Cloud DevOps Engineering

Module1 – DevOps 이해하기

David Yoon | CEO
david@2miles.co.kr

목차

1 클라우드 플랫폼의 이해 및 서비스 모델

1.A 클라우드 컴퓨팅의 이해

1.B 클라우드 플랫폼 별 서비스 모델 및 활용 방안

2 DevOps 이해 및 코드형 인프라 구성 방안

2.A DevOps의 이해 와 방식 및 도구

2.B 코드형 인프라 구성 방안

Chapter 1

클라우드 플랫폼의 이해 및 서비스 모델

클라우드 플랫폼의 이해및 서비스 모델

클라우드 컴퓨팅의 개념과 주요 서비스 모델에 대해 알아 보았습니다.

클라우드 컴퓨팅(Cloud Computing)의 개념:

네트워크를 통해 다양한 IT 기반 리소스와 어플리케이션을 가깝고, 손쉽고, 온디멘드 하게 제공을 하는 서비스

이며 크게 4가지 특징, 3가지 서비스모델, 3가지 구축모델로 구분 할 수 있다.

클라우드 컴퓨팅(Cloud Computing)의 이점과 고려사항:

종량제 비용 구조를 통해 자원 투자에 대한 부담이 적으며 즉각적인 IT 자원 할당을 통해 효율적으로 자원을 관리 가능 하다. 효율적인 클라우드 활용을 위해 보안 수준 확보, 가용성 확보, 제한 사항에 따른 요구사항 파악등의 고려사항들을 사전에 충분히 검토 해야한다.

클라우드 플랫폼별 서비스 모델(AWS, GCP):

AWS 는 전세계 18개의 지역, 49개 가용영역에서 다양한 서비스 블럭들을 제공 하고 있다. GCP는 글로벌 규모로 운영되는 자체 서비스들의 높은 트래픽을 다양한 서비스 블럭들로 감당 하고 있으며 특정 지역이 아닌 다중 지역에 위치되는 서비스도 존재 한다.

Chapter 2

DevOps 이해 및 코드형 인프라 구성 방안

DevOps 이해 및 코드형 인프라 구성 방안

2.A DevOps의 이해와 방식 및 도구

- DevOps 모델 소개
- DevOps 방식 및 도구

2.B 코드형 인프라(Infra as a Code) 구성 방안

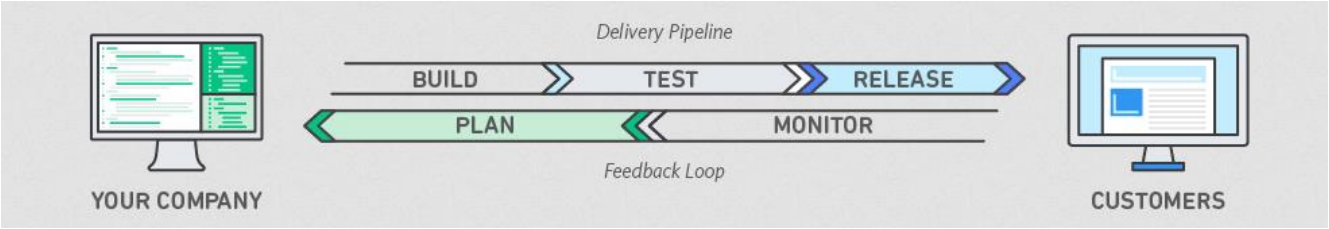
- 코드 형태의 인프라 구성 방안 및 도구
- Terraform 소개 및 구성 방법

DevOps 이해

DevOps 란

DevOps는 애플리케이션과 서비스를 빠른 속도로 제공할 수 있도록 조직의 역량을 향상시키는 문화 철학, 방식 및 도구의 조합이다. 소프트웨어 개발 사이클의 속도를 높여 제품을 더 빠르게 혁신하고 개선할 수 있으며 조직은 고객을 더 잘 지원하고 시장에서 효과적으로 경쟁할 수 있다. DevOps 움직임에는 CAMS 핵심 요소가 필요하다.

*CAMS: Culture, Automation, Measurement과 Sharing



DevOps 움직임의 핵심 가치

DevOps 움직임의 핵심 가치

CAMS

*출처: http://devopsdictionary.com/wiki/CAMS

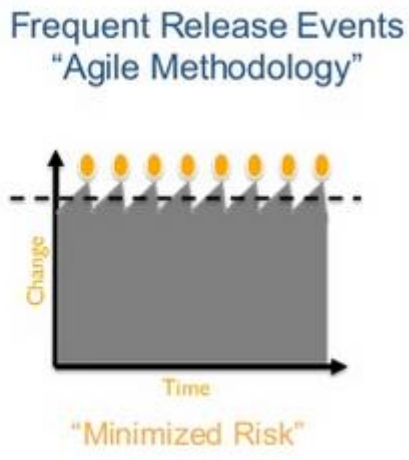
Culture	DevOps는 개발과 운영 두팀의 경계를 허물고 공통된 영역으로 모을 뿐만 아니라 협업 수준을 높여 품질 및 효율성을 향상 시킬수 있도록 문화와 환경을 제공 해야 함
Automation	자동화는 시간을 절약할 뿐만 아니라 작업자 실수를 예방하고 일관성을 유지 하여 추후 self service에도 활용 가능
Measurement	지속적인 개선과 데이터에 기반한 결정을 할 수 있도록 측정 능력 및 방안이 중요하며 데이터는 투명하고 엑세스 가능 해야하며 시각화를 할 수 있어야 함
Sharing	조직에서 DevOps를 성공하기 위해 조직 전반에 걸쳐 비슷한 요구사항을 도출하고 협력할 수 있는 공통된 도구를 활용하여 중복된 작업을 없애고 참여 의식을 갖도록 하여야 함

DevOps 모델

DevOps 방식

소프트웨어 개발과 인프라 관리 프로세스의 자동화 및 간소화를 통해 더 빠르게 혁신할 수 있도록 지원 해야 한다.

- 소규모 업데이트를 자주 수행 하여 혁신의 속도 향상
- 마이크로 서비스 아키텍처를 사용하여 애플리케이션의 유연성을 확보



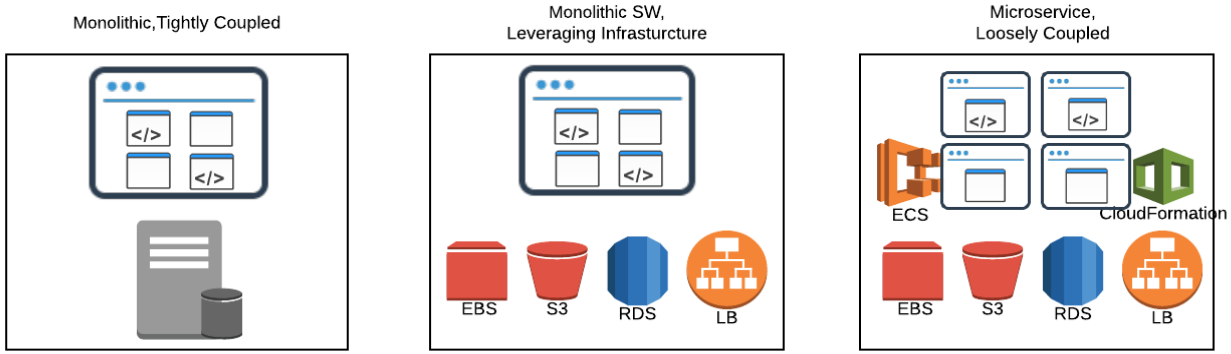
DevOps 모델

DevOps의 어플리케이션 의 변화

Monolithic 구조에서 Microservice 구조로 변화

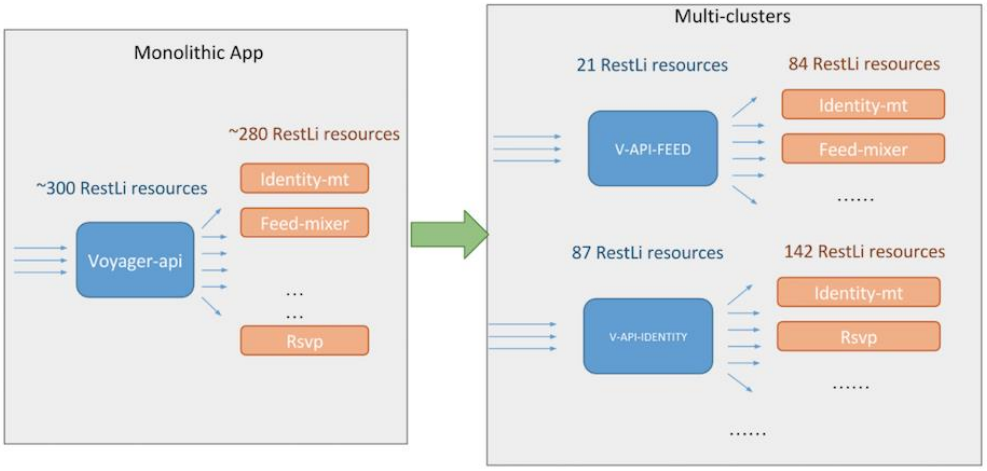
• Monolithic: 단일 리소스에 모든 기능을 구현하고 연동하여 간단히 서비스 구성이 가능하나 단일 형태로 구성이 되어 있고 각 기능이 서로 밀결합 되어 있어 서로의 종속성이 높아 재개발 혹은 장애에 취약하다.

