in을 사용하여 IaC AWS CDK 프로젝트를 TypeScript 만드는 모범 사례

# AWS 규범적 지침

# AWS 규범적 지침: in을 사용하여 IaC AWS CDK 프로젝트를 TypeScript 만드는 모범 사례

# Table of Contents

소개 ..... 1
목표 ..... 2
모범 사례 ..... 3
대규모 프로젝트를 위한 코드 구성 ..... 3
코드 구성이 중요한 이유 ..... 3
규모에 맞게 코드를 구성하는 방법 ..... 3
샘플 코드 구성 ..... 4
재사용 가능한 패턴 개발 ..... 6
추상 팩토리 ..... 6
책임 사슬 ..... 7
구성 생성 또는 확장 ..... 8
구성이란 ..... 8
다양한 유형의 구문 ..... 8
구성을 직접 생성하는 방법 ..... 9
L2 구성 생성 또는 확장 ..... 10
L3 구성 생성 ..... 10
이스케이프 해치 ..... 11
사용자 정의 리소스 ..... 12
TypeScript 모범 사례를 따르세요. ..... 15
데이터 설명 ..... 15
enum 사용 ..... 16
사용자 인터페이스 ..... 16
인터페이스 확장 ..... 17
빈 인터페이스 사용 금지 ..... 18
팩토리 사용 ..... 18
속성에 구조 분해 사용 ..... 18
표준 이름 지정 규칙 정의 ..... 19
var 키워드를 사용하지 마세요. ..... 19
ESLint와 Prettier를 사용해 보세요. ..... 19
액세스 한정자 사용 ..... 20
유틸리티 유형 사용 ..... 20
보안 취약성 및 형식 오류 스캔 ..... 21
보안 접근 방식 및 도구 ..... 21
일반 개발 도구 ..... 22

설명서 개발 및 세부 조정 ..... 22
구문에 코드 문서가 필요한 이유 AWS CDK ..... 23
AWS 구성 라이브러리와 TypeDoc 함께 사용 ..... 23
테스트 기반 개발 접근 방식 채택 ..... 24
유닛 테스트 ..... 25
통합 테스트 ..... 27
구성에 릴리스 및 버전 제어 사용 ..... 27
에 대한 버전 제어 AWS CDK ..... 27
AWS CDK 구문을 위한 리포지토리 및 패키징 ..... 28
를 위한 컨스트럭트 릴리즈 AWS CDK ..... 29
라이브러리 버전 관리 적용 ..... 30
FAQ ..... 31
어떤 문제를 TypeScript 해결할 수 있나요? ..... 31
사용해야 하는 이유는 무엇인가요? TypeScript ..... 31
AWS CDKOR를 써야 하나요 CloudFormation? ..... 31
새로 출시된 AWS CDK 버전을 지원하지 않으면 어떻게 되나요? AWS 서비스 ..... 31
에서 지원하는 다양한 프로그래밍 언어는 무엇입니까? AWS CDK ..... 31
AWS CDK비용은 얼마나 드나요? ..... 32
다음 단계 ..... 33
리소스 ..... 34
문서 기록 ..... 35
용어집 ..... 36
# ..... 36
A ..... 37
B ..... 39
C ..... 41
D ..... 44
E ..... 48
F ..... 50
G ..... 51
H ..... 52
I ..... 53
L ..... 55
M ..... 56
O ..... 60
P ..... 62

Q ..... 64
R ..... 65
S ..... 67
T ..... 71
U ..... 72
V ..... 72
W ..... 73
Z ..... 74
..... lxxv

# AWS CDK를 사용하여 IaC 프로젝트를 TypeScript 생성하는 모범 사례

Sandeep Gawande, Mason Cahill, Sandip Gangapadhyay, Siamak Heshmati 및 Rajneesh Tyagi, Amazon Web Services(AWS)

2024년 2월 (문서 기록)

이 가이드에서는 in을 사용하여 대규모 코드형 인프라 (AWS Cloud Development Kit (AWS CDK)IaC) 프로젝트를 구축하고 TypeScript 배포하기 위한 권장 사항과 모범 사례를 제공합니다. 코드로 클라우드 인프라를 정의하고 해당 인프라를 프로비저닝하기 위한 AWS CDK 프레임워크입니다. AWS CloudFormation 잘 정의된 프로젝트 구조가 없다면 대규모 프로젝트를 위한 AWS CDK 코드베이스를 구축하고 관리하는 것이 어려울 수 있습니다. 이러한 문제를 해결하기 위해 일부 조직에서는 대규모 프로젝트에 안티 패턴을 사용하지만 이러한 패턴으로 인해 프로젝트 속도가 느려지고 조직에 부정적인 영향을 미치는 다른 문제가 발생할 수 있습니다. 예를 들어, 안티 패턴은 개발자 온보딩, 버그 수정 및 새 기능 채택을 복잡하게 만들고 속도를 늦출 수 있습니다.

이 가이드에서는 안티 패턴 사용에 대한 대안을 제시하고 확장성, 테스트 및 보안 모범 사례에 맞게 코드를 구성하는 방법을 보여줍니다. 이 가이드를 사용하여 IaC 프로젝트의 코드 품질을 개선하고 비즈니스 민첩성을 극대화할 수 있습니다. 이 가이드는 건축가, 기술 책임자, 인프라 엔지니어 및 대규모 프로젝트를 위해 잘 설계된 AWS CDK 프로젝트를 구축하려는 기타 모든 역할을 대상으로 합니다.

# 목표

이 가이드는 다음과 같은 목표 비즈니스 성과를 달성하는 데 도움이 될 수 있습니다.

- 비용 절감 — 를 사용하여 조직의 보안, 규정 준수 및 거버넌스 요구 사항을 충족하는 재사용 가능한 구성 요소를 직접 설계할 수 있습니다. AWS CDK 또한 구성 요소를 조직 전체에서 쉽게 공유할 수 있으므로 기본적으로 모범 사례에 맞는 새 프로젝트를 빠르게 부트스트랩할 수 있습니다.
- 출시 시간 단축 — 익숙한 기능을 활용하여 개발 프로세스를 가속화하십시오. AWS CDK 이렇게 하면 배포 시 재사용성이 향상되고 개발 노력이 줄어듭니다.
- 개발자 생산성 향상 — 개발자는 익숙한 프로그래밍 언어를 사용하여 인프라를 정의할 수 있습니다. 이를 통해 개발자는 AWS 리소스를 표현하고 유지 관리할 수 있습니다. 이로 인해 개발자의 효율성과 협업이 향상될 수 있습니다.

# 모범 사례

이 섹션에서는 다음 모범 사례를 간략히 설명합니다.

- 대규모 프로젝트를 위한 코드 구성
- 재사용 가능한 패턴 개발
- 구성 생성 또는 확장
- TypeScript 모범 사례 따르기
- 보안 취약성 및 형식 오류 스캔
- 설명서 개발 및 세부 조정
- 테스트 기반 개발 접근 방식 채택
- 구성에 릴리스 및 버전 제어 사용
- 라이브러리 버전 관리 적용

## 대규모 프로젝트를 위한 코드 구성

### 코드 구성이 중요한 이유

대규모 AWS CDK 프로젝트의 경우 품질이 우수하고 잘 정의된 구조를 갖는 것이 중요합니다. 프로젝트가 커지고 지원되는 기능 및 구문의 수가 늘어날수록 코드 탐색은 더욱 어려워집니다. 이러한 어려움은 생산성에 영향을 미치고 개발자 온보딩 속도를 늦출 수 있습니다.

### 규모에 맞게 코드를 구성하는 방법

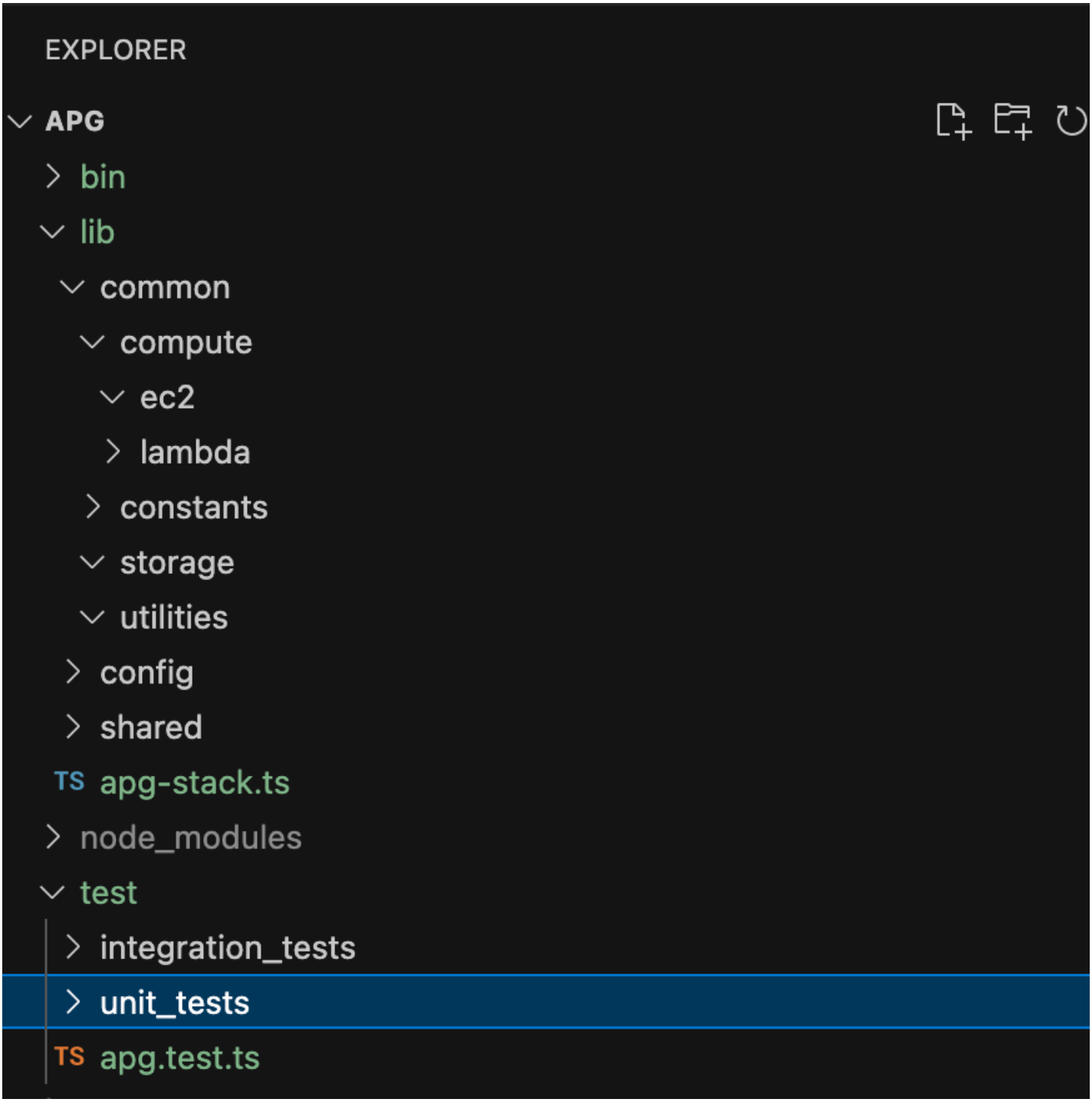
높은 수준의 코드 유연성과 가독성을 확보하려면 기능에 따라 코드를 논리적인 부분으로 나누는 것이 좋습니다. 이 구분은 대부분의 구성이 서로 다른 비즈니스 영역에 사용된다는 사실을 반영합니다. 예를 들어 프론트엔드와 백엔드 애플리케이션 모두 AWS Lambda 함수가 필요하고 동일한 소스 코드를 사용할 수 있습니다. 팩토리는 생성 로직을 클라이언트에 노출하지 않고도 객체를 생성하고 공통 인터페이스를 사용하여 새로 생성된 객체를 참조할 수 있습니다. 팩토리를 효과적인 패턴으로 사용하여 코드베이스에서 일관된 동작을 생성할 수 있습니다. 또한 팩토리는 신뢰할 수 있는 단일 소스 역할을 하므로 코드가 반복되지 않도록 하고 문제를 더 쉽게 해결할 수 있습니다.

팩토리의 운영 방식을 더 잘 이해하려면 자동차 제조업체의 예를 생각해 보세요. 자동차 제조업체는 타이어 제조에 필요한 지식과 인프라를 갖추지 않아도 됩니다. 대신 자동차 제조업체는 전문 지식을 타이

어 전문 제조업체에 아웃소싱한 다음 필요에 따라 해당 제조업체에 타이어를 주문하기만 하면 됩니다. 코드에도 동일한 원칙이 적용됩니다. 예를 들어, 고품질 Lambda 함수를 구축할 수 있는 Lambda 팩토리를 생성한 다음 Lambda 함수를 생성해야 할 때마다 코드에서 Lambda 팩토리를 직접적으로 호출할 수 있습니다. 마찬가지로 동일한 아웃소싱 프로세스를 사용하여 애플리케이션을 분리하고 모듈식 구성 요소를 구축할 수 있습니다.

## 샘플 코드 구성

다음 이미지에 표시된 것처럼 다음 TypeScript 샘플 프로젝트에는 모든 구성 또는 공통 기능을 보관할 수 있는 공통 폴더가 포함되어 있습니다.




예를 들어, common 폴더에 있는 compute 폴더에는 다양한 컴퓨팅 구성에 대한 모든 논리가 들어 있습니다. 신규 개발자는 다른 리소스에 영향을 주지 않고 새 컴퓨팅 구성을 쉽게 추가할 수 있습니다. 다른 모든 구문은 내부적으로 새 리소스를 만들 필요가 없습니다. 대신 이러한 구성은 단순히 공용 구성 팩토리를 직접적으로 호출합니다. 스토리지 등의 다른 구성도 같은 방식으로 조직할 수 있습니다.

구성에는 로직을 보관하는 common 폴더에서 분리해야 하는 환경 기반 데이터가 포함되어 있습니다. 공유 폴더에 공용 config 데이터를 배치하는 것이 좋습니다. 또한 utilities 폴더를 사용하여 모든 도우미 기능을 제공하고 스크립트를 정리하는 것이 좋습니다.

# 재사용 가능한 패턴 개발

소프트웨어 설계 패턴은 소프트웨어 개발의 일반적인 문제에 대한 재사용 가능한 솔루션입니다. 이는 소프트웨어 엔지니어가 모범 사례를 따르는 제품을 만드는 데 도움이 되는 가이드 또는 패러다임 역할을 합니다. 이 섹션에서는 AWS CDK 코드베이스에서 사용할 수 있는 재사용 가능한 두 가지 패턴, 즉 추상 팩토리 패턴과 책임 사슬 패턴에 대한 개요를 제공합니다. 각 패턴을 청사진으로 사용하고 코드의 특정 설계 문제에 맞게 사용자 지정할 수 있습니다. 설계 패턴에 대한 자세한 내용은 Refactoring.Guru 설명서의 Design Patterns를 참조하세요.

## 추상 팩토리

추상 팩토리 패턴은 구체적인 클래스를 지정하지 않고도 관련 객체 또는 종속 객체의 패밀리를 만들 수 있는 인터페이스를 제공합니다. 이 패턴은 다음과 같은 사용 사례에 적용됩니다.

- 클라이언트가 시스템에서 객체를 생성하고 구성하는 방법과 무관한 경우
- 시스템이 여러 객체 패밀리로 구성되어 있고 이러한 패밀리가 함께 사용되도록 설계된 경우
- 특정 종속성을 구성하기 위해 런타임 값이 있어야 하는 경우

추상 팩토리 패턴에 대한 자세한 내용은 Refactoring.Guru 설명서의 추상 팩토리를 참조하십시오.
TypeScript

다음 코드 예제는 Abstract Factory 패턴을 사용하여 Amazon Elastic Block Store(Amazon EBS) 스토리지 팩토리를 구축하는 방법을 보여줍니다.

```
abstract class EBSStorage {
    abstract initialize(): void;
}

class ProductEbs extends EBSStorage{
    constructor(value: String) {
        super();
        console.log(value);
    }
    initialize(): void {}
}
```

```
abstract class AbstractFactory {
    abstract createEbs(): EBSStorage
}

class EbsFactory extends AbstractFactory {
    createEbs(): ProductEbs{
        return new ProductEbs('EBS Created.')
    }
}

const ebs = new EbsFactory();
ebs.createEbs();
```

## 책임 사슬

책임 사슬은 잠재적 핸들러 중 하나가 요청을 처리할 때까지 잠재적 핸들러 사슬을 따라 요청을 전달할 수 있는 동작 디자인 패턴입니다. 책임 사슬 패턴은 다음 사용 사례에 적용됩니다.

- 런타임 시 결정된 여러 객체가 요청을 처리할 후보인 경우
- 코드에서 핸들러를 명시적으로 지정하지 않으려는 경우
- 수신자를 명시적으로 지정하지 않고 여러 객체 중 하나에 요청을 보내려는 경우

책임 사슬 패턴에 대한 자세한 내용은 Refactoring.Guru 설명서의 책임 사슬을 참조하십시오.
TypeScript

다음 코드는 책임 사슬 패턴을 사용하여 작업을 완료하는 데 필요한 일련의 작업을 구성하는 방법의 예를 보여줍니다.

```
interface Handler {
    setNext(handler: Handler): Handler;
    handle(request: string): string;
}
abstract class AbstractHandler implements Handler
{
    private nextHandler: Handler;
    public setNext(handler: Handler): Handler {
        this.nextHandler = handler;
        return handler;
    }
```

```
    public handle(request: string): string {
        if (this.nextHandler) {
            return this.nextHandler.handle(request);
        }
        return '';
    }
}

class KMSHandler extends AbstractHandler {
    public handle(request: string): string {
        return super.handle(request);
    }
}
```

# 구성 생성 또는 확장

## 구성이란

구문은 애플리케이션의 기본 구성 요소입니다. AWS CDK 구문은 Amazon Simple Storage Service (Amazon S3) 버킷과 같은 단일 AWS 리소스를 나타내거나 여러 관련 리소스로 구성된 상위 수준의 추상화일 수 있습니다. AWS 구성 요소에는 연결된 컴퓨팅 용량이 있는 작업자 대기열이나 모니터링 리소스 및 대시보드가 있는 예약된 작업이 포함될 수 있습니다. AWS CDK 여기에는 구성 라이브러리라는 구문 컬렉션이 포함됩니다. AWS 라이브러리에는 모든 구성을 위한 구문이 들어 있습니다. AWS 서비스Construct Hub를 사용하여 타사 및 오픈 소스 AWS AWS CDK 커뮤니티의 추가 구성을 검색할 수 있습니다.

