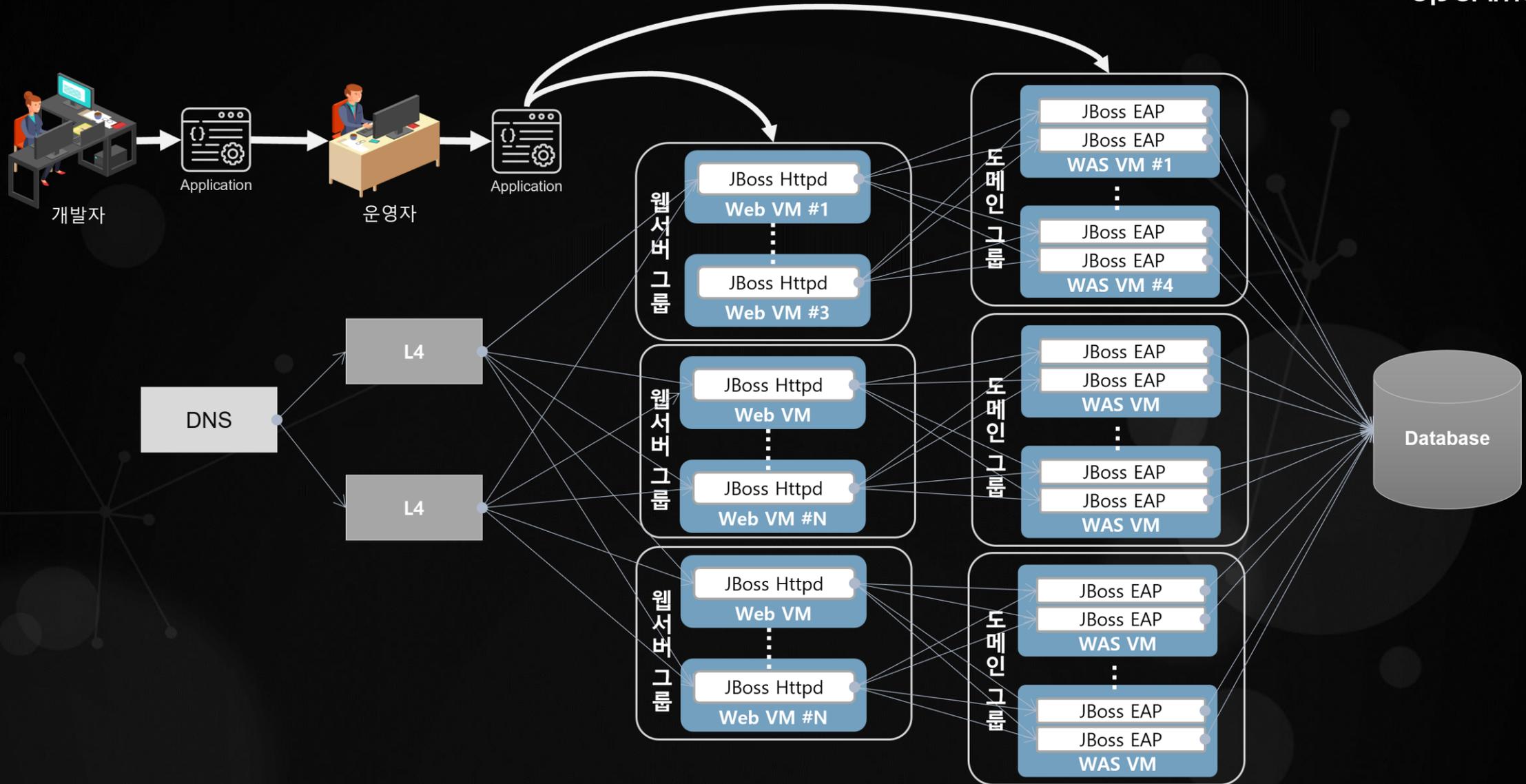
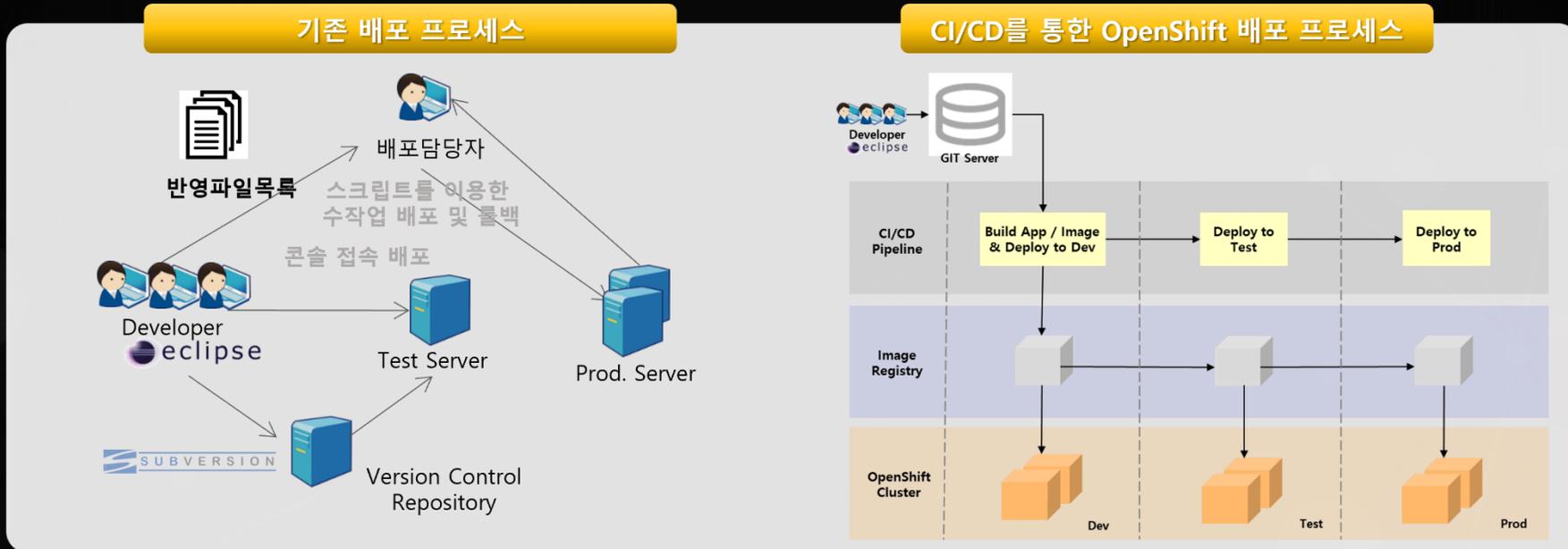


CI/CD
(Continuous Integration / Continuous Delivery)

일반적인 WEB/WAS 환경에서 애플리케이션 배포



컨테이너 환경에서 CI/CD 배포 프로세스 개선



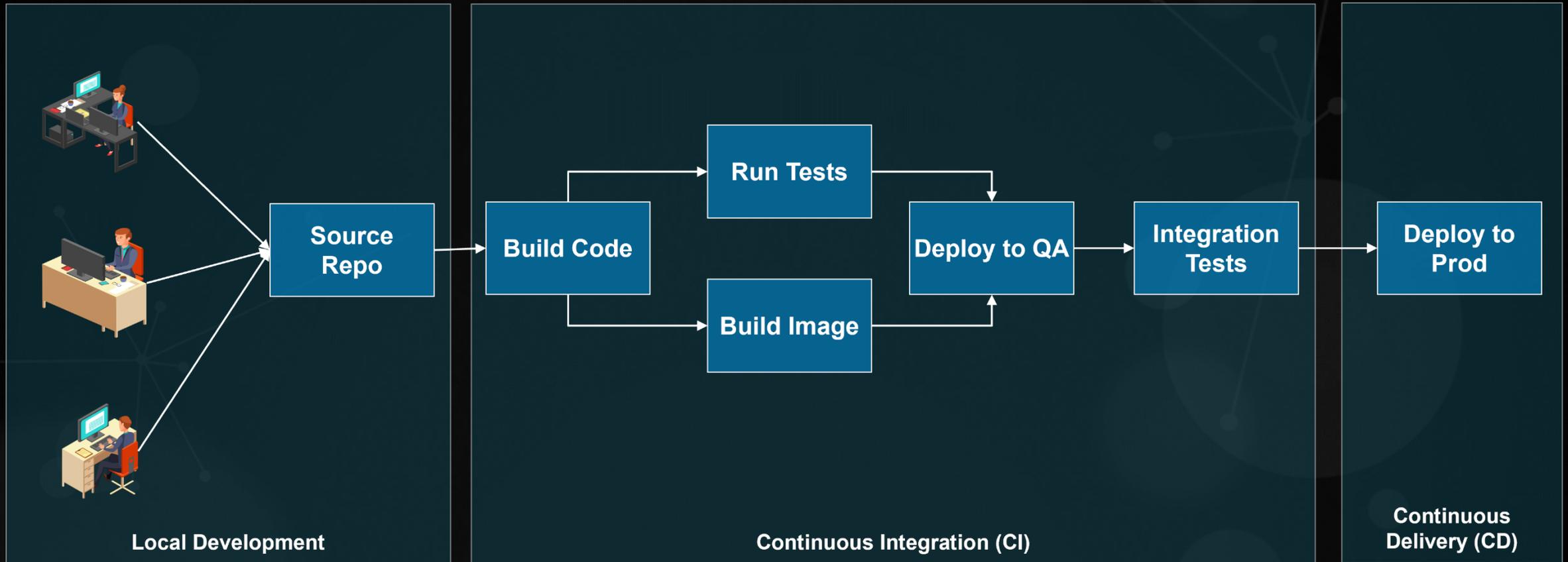
항목	AS-IS	TO-BE
자동화 여부	• 전담 인력에 의한 수작업	• CI도구(GIT)를 이용한 자동 빌드 및 자동 배포
패키징	• 배포 담당자가 수작업 취합 하여 정리	• 형상관리 기준으로 자동 취합
배포 시간	• 수 십분 소요	• 수 분 소요
무중단 배포	• 배포 담당자 수작업에 의한 무중단 배포	• 자동화된 무중단 배포
롤백	• 일부 파일 및 반 자동화 된 롤백	• 기존 운영 버전으로 롤백 (배포될 때 마다 버전을 기록함)
롤백 시간	• 수 십분 소요(배포 시간과 동일)	• 패키징 시간이 없으므로 약 1분이내
배포 인원	• 배포 담당자, 시스템 관리자	• 배포 담당자가 웹기반 UI에서 버튼 클릭 한번 으로 가능
휴먼 장애	• 사람이 하는 일이라 파일 누락 등 실수 가능	• 없음
소스 및 설정 파일	• 서버와 형상 관리와 설정파일 다중 관리	• 형상관리로만 관리
배포검증	• 자동화 불가	• 테스트 케이스, 정적분석(시큐어 코딩), 코드 커버리지 등 다양한 기술 적용 가능

