



다중 계정 DevOps 환경을 위한 Git 분기 전략 선택

AWS 규범적 지침



AWS 규범적 지침: 다중 계정 DevOps 환경을 위한 Git 분기 전략 선택

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

소개	1
목표	1
CI/CD 사례 사용	2
DevOps환경에 대한 이해	3
샌드박스 환경	4
액세스	4
빌드 단계	4
배포 단계	4
개발 환경으로 전환하기 전의 기대치	5
개발 환경	5
액세스	4
빌드 단계	4
배포 단계	4
테스트 환경으로 이동하기 전의 기대치	6
테스트 환경	6
액세스	4
빌드 단계	4
배포 단계	4
스테이징 환경으로 전환하기 전의 기대치	7
스테이징 환경	8
액세스	4
빌드 단계	4
배포 단계	4
프로덕션 환경으로 전환하기 전의 기대치	8
프로덕션 환경	9
액세스	4
빌드 단계	4
배포 단계	4
Git 기반 개발 모범 사례	10
Git 브랜칭 전략	12
트렁크 브랜칭 전략	12
트렁크 전략의 시각적 개요	13
트렁크 전략의 브랜치	14
트렁크 전략의 장점 및 단점	16

GitHub 플로우 브랜칭 전략	18
GitHub 플로우 전략의 시각적 개요	19
플로우 전략의 브랜치 GitHub	20
GitHub 플로우 전략의 장점 및 단점	22
Gitflow 브랜칭 전략	24
깃플로우 전략의 시각적 개요	25
깃플로우 전략의 브랜치	27
깃플로우 전략의 장점 및 단점	30
다음 단계	33
리소스	34
AWS 규범적 지침	34
기타 지침 AWS	34
기타 리소스	34
기여자	36
작성	36
리뷰	36
테크니컬 라이팅	36
문서 기록	37
용어집	38
#	38
A	39
B	41
C	43
D	46
E	50
F	52
G	53
H	54
I	55
L	57
M	58
O	62
P	64
Q	66
R	67
S	69

T	73
U	74
V	74
W	75
Z	76
.....	lxxvii

다중 계정 환경을 위한 Git 브랜칭 전략 선택 DevOps

Amazon Web Services ([기고자](#))

2024년 2월 ([문서 기록](#))

클라우드 기반 접근 방식으로 전환하고 소프트웨어 솔루션을 제공하는 것은 혁신적인 AWS 수 있습니다. 소프트웨어 개발 라이프사이클 프로세스를 변경해야 할 수도 있습니다. 일반적으로에서는 개발 프로세스 중에 여러 AWS 계정 개가 사용됩니다. AWS 클라우드 DevOps 프로세스에 맞는 호환 가능한 Git 브랜칭 전략을 선택하는 것이 성공에 필수적입니다. 조직에 적합한 Git 브랜칭 전략을 선택하면 개발 팀 간에 DevOps 표준과 모범 사례를 간결하게 전달할 수 있습니다. Git 브랜칭은 단일 환경에서는 간단할 수 있지만 샌드박스, 개발, 테스트, 스테이징, 프로덕션 환경 등 여러 환경에 적용할 경우 혼란스러울 수 있습니다. 환경이 여러 개 있으면 구현이 더 복잡해집니다. DevOps

이 가이드에서는 조직에서 다중 계정 프로세스를 구현하는 방법을 보여주는 Git 브랜칭 전략의 시각적 다이어그램을 제공합니다. DevOps 시각적 가이드는 팀이 Git 브랜칭 전략을 실제 사례와 통합하는 방법을 이해하는 데 도움이 됩니다. DevOps 소스 코드 리포지토리를 관리하기 위해 GitHub Gitflow, Flow 또는 Trunk와 같은 표준 브랜칭 모델을 사용하면 개발팀이 작업을 조율하는 데 도움이 됩니다. 또한 이러한 팀은 인터넷의 표준 Git 교육 리소스를 사용하여 이러한 모델 및 전략을 이해하고 구현할 수 있습니다.

에 대한 DevOps 모범 사례는 AWS AWS Well-Architected의 [DevOps 지침](#)을 검토하십시오. 이 가이드를 검토하면서 실사를 통해 조직에 적합한 브랜칭 전략을 선택하십시오. 일부 전략은 다른 전략보다 사용 사례에 더 적합할 수 있습니다.

목표

이 가이드는 여러 AWS 계정조직을 대상으로 DevOps 분기 전략을 선택하고 구현하는 방법에 대한 설명서 시리즈의 일부입니다. 이 시리즈는 처음부터 요구 사항, 목표 및 모범 사례에 가장 잘 맞는 전략을 적용하여 경험을 간소화할 수 있도록 설계되었습니다. AWS 클라우드 DevOps 실행 스크립트는 조직에서 사용하는 지속적 통합 및 지속적 전달 (CI/CD) 엔진 및 기술 프레임워크에 따라 달라지므로 이 가이드에는 실행 가능한 스크립트가 포함되어 있지 않습니다.

이 가이드에서는 세 가지 일반적인 Git 브랜칭 전략인 플로우, GitHub 깃플로우, 트렁크 간의 차이점을 설명합니다. 이 가이드의 권장 사항은 팀이 조직 목표에 맞는 브랜칭 전략을 식별하는 데 도움이 됩니다. 이 가이드를 검토한 후 조직에 맞는 브랜칭 전략을 선택할 수 있을 것입니다. 전략을 선택한 후 다음 패턴 중 하나를 사용하여 개발 팀과 함께 해당 전략을 구현할 수 있습니다.

- [다중 DevOps 계정 환경을 위한 트렁크 브랜칭 전략 구현](#)
- [다중 계정 환경을 위한 GitHub Flow 분기 전략 구현 DevOps](#)
- [다중 계정 환경을 위한 Gitflow 분기 전략 구현 DevOps](#)

한 조직, 팀 또는 프로젝트에 효과가 있는 것이 다른 조직에는 적합하지 않을 수 있다는 점에 유의해야 합니다. Git 브랜칭 전략 중 선택은 팀 규모, 프로젝트 요구 사항, 공동 작업, 통합 빈도, 릴리스 관리 간의 원하는 균형과 같은 다양한 요소에 따라 달라집니다.

CI/CD 사례 사용

AWS 소프트웨어 릴리스 라이프사이클을 자동화하는 프로세스인 지속적 통합 및 지속적 전달 (CI/CD) 을 구현할 것을 권장합니다. 새로운 코드를 개발부터 프로덕션까지 가져오는 데 기존에 필요했던 수동 DevOps 프로세스의 대부분 또는 전체를 자동화합니다. CI/CD 파이프라인은 샌드박스, 개발, 테스트, 스테이징 및 프로덕션 환경을 포함합니다. 각 환경에서 CI/CD 파이프라인은 코드를 배포하거나 테스트하는 데 필요한 모든 인프라를 제공합니다. 개발팀은 CI/CD를 사용하여 코드를 변경한 다음 자동으로 테스트 및 배포할 수 있습니다. 또한 CI/CD 파이프라인은 개발팀을 위한 거버넌스 및 가드레일을 제공합니다. 기능 수용 및 배포에 대한 일관성, 표준, 모범 사례, 최소 허용 수준을 적용합니다. 자세한 내용은 [지속적 통합 및 지속적 전달 실천](#)을 참조하십시오. AWS

이 가이드에서 설명하는 모든 분기 전략은 CI/CD 관행에 매우 적합합니다. CI/CD 파이프라인의 복잡성은 브랜칭 전략의 복잡성에 따라 증가합니다. 예를 들어 Gitflow는 이 가이드에서 설명하는 가장 복잡한 분기 전략입니다. 이 전략을 위한 CI/CD 파이프라인은 규정 준수 등의 이유로 더 많은 단계가 필요하며 여러 개의 동시 프로덕션 릴리스를 지원해야 합니다. 브랜칭 전략이 복잡해짐에 따라 CI/CD를 사용하는 것도 더욱 중요해집니다. 이는 CI/CD가 개발팀을 위한 가드레일과 메커니즘을 설정하여 개발자가 의도적으로 또는 의도하지 않게 정의된 프로세스를 우회하지 못하도록 하기 때문입니다.

AWS CI/CD 파이프라인을 구축하는 데 도움이 되도록 설계된 개발자 서비스 제품군을 제공합니다. 예를 들어 [AWS CodePipeline](#)는 빠르고 안정적인 애플리케이션 및 인프라 업데이트를 위해 릴리스 파이프라인을 자동화하는 데 도움이 되는 완전관리형 지속적 전달 서비스입니다. [AWS CodeCommit](#) 확장 가능한 Git 리포지토리를 안전하게 호스팅하고, 소스 코드를 [AWS CodeBuild](#) 컴파일하고, 테스트를 실행하고, 소프트웨어 패키지를 생성하도록 설계되었습니다. ready-to-deploy [자세한 내용은 개발자 도구 on](#)을 참조하십시오. AWS

DevOps환경에 대한 이해

브랜칭 전략을 이해하려면 각 환경에서 발생하는 목적과 활동을 이해해야 합니다. 여러 환경을 설정하면 개발 활동을 단계별로 분리하고, 해당 활동을 모니터링하고, 승인되지 않은 기능이 의도치 않게 릴리스되는 것을 방지할 수 있습니다. 각 환경에 하나 이상을 둘 수 있습니다 AWS 계정 .

대부분의 조직에는 사용할 수 있는 몇 가지 환경이 설명되어 있습니다. 그러나 환경 수는 조직 및 소프트웨어 개발 정책에 따라 달라질 수 있습니다. 이 설명서 시리즈에서는 서로 다른 이름으로 불릴 수도 있지만 개발 파이프라인에 걸쳐 다음과 같은 5가지 공통 환경이 있다고 가정합니다.

- 샌드박스 — 개발자가 코드를 작성하고, 실수를 저지르고, 개념 증명 작업을 수행하는 환경입니다.
- 개발 — 개발자가 코드를 통합하여 모든 코드가 하나의 응집력 있는 애플리케이션으로 작동하는지 확인하는 환경입니다.
- 테스트 — QA 팀 또는 승인 테스트가 이루어지는 환경입니다. 팀은 종종 이 환경에서 성능 또는 통합 테스트를 수행합니다.
- 스테이징 — 프로덕션 환경과 동일한 환경에서 코드와 인프라가 예상대로 작동하는지 검증하는 사전 프로덕션 환경입니다. 이 환경은 프로덕션 환경과 최대한 유사하도록 구성되어 있습니다.
- 프로덕션 — 최종 사용자와 고객의 트래픽을 처리하는 환경입니다.

이 섹션에서는 각 환경에 대해 자세히 설명합니다. 또한 다음 단계로 넘어갈 수 있도록 각 환경의 빌드 단계, 배포 단계 및 종료 기준에 대해서도 설명합니다. 다음 이미지는 이러한 환경을 순서대로 보여줍니다.



이 단원의 주제:

- [샌드박스 환경](#)
- [개발 환경](#)
- [테스트 환경](#)
- [스테이징 환경](#)
- [프로덕션 환경](#)

샌드박스 환경

샌드박스 환경은 개발자가 코드를 작성하고, 실수를 저지르고, 개념 증명 작업을 수행하는 곳입니다. 로컬 워크스테이션이나 로컬 워크스테이션의 스크립트를 통해 샌드박스 환경에 배포할 수 있습니다.

액세스

개발자는 샌드박스 환경에 대한 전체 액세스 권한을 가져야 합니다.

빌드 단계

개발자는 샌드박스 환경에 변경 내용을 배포할 준비가 되면 로컬 워크스테이션에서 수동으로 빌드를 실행합니다.

1. [git-secrets](#) (GitHub) 를 사용하여 민감한 정보를 스캔할 수 있습니다.
2. 소스 코드 린트
3. 해당하는 경우 소스 코드를 빌드하고 컴파일합니다.
4. 유닛 테스트 수행
5. 코드 커버리지 분석 수행
6. 정적 코드 분석 수행
7. 코드형 인프라 구축 (IaC)
8. IaC 보안 분석 수행
9. 오픈소스 라이선스 추출
10. 빌드 아티팩트 퍼블리싱

배포 단계

Gitflow 또는 Trunk 모델을 사용하는 경우 샌드박스 환경에서 feature 브랜치가 성공적으로 빌드되면 배포 단계가 자동으로 시작됩니다. GitHub Flow 모델을 사용하는 경우 다음 배포 단계를 수동으로 수행합니다. 샌드박스 환경에서의 배포 단계는 다음과 같습니다.

1. 게시된 아티팩트 다운로드
2. 데이터베이스 버전 관리 수행
3. IaC 배포 수행

4. 통합 테스트 수행

개발 환경으로 전환하기 전의 기대치

- 샌드박스 환경에서 feature 브랜치를 성공적으로 구축했습니다.
- 개발자가 샌드박스 환경에서 기능을 수동으로 배포하고 테스트했습니다.

개발 환경

개발 환경은 개발자가 코드를 통합하여 모든 코드가 하나의 응집력 있는 애플리케이션처럼 작동하도록 하는 곳입니다. Gitflow의 개발 환경에는 병합 요청에 포함된 최신 기능이 포함되어 있으며 출시 준비가 완료되었습니다. GitHub Flow 및 Trunk 전략에서 개발 환경은 테스트 환경으로 간주되며 코드 베이스가 불안정하여 프로덕션 환경에 배포하기에 적합하지 않을 수 있습니다.

액세스

최소 권한 원칙에 따라 권한을 할당하세요. 최소 권한은 작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 개발자는 샌드박스 환경에 액세스할 수 있는 권한보다 개발 환경에 대한 액세스 권한이 더 적어야 합니다.

빌드 단계

develop브랜치 (Gitflow) 또는 브랜치 (트렁크 또는 GitHub 플로우) 에 main 대한 병합 요청을 생성하면 빌드가 자동으로 시작됩니다.

1. [git-secrets](#) (GitHub) 를 사용하여 민감한 정보를 스캔할 수 있습니다.
2. 소스 코드 린트
3. 해당하는 경우 소스 코드를 빌드하고 컴파일합니다.
4. 유닛 테스트 수행
5. 코드 커버리지 분석 수행
6. 정적 코드 분석 수행
7. IaC 빌드
8. IaC 보안 분석 수행
9. 오픈소스 라이선스 추출

배포 단계

Gitflow 모델을 사용하는 경우 개발 환경에 develop 브랜치가 성공적으로 구축되면 배포 단계가 자동으로 시작됩니다. GitHub Flow 모델 또는 Trunk 모델을 사용하는 경우 브랜치에 대해 병합 요청이 생성되면 배포 단계가 자동으로 시작됩니다. main 개발 환경에서의 배포 단계는 다음과 같습니다.

1. 빌드 단계에서 게시된 아티팩트를 다운로드합니다.
2. 데이터베이스 버전 관리 수행
3. IaC 배포 수행
4. 통합 테스트 수행

테스트 환경으로 이동하기 전의 기대치

- 개발 환경에서 develop 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 의 성공적인 빌드 및 배포
- 유닛 테스트 통과율 100%
- 성공적인 IaC 빌드
- 배포 아티팩트가 성공적으로 생성되었습니다.
- 개발자가 수동 검증을 수행하여 기능이 예상대로 작동하는지 확인했습니다.

테스트 환경

품질 보증 (QA) 담당자는 테스트 환경을 사용하여 기능을 검증합니다. 테스트를 마친 후 변경 사항을 승인합니다. 승인하면 지사는 다음 환경인 스테이징으로 이동합니다. Gitflow에서는 이 환경 및 그 이상의 다른 환경을 브랜치에서만 배포할 수 있습니다. release 브랜치는 계획된 기능을 포함하는 develop 브랜치를 기반으로 합니다.

액세스

최소 권한 원칙에 따라 권한을 할당하세요. 개발자는 개발 환경에 대한 액세스 권한보다 테스트 환경에 대한 액세스 권한이 더 적어야 합니다. QA 담당자가 기능을 테스트하려면 충분한 권한이 필요합니다.