## 다양한 유형의 구문

다음과 같은 세 가지 유형의 구문이 있습니다. AWS CDK

- L1 구성 — L1 구문은 더도 말고 덜도 말고 CloudFormation —에 의해 정의된 리소스와 정확히 같습니다. 구성에 필요한 리소스는 사용자가 직접 제공해야 합니다. 이러한 L1 구조는 매우 기본적이므로 수동으로 구성해야 합니다. L1 구조에는 `Cfn` 접두사가 있으며 사양과 직접 일치합니다. CloudFormation 이러한 서비스를 지원하는 AWS CDK 즉시 새 CloudFormation 제품이 AWS 서비스 지원됩니다. CfnBucketL1 구문의 좋은 예입니다. 이 클래스는 모든 속성을 명시적으로 구성해야 하는 S3 버킷을 나타냅니다. L2 또는 L3 구문을 찾을 수 없는 경우에만 L1 구문을 사용하는 것이 좋습니다.
- L2 구성 - L2 또는 L2 구성에는 공용 표준 문안 코드와 글루 로직이 있습니다. 이러한 구문은 편리한 기본값과 함께 제공되므로 이에 대해 알아야 할 지식의 양을 줄일 수 있습니다. L2 구조는 인텐트 기

반 API를 사용하여 리소스를 구성하고 일반적으로 해당 L1 모듈을 캡슐화합니다. L2 구성의 좋은 예는 버킷입니다. 이 클래스는 bucket.add Rule () 과 같은 기본 속성과 메서드를 사용하여 S3 버킷을 생성합니다. 이 메서드는 버킷에 수명 주기 규칙을 추가합니다. LifeCycle

- L3 구성 - 계층 3 또는 L3 구성을 패턴이라고 합니다. L3 구조는 종종 여러 종류의 리소스가 포함되는 일반적인 작업을 완료하는 데 도움이 되도록 설계되었습니다. AWS이는 L2 구성보다 훨씬 더 구체적이고 독단적이며 특정 사용 사례에 사용됩니다. AWS-ecs-패턴을 예로 들 수 있습니다. ApplicationLoadBalancedFargate서비스 구조는 Application Load Balancer를 사용하는 AWS Fargate 컨테이너 클러스터를 포함하는 아키텍처를 나타냅니다. 또 다른 예로 AWS-apigateway를 들 수 있습니다. LambdaRestAPI 구조체. 이 구조는 Lambda 함수가 지원하는 Amazon API Gateway API를 나타냅니다.

구성 수준이 높아질수록 이러한 구성이 어떻게 사용될 것인지에 대한 가정도 늘어납니다. 이를 통해 매우 구체적인 사용 사례에 대해 보다 효과적인 기본값이 적용된 인터페이스를 제공할 수 있습니다.

## 구성을 직접 생성하는 방법

구성을 직접 정의하려면 특정 접근 방식을 따라야 합니다. 이는 모든 구성이 `Construct` 클래스를 확장하기 때문입니다. `Construct` 클래스는 구성 트리의 구성 요소입니다. 구성은 `Construct` 기본 클래스를 확장하는 클래스로 구현됩니다. 모든 구성은 초기화될 때 3개의 파라미터를 사용합니다.

- 범위 — 구성의 부모 또는 소유자 (스택 또는 다른 구문) 로, 구성 트리에서의 위치를 결정합니다. 일반적으로 범위에는 현재 객체를 나타내는 `this` 또는 `self`(Python의 경우)를 전달해야 합니다.
- id - 이 범위 내에서 고유해야 하는 식별자입니다. 식별자는 현재 구조 내에 정의된 모든 항목의 네임스페이스 역할을 하며 리소스 이름 및 논리적 ID와 같은 고유한 ID를 할당하는 데 사용됩니다. CloudFormation
- Props — 구문의 초기 구성을 정의하는 속성 집합입니다.

다음 예시에서는 구성을 정의하는 방법을 보여줍니다.

```
import { Construct } from 'constructs';

export interface CustomProps {
  // List all the properties
  Name: string;
}
export class MyConstruct extends Construct {
  constructor(scope: Construct, id: string, props: CustomProps) {
```

```
    super(scope, id);

    // TODO
  }
}
```

## L2 구성 생성 또는 확장

L2 구문은 “클라우드 구성 요소”를 나타내며 구성 요소를 만드는 데 필요한 모든 것을 CloudFormation 캡슐화합니다. L2 구문은 하나 이상의 AWS 리소스를 포함할 수 있으며, 구문을 직접 사용자 정의할 수 있습니다. L2 구문을 만들거나 확장하면 코드를 다시 정의하지 않고도 CloudFormation 스택의 구성 요소를 재사용할 수 있다는 이점이 있습니다. 간단히 구성을 클래스로 가져올 수 있습니다.

기존 구문과 “is a” 관계가 있는 경우 기존 구문을 확장하여 추가 기본 기능을 추가할 수 있습니다. 기존 L2 구문의 속성을 재사용하는 것이 가장 좋습니다. 생성자에서 직접 속성을 수정하여 속성을 덮어쓸 수 있습니다.

다음 예제에서는 모범 사례에 맞춰 `s3.Bucket`이라는 기존 L2 구성을 확장하는 방법을 보여줍니다. 확장은 이 새로운 구성에서 생성된 모든 객체(이 예에서는 S3 버킷)가 항상 이러한 기본값이 설정되도록 `versioned`, `publicReadAccess`, `blockPublicAccess` 등의 기본 속성을 설정합니다.

```
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
export class MySecureBucket extends s3.Bucket {
  constructor(scope: Construct, id: string, props?: s3.BucketProps) {

    super(scope, id, {
      ...props,
      versioned: true,
      publicReadAccess: false,
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL
    });
  }
}
```

## L3 구성 생성

기존 구문 컴포지션과 “a” 관계가 있는 경우에는 컴포지션을 사용하는 것이 좋습니다. 컴퍼지션이란 기존의 다른 구성 위에 자신만의 구성을 구축하는 것을 의미합니다. 고유한 패턴을 생성하여 공유 가능한 단일 상위 수준 L3 구문 내에 모든 리소스와 해당 기본값을 캡슐화할 수 있습니다. L3 구성(패턴)을 직

접 생성하면 코드를 다시 정의하지 않고도 스택의 구성 요소를 재사용할 수 있다는 이점이 있습니다. 간단히 구성을 클래스로 가져올 수 있습니다. 이러한 패턴은 소비자가 제한된 양의 지식으로 공통 패턴을 기반으로 여러 리소스를 간결하게 프로비저닝할 수 있도록 설계되었습니다.

다음 코드 예제에서는 라는 AWS CDK 구문을 만듭니다`ExampleConstruct`. 이 구성을 템플릿으로 사용하여 클라우드 구성 요소를 정의할 수 있습니다.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';

export interface ExampleConstructProps {
  //insert properties you wish to expose
}

export class ExampleConstruct extends Construct {
  constructor(scope: Construct, id: string, props: ExampleConstructProps) {
    super(scope, id);
    //Insert the AWS components you wish to integrate
  }
}
```

다음 예제는 AWS CDK 애플리케이션 또는 스택에서 새로 만든 구문을 가져오는 방법을 보여줍니다.

```
import { ExampleConstruct } from './lib/construct-name';
```

다음 예제에서는 기본 클래스에서 확장한 구성의 인스턴스를 인스턴스화하는 방법을 보여줍니다.

```
import { ExampleConstruct } from './lib/construct-name';

new ExampleConstruct(this, 'newConstruct', {
  //insert props which you exposed in the interface `ExampleConstructProps`
});
```

자세한 내용은 워크숍 설명서의 AWS CDKAWS CDK 워크샵을 참조하십시오.

## 이스케이프 해치

의 이스케이프 해치를 사용하여 추상화 수준을 높이면 더 낮은 수준의 구문에 액세스할 수 있습니다. AWS CDK 이스케이프 해치는 현재 버전에서는 제공되지 않지만 에서 사용할 수 있는 기능의 구조를 확장하는 데 사용됩니다. AWS CloudFormation

다음과 같은 시나리오에서는 이스케이프 해치를 사용하는 것이 좋습니다.

- 를 통해 AWS 서비스 기능을 사용할 수 CloudFormation 있지만 이에 대한 `Construct` 구문은 없습니다.
- AWS 서비스 기능을 통해 사용할 수 CloudFormation 있고 서비스를 위한 구문이 있지만 이러한 `Construct` 구조에서는 아직 기능을 노출하지 않습니다. 구문은 “손으로” 개발되기 때문에 때때로 리소스 `Construct` 구성보다 뒤쳐질 수 있습니다. CloudFormation

다음 예제 코드는 이스케이프 해치를 사용하는 일반적인 사용 사례를 보여줍니다. 이 예에서 상위 수준 구성에 아직 구현되지 않은 기능은 `LaunchConfiguration` 자동 크기 조정을 위해 `httpPutResponseHopLimit`을 추가하는 것입니다.

```
const launchConfig = autoscaling.onDemandASG.node.findChild("LaunchConfig") as
 CfnLaunchConfiguration;
            launchConfig.metadataOptions = {
                   httpPutResponseHopLimit: autoscalingConfig.httpPutResponseHopLimit||
 2
            }
```

앞의 코드 예시는 다음 워크플로를 보여줍니다.

1. L2 구성을 사용하여 `AutoScalingGroup`을 정의합니다. L2 구문은 업데이트를 지원하지 않으므로 이스케이프 `httpPutResponseHopLimit` 해치를 사용해야 합니다.
2. L2 `AutoScalingGroup` 구성의 `node.defaultChild` 속성에 액세스하고 이를 `CfnLaunchConfiguration` 리소스로 캐스팅합니다.
3. 이제 L1 `CfnLaunchConfiguration`에서 `launchConfig.metadataOptions`속성을 설정할 수 있습니다.

## 사용자 정의 리소스

사용자 지정 리소스를 사용하여 스택을 생성, 업데이트 (사용자 지정 리소스를 변경한 경우) 또는 삭제할 때마다 CloudFormation 실행되는 사용자 지정 프로비저닝 로직을 템플릿에 작성할 수 있습니다. 예를 들어 에서 사용할 수 없는 리소스를 포함하려는 경우 사용자 지정 리소스를 사용할 수 있습니다. AWS CDK이러한 방식으로 단일 스택에서 관련 리소스를 모두 관리할 수 있습니다.

사용자 지정 리소스를 구축하려면 리소스의 CREATE, UPDATE 및 DELETE 수명 주기 이벤트에 응답하는 Lambda 함수를 작성해야 합니다. 사용자 지정 리소스에서 API를 한 번만 호출해야 하는 경우

AwsCustomResource 구조를 사용해 보세요. 이렇게 하면 배포 중에 임의의 SDK 호출을 수행할 수 CloudFormation 있습니다. 그렇지 않으면 Lambda 함수를 직접 작성하여 필요한 작업을 수행하는 것이 좋습니다.

커스텀 리소스에 대한 자세한 내용은 설명서의 커스텀 리소스를 참조하십시오. CloudFormation 사용자 지정 리소스를 사용하는 방법에 대한 예는 의 사용자 지정 리소스 리포지토리를 참조하십시오 GitHub.

다음 예제는 사용자 지정 리소스 클래스를 생성하여 Lambda 함수를 시작하고 성공 또는 실패 신호를 CloudFormation 보내는 방법을 보여줍니다.

```
import cdk = require('aws-cdk-lib');
import customResources = require('aws-cdk-lib/custom-resources');
import lambda = require('aws-cdk-lib/aws-lambda');
import { Construct } from 'constructs';

import fs = require('fs');

export interface MyCustomResourceProps {
  /**
   * Message to echo
   */
  message: string;
}

export class MyCustomResource extends Construct {
  public readonly response: string;

  constructor(scope: Construct, id: string, props: MyCustomResourceProps) {
    super(scope, id);

    const fn = new lambda.SingletonFunction(this, 'Singleton', {
      uuid: 'f7d4f730-4ee1-11e8-9c2d-fa7ae01bbebc',
      code: new lambda.InlineCode(fs.readFileSync('custom-resource-handler.py',
 { encoding: 'utf-8' })),
      handler: 'index.main',
      timeout: cdk.Duration.seconds(300),
      runtime: lambda.Runtime.PYTHON_3_6,
    });

    const provider = new customResources.Provider(this, 'Provider', {
      onEventHandler: fn,
    });
```

```
    const resource = new cdk.CustomResource(this, 'Resource', {
      serviceToken: provider.serviceToken,
      properties: props,
    });

    this.response = resource.getAtt('Response').toString();
    }
}
```

다음 예제에서는 사용자 지정 리소스의 기본 로직을 보여줍니다.

```
def main(event, context):
    import logging as log
    import cfnresponse
    log.getLogger().setLevel(log.INFO)

    # This needs to change if there are to be multiple resources in the same stack
    physical_id = 'TheOnlyCustomResource'

    try:
        log.info('Input event: %s', event)

        # Check if this is a Create and we're failing Creates
        if event['RequestType'] == 'Create' and
 event['ResourceProperties'].get('FailCreate', False):
            raise RuntimeError('Create failure requested')

        # Do the thing
        message = event['ResourceProperties']['Message']
        attributes = {
            'Response': 'You said "%s"' % message
        }

        cfnresponse.send(event, context, cfnresponse.SUCCESS, attributes, physical_id)
    except Exception as e:
        log.exception(e)
        # cfnresponse's error message is always "see CloudWatch"
        cfnresponse.send(event, context, cfnresponse.FAILED, {}, physical_id)
```

다음 예제는 AWS CDK 스택이 사용자 지정 리소스를 호출하는 방법을 보여줍니다.

```
import cdk = require('aws-cdk-lib');
```

```
import { MyCustomResource } from './my-custom-resource';

/**
 * A stack that sets up MyCustomResource and shows how to get an attribute from it
 */
class MyStack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const resource = new MyCustomResource(this, 'DemoResource', {
      message: 'CustomResource says hello',
    });

    // Publish the custom resource output
    new cdk.CfnOutput(this, 'ResponseMessage', {
      description: 'The message that came back from the Custom Resource',
      value: resource.response
    });
  }
}

const app = new cdk.App();
new MyStack(app, 'CustomResourceDemoStack');
app.synth();
```

# TypeScript 모범 사례를 따르세요.

TypeScript 의 JavaScript 기능을 확장하는 언어입니다. 형식이 강하고 객체 지향적인 언어입니다. 를 TypeScript 사용하여 코드 내에서 전달되는 데이터 유형을 지정할 수 있으며 형식이 일치하지 않을 경우 오류를 보고할 수 있습니다. 이 섹션에서는 TypeScript 모범 사례의 개요를 제공합니다.

## 데이터 설명

코드에 있는 객체 및 함수의 모양을 설명하는 TypeScript 데 사용할 수 있습니다. any 유형을 사용하는 것은 변수 유형 검사를 선택 해제하는 것과 같습니다. 코드에 any를 사용하지 않는 것이 좋습니다. 다음 예를 참고하세요

```
type Result = "success" | "failure"
function verifyResult(result: Result) {
    if (result === "success") {
        console.log("Passed");
```

```
    } else {
        console.log("Failed")
    }
}
```

## enum 사용

enum을 사용하여 명명된 상수 세트를 정의하고 코드베이스에서 재사용할 수 있는 표준을 정의할 수 있습니다. 글로벌 수준에서 enum을 한 번 내보낸 다음 다른 클래스에서 해당 enum을 가져와서 사용하도록 하는 것이 좋습니다. 코드베이스에서 이벤트를 캡처할 수 있는 가능한 작업 세트를 만들고 싶다고 가정해 보겠습니다. TypeScript 숫자 기반 열거형과 문자열 기반 열거형을 모두 제공합니다. 다음 예에서는 열거형을 사용합니다.

```
enum EventType {
    Create,
    Delete,
    Update
}

class InfraEvent {
    constructor(event: EventType) {
        if (event === EventType.Create) {
            // Call for other function
            console.log(`Event Captured :${event}`);
        }
    }
}

let eventSource: EventType = EventType.Create;
const eventExample = new InfraEvent(eventSource)
```

## 사용자 인터페이스

인터페이스는 클래스에 대한 계약입니다. 계약을 생성하는 경우 사용자는 계약을 준수해야 합니다. 다음 예에서는 인터페이스를 사용하여 `props`를 표준화하고 이 클래스를 사용할 때 호출자가 예상 파라미터를 제공하는지 확인합니다.

```
import { Stack, App } from "aws-cdk-lib";
import { Construct } from "constructs";

interface BucketProps {
```

```
    name: string;
    region: string;
    encryption: boolean;

}

class S3Bucket extends Stack {
    constructor(scope: Construct, props: BucketProps) {
        super(scope);
        console.log(props.name);

    }
}
const app = App();
const myS3Bucket = new S3Bucket(app, {
    name: "my-bucket",
    region: "us-east-1",
    encryption: false
})
```

일부 속성은 객체를 처음 생성할 때만 수정할 수 있습니다. 다음 예제와 같이 속성 이름 앞에 `readonly`를 입력하여 이를 지정할 수 있습니다.

```
interface Position {
    readonly latitude: number;
    readonly longitute: number;
}
```

## 인터페이스 확장

인터페이스를 확장하면 인터페이스 간에 속성을 복사할 필요가 없으므로 중복이 줄어듭니다. 또한 코드를 읽는 사람도 애플리케이션의 관계를 쉽게 이해할 수 있습니다.

```
 interface BaseInterface{
    name: string;
  }
  interface EncryptedVolume extends BaseInterface{
      keyName: string;
  }
  interface UnencryptedVolume extends BaseInterface {
      tags: string[];
  }
```

## 빈 인터페이스 사용 금지

잠재적 위험을 유발할 수 있으므로 빈 인터페이스는 피하는 것이 좋습니다. 다음 예제에는 라는 빈 인터페이스가 있습니다. `BucketProps` `myS3Bucket1` 객체와 `myS3Bucket2` 객체 모두 유효하지만 인터페이스가 어떠한 계약도 적용하지 않기 때문에 서로 다른 표준을 따릅니다. 다음 코드는 속성을 컴파일하고 인쇄하지만 이로 인해 애플리케이션에 불일치가 발생합니다.

