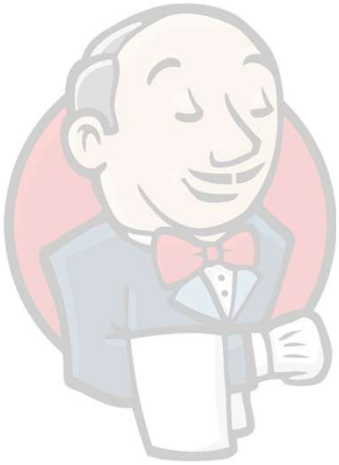
Jenkins 와 CI/CD
Jenkins 를 활용한 실무CI/CD

Jenkins 실습

빌드, 테스트, 배포

김대현
CTO – Anyware co, Ltd.
(anyware1001@gmail.com)

2 Extreme Stories



- CI/CD 란 무엇인가
- Jenkins 의 기본 개념과 동작방식
- 개발환경 및 CI/CD의 기본동작 이해
- CI/CD 파이프라인 구축 및 QnA

CI/CD 란

- **Continuous Integration => 뭘 통합한다는 거야?**
  - 여러 개발자들의 코드베이스를 계속해서 통합하는 것.
  - continuous integration (CI) is the practice of merging all developers' working copies to a shared mainline several times a day
- **Continuous Delivery => 무엇을 배달한다는 거야?**
  - 사용자에게 제품/서비스를 지속적으로 배달한다!
  - 코드베이스가 항상 배포가능한 상태를 유지하는 것.
- **Continuous Deployment**
  - 코드베이스를 사용자가 사용가능한 환경에 배포하는 것을 자동화.

- CI/CD 란 각각의 개발자들이 개발을 하는 개발환경을 사용자가 사용가능한 서비스로 전달하는 모든 과정을 지속가능한 형태로 또 가능하다면 자동으로해서 개발자와 사용자사이의 격차를 없애는 것이다!
- 이러한 과정에는 코드를 빌드하고, 테스트하고 배포하는 활동이 있다.

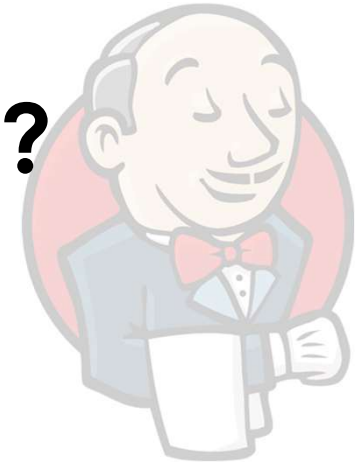
CI 왜 필요한가요?

- 10명의 개발자가 열심히 개발
- 1 Week Later....
- Merge Hell....

- 10명의 개발자가 열심히 개발
- 1 Week Later....
- 마지막 커밋 누구야 내꺼 안되잖아!

- **개발자들이 안심하고 개발을 하기위해, 내 코드와 멘탈의 평안을 위하여....**

CI와 함께라면?



- 10명의 개발자가 열심히 개발
- 커밋!
- 로컬테스트 통과 실패..
- 커밋!
- 코드베이스 머지
- 만족!

가능한 최대한 많이 빨리 내코드를 코드베이스에 안착!

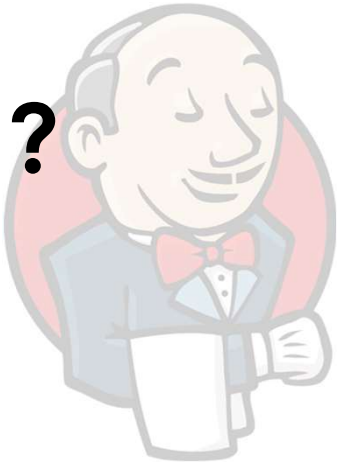
테스트 코드가 없는 코드,
버그덩어리 코드를
코드베이스에서 쫓아내자

CD 왜 필요한가요?

- 백엔드 코드 개발
- 프론트와 협업해야 하니 배포를 해볼까?
- 저기 배포 좀 해주세요...
- 앗 버그...! 다시 배포 좀 해주세요...
- 데브 서버에 누가 배포했나요? 제꺼 안되는데요;;
- 앗 죄송...
- QA배포...
- 프로덕션 배포시 초긴장 유지...
- 열심히 배포스크립트 작성, AWS콘솔 만지작.. 혹은 베어메탈?!

- **개발자가코드만짜면되지뭐이리 할게많아ㅜㅜ**

CD와 함께라면?

