教程平台软件配置与运维文档

文档概述

本文档详细描述了教程平台项目的配置管理、版本控制、持续集成、部署和运维计划，为项目的稳定运行和团队协作提供指导。

1. 配置管理

1.1 环境配置

1.1.1 多环境配置

后端配置

```
├── application.properties              # 基础配置
├── application-dev.properties          # 开发环境配置
├── application-test.properties         # 测试环境配置
└── application-prod.properties         # 生产环境配置
```

前端配置

```
├── .env                               # 默认环境变量
├── .env.development                   # 开发环境配置
├── .env.test                         # 测试环境配置
├── .env.production                   # 生产环境配置
└── config/
    ├── dev.config.js                 # 开发环境配置
    ├── test.config.js                # 测试环境配置
    └── prod.config.js                # 生产环境配置
```

1.1.2 配置项分类

后端数据库配置

```
# 开发环境
spring.datasource.url=jdbc:mysql://localhost:3306/tutorial_platform_dev
spring.datasource.username=dev_user
spring.datasource.password=${DB_PASSWORD:dev123}

# 生产环境
spring.datasource.url=jdbc:mysql://prod-db:3306/tutorial_platform
spring.datasource.username=${DB_USER:prod_user}
spring.datasource.password=${DB_PASSWORD}
```

后端应用配置

```
# 服务端口配置
server.port=${PORT:8088}

# 文件上传配置
file.upload-dir=${UPLOAD_DIR:upload_data}
spring.servlet.multipart.max-file-size=${MAX_FILE_SIZE:10MB}

# JWT配置
jwt.secret=${JWT_SECRET:defaultSecret}
jwt.expiration=${JWT_EXPIRATION:86400}
```

前端环境配置

```
// .env.development
VITE_API_BASE_URL=http://localhost:8088/api
VITE_APP_TITLE=Tutorial Platform Dev
VITE_UPLOAD_MAX_SIZE=10485760
VITE_WEBSOCKET_URL=ws://localhost:8088/ws

// .env.production
VITE_API_BASE_URL=https://api.tutorial-platform.com/api
VITE_APP_TITLE=Tutorial Platform
VITE_UPLOAD_MAX_SIZE=10485760
VITE_WEBSOCKET_URL=wss://api.tutorial-platform.com/ws
VITE_CDN_BASE_URL=https://cdn.tutorial-platform.com
```

1.2 配置管理工具

1.2.1 后端配置处理

Spring Boot Configuration Processor

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

1.2.2 前端配置处理

Vite配置管理

```
// vite.config.js
import { defineConfig, loadEnv } from 'vite'

export default defineConfig(({ command, mode }) => {
  const env = loadEnv(mode, process.cwd(), '')

  return {
    define: {
      __APP_ENV__: env.APP_ENV,
    },
    server: {
      port: 3000,
      proxy: {
        '/api': {
          target: env.VITE_API_BASE_URL,
          changeOrigin: true,
          rewrite: (path) => path.replace(/^\/api/, '')
        }
      }
    }
  }
})
```

1.2.3 配置验证

```
@ConfigurationProperties(prefix = "app")
@Validated
public class AppProperties {
    @NotBlank
    private String name;

    @Min(1)
    @Max(65535)
    private int port;
}
```

1.2.3 配置验证

后端配置验证

```java
@ConfigurationProperties(prefix = "app")
@Validated
public class AppProperties {
    @NotBlank
    private String name;

    @Min(1)
    @Max(65535)
    private int port;

    @NotBlank
    private String jwtSecret;
}
```

前端配置验证

```javascript
// config/validation.js
export const validateConfig = () => {
  const requiredEnvVars = [
    'VITE_API_BASE_URL',
    'VITE_APP_TITLE'
  ];

  const missingVars = requiredEnvVars.filter(
    varName => !import.meta.env[varName]
  );

  if (missingVars.length > 0) {
    throw new Error(`Missing required environment variables: ${missingVars.join(', ')}`);
  }
};
```

1.3 敏感信息管理

1.3.1 环境变量方式

后端环境变量

```bash
export DB_PASSWORD=your_secure_password
export JWT_SECRET=your_jwt_secret_key
export REDIS_PASSWORD=your_redis_password
```