```
interface BucketProps {}

class S3Bucket implements BucketProps {
    constructor(props: BucketProps){
        console.log(props);
    }
}

const myS3Bucket1 = new S3Bucket({
    name: "my-bucket",
    region: "us-east-1",
    encryption: false,
});

const myS3Bucket2 = new S3Bucket({
    name: "my-bucket",
});
```

## 팩토리 사용

추상 팩토리 패턴에서 인터페이스는 클래스를 명시적으로 지정하지 않고 관련 객체의 팩토리를 만드는 역할을 합니다. 예를 들어, Lambda 함수를 생성하기 위한 Lambda 팩토리를 생성할 수 있습니다. 구성 내에서 새 Lambda 함수를 생성하는 대신 생성 프로세스를 팩토리에 위임합니다. 이 디자인 패턴에 대한 자세한 내용은 Refactoring.Guru 설명서의 추상 팩토리를 참조하십시오 TypeScript.

## 속성에 구조 분해 사용

ECMAScript 6 (ES6) 에 도입된 디스트럭처링은 배열 또는 객체에서 여러 데이터를 추출하여 자체 변수에 할당할 수 있는 JavaScript 기능입니다.

```
const object = {
    objname: "obj",
    scope: "this",
};
```

```
const oName = object.objname;
const oScop = object.scope;

const { objname, scope } = object;
```

## 표준 이름 지정 규칙 정의

이름 지정 규칙을 적용하면 코드베이스의 일관성이 유지되고 변수 이름을 지정하는 방법을 고려할 때 오버헤드가 줄어듭니다. 다음과 같이 하는 것이 좋습니다:

- 변수 및 함수 이름에는 CamelCase를 사용합니다.
- 클래스 이름 PascalCase 및 인터페이스 이름에 사용합니다.
- 인터페이스 멤버에는 camelCase를 사용합니다.
- 유형 이름 및 열거형 PascalCase 이름에 사용합니다.
- CamelCase를 사용하여 파일 이름 지정(예: `ebsVolumes.tsx` 또는 `storage.tsb`)

## var 키워드를 사용하지 마세요.

`let`명령문은 에서 지역 변수를 선언하는 데 사용됩니다. TypeScript 키워드와 비슷하지만 `var` 키워드에 비해 범위 지정에 몇 가지 제한이 있습니다. `var` `let`을 사용하여 블록에 선언된 변수는 해당 블록 내에서만 사용할 수 있습니다. `var`키워드는 블록 범위를 지정할 수 없습니다. 즉, 키워드는 특정 블록(으로 {} 표시됨) 외부에서 액세스할 수 있지만 정의된 함수 외부에서는 액세스할 수 없습니다. 변수를 재선언하고 업데이트할 수 있습니다. `var` 키워드는 사용하지 않는 것이 좋습니다. `var`

## ESLint와 Prettier를 사용해 보세요.

ESLint는 코드를 정적으로 분석하여 문제를 빠르게 찾아냅니다. ESLint를 사용하여 코드의 모양이나 동작 방식을 정의하는 일련의 어설션(린트 규칙이라고 함)을 생성할 수 있습니다. ESLint에는 코드 개선에 도움이 되는 자동 수정자 제안도 있습니다. 마지막으로 ESLint를 사용하여 공유 플러그인에서 린트 규칙을 로드할 수 있습니다.

Prettier는 다양한 프로그래밍 언어를 지원하는 잘 알려진 코드 포맷터입니다. 코드 형식을 수동으로 지정하지 않아도 되도록 Prettier를 사용하여 코드 스타일을 설정할 수 있습니다. 설치 후 `package.json` 파일을 업데이트하고 `npm run format` 및 `npm run lint` 명령을 실행할 수 있습니다.

다음 예제는 프로젝트에 ESLint와 Prettier 포맷터를 활성화하는 방법을 보여줍니다. AWS CDK

```
"scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk",
    "lint": "eslint --ext .js,.ts .",
    "format": "prettier --ignore-path .gitignore --write '**/*.+(js|ts|json)'"
}
```

## 액세스 한정자 사용

의 private 수정자는 가시성을 동일한 클래스로만 TypeScript 제한합니다. 속성 또는 메서드에 private 한정자를 추가하면 동일한 클래스 내에서 해당 속성이나 메서드에 액세스할 수 있습니다.

public 한정자를 사용하면 모든 위치에서 클래스 속성과 메서드에 액세스할 수 있습니다. 속성 및 메서드에 액세스 한정자를 지정하지 않으면 기본적으로 public 한정자를 사용합니다.

protected 한정자를 사용하면 같은 클래스와 하위 클래스 내에서 클래스의 속성과 메서드에 액세스할 수 있습니다. 애플리케이션에서 서브클래스를 생성하려는 경우 protected 수정자를 사용하십시오. AWS CDK

## 유틸리티 유형 사용

의 유틸리티 유형은 TypeScript 기존 유형에 대한 변환 및 연산을 수행하는 미리 정의된 유형 함수입니다. 이를 통해 기존 유형을 기반으로 새 유형을 만들 수 있습니다. 예를 들어, 속성을 변경 또는 추출하고, 속성을 선택 또는 필수로 만들거나, 변경할 수 없는 버전의 유형을 만들 수 있습니다. 유틸리티 유형을 사용하면 보다 정확한 유형을 정의하고 컴파일 시 잠재적 오류를 포착할 수 있습니다.

### 부분 <Type>

`Partial`입력 유형의 모든 멤버를 선택 `Type` 사항으로 표시합니다. 이 유틸리티는 지정된 형식의 모든 하위 집합을 나타내는 형식을 반환합니다. 다음은 `Partial`의 예제입니다.

```
interface Dog {
  name: string;
  age: number;
  breed: string;
  weight: number;
}
```

```
let partialDog: Partial<Dog> = {};
```

### 필수 <Type>

Required그 반대입니다Partial. 입력 유형의 모든 멤버를 Type 선택 사항이 아닌 (즉, 필수) 로 만듭니다. 다음은 Required의 예제입니다.

```
interface Dog {
  name: string;
  age: number;
  breed: string;
  weight?: number;
}

let dog: Required<Dog> = {
  name: "scruffy",
  age: 5,
  breed: "labrador",
  weight 55 // "Required" forces weight to be defined
};
```

## 보안 취약성 및 형식 오류 스캔

코드형 인프라(IaC)와 자동화는 기업의 필수 요소가 되었습니다. IaC가 매우 강력하기 때문에 보안 위험을 관리해야 할 책임이 커집니다. 일반적인 IaC 보안 위험에는 다음이 포함될 수 있습니다.

- 과허용 (IAM) 권한 AWS Identity and Access Management
- 보안 그룹 열기
- 암호화되지 않은 리소스
- 액세스 로그가 켜지지 않음

### 보안 접근 방식 및 도구

다음 보안 접근 방식을 이행하는 것이 좋습니다.

- 개발 중인 취약성 탐지 - 소프트웨어 패치의 개발 및 배포가 복잡하기 때문에 프로덕션 환경에서 취약성을 해결하는 데는 많은 비용과 시간이 소요됩니다. 또한 프로덕션 환경의 취약성은 악용의 위험을 수반합니다. 프로덕션에 출시하기 전에 취약성을 탐지하고 수정할 수 있도록 IaC 리소스에서 코드 스캔을 사용하는 것이 좋습니다.

- 규정 준수 및 자동 수정 — AWS 관리형 AWS Config 규칙을 제공합니다. 이러한 규칙을 사용하면 규정 준수를 적용하고 자동화를 사용하여 자동 치료를 시도할 수 있습니다.AWS Systems Manager 규칙을 사용하여 사용자 지정 자동화 문서를 만들고 연결할 수도 있습니다. AWS Config

## 일반 개발 도구

이 섹션에서 다루는 도구는 자체 사용자 지정 규칙을 사용하여 기본 제공 기능을 확장하는 데 도움이 됩니다. 사용자 지정 규칙을 조직의 표준에 맞추는 것이 좋습니다. 고려해야 할 몇 가지 일반적인 개발 도구는 다음과 같습니다.

- cfn-nag를 사용하여 템플릿에서 허용적인 IAM 규칙 또는 암호 리터럴과 같은 인프라 보안 문제를 식별할 수 있습니다. CloudFormation 자세한 내용은 Stelligent의 cfn-nag 리포지토리를 참조하십시오. GitHub
- cfn-nag에서 영감을 받은 cdk-nag를 사용하여 주어진 범위 내의 구성이 정의된 규칙 세트를 준수하는지 검증합니다. 규칙 억제 및 규정 준수 보고에도 cdk-nag를 사용할 수 있습니다. cdk-nag 도구는의 여러 측면을 확장하여 구문을 검증합니다. AWS CDK자세한 내용은 블로그의 및 cdk-nag를 사용한 애플리케이션 보안 및 규정 준수 관리를 참조하십시오. AWS Cloud Development Kit (AWS CDK) AWS DevOps
- 오픈 소스 도구인 Checkov를 사용하여 IaC 환경에서 정적 분석을 수행합니다. Checkov는 Kubernetes, Terraform 또는 에서 인프라 코드를 스캔하여 클라우드 구성 오류를 식별하는 데 도움이 됩니다. CloudFormation Checkov를 사용하여 JSON, JUnit XML 또는 CLI를 비롯한 다양한 형식의 출력을 가져올 수 있습니다. Checkov는 동적 코드 종속성을 보여주는 그래프를 작성하여 변수를 효과적으로 처리할 수 있습니다. 자세한 내용은 Bridgecrew의 Checkov 리포지토리를 참조하십시오. GitHub
- TFlint를 사용하여 오류와 더 이상 사용되지 않는 구문을 확인하고 모범 사례를 적용할 수 있습니다. 참고로 TFlint는 제공업체별 문제를 검증하지 못할 수도 있습니다. TFlint에 대한 자세한 내용은 Terraform Linters의 TFlint 리포지토리를 참조하십시오. GitHub
- CodeWhisperer Amazon을 사용하여 보안 스캔을 수행하십시오. CodeWhisperer 실시간으로 코드 제안을 생성할 수 있는 AI 기반 생산성 도구입니다. 이를 통해 보안 문제를 식별하고, 모범 사례를 따르고, 코드 배포의 생산성을 높일 수 있습니다.

# 설명서 개발 및 세부 조정

설명서는 프로젝트 성공에 매우 중요합니다. 설명서는 코드 작동 방식을 설명할 뿐만 아니라 개발자가 애플리케이션의 특성과 기능을 더 잘 이해하는 데도 도움이 됩니다. 고품질 설명서를 개발하고 개선하

면 소프트웨어 개발 프로세스를 강화하고, 고품질 소프트웨어를 유지하고, 개발자 간의 지식 전달에 도움이 될 수 있습니다.

설명서에는 코드 내 문서화와 코드에 대한 지원 문서라는 두 가지 범주가 있습니다. 코드 내 문서는 주석 형식입니다. 코드에 대한 지원 문서는 README 파일 및 외부 문서일 수 있습니다. 코드 자체가 이해하기 쉽기 때문에 개발자가 문서를 오버헤드로 생각하는 경우는 흔치 않습니다. 소규모 프로젝트의 경우에는 그럴 수 있지만, 여러 팀이 참여하는 대규모 프로젝트에서는 설명서가 매우 중요합니다.

코드 작성자는 문서의 기능을 잘 이해하고 있으므로 문서를 작성하는 것이 가장 좋습니다. 개발자는 별도의 지원 설명서를 유지 관리해야 하는 추가 오버헤드로 인해 어려움을 겪을 수 있습니다. 이 문제를 해결하기 위해 개발자는 코드에 주석을 추가하면 해당 주석을 자동으로 추출하여 모든 버전의 코드와 문서를 동기화할 수 있습니다.

개발자가 코드에서 주석을 추출하고 이에 대한 문서를 생성하는 데 도움이 되는 다양한 도구가 있습니다. 이 가이드에서는 AWS CDK 구문을 위한 기본 TypeDoc 도구로서의 용도에 초점을 맞추고 있습니다.

## 구문에 코드 문서가 필요한 이유 AWS CDK

AWS CDK 공통 구문은 조직의 여러 팀에서 작성하여 여러 팀 간에 공유하여 사용합니다. 좋은 설명서는 구성 라이브러리 소비자가 최소한의 노력으로 구성을 쉽게 통합하고 인프라를 구축하는 데 도움이 됩니다. 모든 문서를 동기화하는 것은 큰 작업입니다. TypeDoc라이브러리를 사용하여 추출되는 코드 내에 문서를 보관하는 것이 좋습니다.

## AWS 구성 라이브러리와 TypeDoc 함께 사용

TypeDoc 를 위한 문서 생성기입니다 TypeScript. 를 TypeDoc 사용하여 TypeScript 소스 파일을 읽고 해당 파일의 주석을 파싱한 다음 코드에 대한 설명서가 포함된 정적 사이트를 생성할 수 있습니다.

다음 코드는 AWS Construct TypeDoc Library와 통합한 다음 `package.json` 파일에 다음 패키지를 추가하는 방법을 보여줍니다. `devDependencies`

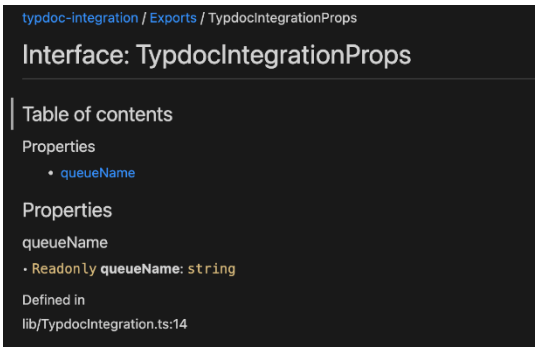
```
{

  "devDependencies": {

    "typedoc-plugin-markdown": "^3.11.7",
    "typescript": "~3.9.7"
  },

}
```

CDK 라이브러리 폴더에서 typedoc.json을 추가하려면 다음 코드를 사용하세요.

```
{
    "$schema": "https://typedoc.org/schema.json",
    "entryPoints": ["./lib"],
}
```

README 파일을 생성하려면 AWS CDK 구성 라이브러리 프로젝트의 루트 디렉터리에서 npx typedoc 명령을 실행합니다.

에서 생성한 TypeDoc 샘플 문서는 다음과 같습니다.




TypeDoc 통합 옵션에 대한 자세한 내용은 TypeDoc 설명서의 문서 설명을 참조하십시오.

## 테스트 기반 개발 접근 방식 채택

와 함께 테스트 주도 개발 (TDD) 접근 방식을 따르는 것이 좋습니다. AWS CDK TDD는 테스트 사례를 개발하여 코드를 지정하고 검증하는 소프트웨어 개발 접근 방식입니다. 간단히 말해, 먼저 각 기능에 대한 테스트 사례를 만들고 테스트가 실패할 경우 테스트를 통과할 수 있도록 새 코드를 작성하여 코드를 간단하고 버그 없이 만듭니다.

TDD를 사용하여 테스트 사례를 먼저 작성할 수 있습니다. 이를 통해 리소스에 보안 정책을 적용하고 프로젝트의 고유한 명명 규칙을 따르는 측면에서 다양한 설계 제약이 있는 인프라를 검증할 수 있습니다. AWS CDK 응용 프로그램을 테스트하는 표준 접근 방식은 AWS CDK어설션 모듈과 널리 사용되는 테스트 프레임워크 (예: Jest for and 및 Python 또는 JavaScript pytest for TypeScript Pytest) 를 사용하는 것입니다.

애플리케이션용으로 작성할 수 있는 테스트에는 두 가지 범주가 있습니다. AWS CDK

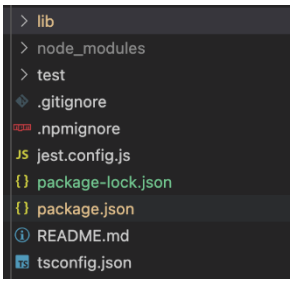
- 세분화된 어설션을 사용하여 생성된 CloudFormation 템플릿의 특정 측면을 테스트하십시오 (예: “이 리소스는 이 값을 가진 이 속성을 가짐”). 이러한 테스트는 회귀를 탐지할 수 있으며 TDD를 사용하

여 새 기능을 개발할 때 유용합니다(먼저 테스트를 작성한 다음 올바른 구현을 작성하여 통과하도록 함). 세분화된 어설션이 가장 많이 작성하게 될 테스트입니다.

- 스냅샷 테스트를 사용하여 이전에 저장된 기준 템플릿과 비교하여 합성된 CloudFormation 템플릿을 테스트할 수 있습니다. 스냅샷 테스트를 통해 리팩터링된 코드가 원본과 정확히 동일한 방식으로 작동하는지 확인할 수 있으므로 자유롭게 리팩터링할 수 있습니다. 의도적으로 변경한 경우 향후 테스트에 사용할 새 기준선을 적용할 수 있습니다. 하지만 AWS CDK 업그레이드로 인해 합성된 템플릿도 변경될 수 있으므로 구현이 올바른지 확인하기 위해 스냅샷에만 의존할 수는 없습니다.

## 유닛 테스트

이 가이드는 TypeScript 특히 단위 테스트 통합에 중점을 둡니다. 테스트를 활성화하려면 `package.json` 파일에 `devDependencies`에 `@types/jest`, `jest` 및 `ts-jest` 라이브러리가 있는지 확인합니다. 이 패키지를 추가하려면 `cdk init lib --language=typescript` 명령을 실행합니다. 이전 명령을 실행하면 다음 구조가 표시됩니다.




다음 코드는 Jest 라이브러리로 활성화된 `package.json` 파일의 예입니다.

```
{
  ...
  "scripts": {
    "build": "npm run lint && tsc",
    "watch": "tsc -w",
    "test": "jest",
  },
  "devDependencies": {
    ...
    "@types/jest": "27.5.2",
    "jest": "27.5.1",
    "ts-jest": "27.1.5",
    ...
  }
}
```

Test 폴더에서 테스트 사례를 작성할 수 있습니다. 다음 예제는 AWS CodePipeline 구문의 테스트 케이스를 보여줍니다.