Application Performance Management

CI/CD 개요

커밋에서 배포까지의 파이프 라인

- CI/CD 프로세스

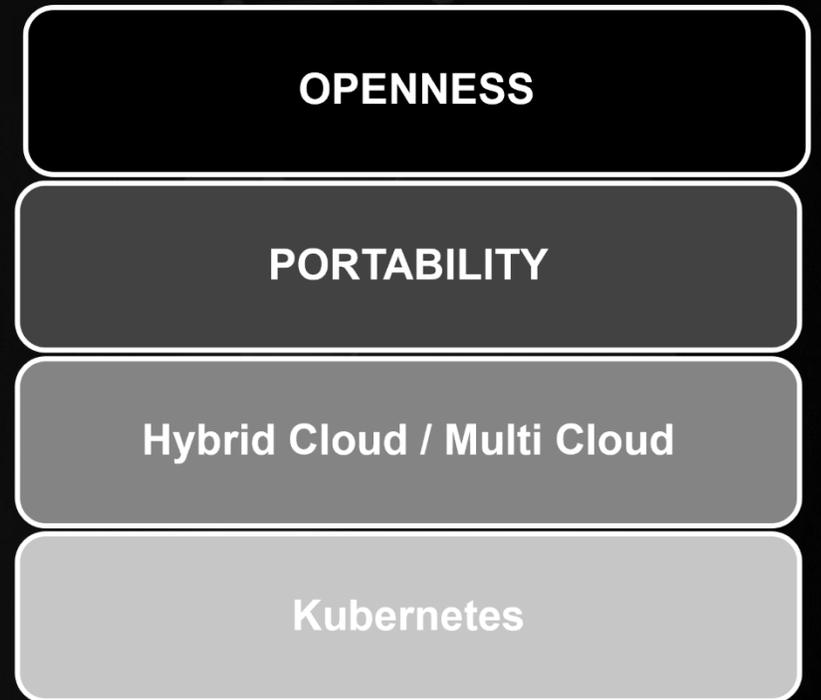


CD Foundation

- 차세대 CD (Continuous Delivery) 협업
- 벤더 중립적 인 기반을 제공

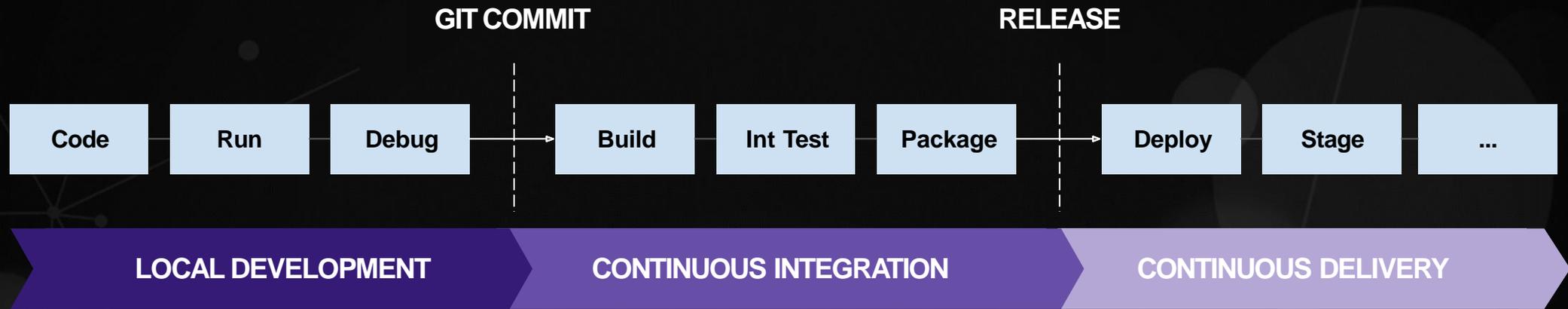


CI / CD 에 요구되는 것



컨테이너와 Kubernetes 등장 후 CI / CD

- 컨테이너의 이동성
 - 애플리케이션의 설정과 종속적인 라이브러리 등을 컨테이너 이미지로 생성
 - 다양한 환경에 손쉽게 애플리케이션 배포
- Kubernetes 선언적인 운영과 환경의 코드화
 - 실행 환경 구성을 코드로 작성하여 여러 클러스터에 동일한 환경을 쉽게 구축



Kubernetes



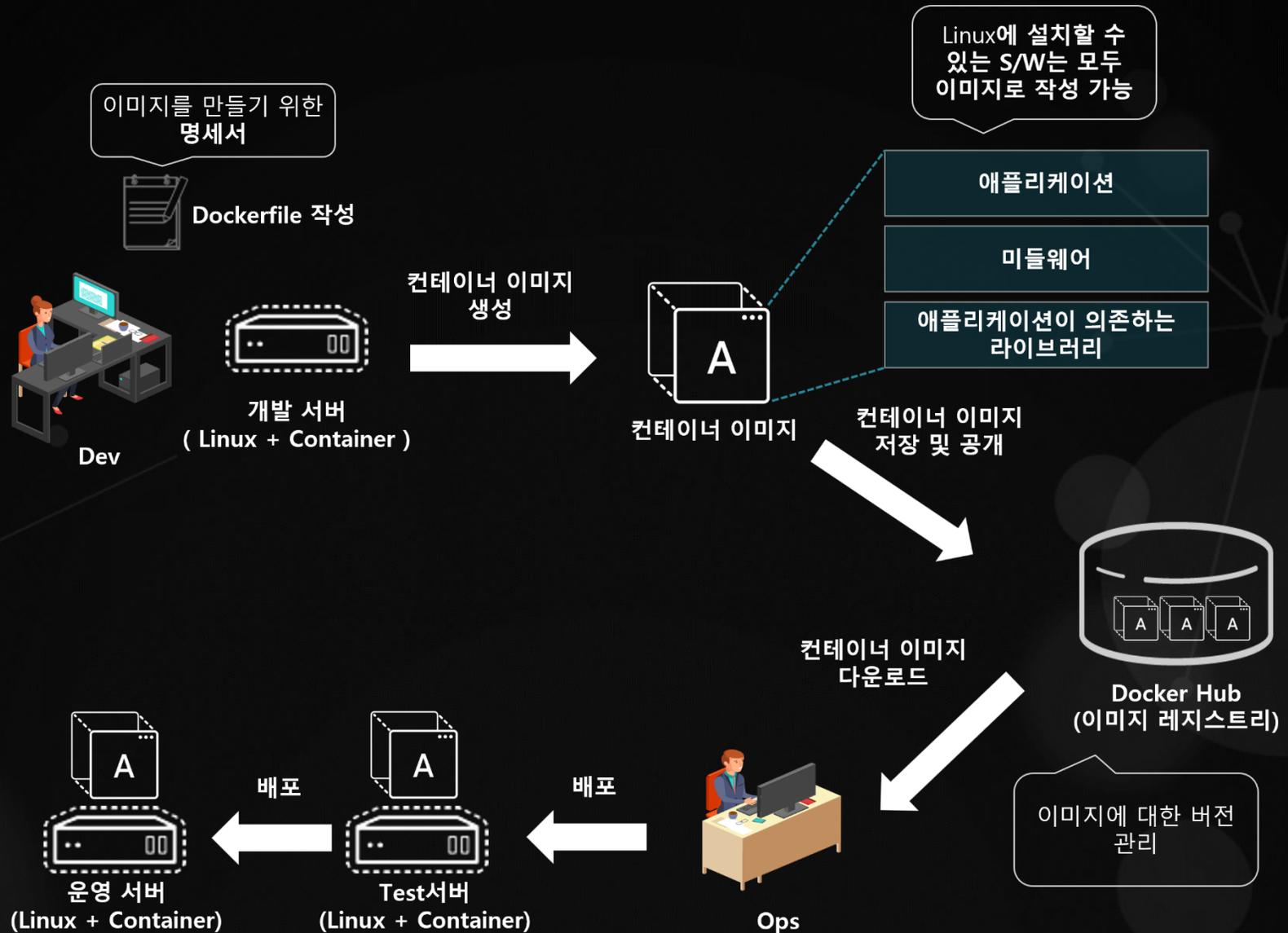
Kubernetes

- Confidential -



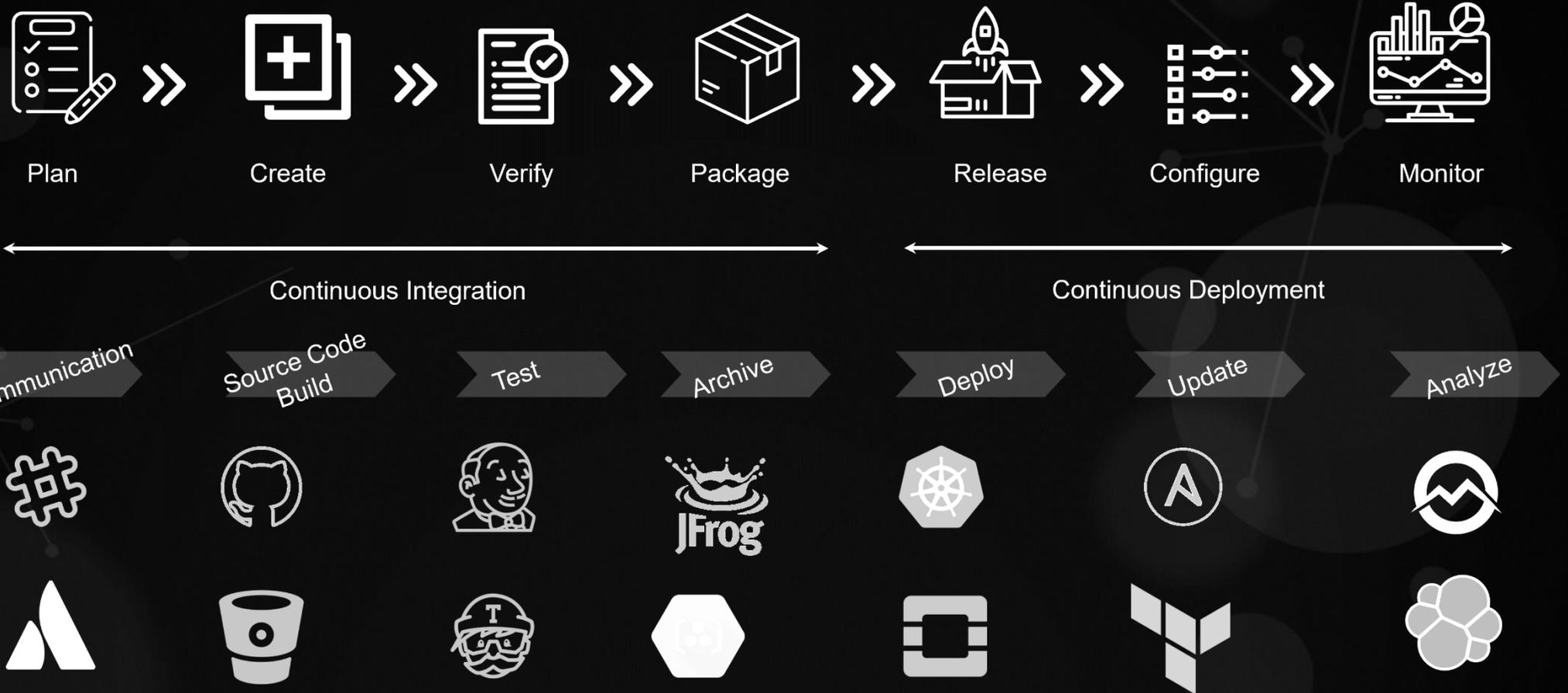
Kubernetes

Build & Deploy Application



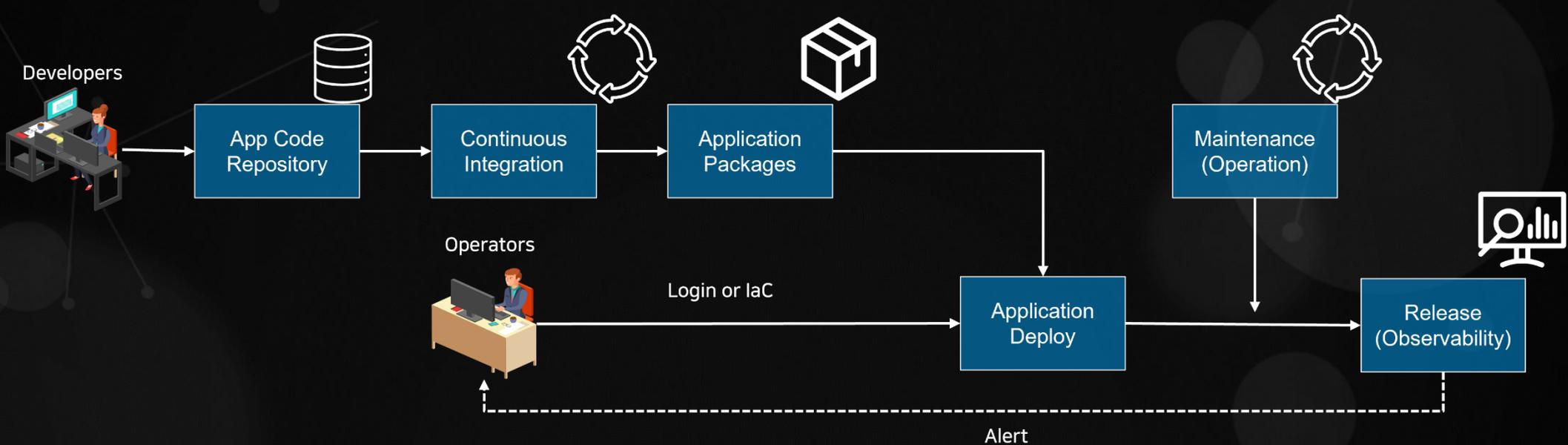
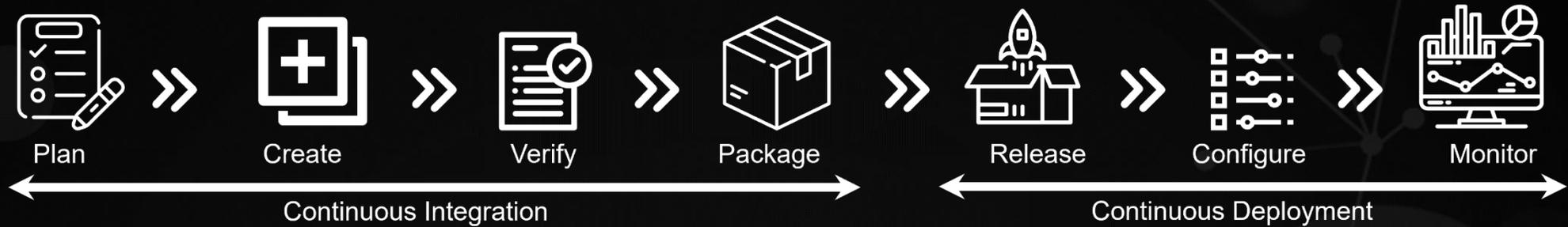
Traditional DevOps

