PowerAutomation 部署链接和资源指南

🚀 主要部署链接

GitHub 仓库

主仓库: https://github.com/alexchuang650730/aicore0615

这是PowerAutomation的官方GitHub仓库，包含完整的源代码、文档和部署配置。

快速部署链接

方法1: 直接克隆部署

```
# 克隆仓库
git clone https://github.com/alexchuang650730/aicore0615.git
cd aicore0615

# 快速启动
cd test
python main.py
```

方法2: 特定组件部署

```
# 克隆仓库
git clone https://github.com/alexchuang650730/aicore0615.git
cd aicore0615

# 启动SmartUI MCP
cd mcp/adapter/smartui_mcp
python cli.py start

# 启动测试框架
cd ../../test
python cli.py status
```

📋 部署资源清单

核心组件链接

- **SmartUI MCP**: https://github.com/alexchuang650730/aicore0615/tree/main/mcp/adapter/smartui_mcp
- **Enhanced Workflow MCP**: https://github.com/alexchuang650730/aicore0615/tree/main/mcp/adapter/enhanced_workflow_mcp
- **测试框架**: https://github.com/alexchuang650730/aicore0615/tree/main/test
- **MCP协调器**: https://github.com/alexchuang650730/aicore0615/tree/main/mcp

配置文件链接

- **测试配置**: https://github.com/alexchuang650730/aicore0615/tree/main/test/config
- **MCP配置**: https://github.com/alexchuang650730/aicore0615/tree/main/mcp/adapter/*/config.toml

文档链接

- **团队指南**: https://github.com/alexchuang650730/aicore0615/tree/main/mcphowto
- **API文档**: https://github.com/alexchuang650730/aicore0615/tree/main/docs
- **部署指南**: https://github.com/alexchuang650730/aicore0615/blob/main/README.md

🛠️ 环境要求

系统要求

- **操作系统**: Ubuntu 22.04+ / macOS 12+ / Windows 10+
- **Python**: 3.11.0+
- **内存**: 最低4GB，推荐8GB+
- **磁盘**: 最低10GB可用空间
- **网络**: 稳定的互联网连接

依赖安装

```
# 核心依赖
pip install asyncio pyyaml pathlib json uuid datetime
pip install fastapi uvicorn flask requests
pip install pandas numpy matplotlib seaborn
pip install beautifulsoup4 markdown reportlab
```

🚀 快速部署步骤

步骤1: 获取代码

```
# 方法1: HTTPS克隆
git clone https://github.com/alexchuang650730/aicore0615.git

# 方法2: SSH克隆（需要配置SSH密钥）
git clone git@github.com:alexchuang650730/aicore0615.git

# 方法3: 下载ZIP包
wget https://github.com/alexchuang650730/aicore0615/archive/
refs/heads/main.zip
unzip main.zip
```

步骤2: 环境配置

```
cd aicore0615

# 检查Python版本
python3 --version

# 安装依赖
pip3 install -r requirements.txt

# 验证安装
python3 -c "import asyncio, yaml, pandas; print('依赖安装成功')"
```

步骤3: 启动系统

```
# 启动测试框架
cd test
python main.py

# 启动SmartUI MCP
cd ../mcp/adapter/smartui_mcp
python cli.py start

# 验证系统状态
python cli.py status
```

🐳 Docker部署

Docker镜像

```
# 构建Docker镜像
docker build -t powerautomation:latest .

# 运行容器
docker run -d -p 8000:8000 --name powerautomation
powerautomation:latest

# 检查容器状态
docker ps
docker logs powerautomation
```

Docker Compose部署

```
# docker-compose.yml
version: '3.8'
services:
  powerautomation:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/
powerautomation
    depends_on:
      - db
      - redis

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: powerautomation
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data

volumes:
```

```
  postgres_data:
  redis_data:
```

☸️ Kubernetes部署

Helm Chart部署

```
# 添加Helm仓库
helm repo add powerautomation https://charts.powerautomation.io
helm repo update

# 安装PowerAutomation
helm install my-powerautomation powerautomation/powerautomation
\
  --set global.storageClass=fast-ssd \
  --set coordinator.replicas=3 \
  --set smartui.enabled=true
```