빌드 단계

이 환경의 빌드 프로세스는 Gitflow 전략을 사용할 때의 버그 수정에만 적용됩니다. bugfix 브랜치에 병합 요청을 생성하면 빌드가 자동으로 시작됩니다.

1. [git-secrets](#) (GitHub) 를 사용하여 민감한 정보를 스캔할 수 있습니다.
2. 소스 코드 린트
3. 해당하는 경우 소스 코드를 빌드하고 컴파일합니다.
4. 유닛 테스트 수행
5. 코드 커버리지 분석 수행
6. 정적 코드 분석 수행
7. IaC 빌드
8. IaC 보안 분석 수행
9. 오픈소스 라이선스 추출

배포 단계

개발 환경에 배포한 후 테스트 환경에서 release 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 배포를 자동으로 시작합니다. 테스트 환경에서의 배포 단계는 다음과 같습니다.

1. 테스트 환경에 release 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 를 배포합니다.
2. 지정된 직원의 수동 승인을 위해 일시 중지하십시오.
3. 게시된 아티팩트 다운로드
4. 데이터베이스 버전 관리 수행
5. IaC 배포 수행
6. 통합 테스트 수행
7. 성능 테스트 수행
8. 품질 보증 승인

스테이징 환경으로 전환하기 전의 기대치

- 개발 및 QA 팀은 조직의 요구 사항을 충족하기에 충분한 테스트를 수행했습니다.
- 개발팀은 bugfix 브랜치를 통해 발견된 버그를 모두 해결했습니다.

스태이징 환경

스태이징 환경은 프로덕션 환경과 동일하게 구성되어 있습니다. 예를 들어 데이터 설정은 범위와 크기가 프로덕션 워크로드와 비슷해야 합니다. 스테이징 환경을 사용하여 코드와 인프라가 예상대로 작동하는지 확인하십시오. 이 환경은 미리 보기 또는 고객 데모와 같은 비즈니스 사용 사례에도 선호되는 선택입니다.

액세스

최소 권한 원칙에 따라 권한을 할당하세요. 개발자는 프로덕션 환경과 마찬가지로 스테이징 환경에 액세스할 수 있어야 합니다.

빌드 단계

없음. 테스트 환경에서 사용된 것과 동일한 아티팩트가 스테이징 환경에서도 재사용됩니다.

배포 단계

테스트 환경에서 승인 및 배포한 후 스테이징 환경에서 release 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 배포를 자동으로 시작합니다. 스테이징 환경에서의 배포 단계는 다음과 같습니다.

1. 스테이징 환경에 release 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 를 배포합니다.
2. 지정된 직원의 수동 승인을 위해 일시 중지합니다.
3. 게시된 아티팩트 다운로드
4. 데이터베이스 버전 관리 수행
5. IaC 배포 수행
6. (선택 사항) 통합 테스트 수행
7. (선택 사항) 부하 테스트 수행
8. 필요한 개발, QA, 제품 또는 비즈니스 승인자로부터 승인 받기

프로덕션 환경으로 전환하기 전의 기대치

- 프로덕션에 해당하는 릴리스가 스테이징 환경에 성공적으로 배포되었습니다.
- (선택 사항) 통합 및 부하 테스트 성공

프로덕션 환경

프로덕션 환경은 실제 고객의 실제 데이터를 처리하여 출시된 제품을 지원합니다. 이 환경은 최소 권한으로 액세스 권한이 할당되는 보호된 환경이므로 제한된 기간 동안 감사를 거친 예외 프로세스를 통해서만 승격된 액세스가 허용되어야 합니다.

액세스

프로덕션 환경에서 개발자는 AWS Management Console에서 제한된 읽기 전용 액세스 권한을 가져야 합니다. 예를 들어 개발자는 day-to-day 작업을 위해 로그 데이터에 액세스할 수 있어야 합니다. 프로덕션에 대한 모든 릴리스는 배포 전 승인 단계를 거쳐 승인되어야 합니다.

빌드 단계

없음. 테스트 및 스테이징 환경에서 사용된 것과 동일한 아티팩트가 프로덕션 환경에서도 재사용됩니다.

배포 단계

스테이징 환경에서 승인 및 배포한 후 프로덕션 환경에서 release 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 배포를 자동으로 시작합니다. 프로덕션 환경에서의 배포 단계는 다음과 같습니다.

1. 프로덕션 환경에 release 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 를 배포합니다.
2. 지정된 직원의 수동 승인을 위해 일시 중지합니다.
3. 게시된 아티팩트 다운로드
4. 데이터베이스 버전 관리 수행
5. IaC 배포 수행

Git 기반 개발 모범 사례

Git 기반 개발을 성공적으로 채택하려면 협업을 촉진하고, 코드 품질을 유지하고, 지속적 통합 및 지속적 전달 (CI/CD) 을 지원하는 일련의 모범 사례를 따르는 것이 중요합니다. 이 가이드의 모범 사례 외에도 [AWS DevOps Well-Architected](#) 지침을 검토하십시오. 다음은 Git 기반 개발을 위한 몇 가지 주요 모범 사례입니다. AWS

- 변경 사항을 작고 빈번하게 유지 — 개발자가 소규모의 점진적인 변경이나 기능을 적용하도록 장려하세요. 이렇게 하면 병합 충돌의 위험이 줄어들고 문제를 더 쉽게 식별하고 신속하게 수정할 수 있습니다.
- 기능 토글 사용 — 불완전하거나 실험적인 기능의 릴리스를 관리하려면 기능 토글 또는 기능 플래그를 사용하십시오. 이를 통해 메인 브랜치의 안정성에 영향을 주지 않으면서 프로덕션 환경에서 특정 기능을 숨기거나 활성화하거나 비활성화할 수 있습니다.
- 강력한 테스트 도구 모음 유지 — 포괄적이고 잘 관리된 테스트 모음은 문제를 조기에 발견하고 코드 베이스가 안정적으로 유지되는지 확인하는 데 매우 중요합니다. 테스트 자동화에 투자하고 실패한 테스트를 우선적으로 해결하세요.
- 지속적 통합 수용 — 지속적 통합 도구 및 프랙티스를 사용하여 코드 변경 사항을 develop 브랜치 (Gitflow) 또는 main 브랜치 (Trunk 또는 Flow) 에 자동으로 빌드, 테스트 및 통합할 수 있습니다. GitHub 이를 통해 문제를 조기에 파악하고 개발 프로세스를 간소화할 수 있습니다.
- 코드 검토 수행 — 코드가 main 브랜치에 통합되기 전에 품질을 유지하고, 지식을 공유하고, 잠재적 문제를 파악할 수 있도록 코드에 대한 동료 검토를 장려합니다. 풀 리퀘스트 또는 기타 코드 리뷰 도구를 사용하여 이 프로세스를 원활하게 진행하세요.
- 손상된 빌드 모니터링 및 수정 — 빌드가 중단되거나 테스트가 실패하는 경우 최대한 빨리 문제를 해결하는 데 우선 순위를 두세요. 이렇게 하면 develop 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 를 릴리스 가능한 상태로 유지하고 다른 개발자에게 미치는 영향을 최소화할 수 있습니다.
- 커뮤니케이션 및 협업 — 팀원 간의 열린 의사소통과 협업을 촉진하세요. 개발자가 진행 중인 작업과 코드 베이스의 변경 사항을 알 수 있도록 하세요.
- 지속적인 리팩터링 — 코드베이스를 정기적으로 리팩터링하여 유지 관리 용이성을 개선하고 기술적 부채를 줄이십시오. 개발자가 코드를 찾은 것보다 더 나은 상태로 유지하도록 권장하세요.
- 복잡한 작업에는 수명이 짧은 브랜치를 사용하세요. 규모가 크거나 복잡한 작업의 경우 수명이 짧은 브랜치 (태스크 브랜치라고도 함) 를 사용하여 변경 사항을 적용하세요. 하지만 브랜치 수명은 보통 하루 미만으로 짧게 유지해야 합니다. 가능한 한 빨리 변경 내용을 develop 브랜치 (Gitflow) 또는 main 브랜치 (트렁크 또는 GitHub 플로우) 에 다시 병합하십시오. 팀은 규모가 작고 빈번한 병합 요청을 한 번에 처리하는 것보다 병합과 검토를 더 쉽게 사용하고 처리할 수 있습니다.

- 팀 교육 및 지원 — Git 기반 개발을 처음 사용하거나 모범 사례를 채택하는 데 지침이 필요한 개발자에게 교육 및 지원을 제공합니다.

Git 브랜칭 전략

이 가이드에서는 다음과 같은 Git 기반 브랜칭 전략을 가장 복잡하지 않은 것부터 가장 복잡한 순서로 자세히 설명합니다.

- 트렁크 — 트렁크 기반 개발은 모든 개발자가 일반적으로 `or` 브랜치라고 하는 단일 브랜치에서 작업하는 소프트웨어 개발 방식입니다. `trunk main` 이 접근 방식의 기본 개념은 코드 변경 사항을 자주 통합하고 자동화된 테스트와 지속적인 통합을 통해 코드베이스를 지속적으로 릴리스할 수 있는 상태로 유지하는 것입니다.
- GitHub Flow — Flow는 에서 개발한 가벼운 분기 기반 워크플로입니다. GitHub 수명이 짧은 feature 브랜치라는 아이디어를 기반으로 합니다. 기능이 완성되어 배포할 준비가 되면 기능이 브랜치에 병합됩니다. `main`
- Gitflow — Gitflow 접근 방식을 사용하면 개별 기능 브랜치에서 개발을 완료할 수 있습니다. 승인 후에는 feature 브랜치를 통합 브랜치로 병합합니다. 통합 브랜치는 보통 이름이 지정됩니다. `develop` `develop` 브랜치에 충분한 기능이 누적되면 `release` 브랜치를 만들어 상위 환경에 기능을 배포합니다.

각 브랜칭 전략에는 장단점이 있습니다. 모두 동일한 환경을 사용하지만 모두 동일한 지사 또는 수동 승인 단계를 사용하지는 않습니다. 가이드의 이 섹션에서는 각 브랜칭 전략을 자세히 검토하여 각 브랜칭 전략의 니앙스를 파악하고 조직의 사용 사례에 맞는지 평가할 수 있도록 하세요.

이 단원의 주제:

- [트렁크 브랜칭 전략](#)
- [GitHub 플로우 브랜칭 전략](#)
- [Gitflow 브랜칭 전략](#)

트렁크 브랜칭 전략

트렁크 기반 개발은 모든 개발자가 일반적으로 `or` 브랜치라고 하는 단일 브랜치에서 작업하는 소프트웨어 개발 방식입니다. `trunk main` 이 접근 방식의 기본 개념은 코드 변경 사항을 자주 통합하고 자동화된 테스트와 지속적인 통합을 통해 코드 베이스를 지속적으로 릴리스할 수 있는 상태로 유지하는 것입니다.

트렁크 기반 개발에서는 개발자가 소량의 점진적 업데이트를 목표로 하루에 여러 번 `main` 브랜치에 변경 사항을 커밋합니다. 이를 통해 피드백 루프가 빨라지고, 병합 충돌 위험이 줄어들고, 팀원 간의 협

업이 촉진됩니다. 이 방법에서는 자동화된 테스트를 통해 잠재적 문제를 조기에 발견하고 코드 베이스가 안정적이고 릴리스 가능한 상태로 유지되도록 하기 때문에 잘 관리되는 테스트 스위트의 중요성을 강조합니다.

트렁크 기반 개발은 기능 기반 개발 (기능 분기 또는 기능 기반 개발이라고도 함) 과 대조를 이루는 경우가 많습니다. 기능 기반 개발에서는 각각의 새로운 기능이나 버그 수정이 기본 브랜치와 분리된 자체 전용 브랜치에서 개발됩니다. 트렁크 기반 개발과 기능 기반 개발 간의 선택은 팀 규모, 프로젝트 요구 사항, 협업, 통합 빈도, 릴리스 관리 간의 원하는 균형과 같은 요인에 따라 달라집니다.

트렁크 브랜칭 전략에 대한 자세한 내용은 다음 리소스를 참조하십시오.

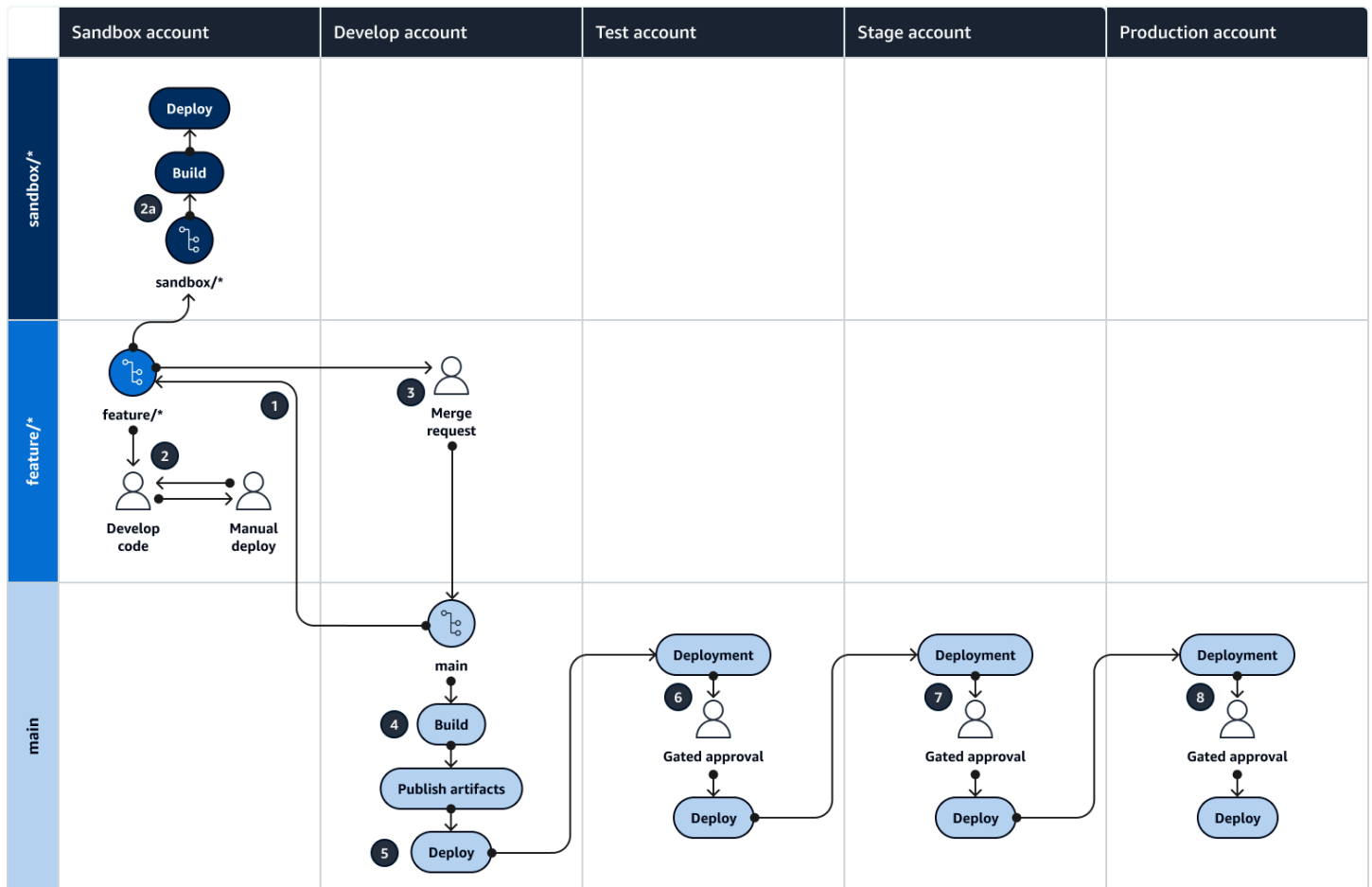
- [다중 계정 DevOps 환경을 위한 트렁크 브랜칭 전략 구현](#) (AWS 규범 지침)
- 트렁크 기반 [개발 소개](#) (트렁크 기반 개발 웹 사이트)