```
import {App,Stack} from 'aws-cdk-lib';
import { Template } from 'aws-cdk-lib/assertions';
import * as CodepipelineModule from '../lib/index';
import { Role, ServicePrincipal } from 'aws-cdk-lib/aws-iam';
import { Repository } from 'aws-cdk-lib/aws-codecommit';
import { PipelineProject } from 'aws-cdk-lib/aws-codebuild';

const testData:CodepipelineModule.CodepipelineModuleProps = {

    pipelineName: "validate-test-pipeline",
    serviceRoleARN: "",
    codeCommitRepositoryARN: "",
    branchName: "master",
    buildStages:[]
}


test('Code Pipeline Created', () => {
    const app = new App();
    const stack = new Stack(app, "TestStack");
    // WHEN
    const serviceRole = new Role(stack, "testRole", { assumedBy: new
 ServicePrincipal('codepipeline.amazonaws.com') })
    const codeCommit = new Repository(stack, "testRepo", {
        repositoryName: "validate-codeCommit-repo"
    });
    const codeBuildProject=new PipelineProject(stack,"TestCodeBuildProject",{});
    testData.serviceRoleARN = serviceRole.roleArn;
    testData.codeCommitRepositoryARN = codeCommit.repositoryArn;
    testData["buildStages"].push({
        stageName:"Deploy",
        codeBuildProject:codeBuildProject
    })
    new CodepipelineModule.CodepipelineModule(stack, 'MyTestConstruct', testData);
    // THEN
    const template = Template.fromStack(stack);

    template.hasResourceProperties('AWS::CodePipeline::Pipeline', {
        Name:testData.pipelineName
    });
```

```
});
```

테스트를 실행하려면 프로젝트에서 `npm run test` 명령을 실행합니다. 이 테스트는 다음과 같은 결과를 반환합니다.

```
PASS  test/codepipeline-module.test.ts (5.972 s)
  # Code Pipeline Created (97 ms)
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        6.142 s, estimated 9 s
```

테스트 사례에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 구문 테스트를 참조하십시오.

### 통합 테스트

`integ-tests`모듈을 사용하여 AWS CDK 구문에 대한 통합 테스트를 포함할 수도 있습니다. 통합 테스트는 AWS CDK 애플리케이션으로 정의해야 합니다. 통합 테스트와 AWS CDK 애플리케이션 간에는 one-to-one 관계가 있어야 합니다. 자세한 내용은 AWS CDK API Reference의 integ-tests-alpha 모듈을 참조하십시오.

## 구성에 릴리스 및 버전 제어 사용

### 에 대한 버전 제어 AWS CDK

AWS CDK 여러 팀에서 공통 구성을 작성하고 조직 전체에서 공유하여 사용할 수 있습니다. 일반적으로 개발자는 공통 구문의 새 기능이나 버그 수정을 릴리스합니다 AWS CDK . 이러한 구문은 AWS CDK 응용 프로그램이나 다른 기존 AWS CDK 구조에서 종속성의 일부로 사용됩니다. 이러한 이유로 개발자는 적절한 의미 체계 버전으로 구성을 독립적으로 업데이트하고 릴리스하는 것이 중요합니다. 다운스트림 AWS CDK 응용 프로그램 또는 기타 AWS CDK 구문은 새로 릴리스된 구문 버전을 사용하도록 종속성을 업데이트할 수 있습니다. AWS CDK

의미 체계 버전 관리(Semver)는 컴퓨터 소프트웨어에 고유한 소프트웨어 번호를 제공하기 위한 일련의 규칙 또는 방법입니다. 버전은 다음과 같이 정의됩니다.

- MAJOR 버전은 호환되지 않는 API 변경 또는 주요 변경 사항으로 구성됩니다.
- MINOR 버전은 이전 버전과 호환되는 방식으로 추가된 기능으로 구성되어 있습니다.
- PATCH 버전은 이전 버전과 호환되는 버그 수정으로 구성됩니다.

시맨틱 버전 관리에 대한 자세한 내용은 시맨틱 버전 관리 설명서의 시맨틱 버전 지정 사양 () 을 참조하십시오. SemVer

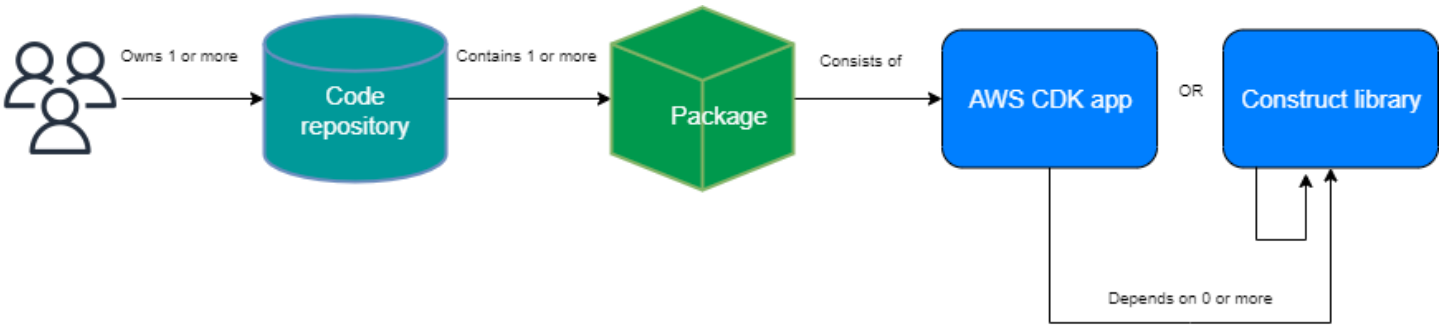
## AWS CDK 구문을 위한 리포지토리 및 패키징

AWS CDK 구성은 여러 팀에서 개발되고 여러 AWS CDK 응용 프로그램에서 사용되므로 각 AWS CDK 구성에 대해 별도의 저장소를 사용할 수 있습니다. 이는 액세스 제어를 강화하는 데도 도움이 될 수 있습니다. 각 리포지토리에는 동일한 AWS CDK 구문과 관련된 모든 소스 코드와 모든 종속성이 포함될 수 있습니다. 단일 응용 프로그램 (즉, AWS CDK 구조) 을 단일 리포지토리에 보관하면 배포 중 변경의 영향 범위를 줄일 수 있습니다.

인프라 배포를 위한 CloudFormation 템플릿을 생성할 뿐만 AWS CDK 아니라 Lambda 함수 및 Docker 이미지와 같은 런타임 자산을 번들로 묶어 인프라와 함께 배포합니다. 인프라를 정의하는 코드와 런타임 로직을 구현하는 코드를 단일 구조로 결합하는 것이 가능할 뿐만 아니라 모범 사례입니다. 이 두 종류의 코드는 별도의 리포지토리나 패키지에 있을 필요가 없습니다.

리포지토리 경계를 넘어 패키지를 사용하려면 npm 또는 Maven Central과 비슷하지만 조직 내부에 있는 개인 패키지 리포지토리가 있어야 합니다. PyPi 또한 패키지를 빌드하고 테스트하여 개인 패키지 리포지토리에 게시하는 릴리스 프로세스가 있어야 합니다. 로컬 가상 머신 (VM) 또는 Amazon S3를 사용하여 PyPi 서버와 같은 프라이빗 리포지토리를 생성할 수 있습니다. 프라이빗 패키지 레지스트리를 설계하거나 생성할 때는 고가용성 및 확장성으로 인한 서비스 중단의 위험을 고려하는 것이 중요합니다. 패키지를 저장하기 위해 클라우드에 호스팅되는 서버리스 관리형 서비스는 유지 관리 오버헤드를 크게 줄일 수 있습니다. 예를 들어, 가장 많이 사용되는 프로그래밍 언어의 패키지를 AWS CodeArtifact호스팅하는 데 사용할 수 있습니다. 를 CodeArtifact 사용하여 외부 리포지토리 연결을 설정하고 내부에 CodeArtifact 복제할 수도 있습니다.

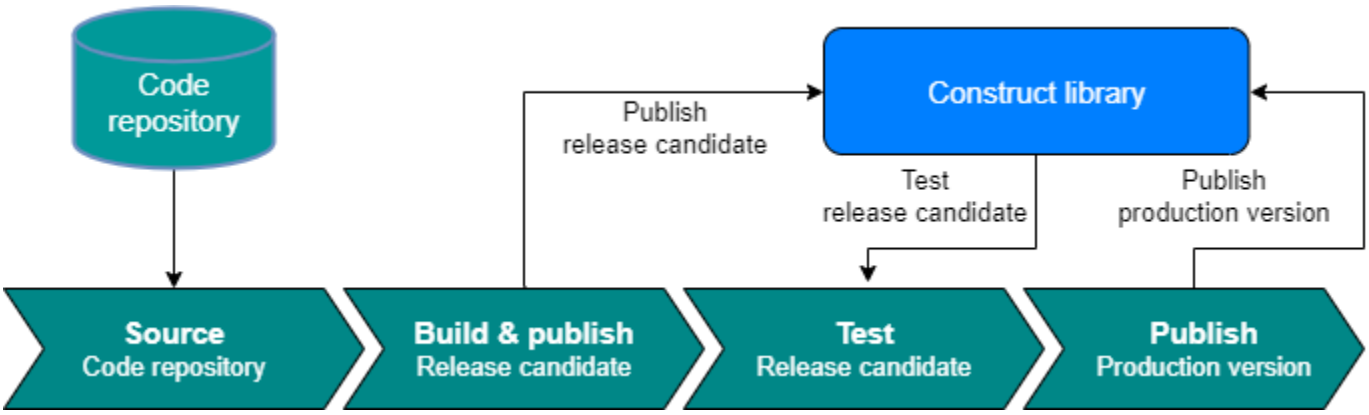
패키지 저장소의 패키지에 대한 종속성은 해당 언어의 패키지 관리자 (예: npm for TypeScript or JavaScript application) 에서 관리합니다. 패키지 관리자는 애플리케이션이 의존하는 모든 패키지의 특정 버전을 기록하여 빌드를 반복할 수 있도록 한 후 다음 다이어그램과 같이 제어된 방식으로 이러한 종속성을 업그레이드할 수 있도록 합니다.



# 를 위한 컨스트럭트 릴리즈 AWS CDK

자동화된 파이프라인을 직접 생성하여 새 AWS CDK 구성 버전을 빌드하고 릴리스하는 것이 좋습니다. 적절한 풀 요청 승인 프로세스를 마련하면 소스 코드를 커밋하여 리포지토리의 기본 브랜치로 푸시하면 파이프라인이 릴리스 후보 버전을 빌드하고 생성할 수 있습니다. 프로덕션 준비 버전을 출시하기 전에 해당 버전을 CodeArtifact 푸시하고 테스트할 수 있습니다. 코드를 기본 브랜치에 병합하기 전에 새 AWS CDK 구성 버전을 로컬에서 테스트할 수도 있습니다. 이로 인해 파이프라인은 프로덕션 지원 버전을 출시합니다. 공유 구성 및 패키지는 일반 사용자에게 공개되는 것처럼 사용 중인 애플리케이션과 독립적으로 테스트해야 한다는 점을 고려하세요.

다음 다이어그램은 샘플 AWS CDK 버전 출시 파이프라인을 보여줍니다.




다음 샘플 명령을 사용하여 npm 패키지를 빌드, 테스트 및 게시할 수 있습니다. 먼저 다음 명령을 실행하여 아티팩트 리포지토리에 로그인합니다.