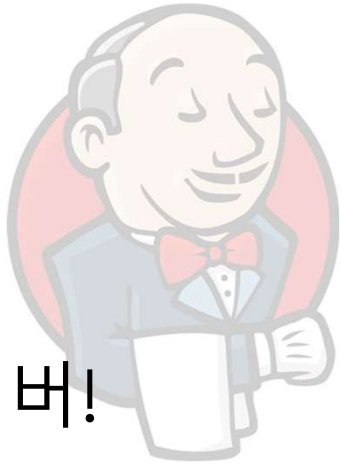


- 10명의 개발자가열심히개발
- 끝. (머지됐으니 내 역할은 여기까지...)

QA 엔지니어와 같은 내부사용자 혹은 실제 production 환경의 사용자에게

지속적이고 안정적으로 서비스를 제공한다.

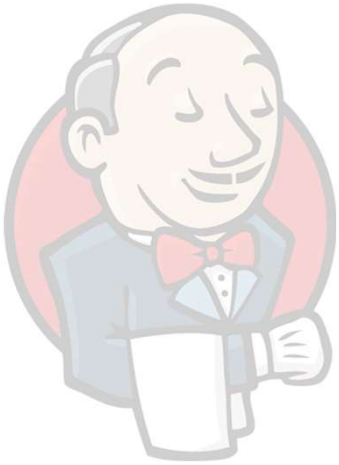
Jenkins



- 빌드, 테스트, 배포 등 모든 것을 자동화해주는 서버!
- 난 개발만~, 귀찮은 건 네가 다해!

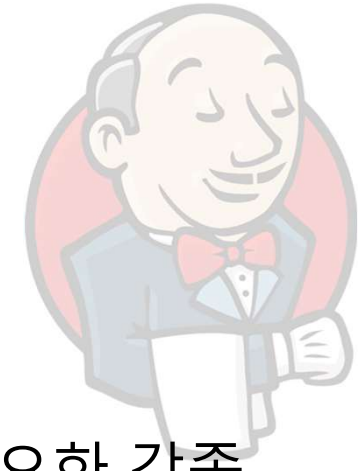
- Java Runtime Environment 에서 동작
- 다양한 플러그인들을 활용해서 각종 자동화 작업을 처리할 수 있음
- 일련의 자동화 작업의 순서들의 집합인 Pipeline을 통해 CI/CD 파이프라인을 구축함

Plugin – 정말 많은 Plogin



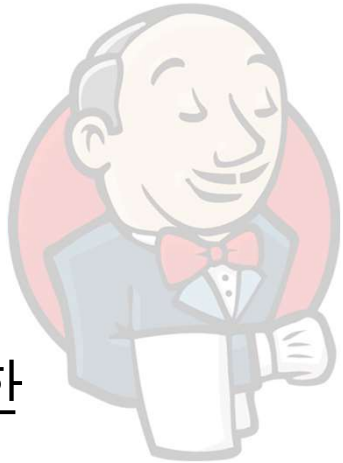
- 정말 많은 플러그인들 존재
- 대표적인 플러그인
  - Credentials Plugin
  - Gid Plugin
  - Pipeline (핵심 기능인 파이프라인도)
- 그냥 Recommend하면 다 설치됨!

Plugin 살펴보기



- **Credentials Plugin**
  - Jenkins 는 그냥 단지 서버일 뿐이기 때문에 배포에 필요한 각종 리소스
    - 예를 들면, 클라우드 리소스, 등에 접근하기 위해서는 여러가지 중요 정보들(AWS token, Git access token, etc…)을 저장하고 있어야 한다.
  - 이런 중요한 정보를 저장하는 플러그인
- **Pipeline Plugin**
  - 젠킨스의 핵심기능인 Pipeline 을 관리할 수 있게 해주는 플러그인
- **Docker plugin and Docker Pipeline**
  - Docker agent 및 도커 컨테이너를 사용하기 위함