이 단원의 주제:

- [트렁크 전략의 시각적 개요](#)
- [트렁크 전략의 브랜치](#)
- [트렁크 전략의 장점 및 단점](#)

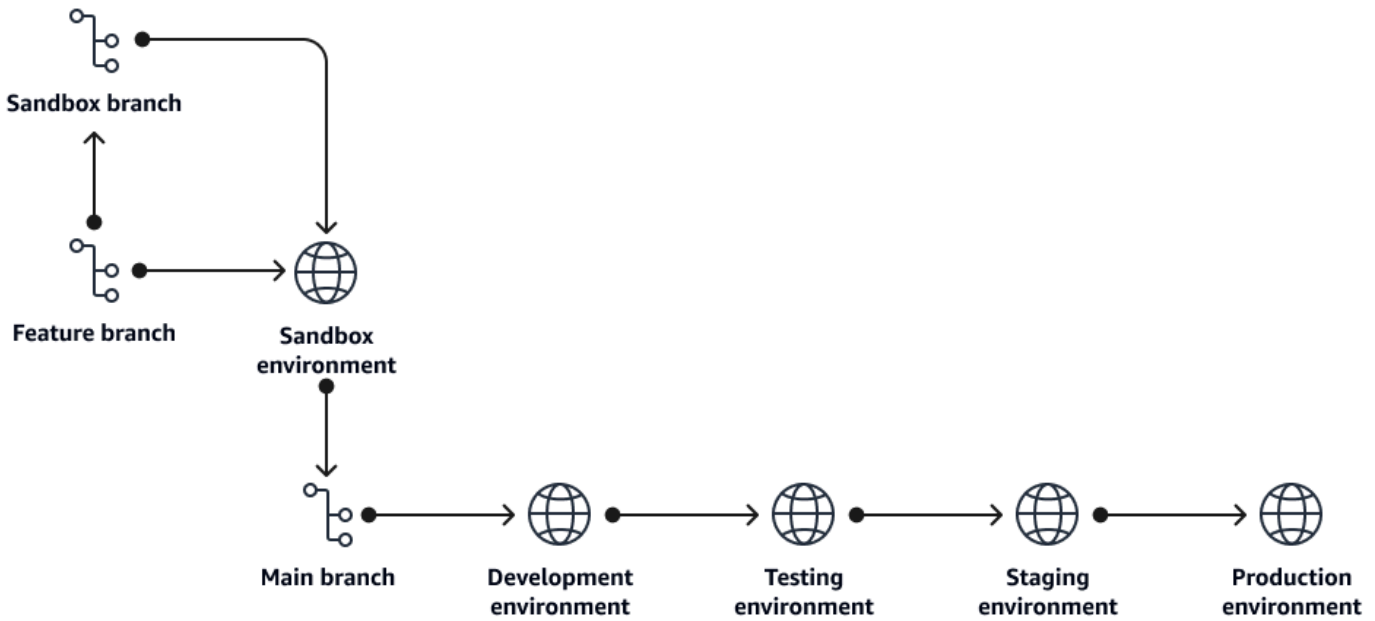
트렁크 전략의 시각적 개요

다음 다이어그램을 [퍼넷 스퀘어](#) (Wikipedia) 처럼 사용하여 트렁크 브랜칭 전략을 이해할 수 있습니다. 세로 축의 가치를 가로 축의 AWS 환경과 정렬하여 각 시나리오에서 수행할 작업을 결정하십시오. 원으로 표시된 숫자는 다이어그램에 표시된 동작 순서를 안내합니다. 이 다이어그램은 샌드박스 환경의 브랜치부터 feature 브랜치의 프로덕션 릴리스에 이르는 트렁크 브랜치 전략의 개발 워크플로를 보여줍니다. main 각 환경에서 발생하는 활동에 대한 자세한 내용은 이 가이드의 [DevOps 환경을](#) 참조하십시오.



트렁크 전략의 브랜치

트렁크 브랜칭 전략에는 일반적으로 다음과 같은 브랜치가 있습니다.



기능 브랜치

브랜치에서 기능을 개발하거나 핫픽스를 생성합니다. feature feature브랜치를 만들려면 브랜치를 벗어나서 브랜치를 생성해야 합니다main. 개발자는 feature 브랜치에서 코드를 반복하고, 커밋하고, 테스트합니다. 기능이 완성되면 개발자가 기능을 홍보합니다. feature브랜치에서 앞으로 나아갈 수 있는 경로는 두 가지뿐입니다.

- sandbox브랜치에 병합
- main브랜치에 병합 요청을 생성하세요.

명명 규칙:

feature/<story number>_<developer initials>_<descriptor>

명명 규칙 예제:

feature/123456_MS_Implement _Feature_A

샌드박스 브랜치

이 브랜치는 비표준 트렁크 브랜치이지만 CI/CD 파이프라인 개발에 유용합니다. sandbox브랜치는 주로 다음과 같은 용도로 사용됩니다.

- CI/CD 파이프라인을 사용하여 샌드박스 환경에 전체 배포를 수행합니다.

- 개발 또는 테스트와 같은 하위 환경에서의 전체 테스트를 위해 병합 요청을 제출하기 전에 파이프라인을 개발하고 테스트하세요.

Sandbox브랜치는 일시적이며 수명이 짧기 위한 것입니다. 특정 테스트가 완료된 후에는 삭제해야 합니다.

명명 규칙: `sandbox/<story number>_<developer initials>_<descriptor>`

명명 규칙 예제: `sandbox/123456_MS_Test_Pipeline_Deploy`

기본 브랜치

main브랜치는 항상 프로덕션 환경에서 실행 중인 코드를 나타냅니다. 코드는 브랜치되고 개발된 후 다시 병합됩니다. main main 의 배포는 모든 환경을 대상으로 main 할 수 있습니다. 삭제를 방지하려면 브랜치에 대한 브랜치 보호를 활성화하세요. main

명명 규칙: `main`

핫픽스 브랜치

트렁크 기반 워크플로에는 전용 hotfix 브랜치가 없습니다. 핫픽스는 브랜치를 사용합니다.

feature

트렁크 전략의 장점 및 단점

트렁크 브랜칭 전략은 강력한 커뮤니케이션 기술을 갖춘 작고 성숙한 개발 팀에 적합합니다. 또한 애플리케이션을 위한 지속적이고 순차적인 기능 릴리스가 있는 경우에도 잘 작동합니다. 개발 팀이 크거나 분산되어 있거나 대규모 예정된 기능 릴리스가 있는 경우에는 적합하지 않습니다. 이 모델에서는 병합 충돌이 발생할 수 있으므로 병합 충돌을 해결하는 것이 핵심 기술이라는 점에 유의하세요. 모든 팀원은 그에 따른 교육을 받아야 합니다.

장점

트렁크 기반 개발은 개발 프로세스를 개선하고, 협업을 간소화하고, 소프트웨어의 전반적인 품질을 향상시킬 수 있는 여러 가지 이점을 제공합니다. 다음은 몇 가지 주요 이점입니다.

- 더 빠른 피드백 루프 — 트렁크 기반 개발을 통해 개발자는 코드 변경 사항을 자주, 보통 하루에 여러 번 통합합니다. 이를 통해 잠재적 문제에 대한 피드백을 더 빠르게 받을 수 있으며 개발자가 기능 기반 개발 모델에서보다 더 빠르게 문제를 식별하고 수정할 수 있습니다.
- 병합 충돌 감소 — 트렁크 기반 개발에서는 변경 내용이 지속적으로 통합되므로 크고 복잡한 병합 충돌의 위험이 최소화됩니다. 이를 통해 코드베이스를 깔끔하게 유지하고 충돌 해결에 소요되는 시간을 줄일 수 있습니다. 기능 기반 개발에서는 충돌을 해결하는 데 시간이 많이 걸리고 오류가 발생하기 쉽습니다.
- 협업 개선 — 트렁크 기반 개발을 통해 개발자들은 같은 브랜치에서 함께 작업할 수 있어 팀 내 의사소통과 협력이 원활해집니다. 이를 통해 문제를 더 빠르게 해결하고 보다 응집력 있는 팀 역동성을 확보할 수 있습니다.
- 간편한 코드 검토 — 트렁크 기반 개발에서는 코드 변경이 더 작고 빈번하기 때문에 철저한 코드 검토를 더 쉽게 수행할 수 있습니다. 일반적으로 변경 내용이 작을수록 이해하고 검토하기가 쉬우므로 잠재적 문제를 더 효과적으로 식별하고 개선할 수 있습니다.
- 지속적 통합 및 전달 — 트렁크 기반 개발은 지속적 통합 및 지속적 전달 (CI/CD) 원칙을 지원합니다. 코드베이스를 릴리스 가능한 상태로 유지하고 변경 사항을 자주 통합함으로써 팀은 CI/CD 관행을 더 쉽게 채택할 수 있으며, 이를 통해 배포 주기를 단축하고 소프트웨어 품질을 개선할 수 있습니다.
- 코드 품질 향상 — 빈번한 통합, 테스트 및 코드 검토를 통해 트렁크 기반 개발을 통해 전반적인 코드 품질을 개선할 수 있습니다. 개발자는 문제를 더 빠르게 파악하고 해결할 수 있으므로 시간이 지남에 따라 기술 부채가 누적될 가능성을 줄일 수 있습니다.
- 간소화된 브랜칭 전략 — 트렁크 기반 개발은 수명이 긴 브랜치의 수를 줄여 브랜칭 전략을 단순화합니다. 따라서 특히 대규모 프로젝트 또는 팀의 경우 코드베이스를 더 쉽게 관리하고 유지할 수 있습니다.

단점

트렁크 기반 개발에는 개발 프로세스와 팀 역학에 영향을 미칠 수 있는 몇 가지 단점이 있습니다. 몇 가지 주목할 만한 단점은 다음과 같습니다.

- 제한적 격리 — 모든 개발자가 같은 브랜치에서 작업하기 때문에 모든 팀원이 변경 내용을 즉시 볼 수 있습니다. 이로 인해 간섭이나 충돌이 발생하여 의도하지 않은 부작용이 발생하거나 빌드가 중단될 수 있습니다. 반면 기능 기반 개발은 변경 사항을 더 잘 분리하여 개발자가 더 독립적으로 작업할 수 있도록 합니다.
- 테스트에 대한 부담 증가 — 트렁크 기반 개발에서는 지속적인 통합과 자동화된 테스트를 통해 문제를 신속하게 포착할 수 있습니다. 그러나 이 접근 방식은 테스트 인프라에 상당한 압력을 가할 수 있

으며 잘 관리된 테스트 스위트가 필요합니다. 테스트가 포괄적이거나 신뢰할 수 없는 경우 메인 브랜치에서 발견되지 않은 문제가 발생할 수 있습니다.

- 릴리스에 대한 통제력 감소 — 트렁크 기반 개발은 코드베이스를 지속적으로 릴리스할 수 있는 상태로 유지하는 것을 목표로 합니다. 이 방법이 유리할 수는 있지만 릴리스 일정이 엄격한 프로젝트나 특정 기능을 함께 릴리스해야 하는 프로젝트에는 적합하지 않을 수 있습니다. 기능 기반 개발을 통해 기능이 출시되는 시기와 방법을 더 잘 제어할 수 있습니다.
- 코드 이탈 — 개발자가 지속적으로 변경 사항을 메인 브랜치에 통합하는 상황에서 트렁크 기반 개발은 코드 이탈을 증가시킬 수 있습니다. 이로 인해 개발자가 코드베이스의 현재 상태를 추적하기 어려워지고 최근 변경의 영향을 이해하려고 할 때 혼란을 야기할 수 있습니다.
- 강력한 팀 문화 필요 — 트렁크 기반 개발에는 팀 구성원 간의 높은 수준의 규율, 커뮤니케이션 및 협업이 필요합니다. 특히 대규모 팀이나 이 접근 방식에 익숙하지 않은 개발자와 함께 작업하는 경우 유지 관리가 어려울 수 있습니다.
- 확장성 문제 — 개발 팀의 규모가 커지면 메인 브랜치에 통합되는 코드 변경 사항의 수가 급격히 증가할 수 있습니다. 이로 인해 빌드 중단과 테스트 실패가 더 자주 발생하여 코드베이스를 릴리스 가능한 상태로 유지하기가 어려울 수 있습니다.

GitHub 플로우 브랜칭 전략

GitHub Flow는 에서 개발한 가벼운 분기 기반 워크플로입니다. GitHub GitHubFlow는 기능이 완성되어 배포할 준비가 되면 기본 브랜치에 병합되는 수명이 짧은 기능 브랜치라는 아이디어를 기반으로 합니다. GitHub Flow의 주요 원칙은 다음과 같습니다.

- 브랜칭은 간단합니다. 개발자는 클릭 몇 번으로 작업에 사용할 기능 브랜치를 만들 수 있으므로 기본 브랜치에 영향을 주지 않고 협업하고 실험할 수 있는 기능이 향상됩니다.
- 지속적 배포 — 변경 사항은 기본 브랜치에 병합되는 즉시 배포되므로 빠른 피드백과 반복이 가능합니다.
- 병합 요청 — 개발자는 변경 내용을 메인 브랜치에 병합하기 전에 병합 요청을 사용하여 토론 및 검토 프로세스를 시작합니다.

GitHub Flow에 대한 자세한 내용은 다음 리소스를 참조하십시오.

