클라우드 기반 CI/CD 구축

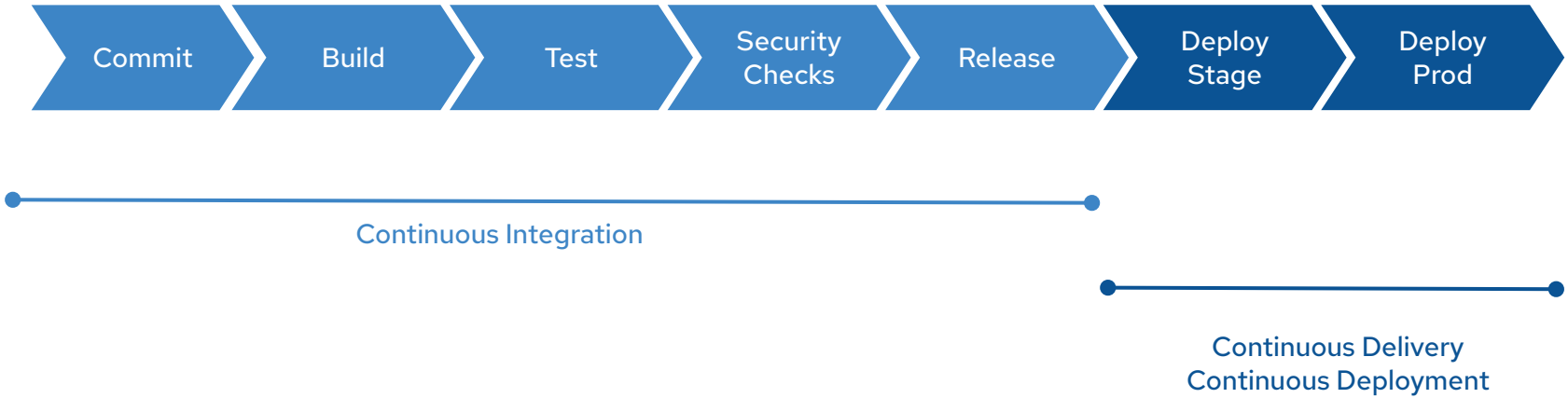
Git 부터 AWS 까지

CI/CD

기본 개념

CI/CD 란 무엇인가?

Continuous Integration(CI) & Continuous Delivery (CD)



CI/CD 란 무엇인가?

Continuous Integration(CI) & Continuous Delivery (CD)

Continuous Integration

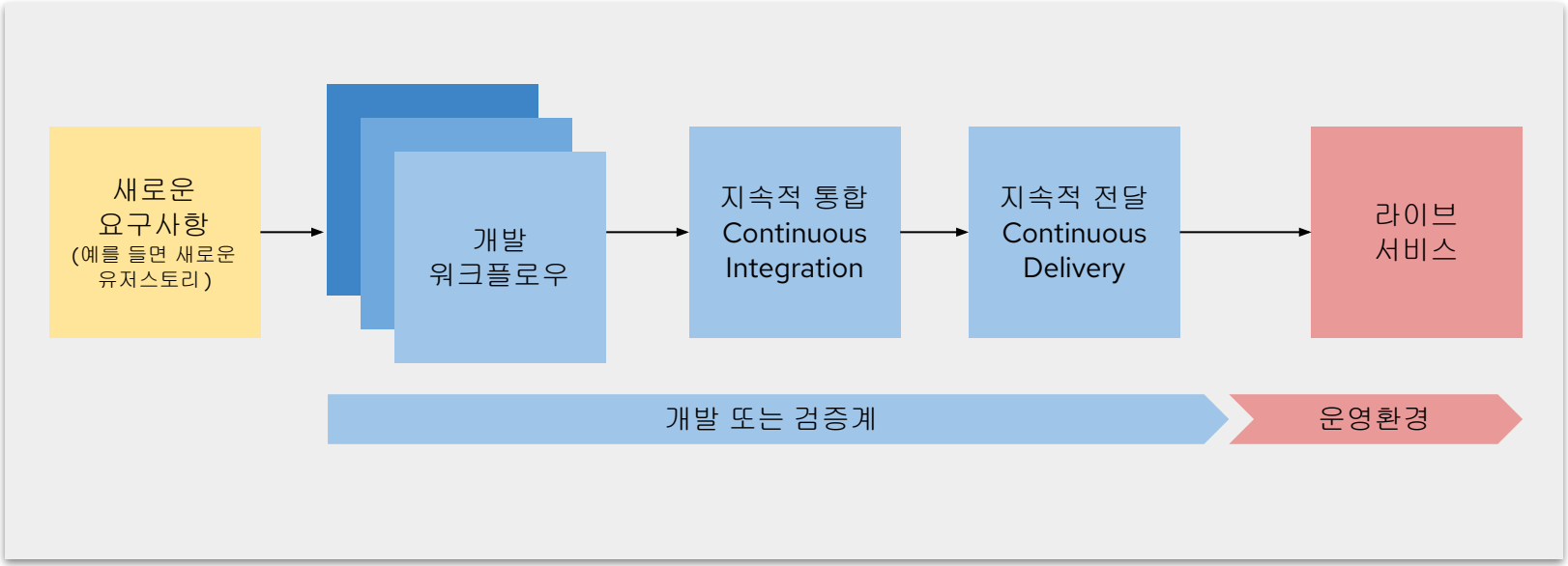
코드가 변경되면 자동으로 빌드,테스트 되고 기존 코드에 최종 머지되는 것

