AWS Essentials

7. 탄력성 있는 구현

Tacademy

CONTENTS

학습 목표

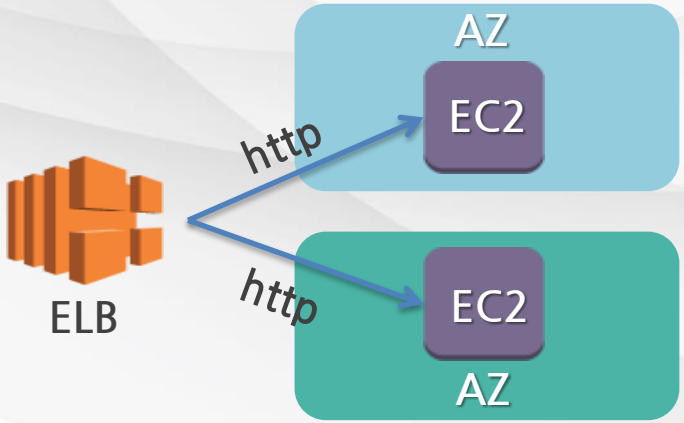
- AWS 컴퓨트 영역의 탄력적인 구성 방법을 이해할 수 있습니다.
- 코드 레벨에서 탄력적인 배포 방법을 이해할 수 있습니다.
- AWS의 관리 영역에서 탄력적인 운영 방법을 파악할 수 있습니다.

1. AWS 컴퓨트 영역의 탄력성

Elastic Load Balancing

네트워크 부하 분배기로 높은 탄력성과 가용성으로 EC2 인스턴스 앞에서 트래픽을 효율적으로 분산시켜 주는 서비스이다.

- 여러 인스턴스 및 여러 가용 영역에서 트래픽을 자동으로 라우팅하여 어플리케이션의 내결함성을 높이며 auto-scaling과 함께 서비스를 탄력적으로 확장시킨다.



Elastic Load Balancing

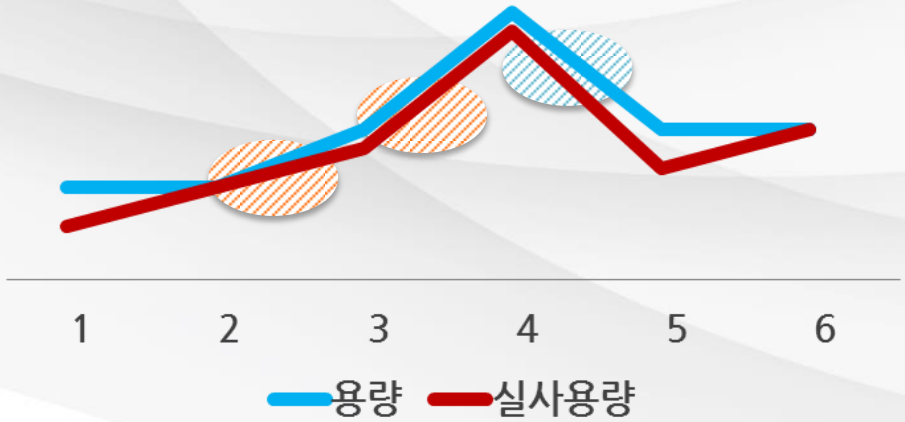
❖ 활용 사례

- AWS EC2 인스턴스 상태 감지를 통해 비정상적인 EC2를 제외한 정상적인 서버에 트래픽을 전달함으로써 **내결함성**을 향상시킨다.
- ELB가 통합 **인증서 관리 및 SSL 복호화를 제공**하므로 중앙에서 인증서 관리를 수행하고 EC2는 서비스 트래픽만 수용하여 자원 활용률을 높인다.
- 외부 서비스용이 아닌 내부 어플리케이션 레이어에 적용하여 구성요소 간의 **트래픽 전달을 탄력적**으로 수행시킨다.

Auto Scaling

EC2 용량을 사용자가 **정의한 조건에 따라 자동으로 확장·축소** 하여 어플리케이션의 품질을 유지하는 서비스이다.