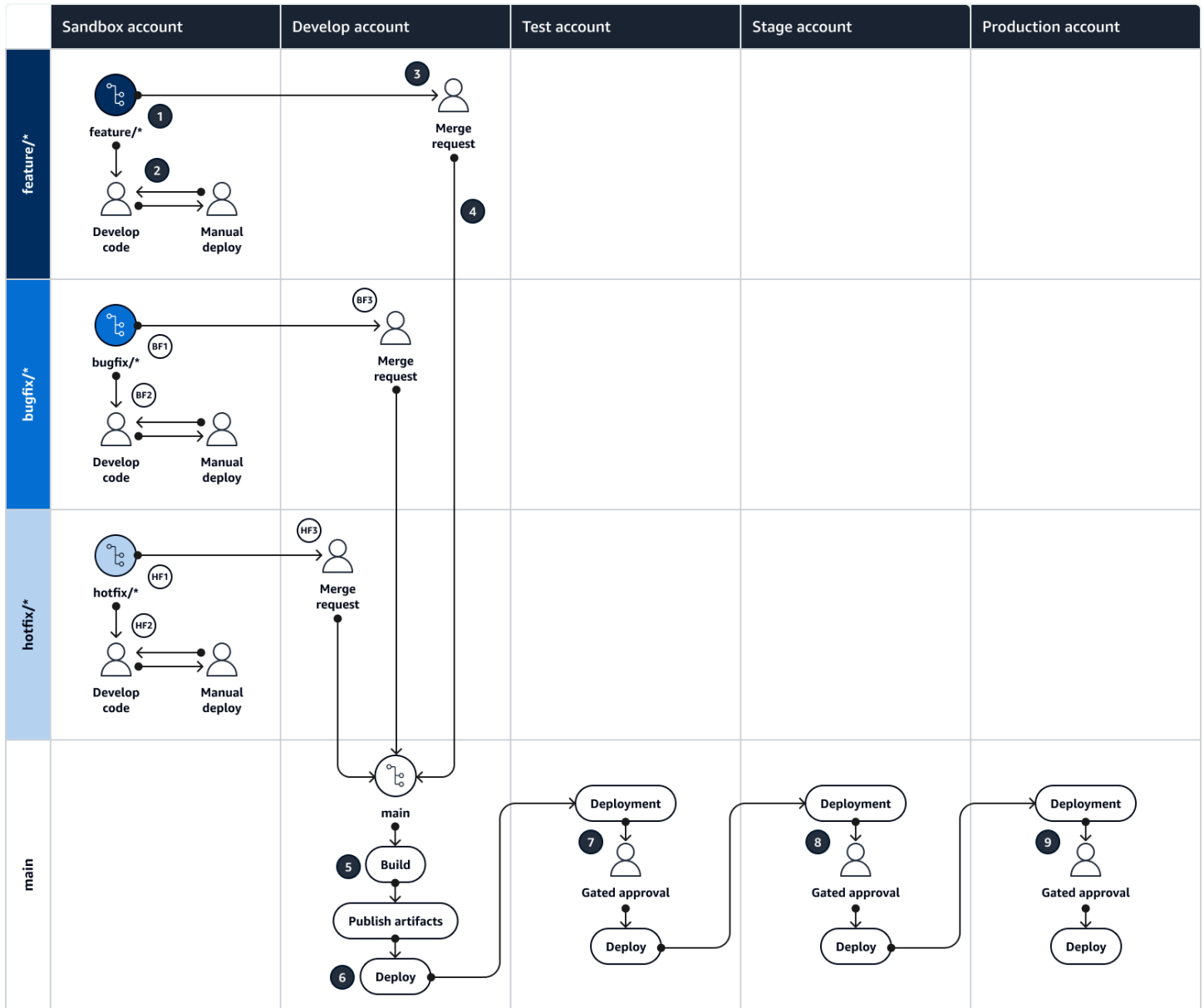
- [다중 계정 DevOps 환경을 위한 GitHub Flow 분기 전략 구현](#) (AWS 규범적 지침)
- [GitHub 플로우 퀵스타트](#) (설명서) GitHub
- [GitHub 플로우를 선택해야 하는 이유](#) (GitHub플로우 웹사이트)

이 단원의 주제:

- [GitHub 플로우 전략의 시각적 개요](#)
- [플로우 전략의 브랜치 GitHub](#)
- [GitHub 플로우 전략의 장점 및 단점](#)

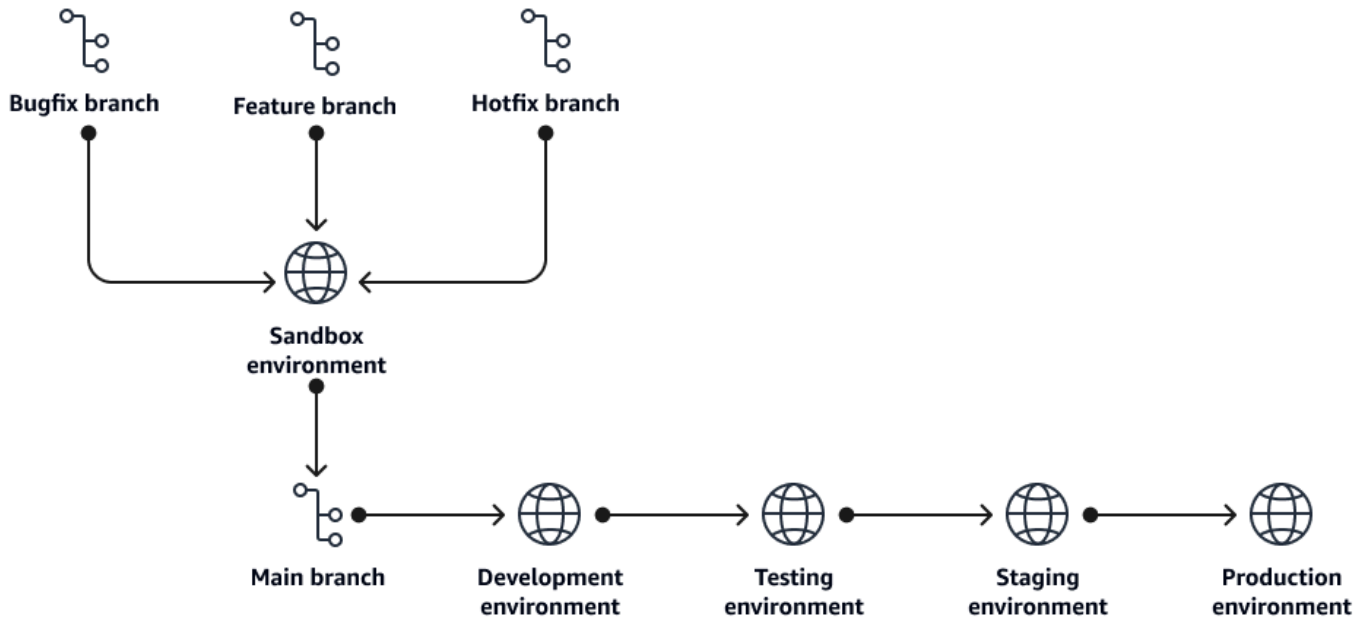
GitHub 플로우 전략의 시각적 개요

다음 다이어그램을 [퍼넷 스쿼어처럼](#) 사용하여 GitHub 플로우 브랜칭 전략을 이해할 수 있습니다. 세로 축의 분기를 가로 축의 AWS 환경과 정렬하여 각 시나리오에서 수행할 작업을 결정하십시오. 원으로 표시된 숫자는 다이어그램에 표시된 동작 순서를 안내합니다. 이 다이어그램은 샌드박스 환경의 기능 브랜치부터 메인 브랜치의 프로덕션 릴리스까지 GitHub Flow 브랜칭 전략의 개발 워크플로를 보여줍니다. 각 환경에서 발생하는 활동에 대한 자세한 내용은 이 가이드의 [DevOps 환경을](#) 참조하십시오.



플로우 전략의 브랜치 GitHub

GitHub 플로우 브랜칭 전략에는 일반적으로 다음과 같은 브랜치가 있습니다.



기능 브랜치

feature브랜치에서 기능을 개발합니다. feature브랜치를 만들려면 브랜치에서 브랜치를 분리해야 main 합니다. 개발자는 feature 브랜치에서 코드를 반복하고 커밋하고 테스트합니다. 기능이 완성되면 개발자는 에 병합 요청을 생성하여 기능을 홍보합니다main.

명명 규칙: `feature/<story number>_<developer initials>_<descriptor>`

명명 규칙 예제: `feature/123456_MS_Implement _Feature_A`

버그 수정 브랜치

bugfix브랜치는 문제를 해결하는 데 사용됩니다. 이러한 main 브랜치는 브랜치 외부에서 브랜치됩니다. 샌드박스 또는 하위 환경에서 버그 수정을 테스트한 후 병합 요청을 통해 이를 병합하여 상위 환경으로 승격할 수 있습니다. main 이는 구성 및 추적을 위해 권장되는 명명 규칙이며, 기능 분기를 사용하여 이 프로세스를 관리할 수도 있습니다.

명명 규칙: `bugfix/<ticket number>_<developer initials>_<descriptor>`

명명 규칙 예제: `bugfix/123456_MS_Fix_Problem_A`

핫픽스 브랜치

hotfix브랜치는 개발 직원과 프로덕션에 배포된 코드 사이의 지연을 최소화하면서 영향력이 큰 중요한 문제를 해결하는 데 사용됩니다. 이러한 main 브랜치는 브랜치 외부에서 분기됩니다. 샌드박스 또는 하위 환경에서 핫픽스를 테스트한 후 병합 요청을 main 통해 이를 병합하여 상위 환경으로 승격할 수 있습니다. 이는 구성 및 추적을 위해 권장되는 명명 규칙이며, 기능 분기를 사용하여 이 프로세스를 관리할 수도 있습니다.

명명 규칙: `hotfix/<ticket number>_<developer initials>_<descriptor>`

명명 규칙 예제: `hotfix/123456_MS_Fix_Problem_A`

기본 브랜치

main브랜치는 항상 프로덕션 환경에서 실행 중인 코드를 나타냅니다. 코드는 병합 요청을 사용하여 main 브랜치의 브랜치에 병합됩니다. feature 삭제를 방지하고 개발자가 코드를 직접 푸시하는 것을 방지하려면 브랜치에 대한 브랜치 보호를 활성화하세요. main main

명명 규칙: `main`

GitHub 플로우 전략의 장점 및 단점

GitHub Flow 브랜칭 전략은 강력한 커뮤니케이션 기술을 갖춘 작고 성숙한 개발 팀에 적합합니다. 이 전략은 지속적 전달을 구현하려는 팀에 매우 적합하며 일반적인 CI/CD 엔진에서도 잘 지원됩니다. GitHub Flow는 가볍고 규칙이 많지 않으며 빠르게 움직이는 팀을 지원할 수 있습니다. 팀이 준수해야 할 엄격한 규정 준수 또는 릴리스 프로세스가 있는 경우에는 적합하지 않습니다. 이 모델에서는 병합 충돌이 흔하며 자주 발생할 가능성이 높습니다. 병합 충돌 해결은 핵심 기술이므로 모든 팀원을 그에 맞게 교육해야 합니다.

장점

GitHub Flow는 개발 프로세스를 개선하고, 협업을 간소화하고, 소프트웨어의 전반적인 품질을 향상시킬 수 있는 몇 가지 이점을 제공합니다. 다음은 몇 가지 주요 이점입니다.

- 유연하고 가벼움 — GitHub Flow는 개발자가 소프트웨어 개발 프로젝트에서 협업하는 데 도움이 되는 가볍고 유연한 워크플로입니다. 이를 통해 복잡성을 최소화하면서 빠르게 반복하고 실험할 수 있습니다.
- 간소화된 협업 — GitHub Flow는 기능 개발을 관리하기 위한 명확하고 간소화된 프로세스를 제공합니다. 이를 통해 작고 집중적인 변경 사항을 장려하고 신속하게 검토 및 병합할 수 있어 효율성이 향상됩니다.
- 명확한 버전 관리 — GitHub Flow를 사용하면 모든 변경이 별도의 브랜치에서 이루어집니다. 이를 통해 명확하고 추적 가능한 버전 관리 기록이 설정됩니다. 이를 통해 개발자는 변경 사항을 추적 및 이해하고, 필요한 경우 되돌리고, 신뢰할 수 있는 코드 베이스를 유지할 수 있습니다.
- 원활한 지속적 통합 — GitHub Flow는 지속적 통합 도구와 통합됩니다. 풀 리퀘스트를 생성하면 자동화된 테스트 및 배포 프로세스를 시작할 수 있습니다. CI 도구를 사용하면 변경 사항이 main 브랜치에 병합되기 전에 철저하게 테스트하여 코드 베이스에 버그가 발생할 위험을 줄일 수 있습니다.
- 신속한 피드백과 지속적인 개선 — GitHub Flow는 풀 리퀘스트를 통해 잦은 코드 검토 및 토론을 촉진하여 피드백 루프가 빠르게 진행되도록 합니다. 이를 통해 문제를 조기에 발견하고 팀 구성원 간의 지식 공유가 촉진되며 궁극적으로 코드 품질이 향상되고 개발 팀 내 협업이 개선됩니다.
- 단순화된 롤백 및 되돌리기 — 코드 변경으로 인해 예상치 못한 버그나 문제가 발생하는 경우 GitHub Flow는 변경 사항을 롤백하거나 되돌리는 프로세스를 단순화합니다. 커밋 및 브랜치 기록이 명확하면 문제가 되는 변경 사항을 쉽게 식별하고 되돌릴 수 있어 안정적이고 기능적인 코드 베이스를 유지하는 데 도움이 됩니다.
- 간단한 학습 곡선 — GitHub Flow는 Gitflow보다 배우고 채택하기가 더 쉬울 수 있습니다. 특히 Git 및 버전 제어 개념에 이미 익숙한 팀의 경우 더욱 그렇습니다. 단순하고 직관적인 브랜칭 모델을 통해 다양한 경험 수준의 개발자가 액세스할 수 있어 새로운 개발 워크플로를 도입하는 데 필요한 학습 곡선을 줄일 수 있습니다.
- 지속적 개발 — GitHub Flow는 모든 변경 사항이 브랜치에 병합되는 즉시 이를 즉시 배포할 수 있도록 함으로써 팀이 지속적 배포 접근 방식을 채택할 수 있도록 합니다. main 이렇게 간소화된 프로세스를 통해 불필요한 지연이 제거되고 최신 업데이트 및 개선 사항을 사용자가 신속하게 사용할 수 있습니다. 그 결과 개발 주기가 더 민첩하고 대응력이 향상됩니다.

단점

GitHub Flow는 여러 가지 장점을 제공하지만 잠재적인 단점도 고려해야 합니다.

- 대규모 프로젝트에 대한 제한적 적합성 — GitHub Flow는 복잡한 코드 기반과 여러 장기 기능 분기가 있는 대규모 프로젝트에 적합하지 않을 수 있습니다. 이러한 경우 Gitflow와 같이 좀 더 구조화된 워크플로를 사용하면 동시 개발 및 릴리스 관리를 더 효과적으로 제어할 수 있습니다.

- 공식적인 릴리스 구조 부족 — GitHub Flow는 릴리스 프로세스를 명시적으로 정의하거나 버전 관리, 핫픽스 또는 유지 관리 브랜치와 같은 기능을 지원하지 않습니다. 이는 엄격한 릴리스 관리가 필요하거나 장기적인 지원 및 유지 관리가 필요한 프로젝트의 경우 제한이 될 수 있습니다.
- 장기 릴리스 계획에 대한 제한적 지원 — GitHub Flow는 엄격한 로드맵이나 광범위한 기능 종속성이 있는 프로젝트와 같이 장기 릴리스 계획이 필요한 프로젝트와 잘 맞지 않을 수 있는 단기 기능 브랜치에 중점을 둡니다. 복잡한 릴리스 일정을 관리하는 것은 Flow의 제약 내에서 어려울 수 있습니다.

GitHub

- 잦은 병합 충돌 가능성 — GitHub Flow는 잦은 분기 및 병합을 권장하므로 특히 개발 활동이 많은 프로젝트에서 병합 충돌이 발생할 가능성이 있습니다. 이러한 충돌을 해결하려면 시간이 많이 걸리고 개발팀의 추가 노력이 필요할 수 있습니다.
- 공식화된 워크플로 단계 부족 — GitHub Flow는 알파, 베타 또는 릴리스 후보 단계와 같은 개발 단계를 명시적으로 정의하지 않습니다. 이로 인해 프로젝트의 현재 상태나 다양한 브랜치 또는 릴리스의 안정성 수준을 전달하기가 더 어려워질 수 있습니다.
- 주요 변경 사항의 영향 — GitHub Flow는 변경 사항을 main 브랜치에 자주 병합하도록 권장하므로 코드 베이스의 안정성에 영향을 미치는 주요 변경 사항이 도입될 위험이 더 높습니다. 이러한 위험을 효과적으로 완화하려면 엄격한 코드 검토 및 테스트 관행이 중요합니다.

Gitflow 브랜칭 전략

Gitflow는 여러 브랜치를 사용하여 개발 단계에서 프로덕션 단계로 코드를 이동하는 브랜칭 모델입니다. Gitflow는 예정된 릴리스 주기가 있고 기능 컬렉션을 릴리스로 정의해야 하는 팀에 적합합니다. 개별 기능 브랜치에서 개발이 완료되며, 이 브랜치는 승인을 받아 통합에 사용되는 개발 브랜치로 병합됩니다. 이 브랜치의 기능은 프로덕션 준비가 완료된 것으로 간주됩니다. 계획된 모든 기능이 개발 브랜치에 누적되면 상위 환경에 배포할 수 있는 릴리스 브랜치가 생성됩니다. 이렇게 분리하면 정의된 일정 따라 어떤 변경 사항이 어떤 명명된 환경으로 이동할지 더 잘 제어할 수 있습니다. 필요한 경우 이 프로세스를 더 빠른 배포 모델로 가속화할 수 있습니다.