Pipeline

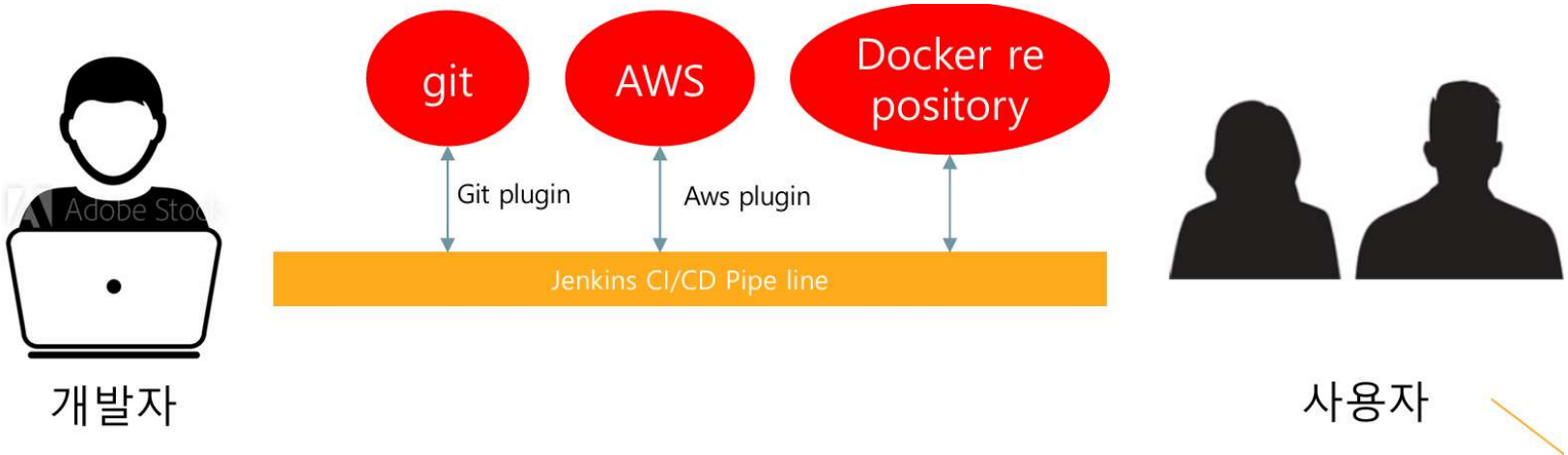


파이프라인이란 CI/CD 파이프라인을 젠킨스에 구현하기 위한 일련의 플러그인들의 집합이자 구성.

여러 플러그인들을 이 파이프라인에서 용도에 맞게 사용하고 정의함으로써 파이프라인을 통해 서비스가 배포됨

Pipeline DSL(Domain Specific Language) 로 작성

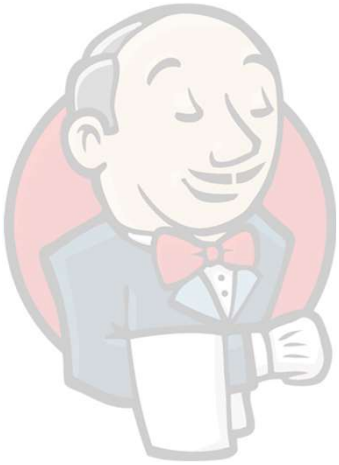
두가지형태의Pipeline syntax가 존재 (Declarative, Scripted Pipeline)



Pipeline Syntax

- Declarative Pipeline syntax
- Sections
  - Agent section
  - Post section
  - Stages section
  - Steps section

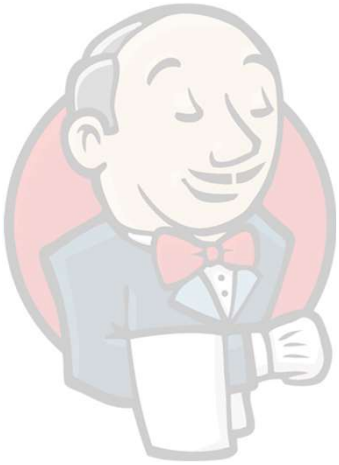
Pipeline Syntax



- **Declaratives**

  Environment, stage, options, parameters, triggers, when 등
  - Environment : pipeline 이나 stage scope 의 환경 변수 설정
  - Parameter : 파이프라인 실행 시 파라미터 지정
  - Triggers : 어떤 형태로 실행할지 지정
    - (cron, pollSCM, upstream)
  - When : 언제 실행되는지 지정

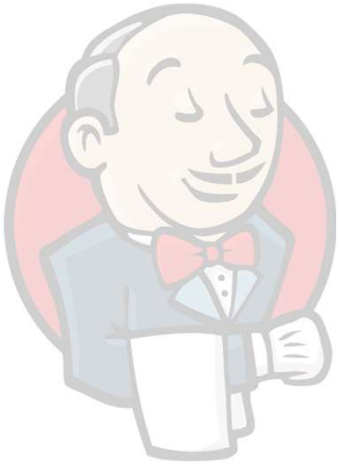
```
environment {
  AWS_ACCESS_KEY_ID = credentials('awsAccessKeyId')
  AWS_SECRET_ACCESS_KEY = credentials('awsSecretAccessKey')
  AWS_DEFAULT_REGION = 'ap-northeast-2'
  HOME = '.' // Avoid npm root owned
}
```



```
triggers {
    pollSCM('*/3 * * * *')
}
```