• 마이크로 서비스: 기능별로 리소스가 분산되어 있으며 서로의 종속성이 낮아 업데이트 및 장애 발생시 전체 서비스에 심각한 영향을 주지 않으나 리소스의 파편화로 자동화 및 협업툴을 통하지 않으면 운영의 어려움이 발생할 가능성이 높다.



어플리케이션의 변화 사례 (Linkedin, 2017.11)

Monolithic to Multi-cluster Architecture



From Monolithic to Multicluster Architecture

*From Monolithic to Multi-cluster Architecture:
https://engineering.linkedin.com/blog/2017/11/improving-resiliency-and-stability-of-a-large-scale-api

DevOps 모델

DevOps 운영 이슈

Amazon refactored its software into small independent services and restructured its organization into small autonomous teams. Each team took on full ownership of the development and operation of a single service, and they worked directly with their customers to improve it. With this clear focus and control, the teams were able to quickly produce new features, but their deployment process soon became a bottleneck. Manual deployment steps slowed down releases and introduced bugs caused by human error. Many teams started to fully automate their deployments to fix this, but that was not as simple as it first appeared.

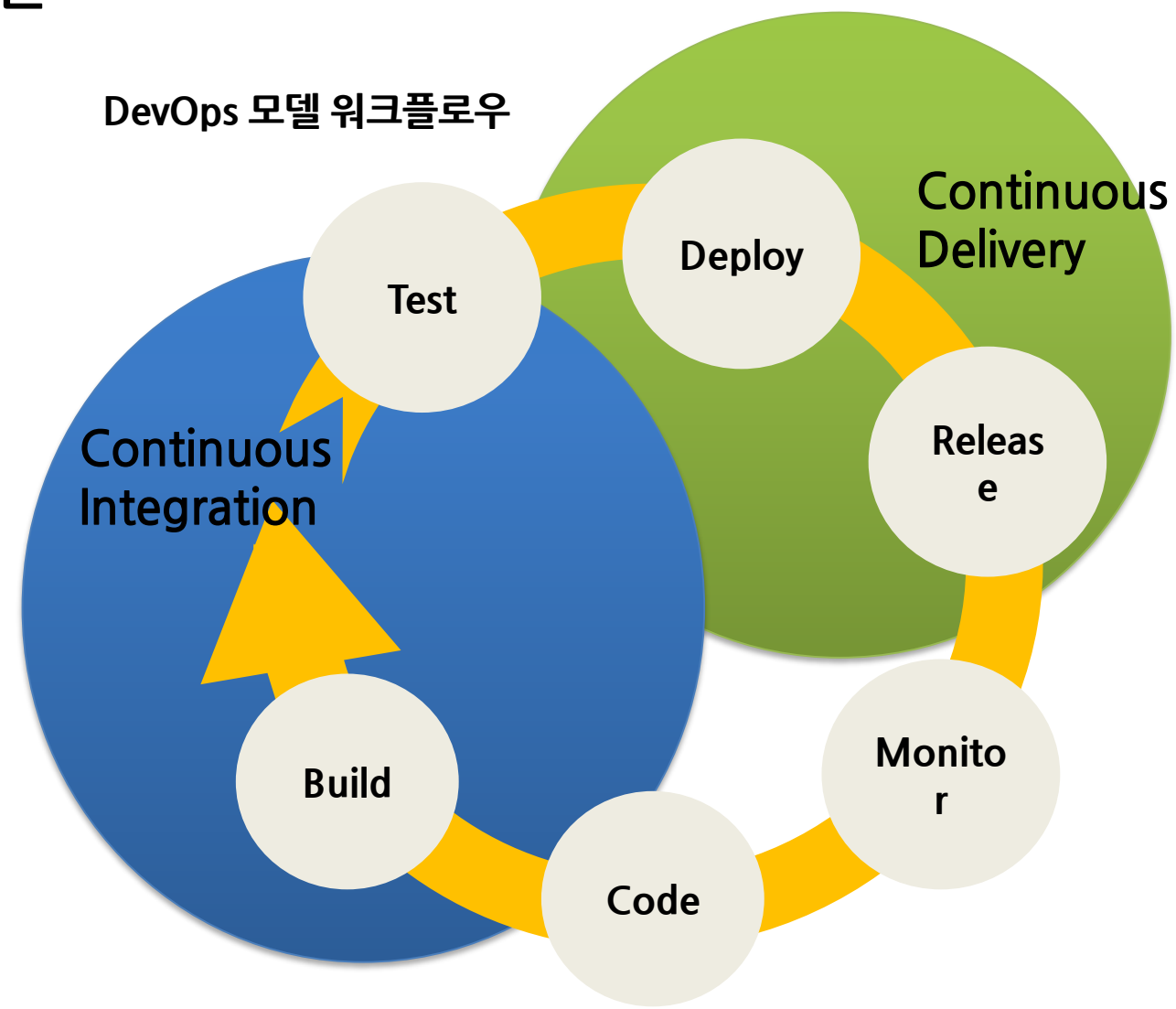
출처: Werner Vogels, Amazon.com CTO 블로그

DevOps 모델

DevOps 고려사항

다양한 변경에 대한 컴퓨팅 리소스의 탄력적 운영	코드 형태의 인프라 구성 및 형상 관리
빈번한 소프트웨어 빌드 및 릴리즈를 효과적으로 운영	지속적 통합, 전달 및 마이크로 서비스
인프라 및 어플리케이션의 성능 추척 및 대응	모니터링 및 로깅의 표준화, 고도화
중복 업무 발생 제거 및 협력 문화 강화	에자일, 협업 툴을 통한 업무의 협력 강화 및 투명화

DevOps 모델


DevOps 모델 워크플로우
Continuous Delivery
Deploy
Releas e
Monito r
Code
Build
Continuous Integration
Test

DevOps 모델

DevOps 모델 구성 요소(Code, Build)