Gitflow 브랜칭 전략에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- 다중 계정 [환경을 위한 Gitflow 브랜칭 전략 구현](#) (규범적 지침) DevOps AWS
- [오리지널 Gitflow](#) 블로그 (빈센트 드리센 블로그 게시물)
- [깃플로우 워크플로 \(아틀라시안\)](#)

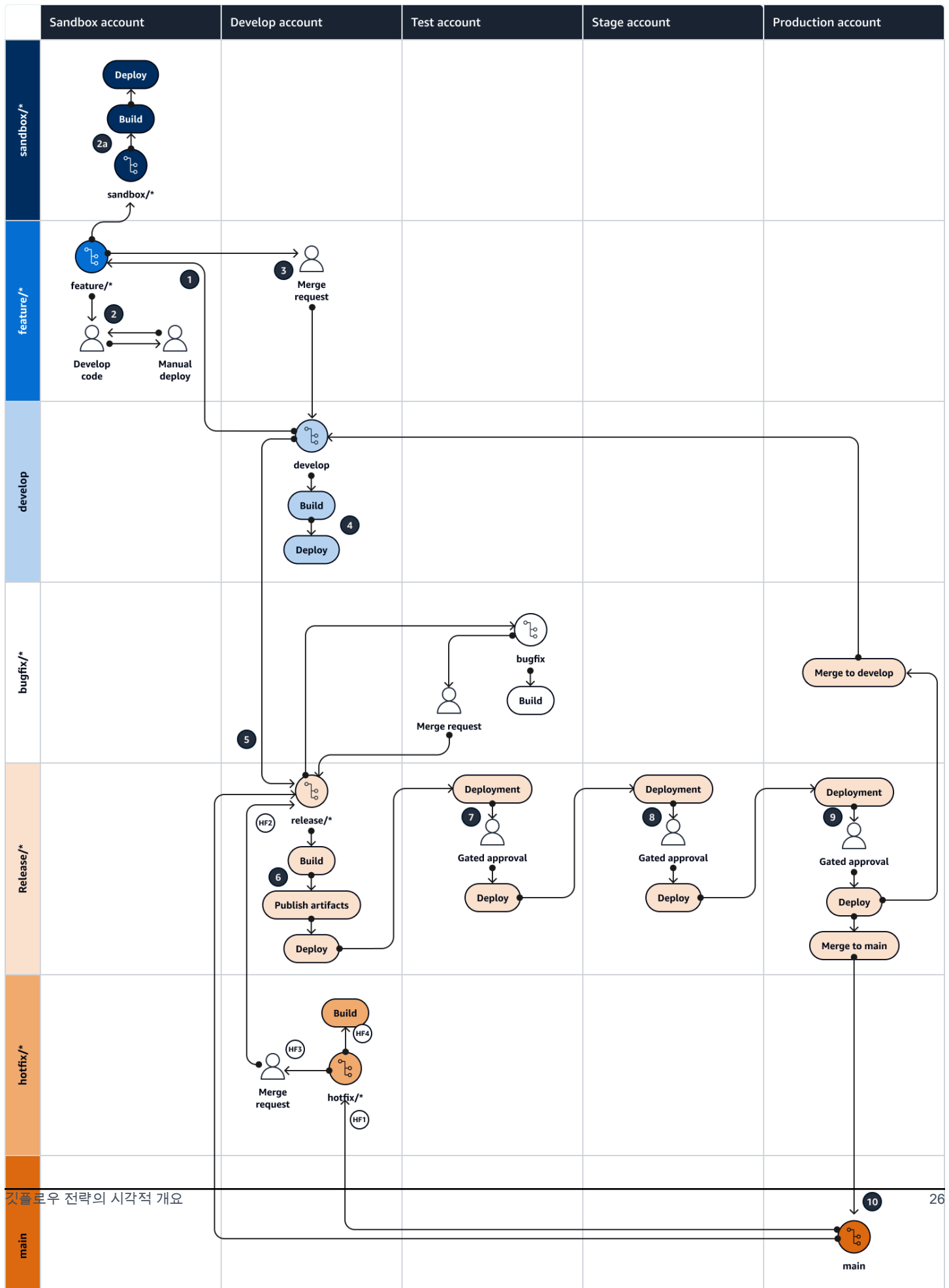
이 단원의 주제:

- [깃플로우 전략의 시각적 개요](#)

- [깃플로우 전략의 브랜치](#)
- [깃플로우 전략의 장점 및 단점](#)

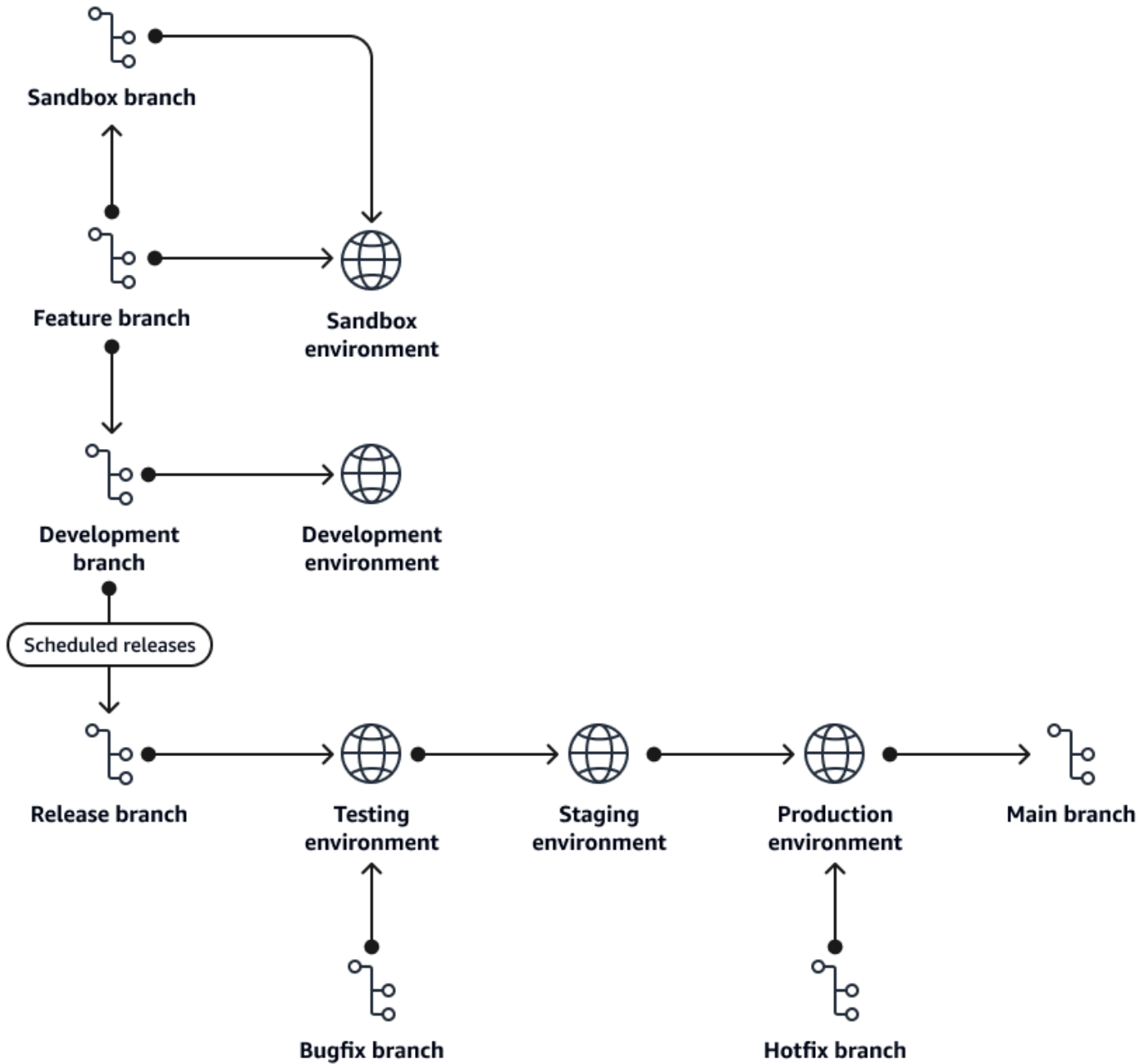
깃플로우 전략의 시각적 개요

다음 다이어그램을 [퍼넷 사각형처럼](#) 사용하여 Gitflow 분기 전략을 이해할 수 있습니다. 세로 축의 분기를 가로 축의 AWS 환경과 정렬하여 각 시나리오에서 수행할 작업을 결정하십시오. 원으로 표시된 숫자는 다이어그램에 표시된 동작 순서를 안내합니다. 각 환경에서 발생하는 활동에 대한 자세한 내용은 이 가이드의 [DevOps 환경을](#) 참조하십시오.



깃플로우 전략의 브랜치

Gitflow 브랜칭 전략에는 일반적으로 다음과 같은 브랜치가 있습니다.



기능 브랜치

Feature브랜치는 기능을 개발하는 단기 브랜치입니다. 브랜치는 브랜치를 분리하여 생성됩니다. feature develop 개발자는 브랜치에서 코드를 반복하고, 커밋하고, 테스트합니다feature. 기능이

완성되면 개발자가 기능을 홍보합니다. 기능 브랜치에서 앞으로 나아갈 수 있는 경로는 두 가지뿐입니다.

- sandbox브랜치에 병합
- develop브랜치에 병합 요청을 생성하세요.

명명 규칙: `feature/<story number>_<developer initials>_<descriptor>`

명명 규칙 예제: `feature/123456_MS_Implement
_Feature_A`

샌드박스 브랜치

sandbox브랜치는 Gitflow의 비표준 단기 브랜치입니다. 하지만 CI/CD 파이프라인 개발에는 유용합니다. sandbox브랜치는 주로 다음과 같은 용도로 사용됩니다.

- 수동 배포 대신 CI/CD 파이프라인을 사용하여 샌드박스 환경에 전체 배포를 수행하십시오.
- 개발 또는 테스트와 같은 하위 환경에서의 전체 테스트를 위해 병합 요청을 제출하기 전에 파이프라인을 개발하고 테스트하세요.

Sandbox브랜치는 일시적이며 오래 사용할 수 있는 것이 아닙니다. 특정 테스트가 완료된 후에는 삭제해야 합니다.

명명 규칙: `sandbox/<story number>_<developer initials>_<descriptor>`

명명 규칙 예제: `sandbox/123456_MS_Test_Pipeline_Deploy`

브랜치 개발

develop브랜치는 기능을 통합, 구축, 검증하고 개발 환경에 배포하는 수명이 긴 브랜치입니다. 모든 feature 브랜치가 브랜치에 병합됩니다. develop develop브랜치로의 병합은 성공적인 빌드와 두

번의 개발자 승인이 필요한 병합 요청을 통해 완료됩니다. 삭제를 방지하려면 브랜치에서 브랜치 보호를 활성화하세요. `develop`

명명 규칙: `develop`

릴리즈 브랜치

Gitflow에서 브랜치는 단기 `release` 브랜치입니다. 이러한 브랜치는 한 번 빌드하고 여러 번 배포하는 방법론을 채택하여 여러 환경에 배포할 수 있다는 점에서 특별합니다. Release브랜치는 테스트, 스테이징 또는 프로덕션 환경을 대상으로 할 수 있습니다. 개발 팀은 상위 환경으로 기능을 승격하기로 결정한 후 새 `release` 브랜치를 만들고 이전 릴리스보다 버전 번호를 늘려 사용합니다. 각 환경의 게이트에서 배포를 진행하려면 수동 승인이 필요합니다. Release지점의 경우 병합 요청을 변경해야 합니다.

`release`브랜치를 프로덕션에 배포한 후에는 `develop` 및 `main` 브랜치에 다시 병합하여 버그 수정이나 핫픽스가 향후 개발 작업에 다시 병합되도록 해야 합니다.

명명 규칙: `release/v{major}.{minor}`

명명 규칙 예제: `release/v1.0`

기본 브랜치

`main`브랜치는 운영 환경에서 실행 중인 코드를 항상 나타내는 수명이 긴 브랜치입니다. 릴리스 파이프라인에서 성공적으로 배포되면 릴리스 브랜치에서 코드가 자동으로 브랜치에 병합됩니다. `main` 삭제를 방지하려면 브랜치에서 브랜치 보호를 활성화하세요. `main`

명명 규칙: `main`

버그 수정 브랜치

`bugfix`브랜치는 프로덕션으로 릴리스되지 않은 릴리스 브랜치의 문제를 해결하는 데 사용되는 단기 브랜치입니다. `bugfix`브랜치는 브랜치의 수정 사항을 테스트, 스테이징 또는 프로덕션 환경으로 홍보하는 용도로만 사용해야 합니다. `release` `bugfix`브랜치는 항상 브랜치에서 분리됩니다. `release`

버그 수정을 테스트한 후 병합 요청을 통해 release 브랜치로 승격할 수 있습니다. 그러면 표준 릴리스 프로세스에 따라 release 브랜치를 추진할 수 있습니다.

명명 규칙: `bugfix/<ticket>_<developer initials>_<descriptor>`

명명 규칙 예제: `bugfix/123456_MS_Fix_Problem_A`

핫픽스 브랜치

hotfix 브랜치는 프로덕션 문제를 해결하는 데 사용되는 단기 브랜치입니다. 프로덕션 환경에 적용하기 위해 신속히 처리해야 하는 수정 사항을 홍보하는 용도로만 사용됩니다. hotfix 브랜치는 항상 브랜치됩니다. main

핫픽스를 테스트한 후에는 생성된 release 브랜치에 대한 병합 요청을 통해 핫픽스를 프로덕션 환경으로 승격시킬 수 있습니다. main 테스트를 위해 표준 릴리스 프로세스에 따라 release 브랜치를 푸시할 수 있습니다.

명명 규칙: `hotfix/<ticket>_<developer initials>_<descriptor>`

명명 규칙 예제: `hotfix/123456_MS_Fix_Problem_A`

깃플로우 전략의 장점 및 단점

Gitflow 브랜칭 전략은 릴리스 및 규정 준수 요구 사항이 엄격한 대규모 분산 팀에 적합합니다. Gitflow는 조직의 예측 가능한 릴리스 주기에 기여하며, 대규모 조직에서는 Gitflow를 선호하는 경우가 많습니다. Gitflow는 소프트웨어 개발 라이프사이클을 제대로 완료하기 위한 가드레일이 필요한 팀에도 적합합니다. 이는 전략에 내장된 검토 및 품질 보증을 위한 여러 기회가 있기 때문입니다. Gitflow는 여러 버전의 프로덕션 릴리스를 동시에 유지 관리해야 하는 팀에도 적합합니다. GitFlow의 몇 가지 단점은 다른 분기 모델보다 더 복잡하고 패턴을 엄격하게 준수해야 성공적으로 완료된다는 점입니다. 릴리스 브랜치를 관리하는 엄격한 특성 때문에 지속적 배포를 추구하는 조직에서는 Gitflow가 제대로 작동하지 않습니다. Gitflow 릴리스 브랜치는 적절한 시기에 적절하게 처리되지 않으면 기술적인 부채가 누적될 수 있는 수명이 긴 브랜치일 수 있습니다.

장점

GitFlow 기반 개발은 개발 프로세스를 개선하고, 협업을 간소화하고, 소프트웨어의 전반적인 품질을 향상시킬 수 있는 몇 가지 이점을 제공합니다. 다음은 몇 가지 주요 이점입니다.

