



GitHub Actions 및 클라우드 기반 CI로 보다 빠르게 빌드하고 오류 줄이기

Emilia Lazer-Walker, 수석 클라우드 담당자

She/Her

안건

- 빌드 시스템 및 CI/CD
- 중요한 이유
- 온프레미스와 클라우드 호스트형 CI 비교
- 특히 게임용 CI가 까다로운 이유
- 사례 연구: 모바일 Unity 게임 빌드
- 대규모 프로젝트에서 고려할 사항
- 시작 리소스 및 고급 경로

'빌드 시스템'이란?

기존 의미

- '사무실 한 구석에 있는 PC 타워' 혹은 작은 Jenkins 클러스터
- 원할 때 언제든지 수동으로 빌드 시작

새로운 의미

- 물리적 컴퓨터를 고려할 필요가 없음
- 자동으로 빌드가 이루어짐
- 빌드에서 코드에 오류가 발생했을 때 사전 예방적으로 사용자에게 알림

지속적인 통합 및 지속적인 배포

지속적인 통합(CI)

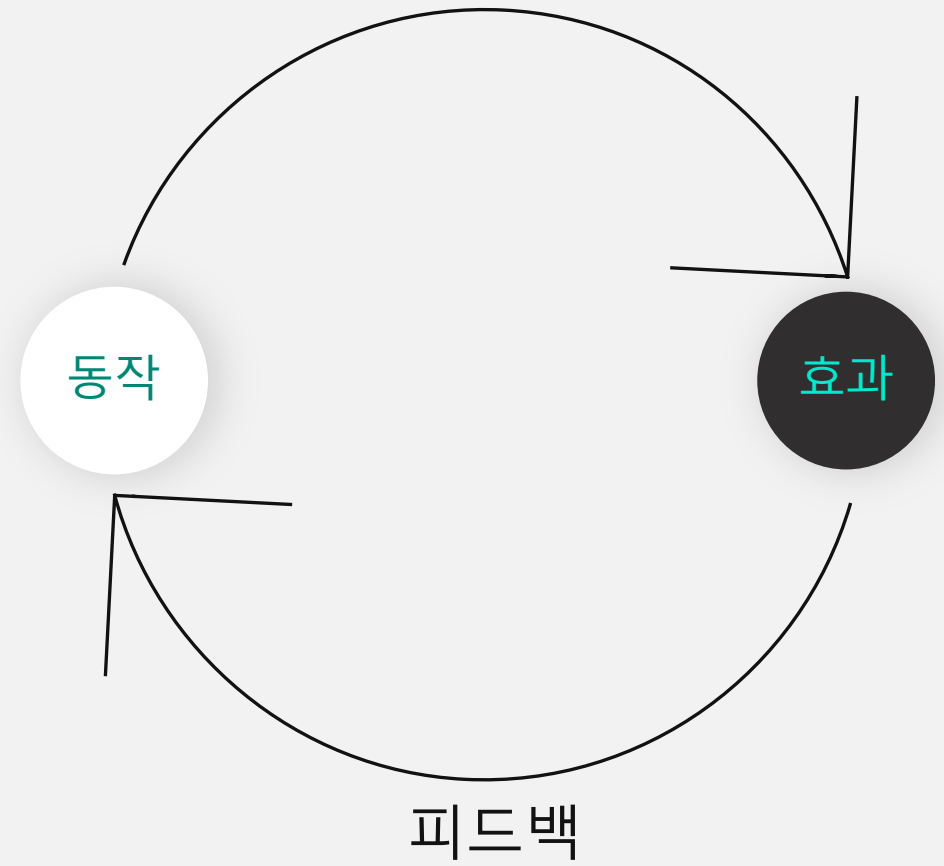
- 개발자는 중앙 분기에 코드를 최대한 자주 병합해야 함. 코드가 병합될 때 자동화된 테스트가 자동으로 실행됨
- **게임의 경우**, 전체 단위 또는 통합/수용 테스트보다는 기본 빌드인 경우가 많으며 콘텐츠 확인일 수도 있음

지속적인 배포(CD)

- 새 코드를 병합할 때 테스트 이외에 제작 빌드를 번들로 묶어 배포(일부 경우 '배포'의 정의)
- **게임 클라이언트의 경우**, 최대한 자주 빌드를 생성(QA 또는 베타 사용자용)하고, 가능한 범위까지 플랫폼 제출 프로세스 자동화



피드백 루프



중요한 이유

- 더 빠르게 오류를 찾고 개발 속도 향상
 - 엔지니어링에서 버그 찾기
 - 자산을 통합하는 아티스트
 - QA와의 공동 작업
 - 베타 테스터들의 피드백

중요한 이유

· 새로운 제작 방식 구현

- 빌드에 하루가 아닌 30분이 걸린다면, 이러한 이점이 근본적으로 어떻게 엔지니어/디자인/QA 커뮤니케이션을 변화시키겠습니까?
- 더 이상 자산을 엔진에 추가하는 데 통합 엔지니어가 필요하지 않다면, 이러한 이점이 비주얼 아티스트가 실험하는 데 어떻게 도움을 주겠습니까?