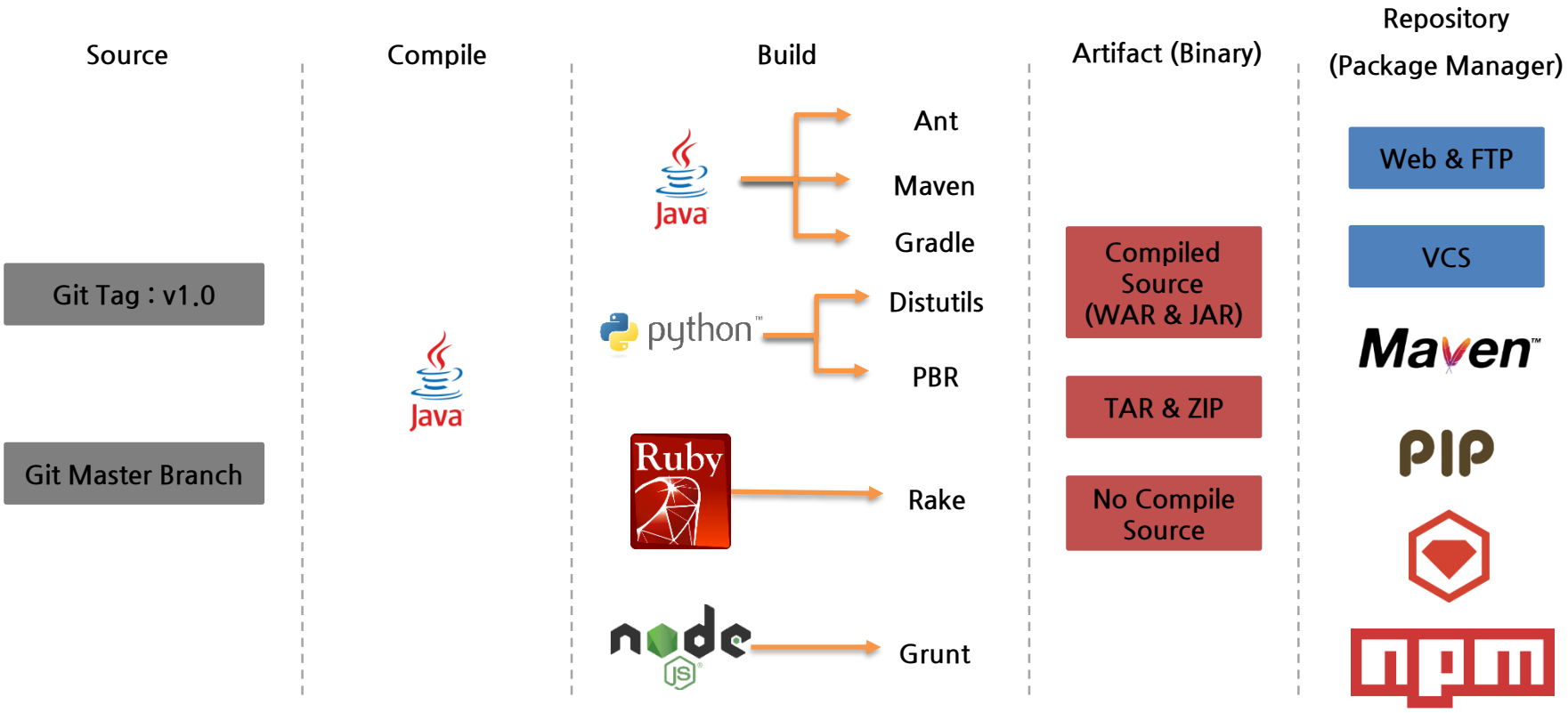
소스코드를 개발 하고 컴파일 및 코드를 배포 가능한 파일 (artifact) 구성 상태로 만드는 단계

Code	Git, Bitbucket
Build	Ant, Gradle, Maven

Code, Build

컴파일 및 빌드

- 컴파일(Compile) : 소스 코드를 분석해 기계어(binary Code)로 번역
- 빌드(Build) : binary Code와 Asset(Library, HTML, Image 등)들을 모아 실행 가능한 상태로 만드는 것
- 아티팩트(Artifact) : 배포 가능한 형태의 컴파일 또는 빌드 된 소스 코드
- 저장소(Repository) : 아티팩트(패키지) 들을 배포하기 위한 저장소



DevOps 모델

DevOps 모델 구성 요소(Test)

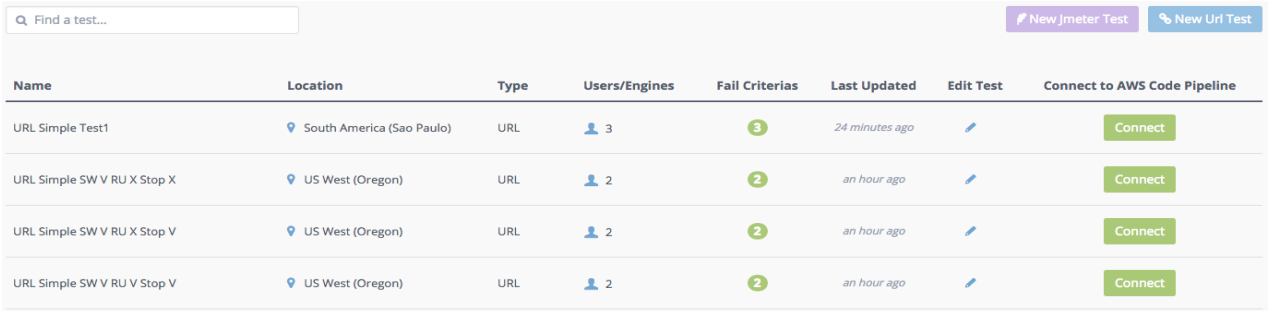
개발된 코드 기능을 테스트 또는 전체 서비스의 기능을 테스트 하는 단계

Test	Unit 테스트	각 기능별로 동작 유무를 확인할 수 있는 custom script
	Integration 테스트	Unit 테스트의 묶음 혹은 전체 서비스 동작 유무를 확인 할 수 있는 custom script
	Load 테스트	네트워크 레벨의 부하를 일으켜 평균 응답 시간 혹은 TPS(Transaction Per Second) 측정 Loadrunner, JMeter 활용

DevOps 모델

DevOps 모델 구성 요소(Test)

개발된 코드 기능을 테스트 또는 전체 서비스의 기능을 테스트 하는 단계



DevOps 모델

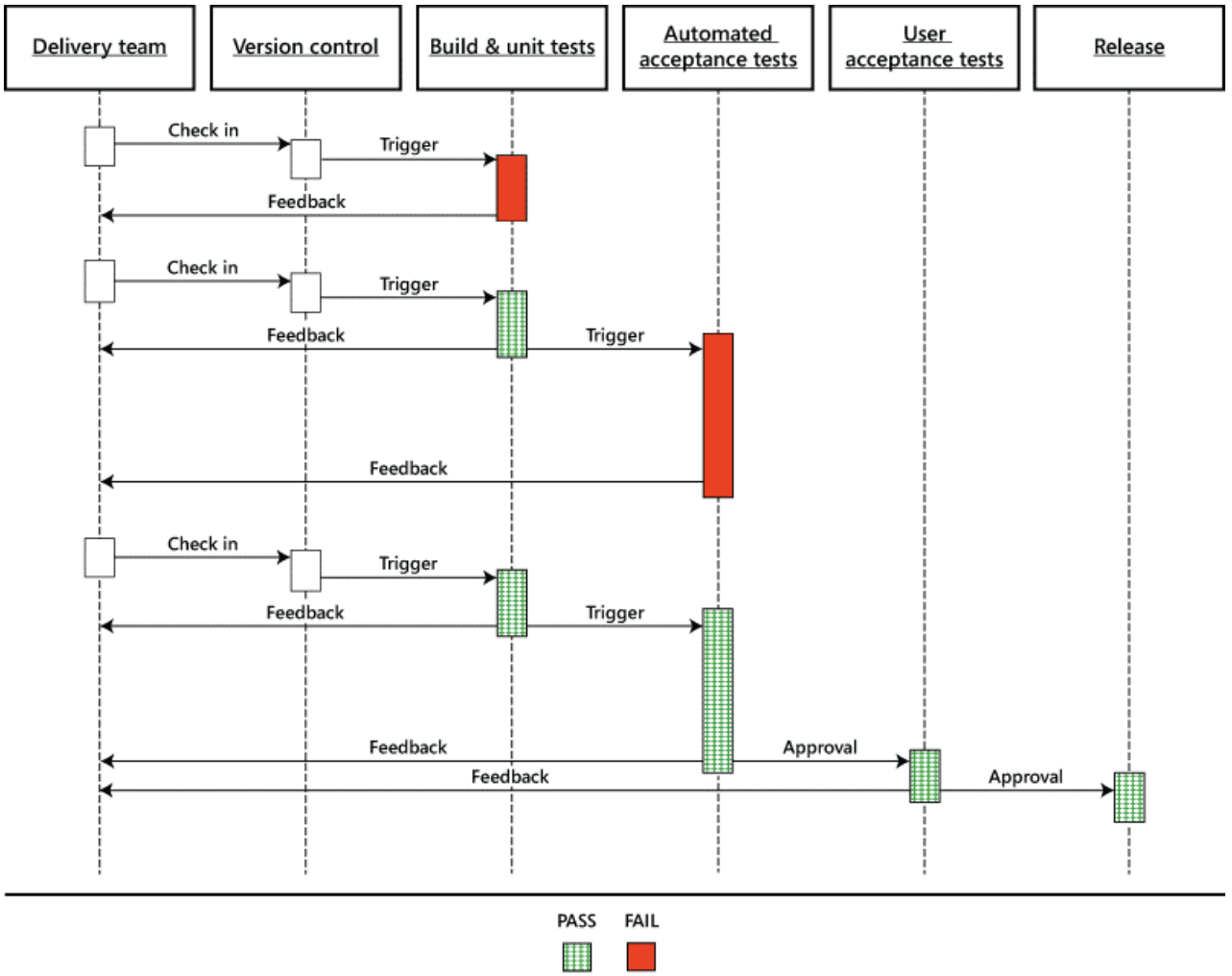
DevOps 모델 구성 요소(Deploy, Release)

배포 가능한 파일(Artifact)을 통해서 소스코드를 배포(deploy) 및
인프라 리소스에 대한 형상을 변경(생성, 삭제) 하여 서비스 투입(release) 하는 단계
* 두 단계를 통합해서 이야기 하기도 한다.