前端环境变量

```bash
# 开发环境
export VITE_API_BASE_URL=http://localhost:8088/api
export VITE_WEBSOCKET_URL=ws://localhost:8088/ws

# 生产环境
export VITE_API_BASE_URL=https://api.tutorial-platform.com/api
export VITE_WEBSOCKET_URL=wss://api.tutorial-platform.com/ws
export VITE_CDN_BASE_URL=https://cdn.tutorial-platform.com
```

1.3.2 配置文件加密

- **后端**: 使用Jasypt进行配置文件敏感信息加密
- **前端**: 构建时环境变量注入，避免敏感信息暴露
- 生产环境密钥通过环境变量注入
- API密钥等敏感信息服务端管理

2. 版本控制

2.1 Git分支策略

2.1.1 GitFlow工作流

```
master/main     # 生产分支，只包含稳定版本
├── develop     # 开发分支，功能集成分支
├── feature/*   # 功能分支，从develop分出
├── release/*   # 发布分支，从develop分出
└── hotfix/*    # 热修复分支，从master分出
```

2.1.2 分支命名规范

```
feature/user-authentication      # 用户认证功能
feature/file-upload             # 文件上传功能
bugfix/login-error             # 登录错误修复
hotfix/security-patch          # 安全补丁
release/v1.0.0                # 版本发布
```

2.2 版本号管理

2.2.1 语义化版本控制

```
版本格式：主版本号.次版本号.修订号 (MAJOR.MINOR.PATCH)
示例：
- 1.0.0  # 初始稳定版本
- 1.1.0  # 新增功能，向后兼容
- 1.1.1  # 问题修复，向后兼容
- 2.0.0  # 重大变更，可能不向后兼容
```

2.2.2 Maven版本管理

```
<version>1.0.0-SNAPSHOT</version>  <!-- 开发版本 -->
<version>1.0.0</version>           <!-- 发布版本 -->
```

2.3 代码审查

2.3.1 Pull Request流程

1. 创建功能分支
2. 完成开发和测试
3. 提交Pull Request
4. 代码审查（至少一人）
5. 通过CI/CD检查
6. 合并到目标分支

2.3.2 代码审查检查点

- 代码规范性
- 功能正确性
- 性能影响
- 安全考虑
- 测试覆盖率

3. 持续集成

3.1 CI/CD流水线设计

3.1.1 GitHub Actions配置示例

```yaml
name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  # 后端测试
  backend-test:
    runs-on: ubuntu-latest

    services:
      mysql:
        image: mysql:8.0
        env:
          MYSQL_ROOT_PASSWORD: test123
          MYSQL_DATABASE: tutorial_platform_test
        options: >-
          --health-cmd="mysqladmin ping"
          --health-interval=10s
          --health-timeout=5s
          --health-retries=3

    steps:
    - uses: actions/checkout@v3

    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'

    - name: Cache Maven packages
      uses: actions/cache@v3
      with:
        path: ~/.m2
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}

    - name: Run backend tests
      run: mvn clean test

    - name: Generate test report
      run: mvn jacoco:report

    - name: Upload coverage to Codecov
      uses: codecov/codecov-action@v3

  # 前端测试
  frontend-test:
    runs-on: ubuntu-latest
```

```yaml

    steps:
    - uses: actions/checkout@v3

    - name: Set up Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18'
        cache: 'npm'
        cache-dependency-path: frontend/package-lock.json

    - name: Install frontend dependencies
      run: |
        cd frontend
        npm ci

    - name: Run frontend linting
      run: |
        cd frontend
        npm run lint

    - name: Run frontend tests
      run: |
        cd frontend
        npm run test:coverage

    - name: Upload frontend coverage
      uses: codecov/codecov-action@v3
      with:
        directory: frontend/coverage

  # 构建部署
  build-deploy:
    needs: [backend-test, frontend-test]
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'

    - name: Set up Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18'
        cache: 'npm'
        cache-dependency-path: frontend/package-lock.json

    - name: Build backend
      run: mvn clean package -DskipTests

    - name: Build frontend
      run: |
        cd frontend
        npm ci
        npm run build

    - name: Build Docker images
      run: |
        # 构建后端镜像
        docker build -t tutorial-platform-backend:${{ github.sha }} .

        # 构建前端镜像
        docker build -f frontend/Dockerfile -t tutorial-platform-frontend:${{ github.sha }} frontend/

        # 标记最新版本
        docker tag tutorial-platform-backend:${{ github.sha }} tutorial-platform-backend:latest
        docker tag tutorial-platform-frontend:${{ github.sha }} tutorial-platform-frontend:latest

    - name: Push to registry
      if: github.ref == 'refs/heads/main'
      run: |
        echo ${{ secrets.DOCKER_PASSWORD }} | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin

        # 推送后端镜像
        docker push tutorial-platform-backend:${{ github.sha }}
        docker push tutorial-platform-backend:latest

        # 推送前端镜像
        docker push tutorial-platform-frontend:${{ github.sha }}
        docker push tutorial-platform-frontend:latest
```