```
aws codeartifact login --tool npm --domain <Domain Name> --domain-owner $(aws sts get-
caller-identity --output text --query 'Account')  \
--repository <Repository Name> --region <AWS Region Name>
```

선택한 후 다음 단계를 완료합니다.

1. `package.json` 파일을 기반으로 필요한 패키지 설치: `npm install`
2. 릴리스 후보 버전 생성: `npm version prerelease --preid rc`
3. npm 패키지 빌드: `npm run build`
4. npm 패키지 테스트: `npm run test`
5. npm 패키지 게시: `npm publish`

# 라이브러리 버전 관리 적용

AWS CDK 코드 베이스를 유지 관리할 때 수명 주기 관리는 매우 어려운 과제입니다. 예를 들어 버전 1.97로 AWS CDK 프로젝트를 시작한 후 나중에 버전 1.169를 사용할 수 있게 된다고 가정해 보겠습니다. 버전 1.169는 새로운 기능과 버그 수정을 제공하지만 이전 버전을 사용하여 인프라를 배포했습니다. 이제 새 버전에 도입될 수 있는 주요 변경 사항으로 인해 이러한 격차가 커짐에 따라 구성을 업데이트하는 것이 점점 어려워지고 있습니다. 환경에 리소스가 많으면 이것이 문제가 될 수 있습니다. 이 섹션에 소개된 패턴은 자동화를 사용하여 AWS CDK 라이브러리 버전을 관리하는 데 도움이 될 수 있습니다. 이 패턴의 워크플로는 다음과 같습니다.

1. 새 CodeArtifact Service Catalog 제품을 시작하면 AWS CDK 라이브러리 버전과 해당 종속성이 `package.json` 파일에 저장됩니다.
2. 주요 변경 사항이 없는 경우 리포지토리에 자동 업그레이드를 적용할 수 있도록 모든 리포지토리를 추적하는 공통 파이프라인을 배포합니다.
3. AWS CodeBuild 스테이지는 종속성 트리를 확인하고 주요 변경 사항을 찾습니다.
4. 파이프라인은 기능 분기를 생성한 다음 새 버전으로 `cdk synth`를 실행하여 오류가 없는지 확인합니다.
5. 새 버전이 테스트 환경에 배포되고 마지막으로 통합 테스트를 실행하여 배포가 정상인지 확인합니다.
6. Amazon Simple Block Store(Amazon SQS) 대기열 2개를 사용하여 스택을 추적할 수 있습니다. 사용자는 예외 대기열에서 스택을 수동으로 검토하고 주요 변경 사항을 해결할 수 있습니다. 통합 테스트를 통과한 항목은 병합하고 릴리스할 수 있습니다.

# FAQ

## 어떤 문제를 TypeScript 해결할 수 있나요?

일반적으로 자동화된 테스트를 작성하고 코드가 예상대로 작동하는지 수동으로 확인한 다음 다른 사람이 코드를 검증하도록 하면 코드의 버그를 제거할 수 있습니다. 프로젝트의 모든 부분 간의 연결을 검증하는 데 시간이 많이 걸립니다. 검증 프로세스의 속도를 높이기 위해 코드 검증을 자동화하고 개발 중에 즉각적인 피드백을 제공하는 등의 TypeScript 유형 검사 언어를 사용할 수 있습니다.

## 사용해야 하는 이유는 무엇인가요? TypeScript

TypeScript JavaScript 코드를 단순화하여 읽고 디버그하기 쉽게 만드는 오픈 소스 언어입니다. TypeScript 정적 검사와 같은 JavaScript IDE 및 실습을 위한 생산성이 높은 개발 도구도 제공합니다. 또한 ECMAScript 6 (ES6) 의 이점을 TypeScript 제공하고 생산성을 높일 수 있습니다. 마지막으로, TypeScript 코드를 유형 JavaScript 검사하여 작성할 때 개발자가 흔히 겪는 골치 아픈 버그를 방지할 수 있습니다.

## AWS CDKOR를 써야 하나요 CloudFormation?

조직에 이를 활용할 수 있는 개발 전문 지식이 있는 AWS CloudFormation 경우 AWS Cloud Development Kit (AWS CDK) 대신 를 사용하는 것이 좋습니다AWS CDK. 프로그래밍 언어와 OOP 개념을 사용할 수 있기 때문에 AWS CDK 가 보다 CloudFormation 유연하기 때문입니다. 를 사용하면 AWS 리소스를 순서대로, 예측 가능한 방식으로 생성할 수 있다는 점을 염두에 두세요. CloudFormation CloudFormation에서는 리소스가 JSON 또는 YAML 형식을 사용하여 텍스트 파일에 작성됩니다.

## 새로 출시된 AWS CDK 버전을 지원하지 않으면 어떻게 되나요? AWS 서비스

원시 오버라이드 또는 CloudFormation 사용자 지정 리소스를 사용할 수 있습니다.

## 에서 지원하는 다양한 프로그래밍 언어는 무엇입니까? AWS CDK

AWS CDK는 일반적으로, Python JavaScript TypeScript, Java, C# 및 Go (개발자 미리 보기) 에서 사용할 수 있습니다.

# AWS CDK비용은 얼마나 드나요?

에 대한 추가 비용은 없습니다AWS CDK. 수동으로 생성한 것과 동일한 AWS CDK 방식으로 사용할 때 생성된 AWS 리소스 (예: Amazon EC2 인스턴스 또는 Elastic Load Balancing 로드 밸런서) 에 대한 비용을 지불하면 됩니다. 사용하는 것에 대해서만 사용할 때 지불합니다. 최소 요금과 필수 선수금은 없습니다.

# 다음 단계

in으로 AWS Cloud Development Kit (AWS CDK) 빌드를 시작하는 것이 좋습니다 TypeScript. 자세한 내용은 AWS CDK 몰입 데이 워크숍을 참조하십시오.

# 리소스

## 참조

- AWS 솔루션 구성 (AWS 솔루션)
- AWS 클라우드 개발 키트 (AWS CDK) (GitHub)
- AWS 구성 라이브러리 API 참조 (AWS CDK 참조 문서)
- AWS CDK 참조 문서 (AWS CDK 참조 문서)
- AWS CDK 몰입 데이 워크숍 (AWS 워크샵 스튜디오)

## 도구

- cdk-nag () GitHub
- TypeScript ESLint (ESLint 문서) TypeScript

## 가이드 및 패턴

- AWS 솔루션 구성 패턴 (문서)AWS

# 문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

| 변경 사항 | 설명 | 날짜 |
| --- | --- | --- |
| 코드 업데이트 | TypeScript 모범 사례 따르기 섹션의 코드 샘플을 업데이트했습니다. | 2024년 2월 16일 |
| 섹션 추가 | 유틸리티 유형 사용 및 통합 테스트 섹션을 추가했습니다. | 2024년 1월 10일 |
| 마이너 업데이트 | L3 구성 생성에 대한 코드 예제를 업데이트했습니다. | 2023년 6월 16일 |
| 최초 게시 | — | 2022년 12월 8일 |

# AWS 규범적 지침 용어집

다음은 규범적 지침에서 제공하는 AWS 전략, 가이드 및 패턴에서 일반적으로 사용되는 용어입니다. 용어집 항목을 제안하려면 용어집 끝에 있는 피드백 제공 링크를 사용하십시오.

## 숫자

7가지 전략

애플리케이션을 클라우드로 이전하기 위한 7가지 일반적인 마이그레이션 전략 이러한 전략은 Gartner가 2011년에 파악한 5가지 전략을 기반으로 하며 다음으로 구성됩니다.

- 리팩터링/리아키텍트 - 클라우드 네이티브 기능을 최대한 활용하여 애플리케이션을 이동하고 해당 아키텍처를 수정함으로써 민첩성, 성능 및 확장성을 개선합니다. 여기에는 일반적으로 운영 체제와 데이터베이스 이식이 포함됩니다. 예를 들어, 온프레미스 Oracle 데이터베이스를 Amazon Aurora PostgreSQL 호환 버전으로 마이그레이션합니다.
- 리플랫포밍(리프트 앤드 리셰이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 클라우드에서 오라클용 Amazon RDS (Amazon RDS) 로 마이그레이션합니다. AWS
- 재구매(드롭 앤드 숍) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예를 들어, 고객 관계 관리(CRM) 시스템을 Salesforce.com으로 마이그레이션합니다.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예: 클라우드의 EC2 인스턴스에서 온프레미스 Oracle 데이터베이스를 Oracle로 마이그레이션합니다. AWS
- 재배치(하이퍼바이저 수준의 리프트 앤 시프트) - 새 하드웨어를 구매하거나, 애플리케이션을 다시 작성하거나, 기존 운영을 수정하지 않고도 인프라를 클라우드로 이동합니다. 이 마이그레이션 시나리오는 온프레미스 환경과 환경 간의 가상 머신 (VM) 호환성 및 워크로드 이동성을 지원하는 VMware Cloud AWS on에만 해당됩니다. AWS인프라를 AWS의 VMware Cloud로 마이그레이션할 때 온프레미스 데이터 센터에서 VMware Cloud Foundation 기술을 사용할 수 있습니다. 예: Oracle 데이터베이스를 호스팅하는 하이퍼바이저를 VMware Cloud on으로 재배치합니다. AWS
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중으로 연기하려는 애플리케이션과 비즈니스 차원에서 마이그레이션할 이유가 없어 유지하려는 레거시 애플리케이션이 여기에 포함될 수 있습니다.
- 사용 중지 - 소스 환경에서 더 이상 필요하지 않은 애플리케이션을 폐기하거나 제거합니다.

# A

ABAC

속성 기반 액세스 제어를 참조하십시오.

추상화된 서비스

관리형 서비스를 참조하십시오.

산

원자성, 일관성, 격리성, 내구성을 참조하십시오.

능동-능동 마이그레이션

양방향 복제 도구 또는 이중 쓰기 작업을 사용하여 소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되고, 두 데이터베이스 모두 마이그레이션 중 연결 애플리케이션의 트랜잭션을 처리하는 데이터베이스 마이그레이션 방법입니다. 이 방법은 일회성 전환이 필요한 대신 소규모의 제어된 배치로 마이그레이션을 지원합니다. 더 유연하지만 액티브-패시브 마이그레이션보다 더 많은 작업이 필요합니다.

능동-수동 마이그레이션

소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되지만 소스 데이터베이스만 연결 애플리케이션의 트랜잭션을 처리하고 데이터는 대상 데이터베이스로 복제되는 데이터베이스 마이그레이션 방법입니다. 대상 데이터베이스는 마이그레이션 중 어떤 트랜잭션도 허용하지 않습니다.

집계 함수

행 그룹에서 연산을 수행하고 그룹에 대한 단일 반환값을 계산하는 SQL 함수입니다. 집계 함수의 예로는 및 등이 `SUM` 있습니다. `MAX`

AI

인공 지능을 참조하십시오.

AIOps

인공 지능 운영을 참조하십시오.

익명화

데이터세트에서 개인 정보를 영구적으로 삭제하는 프로세스입니다. 익명화는 개인 정보 보호에 도움이 될 수 있습니다. 익명화된 데이터는 더 이상 개인 데이터로 간주되지 않습니다.

안티 패턴

솔루션이 다른 솔루션보다 비생산적이거나 비효율적이거나 덜 효과적이어서 반복되는 문제에 자주 사용되는 솔루션입니다.

애플리케이션 제어

시스템을 멀웨어로부터 보호하기 위해 승인된 애플리케이션만 사용할 수 있는 보안 접근 방식입니다.

애플리케이션 포트폴리오

애플리케이션 구축 및 유지 관리 비용과 애플리케이션의 비즈니스 가치를 비롯하여 조직에서 사용하는 각 애플리케이션에 대한 세부 정보 모음입니다. 이 정보는 포트폴리오 검색 및 분석 프로세스의 핵심이며 마이그레이션, 현대화 및 최적화할 애플리케이션을 식별하고 우선순위를 정하는 데 도움이 됩니다.

인공 지능

컴퓨터 기술을 사용하여 학습, 문제 해결, 패턴 인식 등 일반적으로 인간과 관련된 인지 기능을 수행하는 것을 전문으로 하는 컴퓨터 과학 분야입니다. 자세한 내용은 What is Artificial Intelligence?를 참조하십시오.

인공 지능 운영(AIOps)

기계 학습 기법을 사용하여 운영 문제를 해결하고, 운영 인시던트 및 사용자 개입을 줄이고, 서비스 품질을 높이는 프로세스입니다. AWS 마이그레이션 전략에서 AIOps가 사용되는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

비대칭 암호화

한 쌍의 키, 즉 암호화를 위한 퍼블릭 키와 복호화를 위한 프라이빗 키를 사용하는 암호화 알고리즘입니다. 퍼블릭 키는 복호화에 사용되지 않으므로 공유할 수 있지만 프라이빗 키에 대한 액세스는 엄격히 제한되어야 합니다.

원자성, 일관성, 격리성, 내구성(ACID)

오류, 정전 또는 기타 문제가 발생한 경우에도 데이터베이스의 데이터 유효성과 운영 신뢰성을 보장하는 소프트웨어 속성 세트입니다.

ABAC(속성 기반 액세스 제어)

부서, 직무, 팀 이름 등의 사용자 속성을 기반으로 세분화된 권한을 생성하는 방식입니다. 자세한 내용은 AWS Identity and Access Management (IAM) 설명서의 AWS ABAC for를 참조하십시오.

신뢰할 수 있는 데이터 소스

가장 신뢰할 수 있는 정보 소스로 간주되는 기본 버전의 데이터를 저장하는 위치입니다. 익명화, 편집 또는 가명화와 같은 데이터 처리 또는 수정의 목적으로 신뢰할 수 있는 데이터 소스의 데이터를 다른 위치로 복사할 수 있습니다.

가용 영역

다른 가용 영역의 장애로부터 격리되고 동일한 지역 내 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 별도의 위치입니다. AWS 리전

AWS 클라우드 채택 프레임워크 (AWS CAF)

조직이 클라우드로 성공적으로 AWS 전환하기 위한 효율적이고 효과적인 계획을 개발하는 데 도움이 되는 지침 및 모범 사례 프레임워크입니다. AWS CAF는 지침을 관점이라고 하는 6가지 중점 영역, 즉 비즈니스, 사람, 거버넌스, 플랫폼, 보안, 운영으로 분류합니다. 비즈니스, 사람 및 거버넌스 관점은 비즈니스 기술과 프로세스에 초점을 맞추고, 플랫폼, 보안 및 운영 관점은 전문 기술과 프로세스에 중점을 둡니다. 예를 들어, 사람 관점은 인사(HR), 직원 배치 기능 및 인력 관리를 담당하는 이해관계자를 대상으로 합니다. 이러한 관점에서 AWS CAF는 조직이 성공적인 클라우드 채택을 준비할 수 있도록 인력 개발, 교육 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 AWS CAF 웹 사이트와 AWS CAF 백서를 참조하십시오.

AWS 워크로드 검증 프레임워크 (AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고, 마이그레이션 전략을 권장하고, 작업 예상치를 제공하는 도구입니다. AWS WQF는 () 에 포함됩니다. AWS Schema Conversion Tool AWS SCT데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

# B

배드 봇

개인이나 조직을 방해하거나 피해를 입히려는 의도를 가진 봇입니다.

BCP

비즈니스 연속성 계획을 참조하십시오.

동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그온 시도, 의심스러운 API 호출 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 Data in a behavior graph를 참조하십시오.

빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안도 참조하십시오.

바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, '이 제품은 책인가요, 자동차인가요?' 등의 문제를 예측해야 할 수 있습니다.

블룸 필터

요소가 세트의 멤버인지 여부를 테스트하는 데 사용되는 메모리 효율성이 높은 확률론적 데이터 구조입니다.

블루/그린(Blue/Green) 배포

서로 다르지만 동일한 환경을 두 개 만드는 배포 전략입니다. 현재 애플리케이션 버전을 한 환경 (파란색) 에서 실행하고 다른 환경 (녹색) 에서 새 애플리케이션 버전을 실행합니다. 이 전략을 사용하면 영향을 최소화하면서 신속하게 롤백할 수 있습니다.

bot

인터넷을 통해 자동화된 작업을 실행하고 사람의 활동이나 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같은 일부 봇은 유용하거나 유용합니다. 배드 봇으로 알려진 일부 다른 봇은 개인이나 조직을 방해하거나 피해를 입히기 위한 것입니다.

봇넷

멀웨어에감염되어 봇 허더 또는 봇 운영자로 알려진 단일 당사자의 통제 하에 있는 봇 네트워크. 봇넷은 봇과 그 영향을 확장하는 가장 잘 알려진 메커니즘입니다.

브랜치

코드 리포지토리의 포함된 영역입니다. 리포지토리에 생성되는 첫 번째 브랜치가 기본 브랜치입니다. 기존 브랜치에서 새 브랜치를 생성한 다음 새 브랜치에서 기능을 개발하거나 버그를 수정할 수 있습니다. 기능을 구축하기 위해 생성하는 브랜치를 일반적으로 기능 브랜치라고 합니다. 기능을 출시할 준비가 되면 기능 브랜치를 기본 브랜치에 다시 병합합니다. 자세한 내용은 브랜치 정보 (문서) 를 참조하십시오. GitHub

브레이크 글래스 액세스

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스 권한이 없는 데이터에 빠르게 액세스할 수 AWS 계정 있는 수단입니다. 자세한 내용은 Well-Architected AWS 지침의 브레이크 글래스 절차 구현 표시기를 참조하십시오.

브라운필드 전략

사용자 환경의 기존 인프라 시스템 아키텍처에 브라운필드 전략을 채택할 때는 현재 시스템 및 인프라의 제약 조건을 중심으로 아키텍처를 설계합니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

버퍼 캐시

가장 자주 액세스하는 데이터가 저장되는 메모리 영역입니다.

사업 역량

기업이 가치를 창출하기 위해 하는 일(예: 영업, 고객 서비스 또는 마케팅)입니다. 마이크로서비스 아키텍처 및 개발 결정은 비즈니스 역량에 따라 이루어질 수 있습니다. 자세한 내용은 백서의 AWS에서 컨테이너화된 마이크로서비스 실행의 비즈니스 역량 중심의 구성화 섹션을 참조하십시오.

비즈니스 연속성 계획(BCP)

대규모 마이그레이션과 같은 중단 이벤트가 운영에 미치는 잠재적 영향을 해결하고 비즈니스가 신속하게 운영을 재개할 수 있도록 지원하는 계획입니다.

# C

CAF

클라우드 채택 프레임워크를 참조하십시오AWS .

카나리아 배포

최종 사용자에게 버전을 느리고 점진적으로 릴리스하는 것. 확신이 들면 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

CCoE

클라우드 센터 오브 엑설런스를 참조하십시오.

CDC

변경 데이터 캡처를 참조하십시오.

변경 데이터 캡처(CDC)

데이터베이스 테이블과 같은 데이터 소스의 변경 내용을 추적하고 변경 사항에 대한 메타데이터를 기록하는 프로세스입니다. 대상 시스템의 변경 내용을 감사하거나 복제하여 동기화를 유지하는 등의 다양한 용도로 CDC를 사용할 수 있습니다.

카오스 엔지니어링

시스템의 복원력을 테스트하기 위해 의도적으로 장애나 장애를 일으키는 이벤트를 발생시키는 행위 AWS Fault Injection Service (AWS FIS) 를 사용하여 AWS 워크로드에 스트레스를 주는 실험을 수행하고 응답을 평가할 수 있습니다.

CI/CD

지속적 통합 및 지속적 전달을 참조하십시오.

분류

예측을 생성하는 데 도움이 되는 분류 프로세스입니다. 분류 문제에 대한 ML 모델은 이산 값을 예측합니다. 이산 값은 항상 서로 다릅니다. 예를 들어, 모델이 이미지에 자동차가 있는지 여부를 평가해야 할 수 있습니다.

클라이언트측 암호화

대상이 데이터를 AWS 서비스 수신하기 전에 데이터를 로컬로 암호화합니다.

클라우드 혁신 센터(CCoE)

클라우드 모범 사례 개발, 리소스 동원, 마이그레이션 타임라인 설정, 대규모 혁신을 통한 조직 선도 등 조직 전체에서 클라우드 채택 노력을 추진하는 다분야 팀입니다. 자세한 내용은 AWS 클라우드 엔터프라이즈 전략 블로그의 CCoE 게시물을 참조하십시오.