Deploy	리소스 생성 및 삭제	CloudFormation, Terraform
	리소스 형상관리	Ansible, Chef, SaltStack, Puppet
Release	Artifact 배포	Jenkins, Bamboo, GoCD

Code, Build, Deploy

코드 배포 파이프 라인



DevOps 모델

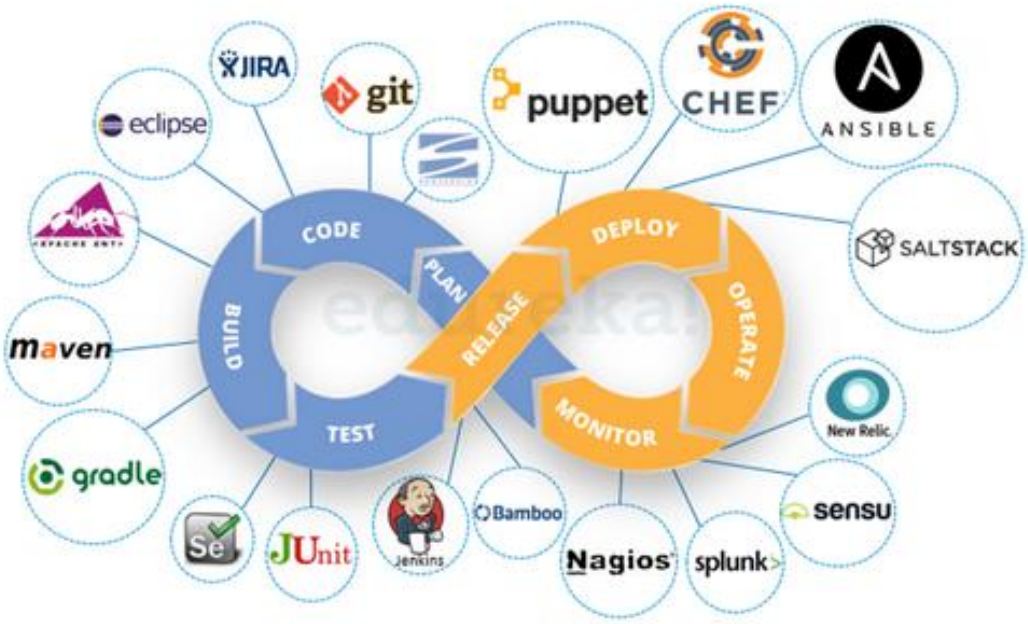
DevOps 모델 구성 요소(Monitor)

인프라, 어플리케이션의 시스템과 서비스 레벨의 지표와 로그를 모니터링 하여 개선 사항을 도출 하는 단계

Monitor	인프라 모니터링	Grafana, InfluxDB, Sensu, Nagios, Zabbix
	어플리케이션 모니터링(APM-Java, NodeJS)	Jennifer, Pinpoint, New Relic
	로깅	ElasticSearch, FluentD, LogD, Splunk

DevOps 모델

DevOps 모델 구성 요소와 도구

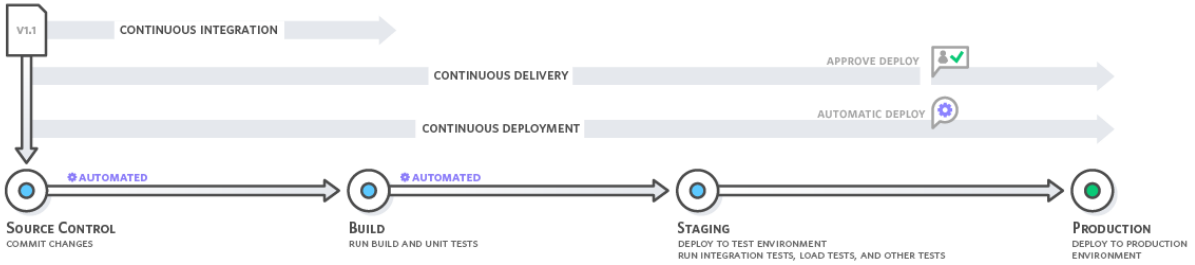


출처: Edureka Introduction DevOps 블로그

DevOps 모델

CI/CD

지속적 통합, 지속적 전달

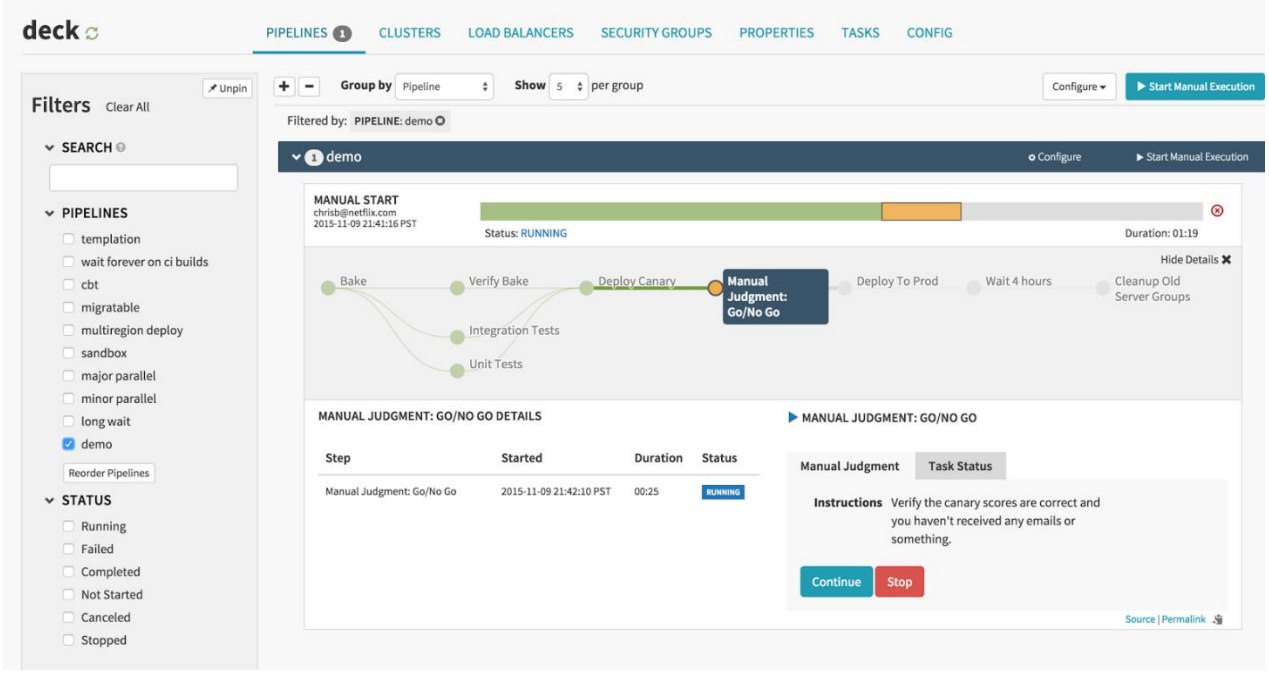


지속적 통합	빌드 및 테스트가 수행된 후 개발자가 코드 변경을 중앙 레포지토리에 정기적으로 통합하는 소프트웨어 개발 방식 이며 빌드/ 통합 단계를 일컫는다. 주요한 목적은 버그를 신속하게 찾아 해결 하고, 소프트웨어 품질을 높이고, 업데이트 검증 및 릴리즈 시간을 단축 시키는 것 이다.
지속적 전달	상용 환경으로 릴리즈 하기 위한 코드 변경이 자동으로 빌드, 테스트 및 준비 되는 소프트웨어 개발 방식이다. 빌드 단계 이후 모든 코드 변경을 개발 환경 및 상용환경에 배포 함으로써 지속적 통합을 확장한다
지속적 배포	전체적인 소프트웨어 릴리즈 절차가 자동화 되어 명시적인 승인 없이 자동으로 상용환경에 배포되는 소프트웨어 개발 방식