- 수요가 급증할 경우, Amazon EC2 인스턴스의 수를 자동으로 증가시키기 때문에 **성능을 그대로 유지**
- 수요가 적을 경우, 자동으로 **용량을 감소시켜 비용 낭비를 제거**



Auto Scaling

❖ 구성 요소

그룹

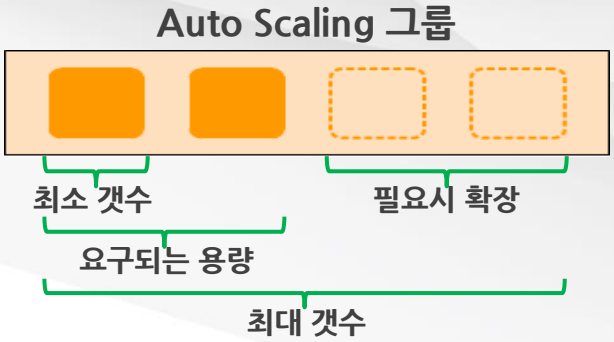
- 논리적인 관리목적의 인스턴스 단위
- 최소와 최대 개수와 요구하는 인스턴스 수를 지정

시작 구성

- EC2를 구성할 때 활용하는 템플릿
- 인스턴스 이미지 정보, 사이즈, 보안그룹 등 시작에 관련한 정보 지정

확장 계획

- 확장을 트리거링하는 임계값이나 방식 지정



2. AWS 코드 레벨의 탄력성

Elastic Beanstalk

Java, .NET, PHP, Node.js, Python, Ruby, Go, Docker를 사용하여 Apache, Nginx, Passenger, IIS와 같은 서버에서 개발된 **웹 애플리케이션 및 서비스를 간편하게 배포하고 확장할 수 있는 서비스**이다.

- 소스 코드 업로드만 하면 EC2 용량 프로비저닝, 로드 밸런싱, Auto Scaling부터 시작하여 애플리케이션 상태 모니터링까지 자동으로 배포 처리

OpsWorks

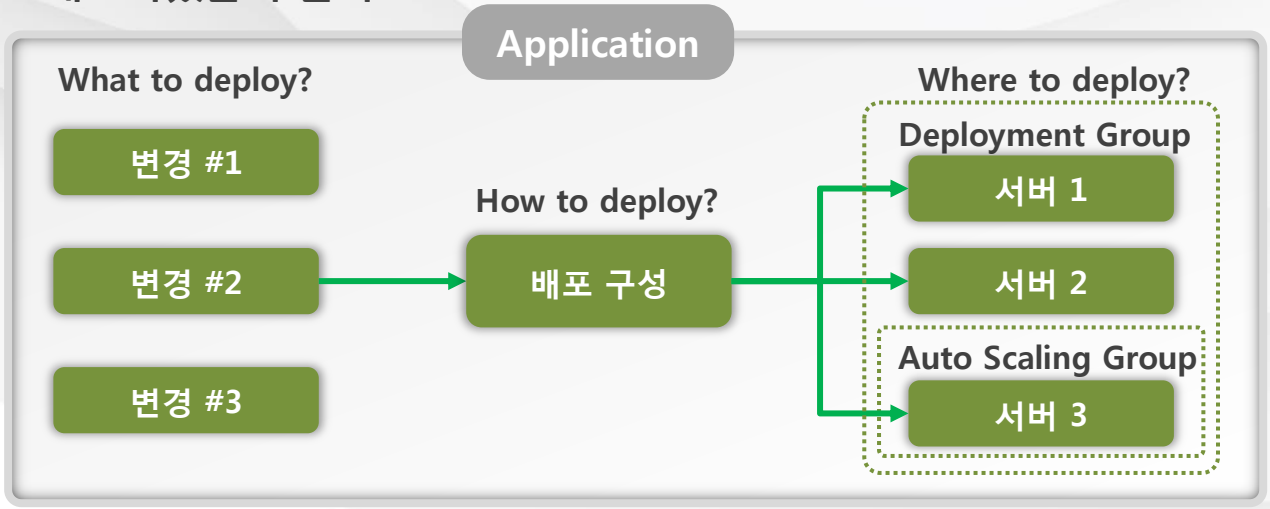
Chef를 사용하여 어플리케이션의 **구성을 자동화**하고 대규모 자원의 **구성 정보를 관리 및 운영** 해주는 서비스이다.