3.2 自动化测试策略

3.2.1 测试层次

```
后端测试层次:
├── 单元测试 (Unit Tests)          # 70%
├── 集成测试 (Integration Tests) # 20%
└── 端到端测试 (E2E Tests)        # 10%

前端测试层次:
├── 单元测试 (Unit Tests)          # 60% - 组件和工具函数
├── 集成测试 (Integration Tests) # 30% - 页面和用户流程
└── 端到端测试 (E2E Tests)        # 10% - 完整用户场景
```

3.2.2 测试配置

后端测试配置

```
<!-- Maven Surefire Plugin -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
        <includes>
            <include>**/*Test.java</include>
            <include>**/*Tests.java</include>
        </includes>
    </configuration>
</plugin>

<!-- JaCoCo Coverage Plugin -->
```

```xml
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

前端测试配置

```javascript
// vitest.config.js
import { defineConfig } from 'vitest/config'
import { resolve } from 'path'

export default defineConfig({
  test: {
    globals: true,
    environment: 'jsdom',
    setupFiles: ['./src/test/setup.js'],
    coverage: {
      reporter: ['text', 'json', 'html'],
      exclude: [
        'node_modules/',
        'src/test/',
        '**/*.d.ts',
      ]
    }
  },
  resolve: {
    alias: {
      '@': resolve(__dirname, './src')
    }
  }
})

// package.json scripts
{
  "scripts": {
    "test": "vitest",
    "test:coverage": "vitest --coverage",
    "test:ui": "vitest --ui",
    "test:e2e": "playwright test"
  }
}
```

E2E测试配置（Playwright）

```
// playwright.config.js
import { defineConfig } from '@playwright/test';

export default defineConfig({
  testDir: './e2e',
  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: 'html',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
    screenshot: 'only-on-failure',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    },
  ],
});
```

3.3 质量门禁

3.3.1 代码质量检查

- **静态代码分析**: SonarQube集成
- **代码覆盖率**: 最低80%覆盖率要求
- **安全扫描**: 依赖漏洞检查
- **性能测试**: 关键接口性能基准测试
- **前端质量检查**:
    - ESLint静态代码分析
    - Prettier代码格式化
    - TypeScript类型检查
    - Bundle分析和性能优化
    - 无障碍性检查（a11y）

3.3.2 质量标准

```
后端质量门禁:
  - code_coverage: ">= 80%"
  - duplicated_lines: "< 3%"
  - maintainability_rating: "A"
  - security_rating: "A"
  - reliability_rating: "A"

前端质量门禁:
  - code_coverage: ">= 75%"
  - bundle_size: "< 2MB"
  - lighthouse_performance: ">= 90"
  - lighthouse_accessibility: ">= 95"
  - lighthouse_best_practices: ">= 90"
  - lighthouse_seo: ">= 90"
```

4. 部署策略

4.1 容器化部署

4.1.1 Dockerfile

后端Dockerfile

```dockerfile
FROM openjdk:17-jre-slim

WORKDIR /app

COPY target/tutorial_platform-*.jar app.jar

EXPOSE 8088

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:8088/actuator/health || exit 1

ENTRYPOINT ["java", "-jar", "app.jar"]
```

前端Dockerfile

```dockerfile
# 多阶段构建
FROM node:18-alpine as build

WORKDIR /app

# 复制package文件
COPY package*.json ./
RUN npm ci --only=production

# 复制源代码并构建
COPY . .
RUN npm run build

# 生产阶段
FROM nginx:alpine

```

```
# 复制构建结果
COPY --from=build /app/dist /usr/share/nginx/html

# 复制nginx配置
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Nginx配置（nginx.conf）

```
events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    gzip on;
    gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript;

    server {
        listen 80;
        server_name localhost;

        location / {
            root /usr/share/nginx/html;
            try_files $uri $uri/ /index.html;
        }

        location /api {
            proxy_pass http://backend:8088;
            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;
        }

        location /ws {
            proxy_pass http://backend:8088;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
        }
    }
}
```