- 예측 가능한 릴리스 프로세스 — Gitflow는 정기적이고 예측 가능한 릴리스 프로세스를 따릅니다. 개발 및 릴리스 주기가 정기적인 팀에 매우 적합합니다.
- 협업 개선 — Gitflow는 및 브랜치의 사용을 권장합니다. feature release 이 두 브랜치는 팀이 서로에 대한 종속성을 최소화하면서 동시에 작업할 수 있도록 도와줍니다.
- 여러 환경에 매우 적합 — Gitflow는 수명이 더 긴 release 브랜치일 수 있는 브랜치를 사용합니다. 이러한 브랜치를 통해 팀은 개별 릴리스를 더 오랜 기간 동안 타겟팅할 수 있습니다.
- 프로덕션 중인 여러 버전 — 팀이 프로덕션 환경에서 여러 버전의 소프트웨어를 지원하는 경우 Gitflow release 브랜치는 이 요구 사항을 지원합니다.
- 내장 코드 품질 검토 — Gitflow는 코드를 다른 환경으로 승격하기 전에 코드 검토 및 승인을 요구하고 사용을 권장합니다. 이 프로세스는 모든 코드 프로모션에 이 단계를 요구하므로 개발자 간의 마찰을 없애줍니다.
- 조직의 이점 — Gitflow는 조직 수준에서도 이점이 있습니다. Gitflow는 표준 릴리스 주기를 사용하도록 권장하며, 이는 조직이 릴리스 일정을 이해하고 예측하는 데 도움이 됩니다. 이제 비즈니스에서 새 기능을 제공할 수 있는 시기를 이해하므로 출시 날짜가 정해져 있기 때문에 일정에 대한 마찰이 줄어듭니다.

단점

GitFlow 기반 개발에는 개발 프로세스와 팀 역학에 영향을 미칠 수 있는 몇 가지 단점이 있습니다. 몇 가지 주목할 만한 단점은 다음과 같습니다.

- 복잡성 — Gitflow는 새로운 팀이 학습하기에는 복잡한 패턴이므로 이를 성공적으로 사용하려면 Gitflow의 규칙을 준수해야 합니다.
- 지속적 배포 — Gitflow는 많은 배포가 빠른 속도로 프로덕션 환경에 릴리스되는 모델에는 적합하지 않습니다. Gitflow를 사용하려면 여러 브랜치를 사용해야 하고 브랜치를 관리하는 엄격한 워크플로가 필요하기 때문입니다. release
- 브랜치 관리 — Gitflow는 많은 브랜치를 사용하므로 유지 관리가 부담스러울 수 있습니다. 브랜치를 서로 적절하게 정렬하기 위해 다양한 브랜치를 추적하고 릴리스된 코드를 병합하는 것은 어려울 수 있습니다.
- 기술적 부채 — Gitflow 릴리스는 일반적으로 다른 분기 모델보다 느리기 때문에 릴리스 시 더 많은 기능이 누적될 수 있으며 이로 인해 기술적 부채가 누적될 수 있습니다.

팀은 GitFlow 기반 개발이 프로젝트에 적합한 접근 방식인지 결정할 때 이러한 단점을 신중하게 고려해야 합니다.

다음 단계

이 가이드에서는 세 가지 일반적인 Git 브랜칭 전략인 플로우, GitHub 깃플로우, 트렁크 간의 차이점을 설명합니다. 워크플로를 자세히 설명하고 각 워크플로의 장단점도 제공합니다. 다음 단계는 조직에 맞는 이러한 표준 워크플로 중 하나를 선택하는 것입니다. 이러한 분기 전략 중 하나를 구현하려면 다음을 참조하십시오.

- [다중 계정 환경을 위한 트렁크 브랜칭 전략 구현 DevOps](#)
- [다중 계정 환경을 위한 GitHub Flow 분기 전략 구현 DevOps](#)
- [다중 계정 환경을 위한 Gitflow 분기 전략 구현 DevOps](#)

DevOps Git과 프로세스를 사용하기 위한 팀의 여정을 어디서부터 시작해야 할지 잘 모르겠다면 표준 솔루션을 선택하여 테스트해 보는 것이 좋습니다. 표준 분기 규칙을 사용하면 팀이 기존 문서를 기반으로 작성하고 자신에게 가장 적합한 것이 무엇인지 알 수 있습니다.

조직 또는 개발 팀에 맞지 않더라도 전략을 변경하는 것을 두려워하지 마세요. 개발 팀의 요구 사항과 요구 사항은 시간이 지남에 따라 변할 수 있으며, 완벽한 단일 솔루션은 없습니다.

리소스

이 가이드에는 Git에 대한 교육이 포함되어 있지 않지만, 이 교육이 필요한 경우 인터넷에서 많은 고품질 리소스를 이용할 수 있습니다. [Git 설명서](#) 사이트에서 시작하는 것이 좋습니다.

다음 리소스는 에서의 Git 브랜칭 여정에 도움이 될 수 있습니다. AWS 클라우드

AWS 규범적 지침

- [다중 계정 환경을 위한 트렁크 브랜칭 전략 구현 DevOps](#)
- [다중 계정 환경을 위한 GitHub Flow 분기 전략 구현 DevOps](#)
- [다중 계정 환경을 위한 Gitflow 분기 전략 구현 DevOps](#)

기타 지침 AWS

- [AWS DevOps 지침](#)
- [AWS 배포 파이프라인 참조 아키텍처](#)
- [DevOps란 무엇입니까?](#)
- [DevOps리소스](#)

기타 리소스

- [트웰-팩터 앱 방법론 \(12factor.net\)](#)
- [기프트 시크릿 \(\)](#) GitHub
- [깃플로우](#)
 - [오리지널 Gitflow 블로그 \(빈센트 드리센 블로그 게시물\)](#)
 - [깃플로우 워크플로 \(아틀라시안\)](#)
 - [Gitflow on GitHub: GitHub 기반 리포지토리에서 Git Flow 워크플로를 사용하는 방법 \(비디오\)](#) YouTube
 - [Git 플로우 초기화 예제 \(비디오\)](#) YouTube
 - [Gitflow 릴리즈 브랜치를 처음부터 끝까지 \(비디오\)](#) YouTube
- [GitHub 플로우](#)

- [GitHub 플로우 퀵스타트 \(GitHub 설명서\)](#)
- [GitHub 플로우를 선택해야 하는 이유 \(GitHub 플로우 웹사이트\)](#)
- 트렁크
 - [트렁크 기반 개발 소개 \(트렁크 기반 개발 웹 사이트\)](#)

기여자

작성

- 마이크 스티븐스, 선임 클라우드 애플리케이션 아키텍트, AWS
- 레이잔 월슨, 선임 클라우드 애플리케이션 아키텍트, AWS
- 압힐라쉬 비노드, 팀장, 선임 클라우드 애플리케이션 아키텍트 AWS

리뷰

- 스티븐 DiCato, 선임 보안 컨설턴트, AWS
- 가우라브 사무드라, 클라우드 애플리케이션 아키텍트, AWS
- 스티븐 구겐하이머, 팀장, 선임 클라우드 애플리케이션 아키텍트, AWS

테크니컬 라이팅

- 릴리 AbouHarb, 선임 테크니컬 라이터, AWS

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 [RSS 피드](#)를 구독하십시오.

변경 사항	설명	날짜
최초 게시	—	2024년 2월 15일

AWS 규범적 지침 용어집

다음은 AWS 규범적 지침에서 제공하는 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. 예: 온프레미스 Oracle 데이터베이스를 Amazon Aurora PostgreSQL-Compatible Edition으로 마이그레이션합니다.
- 리플랫폼(리프트 앤드 리세이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예: 에서 온프레미스 Oracle 데이터베이스를 Oracle용 Amazon Relational Database Service(AmazonRDS)로 마이그레이션합니다 AWS 클라우드.
- 재구매(드롭 앤드 쇼프) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예: 고객 관계 관리(CRM) 시스템을 Salesforce.com 마이그레이션합니다.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예: 의 EC2 인스턴스에서 온프레미스 Oracle 데이터베이스를 Oracle로 마이그레이션합니다 AWS 클라우드.
- 재배포(하이퍼바이저 수준의 리프트 앤 시프트) - 새 하드웨어를 구매하거나, 애플리케이션을 다시 작성하거나, 기존 운영을 수정하지 않고도 인프라를 클라우드로 이동합니다. 온프레미스 플랫폼에서 동일한 플랫폼의 클라우드 서비스로 서버를 마이그레이션합니다. 예: 마이그레이션 Microsoft Hyper-V 에 대한 애플리케이션입니다 AWS.
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중에 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 제어](#) 를 참조하세요.

추상화된 서비스

[관리형 서비스](#) 를 참조하세요.

ACID

[원자성, 일관성, 격리, 내구성](#) 을 참조하세요.

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. 유연성은 뛰어나지만 [능동 수동 마이그레이션](#)보다 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 작동하고 그룹의 단일 반환 값을 계산하는 SQL 함수입니다. 집계 함수의 예로는 SUM 및 MAX가 있습니다.

AI

[인공 지능](#) 을 참조하세요.

AIOps

[인공 지능 작업](#) 을 참조하세요.

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

맬웨어로부터 시스템을 보호하기 위해 승인된 애플리케이션만 사용할 수 있도록 허용하는 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 프로세스](#)의 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 작업(AIOps)

기계 학습 기법을 사용하여 운영 문제를 해결하고, 운영 인시던트 및 사용자 개입을 줄이고, 서비스 품질을 높이는 프로세스입니다. AIOps 가 마이그레이션 전략에 사용되는 AWS 방법에 대한 자세한 내용은 [운영 통합 가이드 섹션](#)을 참조하세요.

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, 격리, 내구성(ACID)

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

속성 기반 액세스 제어(ABAC)

부서, 직무, 팀 이름 등의 사용자 속성을 기반으로 세분화된 권한을 생성하는 방식입니다. 자세한 내용은 AWS Identity and Access Management (IAM) 설명서 [ABAC AWS](#)의 섹션을 참조하세요.

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 절연 AWS 리전 되고 동일한 리전의 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 내 고유 위치입니다.

AWS 클라우드 채택 프레임워크(AWS CAF)

조직이 클라우드로 성공적으로 전환하기 위한 효율적이고 효과적인 계획을 개발하는 AWS 데 도움이 되는 의 지침 및 모범 사례 프레임워크입니다. AWS CAF 는 지침을 비즈니스, 사람, 거버넌스, 플랫폼, 보안 및 운영이라는 6가지 중점 영역으로 구성합니다. 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. 이 관점에서 AWS CAF는 성공적인 클라우드 채택을 위해 조직을 준비하는 데 도움이 되는 인력 개발, 훈련 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 [AWS CAF 웹 사이트](#) 및 [AWS CAF 백서](#)를 참조하세요.

AWS 워크로드 검증 프레임워크(AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고, 마이그레이션 전략을 권장하고, 작업 추정치를 제공하는 도구입니다. AWS WQF 는 AWS Schema Conversion Tool (AWS SCT)에 포함되어 있습니다. 데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

B

잘못된 붓

개인 또는 조직을 방해하거나 해를 입히기 위한 [붓](#)입니다.

BCP

[비즈니스 연속성 계획](#) 을 참조하세요.

동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그인 시도, 의심스러운 API 통화 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 [Data in a behavior graph](#)를 참조하십시오.

빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. [Endianness](#)도 참조하세요.

바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, ‘이 제품은 책임가요, 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

별개의 동일한 두 환경을 생성하는 배포 전략입니다. 현재 애플리케이션 버전은 한 환경(파란색)에서 실행하고 새 애플리케이션 버전은 다른 환경(녹색)에서 실행합니다. 이 전략을 사용하면 영향을 최소화하면서 빠르게 롤백할 수 있습니다.

bot

인터넷을 통해 자동화된 작업을 실행하고 인적 활동 또는 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같은 일부 봇은 유용하거나 유용합니다. 잘못된 봇이라고 하는 다른 봇은 개인 또는 조직에 방해가 되거나 피해를 입히기 위한 것입니다.

봇넷

[맬웨어](#)에 감염되고 [봇](#) 세더 또는 봇 운영자라고 하는 단일 당사자의 제어 하에 있는 봇 네트워크입니다. Botnet은 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](#)(GitHub 문서)를 참조하세요.

브레이크 글라스 액세스

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스할 권한이 없는 에 액세스할 수 있는 빠른 방법입니다. 자세한 내용은 Well-Architected 지침의 [브레이크 글라스 절차 구현](#) 표시기를 AWS 참조하세요.

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

[AWS Cloud Adoption Framework](#) 를 참조하세요.

canary 배포

최종 사용자에게 버전의 느린 증분 릴리스입니다. 확신이 드는 경우 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

CCoE

[Cloud Center of Excellence](#)를 참조하세요.

CDC

[데이터 캡처 변경](#) 을 참조하세요.

데이터 캡처 변경(CDC)

데이터베이스 테이블과 같은 데이터 소스의 변경 내용을 추적하고 변경 사항에 대한 메타데이터를 기록하는 프로세스입니다. 대상 시스템의 변경 사항을 감사하거나 복제하는 등 동기화를 유지하기 위해 CDC 위한 다양한 용도로 사용할 수 있습니다.

혼돈 엔지니어링

시스템 복원력을 테스트하기 위해 의도적으로 장애 또는 중단 이벤트를 도입합니다. [AWS Fault Injection Service \(AWS FIS\)](#)를 사용하여 AWS 워크로드에 스트레스를 가하고 응답을 평가하는 실험을 수행할 수 있습니다.

CI/CD

[지속적 통합 및 지속적 전송](#)을 참조하세요.

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

대상에서 데이터를 AWS 서비스 수신하기 전에 로컬에서 데이터를 암호화합니다.

Cloud Center of Excellence(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 게시물](#)을 참조하세요.

클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 [엣지 컴퓨팅](#) 기술에 연결됩니다.

클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 [클라우드 운영 모델 구축](#)을 참조하십시오.

클라우드 채택 단계

조직이 로 마이그레이션할 때 일반적으로 거치는 4단계: AWS 클라우드

- 프로젝트 - 개념 증명 및 학습 목적으로 몇 가지 클라우드 관련 프로젝트 실행
- 파운데이션 - 클라우드 채택을 확장하기 위한 기본 투자(예: 랜딩 영역 생성, 정의CCoE, 운영 모델 설정)
- 마이그레이션 - 개별 애플리케이션 마이그레이션
- Re-invention - 제품 및 서비스 최적화와 클라우드 혁신

이러한 단계는 Stephen Orban이 블로그 게시물 [The Journey Toward Cloud-First 및 Enterprise Strategy 블로그의 채택 단계에](#) 정의했습니다. AWS 클라우드 AWS 마이그레이션 전략과 관련된 방법에 대한 자세한 내용은 [마이그레이션 준비 가이드 섹션](#)을 참조하세요.

CMDB

[구성 관리 데이터베이스](#) 를 참조하세요.

코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반적인 클라우드 리포지토리에는 GitHub 또는 포함됩니다 AWS CodeCommit. 코드의 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](#) 필드입니다. 예를 들어 온프레미스 카메라 네트워크에 CV를 추가하는 디바이스를 AWS Panorama 제공하고 Amazon은 CV에 대한 이미지 처리 알고리즘을 SageMaker 제공합니다.

구성 드리프트

워크로드의 경우 구성이 예상 상태에서 변경됩니다. 이로 인해 워크로드가 규정 미준수가 될 수 있으며 일반적으로 점진적이고 의도하지 않습니다.

구성 관리 데이터베이스(CMDB)

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

규정 준수 팩

규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 AWS Config 규칙 및 수정 작업 모음입니다. YAML 템플릿을 사용하여 AWS 계정 및 리전 또는 조직 전체에서 적합성 팩을 단일 엔터티로 배포할 수 있습니다. 자세한 내용은 AWS Config 설명서의 [적합성 팩](#)을 참조하세요.

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 이점](#)을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 [지속적 전달\(Continuous Delivery\)](#)과 [지속적인 개발](#)을 참조하십시오.

CV

[컴퓨터 비전](#)을 참조하세요.