클라우드 호스팅의 스펙트럼

자체 호스트 메탈

- '사무실 구석의 빌드 PC'에서 서버 랙 또는 데이터 센터에 이르는 모든 곳
- 하드웨어 + 업그레이드/수리 비용 지불(자본 비용)
- 스케일링 = 새 하드웨어 구입
- 지속적인 유지 관리

자체 호스트 클라우드

- 클라우드 도구와 통합된 클라우드 공급자의 VM 프로비전
- 사용량에 따라 비용 청구(운영 비용)
- 자동으로 스케일링하도록 설정됨
- 지속적인 유지 관리. 단, 하드웨어 유지 관리 없음

클라우드

- CI 공급자에게 특정 테스트를 실행하도록 알리고 방법 모색
- 사용량에 따라 비용 청구(운영 비용)
- 크기에 대해 고려할 필요 없음

온프레미스 방식에 비해 클라우드 호스트 방식이 가진 이점

모든 사용자

- 자체 Mac 하드웨어를 유지 관리할 필요가 없는 iOS 빌드
- 보다 성숙한 OSS 도구와 개발자 친화적인 스크립팅 도구

대형 스튜디오

- 용량 관리 필요 없음
- 새로운 프로젝트를 더 쉽게 시작하고 종료할 수 있으며 팀을 위한 셀프 서비스 옵션 제공

소규모 스튜디오

- It Just Works™
- 사무실에 단일 빌드 박스를 보유하는 것보다 확장성이 높으며 서버에 대해 고민할 필요가 없음

게임별로 존재하는 문제

- 빌드 요구 사항이 까다로운 경우 더 강력한 머신 필요
- Windows용 빌드 머신이 필요한 경우가 많음
(예: 콘솔 SDK)
- CI는 다른 소프트웨어 생태시스템에 비해 주요 게임 엔진에 덜 성숙하거나 덜 개발된 CLI 도구를 사용

클라우드 CI로 게임별 문제를 해결하는 방법

까다로운 빌드 요구 사항

- 일반적으로 클라우드에서 업그레이드는 더 높은 계층의 VM에 비용을 지불하는 것만큼 단순함

빌드 머신은 Windows 기반이어야 함

- 클라우드 CI로 인해 여러 호스트 플랫폼(Win, Linux, Mac)을 위한 빌드 인프라를 유지 관리할 필요가 없음

CLI 게임 엔진 도구 부족

- GitHub Actions 및 OSS 프레임워크와 같은 플랫폼을 사용하면 완전히 다시 새로 만들 필요 없이 커뮤니티의 도움을 받을 수 있음

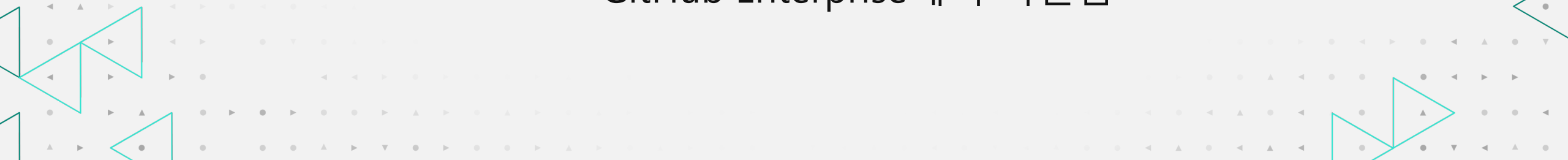


<https://github.com/lazerwalker/boatattack>

사례 연구: 모바일 Unity 게임

- Unity는 GitHub Actions 도구로 이루어진 강력한 에코시스템 보유
- iOS와 Android 코드 서명 필요
- SDK에 NDA 없음
- 클라우드 CI로 인해 물리적 Mac 하드웨어를 소유하고 유지 관리할 필요가 없음
- PC가 아닌 대부분의 플랫폼에 비해 모바일 하드웨어는 종류와 OS 버전이 다양하므로 CI를 통해 테스트 매트릭스를 제공할 수 있음

GitHub Actions

- GitHub에 내장된 CI/CD 시스템
 - 워크플로 작성을 위한 YAML 구문
 - 작성 가능: 워크플로에 커뮤니티에서 작성한 동작 포함
 - Windows, Linux, macOS용 실행기
 - 호스트형 실행기, 자체 호스트 옵션
 - OSS에 무료
 - GitHub Enterprise에서 지원됨
- 

Windows 빌드

1단계: 워크플로 작업 설정

`.github/workflows/build.yml`

name: Windows Build

on:

push:

branches: [main]

env:

UNITY_EMAIL: ...