- Toolchain 스테이지마다 다른 소프트웨어를 통합



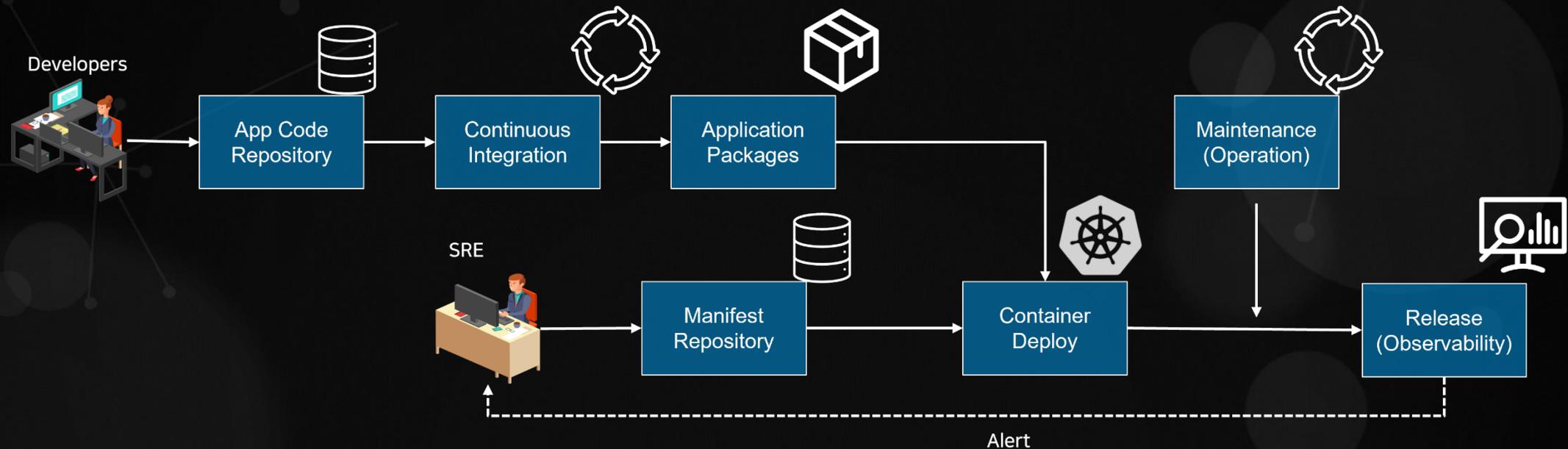
Traditional DevOps

- Toolchain 스테이지마다 다른 소프트웨어를 통합



Container CI / CD

- Manifest 업데이트하면서 사용자 환경에 따라 배포 관리



WHAT IS CI/CD?

- Why Cloud-Native CI/CD?



Jenkins



TEKTON

Traditional CI/CD

Designed for Virtual Machines

Require IT Ops for CI engine maintenance

Plugins shared across CI engine

Plugin dependencies with undefined update cycles

No interoperability with Kubernetes resources

Admin manages persistence

Config baked into CI engine container

Cloud-Native CI/CD

Designed for Containers and Kubernetes

Pipeline as a service with no Ops overhead

Pipelines fully isolated from each other

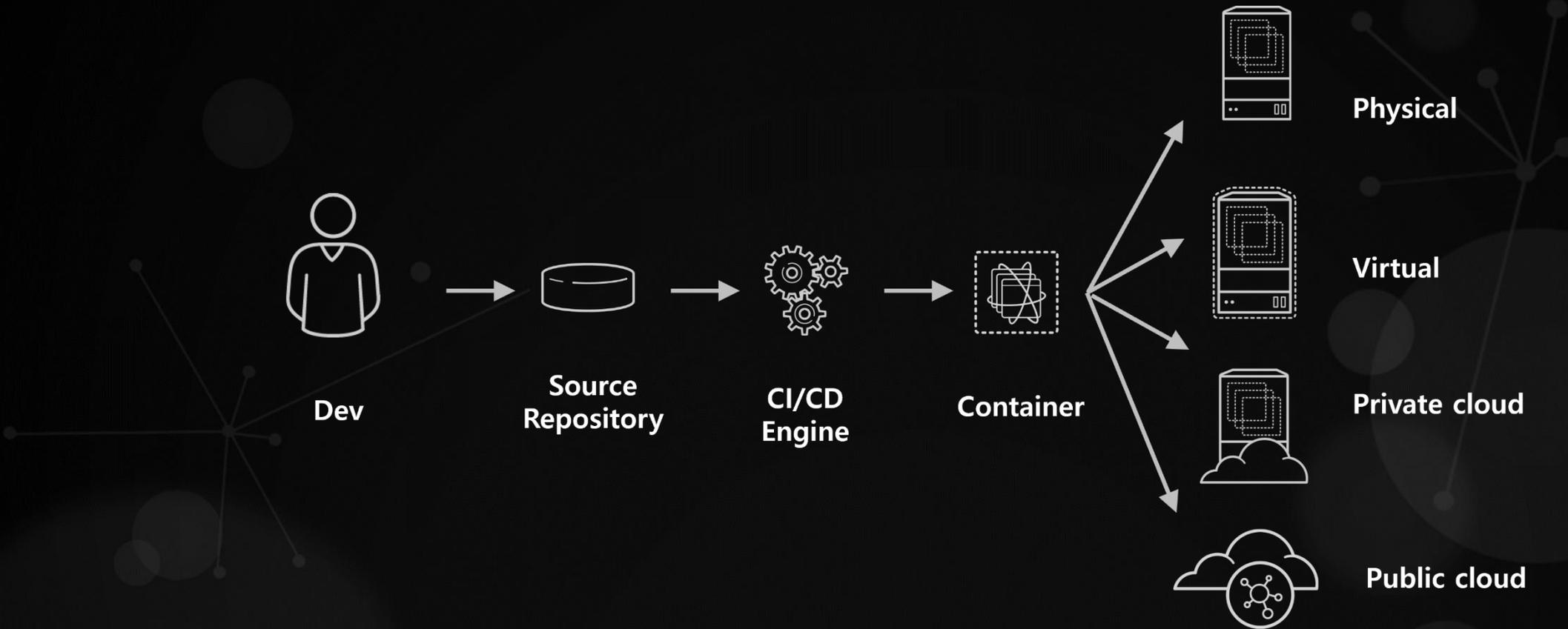
Everything lifecycle'd as container images

Native Kubernetes resources

Platform manages persistence

Configured via Kubernetes ConfigMaps

DevOps With Containers Across the Hybrid Cloud?