클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 엣지 컴퓨팅 기술과 연결됩니다.

클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 클라우드 운영 모델 구축을 참조하십시오.

클라우드 채택 단계

조직이 AWS 클라우드로 마이그레이션할 때 일반적으로 거치는 4단계는 다음과 같습니다.

- 프로젝트 - 개념 증명 및 학습 목적으로 몇 가지 클라우드 관련 프로젝트 실행
- 기반 – 클라우드 채택 확장을 위한 기초 투자(예: 랜딩 존 생성, CCoE 정의, 운영 모델 구축)
- 마이그레이션 - 개별 애플리케이션 마이그레이션
- Re-invention - 제품 및 서비스 최적화와 클라우드 혁신

Stephen Orban은 클라우드 우선주의를 향한 여정과 클라우드 엔터프라이즈 전략 블로그의 채택 단계에 대한 블로그 게시물에서 이러한 단계를 정의했습니다. AWS 이들이 AWS 마이그레이션 전략과 어떤 관련이 있는지에 대한 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오.

CMDB

구성 관리 데이터베이스를 참조하십시오.

코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반 클라우드 리포지토리에는 또는 이 포함됩니다 GitHub . AWS CodeCommit코드의 각 버전을 브랜치라고 합니다. 마이크로서비스 구조에서 각 리포지토리는 단일 기능 전용입니다. 단일 CI/CD 파이프라인은 여러 리포지토리를 사용할 수 있습니다.

콜드 캐시

비어 있거나, 제대로 채워지지 않았거나, 오래되었거나 관련 없는 데이터를 포함하는 버퍼 캐시입니다. 주 메모리나 디스크에서 데이터베이스 인스턴스를 읽어야 하기 때문에 성능에 영향을 미치며, 이는 버퍼 캐시에서 읽는 것보다 느립니다.

콜드 데이터

거의 액세스되지 않고 일반적으로 과거 데이터인 데이터. 이런 종류의 데이터를 쿼리할 때는 일반적으로 느린 쿼리가 허용됩니다. 이 데이터를 성능이 낮고 비용이 저렴한 스토리지 계층 또는 클래스로 옮기면 비용을 절감할 수 있습니다.

컴퓨터 비전 (CV)

기계 학습을 사용하여 디지털 이미지 및 비디오와 같은 시각적 형식에서 정보를 분석하고 추출하는 AI 분야. 예를 들어 AWS Panorama 는 온프레미스 카메라 네트워크에 CV를 추가하는 디바이스를 제공하고, SageMaker Amazon은 CV용 이미지 처리 알고리즘을 제공합니다.

구성 드리프트

워크로드의 경우 구성이 예상 상태에서 변경됩니다. 이로 인해 워크로드가 규정을 준수하지 않게 될 수 있으며, 일반적으로 점진적이고 의도하지 않은 방식으로 진행됩니다.

구성 관리 데이터베이스(CMDB)

하드웨어 및 소프트웨어 구성 요소와 해당 구성을 포함하여 데이터베이스와 해당 IT 환경에 대한 정보를 저장하고 관리하는 리포지토리입니다. 일반적으로 마이그레이션의 포트폴리오 검색 및 분석 단계에서 CMDB의 데이터를 사용합니다.

규정 준수 팩

AWS Config 규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 규칙 및 수정 조치 모음입니다. YAML 템플릿을 사용하여 한 AWS 계정 및 지역 또는 조직 전체에 단일 엔티티로 적합성 팩을 배포할 수 있습니다. 자세한 내용은 설명서의 적합성 팩을 참조하십시오. AWS Config

지속적 통합 및 지속적 전달(CI/CD)

소프트웨어 릴리스 프로세스의 소스, 빌드, 테스트, 스테이징 및 프로덕션 단계를 자동화하는 프로세스입니다. CI/CD는 일반적으로 파이프라인으로 설명됩니다. CI/CD를 통해 프로세스를 자동화하고, 생산성을 높이고, 코드 품질을 개선하고, 더 빠르게 제공할 수 있습니다. 자세한 내용은 지속적 전달의 이점을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 지속적 전달(Continuous Delivery)과 지속적인 개발을 참조하십시오.

CV

컴퓨터 비전을 참조하십시오.

# D

저장 데이터

스토리지에 있는 데이터와 같이 네트워크에 고정되어 있는 데이터입니다.

데이터 분류

중요도와 민감도를 기준으로 네트워크의 데이터를 식별하고 분류하는 프로세스입니다. 이 프로세스는 데이터에 대한 적절한 보호 및 보존 제어를 결정하는 데 도움이 되므로 사이버 보안 위험 관리 전략의 중요한 구성 요소입니다. 데이터 분류는 AWS Well-Architected 프레임워크의 보안 핵심 요소입니다. 자세한 내용은 데이터 분류를 참조하십시오.

데이터 드리프트

프로덕션 데이터와 ML 모델 학습에 사용된 데이터 간의 상당한 차이 또는 시간 경과에 따른 입력 데이터의 의미 있는 변화. 데이터 드리프트는 ML 모델 예측의 전반적인 품질, 정확성 및 공정성을 저하시킬 수 있습니다.

전송 중 데이터

네트워크를 통과하고 있는 데이터입니다. 네트워크 리소스 사이를 이동 중인 데이터를 예로 들 수 있습니다.

데이터 메시

중앙 집중식 관리 및 거버넌스와 함께 분산되고 분산된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터 최소화를 실천하면 개인 정보 보호 위험, 비용 및 분석에 따른 탄소 발자국을 줄일 AWS 클라우드 수 있습니다.

데이터 경계

신뢰할 수 있는 ID만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스하도록 하는 데 도움이 되는 AWS 환경 내 일련의 예방 가드레일입니다. 자세한 내용은 데이터 경계 구축을 참조하십시오. AWS

데이터 사전 처리

원시 데이터를 ML 모델이 쉽게 구문 분석할 수 있는 형식으로 변환하는 것입니다. 데이터를 사전 처리한다는 것은 특정 열이나 행을 제거하고 누락된 값, 일관성이 없는 값 또는 중복 값을 처리함을 의미할 수 있습니다.

데이터 출처

라이프사이클 전반에 걸쳐 데이터의 출처와 기록을 추적하는 프로세스(예: 데이터 생성, 전송, 저장 방법).

데이터 주체

데이터를 수집 및 처리하는 개인입니다.

데이터 웨어하우스

분석과 같은 비즈니스 인텔리전스를 지원하는 데이터 관리 시스템. 데이터 웨어하우스에는 일반적으로 대량의 과거 데이터가 포함되며 일반적으로 쿼리 및 분석에 사용됩니다.

데이터 정의 언어(DDL)

데이터베이스에서 테이블 및 객체의 구조를 만들거나 수정하기 위한 명령문 또는 명령입니다.

데이터베이스 조작 언어(DML)

데이터베이스에서 정보를 수정(삽입, 업데이트 및 삭제)하기 위한 명령문 또는 명령입니다.

DDL

데이터베이스 정의 언어를 참조하십시오.

딥 앙상블

예측을 위해 여러 딥 러닝 모델을 결합하는 것입니다. 딥 앙상블을 사용하여 더 정확한 예측을 얻거나 예측의 불확실성을 추정할 수 있습니다.

딥 러닝

여러 계층의 인공 신경망을 사용하여 입력 데이터와 관심 대상 변수 간의 매핑을 식별하는 ML 하위 분야입니다.

defense-in-depth

네트워크와 그 안의 데이터 기밀성, 무결성 및 가용성을 보호하기 위해 컴퓨터 네트워크 전체에 일련의 보안 메커니즘과 제어를 신중하게 계층화하는 정보 보안 접근 방식입니다. 이 전략을 채택하면 AWS Organizations 구조의 여러 계층에 AWS여러 컨트롤을 추가하여 리소스를 보호하는 데 도움이 됩니다. 예를 들어 다단계 인증, 네트워크 세분화, 암호화를 결합한 defense-in-depth 접근 방식을 사용할 수 있습니다.

위임된 관리자

에서 AWS Organizations호환 가능한 서비스는 AWS 구성원 계정을 등록하여 조직의 계정을 관리하고 해당 서비스에 대한 권한을 관리할 수 있습니다. 이러한 계정을 해당 서비스의 위임된 관리자라고 합니다. 자세한 내용과 호환되는 서비스 목록은 AWS Organizations 설명서의 AWS Organizations와 함께 사용할 수 있는 AWS 서비스를 참조하십시오.

배포

대상 환경에서 애플리케이션, 새 기능 또는 코드 수정 사항을 사용할 수 있도록 하는 프로세스입니다. 배포에는 코드 베이스의 변경 사항을 구현한 다음 애플리케이션 환경에서 해당 코드베이스를 구축하고 실행하는 작업이 포함됩니다.

개발 환경

환경을 참조하십시오.

탐지 제어

이벤트 발생 후 탐지, 기록 및 알림을 수행하도록 설계된 보안 제어입니다. 이러한 제어는 기존의 예방적 제어를 우회한 보안 이벤트를 알리는 2차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 Detective controls를 참조하십시오.

개발 가치 흐름 매핑 (DVSM)

소프트웨어 개발 라이프사이클에서 속도와 품질에 부정적인 영향을 미치는 제약 조건을 식별하고 우선 순위를 지정하는 데 사용되는 프로세스입니다. DVSM은 원래 린 제조 방식을 위해 설계된 가치 흐름 매핑 프로세스를 확장합니다. 소프트웨어 개발 프로세스를 통해 가치를 창출하고 이동하는 데 필요한 단계와 팀에 중점을 둡니다.

디지털 트윈

건물, 공장, 산업 장비 또는 생산 라인과 같은 실제 시스템을 가상으로 표현한 것입니다. 디지털 트윈은 예측 유지 보수, 원격 모니터링, 생산 최적화를 지원합니다.

치수 표

스타 스키마에서 팩트 테이블의 양적 데이터에 대한 데이터 속성을 포함하는 작은 테이블입니다. 차원 테이블 속성은 일반적으로 텍스트처럼 동작하는 텍스트 필드 또는 불연속형 숫자입니다. 이러한 속성은 일반적으로 쿼리 제한, 필터링 및 결과 집합 레이블 지정에 사용됩니다.

재해

워크로드 또는 시스템이 기본 배포 위치에서 비즈니스 목표를 달성하지 못하게 방해하는 이벤트입니다. 이러한 이벤트는 자연재해, 기술적 오류, 의도하지 않은 구성 오류 또는 멀웨어 공격과 같은 사람의 행동으로 인한 결과일 수 있습니다.

재해 복구(DR)

재해로 인한 다운타임과 데이터 손실을 최소화하기 위해 사용하는 전략과 프로세스입니다. 자세한 내용은 워크로드의 재해 복구 AWS: AWS Well-Architected 프레임워크에서의 클라우드 복구를 참조하십시오.

DML

데이터베이스 조작 언어를 참조하십시오.

도메인 기반 설계

구성 요소를 각 구성 요소가 제공하는 진화하는 도메인 또는 핵심 비즈니스 목표에 연결하여 복잡한 소프트웨어 시스템을 개발하는 접근 방식입니다. 이 개념은 에릭 에반스에 의해 그의 저서인 도메인 기반 디자인: 소프트웨어 중심의 복잡성 해결(Boston: Addison-Wesley Professional, 2003)에서 소개되었습니다. Strangler Fig 패턴과 함께 도메인 기반 설계를 사용하는 방법에 대한 자세한 내용은 컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화하는 방법을 참조하십시오.

DR

재해 복구를 참조하십시오.

드리프트 감지

기준 구성으로부터의 편차 추적. 예를 들어 시스템 리소스의 편차를 감지하는 AWS CloudFormation 데 사용하거나 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 착륙 지대의 변경을 탐지하는 AWS Control Tower 데 사용할 수 있습니다.

DVSM

개발 가치 흐름 매핑을 참조하십시오.

# E

EDA

탐색적 데이터 분석을 참조하십시오.

엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 클라우드 컴퓨팅과 비교할 때 엣지 컴퓨팅은 통신 대기 시간을 줄이고 응답 시간을 개선할 수 있습니다.

암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 암호문으로 변환하는 컴퓨팅 프로세스입니다.

암호화 키

암호화 알고리즘에 의해 생성되는 무작위 비트의 암호화 문자열입니다. 키의 길이는 다양할 수 있으며 각 키는 예측할 수 없고 고유하게 설계되었습니다.

엔디안

컴퓨터 메모리에 바이트가 저장되는 순서입니다. 빅 엔디안 시스템은 가장 중요한 바이트를 먼저 저장합니다. 리틀 엔디안 시스템은 가장 덜 중요한 바이트를 먼저 저장합니다.

엔드포인트

서비스 엔드포인트를 참조하십시오.

엔드포인트 서비스

Virtual Private Cloud(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 다른 주체 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 부여하여 엔드포인트 서비스를 생성하고 권한을 부여할 수 있습니다. AWS PrivateLink 이러한 계정 또는 보안 주체는 인터페이스 VPC 엔드포인트를 생성하여 엔드포인트 서비스에 비공개로 연결할 수 있습니

다. 자세한 내용은 Amazon Virtual Private Cloud(VPC) 설명서의 엔드포인트 서비스 생성을 참조하십시오.

ERP (전사적 자원 관리)

기업의 주요 비즈니스 프로세스 (예: 회계, MES, 프로젝트 관리) 를 자동화하고 관리하는 시스템입니다.

봉투 암호화

암호화 키를 다른 암호화 키로 암호화하는 프로세스입니다. 자세한 내용은 AWS Key Management Service (AWS KMS) 설명서의 봉투 암호화를 참조하십시오.

환경

실행 중인 애플리케이션의 인스턴스입니다. 다음은 클라우드 컴퓨팅의 일반적인 환경 유형입니다.

- 개발 환경 - 애플리케이션 유지 관리를 담당하는 핵심 팀만 사용할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. 개발 환경은 변경 사항을 상위 환경으로 승격하기 전에 테스트하는 데 사용됩니다. 이러한 유형의 환경을 테스트 환경이라고도 합니다.
- 하위 환경 - 초기 빌드 및 테스트에 사용되는 환경을 비롯한 애플리케이션의 모든 개발 환경입니다.
- 프로덕션 환경 - 최종 사용자가 액세스할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. CI/CD 파이프라인에서 프로덕션 환경이 마지막 배포 환경입니다.
- 상위 환경 - 핵심 개발 팀 이외의 사용자가 액세스할 수 있는 모든 환경입니다. 프로덕션 환경, 프로덕션 이전 환경 및 사용자 수용 테스트를 위한 환경이 여기에 포함될 수 있습니다.

에픽

애자일 방법론에서 작업을 구성하고 우선순위를 정하는 데 도움이 되는 기능적 범주입니다. 에픽은 요구 사항 및 구현 작업에 대한 개괄적인 설명을 제공합니다. 예를 들어 AWS CAF 보안 에픽에는 ID 및 액세스 관리, 탐지 제어, 인프라 보안, 데이터 보호, 사고 대응 등이 포함됩니다. AWS 마이그레이션 전략의 에픽에 대한 자세한 내용은 프로그램 구현 가이드를 참조하십시오.

ERP

엔터프라이즈 리소스 계획을 참조하십시오.

탐색 데이터 분석(EDA)

데이터 세트를 분석하여 주요 특성을 파악하는 프로세스입니다. 데이터를 수집 또는 집계한 다음 초기 조사를 수행하여 패턴을 찾고, 이상을 탐지하고, 가정을 확인합니다. EDA는 요약 통계를 계산하고 데이터 시각화를 생성하여 수행됩니다.

# F

팩트 테이블

[스타 스키마의](#) 중앙 테이블. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블에는 측정값이 포함된 열과 차원 테이블의 외부 키가 포함된 열 등 두 가지 유형의 열이 포함됩니다.

빨리 실패하세요

빈번하고 점진적인 테스트를 통해 개발 라이프사이클을 단축하는 철학. 이는 애자일 접근 방식의 중요한 부분입니다.

장애 격리 경계

장애 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역 AWS 리전, 컨트롤 플레인 또는 데이터 플레인과 같은 경계 AWS 클라우드자세한 내용은 [AWS 장애 격리](#) 경계를 참조하십시오.

기능 브랜치

[브랜치를](#) 참조하십시오.

기능

예측에 사용하는 입력 데이터입니다. 예를 들어, 제조 환경에서 기능은 제조 라인에서 주기적으로 캡처되는 이미지일 수 있습니다.

기능 중요도

모델의 예측에 특성이 얼마나 중요한지를 나타냅니다. 이는 일반적으로 SHAP(Shapley Additive Descriptions) 및 통합 그레디언트와 같은 다양한 기법을 통해 계산할 수 있는 수치 점수로 표현됩니다. 자세한 내용은 [다음을AWS사용한 기계 학습 모델 해석 가능성을](#) 참조하십시오.

기능 변환

추가 소스로 데이터를 보강하거나, 값을 조정하거나, 단일 데이터 필드에서 여러 정보 세트를 추출하는 등 ML 프로세스를 위해 데이터를 최적화하는 것입니다. 이를 통해 ML 모델이 데이터를 활용할 수 있습니다. 예를 들어, 날짜 '2021-05-27 00:15:37'을 '2021년', '5월', '목', '15일'로 분류하면 학습 알고리즘이 다양한 데이터 구성 요소와 관련된 미묘한 패턴을 학습하는 데 도움이 됩니다.

FGAC

[세분화된 액세스 제어를](#) 참조하십시오.

세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

플래시컷 마이그레이션

단계별 접근 방식 대신 변경 데이터 캡처를 통한 지속적인 데이터 복제를 통해 최단 시간에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

# G

지리적 차단

지리적 제한을 참조하십시오.

지리적 제한(지리적 차단)

CloudFrontAmazon에서는 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 옵션을 제공합니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 설명서의 콘텐츠의 지리적 배포 제한을 참조하십시오. CloudFront

Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 트렁크 기반 워크플로는 현대적이고 선호되는 접근 방식입니다.

브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 브라운필드라고도 하는 기존 인프라와의 호환성 제한 없이 모든 새로운 기술을 선택할 수 있습니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

가드레일

조직 단위(OU) 전체에서 리소스, 정책 및 규정 준수를 관리하는 데 도움이 되는 중요 규칙입니다. 예방 가드레일은 규정 준수 표준에 부합하도록 정책을 시행하며, 서비스 제어 정책과 IAM 권한 경계를 사용하여 구현됩니다. 탐지 가드레일은 정책 위반 및 규정 준수 문제를 감지하고 해결을 위한 알림을 생성하며, 이들은, Amazon AWS Config AWS Security Hub GuardDuty AWS Trusted Advisor, Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

# H

하

고가용성을 확인하세요.

이기종 데이터베이스 마이그레이션

다른 데이터베이스 엔진을 사용하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Oracle에서 Amazon Aurora로) 이기종 마이그레이션은 일반적으로 리아키텍트 작업의 일부이며 스키마를 변환하는 것은 복잡한 작업일 수 있습니다.AWS 는 스키마 변환에 도움이 되는 AWS SCT를 제공합니다.

높은 가용성(HA)

문제나 재해 발생 시 개입 없이 지속적으로 운영할 수 있는 워크로드의 능력. HA 시스템은 자동으로 장애 조치되고, 지속적으로 고품질 성능을 제공하고, 성능에 미치는 영향을 최소화하면서 다양한 부하와 장애를 처리하도록 설계되었습니다.

히스토리언 현대화

제조 산업의 요구 사항을 더 잘 충족하도록 운영 기술(OT) 시스템을 현대화하고 업그레이드하는 데 사용되는 접근 방식입니다. 히스토리언은 공장의 다양한 출처에서 데이터를 수집하고 저장하는 데 사용되는 일종의 데이터베이스입니다.

동종 데이터베이스 마이그레이션

동일한 데이터베이스 엔진을 공유하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Microsoft SQL Server에서 Amazon RDS for SQL Server로) 동종 마이그레이션은 일반적으로 리호스팅 또는 리플랫포밍 작업의 일부입니다. 네이티브 데이터베이스 유틸리티를 사용하여 스키마를 마이그레이션할 수 있습니다.

