrics(self, pid):
        """Get JVM performance metrics"""
        metrics = {}

        try:
            # Heap usage
            jstat_output = subprocess.run(['jstat', '-gc', str(pid)],
                                          capture_output=True, text=True)
            if jstat_output.returncode == 0:
                lines = jstat_output.stdout.strip().split('\n')
                if len(lines) >= 2:
                    values = lines[1].split()
                    metrics['heap_used'] = float(values[2]) + float(values[3])
+ \
                                           float(values[5]) + float(values[7])
                    metrics['gc_count'] = int(values[11]) + int(values[13])
```

```
            # Thread count
            proc = psutil.Process(pid)
            metrics['thread_count'] = proc.num_threads()
            metrics['memory_percent'] = proc.memory_percent()
            metrics['cpu_percent'] = proc.cpu_percent(interval=1)

        except Exception as e:
            print(f"Error getting JVM metrics: {e}")

        return metrics

    def get_database_metrics(self):
        """Get database performance metrics"""
        metrics = {}

        try:
            # Database connection test
            start_time = time.time()
            # This would require database connection
            # result = psycopg2.connect(...)
            metrics['connection_time'] = time.time() - start_time

            # Query performance would be tested here
            metrics['slow_queries'] = 0  # Placeholder

        except Exception as e:
            print(f"Error getting database metrics: {e}")

        return metrics

    def get_repository_metrics(self):
        """Get Git repository metrics"""
        metrics = {}

        try:
            git_dir = f"{self.gerrit_site}/git"

            # Count repositories
            result = subprocess.run(['find', git_dir, '-name', '*.git', '-
type', 'd'],
                                    capture_output=True, text=True)
            repo_count = len(result.stdout.strip().split('\n')) if
result.stdout.strip() else 0
            metrics['repository_count'] = repo_count

            # Find large repositories
            large_repos = []
            for line in result.stdout.strip().split('\n'):
                if line:
                    size_result = subprocess.run(['du', '-sh', line],
                                                 capture_output=True, text=True)
                    if size_result.returncode == 0:
```

```
                        size = size_result.stdout.split()[0]
                        if 'G' in size:  # Gigabyte-sized repos
                            large_repos.append((line, size))

            metrics['large_repositories'] = large_repos

        except Exception as e:
            print(f"Error getting repository metrics: {e}")

        return metrics

    def get_web_performance(self):
        """Test web interface performance"""
        metrics = {}

        endpoints = [
            ('dashboard', '/dashboard/'),
            ('changes', '/q/status:open'),
            ('projects', '/admin/projects/'),
        ]

        for name, endpoint in endpoints:
            try:
                start_time = time.time()
                response = requests.get(f"{self.gerrit_url}{endpoint}",
timeout=30)
                response_time = time.time() - start_time

                metrics[f'{name}_response_time'] = response_time
                metrics[f'{name}_status_code'] = response.status_code

            except Exception as e:
                metrics[f'{name}_error'] = str(e)

        return metrics

    def generate_optimization_recommendations(self):
        """Generate optimization recommendations based on metrics"""
        recommendations = []

        jvm_metrics = self.metrics.get('jvm', {})

        # JVM recommendations
        if jvm_metrics.get('heap_used', 0) > 3000000:  # > 3GB
            recommendations.append({
                'category': 'JVM',
                'priority': 'HIGH',
                'issue': 'High heap usage detected',
                'recommendation': 'Increase heap size: -Xmx6g or higher',
                'config': '[container]\n    javaOptions = -Xmx6g'
            })

        if jvm_metrics.get('gc_count', 0) > 1000:
```

```
            recommendations.append({
                'category': 'JVM',
                'priority': 'MEDIUM',
                'issue': 'High GC count',
                'recommendation': 'Use G1GC for better performance',
                'config': '[container]\n    javaOptions = -XX:+UseG1GC'
            })

        if jvm_metrics.get('thread_count', 0) > 500:
            recommendations.append({
                'category': 'JVM',
                'priority': 'MEDIUM',
                'issue': 'High thread count',
                'recommendation': 'Check for thread leaks, tune thread pools',
                'config': '[sshd]\n    threads = 50\n[httpd]\n    maxThreads =
100'
            })