4.1.2 Docker Compose

```yaml
version: '3.8'

services:
  # 前端服务
  frontend:
    image: tutorial-platform-frontend:latest
    ports:
      - "80:80"
    environment:
      - NGINX_HOST=localhost
    depends_on:
      - backend
    restart: unless-stopped

  # 后端服务
  backend:
    image: tutorial-platform-backend:latest
    ports:
      - "8088:8088"
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - DB_HOST=mysql
      - DB_PASSWORD=${DB_PASSWORD}
      - JWT_SECRET=${JWT_SECRET}
    depends_on:
      - mysql
      - redis
    restart: unless-stopped

  # 数据库服务
  mysql:
    image: mysql:8.0
    environment:
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - MYSQL_DATABASE=tutorial_platform
    volumes:
      - mysql_data:/var/lib/mysql
    restart: unless-stopped

  # Redis服务
  redis:
    image: redis:7-alpine
    command: redis-server --requirepass ${REDIS_PASSWORD}
    restart: unless-stopped

  # Nginx反向代理（可选，用于负载均衡）
  nginx:
    image: nginx:alpine
    ports:
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/ssl:/etc/nginx/ssl
    depends_on:
      - frontend
      - backend
    restart: unless-stopped

volumes:
  mysql_data:
```

4.2 Kubernetes部署

4.2.1 部署配置

```yaml
# 后端部署配置
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tutorial-platform-backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: tutorial-platform-backend
  template:
    metadata:
      labels:
        app: tutorial-platform-backend
    spec:
      containers:
      - name: backend
        image: tutorial-platform-backend:latest
        ports:
        - containerPort: 8088
        env:
        - name: SPRING_PROFILES_ACTIVE
          value: "prod"
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: password
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /actuator/health
            port: 8088
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /actuator/health/readiness
            port: 8088
          initialDelaySeconds: 5
          periodSeconds: 5

---
# 前端部署配置
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tutorial-platform-frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tutorial-platform-frontend
  template:
    metadata:
      labels:
        app: tutorial-platform-frontend
    spec:
      containers:
      - name: frontend
        image: tutorial-platform-frontend:latest
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "50m"
          limits:
            memory: "128Mi"
            cpu: "100m"
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 10
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 10

---
# 服务配置
apiVersion: v1
kind: Service
metadata:
  name: tutorial-platform-backend-service
spec:
  selector:
    app: tutorial-platform-backend
  ports:
  - protocol: TCP
    port: 8088
    targetPort: 8088
  type: ClusterIP

---
apiVersion: v1
kind: Service
metadata:
  name: tutorial-platform-frontend-service
spec:
  selector:
    app: tutorial-platform-frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP

---
# Ingress配置
apiVersion: networking.k8s.io/v1
```

```yaml
kind: Ingress
metadata:
  name: tutorial-platform-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - tutorial-platform.com
    secretName: tutorial-platform-tls
  rules:
  - host: tutorial-platform.com
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: tutorial-platform-backend-service
            port:
              number: 8088
      - path: /
        pathType: Prefix
        backend:
          service:
            name: tutorial-platform-frontend-service
            port:
              number: 80
```

4.3 部署环境

4.3.1 环境划分

```
开发环境 (Development)
├── 后端：单实例，内存数据库
├── 前端：开发服务器 (Vite Dev Server)
├── 配置：热重载，调试模式
└── 更新：自动部署develop分支
```

```

测试环境 (Testing)
├── 后端：单实例，独立数据库
├── 前端：构建版本
├── 配置：模拟生产环境
└── 更新：手动部署feature分支

预生产环境 (Staging)
├── 后端：生产环境配置
├── 前端：生产构建版本
├── 配置：生产数据库镜像
└── 更新：手动部署release分支

生产环境 (Production)
├── 后端：高可用，负载均衡
├── 前端：CDN分发，缓存优化
├── 配置：监控告警，备份策略
└── 更新：手动部署main分支

CDN和静态资源
├── 静态文件：CSS、JS、图片
├── 用户上传文件：头像、简历等
├── 缓存策略：版本控制，过期时间
└── 全球分发：多地域节点
```