Continuous Delivery

최종 산출물이 만들어지면 자동으로 검증, 운영계에 배포 되는 것

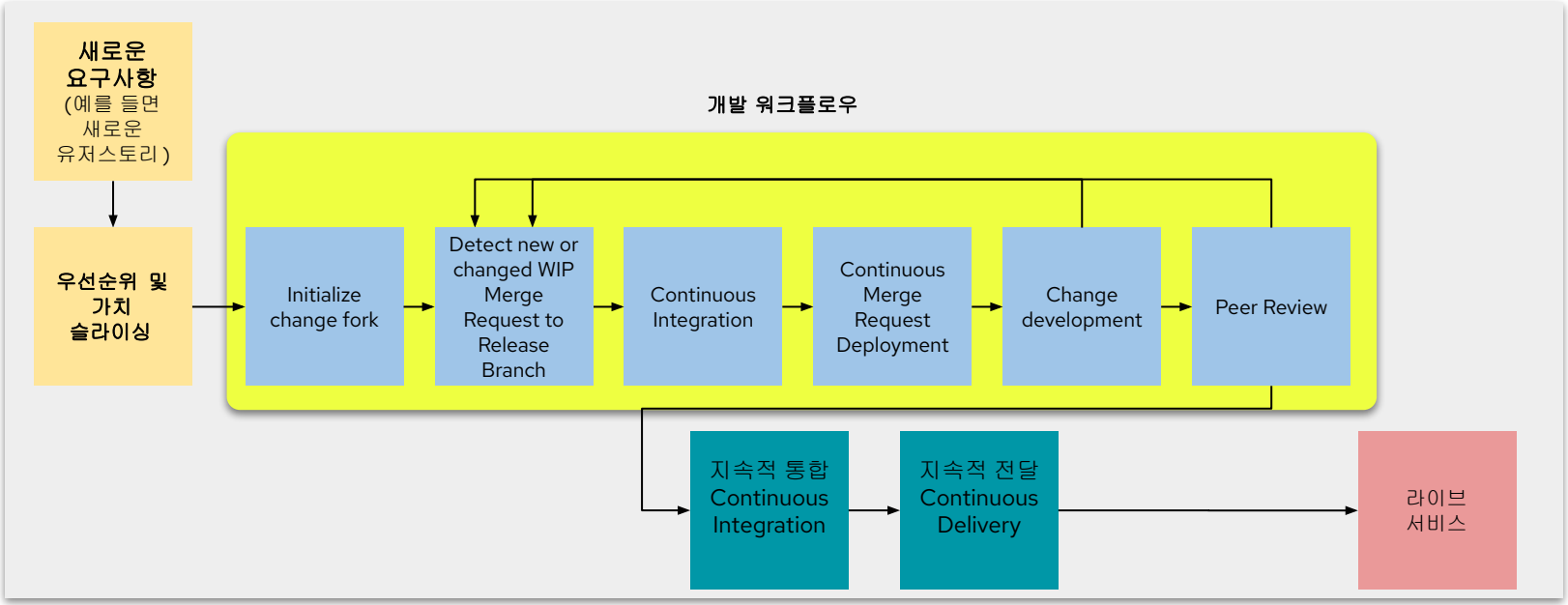
신뢰가능한 소프트웨어 공급망 (TSSC)

Trusted Software Supply Chain (TSSC)



