

배포 프로세스 및 특이사항

블루/그린 배포 전략 설명 및 전환 방법

1. 블루/그린 배포 전략 개요

블루/그린 배포는 무중단 배포 방식의 하나로, 두 개의 동일한 프로덕션 환경(블루와 그린)을 유지하며 트래픽을 한 환경에서 다른 환경으로 전환하는 방식입니다. 이를 통해 다운타임 없이 새로운 버전을 배포할 수 있으며, 문제 발생 시 빠르게 이전 환경으로 롤백할 수 있습니다.

2. 시스템 구성

- 프론트엔드 서비스: Next.js (블루: 3001 포트, 그린: 3002 포트)
- 백엔드 서비스: Spring Boot (블루: 8081 포트, 그린: 8082 포트)
- 라우팅 서비스: NGINX (443 포트, HTTP/HTTPS)
- 데이터베이스: MySQL (3307 포트)
- 캐시 서버: Redis (6379 포트)

3. 배포 흐름

1. 배포 대상 결정: 현재 활성화된 환경 확인 (블루 또는 그린)
2. 비활성 환경 배포: 현재 사용되지 않는 환경에 새 버전을 배포
3. 헬스 체크: 신규 배포된 환경의 정상 동작 확인
4. 트래픽 전환: NGINX 설정 변경을 통한 트래픽 전환
5. 검증 및 정리: 배포 검증 및 불필요한 리소스 정리

4. 전환 메커니즘

4.1 NGINX 설정 관리

NGINX `bluegreen.conf` 파일을 통해 트래픽 라우팅을 제어합니다:

```
# 블루/그린 프론트엔드 서비스 (Next.js)
upstream bluegreen_client {
    server localhost:3001; # 블루 서버
    # server localhost:3002; # 그린 서버 (주석 처리)
```

```

}

# 블루/그린 백엔드 서비스 (Spring)
upstream bluegreen_server {
    server localhost:8081; # 블루 서버
    # server localhost:8082; # 그린 서버 (주석 처리)
}

```

4.2 전환 프로세스

1. Jenkins 파이프라인에서 현재 활성 환경 확인:

```

def getCurrentActiveServer() {
    def nginxConf = sh(script: "cat ${NGINX_BLUEGREEN_CONF}", returnStdout: true).trim()
    if (nginxConf.contains("server localhost:8081") && !nginxConf.contains("# server localhost:8081")) {
        return "blue"
    } else if (nginxConf.contains("server localhost:8082") && !nginxConf.contains("# server localhost:8082")) {
        return "green"
    }
    return "blue" // 기본값
}

```

2. NGINX 설정 파일 수정:

```

if (env.TARGET_SERVER == "blue") {
    // Blue 서버로 트래픽 전환
    sh """
    sed -i -e 's/# server localhost:3001;/server localhost:3001;/g' \\\
    -e 's/server localhost:3002;/# server localhost:3002;/g' \\\
    -e 's/# server localhost:8081;/server localhost:8081;/g' \\\
    -e 's/server localhost:8082;/# server localhost:8082;/g' ${nginxTempFile}
    """
} else {
    // Green 서버로 트래픽 전환
}

```

```

sh """
sed -i -e 's/server localhost:3001;/# server localhost:3001;/g' \
-e 's/# server localhost:3002;/server localhost:3002;/g' \
-e 's/server localhost:8081;/# server localhost:8081;/g' \
-e 's/# server localhost:8082;/server localhost:8082;/g' ${nginx
TempFile}
"""
}

```

3. NGINX 설정 적용 및 리로드:

```

sshagent(['ssh-ec2-ubuntu']) {
  sh """
    scp -o StrictHostKeyChecking=no ${nginxTempFile} ubuntu@j12b1
10.p.ssafy.io:/tmp/bluegreen.conf &&
    ssh -o StrictHostKeyChecking=no ubuntu@j12b110.p.ssafy.io '
      sudo cp /tmp/bluegreen.conf /etc/nginx/conf.d/bluegreen.conf
    &&
      sudo nginx -t &&
      sudo systemctl reload nginx &&
      rm -f /tmp/bluegreen.conf
    '
  """
}

```

5. 롤백 프로세스

문제 발생 시 이전 환경(블루 또는 그린)으로 즉시 롤백이 가능합니다. 이는 단순히 NGINX 설정 파일의 주석을 조정하고 NGINX를 리로드하는 과정만으로 이루어집니다. 이전 환경은 배포 후에도 계속 유지되므로 빠른 롤백이 가능합니다.