        # Repository recommendations
        repo_metrics = self.metrics.get('repositories', {})
        large_repos = repo_metrics.get('large_repositories', [])

        if large_repos:
            recommendations.append({
                'category': 'REPOSITORIES',
                'priority': 'MEDIUM',
                'issue': f'Found {len(large_repos)} large repositories',
                'recommendation': 'Run git maintenance on large repositories',
                'action': 'git gc --aggressive && git repack -ad'
            })

        # Web performance recommendations
        web_metrics = self.metrics.get('web_performance', {})
        slow_endpoints = []

        for key, value in web_metrics.items():
            if key.endswith('_response_time') and value > 5:  # > 5 seconds
                slow_endpoints.append(key.replace('_response_time', ''))

        if slow_endpoints:
            recommendations.append({
                'category': 'WEB',
                'priority': 'HIGH',
                'issue': f'Slow response times for: {",
".join(slow_endpoints)}',
                'recommendation': 'Optimize database queries and increase cache
sizes',
                'config': '[cache "web_sessions"]\n    memoryLimit =
1024\n[cache "projects"]\n    memoryLimit = 512'
            })

        return recommendations
```

```
    def apply_optimizations(self, recommendations):
        """Apply optimization recommendations"""
        print("Applying optimizations...")

        config_updates = []
        actions = []

        for rec in recommendations:
            if rec['priority'] in ['HIGH', 'MEDIUM']:
                if 'config' in rec:
                    config_updates.append(rec['config'])
                if 'action' in rec:
                    actions.append(rec['action'])

        # Update configuration
        if config_updates:
            with open(f"{self.gerrit_site}/etc/gerrit.config", 'a') as f:
                f.write('\n# Performance optimizations\n')
                for config in config_updates:
                    f.write(f"{config}\n")

        # Execute actions
        for action in actions:
            print(f"Executing: {action}")
            # Be careful with automated actions

        print("Optimizations applied. Restart Gerrit to take effect.")

    def benchmark_performance(self, duration=300):
        """Run performance benchmark"""
        print(f"Running performance benchmark for {duration} seconds...")

        start_time = time.time()
        end_time = start_time + duration

        results = {
            'requests': 0,
            'errors': 0,
            'response_times': []
        }

        while time.time() < end_time:
            try:
                start_req = time.time()
                response = requests.get(f"
{self.gerrit_url}/config/server/version")
                req_time = time.time() - start_req

                results['requests'] += 1
                results['response_times'].append(req_time)

                if response.status_code != 200:
                    results['errors'] += 1
```

```
            except:
                results['errors'] += 1

            time.sleep(1)  # 1 request per second

        # Calculate statistics
        if results['response_times']:
            results['avg_response_time'] = sum(results['response_times']) /
len(results['response_times'])
            results['max_response_time'] = max(results['response_times'])
            results['min_response_time'] = min(results['response_times'])

        return results

def main():
    optimizer = GerritPerformanceOptimizer(
        gerrit_site="/opt/gerrit/review_site",
        gerrit_url="http://localhost:8080"
    )

    # Analyze current performance
    metrics = optimizer.analyze_current_performance()
    print("Current Performance Metrics:")
    print(json.dumps(metrics, indent=2))

    # Generate recommendations
    recommendations = optimizer.generate_optimization_recommendations()
    print("\nOptimization Recommendations:")
    for rec in recommendations:
        print(f"[{rec['priority']}] {rec['category']}: {rec['issue']}")
        print(f"  Recommendation: {rec['recommendation']}")
        if 'config' in rec:
            print(f"  Config: {rec['config']}")
        print()

    # Run benchmark
    print("Running performance benchmark...")
    benchmark_results = optimizer.benchmark_performance(60)  # 1 minute test
    print("Benchmark Results:")
    print(json.dumps(benchmark_results, indent=2))