直接Kubernetes部署

```
# 应用Kubernetes配置
kubectl apply -f k8s/namespace.yaml
kubectl apply -f k8s/configmap.yaml
kubectl apply -f k8s/deployment.yaml
kubectl apply -f k8s/service.yaml
kubectl apply -f k8s/ingress.yaml

# 检查部署状态
kubectl get pods -n powerautomation
kubectl get services -n powerautomation
```

🌐 云平台部署

AWS部署

```
# 使用AWS EKS
eksctl create cluster --name powerautomation-cluster --region
us-west-2

# 部署到EKS
kubectl apply -f aws/eks-deployment.yaml
```

```
# 配置负载均衡器
kubectl apply -f aws/alb-ingress.yaml
```

Azure部署

```
# 使用Azure AKS
az aks create --resource-group myResourceGroup --name
powerautomation-aks

# 获取凭据
az aks get-credentials --resource-group myResourceGroup --name
powerautomation-aks

# 部署应用
kubectl apply -f azure/aks-deployment.yaml
```

Google Cloud部署

```
# 使用GKE
gcloud container clusters create powerautomation-cluster --zone
us-central1-a

# 部署应用
kubectl apply -f gcp/gke-deployment.yaml
```

🔧 配置管理

环境变量配置

```
# 设置环境变量
export POWERAUTOMATION_ENV=production
export DATABASE_URL=postgresql://user:pass@localhost:5432/
powerautomation
export REDIS_URL=redis://localhost:6379/0
export SECRET_KEY=your-secret-key-here
```

配置文件模板

```
# config/production.yaml
database:
  url: ${DATABASE_URL}
  pool_size: 20
```

```
  max_overflow: 30

cache:
  redis_url: ${REDIS_URL}
  default_timeout: 3600

security:
  secret_key: ${SECRET_KEY}
  jwt_expiry: 3600

logging:
  level: INFO
  format: json
```

📊 监控和健康检查

健康检查端点

```
# 检查系统健康状态
curl http://localhost:8000/health

# 检查各组件状态
curl http://localhost:8000/api/v1/status

# 检查指标
curl http://localhost:8000/metrics
```

监控配置

```
# monitoring/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'powerautomation'
    static_configs:
      - targets: ['localhost:8000']
    metrics_path: /metrics
    scrape_interval: 5s
```

🔐 安全配置

SSL/TLS配置

```
# nginx.conf
server {
    listen 443 ssl;
    server_name powerautomation.yourdomain.com;

    ssl_certificate /path/to/certificate.crt;
    ssl_certificate_key /path/to/private.key;

    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

防火墙配置

```
# UFW防火墙配置
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw allow 8000/tcp
sudo ufw enable
```

📝 部署验证

功能验证脚本

```
#!/bin/bash
echo "PowerAutomation 部署验证"
echo "========================"

# 检查服务状态
curl -f http://localhost:8000/health || echo "健康检查失败"

# 检查API响应
curl -f http://localhost:8000/api/v1/status || echo "API检查失败"

# 检查SmartUI MCP
cd mcp/adapter/smartui_mcp
```

```
python cli.py status || echo "SmartUI MCP检查失败"