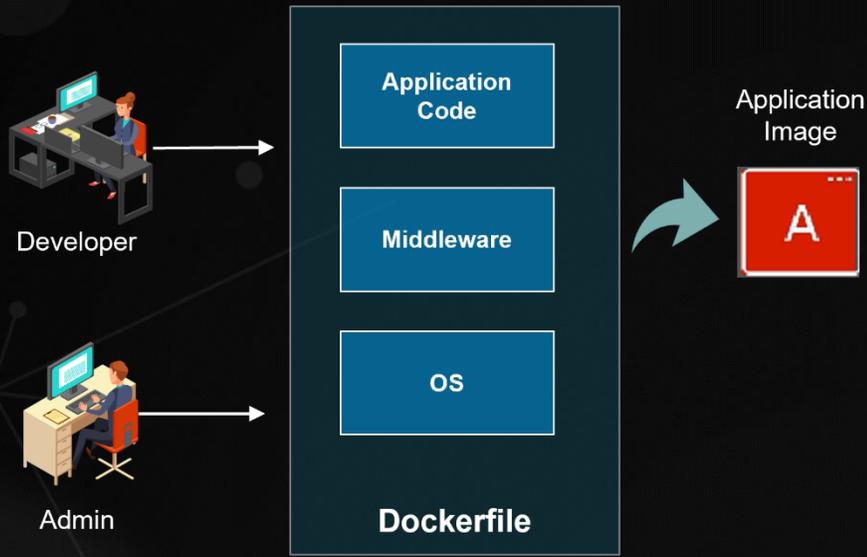




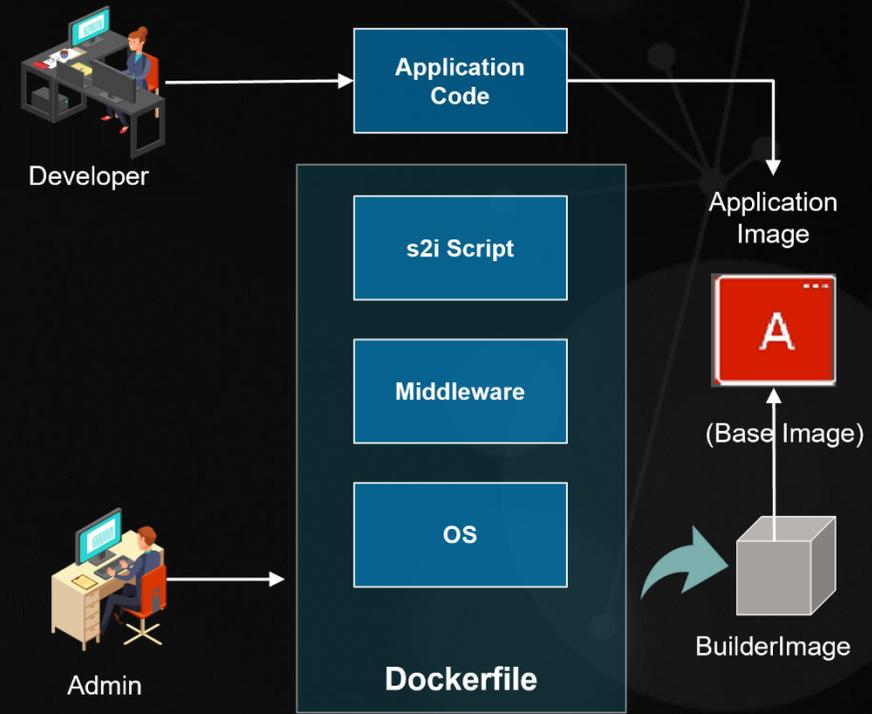
OpenShift Pipelines

OpenShift 빌드

- 개발자와 운영자의 역할을 명확하게 구분하는 빌드 프로세스



기존 컨테이너 빌드



Source-to-Image

OpenShift Pipelines Architecture

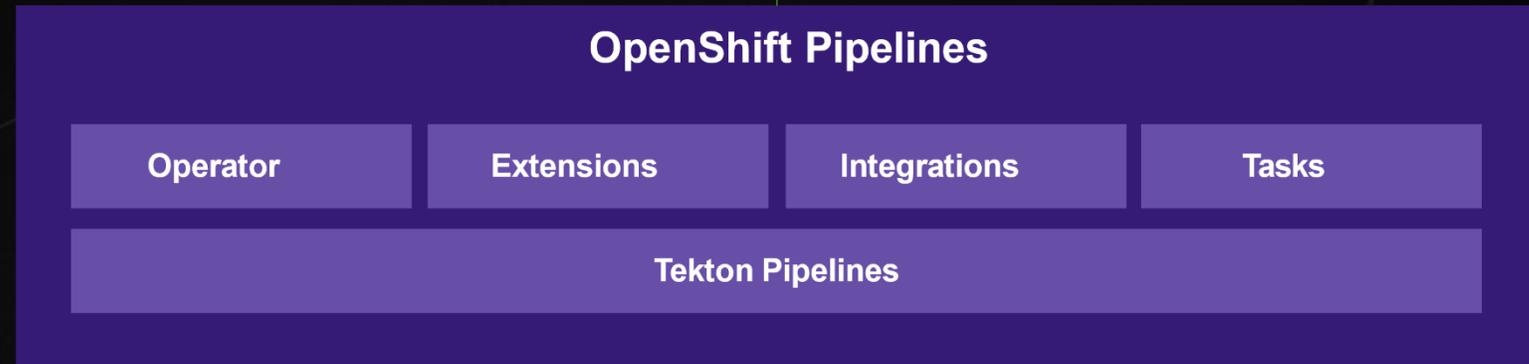


Developer Tools



⚡ API

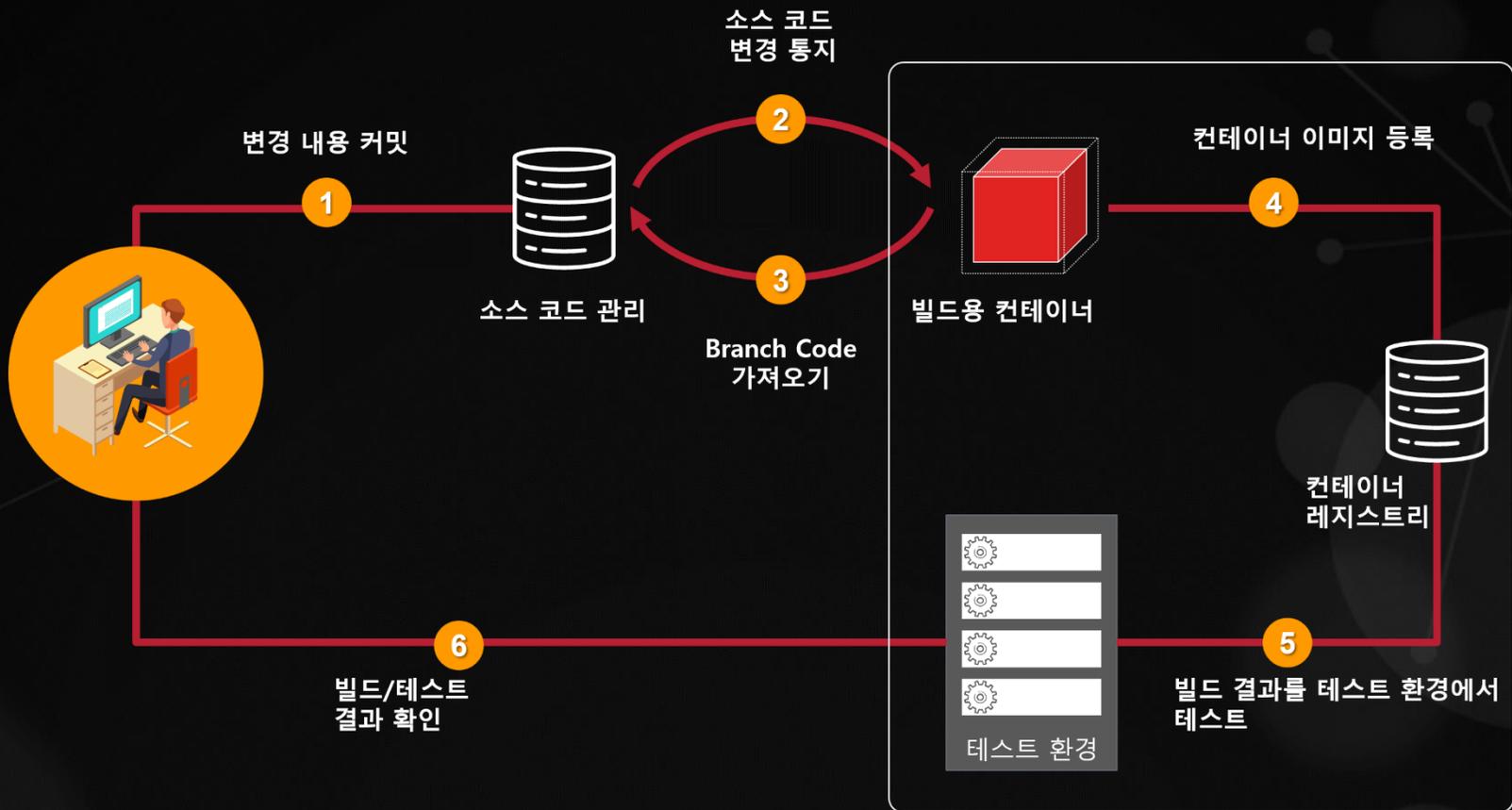
CI/CD Core



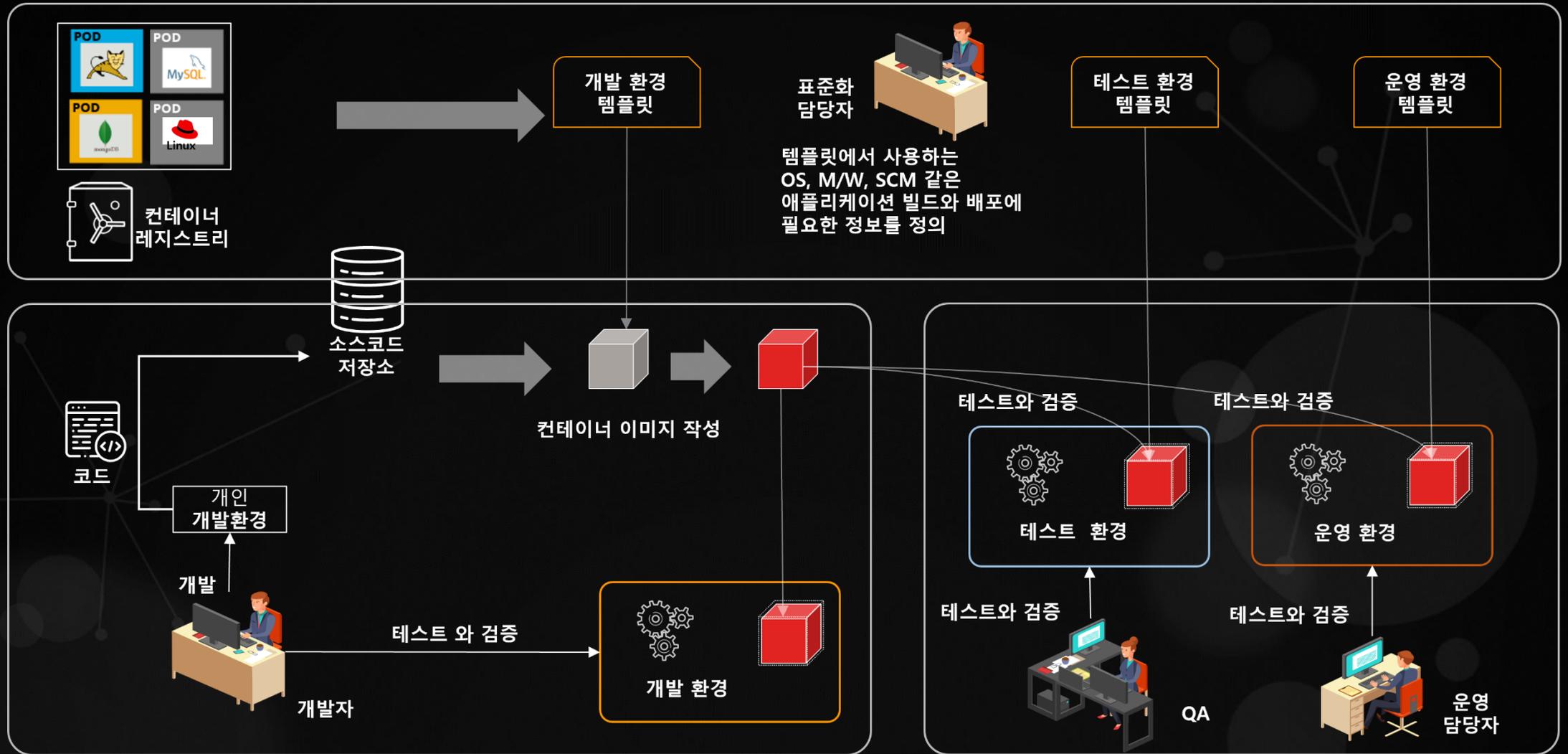
Kubernetes



지속적인 통합 구현



OpenShift 기반의 DevOps 워크 플로우

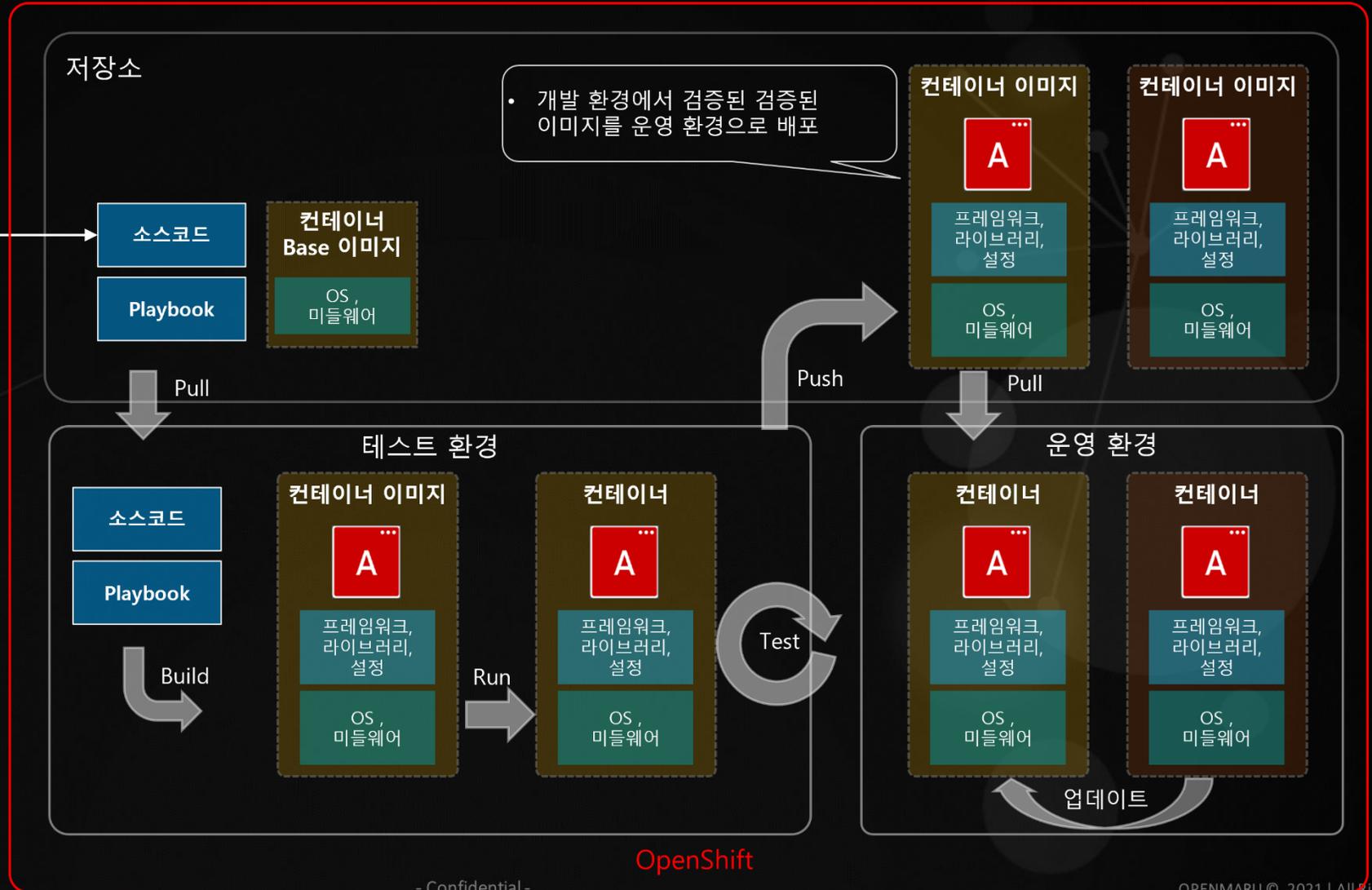


OpenShift 에 의한 테스트 및 배포 자동화

- 소스 코드를 체크인하면 테스트 이후의 프로세스가 자동으로 진행
- 새로운 기능을 개발하는 데 집중