if __name__ == "__main__":
    main()
```

Exercise Completion and Certification

Final Project: Complete Gerrit Deployment

Objective: Deploy a complete enterprise-ready Gerrit system

Requirements:

- High availability setup
- Security hardening
- CI/CD integration
- Monitoring and alerting
- Documentation

Deliverables:

1. Complete deployment documentation
2. Configuration files
3. Monitoring dashboards
4. Security audit report
5. Performance benchmark results

Evaluation Criteria:

- Functionality (40%)
- Security (25%)
- Performance (20%)
- Documentation (15%)

Knowledge Assessment

Quiz Questions:

1. **What is the purpose of Change-Id in Gerrit?**
    - A) To identify the author
    - B) To track related patch sets
    - C) To set review priority
    - D) To enable notifications
2. **Which command pushes code for review?**
    - A) `git push origin master`
    - B) `git push origin HEAD:refs/for/master`
    - C) `git push --review origin master`
    - D) `git review push master`
3. **What does a -2 Code-Review score mean?**
    - A) Minor issues found
    - B) Major issues, needs work
    - C) This shall not be submitted
    - D) Looks good to me
4. **In HA setup, what is the purpose of HAProxy?**
    - A) Database replication

- B) Load balancing
- C) SSL termination
- D) All of the above

5. **Which file stores Gerrit's main configuration?**

   - A) `etc/config.ini`
   - B) `etc/gerrit.config`
   - C) `conf/gerrit.conf`
   - D) `config/settings.cfg`

Practical Assessment:

Create a script that:

1. Sets up a new Gerrit project
2. Configures appropriate access controls
3. Creates a sample change
4. Demonstrates the review workflow
5. Shows integration with external CI

Exercise Solutions

Solution for Exercise 1.1

```
#!/bin/bash
# solution-1-1.sh

# Install Java
sudo apt update
sudo apt install openjdk-11-jdk -y

# Download Gerrit
wget https://gerrit-releases.storage.googleapis.com/gerrit-3.8.4.war

# Initialize Gerrit
java -jar gerrit-3.8.4.war init -d ~/gerrit_testsite

# Modify configuration for port 8081
sed -i 's/listenUrl = http:\/\/\*:8080\//listenUrl = http:\/\/*:8081\//'
~/gerrit_testsite/etc/gerrit.config

# Start Gerrit
~/gerrit_testsite/bin/gerrit.sh start

# Verify installation
sleep 10
if curl -s http://localhost:8081 >/dev/null; then
    echo "✅ Gerrit successfully installed and running on port 8081"
else
```

```
    echo "❌ Installation failed"
    exit 1
fi
```

Solution for Exercise 2.1

```
#!/bin/bash
# solution-2-1.sh - Branch Management Solution

# Create project.config for branch policies
cat > /tmp/project.config << 'EOF'
[project]
    description = Test project with branch policies

[access "refs/heads/master"]
    label-Code-Review = -2..+2 group Administrators
    label-Code-Review = -1..+1 group Developers
    submit = group Submitters
    push = deny group Developers
    pushMerge = group Administrators

[access "refs/heads/develop"]
    label-Code-Review = -2..+2 group Developers
    submit = group Developers
    push = group Developers
    pushMerge = group Developers

[access "refs/heads/release/*"]
    label-Code-Review = -2..+2 group Release-Managers
    submit = group Release-Managers
    push = group Release-Managers
    create = group Release-Managers

[access "refs/for/refs/heads/*"]
    push = group Registered Users

[submit-requirement "Code-Review"]
    description = Requires Code-Review approval
    submittableIf = label:Code-Review=MAX AND -label:Code-Review=MIN
    canOverrideInChildProjects = false
EOF

echo "☑ Branch policies configured"
echo "Master branch: Restricted to administrators"
echo "Develop branch: Open to developers"
echo "Release branches: Release managers only"
```

This comprehensive exercise chapter provides hands-on experience with all aspects of Gerrit, from basic setup to enterprise deployment. Each exercise builds practical skills while reinforcing theoretical

knowledge from previous chapters.

Chapter Summary

You've completed comprehensive practical exercises covering:

- **Basic Setup and Configuration** - Foundation skills for Gerrit deployment
- **Review Workflow Mastery** - Hands-on experience with code review process
- **Advanced Feature Implementation** - Complex configurations and integrations
- **Enterprise Deployment** - High availability and security hardening
- **Performance Optimization** - Systematic approach to system tuning
- **Real-world Scenarios** - Practical problem-solving exercises

These exercises prepare you for real-world Gerrit administration and provide certification-level competency.

Congratulations! You've completed the comprehensive Gerrit tutorial. You now have the knowledge and practical experience to deploy, configure, and maintain enterprise-grade Gerrit installations.

This concludes the Complete Gerrit Tutorial. Continue practicing with your own projects!