핫 데이터

자주 액세스하는 데이터(예: 실시간 데이터 또는 최근 번역 데이터). 일반적으로 이 데이터에는 빠른 쿼리 응답을 제공하기 위한 고성능 스토리지 계층 또는 클래스가 필요합니다.

핫픽스

프로덕션 환경의 중요한 문제를 해결하기 위한 긴급 수정입니다. 긴급성 때문에 핫픽스는 일반적으로 일반적인 DevOps 릴리스 워크플로 외부에서 만들어집니다.

하이퍼케어 기간

전환 직후 마이그레이션 팀이 문제를 해결하기 위해 클라우드에서 마이그레이션된 애플리케이션을 관리하고 모니터링하는 기간입니다. 일반적으로 이 기간은 1~4일입니다. 하이퍼케어 기간이 끝나면 마이그레이션 팀은 일반적으로 애플리케이션에 대한 책임을 클라우드 운영 팀에 넘깁니다.

## I

IaC

인프라를 코드로 보세요.

자격 증명 기반 정책

환경 내에서 권한을 정의하는 하나 이상의 IAM 보안 주체에 연결된 정책입니다. AWS 클라우드

유휴 애플리케이션

90일 동안 평균 CPU 및 메모리 사용량이 5~20%인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하거나 온프레미스에 유지하는 것이 일반적입니다.

IIoT

산업용 사물 인터넷을 참조하십시오.

불변의 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드용 새 인프라를 배포하는 모델입니다. 변경 불가능한 인프라는 기본적으로 변경 가능한 인프라보다 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 Well-Architected AWS 프레임워크의 변경 불가능한 인프라를 사용한 배포 모범 사례를 참조하십시오.

인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 VPC는 애플리케이션 외부에서 네트워크 연결을 허용, 검사 및 라우팅합니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

증분 마이그레이션

한 번에 전체 전환을 수행하는 대신 애플리케이션을 조금씩 마이그레이션하는 전환 전략입니다. 예를 들어, 처음에는 소수의 마이크로서비스나 사용자만 새 시스템으로 이동할 수 있습니다. 모든 것

이 제대로 작동하는지 확인한 후에는 레거시 시스템을 폐기할 수 있을 때까지 추가 마이크로서비스 또는 사용자를 점진적으로 이동할 수 있습니다. 이 전략을 사용하면 대규모 마이그레이션과 관련된 위험을 줄일 수 있습니다.

Industry 4.0

Klaus Schwab이 연결성, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통한 제조 프로세스의 현대화를 지칭하기 위해 2016년에 도입한 용어입니다.

인프라

애플리케이션의 환경 내에 포함된 모든 리소스와 자산입니다.

코드형 인프라(IaC)

구성 파일 세트를 통해 애플리케이션의 인프라를 프로비저닝하고 관리하는 프로세스입니다. IaC는 새로운 환경의 반복 가능성, 신뢰성 및 일관성을 위해 인프라 관리를 중앙 집중화하고, 리소스를 표준화하고, 빠르게 확장할 수 있도록 설계되었습니다.

산업용 사물 인터넷(IIoT)

제조, 에너지, 자동차, 의료, 생명과학, 농업 등의 산업 부문에서 인터넷에 연결된 센서 및 디바이스의 사용 자세한 내용은 산업용 사물 인터넷(IoT) 디지털 트랜스포메이션 전략 구축을 참조하십시오.

검사 VPC

AWS 다중 계정 아키텍처에서 VPC (동일하거나 AWS 리전다른), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 관리하는 중앙 집중식 VPC입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

사물 인터넷(IoT)

인터넷이나 로컬 통신 네트워크를 통해 다른 디바이스 및 시스템과 통신하는 센서 또는 프로세서가 내장된 연결된 물리적 객체의 네트워크 자세한 내용은 IoT란?을 참조하십시오.

해석력

모델의 예측이 입력에 따라 어떻게 달라지는지를 사람이 이해할 수 있는 정도를 설명하는 기계 학습 모델의 특성입니다. 자세한 내용은 Machine learning model interpretability with AWS를 참조하십시오.

IoT

사물 인터넷을 참조하십시오.

IT 정보 라이브러리(TIL)

IT 서비스를 제공하고 이러한 서비스를 비즈니스 요구 사항에 맞게 조정하기 위한 일련의 모범 사례 ITIL은 ITSM의 기반을 제공합니다.

IT 서비스 관리(TSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 운영을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

ITIL

IT 정보 라이브러리를 참조하십시오.

ITSM

IT 서비스 관리를 참조하십시오.

# L

레이블 기반 액세스 제어(LBAC)

사용자 및 데이터 자체에 각각 보안 레이블 값을 명시적으로 할당하는 필수 액세스 제어(MAC)를 구현한 것입니다. 사용자 보안 레이블과 데이터 보안 레이블 간의 교차 부분에 따라 사용자가 볼 수 있는 행과 열이 결정됩니다.

랜딩 존

Landing Zone은 확장 가능하고 안전한 잘 설계된 다중 계정 AWS 환경입니다. 조직은 여기에서부터 보안 및 인프라 환경에 대한 확신을 가지고 워크로드와 애플리케이션을 신속하게 시작하고 배포할 수 있습니다. 랜딩 존에 대한 자세한 내용은 안전하고 확장 가능한 다중 계정 AWS 환경 설정을 참조하십시오.

대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

LBAC

레이블 기반 액세스 제어를 참조하십시오.

최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 최소 권한 적용을 참조하십시오.

리프트 앤드 시프트

7 R을 참조하십시오.

리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안 참조.

하위 환경

환경 참조.

# M

기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공 지능의 한 유형입니다. ML은 사물 인터넷(IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 기계 학습을 참조하십시오.

기본 브랜치

브랜치 참조.

악성 코드

컴퓨터 보안 또는 개인 정보를 침해하도록 설계된 소프트웨어. 멀웨어는 컴퓨터 시스템을 방해하거나, 민감한 정보를 유출하거나, 무단 액세스를 얻을 수 있습니다. 멀웨어의 예로는 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어, 키로거 등이 있습니다.

매니지드 서비스

AWS 서비스 인프라 계층, 운영 체제 및 플랫폼을 AWS 운영하며 사용자는 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. 관리형 서비스의 예로는 아마존 심플 스토리지 서비스 (Amazon S3) 와 아마존 DynamoDB가 있습니다. 이러한 서비스를 추상화된 서비스라고도 합니다.

제조 실행 시스템 (MES)

제조 현장에서 원자재를 완제품으로 전환하는 생산 프로세스를 추적, 모니터링, 문서화 및 제어하기 위한 소프트웨어 시스템입니다.

MAP

Migration Acceleration 프로그램을 참조하십시오.

기구

도구를 만들고 도구 채택을 유도한 다음 결과를 검토하여 조정하는 전체 프로세스입니다. 메커니즘은 작동하면서 자체적으로 강화되고 개선되는 사이클입니다. 자세한 내용은 AWS Well-Architected 프레임워크에서의 메커니즘 구축을 참조하십시오.

멤버 계정

조직의 일부인 관리 계정을 AWS 계정 제외한 모든 계정 AWS Organizations하나의 계정은 한 번에 하나의 조직 멤버만 될 수 있습니다.

MES

제조 실행 시스템을 참조하십시오.

메시지 큐 텔레메트리 전송 (MQTT)

퍼블리시/구독 패턴을 기반으로 하는 리소스가 제한된 IoT 디바이스를 위한 경량 machine-to-machine (M2M) 통신 프로토콜입니다.

마이크로서비스

잘 정의된 API를 통해 통신하고 일반적으로 소규모 자체 팀이 소유하는 소규모 독립 서비스입니다. 예를 들어, 보험 시스템에는 영업, 마케팅 등의 비즈니스 역량이나 구매, 청구, 분석 등의 하위 영역에 매핑되는 마이크로 서비스가 포함될 수 있습니다. 마이크로서비스의 이점으로 민첩성, 유연한 확장, 손쉬운 배포, 재사용 가능한 코드, 복원력 등이 있습니다. 자세한 내용은 서버리스 서비스를 사용하여 마이크로서비스 통합을 참조하십시오. AWS

마이크로서비스 아키텍처

각 애플리케이션 프로세스를 마이크로서비스로 실행하는 독립 구성 요소를 사용하여 애플리케이션을 구축하는 접근 방식입니다. 이러한 마이크로서비스는 경량 API를 사용하여 잘 정의된 인터페이스를 통해 통신합니다. 애플리케이션의 특정 기능에 대한 수요에 맞게 이 아키텍처의 각 마이크로서비스를 업데이트, 배포 및 조정할 수 있습니다. 자세한 내용은 마이크로서비스 구현을 참조하십시오. AWS

Migration Acceleration Program(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄할 수 있도록 컨설팅 지원, 교육 및 서비스를 제공하는 AWS 프로그램입니다. MAP에는 레거시 마이그레이션을 체계적인 방식으로 실행하기 위한 마이그레이션 방법론과 일반적인 마이그레이션 시나리오를 자동화하고 가속화하는 도구 세트가 포함되어 있습니다.

대규모 마이그레이션

애플리케이션 포트폴리오의 대다수를 웨이브를 통해 클라우드로 이동하는 프로세스로, 각 웨이브에서 더 많은 애플리케이션이 더 빠른 속도로 이동합니다. 이 단계에서는 이전 단계에서 배운 모범 사례와 교훈을 사용하여 팀, 도구 및 프로세스의 마이그레이션 팩토리를 구현하여 자동화 및 민첩한 제공을 통해 워크로드 마이그레이션을 간소화합니다. 이것은 AWS 마이그레이션 전략의 세 번째 단계입니다.

마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자 및 스프린트에서 일하는 DevOps 전문가가 포함됩니다. 엔터프라이즈 애플리케이션 포트폴리오의 20~50%는 공장 접근 방식으로 최적화할 수 있는 반복되는 패턴으로 구성되어 있습니다. 자세한 내용은 이 콘텐츠 세트의 클라우드 마이그레이션 팩토리 가이드와 마이그레이션 팩토리에 대한 설명을 참조하십시오.

마이그레이션 메타데이터

마이그레이션을 완료하는 데 필요한 애플리케이션 및 서버에 대한 정보 각 마이그레이션 패턴에는 서로 다른 마이그레이션 메타데이터 세트가 필요합니다. 마이그레이션 메타데이터의 예로는 대상 서브넷, 보안 그룹, 계정 등이 있습니다. AWS

마이그레이션 패턴

사용되는 마이그레이션 전략, 마이그레이션 대상, 마이그레이션 애플리케이션 또는 서비스를 자세히 설명하는 반복 가능한 마이그레이션 작업입니다. 예: 애플리케이션 마이그레이션 서비스를 사용하여 Amazon AWS EC2로의 리호스트 마이그레이션.

Migration Portfolio Assessment(MPA)

클라우드로 마이그레이션하기 위한 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다. AWS MPA는 상세한 포트폴리오 평가(서버 적정 규모 조정, 가격 책정, TCO 비교, 마이그레이션 비용 분석)와 마이그레이션 계획(애플리케이션 데이터 분석 및 데이터 수집, 애플리케이션 그룹화, 마이그레이션 우선순위 지정, 웨이브 계획)을 제공합니다. MPA 도구 (로그인 필요) 는 모든 컨설턴트와 APN 파트너 AWS 컨설턴트에게 무료로 제공됩니다.

마이그레이션 준비 상태 평가(MRA)

CAF를 사용하여 조직의 클라우드 준비 상태에 대한 통찰력을 얻고, 강점과 약점을 파악하고, 식별된 격차를 해소하기 위한 실행 계획을 수립하는 프로세스입니다. AWS 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오. MRA는 AWS 마이그레이션 전략의 첫 번째 단계입니다.

마이그레이션 전략

워크로드를 클라우드로 마이그레이션하는 데 사용된 접근 방식. AWS 자세한 내용은 이 용어집의 7R 항목 및 대규모 마이그레이션 가속화를 위한 조직 동원을 참조하십시오.

ML

기계 학습을 참조하십시오.

현대화

비용을 절감하고 효율성을 높이고 혁신을 활용하기 위해 구식(레거시 또는 모놀리식) 애플리케이션과 해당 인프라를 클라우드의 민첩하고 탄력적이고 가용성이 높은 시스템으로 전환하는 것입니다. 자세한 내용은 의 AWS 클라우드애플리케이션 현대화 전략을 참조하십시오.

현대화 준비 상태 평가

조직 애플리케이션의 현대화 준비 상태를 파악하고, 이점, 위험 및 종속성을 식별하고, 조직이 해당 애플리케이션의 향후 상태를 얼마나 잘 지원할 수 있는지를 확인하는 데 도움이 되는 평가입니다. 평가 결과는 대상 아키텍처의 청사진, 현대화 프로세스의 개발 단계와 마일스톤을 자세히 설명하는 로드맵 및 파악된 격차를 해소하기 위한 실행 계획입니다. 자세한 내용은 AWS 클라우드에서 애플리케이션의 현대화 준비 상태 평가를 참조하십시오.

모놀리식 애플리케이션(모놀리식 유형)

긴밀하게 연결된 프로세스를 사용하여 단일 서비스로 실행되는 애플리케이션입니다. 모놀리식 애플리케이션에는 몇 가지 단점이 있습니다. 한 애플리케이션 기능에 대한 수요가 급증하면 전체 아키텍처 규모를 조정해야 합니다. 코드 베이스가 커지면 모놀리식 애플리케이션의 기능을 추가하거나 개선하는 것도 더 복잡해집니다. 이러한 문제를 해결하기 위해 마이크로서비스 아키텍처를 사용할 수 있습니다. 자세한 내용은 마이크로서비스로 모놀리식 유형 분해를 참조하십시오.

MPA

마이그레이션 포트폴리오 평가를 참조하십시오.

MQTT

메시지 큐 원격 분석 전송을 참조하십시오.

멀티클래스 분류

여러 클래스에 대한 예측(2개 이상의 결과 중 하나 예측)을 생성하는 데 도움이 되는 프로세스입니다. 예를 들어, ML 모델이 '이 제품은 책인가요, 자동차인가요, 휴대폰인가요?' 또는 '이 고객이 가장 관심을 갖는 제품 범주는 무엇인가요?'라고 물을 수 있습니다.

변경 가능한 인프라

프로덕션 워크로드를 위해 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 안정성 및 예측 가능성을 개선하기 위해 AWS Well-Architected Framework는 변경 불가능한 인프라를 모범 사례로 사용할 것을 권장합니다.

# O

OAC

원본 액세스 제어를 참조하십시오.

좋아요

원본 액세스 ID를 참조하십시오.

OCM

조직 변경 관리를 참조하십시오.

오프라인 마이그레이션

마이그레이션 프로세스 중 소스 워크로드가 중단되는 마이그레이션 방법입니다. 이 방법은 가동 중지 증가를 수반하며 일반적으로 작고 중요하지 않은 워크로드에 사용됩니다.

OI

운영 통합을 참조하십시오.

안녕하세요.

운영 수준 계약을 참조하십시오.

온라인 마이그레이션

소스 워크로드를 오프라인 상태로 전환하지 않고 대상 시스템에 복사하는 마이그레이션 방법입니다. 워크로드에 연결된 애플리케이션은 마이그레이션 중에도 계속 작동할 수 있습니다. 이 방법은 가동 중지 차단 또는 최소화를 수반하며 일반적으로 중요한 프로덕션 워크로드에 사용됩니다.

OPC-UA

오픈 프로세스 커뮤니케이션 - 통합 아키텍처를 참조하십시오.

오픈 프로세스 커뮤니케이션 - 통합 아키텍처 (OPC-UA)

산업 machine-to-machine 자동화를 위한 (M2M) 통신 프로토콜. OPC-UA는 데이터 암호화, 인증 및 권한 부여 체계와 함께 상호 운용성 표준을 제공합니다.

운영 수준 협약(OLA)

서비스 수준에 관한 계약(SLA)을 지원하기 위해 직무 IT 그룹이 서로에게 제공하기로 약속한 내용을 명확히 하는 계약입니다.

운영 준비 검토 (ORR)

인시던트 및 발생 가능한 실패의 범위를 이해, 평가, 예방 또는 줄이는 데 도움이 되는 질문 및 관련 모범 사례로 구성된 체크리스트입니다. 자세한 내용은 Well-Architected AWS 프레임워크의 운영 준비 상태 검토 (ORR) 를 참조하십시오.

운영 기술 (OT)

물리적 환경과 함께 작동하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템. 제조 분야에서는 OT와 정보 기술 (IT) 시스템의 통합이 인더스트리 4.0 혁신의 핵심 초점입니다.

운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 운영 통합 가이드를 참조하십시오.

조직 트레일

이를 통해 AWS CloudTrail 생성되는 트레일은 조직 AWS 계정 내 모든 사용자의 모든 이벤트를 기록합니다. AWS Organizations이 트레일은 조직에 속한 각 AWS 계정 에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 CloudTrail 설명서에서 조직을 위한 트레일 만들기를 참조하십시오.

조직 변경 관리(OCM)

사람, 문화 및 리더십 관점에서 중대하고 파괴적인 비즈니스 혁신을 관리하기 위한 프레임워크입니다. OCM은 변화 채택을 가속화하고, 과도기적 문제를 해결하고, 문화 및 조직적 변화를 주도함으로써 조직이 새로운 시스템 및 전략을 준비하고 전환할 수 있도록 지원합니다. 클라우드 채택 프로젝트에 필요한 변화 속도 때문에 AWS 마이그레이션 전략에서는 이 프레임워크를 사용자 가속화라고 합니다. 자세한 내용은 사용 가이드를 참조하십시오.

오리진 액세스 제어(OAC)

CloudFront에서는 Amazon Simple Storage Service (Amazon S3) 콘텐츠의 보안을 위해 액세스를 제한하는 향상된 옵션을 제공합니다. OAC는 모든 S3 버킷 AWS 리전, AWS KMS (SSE-KMS) 를 사용한 서버 측 암호화, S3 버킷에 대한 동적 및 요청을 모두 지원합니다. PUT DELETE

오리진 액세스 ID(OAI)

CloudFront에서는 Amazon S3 콘텐츠 보안을 위해 액세스를 제한하는 옵션입니다. OAI를 사용하면 Amazon S3가 인증할 수 있는 보안 주체를 CloudFront 생성합니다. 인증된 보안 주체는 특정 배

포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. CloudFront 더 세분화되고 향상된 액세스 제어를 제공하는 OAC도 참조하십시오.

또는

운영 준비 상태 검토를 참조하십시오.

아니요

운영 기술을 참조하십시오.

아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작되는 네트워크 연결을 처리하는 VPC입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

# P

권한 경계

사용자나 역할이 가질 수 있는 최대 권한을 설정하기 위해 IAM 보안 주체에 연결되는 IAM 관리 정책입니다. 자세한 내용은 IAM 설명서의 권한 경계를 참조하십시오.

개인 식별 정보(PII)

직접 보거나 다른 관련 데이터와 함께 짝을 지을 때 개인의 신원을 합리적으로 추론하는 데 사용할 수 있는 정보입니다. PII의 예로는 이름, 주소, 연락처 정보 등이 있습니다.

PII

개인 식별 정보를 참조하십시오.

플레이북

클라우드에서 핵심 운영 기능을 제공하는 등 마이그레이션과 관련된 작업을 캡처하는 일련의 사전 정의된 단계입니다. 플레이북은 스크립트, 자동화된 런북 또는 현대화된 환경을 운영하는 데 필요한 프로세스나 단계 요약의 형태를 취할 수 있습니다.

PLC

프로그래머블 로직 컨트롤러를 참조하십시오.

PLM

제품 라이프사이클 관리를 참조하십시오.

정책

권한을 정의 (ID 기반 정책 참조) 하거나, 액세스 조건을 지정 (리소스 기반 정책 참조) 하거나, 조직 내 모든 계정에 대한 최대 권한을 정의 AWS Organizations (서비스 제어 정책 참조) 할 수 있는 개체입니다.

다국어 지속성