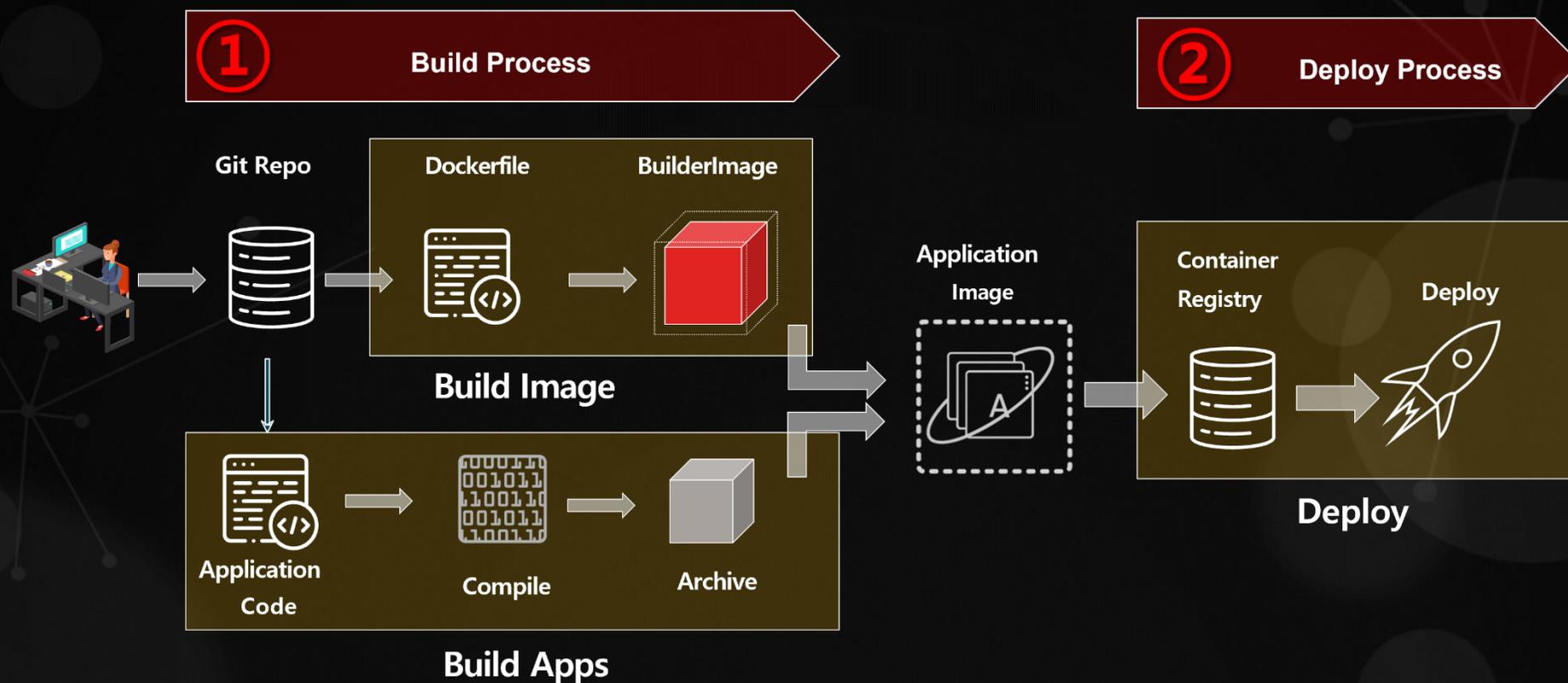


체크인



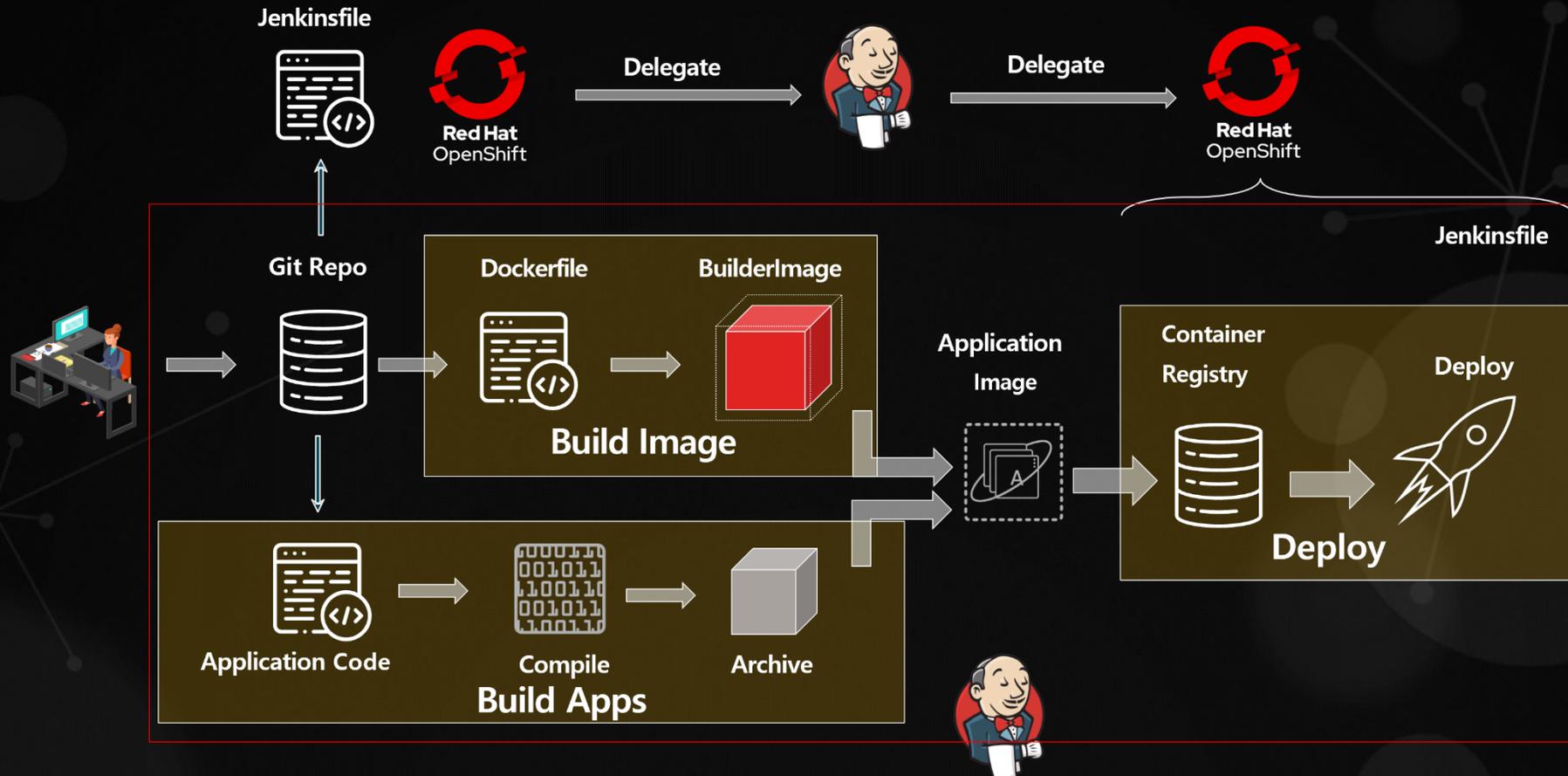
Build & Deploy Application

- Build Process 소스 코드 빌드 및 기반 이미지 빌드로 구성
- 애플리케이션 실행은 " Build Process "와" Deploy Process "로 구성



OpenShift Pipeline (spec.strategy.type = JenkinsPipeline)

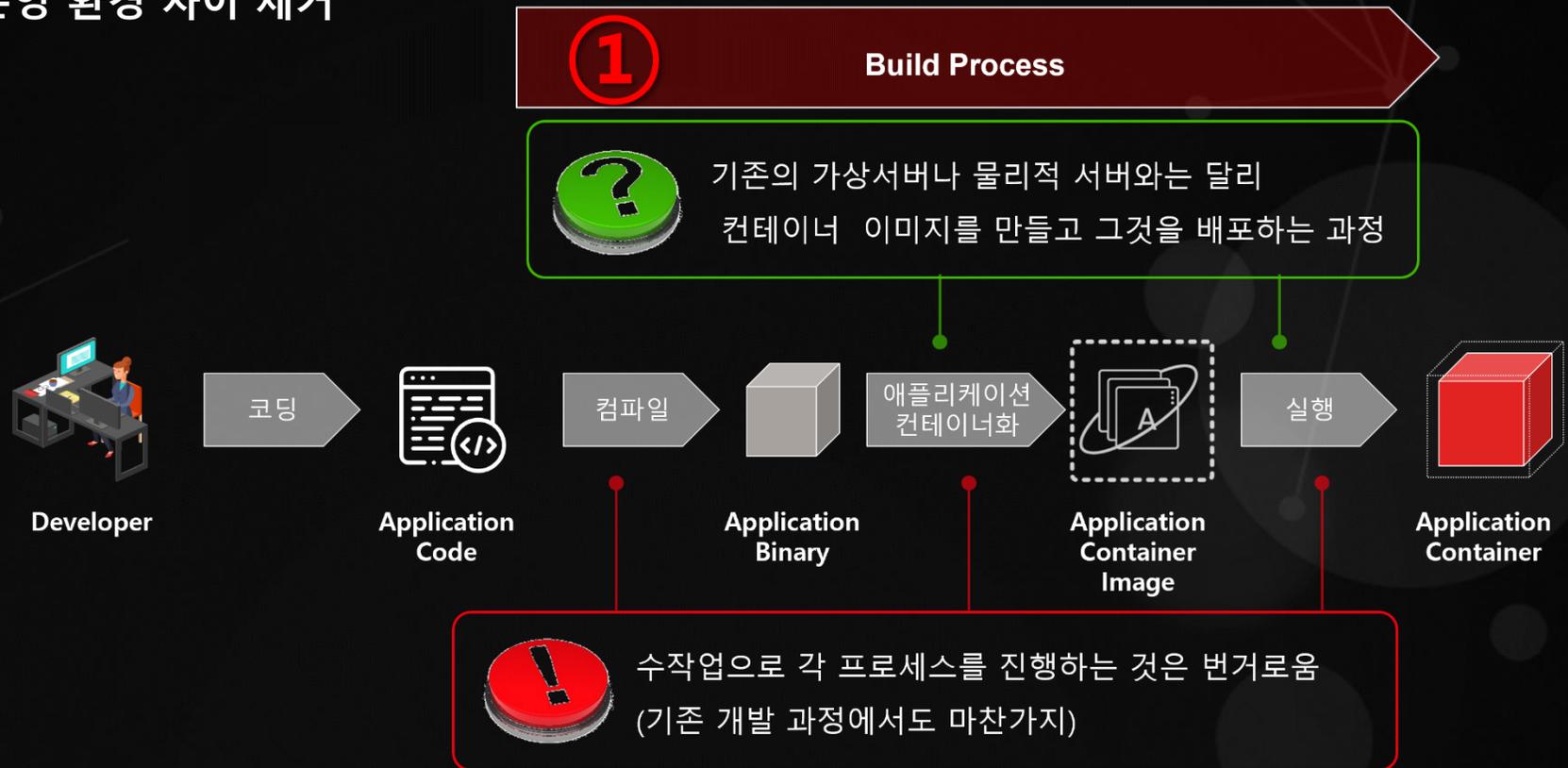
- Build Config 중 Strategy 로 JenkinsPipeline 을 활용



<https://blog.openshift.com/openshift-cloudbees-jenkins-enterprise-devops/>

컨테이너 이미지 작성과 배포

- 컨테이너 기반 어플리케이션 개발 할 때 가장 먼저 접하는 어려운 부분
- 컨테이너 이미지를 통한 개발 생산성 향상
 - 손쉬운 개발 환경 구성
 - 개발/스태이징/운영 환경 차이 제거



