bootC + GitHub Actions Workflow + GitHub Self-hosted Runner

무엇이 헷갈렸는가?

Manus AI Agent

GitHub Actions Workflow란?

자동화 도구

코드 이벤트 → 자동 실행

수동 작업 → 자동화

- **Workflows:** YAML 자동화 정의
- **Jobs:** 작업 그룹
- **Steps:** 개별 명령어
- **Actions:** 재사용 코드
- **Runners:** 실행 환경

왜 GitHub Actions가 필요한가?

수동 작업의 문제점

반복적 빌드 → 테스트 → 배포

실수 발생 → 비효율성 증대

코드 푸시 → 자동 빌드/테스트/배포

생산성 극대화!

GitHub Actions 사용법

1. YAML 파일로 워크플로우 정의

2. 이벤트 발생 시 트리거

3. Job 실행 (순차/병렬)

4. Step 순서대로 실행

Self-hosted Runner란?

사용자 서버에서 GitHub Actions 실행

구분	GitHub-hosted	Self-hosted
관리	GitHub 완전 관리	직접 관리
성능	제한된 스펙	무제한 확장
비용	사용량 기반	인프라 비용만
커스터마이징	제한적	완전 자유
네트워크	제한된 접근	내부 네트워크 접근

왜 Self-hosted Runner?

GitHub-hosted 한계

- ❓메모리/CPU 부족
- ❓시간 제한 (6시간)
- ❓내부 네트워크 접근 불가
- ❓높은 비용

Self-hosted Runner

→ 무제한 리소스 + 내부 접근 + 비용 절감

Self-hosted Runner 사용법

1. GitHub에서 Runner 등록

↓

2. 서버에 Runner 설치

↓

3. 토큰으로 등록

↓

4. `./run.sh` 실행

↓

5. `runs-on: self-hosted` 지정

bootC + GitHub 조합

왜 함께 사용하나?

bootc 이미지 빌드 자동화

상당한 리소스 + 특수 권한 필요

GitHub-hosted: 14GB 디스크 공간 부족
bootc 빌드: --privileged 모드 필요
→ Self-hosted Runner 필수!

조합의 이점

엣지 디바이스 관리 시나리오

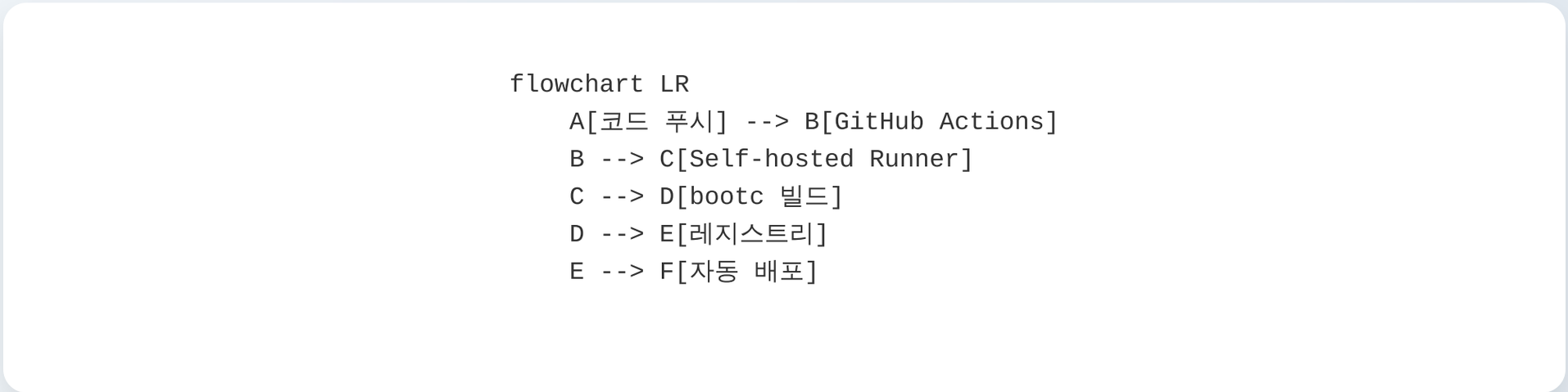
- ✓ 코드 푸시 → 자동 트리거
- ✓ Self-hosted에서 bootc 빌드
- ✓ 레지스트리에 자동 푸시
- ✓ GitOps로 자동 배포

코드 한 번 푸시 → 전체 시스템 업데이트

수백 대 디바이스 자동 관리!

구현 방법

- **준비:** Self-hosted Runner 서버 설정
- **워크플로우:** YAML 파일 작성
- **빌드:** bootc-image-builder 자동 실행
- **배포:** GitOps 연동



결론

자동화 + 확장성

bootc 수동 빌드 → 비효율적

GitHub Actions + Self-hosted Runner = 완전 자동화

생산성 향상
코드 푸시 → 전체 업데이트

비용 절감
GitHub 비용 + 수동 작업 감소

보안 강화
내부 네트워크 안전 빌드

확장성
수백 시스템 일관 관리