DevOps 모델

SPINNAKER 를 통한 CI/CD 활용

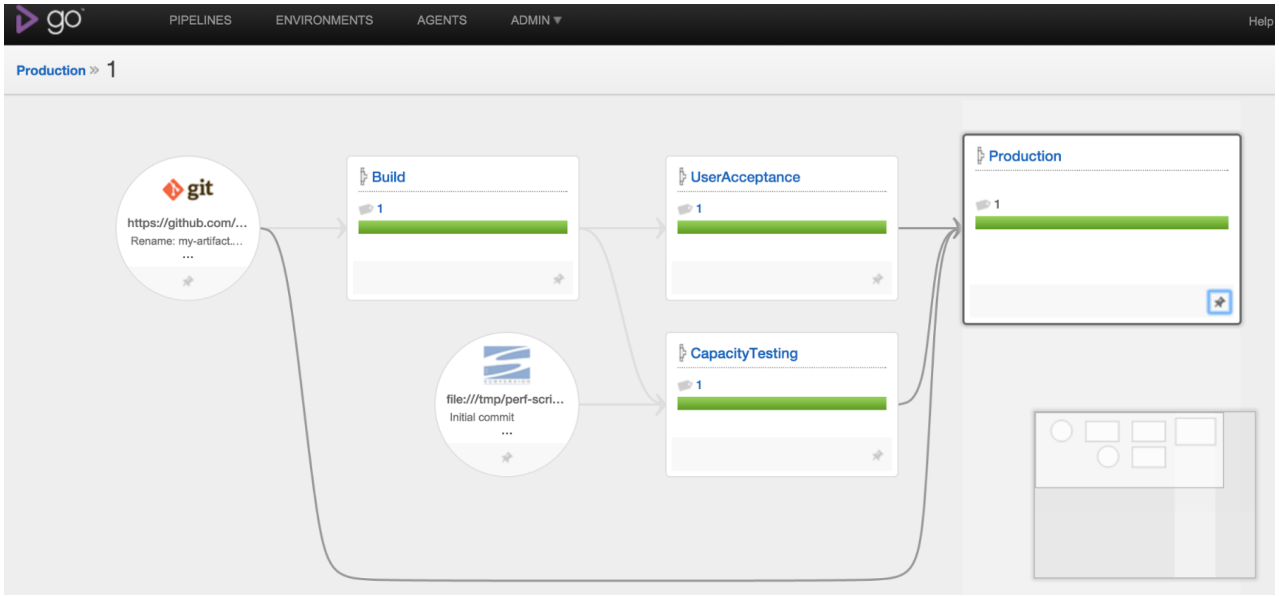
Spinnaker를 통한 CI/CD 활용 방법 및 QA, 승인 절차



DevOps 모델

goCD 를 통한 CI/CD 활용

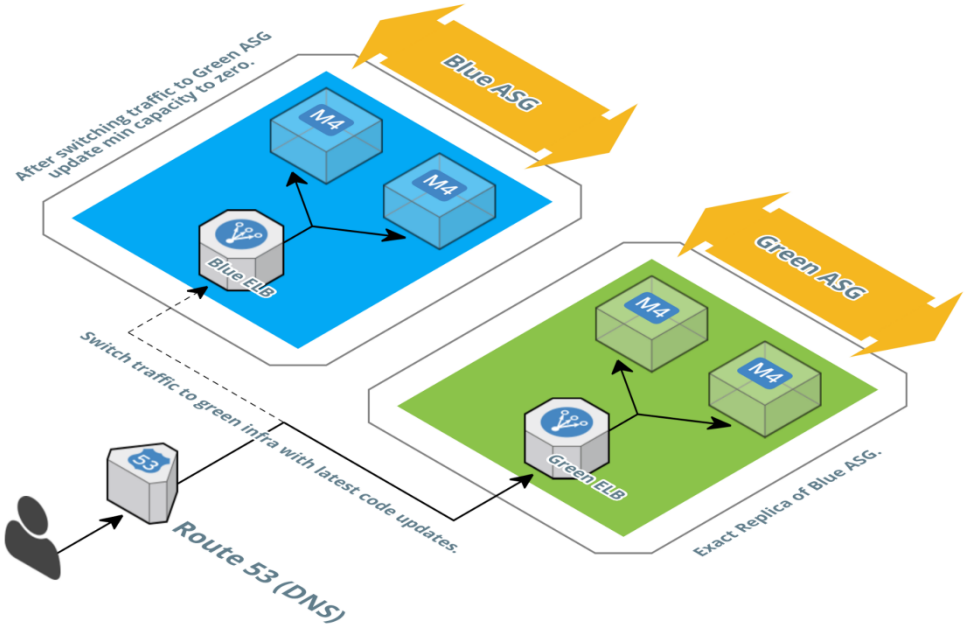
goCD를 통한 CI/CD 활용 방법 및 QA, 승인 절차



DevOps 모델

무중단 배포 전략

새로운 소스 코드 버전을 포함한 환경(Green)를 코드형 인프라 구성을 통해 손쉽게 인프라 배포후 앞단 도메인 혹은 네트워크 장비에서 기존 버전(Blue) 에서 새로운 환경으로 트레픽을 seamless 하게 서비스 업데이트



DevOps 모델

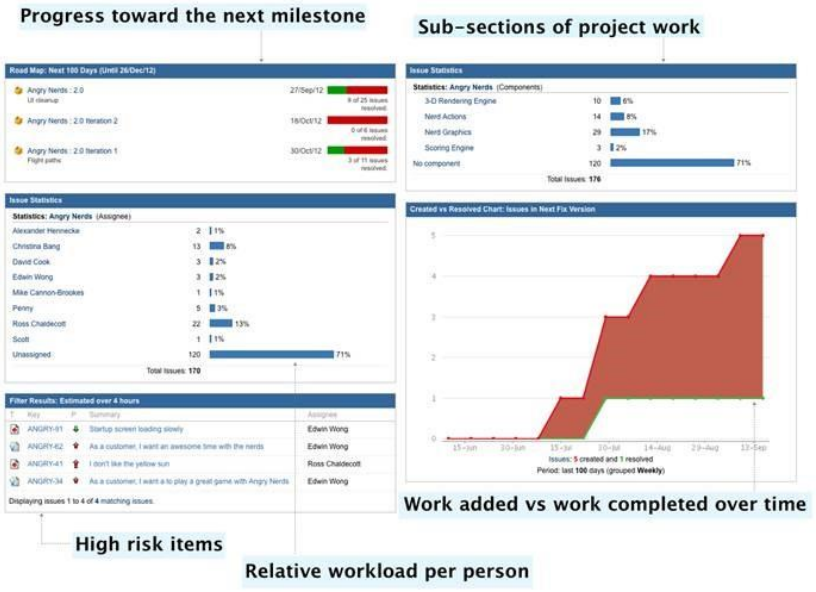
협업 도구를 통한 문화 강화

Jira 를 통한 스크럼 적용 및 업무 협업도구 강화

- Scrum Master(SM): 스크럼 주최 및 사회자 역할
- Product Owner(PO): 서비스별 업무 요구사항에 대해 의사결정 역할

Ex) 어떤 업무가 신규로 들어왔는지 긴급도는 얼마인지 에 대해서 결정

- Team Member: 업무를 수행하는 인원



DevOps 모델

협업 도구를 통한 문화 강화

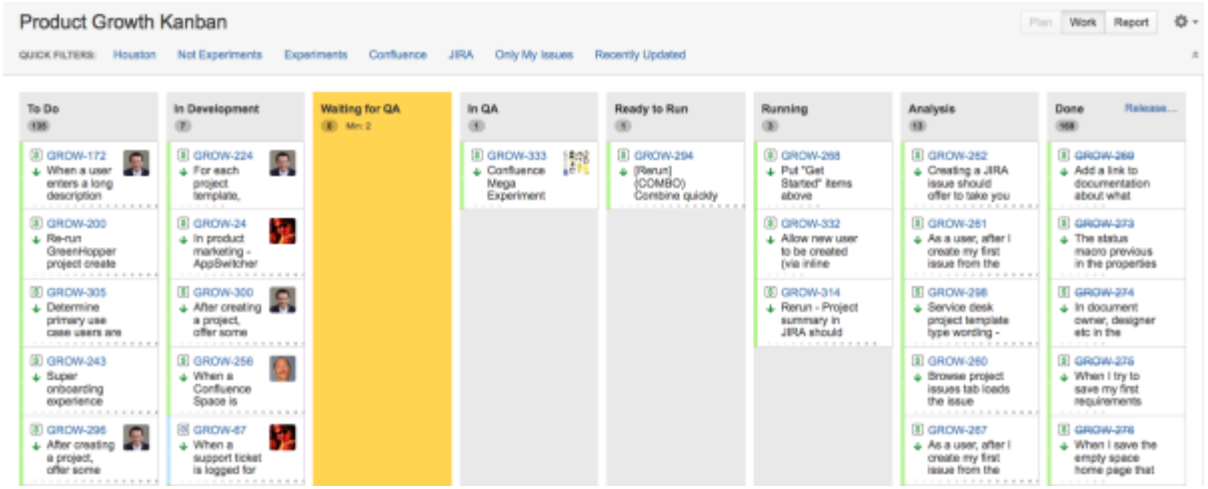
Jira 를 통한 칸반 활용

• Backlog: PO가 개발팀으로부터 받아온 요구사항 혹은 안정화, 성능 향상을 위해 해결 해야하는 이슈, 서비스데스크로부터 요청 등에 대한 일감을 모두 입력 및 수행 대기.(일감의 최대 개수 없음)

• Planned: Backlog에 있는 일감 중 시작 해야 하는 일에 대해 Planned 상태로 변경(팀원에 비례하여 최대 일감개수 협의 산정.)