5. 运维计划

5.1 监控体系

5.1.1 应用监控

```
后端监控指标：
  - 应用健康状态：/actuator/health
  - 系统指标：CPU、内存、磁盘使用率
  - 业务指标：请求量、响应时间、错误率
  - JVM指标：堆内存、GC情况、线程数

前端监控指标：
  - 页面性能：FCP、LCP、FID、CLS
  - 用户体验: 页面加载时间、交互响应
  - 错误监控: JavaScript错误、网络错误
  - 用户行为: 页面访问、功能使用统计
  - 资源加载: 静态资源加载成功率
```

5.1.2 监控工具栈

```
后端监控工具:
Prometheus  # 指标收集
├── Grafana    # 指标可视化
├── AlertManager # 告警管理
└── Spring Boot Actuator # 应用指标暴露

前端监控工具:
Web Analytics # 用户行为分析
├── Google Analytics / Umami # 访问统计
├── Sentry # 错误监控
├── Lighthouse CI # 性能监控
└── Real User Monitoring (RUM) # 真实用户监控

统一监控平台:
├── ELK Stack # 日志聚合分析
├── Jaeger # 分布式链路追踪
└── Datadog / New Relic # 全栈监控
```

5.1.3 Grafana仪表板配置

```
{
  "dashboard": {
    "title": "Tutorial Platform Monitoring",
    "panels": [
      {
        "title": "Backend Request Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(http_requests_total[5m])"
          }
        ]
      },
      {
        "title": "Backend Response Time",
        "type": "graph",
        "targets": [
          {
            "expr": "histogram_quantile(0.95,
rate(http_request_duration_seconds_bucket[5m]))"
          }
        ]
      },
      {
        "title": "Frontend Page Load Time",
        "type": "graph",
        "targets": [
          {
            "expr": "avg(frontend_page_load_duration_seconds)"
          }
        ]
      },
      {
        "title": "Frontend Error Rate",
        "type": "graph",
        "targets": [
          {
            "expr": "rate(frontend_errors_total[5m])"
          }
        ]
      },
      {
        "title": "User Sessions",
        "type": "stat",
        "targets": [
          {
            "expr": "frontend_active_sessions"
          }
        ]
      }
    ]
  }
}
```

前端性能监控集成

```javascript
// 性能监控SDK集成
import { getCLS, getFID, getFCP, getLCP, getTTFB } from 'web-vitals';

// Web Vitals监控
getCLS(console.log);
getFID(console.log);
getFCP(console.log);
getLCP(console.log);
getTTFB(console.log);

// 自定义性能指标上报
function reportMetric(metric) {
  fetch('/api/metrics', {
    method: 'POST',
    body: JSON.stringify(metric),
    headers: { 'Content-Type': 'application/json' }
  });
}
```

5.2 日志管理

5.2.1 日志配置

```xml
<!-- logback-spring.xml -->
<configuration>
    <springProfile name="!prod">
        <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
            <encoder>
                <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
            </encoder>
        </appender>
        <root level="INFO">
            <appender-ref ref="CONSOLE"/>
        </root>
    </springProfile>

    <springProfile name="prod">
        <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
            <file>logs/application.log</file>
            <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
                <fileNamePattern>logs/application-%d{yyyy-MM-dd}.%i.gz</fileNamePattern>
                <maxFileSize>100MB</maxFileSize>
                <maxHistory>30</maxHistory>
                <totalSizeCap>3GB</totalSizeCap>
            </rollingPolicy>
            <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
                <providers>
                    <timestamp/>
                    <logLevel/>
                    <loggerName/>
                    <message/>
                    <mdc/>
                    <stackTrace/>
                </providers>
            </encoder>
        </appender>
        <root level="INFO">
            <appender-ref ref="FILE"/>
        </root>
    </springProfile>
</configuration>
```

5.2.2 日志收集

```
# ELK Stack集成
Filebeat     # 日志收集
├── Logstash    # 日志处理
├── Elasticsearch # 日志存储和搜索
└── Kibana      # 日志可视化
```