- 스택, 계층, 앱 같은 개념을 사용하여 어플리케이션을 모델링 및 시각화하고 대시보드를 통해 어플리케이션 레이어 상태 관리

CodeDeploy

실행 중인 모든 인스턴스에 대한 **코드 배포를 자동화**하여
안전하고 신속하게 어플리케이션을 배포할 수 있는 서비스이다.

- Console 또는 CLI를 통해 손쉽게 배포 상태를 중앙에서 추적할 수 있으며, 각 어플리케이션 수정 버전이 언제 어떤 인스턴스에 배포되었는지 관리

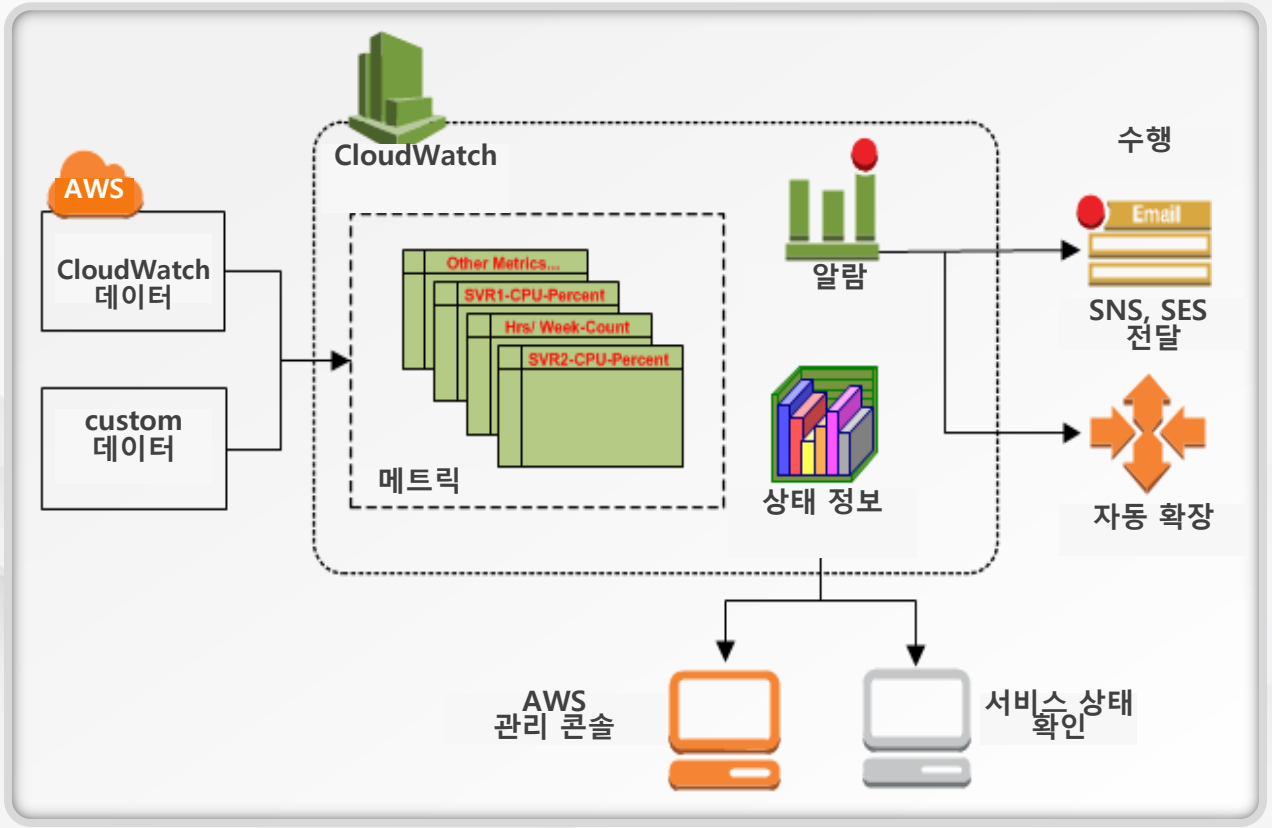


3. AWS 관리 영역의 탄력성

3. AWS 관리 영역의 탄력성

CloudWatch

- 장애에 대한 디자인 구성
- 실시간 모니터링 및 장애 감지 수행
- 대시보드, API를 통해 관리의 편의성 제공
- Auto-Scaling 그룹 설정 시 확장에 대한 임계값 요소에도 모니터링 값이 필수로 작용

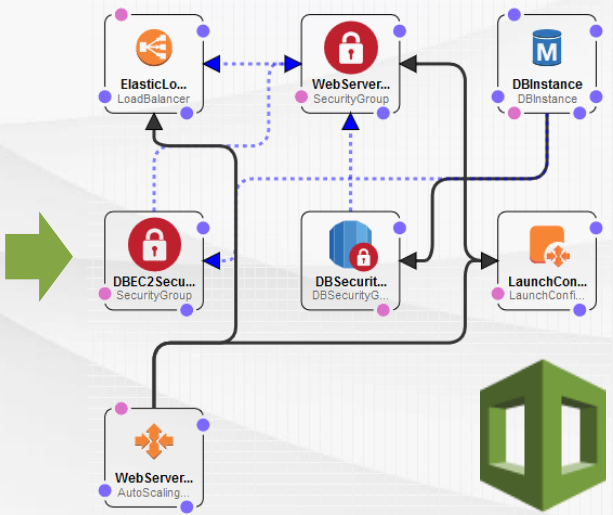


CloudFormation

인프라 리소스의 묶음을 **순서에 따라 쉽게 생성, 구성, 관리**하는 서비스이다.

- 리소스 환경을 **명시적으로 코드화**하여 프로비저닝 및 업데이트를 탄력적으로 관리

```
{
  "Parameters": {
    "KeyName": {...},
    "InstanceType": {
      "Description": "WebServer EC2 instance type",
      "Type": "String",
      "Default": "t2.small",
      "AllowedValues": [...],
      "ConstraintDescription": "must be a valid EC2 instance type."
    },
    "SSHLocation": {...},
    "DBClass": {
      "Description": "Database instance class",