• InWork: 현재 진행중인 업무(최대 진행중인 일감 개수는 인원에 맞춰서 정의)

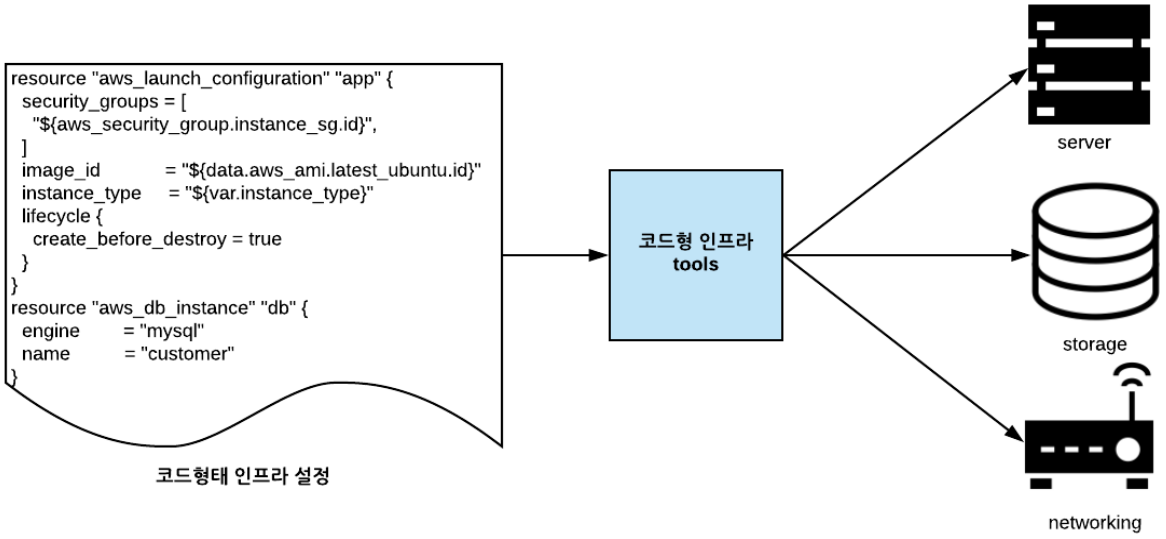
• Done: 완료된 업무



코드 형태의 인프라 구성 방안 및 도구

코드 형태의 인프라 의 이해

Infrastructure as code(IAC) 는 코드 버전 관리와 같은 소프트웨어 개발 기술을 사용하여 명시적인 코드 형태로 인프라를 프로비저닝하고 관리하는 방식 이며, 또한 인프라가 코드를 통해 정의되므로 인프라와 서버를 표준화된 패턴을 사용하여 배포하고, 최신 패치와 버전으로 업데이트하거나, 복제를 통해 다양한 환경에 동일한 인프라 구성 매칭이 가능하다.



코드 형태의 인프라 구성 방안 및 도구

코드 형태의 인프라 구성 영역

실행 가능한 코드 형태로 인프라를 명시적으로 정의 하고 DevOps 모델을 적용/ 운영 하기 위해서는 IAC를 통해 서버 뿐만이 아니라 데이터베이스, 네트워크, 어플리케이션등 의 형상을 변경할 수 있어야 한다. 크게 4가지 영역으로 IAC를 범주화 시킬수 있다.

Ad Hoc Script	일회성으로 인프라 구성/ 변경을 절차 지향적으로 작성한 스크립트(Bash, Perl, Python등)
Configuration Management	미리 정의된 형태 혹은 구조대로 표준에 맞게 코드를 정의 할 수 있으며 재사용성과 모듈화 그리고 분산된 환경에 동작 할 수 있도록 구성된 도구(Ansible, Chef..)
Server template	서버, 컨테이너 이미지 혹은 스냅샷을 통해 손쉽게 인프라를 확장 및 복제 할수 있게 하는 도구(Packer, Vagrant, Docker)
Server provisioning tool	정의된 형태 혹은 구조대로 서버 뿐만이 아니라 데이터베이스, 네트워크 및 플랫폼서비스들을 묶음 딴위로 배포 및 변경 가능하게 해주는 도구 (CloudFormation, Terraform)

Ad hoc script, Configuration management tool

Ad hoc, CM tool 비교

```
wget http://myrepo/software/redis/3.0.5/redis-3.0.5.tar.gz \
-o /opt/redis-3.0.5.tar.gz
cd /opt
tar zvfx /opt/redis-3.0.5.tar.gz
cd /opt/redis-3.0.5/
make
make install
```

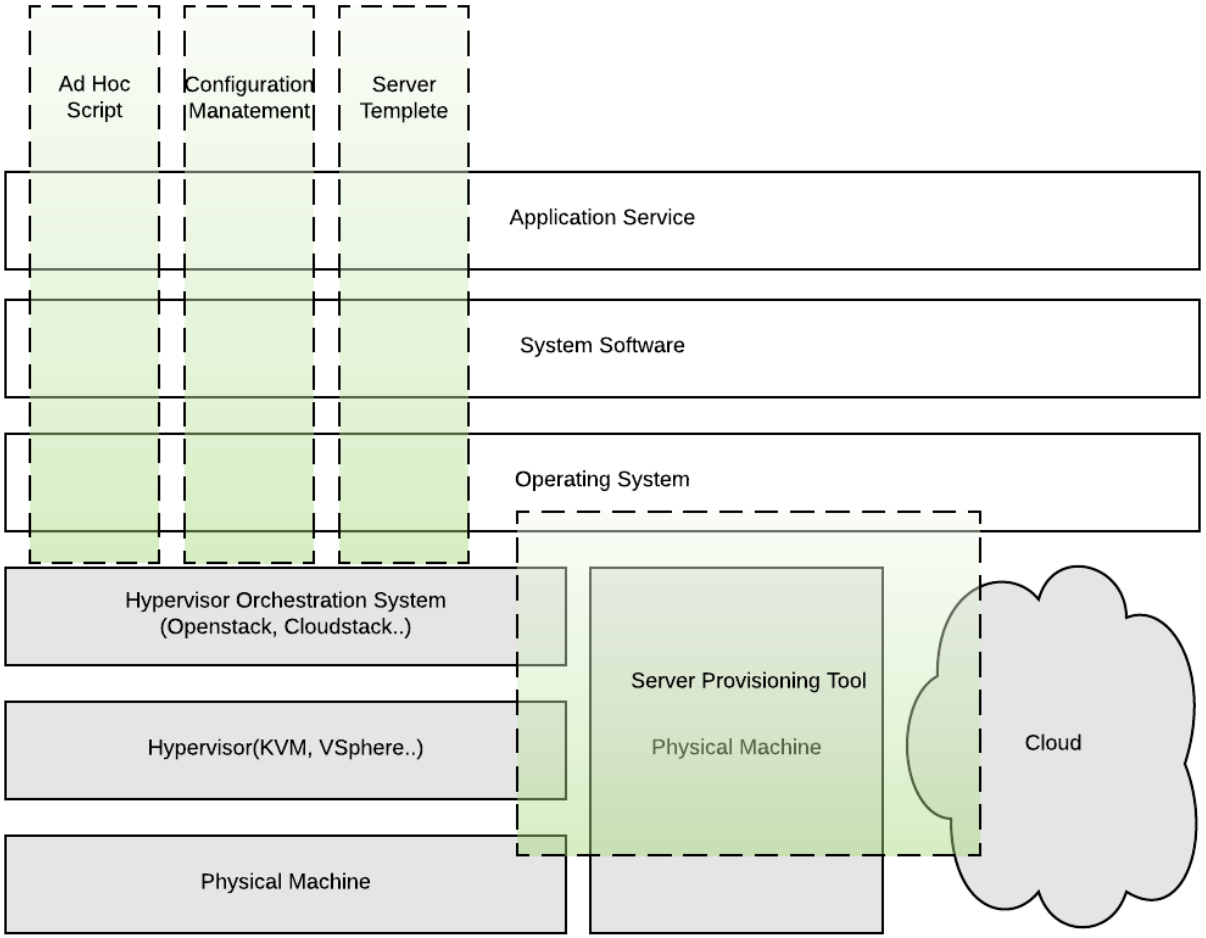
```
base:
    pkg.installed:
        - pkgs:
            - gcc
            - gcc-c++
            - make

redis-3.0.5:
    file.managed:
        - name: /srv/redis-3.0.5.tar.gz
        - source: salt://apps/redis/redis-3.0.5.tar.gz
        - source_hash: md5=c7ba233e5f92ad2f48860c815bb05480

extract-redis:
    cmd:
        - cwd: /opt
        - names:
            - tar xvf redis-3.0.5.tar.gz
        - run
        - require:
            - file: redis-3.0.5

configure-redis:
    cmd:
        - cwd: /opt/redis-3.0.5
        - names:
            - make
            - make install
        - run
        - require:
            - cmd: extract-redis
```