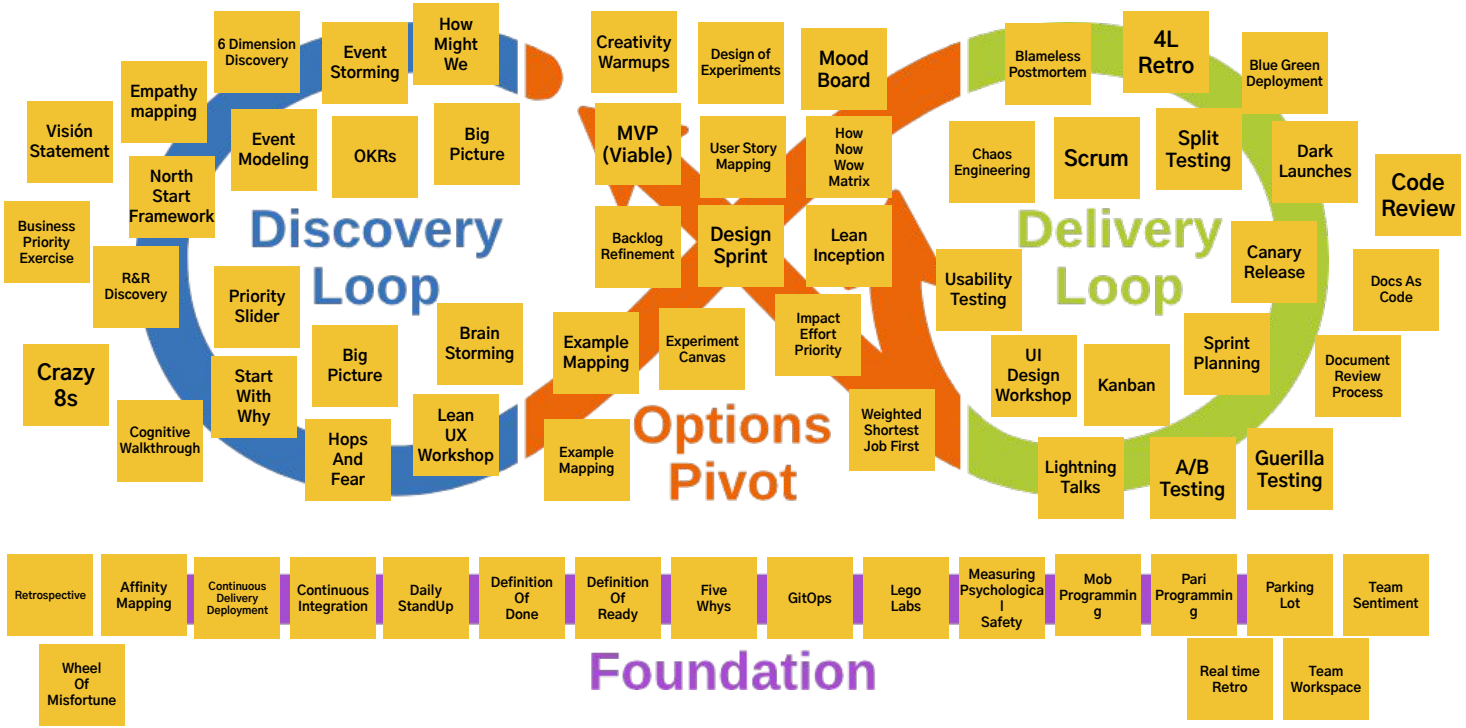
신뢰가능한 소프트웨어 공급망 (TSSC)

Trusted Software Supply Chain (TSSC)



신뢰가능한 소프트웨어 공급망 (TSSC)

Trusted Software Supply Chain (TSSC)



신뢰가능한 소프트웨어 공급망 (TSSC)