OpenShift 에서 개발 및 빌드 프로세스

② Deploy Process

Source-2-Image

- 응용 프로그램 코드를 컨테이너 이미지화
- BuildConfig 자원 정의
- Webhook 트리거를 설정 가능

DeploymentConfig

- Kubernetes 의 Deployment 트리거 기능을 추가
- 컨테이너 이미지의 업데이트를 트리거 OpenShift 위로 컨테이너 이미지 열기



Developer

코딩



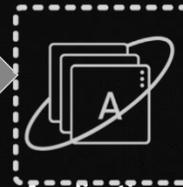
Application Code

컴파일



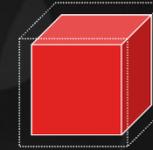
Application Binary

애플리케이션 컨테이너화



Application Container Image

실행



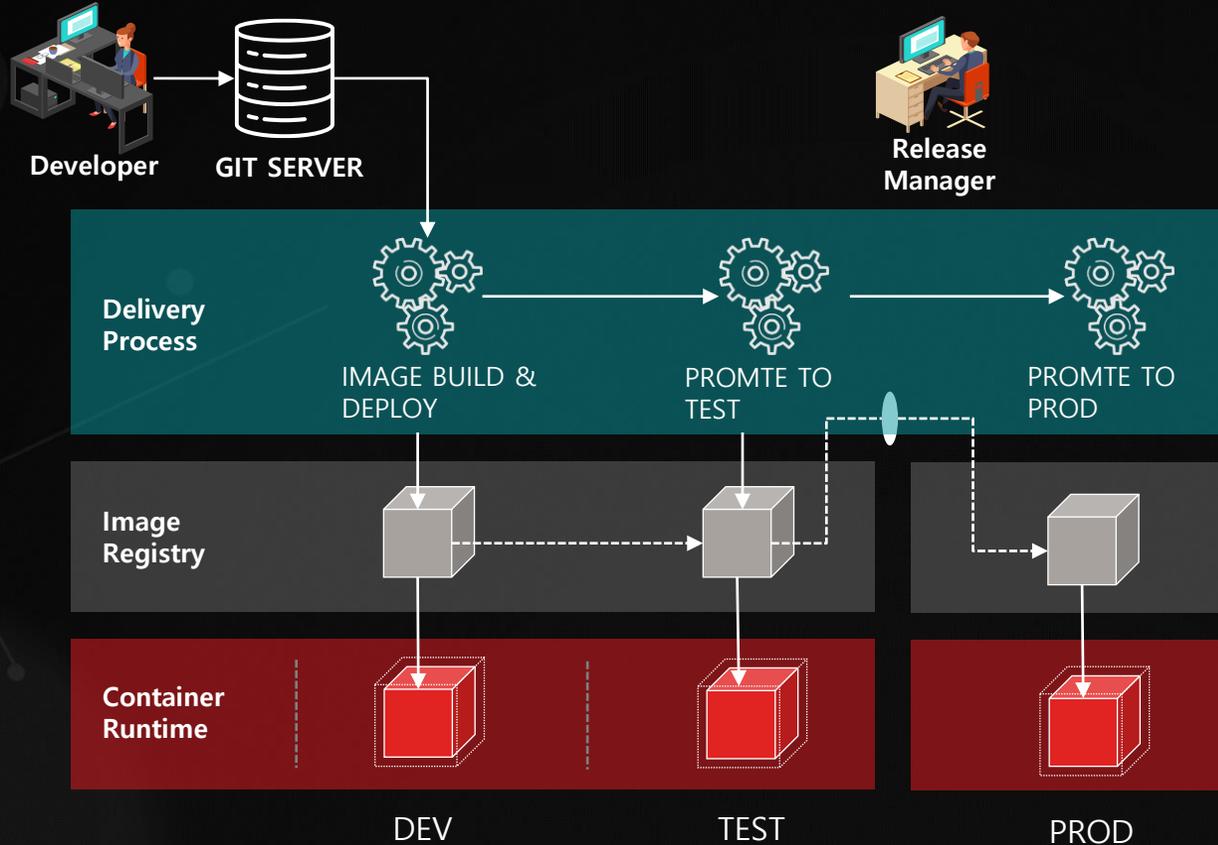
Application Container

oc new-app 명령 (클라이언트 명령) / Catalog (Web 인터페이스)

- BuildConfig / DeploymentConfig 은 yaml 파일에서 정의
- oc new-app catalog 를 사용하여 여러 매개 변수를 입력하는 것으로, yaml 파일을 자동 생성

컨테이너 기반 애플리케이션 릴리스 프로세스 예 - Continuous Delivery

- 애플리케이션 코드 수정 후 컨테이너 빌드, 테스트 및 배포까지의 자동 실행 및 릴리스 하는 방법



OpenShift Pipeline - Jenkins Pipeline



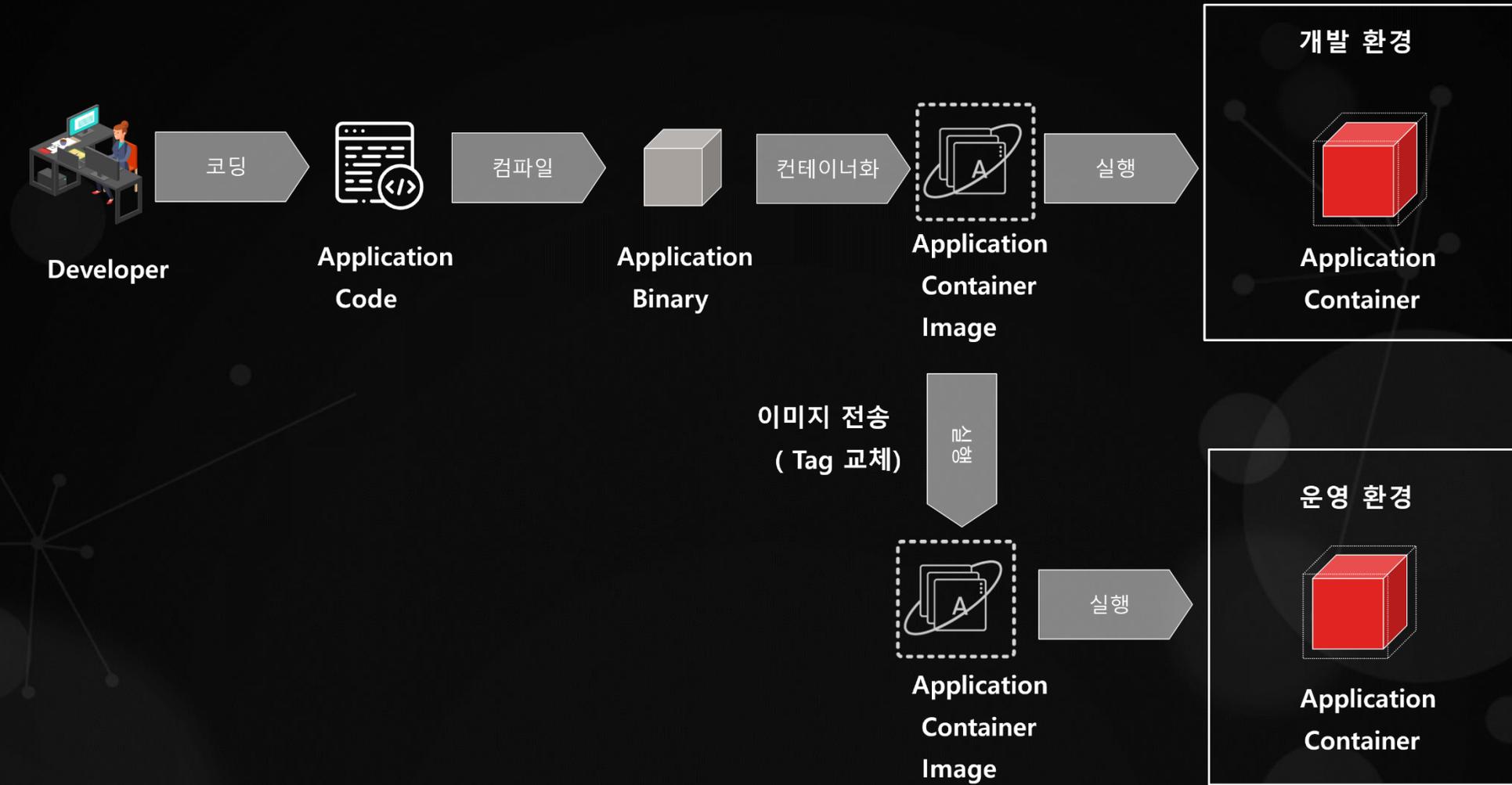
- OpenShift Pipelines 를 사용하면 Jenkins 파이프 라인을 통해 출시 프로세스를 정의하고 시작하고 모니터링 할 수 있습니다.

The screenshot displays the OpenShift Pipelines console interface. On the left is a navigation sidebar with categories like Workloads, Networking, Storage, Builds, Image Streams, Monitoring, Compute, and Administration. The 'Builds' section is expanded, showing 'Builds' as the selected item. The main content area shows the 'tasks-pipeline-1' build details. At the top, there are tabs for Overview, YAML, Environment, Logs, and Events. The 'Overview' tab is active, showing a 'Build Overview' section with a pipeline graph. The graph consists of six stages: Build App (6 minutes ago), Test (5 minutes ago), Code Analysis (5 minutes ago), Archive App (5 minutes ago), Build Image (5 minutes ago), and Deploy DEV (2 minutes ago). All stages are marked with green checkmarks, indicating successful completion. Below the graph, there is a 'Promote to ST...' button with an 'Input Required' warning. At the bottom, a table provides metadata for the build: NAME: tasks-pipeline-1, STATUS: Running, NAMESPACE: cicd-smx, and TYPE: JenkinsPipeline.

NAME	STATUS
tasks-pipeline-1	Running

NAMESPACE	TYPE
cicd-smx	JenkinsPipeline

개발환경과 운영환경 통합



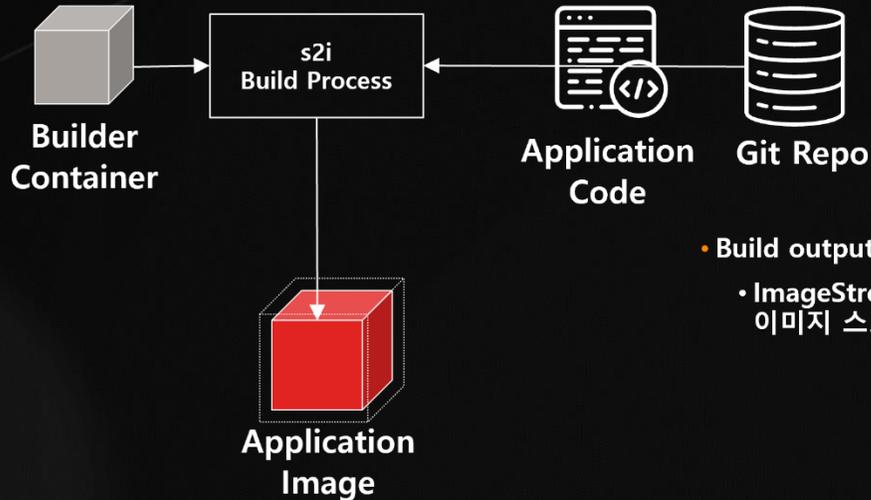
Application Create Objects

- `oc new-app` 를 • 것이다 것으로, 다음 오브젝트가 작성되는

Object	설명
Build Config	<ul style="list-style-type: none">• BuildConfig 명령 줄에서 지정된 각 소스 저장소에 생성됩니다. BuildConfig 이를 사용 •하는 전략, 소스의 위치 및 빌드 나와 • 위치를 지정합니다.
ImageStream	<ul style="list-style-type: none">• BuildConfig 는 보통 2 하나의 ImageStream 가 생성됩니다.• ↗•은 입력 이미지를 나타냅니다. Source 빌드에서는이 빌더 이미지입니다. Docker 빌드에서는이 FROM 이미지입니다• ↘•는 출력 이미지를 나타냅니다. 컨테이너 이미지 new- app 에 입력으로 지정된 경우이 이미지에 대해서도 이미지 스트림이 생성됩니다.
Deployment Config	<ul style="list-style-type: none">• DeploymentConfig 빌드의 출력 • 또는 지정된 이미지 중 하나를 배포하기 위해 만들어집니다. new-app 명령은 결과 로 •되는 DeploymentConfig 에 포함 된 컨테이너에 지정되는 전 Docker 볼륨 emptyDir 볼륨을 만듭니다.
Service	<ul style="list-style-type: none">• new-app 명령은 입력 이미지에서 공개 포트를 감지하려고 시도합니다. 공개 된 포트에서 수치가 가장 낮은 것을 사용 •하 여 해당 포트를 공개하는 서비스를 •성합니다. new-app 완료 후 다른 포트를 게시하려면 단순히 oc expose 명령을 사용 •하 고 추가 서비스를 •성하면됩니다.