# 检查测试框架
cd ../../../test
python cli.py status || echo "测试框架检查失败"

echo "部署验证完成"
```

🆘 故障排除

常见问题解决

```
# 端口占用问题
sudo lsof -i :8000
sudo kill -9 <PID>

# 权限问题
sudo chown -R $USER:$USER /opt/powerautomation
chmod +x scripts/*.sh

# 依赖问题
pip3 install --upgrade pip
pip3 install -r requirements.txt --force-reinstall
```

日志查看

```
# 查看应用日志
tail -f logs/powerautomation.log

# 查看系统日志
sudo journalctl -u powerautomation -f

# 查看Docker日志
docker logs -f powerautomation
```

📞 支持和帮助

官方资源

- **GitHub Issues**: https://github.com/alexchuang650730/aicore0615/issues
- **文档**: https://github.com/alexchuang650730/aicore0615/tree/main/docs
- **Wiki**: https://github.com/alexchuang650730/aicore0615/wiki

社区支持

- **讨论区**: https://github.com/alexchuang650730/aicore0615/discussions
- **Stack Overflow**: 标签 `powerautomation`
- **Reddit**: r/PowerAutomation

商业支持

- **技术支持**: support@powerautomation.io
- **企业服务**: enterprise@powerautomation.io
- **培训服务**: training@powerautomation.io

最后更新: 2025年6月17日
版本: 1.0.0
维护者: PowerAutomation团队

🎯 详细部署指导

生产环境部署最佳实践

PowerAutomation的生产环境部署需要考虑高可用性、安全性、性能和可维护性等多个方面。以下是经过验证的生产环境部署最佳实践。

架构规划

生产环境建议采用分层架构设计，包括负载均衡层、应用服务层、数据存储层和监控管理层。负载均衡层使用Nginx或HAProxy实现请求分发和SSL终止。应用服务层部署多个PowerAutomation实例，确保高可用性。数据存储层包括PostgreSQL主从集群和Redis集群。监控管理层集成Prometheus、Grafana和ELK Stack。

```yaml
# 生产环境架构配置
production_architecture:
  load_balancer:
    type: nginx
    instances: 2
    ssl_termination: true
    health_check: enabled

  application_tier:
    powerautomation_instances: 3
    resource_allocation:
      cpu: "4 cores"
      memory: "8GB"
      storage: "100GB SSD"
```

```
  data_tier:
    postgresql:
      primary: 1
      replicas: 2
      backup_strategy: "continuous"
    redis:
      cluster_nodes: 3
      replication: "master-slave"

  monitoring:
    prometheus: enabled
    grafana: enabled
    elasticsearch: enabled
    log_retention: "90 days"
```

安全加固

生产环境的安全配置至关重要，需要从网络、应用、数据等多个层面进行加固。网络层面配置防火墙规则，只开放必要的端口。应用层面启用HTTPS、配置身份认证、实施权限控制。数据层面进行加密存储、定期备份、访问审计。

```
 # 安全加固脚本
#!/bin/bash

# 配置防火墙
sudo ufw --force reset
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp   # SSH
sudo ufw allow 80/tcp   # HTTP
sudo ufw allow 443/tcp  # HTTPS
sudo ufw --force enable

# 配置SSL证书
sudo certbot --nginx -d powerautomation.yourdomain.com

# 设置文件权限
sudo chown -R powerautomation:powerautomation /opt/
powerautomation
sudo chmod 750 /opt/powerautomation
sudo chmod 640 /opt/powerautomation/config/*.yaml

# 配置日志轮转
sudo tee /etc/logrotate.d/powerautomation << EOF
/opt/powerautomation/logs/*.log {
    daily
    missingok
    rotate 30
    compress
```

```
    delaycompress
    notifempty
    create 644 powerautomation powerautomation
}
EOF
```

性能优化

生产环境的性能优化包括应用配置优化、数据库调优、缓存策略优化等方面。应用配置需要根据实际负载调整连接池大小、工作进程数量、超时设置等参数。数据库调优包括索引优化、查询优化、连接池配置等。缓存策略需要合理设置缓存层次、过期时间、淘汰策略等。

```yaml
# 性能优化配置
performance_config:
  application:
    worker_processes: 4
    worker_connections: 1024
    keepalive_timeout: 65
    client_max_body_size: "50M"

  database:
    max_connections: 200
    shared_buffers: "2GB"
    effective_cache_size: "6GB"
    work_mem: "64MB"
    maintenance_work_mem: "512MB"

  cache:
    redis_maxmemory: "4GB"
    redis_maxmemory_policy: "allkeys-lru"
    cache_default_timeout: 3600

  monitoring:
    metrics_retention: "15d"
    log_level: "INFO"
    slow_query_threshold: "1s"
```

开发环境快速搭建

开发环境的搭建注重快速部署和便于调试，可以使用Docker Compose实现一键部署。开发环境包含所有必要的服务组件，但配置相对简化，资源要求较低。

Docker Compose开发环境