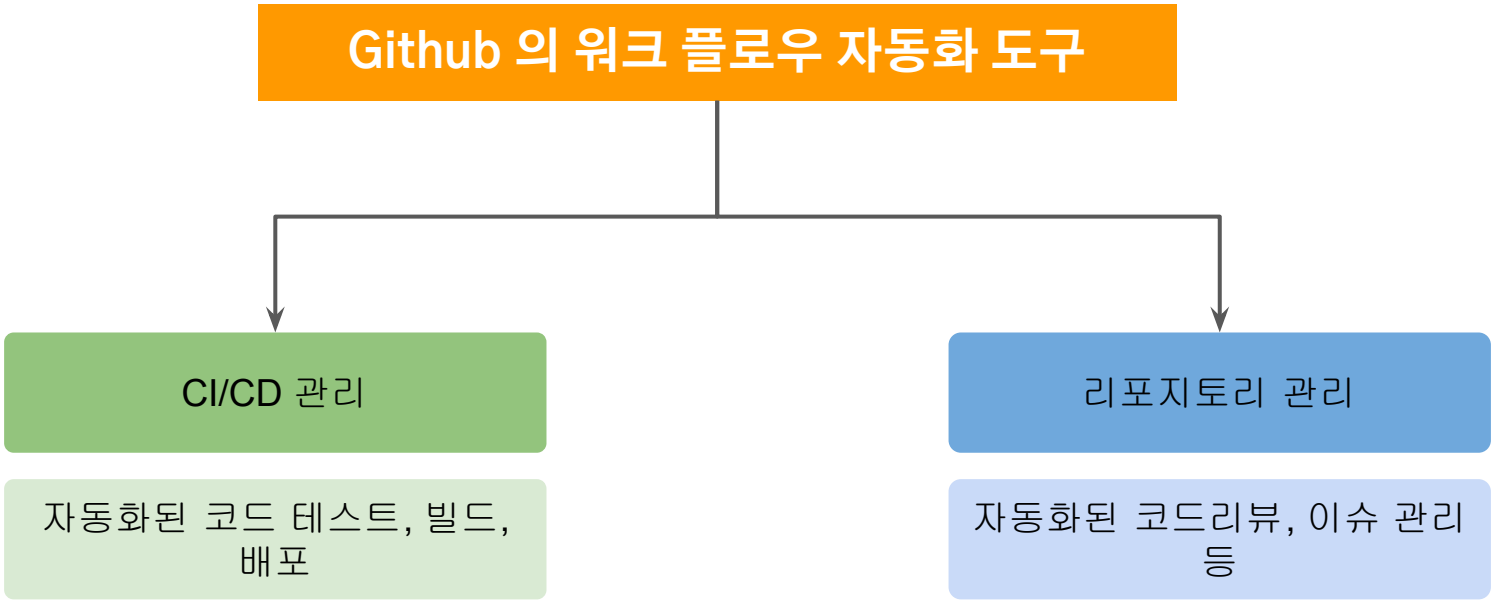
CI/CD 의 변화

Traditional vs Cloud-Native

Traditional CI/CD	Cloud-Native CI/CD
Designed for **Virtual Machines**	Designed for **Containers and Kubernetes**
Require IT Ops for CI engine maintenance	Pipeline as a service with no Ops overhead
Plugins shared across CI engine	Pipelines fully isolated from each other
Plugin dependencies with undefined update cycles	Everything lifecycle as container images
No interoperability with Kubernetes resources	Native Kubernetes resources
Admin manages persistence	Platform manages persistence
Config baked into CI engine container	Configured via Kubernetes ConfigMaps

Github Action

Github Action 이란?



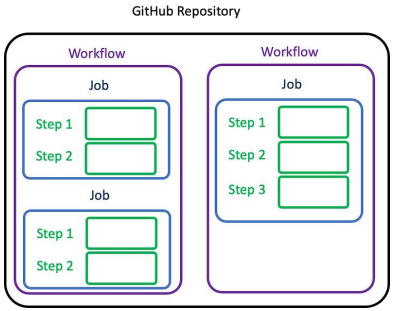
Github Action 이란?

- 소프트웨어 개발 **워크플로우**를 **자동화** 해주는 도구 입니다.
- 리포지토리 **내부 또는 외부에서 발생하는 이벤트**에 대해 **반응** 할 수 있습니다.
- 해당 이벤트에 대응하는 **워크플로우를 실행** 할 수 있습니다.

워크플로우란 무엇일까요?

- 워크플로우란 **리포지토리에 안**에서 **특정 작업을 수행**하기 위한 **설정가능한 자동화 프로세스** 입니다.
  - 테스트 코드 수행
  - 패키지 배포
  - 애플리케이션 배포
  - 슬랙 메시지 전송
  - 이슈 오픈 / 전송
  - 코드 정적 검사

하나의 **워크플로우**는 하나 이상의 **JOB**을 가지고 있고, 각각의 JOB은 여러개의 **STEP** 으로 구성되어 있습니다.