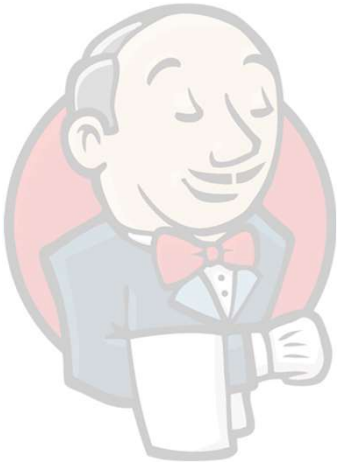
```
stage('Only for production') {
    when {
        branch 'production'
        environment name: 'APP_ENV', value: 'prod'
        anyOf {
            environment name: 'DEPLOY_TO', value: 'production'
            environment name: 'DEPLOY_TO', value: 'staging'
        }
    }

You, 4 minutes ago • Uncommitted changes
}
```

Pipeline Syntax



- **Agent Section**
  - 젠킨스는 많은 일들을 해야 하기 때문에 혼자하기 버겁다.
  - 여러 slave node 를 두고 일을 시킬수있는데, 이처럼 어떤 젠킨스가 일을 하게 할 것인지를 지정한다.
  - 젠킨스 노드 관리에서 새로 노드를 띄우거나 혹은 docker 이미지등을 통해서 처리할 수 있음

Pipeline Syntax



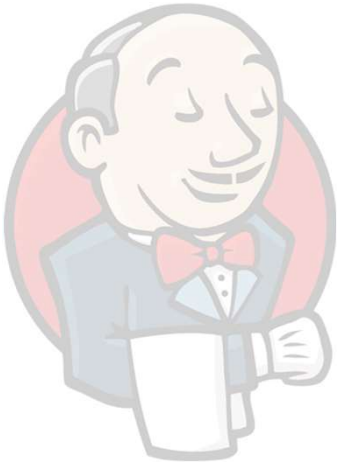
- **Post section**
  - 스테이지가 끝난 이후의 결과에 따라서 후속조치를 취할 수 있다.

    Ex) success, failure, always, cleanup

    Ex) 성공시에 성공 이메일, 실패하면 중단 혹은 건너뛰기 등등…

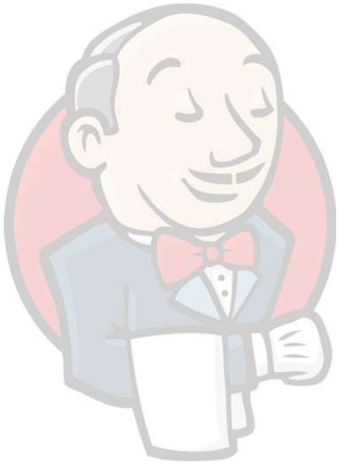
```
post {
    // If Maven was able to run the tests, even if some of the test
    // failed, record the test results and archive the jar file.
    success {
        echo 'success'
    }
}
```

Pipeline Syntax



- **Stages Section**
  - 어떤 일들을 처리할 건지 일련의 stage 를 정의함
- **Steps Section**
  - 한 스테이지 안에서의 단계로 일련의 스텝을 보여줌
  - Steps 내부는 여러가지 스텝들로 구성
    - 여러 작업들을 실행가능
    - 플러그인을 깔면 사용할 수 있는 스텝들이 생겨남
  - 플러그인별스텝종류들
    https://www.jenkins.io/doc/pipeline/steps/

Pipeline Syntax



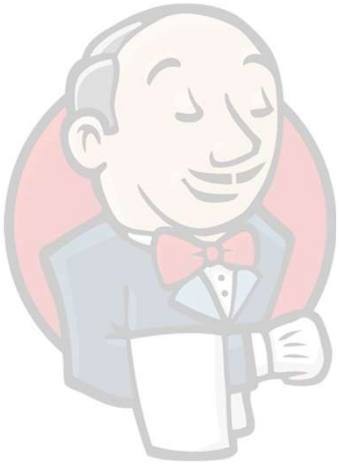
- Steps Section
  - Steps 내부는 여러가지 스텝들로 구성
  - 여러 작업들을 실행가능
  - 플러그인을 깔면 사용할 수 있는 스텝들이 생겨남

플러그인별스텝종류들

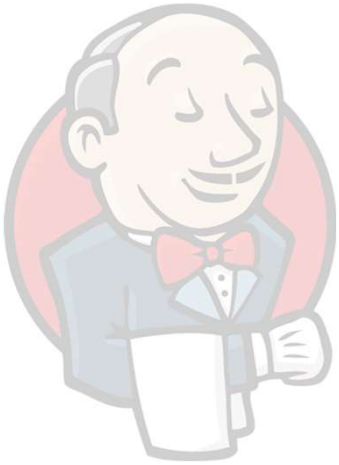
– https://www.jenkins.io/doc/pipeline/steps/