데이터 액세스 패턴 및 기타 요구 사항을 기반으로 독립적으로 마이크로서비스의 데이터 스토리지 기술 선택. 마이크로서비스가 동일한 데이터 스토리지 기술을 사용하는 경우 구현 문제가 발생하거나 성능이 저하될 수 있습니다. 요구 사항에 가장 적합한 데이터 스토어를 사용하면 마이크로서비스를 더 쉽게 구현하고 성능과 확장성을 높일 수 있습니다. 자세한 내용은 마이크로서비스에서 데이터 지속성 활성화를 참조하십시오.

포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 마이그레이션 준비 상태 평가를 참조하십시오.

조건자

일반적으로 조항에 있는 `true` `false` OR를 반환하는 쿼리 조건입니다. `WHERE`

조건부 푸시다운

전송하기 전에 쿼리의 데이터를 필터링하는 데이터베이스 쿼리 최적화 기법입니다. 이렇게 하면 관계형 데이터베이스에서 검색하고 처리해야 하는 데이터의 양이 줄어들고 쿼리 성능이 향상됩니다.

예방적 제어

이벤트 발생을 방지하도록 설계된 보안 제어입니다. 이 제어는 네트워크에 대한 무단 액세스나 원치 않는 변경을 방지하는 데 도움이 되는 1차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 Preventative controls를 참조하십시오.

보안 주체

작업을 수행하고 리소스에 액세스할 수 AWS 있는 엔티티 이 엔티티는 일반적으로 IAM 역할의 루트 사용자 또는 사용자입니다. AWS 계정자세한 내용은 IAM 설명서의 역할 용어 및 개념의 보안 주체를 참조하십시오.

개인 정보 보호 중심 설계

전체 엔지니어링 프로세스에서 개인 정보를 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

프라이빗 호스팅 영역

Amazon Route 53에서 하나 이상의 VPC 내 도메인과 하위 도메인에 대한 DNS 쿼리에 응답하는 방법에 대한 정보가 담긴 컨테이너입니다. 자세한 내용은 Route 53 설명서의 프라이빗 호스팅 영역 작업을 참조하십시오.

사전 예방 제어

규정을 준수하지 않는 리소스의 배포를 방지하도록 설계된 보안 제어입니다. 이러한 컨트롤은 리소스를 프로비저닝하기 전에 리소스를 스캔합니다. 리소스가 컨트롤과 호환되지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 컨트롤 참조 안내서를 참조하고 보안 제어 구현의 사전 제어를 참조하십시오. AWS

제품 라이프사이클 관리 (PLM)

설계, 개발, 출시부터 성장 및 성숙도, 폐기 및 제거에 이르는 전체 라이프사이클에 걸친 제품 데이터 및 프로세스 관리.

프로덕션 환경

환경을 참조하십시오.

프로그래머블 로직 컨트롤러 (PLC)

제조 분야에서 기계를 모니터링하고 제조 프로세스를 자동화하는 매우 안정적이고 적응력이 뛰어난 컴퓨터입니다.

가명화

데이터세트의 개인 식별자를 자리 표시자 값으로 바꾸는 프로세스입니다. 가명화는 개인 정보를 보호하는 데 도움이 될 수 있습니다. 가명화된 데이터는 여전히 개인 데이터로 간주됩니다.

게시/구독 (게시/구독)

마이크로서비스 간의 비동기 통신을 통해 확장성과 응답성을 개선할 수 있는 패턴입니다. 예를 들어 마이크로서비스 기반 MES에서 마이크로서비스는 다른 마이크로서비스가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로서비스를 추가할 수 있습니다.

# Q

쿼리 계획

SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 일련의 단계 (예: 지침).

쿼리 계획 회귀

데이터베이스 서비스 최적화 프로그램이 데이터베이스 환경을 변경하기 전보다 덜 최적의 계획을 선택하는 경우입니다. 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩 및 데이터베이스 엔진 업데이트의 변경으로 인해 발생할 수 있습니다.

# R

RACI 매트릭스

RACI (책임, 책임, 상담, 정보 제공) 를 참조하십시오.

랜섬웨어

결제가 완료될 때까지 컴퓨터 시스템이나 데이터에 대한 액세스를 차단하도록 설계된 악성 소프트웨어입니다.

RASCI 매트릭스

책임, 책임, 상담, 정보 제공 (RACI) 을 참조하십시오.

RCAC

행 및 열 액세스 제어를 참조하십시오.

읽기 전용 복제본

읽기 전용 용도로 사용되는 데이터베이스의 사본입니다. 쿼리를 읽기 전용 복제본으로 라우팅하여 기본 데이터베이스의 로드를 줄일 수 있습니다.

재설계

7 R을 참조하십시오.

Recovery Point Objective(RPO)

마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이에 따라 마지막 복구 시점과 서비스 중단 사이에 허용되는 데이터 손실로 간주되는 범위가 결정됩니다.

Recovery Time Objective(RTO)

서비스 중단과 서비스 복원 사이의 허용 가능한 지연 시간입니다.

리팩터링

7 R을 참조하십시오.

지역

지리적 영역의 AWS 리소스 모음. AWS 리전 각각은 격리되어 있고 서로 독립적이므로 내결함성, 안정성 및 복원력을 제공합니다. 자세한 내용은 사용할 수 있는 AWS 리전 계정 지정을 참조하십시오.

회귀

숫자 값을 예측하는 ML 기법입니다. 예를 들어, '이 집은 얼마에 팔릴까?'라는 문제를 풀기 위해 ML 모델은 선형 회귀 모델을 사용하여 주택에 대해 알려진 사실(예: 면적)을 기반으로 주택의 매매 가격을 예측할 수 있습니다.

리호스팅

7 R을 참조하십시오.

release

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

고쳐 놓다

7 R을 참조하십시오.

리플랫포밍

7 R을 참조하십시오.

환매

7 R을 참조하십시오.

복원력

장애를 견디거나 장애를 복구할 수 있는 애플리케이션의 능력. 고가용성 및 재해 복구는 복원력을 계획할 때 일반적으로 고려해야 할 사항입니다. AWS 클라우드자세한 내용은 AWS 클라우드 복원력을 참조하십시오.

리소스 기반 정책

Amazon S3 버킷, 엔드포인트, 암호화 키 등의 리소스에 연결된 정책입니다. 이 유형의 정책은 액세스가 허용된 보안 주체, 지원되는 작업 및 충족해야 하는 기타 조건을 지정합니다.

RACI(Responsible, Accountable, Consulted, Informed) 매트릭스

마이그레이션 활동 및 클라우드 운영에 참여하는 모든 당사자의 역할과 책임을 정의하는 매트릭스입니다. 매트릭스 이름은 매트릭스에 정의된 책임 유형에서 파생됩니다. 실무 담당자 (R), 의사 결

정권자 (A), 업무 수행 조언자 (C), 결과 통보 대상자 (I). 지원자는 (S) 선택사항입니다. 지원자를 포함하면 매트릭스를 RASCI 매트릭스라고 하고, 지원자를 제외하면 RACI 매트릭스라고 합니다.

대응 제어

보안 기준에서 벗어나거나 부정적인 이벤트를 해결하도록 설계된 보안 제어입니다. 자세한 내용은 Implementing security controls on AWS의 Responsive controls를 참조하십시오.

retain

7 R을 참조하십시오.

은퇴

7 R을 참조하십시오.

회전

공격자가 자격 증명에 액세스하는 것을 더 어렵게 만들기 위해 암호를 주기적으로 업데이트하는 프로세스입니다.

행 및 열 액세스 제어(RCAC)

액세스 규칙이 정의된 기본적이고 유연한 SQL 표현식을 사용합니다. RCAC는 행 권한과 열 마스크로 구성됩니다.

RPO

복구 지점 목표를 참조하십시오.

RTO

복구 시간 목표를 참조하십시오.

런북

특정 작업을 수행하는 데 필요한 일련의 수동 또는 자동 절차입니다. 일반적으로 오류율이 높은 반복 작업이나 절차를 간소화하기 위해 런북을 만듭니다.

# S

SAML 2.0

많은 ID 제공업체 (IdPs) 가 사용하는 개방형 표준입니다. 이 기능을 사용하면 페더레이션 싱글 사인온 (SSO) 이 가능하므로 조직의 모든 사용자를 위해 IAM에서 사용자를 생성하지 않고도 사용

자가 AWS API 작업에 AWS Management Console 로그인하거나 API 작업을 호출할 수 있습니다. SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 SAML 2.0 기반 페더레이션 정보를 참조하십시오.

SCADA

감독, 통제 및 데이터 수집을 참조하십시오.

SCP

서비스 제어 정책을 참조하십시오.

secret

에는 AWS Secrets Manager비밀번호나 사용자 자격 증명과 같은 기밀 또는 제한된 정보를 암호화된 형태로 저장합니다. 비밀 값과 해당 메타데이터로 구성됩니다. 비밀 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 Secrets Manager 문서의 시크릿을 참조하십시오.

보안 제어

위협 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어에는 예방적, 탐정적, 대응적, 사전 예방적 등 네 가지 기본 유형이 있습니다.

보안 강화

공격 표면을 줄여 공격에 대한 저항력을 높이는 프로세스입니다. 더 이상 필요하지 않은 리소스 제거, 최소 권한 부여의 보안 모범 사례 구현, 구성 파일의 불필요한 기능 비활성화 등의 작업이 여기에 포함될 수 있습니다.

보안 정보 및 이벤트 관리(SIEM) 시스템

보안 정보 관리(SIM)와 보안 이벤트 관리(SEM) 시스템을 결합하는 도구 및 서비스입니다. SIEM 시스템은 서버, 네트워크, 디바이스 및 기타 소스에서 데이터를 수집, 모니터링 및 분석하여 위협과 보안 침해를 탐지하고 알림을 생성합니다.

보안 대응 자동화

보안 이벤트에 자동으로 대응하거나 보안 이벤트를 해결하도록 설계된 사전 정의되고 프로그래밍된 조치입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 탐지 또는 대응형 보안 제어 역할을 합니다. AWS 자동 응답 조치의 예로는 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치, 자격 증명 교체 등이 있습니다.

서버 측 암호화

수신자에 의한 목적지의 데이터 암호화 AWS 서비스

서비스 제어 정책(SCP)

AWS Organizations에 속한 조직의 모든 계정에 대한 권한을 중앙 집중식으로 제어하는 정책입니다. SCP는 관리자가 사용자 또는 역할에 위임할 수 있는 작업에 대해 제한을 설정하거나 가드레일을 정의합니다. SCP를 허용 목록 또는 거부 목록으로 사용하여 허용하거나 금지할 서비스 또는 작업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 서비스 제어 정책을 참조하십시오.

서비스 엔드포인트

의 진입점 URL입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 AWS 서비스 엔드포인트를 참조하십시오.

서비스 수준에 관한 계약(SLA)

IT 팀이 고객에게 제공하기로 약속한 내용(예: 서비스 가동 시간 및 성능)을 명시한 계약입니다.

서비스 수준 표시기 (SLI)

오류율, 가용성 또는 처리량과 같은 서비스의 성능 측면을 측정하는 것입니다.

서비스 수준 목표 (SLO)

서비스 수준 지표로 측정되는 서비스 상태를 나타내는 대상 지표입니다.

공동 책임 모델

클라우드 보안 및 규정 준수에 AWS 대한 책임을 공유하는 것을 설명하는 모델입니다. AWS 클라우드의 보안을 책임지는 반면, 사용자는 클라우드에서의 보안을 담당합니다. 자세한 내용은 공동 책임 모델을 참조하십시오.

시엠

보안 정보 및 이벤트 관리 시스템을 참조하십시오.

단일 장애 지점 (SPOF)

응용 프로그램의 중요한 단일 구성 요소에서 발생한 오류로 인해 시스템이 중단될 수 있습니다.

SLA

SLA (서비스 수준 계약) 를 참조하십시오.

SLI

서비스 수준 표시기 참조.

SLO

서비스 수준 목표를 참조하십시오.

split-and-seed 모델

현대화 프로젝트를 확장하고 가속화하기 위한 패턴입니다. 새로운 기능과 제품 릴리스가 정의되면 핵심 팀이 분할되어 새로운 제품 팀이 만들어집니다. 이를 통해 조직의 역량과 서비스 규모를 조정하고, 개발자 생산성을 개선하고, 신속한 혁신을 지원할 수 있습니다. 자세한 내용은 의 애플리케이션 현대화를 위한 단계별 접근 방식을 참조하십시오. AWS 클라우드

SPOF

단일 장애 지점 보기

스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정 데이터를 저장하고 하나 이상의 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 구성 구조입니다. 이 구조는 데이터 웨어하우스에서 사용하거나 비즈니스 인텔리전스 용도로 설계되었습니다.

Strangler Fig 패턴

레거시 시스템을 폐기할 수 있을 때까지 시스템 기능을 점진적으로 다시 작성하고 교체하여 모놀리식 시스템을 현대화하기 위한 접근 방식. 이 패턴은 무화과 덩굴이 나무로 자라 결국 숙주를 압도하고 대체하는 것과 비슷합니다. Martin Fowler가 모놀리식 시스템을 다시 작성할 때 위험을 관리하는 방법으로 이 패턴을 도입했습니다. 이 패턴을 적용하는 방법의 예는 컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화하는 방법을 참조하십시오.

서브넷

VPC의 IP 주소 범위입니다. 서브넷은 단일 가용 영역에 상주해야 합니다.

감독 통제 및 데이터 수집 (SCADA)

제조 시 하드웨어와 소프트웨어를 사용하여 물리적 자산과 생산 작업을 모니터링하는 시스템입니다.

대칭 암호화

동일한 키를 사용하여 데이터를 암호화하고 복호화하는 암호화 알고리즘입니다.

합성 테스트

잠재적 문제를 감지하거나 성능을 모니터링하기 위해 사용자 상호 작용을 시뮬레이션하는 방식으로 시스템을 테스트합니다. Amazon CloudWatch Synthetics를 사용하여 이러한 테스트를 생성할 수 있습니다.

# T

tags

리소스 구성을 위한 메타데이터 역할을 하는 키-값 쌍. AWS 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다. 자세한 내용은 AWS 리소스에 태그 지정을 참조하십시오.

대상 변수

지도 ML에서 예측하려는 값으로, 결과 변수라고도 합니다. 예를 들어, 제조 설정에서 대상 변수는 제품 결함일 수 있습니다.

작업 목록

런북을 통해 진행 상황을 추적하는 데 사용되는 도구입니다. 작업 목록에는 런북의 개요와 완료해야 할 일반 작업 목록이 포함되어 있습니다. 각 일반 작업에 대한 예상 소요 시간, 소유자 및 진행 상황이 작업 목록에 포함됩니다.

테스트 환경

환경을 참조하십시오.

훈련

ML 모델이 학습할 수 있는 데이터를 제공하는 것입니다. 훈련 데이터에는 정답이 포함되어야 합니다. 학습 알고리즘은 훈련 데이터에서 대상(예측하려는 답)에 입력 데이터 속성을 매핑하는 패턴을 찾고, 이러한 패턴을 캡처하는 ML 모델을 출력합니다. 그런 다음 ML 모델을 사용하여 대상을 모르는 새 데이터에 대한 예측을 할 수 있습니다.

전송 게이트웨이

VPC와 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 트랜짓 게이트웨이란 무엇입니까? 를 참조하십시오.

트렁크 기반 워크플로

개발자가 기능 브랜치에서 로컬로 기능을 구축하고 테스트한 다음 해당 변경 사항을 기본 브랜치에 병합하는 접근 방식입니다. 이후 기본 브랜치는 개발, 프로덕션 이전 및 프로덕션 환경에 순차적으로 구축됩니다.

신뢰할 수 있는 액세스

조직 내 AWS Organizations 및 해당 계정에서 사용자를 대신하여 작업을 수행하도록 지정한 서비스에 권한 부여 신뢰할 수 있는 서비스는 필요할 때 각 계정에 서비스 연결 역할을 생성하여 관

리 작업을 수행합니다. 자세한 내용은 AWS Organizations 설명서의 다른 AWS 서비스와 AWS Organizations 함께 사용을 참조하십시오.

튜닝

ML 모델의 정확도를 높이기 위해 훈련 프로세스의 측면을 여러 변경하는 것입니다. 예를 들어, 레이블링 세트를 생성하고 레이블을 추가한 다음 다양한 설정에서 이러한 단계를 여러 번 반복하여 모델을 최적화하는 방식으로 ML 모델을 훈련할 수 있습니다.

피자 두 판 팀

피자 두 판만 들고 배불리 먹을 수 있는 소규모 DevOps 팀 피자 두 판 팀 규모는 소프트웨어 개발에 있어 가능한 최상의 공동 작업 기회를 보장합니다.

# U

불확실성

예측 ML 모델의 신뢰성을 저해할 수 있는 부정확하거나 불완전하거나 알려지지 않은 정보를 나타내는 개념입니다. 불확실성에는 두 가지 유형이 있습니다. 인식론적 불확실성은 제한적이고 불완전한 데이터에 의해 발생하는 반면, 우연한 불확실성은 데이터에 내재된 노이즈와 무작위성에 의해 발생합니다. 자세한 내용은 Quantifying uncertainty in deep learning systems 가이드를 참조하십시오.

차별화되지 않은 작업

애플리케이션을 만들고 운영하는 데 필요하지만 최종 사용자에게 직접적인 가치를 제공하거나 경쟁 우위를 제공하지 못하는 작업을 헤비 리프팅이라고도 합니다. 차별화되지 않은 작업의 예로는 조달, 유지보수, 용량 계획 등이 있습니다.

상위 환경

환경을 보세요.

# V

정리

스토리지를 회수하고 성능을 향상시키기 위해 증분 업데이트 후 정리 작업을 수반하는 데이터베이스 유지 관리 작업입니다.

버전 제어

리포지토리의 소스 코드 변경과 같은 변경 사항을 추적하는 프로세스 및 도구입니다.

VPC 피어링

프라이빗 IP 주소를 사용하여 트래픽을 라우팅할 수 있게 하는 두 VPC 간의 연결입니다. 자세한 내용은 Amazon VPC 설명서의 VPC 피어링이란?을 참조하십시오.

취약성

시스템 보안을 손상시키는 소프트웨어 또는 하드웨어 결함입니다.

## W

웜 캐시

자주 액세스하는 최신 관련 데이터를 포함하는 버퍼 캐시입니다. 버퍼 캐시에서 데이터베이스 인스턴스를 읽을 수 있기 때문에 주 메모리나 디스크에서 읽는 것보다 빠릅니다.

웜 데이터

자주 액세스하지 않는 데이터입니다. 이런 종류의 데이터를 쿼리할 때는 일반적으로 적절히 느린 쿼리가 허용됩니다.

윈도우 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에 대해 계산을 수행하는 SQL 함수입니다. 윈도우 함수는 이동 평균을 계산하거나 현재 행의 상대적 위치를 기반으로 행 값에 액세스하는 등의 작업을 처리하는 데 유용합니다.

워크로드

고객 대면 애플리케이션이나 백엔드 프로세스 같이 비즈니스 가치를 창출하는 리소스 및 코드 모음입니다.

워크스트림

마이그레이션 프로젝트에서 특정 작업 세트를 담당하는 직무 그룹입니다. 각 워크스트림은 독립적이지만 프로젝트의 다른 워크스트림을 지원합니다. 예를 들어, 포트폴리오 워크스트림은 애플리케이션 우선순위 지정, 웨이브 계획, 마이그레이션 메타데이터 수집을 담당합니다. 포트폴리오 워크스트림은 이러한 자산을 마이그레이션 워크스트림에 전달하고, 마이그레이션 워크스트림은 서버와 애플리케이션을 마이그레이션합니다.

웜

한 번 쓰고, 많이 읽으세요.

WQF

AWS 워크로드 검증 프레임워크를 참조하십시오.

한 번 작성하고 여러 번 읽기 (WORM)

데이터를 한 번 쓰고 데이터가 삭제되거나 수정되지 않도록 하는 스토리지 모델입니다. 인증된 사용자는 필요한 만큼 데이터를 여러 번 읽을 수 있지만 변경할 수는 없습니다. 이 데이터 스토리지 인프라는 변경할 수 없는 것으로 간주됩니다.

# Z

제로데이 익스플로잇

제로데이 취약점을 악용하는 공격 (일반적으로 멀웨어) 입니다.

제로데이 취약성

프로덕션 시스템의 명백한 결함 또는 취약성입니다. 위협 행위자는 이러한 유형의 취약성을 사용하여 시스템을 공격할 수 있습니다. 개발자는 공격의 결과로 취약성을 인지하는 경우가 많습니다.

좀비 애플리케이션

평균 CPU 및 메모리 사용량이 5% 미만인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하는 것이 일반적입니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.