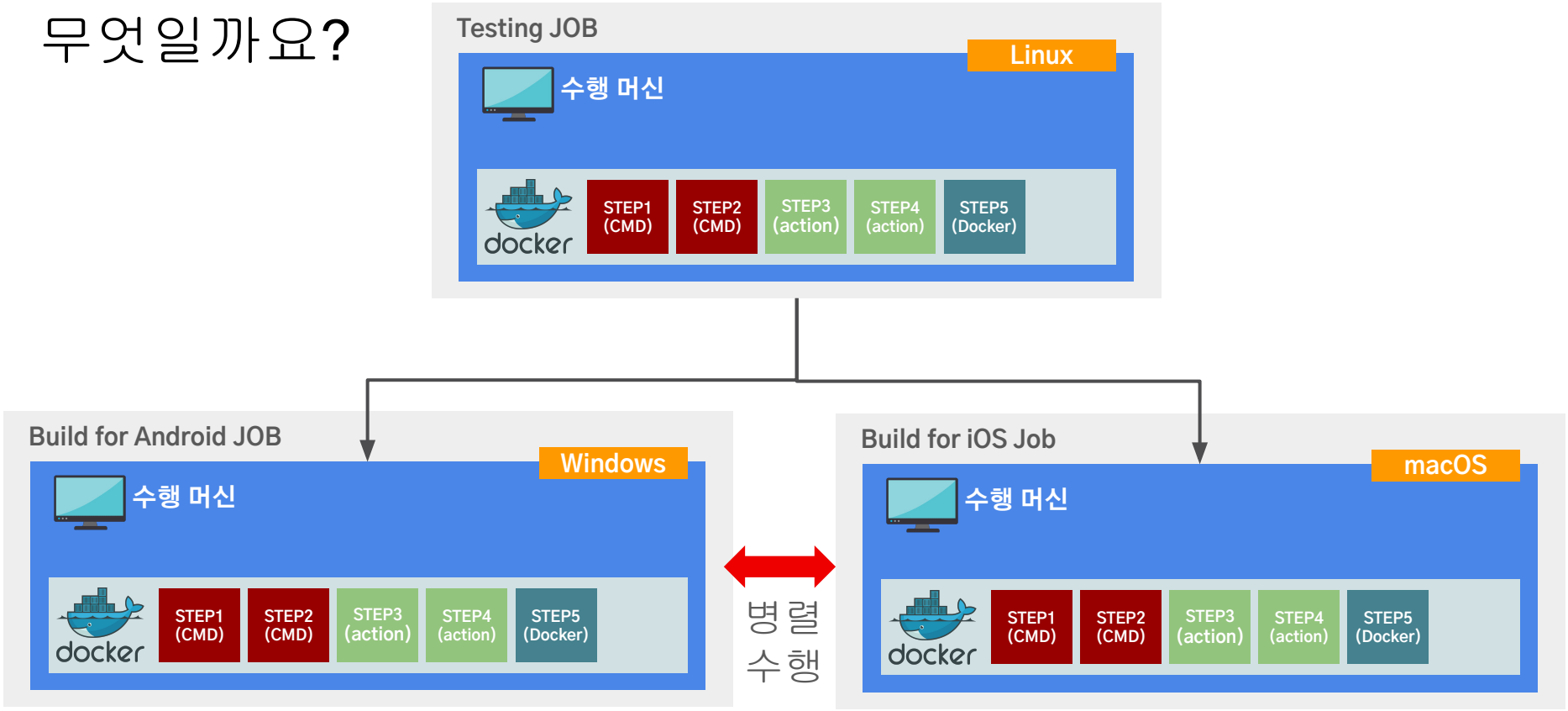


워크플로우란 무엇일까요?