- **Build Strategies**

- S2I : 응용 프로그램의 소스 코드를 빌드하고 컨테이너 이미지에 포함
- Docker : Dockerfile 하면 docker build 뿐만 아니라 처리
- Pipeline : Jenkins pipeline 에 의해 빌드 및 배포를 제어
- Custom : 빌드 이미지를 독일 · 만들고 빌드 프로세스를 사용자 정의 할



- **Build inputs**

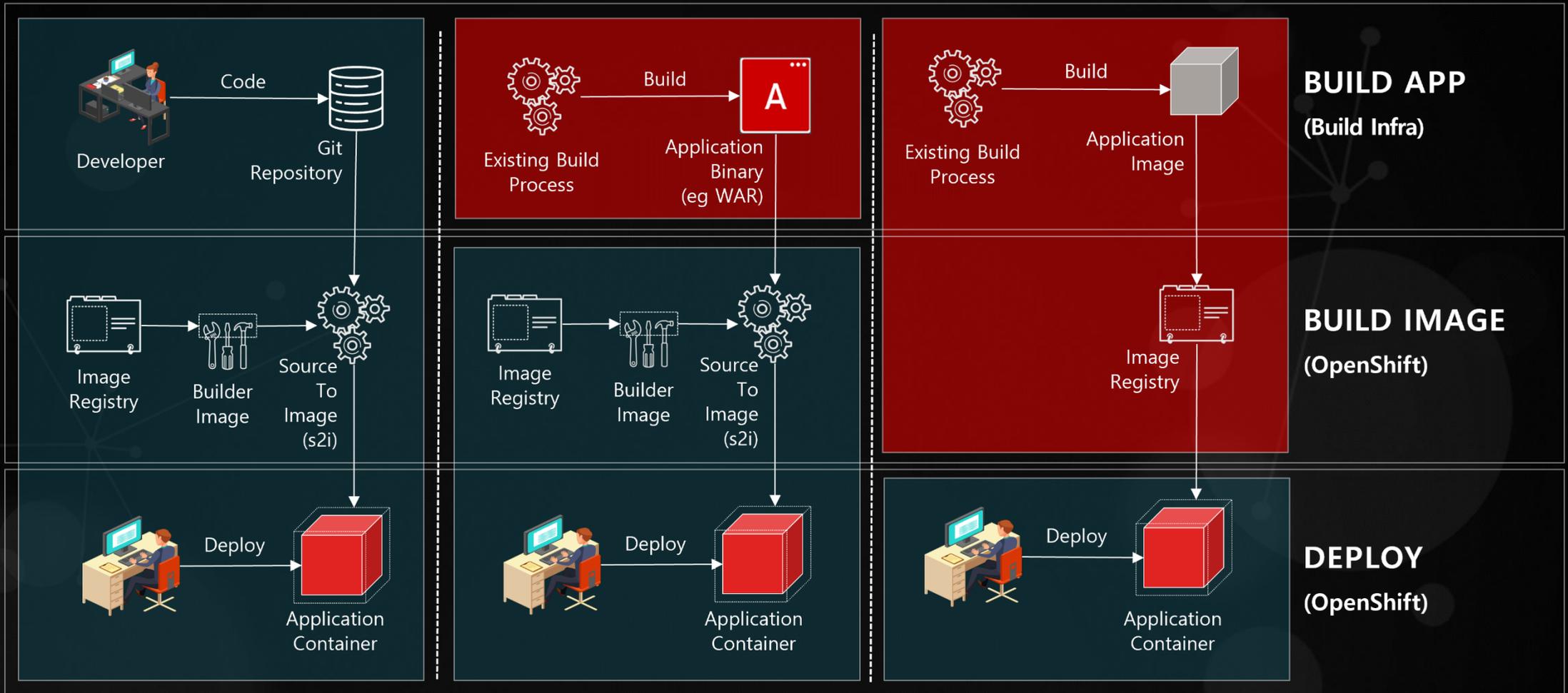
- Git : Git 저장소에서 응용 프로그램을 빌드 배포하는 경우
- Dockerfile : Dockerfile 에서 빌드를 완결시키는 경우
- Binary : 로컬 호스트의 Binary (EAR, WAR) 이미지에 배치하는 경우

- **Build outputs**

- ImageStreamTag : 컨테이너 레지스트리에 밀려, 지정된 이미지 스트림에 대해 태그를

Deploy in OpenShift

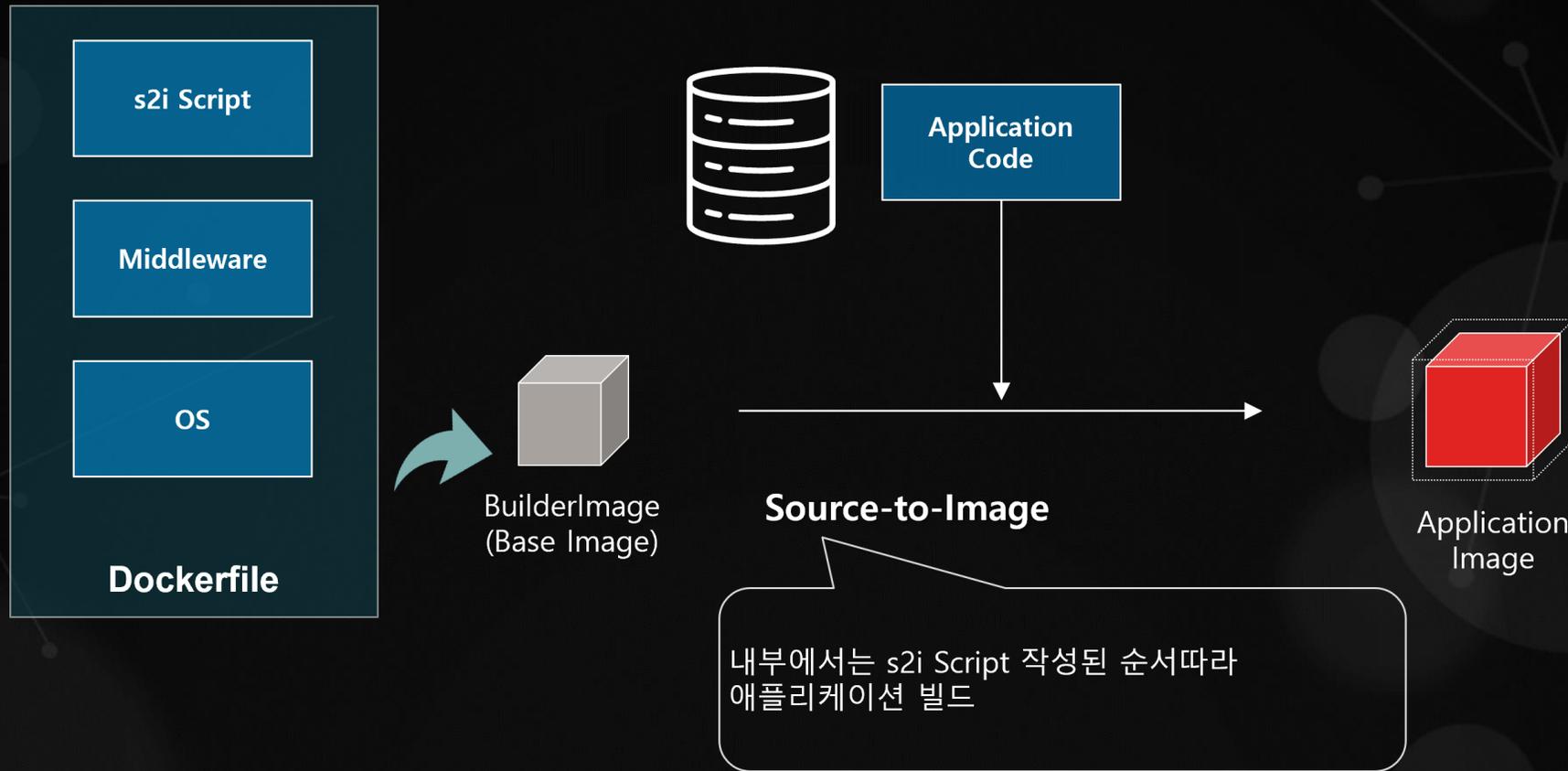
- ① Deploy Source Code with s2i
- ② Deploy App Binary with s2i
- ③ Deploy Container Images



 User / Tool Does  OpenShift Does
- Confidential -

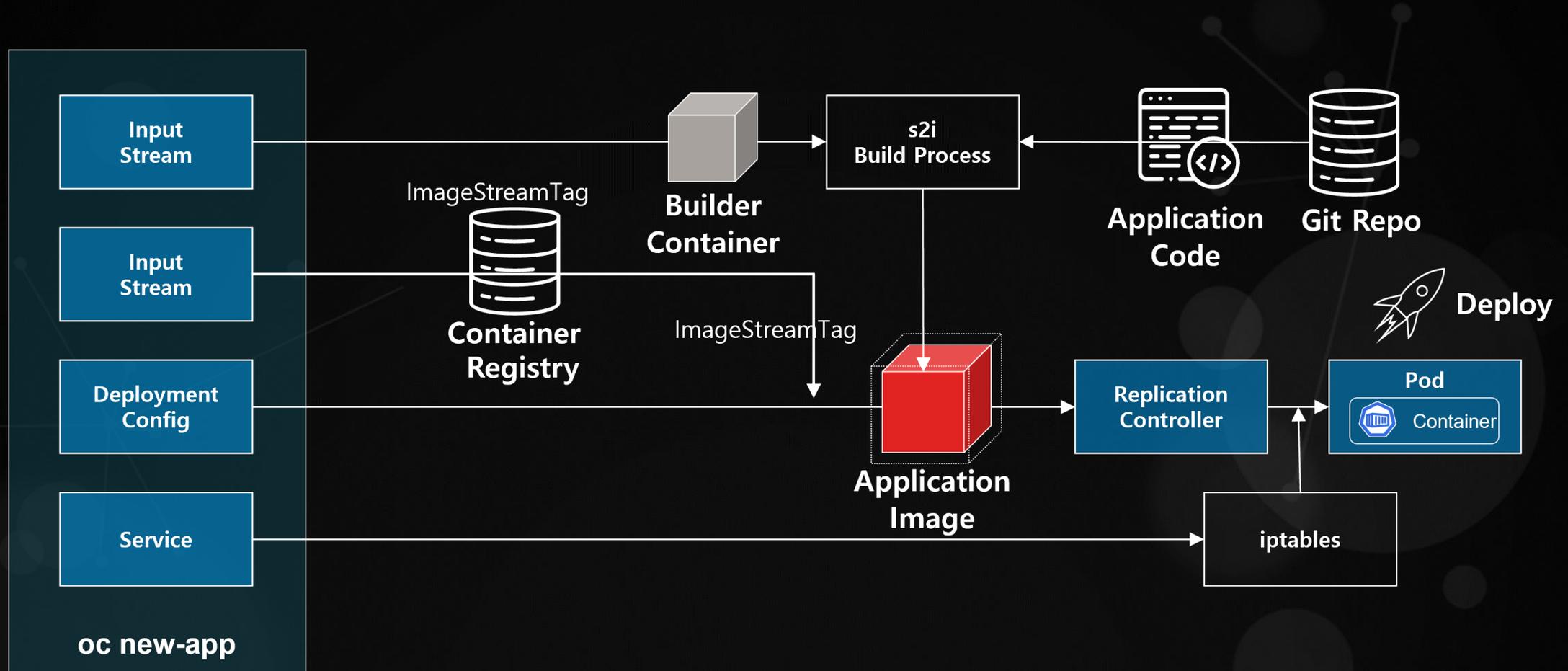
Source-to-Image (s2i)

- s2i 은 BuilderImage 레이블에서 s2i Script 위치를 로드하고 그 결과를 컨테이너 이미지로 빌드하는 방식



Source-to-Image (s2i)