Jenkins User Guide

```
pipeline {
    agent any

    stages {
        stage('Prepare') {
            steps {
                git url: 'https://github.com/frontalnh/pet-insurance.git',
                    branch: 'master',
                    credentialsId: 'jenkinsgit'
                sh 'ls'
                dir ('./docs'){
                    sh '''
                    aws s3 sync ./ s3://namhoontest
                    '''
                }

            }

            post {
                // If Maven was able to run the tests, even if some of the test
                // failed, record the test results and archive the jar file.
                success {
                    echo 'success'
                }
            }
        }

        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
    }
}
```



개발 환경의 종류

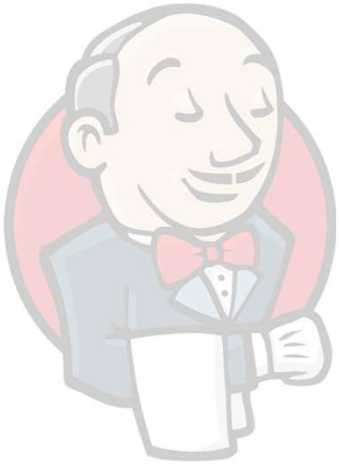


- 개발자 **Local 환경**(개발자 PC의 workspace)
- 개발자들끼리 개발 내용에 대한 통합테스트를 하는 **Development 환경**
- 개발이 끝난 뒤 QA엔지니어 및 내부사용자들이 사용해 보기위한 **QA환경**
- 실제 유저가 사용하는 **Production 환경**
- 쉽게 **DEV**, **QA**, **PROD** 로 부름

개발프로세스 예시

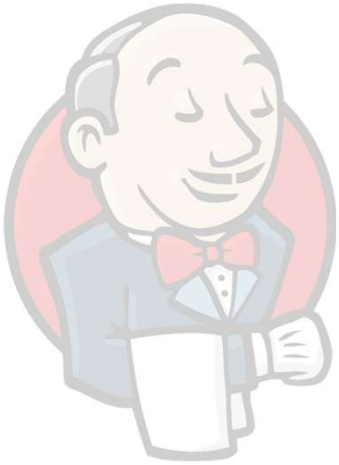
1. 개발자가 PC에서 개발한다.
2. 다른 개발자가 작성한 코드와 문제가 없는지 내부 테스트를 진행한다.
3. 테스트완료한 개발 내용을 다른 개발자들과 공유하기 위해 git과 같은 SCM에 올린다. (예. dev 브랜치)
4. dev 브랜치의 내용을 개발 환경에 배포하기 전에 통합 테스트와 Lint 등 코드 포맷팅을 한다.
5. 배포하기 위한 빌드 과정을 진행한다.
6. 코드를 배포한다.
7. 테스트를진행한다.
8. 위 모든 과정을 DEV, QA, PROD환경에서 모두 진행하고 각 환경에 맞게 배포한다.

배포 환경 관리



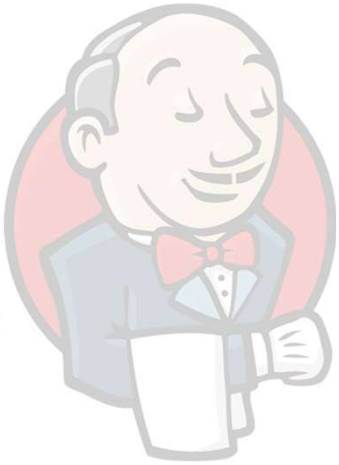
다양한 환경의 배포를 관리하는데 핵심은
-. 인프라를 모듈화
-. 어떤 것이 변수인지 설정
-. 이를 잘 설계하는 것!
예를 들면,
APP_ENV : 배포하는 서버의 환경과 앱에서 사용하는
다양한 변수를 설정하는 것이 핵심.
서비스 내부의 변수만 아니라 클라우드 리소스 내
인프라별 키관리가 매우 중요, AWS의 system
manager 의 parameter store 와 같은 키 관리
서비스를 쓰는 것이 좋음.