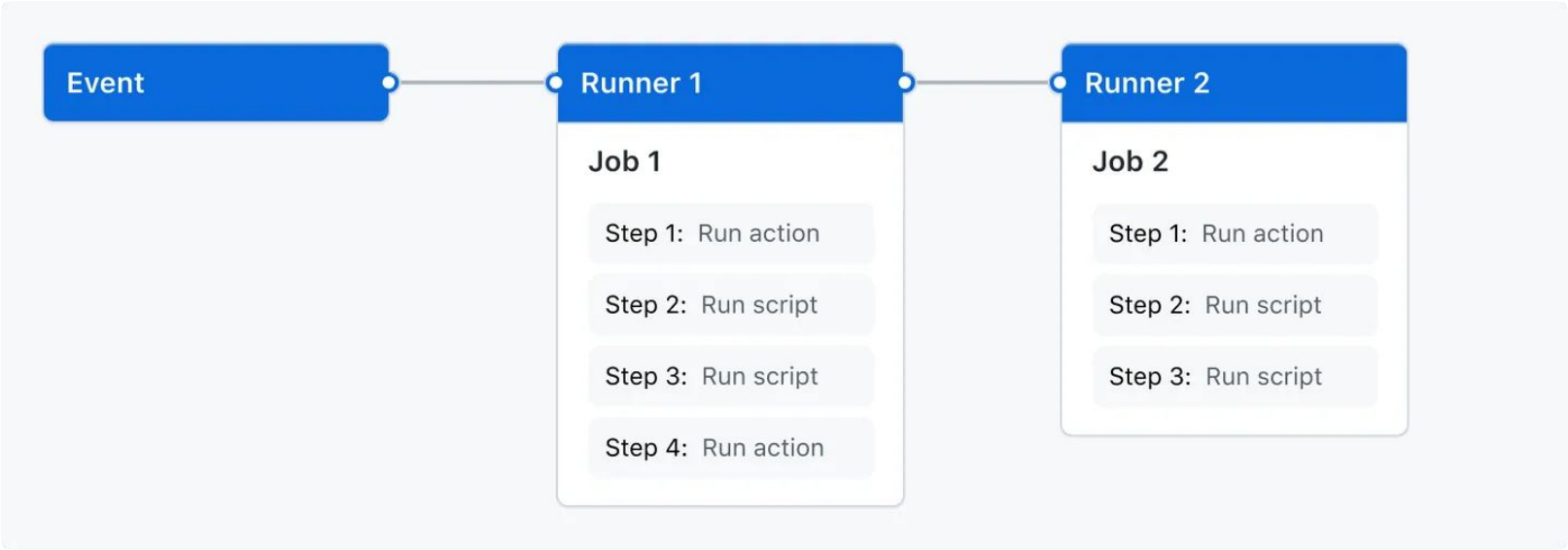
이벤트

- **리포지토리** 이벤트 : Push , Pull, Request, Issue
- **외부** 이벤트 : Sending a POST request to a REST API
- **스케줄된** 이벤트 : 매일 오전 6시
- **수동** 이벤트 : 화면의 버튼 클릭

워크플로우란 무엇일까요?



워크플로우란 무엇일까요?



Github-Hosted Runner

GitHub에서 관리하는 기본 런너로, 여러 운영 체제와 도구가 사전 설치되어 있습니다. 매 작업(run)마다 새로운 클린 환경이 제공됩니다.

주요 특징:

- 각 운영 체제는 다양한 프로그래밍 언어, 빌드 도구, CI/CD 관련 도구들이 사전 설치되어 제공됩니다.
- 매 작업(run) 종료 후 환경이 초기화되어 보안성이 높습니다.
- 퍼블릭 리포지토리에서 무료로 사용할 수 있고, 프라이빗 리포지토리는 사용량에 따라 요금이 부과됩니다.

Self-Hosted Runner

사용자가 직접 관리하는 런너로, GitHub에서 제공하지 않는 고유한 환경 (클라우드나 자체 데이터 센터) 이 필요한 경우 사용할 수 있습니다.

주요 특징:

- 고유한 환경이나 내부 네트워크 리소스가 필요한 경우 유용합니다.
- GitHub-hosted와 달리 사용량에 따른 요금이 없지만, 하드웨어, 운영 체제 라이선스, 유지보수에 대한 비용이 발생합니다.
- 매 작업마다 환경이 초기화되지 않으므로, 사용 후 환경 관리를 직접 해야 합니다.

Runner 비교

항목	Github-hosted Runner	Self-hosted Runner
관리 및 유지보수	Github이 제공하고 및 유지보수	사용자가 직접 관리
환경 설정	사전 설정된 OS와 툴 제공	원하는 설정과 스펙으로 직접 설정 가능
비용	사용량에 따라 부과(퍼블릭 무료)	하드웨어 등의 유지보수 비용 발생
사용사례	일반적인 CI/CD 작업	특수하고, 고성능 하드웨어 필요한 경우
보안성	매번 실행 할때마다 클린 환경제공	작업 완료 후 환경이 유지됨

Runner 별 지원되는 OS는 아래 링크 확인
Github-hosted Runner : https://docs.github.com/en/actions/using-github-hosted-runners/using-github-hosted-runners/about-github-hosted-runners
Self-hosted Runner : https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/about-self-hosted-runners

Runner Machine의 기본 환경 변수

주요 환경변수	설명
`$GITHUB_ACTOR`	워크플로를 시작한 사람 또는 앱의 이름
`$GITHUB_REPOSITORY`	소유자 및 리포지토리 이름 (예: dangtong76/actions)
`$GITHUB_SHA`	워크플로우를 트리거한(호출) Commit 의 해쉬값
`$GITHUB_WORKSPACE`	STEP에 대한 Runner Machine의 기본 작업 디렉토리 및 체크아웃 작업을 사용할 때 기본 위치입니다.
`$GITHUB_REF`	워크플로우를 트리거한(호출) Git 참조 (브렌치 또는 태그)의 전체 경로 브랜치 : refs/heads/<branch_name> 태그 : refs/tags/<tag_name>

참조 URL : https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/store-information-in-variables

워크플로우 환경변수

주요 환경변수	설명
$GITHUB_ACTOR	워크플로를 시작한 사람 또는 앱의 이름
$GITHUB_REPOSITORY	소유자 및 리포지토리 이름 (예: dangtong76/actions)
$GITHUB_SHA	워크플로우를 트리거한(호출) Commit 의 해쉬값
$GITHUB_WORKSPACE	STEP에 대한 Runner Machine의 기본 작업 디렉토리 및 체크아웃 작업을 사용할 때 기본 위치입니다.
$GITHUB_REF	워크플로우를 트리거한(호출) Git 참조 (브렌치 또는 태그)의 전체 경로 브랜치 : refs/heads/<branch_name> 태그 : refs/tags/<tag_name>

참조 URL : https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/store-information-in-variables

Event 다루기

Event란?