- 애플리케이션 코드를 Input 하고 애플리케이션과 기본 이미지를 빌드하여 새 Docker 이미지를 생성



배포 편의성

BuildConfig jenkinsPipelineStrategy

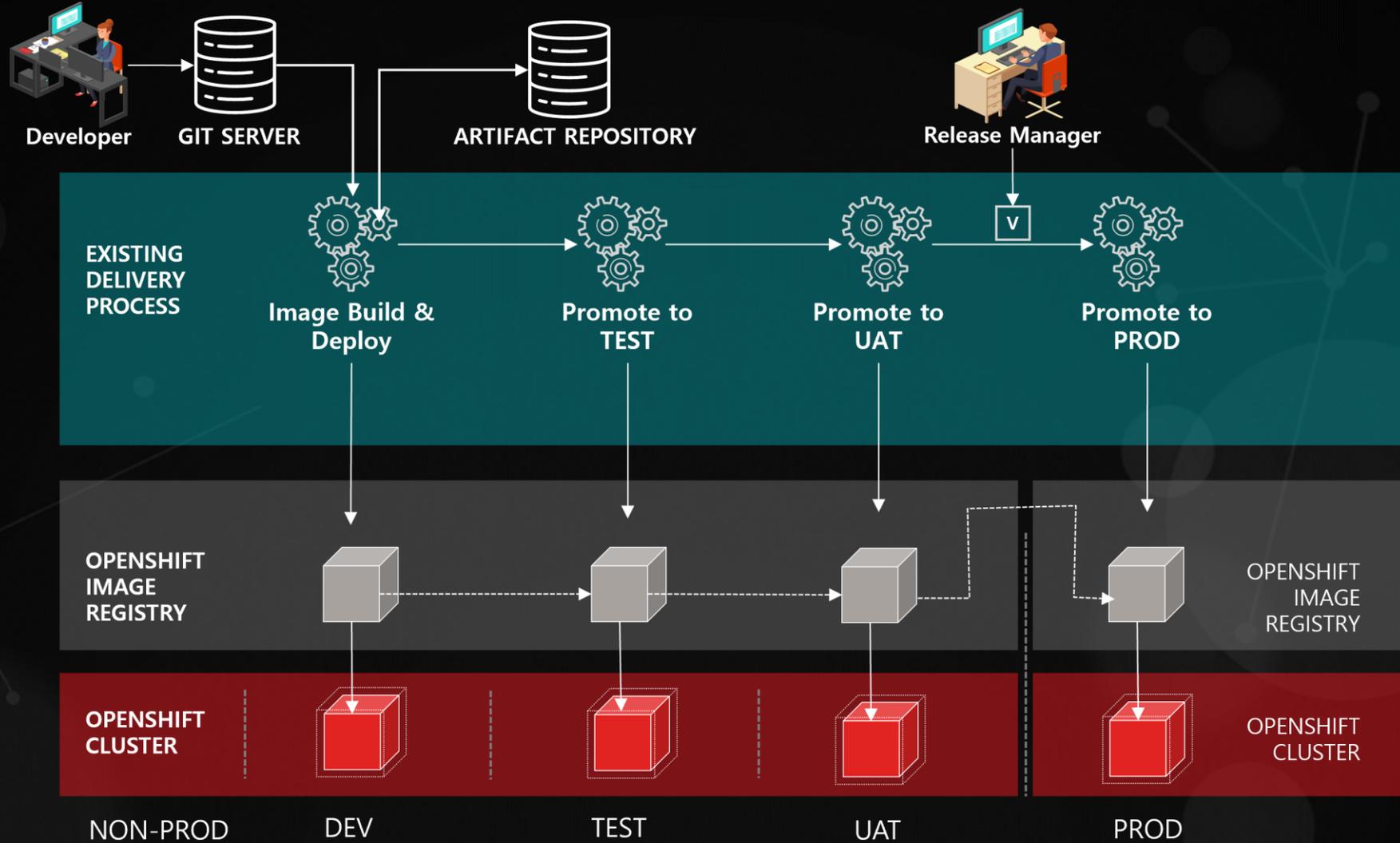
- OpenShift Pipelines 를 사용하면 Jenkins 파이프 라인을 통해 CI / CD 워크플로우를 정의
- 파이프 라인은 시작, 모니터링 및 관리 지원
 - Jenkins 슬레이브의 동적 프로비저닝
 - Jenkins 서버 • 動 프로비저닝
 - 포함 Jenkinsfile
 - Git 저장소에서 Jenkinsfile

```
apiVersion : v1 kind :
BuildConfig metadata :

name : app-pipeline spec :
strategy :
  type : JenkinsPipeline je
  nkinsPipelineStrategy :
    jenkinsfile : | -
      node ( 'maven') { stag
        e ( 'build app') {
          git url : 'https : //git/app.git'sh 'mvn package "
        } stage ( 'build image') {
          sh "oc start-build app --from-file = target /app.jar} stage ( 'deploy'){
            openshiftDeploy deploymentConfig : 'app'}}

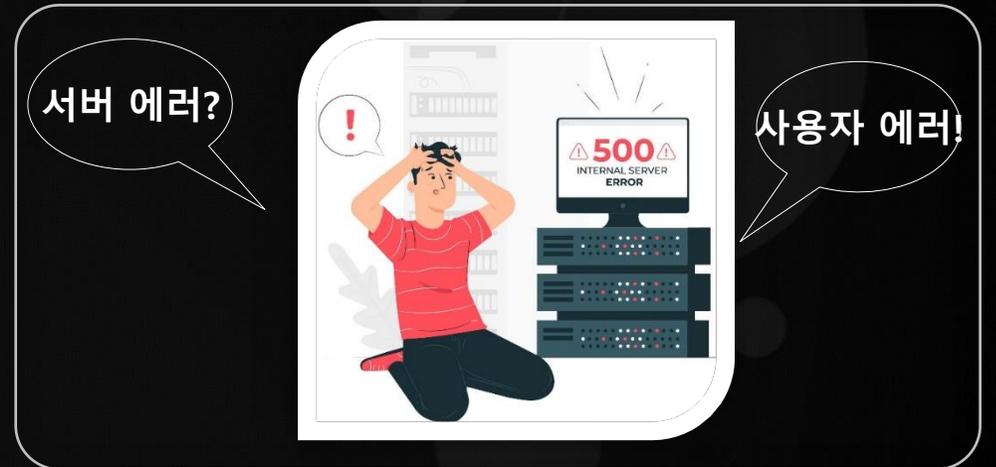
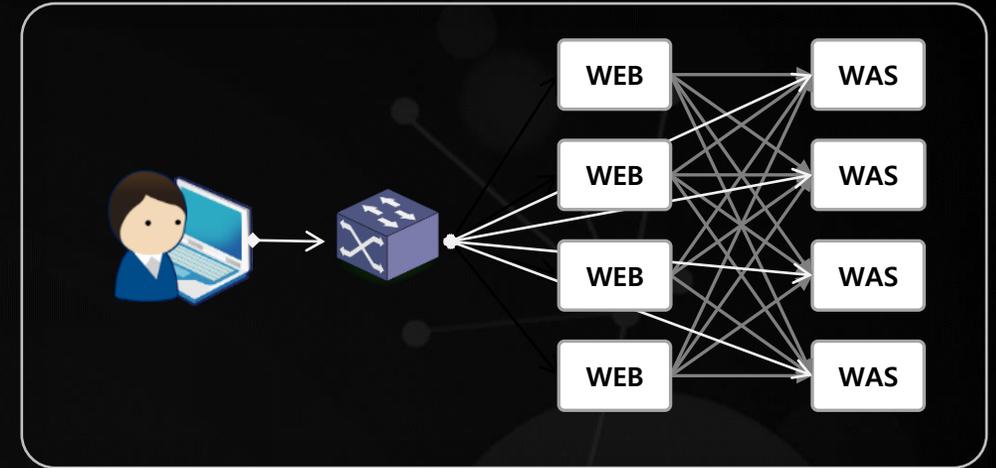
Provision a Jenkins sla
ve for running
Maven
```

Continuous Delivery Pipeline

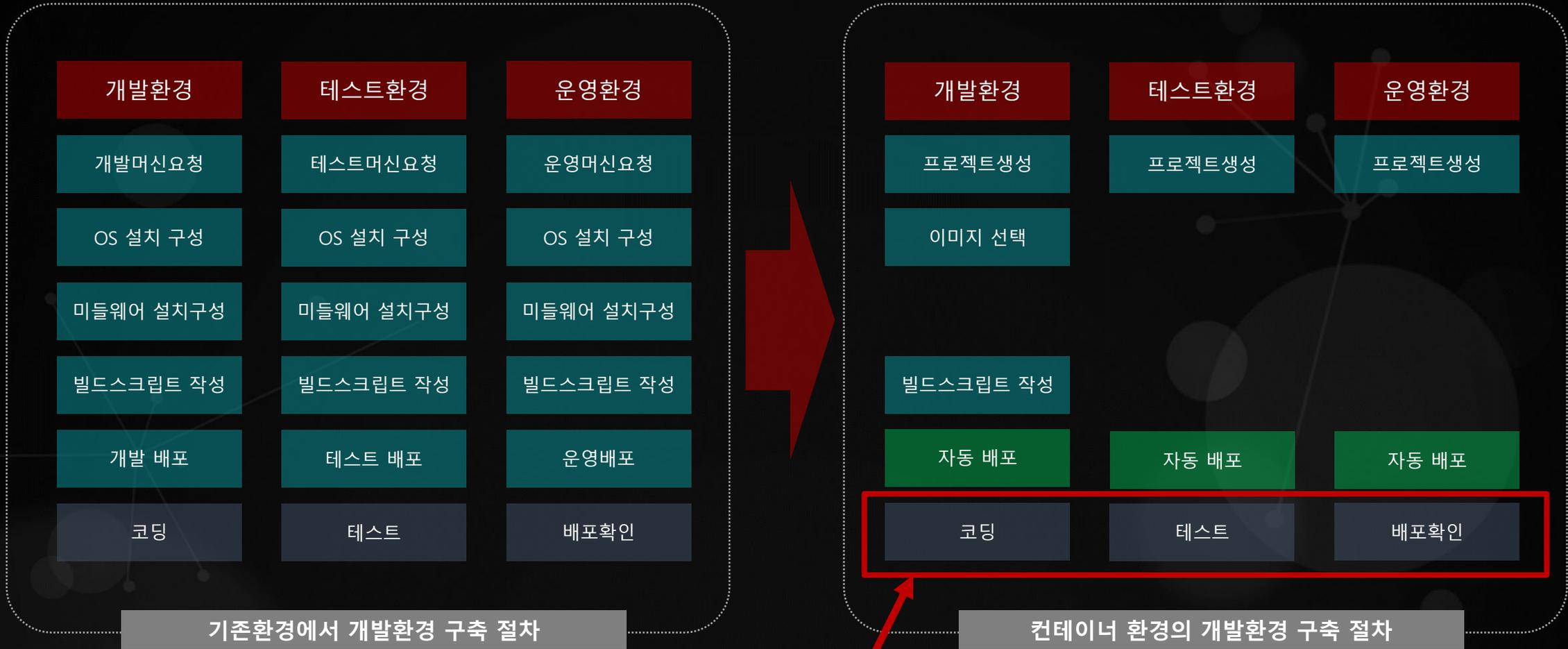


웹시스템의 요구사항과 특징

- Scale-out 형 인프라로 계층 형 아키텍처
- 가상화와 클라우드 환경에 적합하며 인스턴스 개수가 많으며 노드 간 연결성이 높음
- 장애 발생 시 신속한 장애 복구와 재확보가 중요하며 신속한 장애 복구와 재확보가 중요하며 신속한 장애 복구와 재확보가 중요
- 시스템 운영 이슈
 - 시스템 환경의 불일치(Dev/Stage/Prod, 서버 별)
 - 긴 배포 시간
 - 수 작업으로 인한 Human Errors
 - IT Agility 부족으로 인한 운영팀 축소
 - 개발팀에서 직접 IT 인프라 운영
 - 문제점의 발견과 조치에 많은 시간 소요



컨테이너 기반의 빠른 표준 개발 환경 구축



개발자의 핵심업무에 집중



제품 / 서비스에 관한 문의

- 콜 센터 : 02-469-5426 (휴대폰 : 010-2243-3394)
- 전자 메일 : sales@openmaru.com