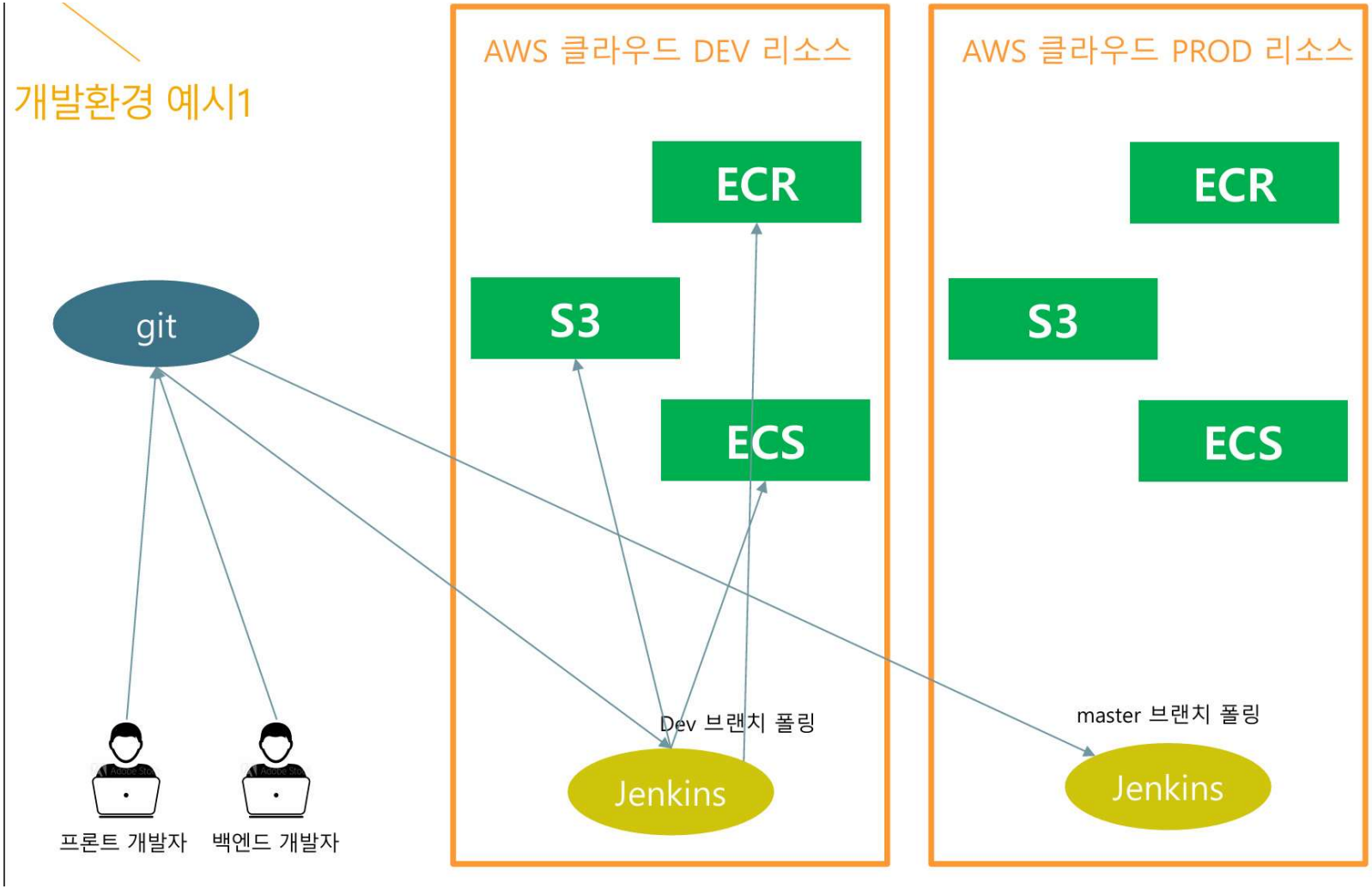
배포 환경 시나리오 예시



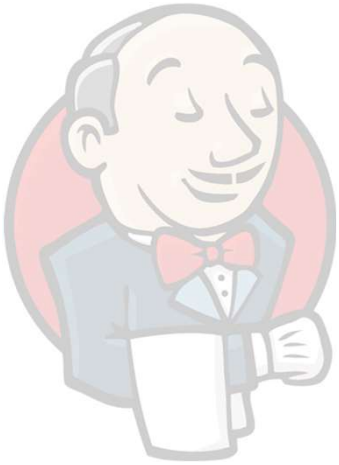
1. 웹사이트 코드를 작성
2. 웹사이트 코드를 린트, 웹팩 빌드해서 AWS S3 bucket에 html파일 업로드
3. Node.js 백엔드 코드를 typescript로 작성
4. 코드를 javascript compile하고, 테스트 코드를 돌려서 도커 이미지를 만들어 ECR 에 올림
5. 업로드한 ECR이미지로 ECS서비스를 재시작한다
    1. (rolling deploy) => continuouse deploy