      "Type": "String",
      "Default": "db.t2.small",
      "AllowedValues": [...],
      "ConstraintDescription": "must select a valid database instance type."
    },
    "DBName": {...},
    "DBUser": {...},
    "DBPassword": {...},
    "MultiAZDatabase": {
      "Default": "false",
      "Description": "Create a Multi-AZ MySQL Amazon RDS database instance",
      "Type": "String",
      "AllowedValues": [
        "true",
        "false"
      ],
      "ConstraintDescription": "must be either true or false."
    },
    "WebServerCapacity": {...},
```



학습정리

지금까지 [탄력성 있는 구현]에 대해서 살펴보았습니다.

AWS 컴퓨트 영역의 탄력성

Elastic LoadBalancing, Auto scaling 등을 통해 컴퓨트 영역을 탄력적으로 운영하여 내결함성과 성능을 향상시켜 높은 품질로 어플리케이션을 서비스할 뿐만 아니라 효율적인 비용 운영도 가능하게 한다.

AWS 코드 레벨의 탄력성

Elastic Beanstalk, OpsWorks, CodeDeploy 등을 통해 코드 레벨을 탄력적으로 운영하여 손쉽고 빠르게 다양한 서비스 레이어를 효과적으로 관리한다.

AWS 관리 영역의 탄력성

CloudWatch, CloudFormation 등을 통해 관리 영역에서 모니터링과 리소스들을 Code화하여 보다 명시적·탄력적으로 운영한다.