6. 모니터링 및 검증

각 배포 단계마다 서버 헬스 체크를 수행하여 정상 동작을 검증합니다:

```

def checkServerHealth(port) {
  // 내부 포트 확인
  sh(script: "docker exec server-${env.TARGET_SERVER} netstat -tulpn | g
rep 8080", returnStatus: true)
}

```

```
// 헬스체크 시도
def response = sh(script: "curl -s -m 5 http://j12b110.p.ssafy.io:${port}/api/test/health", returnStdout: true).trim()

// 응답 확인
if (response && (response.contains("UP") || response.contains("ok") || response.contains("OK"))) {
    return true
}

// 로그로 서버 상태 확인
return sh(script: "docker logs server-${env.TARGET_SERVER} | grep 'Started JjeonchongmuApplication'", returnStatus: true) == 0
}
```

블루/그린 배포 전략을 통해 사용자 경험을 유지하면서 안정적인 배포를 수행할 수 있습니다.

Docker 컨테이너 관리 방법

1. 컨테이너 상태 확인

- 현재 실행 중인 컨테이너 목록 확인:

```
docker ps
```

- 모든 컨테이너 확인(중지된 컨테이너 포함):

```
docker ps -a
```

- 블루/그린 컨테이너 필터링:

```
docker ps --filter "name=client-blue" --filter "name=server-blue"do
```

```
docker ps --filter "name=client-green" --filter "name=server-green"
```

2. 컨테이너 로그 확인

- 실시간 로그 확인:

```
docker logs -f [컨테이너명] # 예: docker logs -f server-blue
```

- 최근 로그 확인:

```
docker logs --tail 100 [컨테이너명]
```

3. docker compose를 통한 컨테이너 관리

- 블루/그린 환경 시작:

```
# 블루 환경 시작
docker compose -f docker-compose.blue.yml up -d
# 그린 환경 시작
docker compose -f docker-compose.green.yml up -d
```

- 환경 중지:

```
# 블루 환경 중지
docker compose -f docker-compose.blue.yml down
# 그린 환경 중지
docker compose -f docker-compose.green.yml down
```

- 컨테이너 재시작:

```
docker compose -f docker-compose.blue.yml restart
```

- 환경 상태 확인:

```
docker compose -f docker-compose.blue.yml ps
```

4. 개별 컨테이너 관리

- 컨테이너 시작/중지/재시작:

```
docker start [컨테이너명]
docker stop [컨테이너명]
docker restart [컨테이너명]
```

5. 리소스 모니터링 및 정리

- 컨테이너 리소스 사용량 확인:

```
docker stats
```

- 미사용 리소스 정리:

```
# 중지된 컨테이너, 미사용 이미지, 네트워크, 볼륨 정리
docker system prune
# 볼륨 포함 정리
docker system prune --volumes
```

6. 컨테이너 내부 접속

- 컨테이너 내부 셸 접속:

```
docker exec -it [컨테이너명] /bin/sh
```

- 예시:

```
exec -it client-green /bin/sh
```

7. 도커 이미지 관리

- 이미지 목록 확인:

```
docker images
```

- 이미지 빌드:

```
# docker compose를 통한 빌드
docker compose -f docker-compose.blue.yml build
```

- 미사용 이미지 정리:

```
docker image prune
```

8. 볼륨 및 네트워크 관리

- 볼륨 목록 확인:

```
docker volume ls
```

- 네트워크 목록 확인:

```
docker network ls
```

배포 자동화 파이프라인 설명

1. CI/CD 구성 개요

- Jenkins를 활용한 지속적 통합 및 배포(CI/CD) 파이프라인을 구축하였습니다.
- GitLab의 develop 브랜치에 코드가 푸시되거나 머지 리퀘스트가 발생할 때 자동으로 빌드가 트리거됩니다.
- 블루/그린 배포 전략을 통해 서비스 중단 없이 신규 버전을 배포합니다.
- Jenkins 서버는 9090 포트를 통해 접근 가능합니다.

2. 파이프라인 주요 단계

- 코드 체크아웃:
 - GitLab 저장소에서 최신 코드를 가져옵니다.
 - 프로젝트가 이미 존재하는 경우 빠른 업데이트를 위해 git fetch와 reset만 수행하고, 없는 경우 새로 클론합니다.
 - 빠른 클론을 위해 `-depth=1` 옵션을 사용합니다.
- 환경 초기화:
 - NGINX 설정 파일(bluegreen.conf)과 Docker Compose 파일(blue/green)을 확인하고 필요시 생성합니다.