개발환경 예시

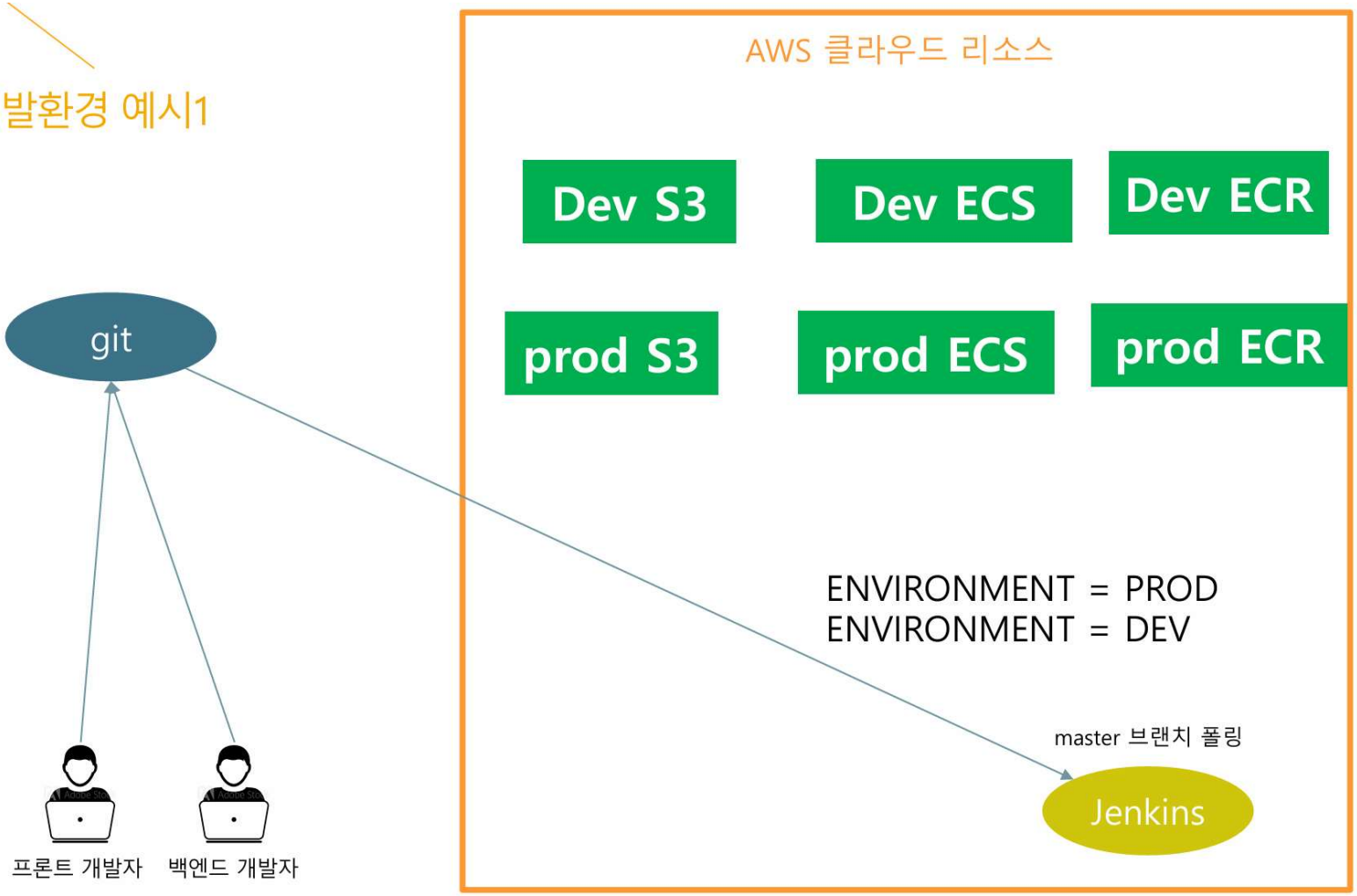



개발환경 예시1
AWS 클라우드 DEV 리소스
AWS 클라우드 PROD 리소스
ECR
ECR
S3
S3
ECS
ECS
git
Dev 브랜치 폴링
master 브랜치 폴링
Jenkins
Jenkins
프론트 개발자
백엔드 개발자