```yaml
# docker-compose.dev.yml
version: '3.8'
```

```yaml
services:
  powerautomation:
    build:
      context: .
      dockerfile: Dockerfile.dev
    ports:
      - "8000:8000"
      - "5678:5678"  # 调试端口
    volumes:
      - .:/app
      - /app/node_modules
    environment:
      - FLASK_ENV=development
      - DEBUG=True
      - DATABASE_URL=postgresql://dev:dev@db:5432/
powerautomation_dev
      - REDIS_URL=redis://redis:6379/0
    depends_on:
      - db
      - redis
    command: python -m debugpy --listen 0.0.0.0:5678 --wait-for-
client app.py

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: powerautomation_dev
      POSTGRES_USER: dev
      POSTGRES_PASSWORD: dev
    ports:
      - "5432:5432"
    volumes:
      - postgres_dev_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_dev_data:/data

  adminer:
    image: adminer
    ports:
      - "8080:8080"
    depends_on:
      - db

volumes:
  postgres_dev_data:
  redis_dev_data:
```

开发环境启动脚本

```
#!/bin/bash
# dev-setup.sh

echo "PowerAutomation 开发环境搭建"
echo "============================"

# 检查Docker和Docker Compose
if ! command -v docker &> /dev/null; then
    echo "错误: Docker未安装"
    exit 1
fi

if ! command -v docker-compose &> /dev/null; then
    echo "错误: Docker Compose未安装"
    exit 1
fi

# 创建开发环境配置
cp config/development.yaml.example config/development.yaml

# 构建和启动服务
echo "构建Docker镜像..."
docker-compose -f docker-compose.dev.yml build

echo "启动开发环境..."
docker-compose -f docker-compose.dev.yml up -d

# 等待服务启动
echo "等待服务启动..."
sleep 30

# 运行数据库迁移
echo "运行数据库迁移..."
docker-compose -f docker-compose.dev.yml exec powerautomation
python manage.py migrate

# 创建测试数据
echo "创建测试数据..."
docker-compose -f docker-compose.dev.yml exec powerautomation
python manage.py seed_data

# 显示服务状态
echo "服务状态:"
docker-compose -f docker-compose.dev.yml ps

echo ""
echo "开发环境搭建完成!"
echo "PowerAutomation: http://localhost:8000"
echo "数据库管理: http://localhost:8080"
```

```
echo "调试端口: 5678"
echo ""
echo "停止环境: docker-compose -f docker-compose.dev.yml down"
echo "查看日志: docker-compose -f docker-compose.dev.yml logs -f"
```

测试环境部署

测试环境用于集成测试、性能测试和用户验收测试，需要尽可能接近生产环境的配置，但可以适当简化以降低成本。

测试环境配置

```yaml
# test-environment.yml
apiVersion: v1
kind: Namespace
metadata:
  name: powerautomation-test

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: powerautomation-test
  namespace: powerautomation-test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: powerautomation-test
  template:
    metadata:
      labels:
        app: powerautomation-test
    spec:
      containers:
      - name: powerautomation
        image: powerautomation:test
        ports:
        - containerPort: 8000
        env:
        - name: ENVIRONMENT
          value: "test"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: test-db-secret
              key: url
        resources:
          requests:
```

```
            cpu: "1"
            memory: "2Gi"
          limits:
            cpu: "2"
            memory: "4Gi"

---
apiVersion: v1
kind: Service
metadata:
  name: powerautomation-test-service
  namespace: powerautomation-test
spec:
  selector:
    app: powerautomation-test
  ports:
  - port: 80
    targetPort: 8000
  type: LoadBalancer
```

自动化测试集成

```
#!/bin/bash
# test-deployment.sh

echo "PowerAutomation 测试环境部署"
echo "=========================="

# 部署测试环境
kubectl apply -f test-environment.yml

# 等待部署完成
kubectl wait --for=condition=available --timeout=300s
deployment/powerautomation-test -n powerautomation-test

# 获取服务地址
TEST_URL=$(kubectl get service powerautomation-test-service -n
powerautomation-test -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')

echo "测试环境部署完成"
echo "测试地址: http://$TEST_URL"

# 运行健康检查
echo "运行健康检查..."
curl -f http://$TEST_URL/health || echo "健康检查失败"

# 运行自动化测试
echo "运行自动化测试..."
cd test
```

```
python -m pytest tests/ --url=http://$TEST_URL --junit-xml=test-
results.xml

echo "测试完成"
```

CI/CD集成

PowerAutomation支持与主流CI/CD平台集成，实现自动化构建、测试和部署。

GitHub Actions配置