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 기둥 구성 요소입니다. 자세한 내용은 [데이터 분류](#)를 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 거버넌스를 통해 분산되고 분산된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터를 최소화하면 프라이버시 위험, 비용 및 분석 탄소 발자국을 줄일 AWS 클라우드 수 있습니다.

데이터 경계

신뢰할 수 있는 자격 증명만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스할 수 있도록 하는 AWS 환경의 예방 가드레일 세트입니다. 자세한 내용은 [에서 데이터 경계 구축을 AWS](#) 참조하세요.

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 시스템입니다. 데이터 웨어하우스에는 일반적으로 많은 양의 기록 데이터가 포함되어 있으며 일반적으로 쿼리 및 분석에 사용됩니다.

데이터베이스 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 언어](#) 를 참조하세요.

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. 에서 이 전략을 채택 AWS하면 AWS Organizations 구조의 여러 계층에 여러 제어를 추가하여 리소스를 보호하는 데 도움이 됩니다. 예를 들어, 접근 defense-in-depth 방식은 다중 인증, 네트워크 세분화 및 암호화를 결합할 수 있습니다.

위임된 관리자

에서 AWS Organizations호환되는 서비스는 AWS 멤버 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 Organizations 와 함께 사용할 수 있는 AWS 서비스](#)를 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

[환경](#) 을 참조하세요.

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 controls](#)를 참조하십시오.

개발 값 스트림 매핑(DVSM)

소프트웨어 개발 라이프사이클에서 속도와 품질에 부정적인 영향을 미치는 제약 조건을 식별하고 우선 순위를 지정하는 데 사용되는 프로세스입니다. DVSM 는 원래 린 제조 관행을 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.

차원 테이블

[스타 스키마](#)에서 팩트 테이블의 정량적 데이터에 대한 데이터 속성을 포함하는 더 작은 테이블입니다. 차원 테이블 속성은 일반적으로 텍스트 필드 또는 텍스트처럼 동작하는 개별 숫자입니다. 이러한 속성은 쿼리 제약, 필터링 및 결과 세트 레이블 지정에 일반적으로 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 가동 중지 시간 및 데이터 손실을 최소화하기 위해 사용하는 전략 및 프로세스](#)입니다. 자세한 내용은 [AWS Well-Architected Framework의 클라우드에서 AWS: 복구에서 워크로드의 재해 복구](#)를 참조하세요.

DML

[데이터베이스 조작 언어](#)를 참조하세요.

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. 스트랭글러 무화과 패턴으로 도메인 기반 설계를 사용하는 방법에 대한 자세한 내용은 [컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET \(ASMX\) 웹 서비스 점진적으로 현대화를](#) 참조하세요.

DR

[재해 복구](#)를 참조하세요.

드리프트 감지

기존 구성과의 편차 추적. 예를 들어 AWS CloudFormation 를 사용하여 [시스템 리소스의 드리프트를 감지하거나](#) 를 사용하여 AWS Control Tower 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 [런타임의 변경 사항을 감지](#)할 수 있습니다.

DVSM

[개발 값 스트림 매핑](#)을 참조하세요.

E

EDA

[탐색적 데이터 분석](#)을 참조하세요.

엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 엣지 컴퓨팅은 [클라우드 컴퓨팅과](#) 비교할 때 통신 지연 시간을 줄이고 응답 시간을 개선할 수 있습니다.

암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 암호 텍스트로 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 엔드포인트](#)를 참조하세요.

엔드포인트 서비스

가상 프라이빗 클라우드(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 를 사용하여 엔드포인트 서비스를 생성하고 다른 보안 주체 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 AWS PrivateLink 부여할 수 있습니다. 이러한 계정 또는 보안 주체는 인터페이스 엔드포인트를 생성하여 VPC 엔드포인트 서비스에 비공개로 연결할 수 있습

니다. 자세한 내용은 Amazon Virtual Private Cloud(AmazonVPC) 설명서의 [엔드포인트 서비스 생성을](#) 참조하세요.

엔터프라이즈 리소스 계획(ERP)

엔터프라이즈의 주요 비즈니스 프로세스(예: 회계, 및 프로젝트 관리)를 자동화 [MES](#) 하고 관리하는 시스템입니다.

봉투 암호화

암호화 키를 다른 암호화 키로 암호화하는 프로세스입니다. 자세한 내용은 AWS Key Management Service (AWS KMS) 설명서의 [봉투 암호화](#)를 참조하세요.

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 보안 AWS CAF 에픽에는 자격 증명 및 액세스 관리, 탐지 제어, 인프라 보안, 데이터 보호 및 인시던트 대응이 포함됩니다. AWS 마이그레이션 전략의 에픽에 대한 자세한 내용은 [프로그램 구현 가이드](#)를 참조하십시오.

ERP

[엔터프라이즈 리소스 계획을](#) 참조하세요.

탐색적 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

[별표 스키마](#)의 중앙 테이블입니다. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블에는 측정값이 포함된 열과 차원 테이블에 대한 외부 키가 포함된 열의 두 가지 유형이 있습니다.

빠른 실패

자주 증분 테스트를 사용하여 개발 수명 주기를 줄이는 철학입니다. 애자일 접근 방식의 중요한 부분입니다.

장애 격리 경계

에서 장애의 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역, AWS 리전 제어 영역 또는 데이터 영역과 같은 AWS 클라우드경계입니다. 자세한 내용은 [AWS 장애 격리 경계를 참조하세요](#).

기능 브랜치

[브랜치를 참조하세요](#).

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 Shapley Additive Explanations(SHAP) 및 통합 그래데이션과 같은 다양한 기법을 통해 계산할 수 있는 수치 점수로 표현됩니다. 자세한 내용은 [를 사용한 기계 학습 모델 해석 가능성을 참조하세요AWS](#).

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 제어](#)를 참조하세요.

세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

플래시컷 마이그레이션

단계적 접근 방식을 사용하는 대신 [변경 데이터 캡처](#)를 통해 지속적인 데이터 복제를 사용하여 가능한 최단 시간 내에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

G

지리적 차단

[지리적 제한 사항](#)을 참조하세요.

지리적 제한(지리적 차단)

Amazon에서는 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 CloudFront 옵션입니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 CloudFront 설명서의 [콘텐츠의 지리적 배포 제한을 참조하세요](#).

Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 [트렁크 기반 워크플로](#)는 현대적이고 선호하는 접근 방식입니다.

브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 [브라운필드](#)라고도 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 전반의 리소스, 정책 및 규정 준수를 관리하는 데 도움이 되는 상위 수준 규칙입니다 (OUs). 예방 가드레일은 규정 준수 표준에 부합하도록 정책을 시행하며, 서비스 제어 정책 및 IAM 권한 경계를 사용하여 구현됩니다. 탐지 가드레일은 정책 위반 및 규정 준수 문제를 감지하고 해결을 위한 알림을 생성하며, AWS Config, Amazon AWS Security Hub, GuardDuty, AWS Trusted Advisor Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

H

HA

[고가용성 섹션](#)을 참조하세요.

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 SCT](#)를 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

소스 데이터베이스를 동일한 데이터베이스 엔진을 공유하는 대상 데이터베이스로 마이그레이션(예: Microsoft SQL Server에서 Amazon RDS for SQL Server로). 동종 마이그레이션은 일반적으로 리호스팅 또는 리플랫폼 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. 긴급성으로 인해 핫픽스는 일반적으로 일반적인 DevOps 릴리스 워크플로 외부에서 이루어집니다.

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 코드 로 참조하세요.](#)

자격 증명 기반 정책

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

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 인터넷](#) 을 참조하세요.

변경할 수 없는 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드를 위해 새 인프라를 배포하는 모델입니다. 변경 가능한 인프라는 본질적으로 [변경 가능한 인프라](#)보다 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 AWS Well-Architected Framework의 [변경할 수 없는 인프라를 사용한 배포](#) 모범 사례를 참조하세요.

인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 외부에서 네트워크 연결을 수락, 검사 및 라우팅VPC 하는 . [AWS Security Reference Architecture](#)는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 VPCs 위해 인바운드, 아웃바운드 및 검사로 네트워크 계정을 설정하는 것이 좋습니다.

중분 마이그레이션

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

연결성, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통한 제조 프로세스의 현대화를 언급하기 위해 2016년에 [Klaus Schwab](#)에서 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 구축\(IIoT\) 디지털 변환 전략 단원을 참조하세요.](#)

검사 VPC

AWS 다중 계정 아키텍처에서 (VPCs 동일하거나 다른 AWS 리전), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 VPC 관리하는 중앙 집중식 아키텍처입니다. [AWS Security Reference Architecture](#)는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사로 네트워크 계정을 설정하는 것이 좋습니다.

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 [를 사용한 기계 학습 모델 해석 가능성을 AWS](#)참조하세요.

IoT

[사물 인터넷을 참조하세요.](#)

IT 정보 라이브러리(ITIL)

IT 서비스를 제공하고 이러한 서비스를 비즈니스 요구 사항에 맞게 조정하기 위한 일련의 모범 사례 ITIL 는 의 기반을 제공합니다ITSM.

IT 서비스 관리(ITSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 작업을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 [작업 통합 가이드 단원](#)을 참조하세요.

ITIL

[IT 정보 라이브러리](#)를 참조하세요.

ITSM

[IT 서비스 관리 섹션](#)을 참조하세요.

L

레이블 기반 액세스 제어(LBAC)

사용자와 데이터 자체에 각각 보안 레이블 값이 명시적으로 할당되는 필수 액세스 제어(MAC)의 구현입니다. 사용자 보안 레이블과 데이터 보안 레이블 간의 교차 부분에 따라 사용자가 볼 수 있는 행과 열이 결정됩니다.

랜딩 존

랜딩 존은 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 설정](#)을 참조하십시오.

대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

LBAC

[레이블 기반 액세스 제어 섹션](#)을 참조하세요.

최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 [최소 권한 적용](#)을 참조하세요.

리프트 앤드 시프트

[7 Rs](#)를 참조하세요.

리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. [Endianness](#) 도 참조하세요.

하위 환경

[환경](#) 을 참조하세요.

M

기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공지능의 한 유형입니다. ML은 사물 인터넷 (IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 [기계 학습](#) 을 참조하십시오.

기본 브랜치

[브랜치를 참조하세요.](#)

맬웨어

컴퓨터 보안 또는 프라이버시를 침해하도록 설계된 소프트웨어입니다. 맬웨어는 컴퓨터 시스템을 중단하거나, 민감한 정보를 유출하거나, 무단 액세스를 가져올 수 있습니다. 맬웨어의 예로는 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어, 키로거 등이 있습니다.

관리형 서비스

AWS 서비스 는 인프라 계층, 운영 체제 및 플랫폼을 AWS 운영하고 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. Amazon Simple Storage Service(Amazon S3) 및 Amazon DynamoDB 는 관리형 서비스의 예입니다. 이를 추상화된 서비스 라고도 합니다.

제조 실행 시스템(MES)

원재료를 생산 현장의 완제품으로 변환하는 생산 프로세스를 추적, 모니터링, 문서화 및 제어하기 위한 소프트웨어 시스템입니다.

MAP

[마이그레이션 가속화 프로그램 단원을 참조하세요.](#)

메커니즘

도구를 생성하고 도구 채택을 유도한 다음 결과를 검사하여 조정하는 전체 프로세스입니다. 메커니즘은 작동 시 자체를 강화하고 개선하는 주기입니다. 자세한 내용은 AWS Well-Architected Framework의 [메커니즘 구축](#) 을 참조하세요.

멤버 계정

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

MES

[제조 실행 시스템](#) 을 참조하세요.

메시지 대기열 원격 측정 전송(MQTT)

리소스가 제한된 [IoT](#) 디바이스에 대한 [게시/구독](#) 패턴을 기반으로 하는 경량 machine-to-machine (M2M) 통신 프로토콜입니다.

마이크로서비스

잘 정의된 를 통해 통신APIs하고 일반적으로 소규모 독립 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 [AWS 서버리스 서비스를 사용하여 마이크로서비스 통합을 참조하세요](#).

마이크로서비스 아키텍처

각 애플리케이션 프로세스를 마이크로서비스로 실행하는 독립 구성 요소를 사용하여 애플리케이션을 구축하는 접근 방식입니다. 이러한 마이크로서비스는 경량 를 사용하여 잘 정의된 인터페이스를 통해 통신합니다APIs. 애플리케이션의 특정 기능에 대한 수요에 맞게 이 아키텍처의 각 마이크로서비스를 업데이트, 배포 및 조정할 수 있습니다. 자세한 내용은 [에서 마이크로서비스 구현을 참조하세요 AWS](#).

마이그레이션 가속화 프로그램(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄하는데 도움이 되는 컨설팅 지원, 훈련 및 서비스를 제공하는 AWS 프로그램입니다. MAP에는 체계적인 방식으로 레거시 마이그레이션을 실행하기 위한 마이그레이션 방법론과 일반적인 마이그레이션 시나리오를 자동화하고 가속화하기 위한 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 전략](#)의 세 번째 단계입니다.

마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 스프린트에서 작업하는 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자 및 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 가이드](#)와 [마이그레이션 팩토리에 대한 설명](#)을 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 및 AWS 계정이 있습니다.

마이그레이션 패턴

사용되는 마이그레이션 전략, 마이그레이션 대상, 마이그레이션 애플리케이션 또는 서비스를 자세히 설명하는 반복 가능한 마이그레이션 작업입니다. 예: AWS Application Migration Service를 EC2 사용하여 Amazon으로 마이그레이션을 다시 호스팅합니다.

마이그레이션 포트폴리오 평가(MPA)

로 마이그레이션하기 위한 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다 AWS 클라우드. MPA 는 자세한 포트폴리오 평가(서버 적정 규모, 요금, TCO 비교, 마이그레이션 비용 분석)와 마이그레이션 계획(애플리케이션 데이터 분석 및 데이터 수집, 애플리케이션 그룹화, 마이그레이션 우선 순위 지정 및 웨이브 계획)을 제공합니다. 이 [MPA 도구](#)(로그인 필요)는 모든 AWS 컨설턴트 및 APN 파트너 컨설턴트에게 무료로 제공됩니다.

마이그레이션 준비 평가(MRA)

를 사용하여 조직의 클라우드 준비 상태에 대한 통찰력을 얻고, 강점과 약점을 식별하고, 식별된 격차를 해소하기 위한 실행 계획을 수립하는 프로세스입니다 AWS CAF. 자세한 내용은 [마이그레이션 준비 가이드](#)를 참조하십시오. MRA 는 [AWS 마이그레이션 전략의 첫 번째 단계](#)입니다.

마이그레이션 전략

워크로드를 로 마이그레이션하는 데 사용되는 접근 방식입니다 AWS 클라우드. 자세한 내용은 이 용어집의 [7 Rs](#) 항목을 참조하고 [대규모 마이그레이션을 가속화하기 위해 조직 동원을 참조하세요](#).

ML

[기계 학습](#)을 참조하세요.

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 애플리케이션 현대화 전략을 참조하세요 AWS 클라우드](#).

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 [의 애플리케이션에 대한 현대화 준비 상태 평가를 참조하세요 AWS 클라우드](#).

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 분해](#)를 참조하십시오.

MPA

[마이그레이션 포트폴리오 평가 단원을 참조하세요](#).

MQTT

[메시지 대기열 원격 측정 전송을 참조하세요](#).

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 워크로드에 대한 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 신뢰성 및 예측 가능성을 높이기 위해 AWS Well-Architected Framework는 [변경할 수 없는 인프라](#)를 모범 사례로 사용할 것을 권장합니다.

O

OAC

[오리진 액세스 제어 섹션을](#) 참조하세요.

OAI

[오리진 액세스 자격 증명](#) 을 참조하세요.

OCM

[조직 변경 관리](#) 를 참조하세요.