UNITY_PASSWORD: ...

UNITY_SERIAL: ...

jobs:

windows:

name: Unity Windows build

runs-on: ubuntu-latest

steps:

- name: Checkout repository
uses: actions/checkout@v2
with:
lfs: true
- name: Build project
uses: game-ci/unity-builder@v2
with:
targetPlatform: windows
- uses: actions/upload-artifact@v2
with:
name: windows
path: build/windows

Windows 빌드

2단계: 리포지토리 및 빌드 확인
(캐싱 표시 안 됨)

GameCI 사용(<https://game.ci>)

- Unity Docker 이미지
- 사전 작성된 동작
- Unity 라이선스 관리
- 또한 Gitlab 지원 및 자체 호스트 실행기의 자동 스케일링

iOS용 빌드와 Windows용 빌드의 차이

-
- 코드 서명
 - Linux와 Mac VM 모두 사용
 - 플랫폼 제출

iOS 빌드

1단계: iOS 코드 서명

Fastlane 사용(<https://fastlane.tools>), 코드 서명 및 빌드를 관리하기 위한 OSS 라이브러리

1. App Store Connect 키 생성
2. Apple App Store 목록 생성
3. 코드 서명 인증서용 새 Git 리포지토리 생성
4. 코드 서명 인증서를 생성하고 git 리포지토리에 저장하는 GH 동작 실행
(이 GH 동작은 링크로 연결된 리포지토리에서 읽을 수 있음 – ios_setup.yml 및 generate_certs.yml)

이 워크플로는 물리적 Mac 하드웨어가 필요하지 **않으며** GH Actions Mac 실행기만 필요합니다.

iOS 빌드

2단계: 빌드 매트릭스를 사용하여
여러 플랫폼용으로 빌드하는 방법

jobs:

unity:

name: \${matrix.targetPlatform}}

runs-on: ubuntu-latest

strategy:

matrix:

targetPlatform:

- iOS

- Android

...

- name: Build project

uses: game-ci/unity-builder@v2

with:

targetPlatform:

\${{matrix.targetPlatform}}

apple:

name: Build for Xcode

needs: unity

runs-on: macos-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- uses: actions/dl-artifact@v2

with:

name: Build-iOS

- name: Build iOS

shell: bash

run: bundle exec fastlane build

iOS 빌드

3단계: iOS에서 생성된 Xcode
프로젝트 빌드

(표시되지 않는 사항: Ruby 설정,
여러 환경 변수 및 기타 설정,
Fastlane 스크립트, 생성된 IPA를
사용한 모든 작업)

iOS 빌드

4단계: TestFlight 또는 App Store에
업로드

자세한 내용은 전체 Fastfile 확인

푸시할 때마다 호출하면 속도가
제한되므로 매번 푸시하지 말 것

Fastfile

```
desc "Push a new build to TestFlight"
lane :testflight_beta do
  build
  upload_to_testflight
end
```

```
desc "Push a new App Store build"
lane :app_store do
  build

  upload_to_app_store(
    force: true,
    skip_metadata: true,
    skip_screenshots: true
  )
end
```

Android: iOS와 동일하지만 더 단순

- 코드 서명은 저장할 수 있는 단일 파일임
- 여전히 Unity를 사용하여 프로젝트를 생성하고 Fastlane을 사용하여 빌드하지만 macOS 없이 단일 Linux 실행기에서 진행할 수 있음

Unity 라이선싱: 개별 상황에 따라 다를 수 있음

- GameCI에 의해 Personal 및 Pro 지원. 워크플로는 달라질 수 있음
- 공식적으로 Personal/Pro Unity 라이선스당 한 사람만 사용할 수 있음
- 대규모 고객의 경우 Unity 담당자에게 문의



<https://github.com/lazerwalker/boatattack>

대규모 프로젝트에서 고려할 사항

- 현재 디스크 크기 및 GH Actions 호스트 실행기의 캐싱 제한
- 자체 호스트 실행기로 마이그레이션해야 할 수 있음
- 실행기 비용 최적화(Mac 대 Linux)
- Windows 기반 클라우드 Unity 실행기의 가용성

시작하기

-
- <https://docs.github.com/en/actions>
 - <https://game.ci>
 - <https://fastlane.tools>
 - <https://github.com/lazerwalker/BoatAttack>

CI 여정의 다음 단계

- Unity 이외의 엔진을 사용한 GH Actions 경로
- Azure DevOps 및 git 이외의 VCS를 사용하는 프로젝트를 위한 옵션
- 더 많은 테스트 추가
- 플랫폼 제출 워크플로에 CI를 사용하는 방법
- 창의성을 발휘하여 CI로 실현할 수 있는 사항 모색



감사합니다. 🙌

Emilia Lazer-Walker

<https://github.com/lazerwalker/BoatAttack>

@lazerwalker