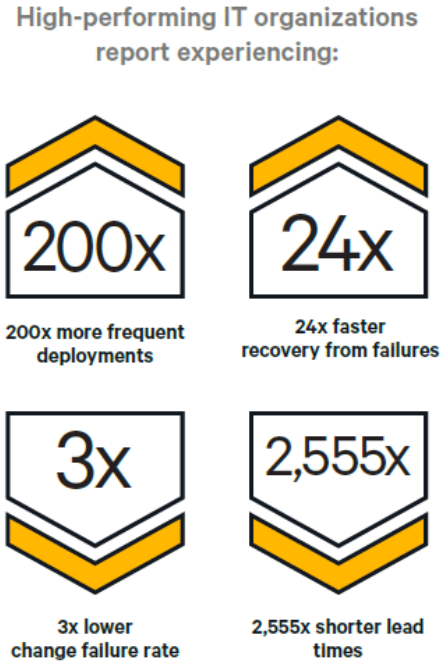
코드 형태의 인프라 구성 방안 및 도구



코드 형태의 인프라 구성 방안 및 도구

코드 형태의 인프라의 이점

IAC으로 인프라 리소스 구성을 자동화 하였을때 self service 형태로 인프라를 서비스화 가능하며. 빠르고 안전하게 리소스를 증설 및 복구가 가능하다. 또한 명시적인 코드 형태 관리를 통해 문서화와 버전 관리가 손쉬워 변경 이력 및 서비스 영향도에 대해 명확하게 파악 가능하다.



출처: 2016 State of DevOps Report - Puppet

코드 형태의 인프라 구성 방안 및 도구

코드 형태의 인프라 구성 도구

코드 형태의 인프라 도구에 대해서는 각 클라우드 플랫폼 별로 존재 하며 각 클라우드 플랫폼 API를 통해 Mashup 인터페이스 형태로 제공 하는 Terraform 소프트웨어가 존재 한다.

*Mashup: 각 플랫폼의 공개된 API 들의 기반 위에 독자적인 인터페이스를 구성하여 여러 플랫폼의 API 들을 하나의 인터페이스로 관리하는 패턴

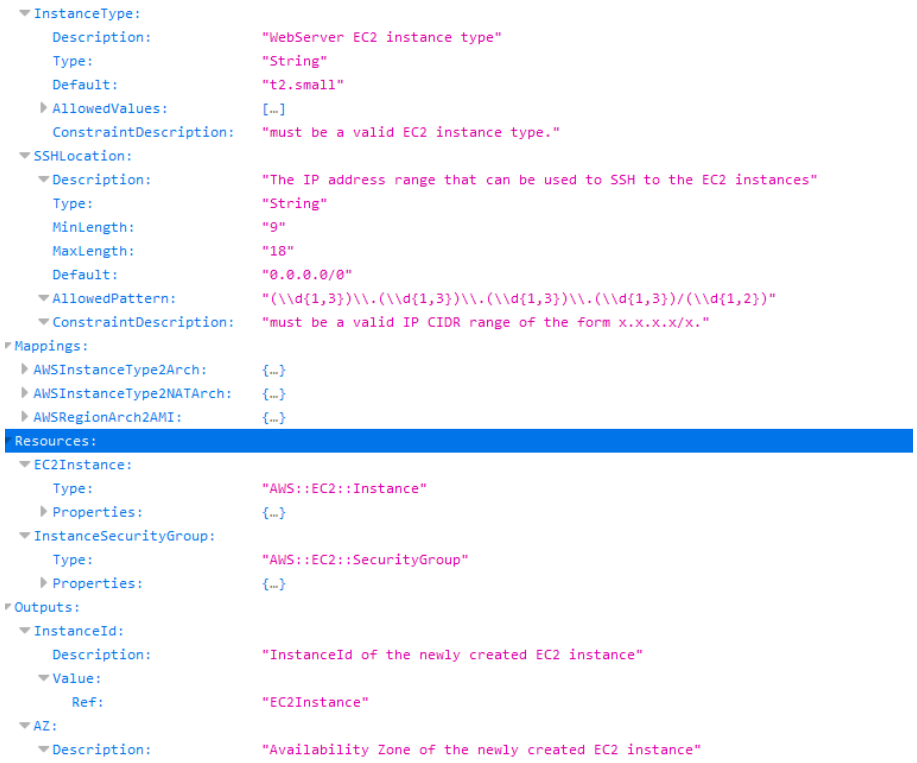
*코드 형태의 인프라 구성 도구

AWS CloudFormation	AWS 리소스를 배포 하기 위한 리소스 묶음(Stack)들을 JSON, YAML 형태로 구성
Google Cloud Deployment Manager	
HashiCorp Terraform	Multi Cloud 환경(AWS, GCP, Openstack)에서 동일한 인터페이스 형태(Terraform api)로 속성 및 리소스를 정의

코드 형태의 인프라 구성 방안 및 도구

AWS CloudFormation

간편하게 AWS 리소스 모음을 정리하고 배포할 수 있으며 수행 시 파라미터를 사전에 설정된 값 혹은 사용자 입력 또는 미리 정의된 종속성에 따라 구성 값을 입력 받으며 결과 데이터 항목도 정의 가능하며 여러 템플릿을 기반으로 새로운 템플릿도 구성 가능하다.



코드 형태의 인프라 구성 방안 및 도구

Google Cloud Deployment Manager

간편하게 GCP 리소스 모음을 정리하고 배포할 수 있으며 하나의 템플릿 파일이 아닌 스키마, 설정과 템플릿 파일로 각각 정의 하여 다양한 환경의 설정 관리를 별도의 파일 형태고 관리 가능하다.

```
imports:
  - path: network.jinja
resources:
- name: network
  type: network.jinja
  properties:
    region: us-central1
    subnets:
      - range: 10.177.0.0/17
        name: web
      - range: 10.177.128.0/17
        name: data
- name: web-instance
  type: compute.v1.instance
  properties:
    zone: us-central1-f
    machineType: https://www.googleapis.com/compute/v1/projects/myproject/zones/us-central1-f
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: https://www.googleapis.com/compute/v1/projects/debian-cloud/global/image
    networkInterfaces:
    - network: https://www.googleapis.com/compute/v1/projects/myproject/global/networks/defau
      accessConfigs:
      - name: External NAT
        type: ONE_TO_ONE_NAT
```

코드 형태의 인프라 구성 방안 및 도구

Hashicorp Terraform

특정 클라우드 플랫폼에 종속되지 않고 멀티 클라우드 환경에 통일화된 인터페이스로 구성 가능하며 사용자가 직접 정의한 표준대로 인프라를 코드화 하여 관리 및 버저닝 가능하다. 또한 실행 전에 변경될 리소스들에 대해 시뮬레이션 가능하며 서비스의 영향도를 사전에 파악 가능 하다.

```
resource "aws_elb" "frontend" {
  name = "frontend-load-balancer"
  listener {
    instance_port     = 8000
    instance_protocol = "http"
    lb_port           = 80
    lb_protocol       = "http"
  }

  instances = ["${aws_instance.app.*.id}"]
}

resource "aws_instance" "app" {
  count = 5

  ami           = "ami-408c7f28"
  instance_type = "t1.micro"
}
```