오프라인 마이그레이션

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

OI

[작업 통합](#) 을 참조하세요.

OLA

[운영 수준 계약](#) 을 참조하세요.

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

[Open Process Communications - 통합 아키텍처](#) 를 참조하세요.

Open Process Communications - 통합 아키텍처(OPC-UA)

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

운영 수준 계약(OLA)

서비스 수준 계약을 지원하기 위해 기능 IT 그룹이 서로에게 제공할 것을 명확히 하는 계약입니다 (SLA).

운영 준비 검토(ORR)

인시던트 및 가능한 장애의 범위를 이해, 평가, 예방 또는 줄이는 데 도움이 되는 질문 및 관련 모범 사례 체크리스트입니다. 자세한 내용은 AWS Well-Architected 프레임워크의 [운영 준비 검토 \(ORR\)](#)를 참조하세요.

운영 기술(OT)

물리적 환경과 협력하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템입니다. 제조업에서 OT와 정보 기술(IT) 시스템의 통합은 [Industry 4.0](#) 혁신의 주요 초점입니다.

운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 [운영 통합 가이드](#)를 참조하십시오.

조직 트레일

에서 생성한 추적은 의 조직 AWS 계정 내 모든 에 대한 모든 이벤트를 AWS CloudTrail 기록합니다 AWS Organizations. 이 트레일은 조직에 속한 각 AWS 계정 에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 설명서의 [조직에 대한 추적 생성](#)을 참조하세요 CloudTrail.

조직 변경 관리(OCM)

사람, 문화 및 리더십 관점에서 중대하고 파괴적인 비즈니스 혁신을 관리하기 위한 프레임워크입니다. OCM 는 변화 채택을 가속화하고, 전환 문제를 해결하고, 문화적 및 조직적 변화를 주도하여 조직이 새로운 시스템 및 전략을 준비하고 전환하도록 지원합니다. AWS 마이그레이션 전략에서는 클라우드 채택 프로젝트에 필요한 변경 속도 때문에 이 프레임워크를 인력 가속화 라고 합니다. 자세한 내용은 [OCM 안내서](#)를 참조하세요.

오리진 액세스 제어(OAC)

에서는 Amazon Simple Storage Service(Amazon S3) 콘텐츠를 보호하기 위해 액세스를 제한하는 CloudFront향상된 옵션입니다. OAC 는 모든 의 모든 S3 버킷 AWS 리전, AWS KMS (-SSEKMS)를 사용한 서버 측 암호화, S3 버킷에 대한 동적 PUT 및 DELETE 요청을 지원합니다.

오리진 액세스 자격 증명(OAI)

에서 Amazon S3 콘텐츠를 보호하기 위해 액세스를 제한하는 CloudFront옵션입니다. OAI를 사용하면 Amazon S3가 인증할 수 있는 보안 주체가 CloudFront 생성됩니다. 인증된 보안 주체는 특정 CloudFront 배포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. 보다 세분화되고 향상된 액세스 제어를 [OAC](#)제공하는 도 참조하세요.

ORR

[운영 준비 검토 섹션](#)을 참조하세요.

OT

[운영 기술](#) 을 참조하세요.

아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작된 네트워크 연결을 VPC 처리하는입니다. [AWS Security Reference Architecture](#)는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 VPCs 위해 인바운드, 아웃바운드 및 검사로 네트워크 계정을 설정하는 것이 좋습니다.

P

권한 경계

보안 IAM 주체에 연결되어 사용자 또는 역할이 가질 수 있는 최대 권한을 설정하는 IAM 관리 정책입니다. 자세한 내용은 IAM 설명서의 [권한 경계](#)를 참조하세요.

개인 식별 정보(PII)

직접 보거나 다른 관련 데이터와 함께 짝을 지을 때 개인의 신원을 합리적으로 추론하는 데 사용할 수 있는 정보입니다. 의 예로는 이름, 주소 및 연락처 정보가 PII 있습니다.

PII

[개인 식별 정보](#) 를 참조하세요.

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

[프로그래밍 가능한 로직 컨트롤러](#) 를 참조하세요.

PLM

[제품 수명 주기 관리](#) 섹션을 참조하세요.

정책

권한을 정의하거나([자격 증명 기반 정책](#) 참조), 액세스 조건을 지정하거나([리소스 기반 정책](#) 참조), 조직의 모든 계정에 대한 최대 권한을 정의할 수 있는 객체입니다 AWS Organizations ([서비스 제어 정책](#) 참조).

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 활성화](#)를 참조하십시오.

포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 [마이그레이션 준비 상태 평가](#)를 참조하십시오.

조건자

WHERE 절에서 false 일반적으로 위치한 true 또는 를 반환하는 쿼리 조건입니다.

조건부 푸시다운

전송 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 controls](#)를 참조하십시오.

보안 주체

작업을 수행하고 리소스에 액세스할 수 있는 AWS 있는 엔터티입니다. 이 엔터티는 일반적으로 AWS 계정, IAM 역할 또는 사용자의 루트 사용자입니다. 자세한 내용은 IAM 설명서의 역할의 보안 주체 용어 및 개념을 참조하세요. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html#id_roles_terms-and-concepts

개인 정보 보호 중심 설계

전체 엔지니어링 프로세스에서 개인 정보를 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

프라이빗 호스팅 영역

Amazon Route 53이 하나 이상의 내에서 도메인 및 하위 도메인에 대한 DNS 쿼리에 응답하는 방법에 대한 정보를 포함하는 컨테이너입니다VPCs. 자세한 내용은 Route 53 설명서의 [프라이빗 호스팅 영역 작업](#)을 참조하십시오.

사전 예방적 제어

규정 미준수 리소스의 배포를 방지하도록 설계된 [보안 제어](#)입니다. 이러한 제어는 리소스가 프로비저닝되기 전에 스캔 리소스를 제어합니다. 리소스가 컨트롤을 준수하지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 [제어 참조 가이드](#)를 참조하고 의 보안 [제어 구현의 사전](#) 예방적 제어를 참조하세요. AWS

제품 수명 주기 관리(PLM)

설계, 개발 및 출시부터 성장 및 성숙도, 거부 및 제거에 이르기까지 전체 수명 주기 동안 제품의 데이터 및 프로세스 관리.

프로덕션 환경

[환경](#) 을 참조하세요.

프로그래밍 가능한 로직 컨트롤러(PLC)

제조에서 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(pub/sub)

마이크로서비스 간의 비동기 통신을 가능하게 하여 확장성과 응답성을 개선하는 패턴입니다. 예를 들어 마이크로서비스 기반 에서 마이크로서비스는 다른 마이크로서비스 [MES](#)가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로 서비스를 추가할 수 있습니다.

Q

쿼리 계획

SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 지침과 같은 일련의 단계입니다.

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

[책임, 책임, 상담, 정보 제공\(RACI\)을 참조하세요.](#)

랜섬웨어

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

RASCI 매트릭스

[책임, 책임, 상담, 정보 제공\(RACI\)을 참조하세요.](#)

RCAC

[행 및 열 액세스 제어](#)를 참조하세요.

읽기 전용 복제본

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

재설계

[7 Rs](#)를 참조하세요.

복구 시점 목표(RPO)

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

복구 시간 목표(RTO)

서비스 중단과 서비스 복원 사이의 허용 가능한 지연 시간입니다.

리팩터

[7 Rs](#)를 참조하세요.

리전

지리적 영역의 AWS 리소스 모음입니다. 각 AWS 리전은 내결함성, 안정성 및 복원력을 제공하기 위해 다른 과는 격리되고 독립적입니다. 자세한 내용은 [계정에서 사용할 수 있는 항목 지정을 참조 AWS 리전](#) 하세요.

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 Rs 를 참조하세요.](#)

release

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

재배치

[7 Rs 를 참조하세요.](#)

리플랫폼

[7 Rs 를 참조하세요.](#)

재구매

[7 Rs 를 참조하세요.](#)

복원력

중단에 저항하거나 복구할 수 있는 애플리케이션의 기능입니다. 에서 복원력을 계획할 때 [고가용성](#) 및 [재해 복구](#)는 일반적인 고려 사항입니다 AWS 클라우드. 자세한 내용은 [AWS 클라우드 복원력 섹션을 참조하세요.](#)

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.

책임, 책임, 상담, 정보 제공(RACI) 매트릭스

마이그레이션 활동 및 클라우드 운영에 참여하는 모든 당사자의 역할과 책임을 정의하는 매트릭스입니다. 매트릭스 이름은 매트릭스에 정의된 책임 유형에서 파생됩니다. 실무 담당자 (R), 의사 결정권자 (A), 업무 수행 조연자 (C), 결과 통보 대상자 (I). 지원자는 (S) 선택사항입니다. 지원을 포함하는 경우 매트릭스를 RASCI 매트릭스 라고 하고, 매트릭스를 제외하면 RACI 매트릭스 라고 합니다.

대응 제어

보안 기준에서 벗어나거나 부정적인 이벤트를 해결하도록 설계된 보안 제어입니다. 자세한 내용은 [Implementing security controls on AWS의 Responsive controls](#)를 참조하십시오.

retain

[7 Rs 를 참조하세요.](#)

사용 중지

[7 Rs 를 참조하세요.](#)

교체

공격자가 보안 인증 정보에 액세스하는 것을 더 어렵게 만들기 위해 [보안 암호](#)를 주기적으로 업데이트하는 프로세스입니다.

행 및 열 액세스 제어(RCAC)

액세스 규칙을 정의한 기본적이고 유연한 SQL 표현식 사용. RCAC 는 행 권한과 열 마스크로 구성됩니다.

RPO

[복구 시점 목표 를 참조하세요.](#)

RTO

[복구 시간 목표 를 참조하세요.](#)

런복

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

S

SAML 2.0

많은 자격 증명 공급자(IdPs)가 사용하는 개방형 표준입니다. 이 기능을 사용하면 페더레이션 Single Sign-On(SSO)을 사용할 수 있으므로 사용자는 조직의 모든 사용자에게 AWS API IAM 대해 에서 사용자를 만들지 않고도 AWS Management Console 에 로그인하거나 작업을 호출할 수 있습니다.

다. SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 [SAML 2.0 기반 페더레이션 정보를 참조하세요.](#)

SCADA

[관리 제어 및 데이터 수집](#) 을 참조하세요.

SCP

[서비스 제어 정책](#) 을 참조하세요.

secret

에서 암호 또는 사용자 자격 증명과 같이 암호화된 형식으로 저장하는 AWS Secrets Manager 기밀 또는 제한된 정보입니다. 보안 암호 값과 메타데이터로 구성됩니다. 보안 암호 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 [Secrets Manager 설명서의 Secrets Manager 보안 암호의 내용을 참조하세요.](#)

보안 제어

위험 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어에는 [예방](#), [탐지](#), [대응](#), [사전](#) 예방 의 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) 및 보안 이벤트 관리(SEM) 시스템을 결합하는 도구 및 서비스입니다. SIEM 시스템은 서버, 네트워크, 디바이스 및 기타 소스에서 데이터를 수집, 모니터링 및 분석하여 위협 및 보안 위반을 탐지하고 알림을 생성합니다.

보안 응답 자동화

보안 이벤트에 자동으로 응답하거나 수정하도록 설계된 사전 정의되고 프로그래밍된 작업입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 [탐지적](#) 또는 [대응적](#) AWS 보안 제어 역할을 합니다. 자동 응답 작업의 예로는 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치 적용 또는 보안 인증 정보 교체 등이 있습니다.

서버 측 암호화

대상에서 데이터를 AWS 서비스 수신하는 에 의한 데이터 암호화.

서비스 제어 정책(SCP)

AWS Organizations에 속한 조직의 모든 계정에 대한 권한을 중앙 집중식으로 제어하는 정책입니다. SCPs 가드레일을 정의하거나 관리자가 사용자 또는 역할에 위임할 수 있는 작업에 대한 제한을 설정합니다. 허용 목록 또는 거부 목록 SCPs으로 를 사용하여 허용되거나 금지된 서비스 또는 작업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 [서비스 제어 정책을](#) 참조하세요.

서비스 엔드포인트

에 대한 진입점 URL의 입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 [AWS 서비스 엔드포인트](#)를 참조하십시오.

서비스 수준 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 측면 측정입니다.

서비스 수준 목표(SLO)

서비스 [수준 지표](#) 로 측정된 서비스 상태를 나타내는 대상 지표입니다.

공동 책임 모델

클라우드 보안 및 규정 준수에 AWS 대해 공유하는 책임을 설명하는 모델입니다. AWS 는 클라우드의 보안을 책임지고, 는 클라우드의 보안을 책임집니다. 자세한 내용은 [공동 책임 모델](#)을 참조하십시오.

SIEM

[보안 정보 및 이벤트 관리 시스템](#) 을 참조하세요.

단일 장애 지점(SPOF)

시스템을 중단시킬 수 있는 애플리케이션의 중요한 단일 구성 요소에서 발생하는 장애입니다.

SLA

[서비스 수준 계약을](#) 참조하세요.

SLI

[서비스 수준 표시기](#) 를 참조하세요.

SLO

[서비스 수준 목표를 참조하세요.](#)

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 [에서 애플리케이션 현대화에 대한 단계별 접근 방식을 참조하세요 AWS 클라우드.](#)

SPOF

[단일 장애 지점을 참조하세요.](#)

스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정된 데이터를 저장하고 하나 이상의 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 조직 구조입니다. 이 구조는 [데이터 웨어하우스](#) 또는 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 Fowler](#)가 모놀리식 시스템을 다시 작성할 때 위험을 관리하는 방법으로 이 패턴을 도입했습니다. 이 패턴을 적용하는 방법의 예는 [컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET \(ASMX\) 웹 서비스 증분 현대화를 참조하세요.](#)

서브넷

의 IP 주소 범위입니다VPC. 서브넷은 단일 가용 영역에 상주해야 합니다.

감독 제어 및 데이터 수집(SCADA)

제조에서 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 Synthetics](#)를 사용하여 이러한 테스트를 생성할 수 있습니다.

T

tags

AWS 리소스를 구성하기 위한 메타데이터 역할을 하는 키-값 페어입니다. 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다. 자세한 내용은 [AWS 리소스에 태그 지정](#)을 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

[환경](#)을 참조하세요.

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

VPCs 및 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 [전송 게이트웨이란 무엇입니까?](#)를 참조하세요.

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

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

관리 작업을 수행합니다. 자세한 내용은 설명서의 [다른 AWS 서비스와 AWS Organizations 함께 사용을 참조하세요](#) AWS Organizations .

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

두 개의 피자로 먹을 수 있는 작은 DevOps 팀입니다. 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

[환경](#) 을 참조하세요.

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 VPCs 있는 두 개의 연결입니다. 자세한 내용은 Amazon VPC 설명서의 [VPC 피어링이란 무엇입니까?](#)를 참조하세요.

취약성

시스템 보안을 손상시키는 소프트웨어 또는 하드웨어 결함입니다.

W

웹 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.

창 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에 대해 계산을 수행하는 SQL 함수입니다. 창 함수는 이동 평균 계산 또는 현재 행의 상대적 위치를 기반으로 행 값에 액세스하는 등의 작업을 처리하는데 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.

WORM

[쓰기를 한 번 참조하고 많은 을 읽습니다.](#)

WQF

[AWS 워크로드 검증 프레임워크를](#) 참조하세요.

한 번 쓰기, 많이 읽기(WORM)

데이터를 한 번에 쓰고 데이터가 삭제되거나 수정되는 것을 방지하는 스토리지 모델입니다. 권한 있는 사용자는 필요한 만큼 데이터를 읽을 수 있지만 변경할 수는 없습니다. 이 데이터 스토리지 인 프라는 [변경할 수 없는](#) 로 간주됩니다.

Z

제로데이 익스플로잇

[제로데이 취약성을 활용하는 공격, 일반적으로 맬웨어입니다.](#)

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.