개발환경 예시



개발환경 예시1

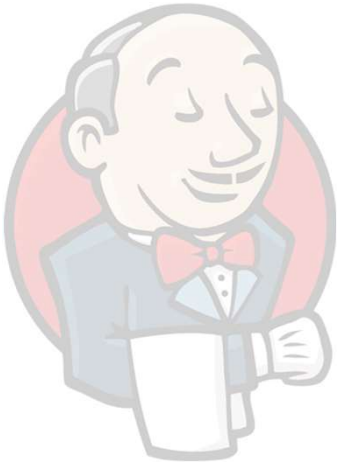


파이프라인 구축 실습 및 QnA

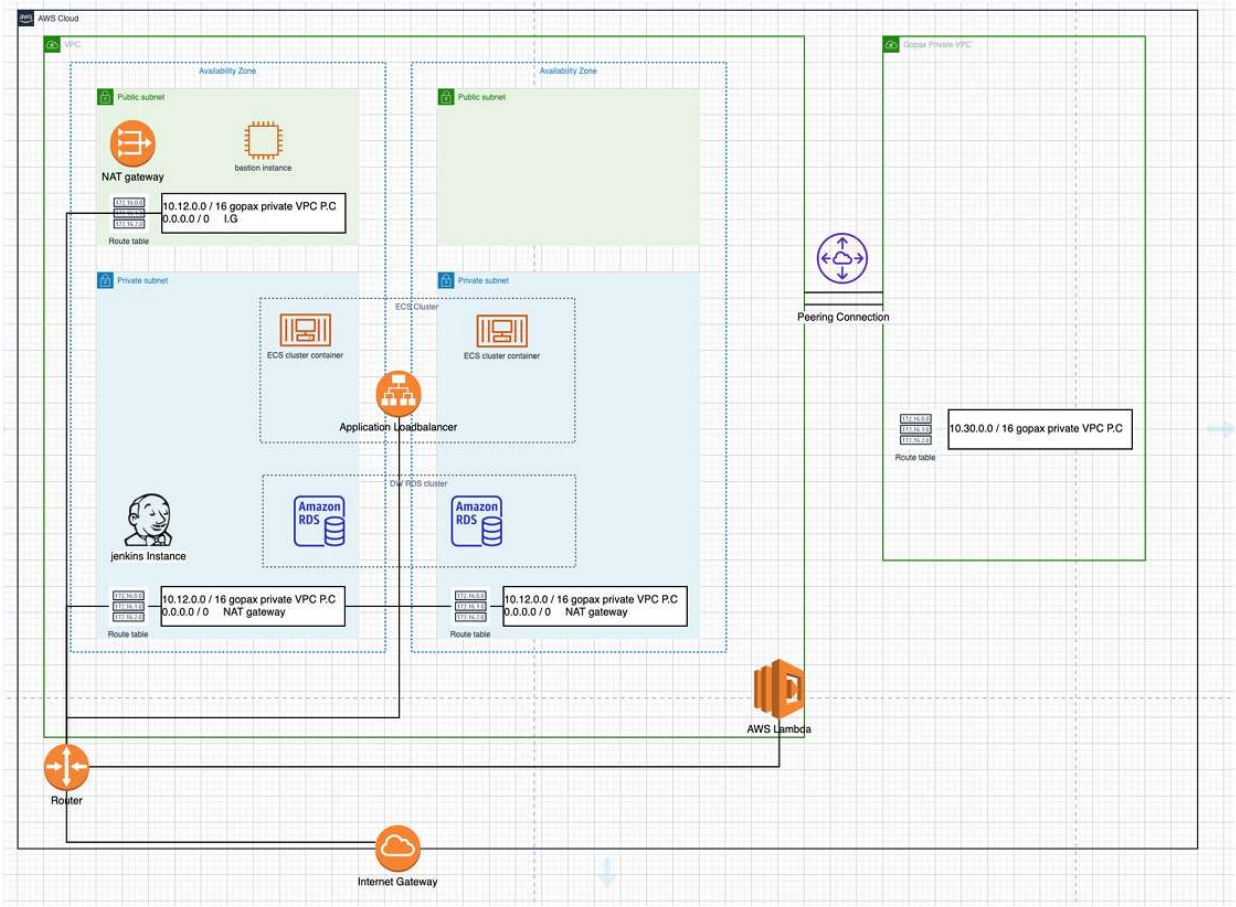
- 목표: git 에 코드가 머지되면 jenkins 에서 코드를 받아와 테스트와 빌드를 거쳐서 docker 로 서버를 띄우고, s3 에 웹사이트를 배포한다.

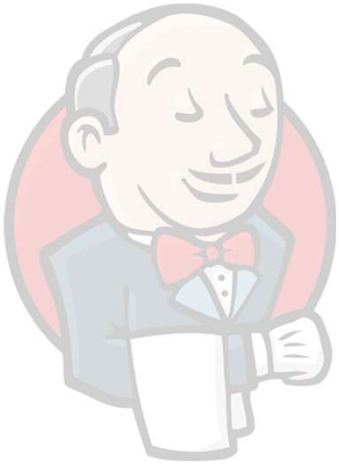
1. S3 버킷 만들기
2. HTML 파일 작성하기
3. AWS credential 발급하기
4. Jenkins 에 aws, git credential 등록하기
5. Pipeline 작성하기

파이프라인 구축 실습 및 QnA



ETL 파이프라인과 모든 리소스를 Cloud에 구축 했기 때문에 각종 클라우드 리소스까지 배포 자동화를 위해 IaaC 툴인 Terraform 사용 및
Dev, Qa, Prod 환경에서의 인프라 와 서비스 자동 배포 파이프라인
* 서버 등 운영 서비스뿐만 아니라 클라우드 리소스도 자동배포.





Thank You for Listening