Event는 **워크플로우를 트리거(실행)하는 특정 작업 또는 조건**을 의미합니다. Event는 GitHub 리포지토리에서 발생하는 다양한 활동에 따라 자동으로 워크플로우를 실행하도록 설정할 수 있으며, 이를 통해 **개발 과정의 여러 단계를 자동화**할 수 있습니다

리포지토리 관련 이벤트

- Push
- pull_request
- create
- fork
- issues
- issue_comment
- watch
- discussion
- And more!

기타 이벤트

- workflow_dispatch
- repository_dispatch
- schedule
- workflow_call

Event 다루는 방식

Triggered event

```yaml
name: Repository Events
on: [push, pull_request, issues]

jobs:
 checkout-test:
   runs-on: ubuntu-latest
   steps:
     - run: |
         echo $GITHUB_SHA
         echo $GITHUB_REF
     - uses: actions/checkout@v3
```

Triggered event with activity type

```yaml
name: Repository Events
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened]
jobs:
 checkout-test:
   runs-on: ubuntu-latest
   steps:
     - run: |
         echo $GITHUB_SHA
         echo $GITHUB_REF
     - uses: actions/checkout@v3
```

Event 와 Activity Type

Webhook event payload	Activity types	GITHUB_SHA	GITHUB_REF
issues	- opened - edited - deleted - transferred - pinned - unpinned - closed - reopened - assigned - unassigned - labeled - unlabeled - locked - unlocked - milestoned - demilestoned	Last commit on default branch	Default branch

> **Note**: More than one activity type triggers this event. For information about each activity type, see "Webhook events and payloads." By default, all activity types trigger workflows that run on this event. You can limit your workflow runs to specific activity types using the `types` keyword. For more information, see "Workflow syntax for GitHub Actions."

pull_request

Webhook event payload	Activity types	GITHUB_SHA	GITHUB_REF
pull_request	- assigned - unassigned - labeled - unlabeled - opened - edited - closed - reopened - synchronize - converted_to_draft - locked - unlocked - enqueued - dequeued - milestoned - demilestoned - ready_for_review - review_requested - review_request_removed - auto_merge_enabled - auto_merge_disabled	Last merge commit on the GITHUB_REF branch	PR merge branch refs/pull/PULL_REQUEST_NUMBER/merge

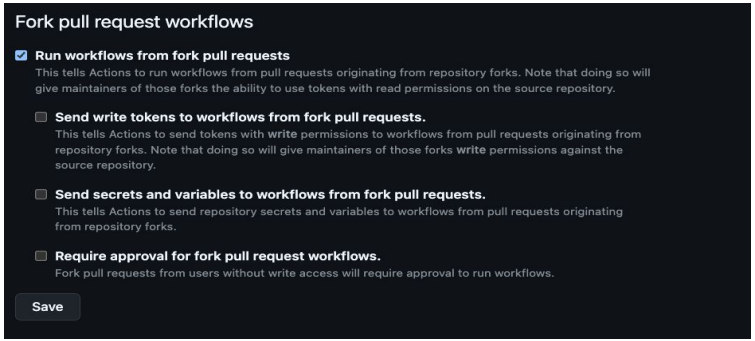
> **Notes**:
>
> - More than one activity type triggers this event. For information about each activity type, see "Webhook events and payloads." By default, a workflow only runs when a `pull_request` event's activity type is `opened`, `synchronize`, or `reopened`. To trigger workflows by different activity types, use the `types` keyword. For more information, see "Workflow syntax for GitHub Actions."