5.3 备份策略

5.3.1 数据库备份

```bash
#!/bin/bash
# 数据库备份脚本

DB_NAME="tutorial_platform"
BACKUP_DIR="/backups/mysql"
DATE=$(date +%Y%m%d_%H%M%S)

# 创建备份
mysqldump -u root -p${DB_PASSWORD} \
  --single-transaction \
  --routines \
  --triggers \
  ${DB_NAME} > ${BACKUP_DIR}/${DB_NAME}_${DATE}.sql

# 压缩备份文件
gzip ${BACKUP_DIR}/${DB_NAME}_${DATE}.sql

# 删除7天前的备份
find ${BACKUP_DIR} -name "*.sql.gz" -mtime +7 -delete

echo "Database backup completed: ${DB_NAME}_${DATE}.sql.gz"
```

5.3.2 文件备份

```bash
#!/bin/bash
# 应用文件备份脚本

APP_DIR="/app/upload_data"
BACKUP_DIR="/backups/files"
DATE=$(date +%Y%m%d)

# 创建文件备份
tar -czf ${BACKUP_DIR}/files_${DATE}.tar.gz -C ${APP_DIR} .

# 保留30天的备份
find ${BACKUP_DIR} -name "files_*.tar.gz" -mtime +30 -delete
```

5.4 容灾恢复

5.4.1 恢复流程

```
1. 故障检测
   ├── 监控告警触发
   ├── 故障确认和评估
   └── 启动应急响应

2. 快速恢复
   ├── 切换到备用实例
   ├── 数据库故障转移
   └── 服务健康检查

3. 完全恢复
   ├── 数据恢复验证
   ├── 服务功能测试
   └── 恢复正常运行
```

5.4.2 RTO/RPO目标

```
RTO (恢复时间目标): 30分钟
RPO (恢复点目标): 15分钟
可用性目标: 99.9%
```

5.5 安全运维

5.5.1 安全监控

```
安全监控点:
  - 异常登录检测
  - API调用频率监控
  - 文件上传安全检查
  - 数据库访问审计
  - 系统漏洞扫描
```

5.5.2 安全更新流程

1. 定期安全扫描
2. 漏洞评估和优先级排序
3. 测试环境验证
4. 生产环境热修复
5. 安全事件记录和分析

6. 运维自动化

6.1 自动化脚本

6.1.1 部署脚本

```bash
#!/bin/bash
# 全栈自动化部署脚本

ENV=${1:-dev}
VERSION=${2:-latest}

echo "Deploying tutorial-platform version ${VERSION} to ${ENV} environment..."

# 部署后端
echo "Deploying backend..."
docker pull tutorial-platform-backend:${VERSION}

# 部署前端
echo "Deploying frontend..."
docker pull tutorial-platform-frontend:${VERSION}

# 停止旧服务
docker-compose -f docker-compose.${ENV}.yml down

# 启动新服务
docker-compose -f docker-compose.${ENV}.yml up -d

# 后端健康检查
```

```
echo "Checking backend health..."
sleep 30
for i in {1..10}; do
  if curl -f http://localhost:8088/actuator/health; then
    echo "Backend is healthy"
    break
  fi
  echo "Waiting for backend to be ready... ($i/10)"
  sleep 10
done

# 前端健康检查
echo "Checking frontend health..."
for i in {1..5}; do
  if curl -f http://localhost:80; then
    echo "Frontend is healthy"
    break
  fi
  echo "Waiting for frontend to be ready... ($i/5)"
  sleep 5
done

# 清理旧镜像
docker image prune -f

echo "Deployment completed successfully!"
```

前端单独部署脚本

```
#!/bin/bash
# 前端部署脚本

ENV=${1:-production}
BUILD_DIR="dist"

echo "Building frontend for ${ENV} environment..."

# 安装依赖
npm ci

# 构建项目
```

```bash
npm run build:${ENV}

# 部署到CDN或静态服务器
if [ "$ENV" = "production" ]; then
  # 上传到CDN
  aws s3 sync ${BUILD_DIR} s3://tutorial-platform-cdn/ --delete

  # 清除CDN缓存
  aws cloudfront create-invalidation --distribution-id XXXXX --paths "/*"

  echo "Frontend deployed to CDN successfully!"
else
  # 部署到测试服务器
  scp -r ${BUILD_DIR}/* user@test-server:/var/www/html/
  echo "Frontend deployed to test server successfully!"
fi
```