```yaml
# .github/workflows/ci-cd.yml
name: PowerAutomation CI/CD

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

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: test_db
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

      redis:
        image: redis:7
        options: >-
          --health-cmd "redis-cli ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
```

```yaml
      with:
        python-version: '3.11'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install -r requirements-test.txt

    - name: Run tests
      env:
        DATABASE_URL: postgresql://postgres:postgres@localhost:
5432/test_db
        REDIS_URL: redis://localhost:6379/0
      run: |
        python -m pytest tests/ --cov=powerautomation --cov-
report=xml

    - name: Upload coverage
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage.xml

  build:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2

    - name: Login to Docker Hub
      uses: docker/login-action@v2
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Build and push
      uses: docker/build-push-action@v4
      with:
        context: .
        push: true
        tags: powerautomation/powerautomation:latest
        cache-from: type=gha
        cache-to: type=gha,mode=max

  deploy:
    needs: build
    runs-on: ubuntu-latest
```

```
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v3

    - name: Deploy to staging
      run: |
        # 部署到测试环境
        kubectl apply -f k8s/staging/
        kubectl rollout status deployment/powerautomation-
staging

    - name: Run integration tests
      run: |
        # 运行集成测试
        python -m pytest tests/integration/ --url=https://
staging.powerautomation.io

    - name: Deploy to production
      if: success()
      run: |
        # 部署到生产环境
        kubectl apply -f k8s/production/
        kubectl rollout status deployment/powerautomation-
production
```

Jenkins Pipeline配置

```
// Jenkinsfile
pipeline {
    agent any

    environment {
        DOCKER_REGISTRY = 'your-registry.com'
        IMAGE_NAME = 'powerautomation'
        KUBECONFIG = credentials('kubeconfig')
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Test') {
            steps {
                script {
                    docker.image('python:3.11').inside {
                        sh '''
```

```
                            pip install -r requirements.txt
                            python -m pytest tests/ --junit-
xml=test-results.xml
                        '''
                    }
                }
            }
            post {
                always {
                    junit 'test-results.xml'
                }
            }
        }

        stage('Build') {
            when {
                branch 'main'
            }
            steps {
                script {
                    def image = docker.build("$
{DOCKER_REGISTRY}/${IMAGE_NAME}:${BUILD_NUMBER}")
                    docker.withRegistry("https://$
{DOCKER_REGISTRY}", 'docker-registry-credentials') {
                        image.push()
                        image.push('latest')
                    }
                }
            }
        }

        stage('Deploy to Staging') {
            when {
                branch 'main'
            }
            steps {
                sh '''
                    helm upgrade --install powerautomation-
staging ./helm/powerautomation \
                        --namespace staging \
                        --set image.tag=${BUILD_NUMBER} \
                        --set environment=staging
                '''
            }
        }

        stage('Integration Tests') {
            when {
                branch 'main'
            }
            steps {
                sh '''
```

```
                    python -m pytest tests/integration/ \
                        --url=https://staging.powerautomation.io
\
                        --junit-xml=integration-test-results.xml
                '''
            }
            post {
                always {
                    junit 'integration-test-results.xml'
                }
            }
        }

        stage('Deploy to Production') {
            when {
                allOf {
                    branch 'main'
                    expression { currentBuild.result == null ||
currentBuild.result == 'SUCCESS' }
                }
            }
            steps {
                input message: 'Deploy to production?', ok:
'Deploy'
                sh '''
                    helm upgrade --install powerautomation-
production ./helm/powerautomation \
                        --namespace production \
                        --set image.tag=${BUILD_NUMBER} \
                        --set environment=production
                '''
            }
        }
    }

    post {
        always {
            cleanWs()
        }
        failure {
            emailext (
                subject: "Build Failed: ${env.JOB_NAME} - $
{env.BUILD_NUMBER}",
                body: "Build failed. Check console output at $
{env.BUILD_URL}",
                to: "${env.CHANGE_AUTHOR_EMAIL}"
            )
        }
    }
}
```