관련 URL : https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows

Fork 사용자의 Pull request 와 Workflow 접근 제어

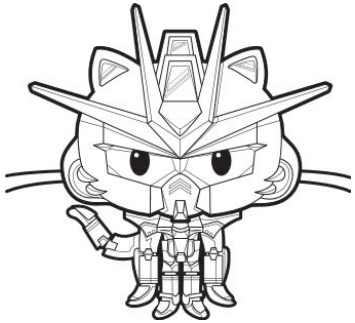
Public Repository

Private Repository



Github API 사용

공식적으로 제공되는 Octokit 라이브러리를 사용하거나 Third party 라이브러리를 사용하여 Github API 사용.
REST API를 사용하는 것보다 Octokit 을 사용 하는 것이 일반적.
API호출이 간편하고, 자동화된 인증 및 오류 처리를 제공하며 타입지원 및 자동완성 등등을 제공하여 octokit 을 더선호



Octokit - GitHub API Client Library for .NET, JS, Ruby, Terraform

참조 URL : https://docs.github.com/en/rest/using-the-rest-api/libraries-for-the-rest-api?apiVersion=2022-11-28

https://octokit.github.io/rest.js/v19/

선행 워크플로우 및 후행 워크플로우

- 워크 플로우 A 가 완료 된 위에 워크 플로우 B를 호출 할 수 있습니다.

```
name: Workflow A
on: [push, pull_request, issues]
```

```
name: Workflow B
on:
 workflow_run:
   workflows: [Workflow A]
   types: [completed] #requested 도 있음
```

- 선행 워크 플로우 A 의 종료 상태 조회는 github.event.workflow_run.conclusion 를 사용하면 됩니다.
- conclusion은 4가지 상태 [success, failure, cancelled, timed_out] 을 가집니다.

```
${{ github.event.workflow_run.conclusion == 'success' }}
${{ github.event.workflow_run.conclusion == 'failure' }}
${{ github.event.workflow_run.conclusion == 'cancelled' || github.event.workflow_run.conclusion == 'timed_out' }}
```

- 예를 들면 아래와 같이 Step 에서 사용 할 수 있습니다.

```
steps:
  - name: Check conclusion of preceding workflow
    if: ${{ github.event.workflow_run.conclusion == 'success' }}
    run: echo "Preceding workflow succeeded. Proceeding with follow-up tasks."
```

이벤트 필터 패턴

Pattern	Description (한글 번역)	Example Matches
feature/*	`*` 와일드카드는 슬래시(`/`)를 제외한 모든 문자를 매칭합니다. (feature/ 뒤에 슬래시가 없는 브랜치나 태그 이름과만 일치)	feature/my-branch, feature/your-branch
feature/**	`**` 와일드카드는 슬래시(`/`)를 포함한 모든 문자를 브랜치와 태그 이름에서 매칭합니다. (feature/ 뒤에 여러 번의 슬래시가 있어도 매칭)	feature/beta-a/my-branch, feature/your-branch, feature/mona/the-octocat
main	브랜치나 태그 이름과 정확히 일치하는 이름을 매칭합니다.	main
releases/mona-the-octocat	브랜치나 태그 이름과 정확히 일치하는 이름을 매칭합니다.	releases/mona-the-octocat
'*'	슬래시(`/`)가 포함되지 않은 모든 브랜치 및 태그 이름을 매칭합니다. `*` 문자는 YAML에서 특수 문자입니다. 패턴을 `*`로 시작할 때는 반드시 따옴표를 사용해야 합니다.	main, releases
'**'	모든 브랜치와 태그 이름을 매칭합니다. 이는 `branches`나 `tags` 필터를 사용하지 않을 때의 기본 동작입니다.	all/the/branches, every/tag
'*feature'	`*` 문자는 YAML에서 특수 문자입니다. 패턴을 `*`로 시작할 때는 반드시 따옴표를 사용해야 합니다.	mona-feature, feature, ver-10-feature
v2*	`v2`로 시작하는 브랜치와 태그 이름을 매칭합니다.	v2, v2.0, v2.9
v[12].[0-9]+.[0-9]+	주 버전이 1 또는 2인 모든 시맨틱 버전 브랜치와 태그 이름을 매칭합니다.	v1.10.1, v2.0.0

https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#onpushpull_requestpull_request_targetpathspaths-ignore

Github Environment

GitHub의 **Environments**는 환경별로 **시크릿 관리, 보호 규칙, 승인 절차** 등을 설정할 수 있어 배포 안정성과 보안을 높입니다. 환경마다 다른 요구 사항을 적용하고, 승인과 시크릿 사용을 제한하여 실수를 방지하고 안전한 배포 파이프라인을 구축하는 데 유용합니다.

환경 별 Secret 관리

각 환경(production, staging 등)에 고유한 시크릿을 설정. 예를 들면,
환경 별 데이터베이스의 비밀번호 설정

환경 별 보호 규칙

특정 환경에 배포하기 전에 특정 조건을 설정. 예를 들면,
배포 전 **승인이 필요 승인자**를 지정