6.1.2 监控脚本

```bash
#!/bin/bash
# 系统监控脚本

# 检查后端服务状态
BACKEND_STATUS=$(curl -s -o /dev/null -w "%{http_code}" http://localhost:8088/actuator/health)

if [ $BACKEND_STATUS -ne 200 ]; then
    echo "Backend service health check failed. Status: $BACKEND_STATUS"
    # 发送告警
    curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
         -H 'Content-type: application/json' \
         --data '{"text":"Tutorial Platform backend service is down!"}'
fi

# 检查前端服务状态
FRONTEND_STATUS=$(curl -s -o /dev/null -w "%{http_code}" http://localhost:80)
```

```

if [ $FRONTEND_STATUS -ne 200 ]; then
    echo "Frontend service health check failed. Status: $FRONTEND_STATUS"
    # 发送告警
    curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
         -H 'Content-type: application/json' \
         --data '{"text":"Tutorial Platform frontend service is down!"}'
fi

# 检查磁盘使用率
DISK_USAGE=$(df -h / | awk 'NR==2 {print $5}' | sed 's/%//')
if [ $DISK_USAGE -gt 80 ]; then
    echo "Disk usage is high: ${DISK_USAGE}%"
    # 发送告警
    curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
         -H 'Content-type: application/json' \
         --data "{\"text\":\"Disk usage is high: ${DISK_USAGE}%\"}"
fi

# 检查内存使用率
MEMORY_USAGE=$(free | grep Mem | awk '{printf("%.0f", $3/$2 * 100.0)}')
if [ $MEMORY_USAGE -gt 85 ]; then
    echo "Memory usage is high: ${MEMORY_USAGE}%"
    # 发送告警
    curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
         -H 'Content-type: application/json' \
         --data "{\"text\":\"Memory usage is high: ${MEMORY_USAGE}%\"}"
fi

# 检查Docker容器状态
UNHEALTHY_CONTAINERS=$(docker ps --filter "health=unhealthy" --format "table {{.Names}}" | tail -n +2)
if [ ! -z "$UNHEALTHY_CONTAINERS" ]; then
    echo "Unhealthy containers found: $UNHEALTHY_CONTAINERS"
    # 发送告警
    curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
         -H 'Content-type: application/json' \
         --data "{\"text\":\"Unhealthy containers: $UNHEALTHY_CONTAINERS\"}"
fi

# 前端性能检查（可选）
check_frontend_performance() {
    # 使用lighthouse进行性能检查
    if command -v lighthouse &> /dev/null; then
        PERFORMANCE_SCORE=$(lighthouse http://localhost:80 --only-categories=performance --output=json --quiet | jq '.categories.performance.score * 100')
        if [ $(echo "$PERFORMANCE_SCORE < 80" | bc -l) -eq 1 ]; then
            echo "Frontend performance score is low: ${PERFORMANCE_SCORE}"
            # 发送告警
            curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
                 -H 'Content-type: application/json' \
                 --data "{\"text\":\"Frontend performance score is low: ${PERFORMANCE_SCORE}\"}"
        fi
    fi
}

# 执行性能检查（每小时执行一次）
if [ $(date +%M) -eq 0 ]; then
    check_frontend_performance
fi
```

6.2 定时任务

6.2.1 Crontab配置

```
# 后端相关定时任务
# 每日数据库备份
0 2 * * * /scripts/backup_database.sh

# 每小时日志轮转检查
0 * * * * /scripts/rotate_logs.sh

# 每5分钟系统监控
*/5 * * * * /scripts/system_monitor.sh

# 每周系统更新检查
0 3 * * 0 /scripts/system_update_check.sh

# 前端相关定时任务
# 每日CDN缓存预热
0 4 * * * /scripts/cdn_cache_warmup.sh

# 每小时前端性能检查
0 * * * * /scripts/frontend_performance_check.sh

# 每日静态资源清理
0 1 * * * /scripts/static_resource_cleanup.sh

# 每周前端安全扫描
0 5 * * 0 /scripts/frontend_security_scan.sh

# 数据分析相关定时任务
# 每日用户行为数据分析
0 6 * * * /scripts/user_analytics_daily.sh

# 每月生成业务报告
0 8 1 * * /scripts/monthly_business_report.sh
```