코드 형태의 인프라 구성 방안 및 도구

Terraform 특성

- Infrastructure as Code
- Execution Plans
- Resource Graph
- Change Automation

코드 형태의 인프라 구성 방안 및 도구

Terraform 기능 항목

import	다른 곳에서 사용중인 terrafom state파일을 import 가능 하며 동일한 인프라 구성 환경을 추가 가능
	
	resource "aws_instance" "example" {
	  # ...instance configuration...
	}
	
	$ terraform import aws_instance.example i-abcd1234

State	Terrafom은 인프라 관리와 설정을 위해 상태를 별도의 terraform.tfstate라는 이름의 file형태로 저장 한다. 이 상태 정보는 terrafom이 실제 리소스에 설정을 매핑 하고 metadata를 트레킹 하고 대규모의 인프라 구성시에 최소의 배포 시간을 갖도록 활용된다.
Provider	Cloud, VM, 물리환경등의 인프라 리소스를 생성하고 관리 하고 업데이트 하기 위해 API와 리소스를 상호 작용하는 역할을 담당

코드 형태의 인프라 구성 방안 및 도구

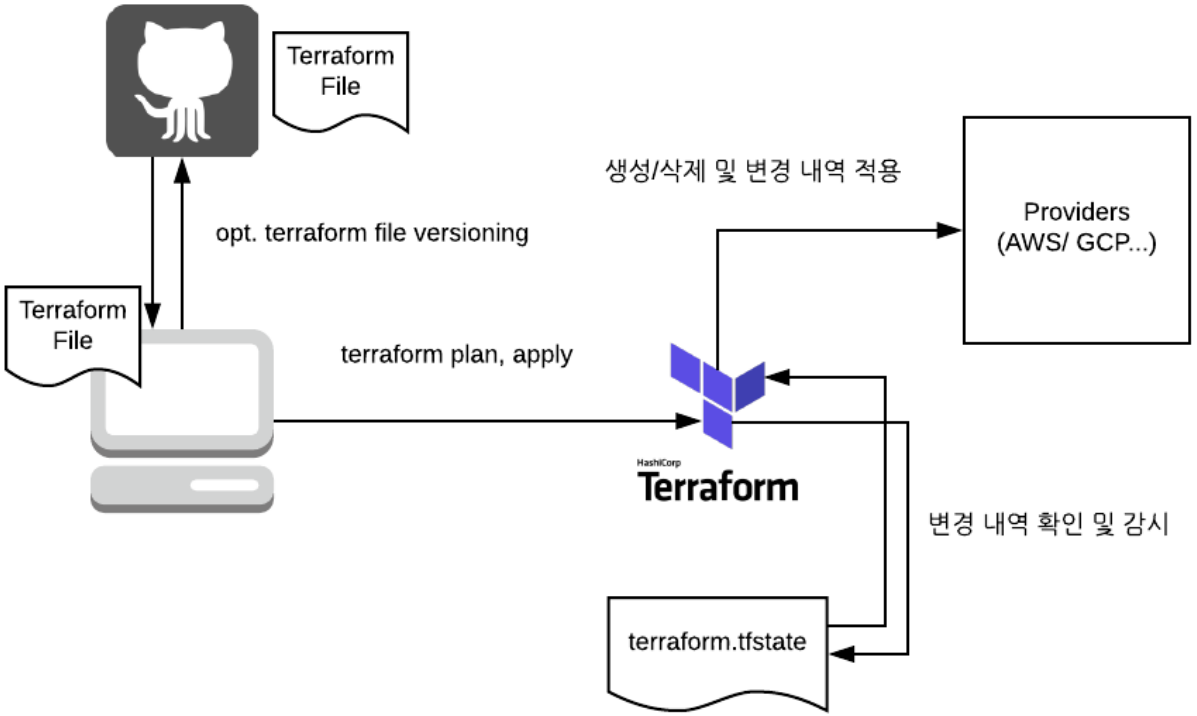
Terraform 기능 항목 - *Continued*

Provisioner	리소스의 생성과 삭제 단계에서 로컬과 원격 머신에 수행되는 스크립트를 수행할수 있도록 하는 기능이며 리소스를 로드(bootstrapping) 하거나 삭제전에 정리 하는 작업을 하거나 다른 설정 관리 툴을 수행 하는 등의 작업을 수행
	
	resource "aws_instance" "web" {
	  # …
	
	  provisioner "local-exec" {
	  command = "echo ${self.private_ip} > file.txt"
	  }
	}

Modules	모듈은 Terrafom 그룹으로 관리 되는 설정의 자체 내장한 패키지 이미 기본코드 구성은 물론 Terrafom에서 재사용 가능한 구성 요소를 만드는데 사용

코드 형태의 인프라 구성 방안 및 도구

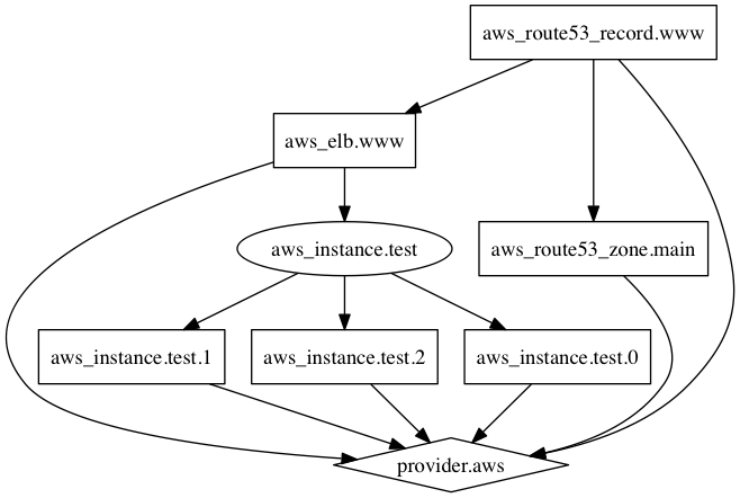
Terraform의 동작 프로세스

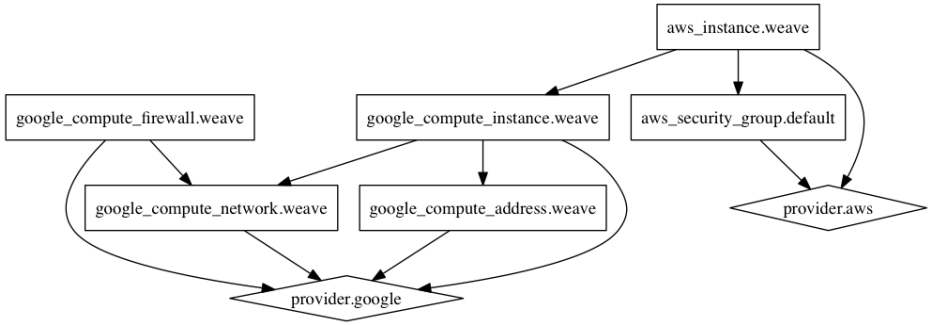


코드 형태의 인프라 구성 방안 및 도구

Terraform의 종속성 관계

Provider별로 리소스 종속성 (AWS, GCP)





DevOps 이해 및 코드형 인프라 구성 방안

DevOps 이해 및 코드형 인프라 구성 방안에 대해 알아 보았습니다.

DevOps 모델 소개와 방식 및 도구:

DevOps는 어플리케이션과 서비스를 빠른 속도로 제공할 수 있도록 조직의 역량을 향상시키는 문화 철학, 방식 및 도구의 조합이다. 소규모 업데이트를 자주 수행 하여 혁신의 속도 향상 하고 마이크로 서비스 아키텍처를 사용하여 애플리케이션의 유연성을 확보 해야 한다. 하지만 잦은 소프트웨어 빌드 및 릴리즈를 효과적으로 운영하기 위해 지속적 통합, 전달 및 마이크로 서비스와 코드 형태로 관리 해야 한다.

코드형 인프라 구성 방안 및 도구:

각 플랫폼 별 인프라 리소스 배포 도구인 AWS CloudFormation, Google Deployment manager 를 사용 하여 코드형태로 인프라 구성이 가능하며 특정 플랫폼에 종속되지 않고 사용자가 직접 정의한 표준대로 인프라를 코드화 하여 관리 하기 위해서는 Terraform활용이 보다 효과적이다.

Terraform 소개 및 활용 방법:

멀티 클라우드 환경에 통일화된 인터페이스로 구성 가능하며 사용자가 직접 정의한 표준대로 인프라를 코드화 하여 관리 및 버저닝 가능하며 실행전에 시뮬레이션을 통해 사전 영향도를 파악 가능하다.