- Docker Compose 파일에는 서비스 포트 매핑 정보가 포함되어 있습니다(블루: 8081/3001, 그린: 8082/3002).
- **배포 대상 결정:**
 - NGINX 설정 파일을 분석하여 현재 활성화된 서버(블루 또는 그린)를 식별합니다.
 - 비활성 서버를 새로운 배포 대상으로 선택합니다.
 - 대상 서버의 포트 정보를 환경 변수로 설정합니다.
- **환경 변수 설정:**
 - Jenkins Credentials에 저장된 데이터베이스 접속 정보, API 키, JWT 시크릿 등을 환경 변수로 설정합니다.
 - Docker Compose 및 프론트엔드 애플리케이션용 .env 파일을 생성합니다.
 - 설정한 환경 변수를 기반으로 Docker 이미지를 빌드하고 컨테이너를 시작합니다.
- **컨테이너 배포:**
 - 대상 환경(블루 또는 그린)에 해당하는 Docker Compose 파일을 사용하여 서비스를 배포합니다.
 - 기존 컨테이너가 있다면 중지하고 새로운 버전으로 컨테이너를 시작합니다.
- **헬스 체크:**
 - 새로 배포된 서버의 정상 작동 여부를 확인합니다.
 - `/api/test/health` 엔드포인트를 호출하여 서버 응답을 확인합니다.
 - 응답이 없거나 실패할 경우 서버 로그를 확인하여 문제를 진단합니다.
 - 최대 10번의 시도를 통해 서버가 안정적으로 시작되었는지 확인합니다.
- **트래픽 전환:**
 - NGINX 설정 파일을 수정하여 트래픽을 새로 배포된 서버로 전환합니다.
 - 프론트엔드와 백엔드 모두 동시에 전환하여 일관성을 유지합니다.
 - SSH를 통해 EC2 인스턴스에 접속하여 NGINX 설정을 적용하고 서비스를 리로드합니다.
- **검증 및 정리:**

- 트래픽 전환 후 새로운 활성 서버를 확인하여 전환이 성공적으로 이루어졌는지 검증합니다.
- 불필요한 이미지와 볼륨을 정리하여 디스크 공간을 확보합니다.
- 이전 환경은 롤백을 위해 유지합니다.

3. 자동화 트리거

- GitLab 웹훅을 통해 develop 브랜치에 코드가 푸시되거나 머지 리퀘스트가 생성 될 때 자동으로 빌드가 시작됩니다.
- 트리거 설정:

```
triggers {
  gitlab(triggerOnPush: true, triggerOnMergeRequest: true, branchFilterType: 'NameBasedFilter', includeBranchesSpec: 'develop')
}
```

- 이를 통해 개발자가 변경 사항을 푸시할 때마다 자동으로 배포 프로세스가 시작됩니다.

4. 보안 관리


- Jenkins Credentials 플러그인을 사용하여 민감한 정보를 안전하게 관리합니다.
- 데이터베이스 비밀번호, API 키, JWT 시크릿 등 중요 정보는 코드에 직접 포함되지 않고 Jenkins 보안 저장소에서 관리됩니다.
- SSH 키를 통해 EC2 인스턴스에 안전하게 접속하여 NGINX 설정을 적용합니다.
- 보안 정보 사용 예시:

```
withCredentials([
  string(credentialsId: 'mysql-root-password', variable: 'MYSQL_ROOT_PASSWORD'),
  string(credentialsId: 'mysql-database', variable: 'MYSQL_DATABASE'),
  string(credentialsId: 'mysql-app-user', variable: 'MYSQL_USER'),
  string(credentialsId: 'mysql-app-password', variable: 'MYSQL_PASSWORD'),
  // 기타 보안 정보...
]) {
```

```
// 환경 변수 설정 및 컨테이너 배포
}
```

5. 알림 시스템

- Mattermost 채팅 플랫폼을 통해 배포 결과를 자동으로 통지합니다.
- 배포 성공 시 녹색으로 표시된 메시지와 함께 다음 정보를 포함합니다:
 - 작업명 및 빌드 번호
 - 브랜치 정보
 - 커밋 해시 및 메시지
 - 작성자 정보
 - 배포 완료 시간
 - 현재 활성화된 서버(블루 또는 그린)
- 배포 실패 시 빨간색으로 표시된 메시지와 함께 실패 원인 및 로그 확인 링크를 제공합니다.
- 알림 설정 예시:

```
mattermostSend(
  color: 'good',
  message: "" *무중단 배포 성공*
  • *작업명*: ${env.JOB_NAME}
  • *빌드*: #${env.BUILD_NUMBER}
  • *브랜치*: ${BRANCH_NAME}
  • *커밋*: ${Commit_Hash} - ${Commit_Msg}
  • *작성자*: ${Author_ID} <${Author_Email}>
  • *완료 시간*: ${kTime}
  • *활성 서버*: ${env.TARGET_SERVER}""",
  endpoint: 'https://meeting.ssafy.com/hooks/1m96q98oybfi9k67nkjipxjddy',
  channel: 'B110-Jenkins'
)
```

6. 장애 대응

- 배포 프로세스에 30분 타임아웃을 설정하여 무한 대기 상태를 방지합니다.

- 배포 중 오류 발생 시 즉시 Mattermost로 알림을 전송하고 실패 로그 링크를 제공합니다.
- 헬스 체크 실패 시 컨테이너 로그를 확인하여 문제의 원인을 진단합니다.
- 모든 빌드 후에는 작업 공간을 정리하여 디스크 공간을 확보합니다.

```
post {  
  failure {  
    // 실패 알림 전송  
  }  
  always {  
    // 작업 공간 정리  
    cleanWs()  
  }  
}
```