前端性能检查脚本

```
#!/bin/bash

```

```bash
# 前端性能检查脚本

SITE_URL="https://tutorial-platform.com"
THRESHOLD_PERFORMANCE=80
THRESHOLD_ACCESSIBILITY=90

# 执行Lighthouse检查
lighthouse_result=$(lighthouse $SITE_URL \
  --only-categories=performance,accessibility,best-practices \
  --output=json \
  --quiet \
  --chrome-flags="--headless --no-sandbox")

# 解析性能分数
performance_score=$(echo $lighthouse_result | jq '.categories.performance.score * 100')
accessibility_score=$(echo $lighthouse_result | jq '.categories.accessibility.score * 100')
best_practices_score=$(echo $lighthouse_result | jq '.categories["best-practices"].score * 100')

# 检查性能阈值
if [ $(echo "$performance_score < $THRESHOLD_PERFORMANCE" | bc -l) -eq 1 ]; then
    echo "Performance score below threshold: $performance_score"
    # 发送告警
    curl -X POST "$SLACK_WEBHOOK" \
         -H 'Content-type: application/json' \
         --data "{\"text\":\"Frontend performance alert: Score is $performance_score (threshold: $THRESHOLD_PERFORMANCE)\"}"
fi

# 检查可访问性阈值
if [ $(echo "$accessibility_score < $THRESHOLD_ACCESSIBILITY" | bc -l) -eq 1 ]; then
    echo "Accessibility score below threshold: $accessibility_score"
    # 发送告警
    curl -X POST "$SLACK_WEBHOOK" \
         -H 'Content-type: application/json' \
        --data "{\"text\":\"Frontend accessibility alert: Score is $accessibility_score (threshold: $THRESHOLD_ACCESSIBILITY)\"}"
fi

echo "Performance check completed. Scores: Performance=$performance_score, Accessibility=$accessibility_score, Best Practices=$best_practices_score"
```

7. 总结

本软件配置与运维文档建立了完整的DevOps体系，涵盖前后端全栈应用的运维需求：

7.1 配置管理成果

1. **多环境配置策略**: 后端Spring Boot多环境配置 + 前端环境变量管理
2. **敏感信息保护**: 后端Jasypt加密 + 前端构建时注入
3. **配置验证机制**: 后端Bean Validation + 前端配置校验

7.2 版本控制与协作

1. **GitFlow工作流**: 标准化的分支管理策略
2. **语义化版本管理**: 前后端统一的版本发布规范
3. **代码审查流程**: 保证代码质量和团队协作

7.3 持续集成成果

1. **自动化测试**: 后端Maven测试 + 前端Vitest/Playwright测试
2. **质量门禁**: 后端80%覆盖率 + 前端75%覆盖率 + Lighthouse性能检查
3. **并行构建**: 前后端独立构建和部署流程

7.4 部署策略优势

1. **容器化部署**: Docker多阶段构建优化
2. **Kubernetes集群**: 高可用和弹性伸缩
3. **多环境支持**: 开发/测试/预生产/生产环境隔离

4. **CDN集成**: 前端静态资源全球分发

7.5 运维监控体系

1. **全栈监控**: 后端Prometheus + 前端Web Vitals监控
2. **日志管理**: ELK Stack统一日志收集和分析
3. **备份策略**: 数据库定期备份 + 静态资源备份
4. **容灾恢复**: RTO 30分钟，RPO 15分钟，99.9%可用性目标

7.6 安全运维保障

1. **多层防护**: 前端XSS/CSRF防护 + 后端JWT认证
2. **安全监控**: 异常检测、漏洞扫描、访问审计
3. **定期更新**: 依赖包安全更新和漏洞修复

7.7 自动化运维

1. **部署自动化**: 一键部署脚本和CI/CD流水线
2. **监控自动化**: 定时健康检查和性能监控
3. **告警机制**: Slack集成的实时告警通知
4. **性能优化**: 前端包体积优化和CDN缓存策略

7.8 技术创新点

1. **前后端分离**: 独立开发、测试、部署
2. **微服务架构**: 支持服务拆分和独立扩展
3. **现代化技术栈**: Spring Boot 3.x + 现代前端框架
4. **云原生**: Kubernetes + Docker容器化

这些措施确保了教程平台系统的高可用性、可维护性和安全性，为项目的长期稳定运行和快速迭代提供了坚实的技术保障。通过完整的DevOps实践，实现了从代码提交到生产部署的全流程自动化，大大提高了开发效率和运维质量。