



Running Containers on Amazon EKS

모듈 5:

Amazon EKS를 사용하여 대규모로
애플리케이션 관리



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



| 강사용 노트

| 이 모듈을 강의하는 데는 약 **1시간 15분** 정도 걸립니다.

| 이 모듈을 마치면 실습 **3**을 진행해야 합니다.

| 수강생용 노트

이 모듈은 **Amazon EKS**를 사용하여 대규모로 애플리케이션 관리입니다



Running Containers on Amazon EKS

모듈 5 개요

- Amazon EKS에서 수요에 맞게 크기 조정
- Amazon EKS에 지속적 배포
- GitOps 및 Amazon EKS
- 실습 3 준비



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



Running Containers on Amazon EKS

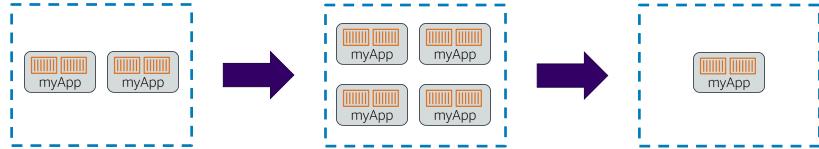
Amazon EKS에서 수요에 맞게 크기 조정



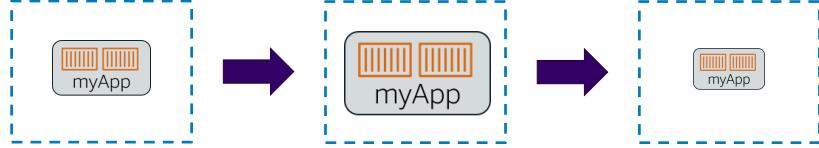
©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

복습: 수평 및 수직적 크기 조정

수평: 스케일 인 또는 스케일 아웃합니다. 리소스의 **수**를 조정합니다.



수직: 스케일 업 또는 스케일 다운합니다. 리소스의 **크기**를 조정합니다.



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

~Dev notes

~Alt text: 수평적 크기 조정: Pod 수가 확장 후 축소됩니다.

~Alt text: 수직적 크기 조정: Pod가 확장 후 축소됩니다.

~

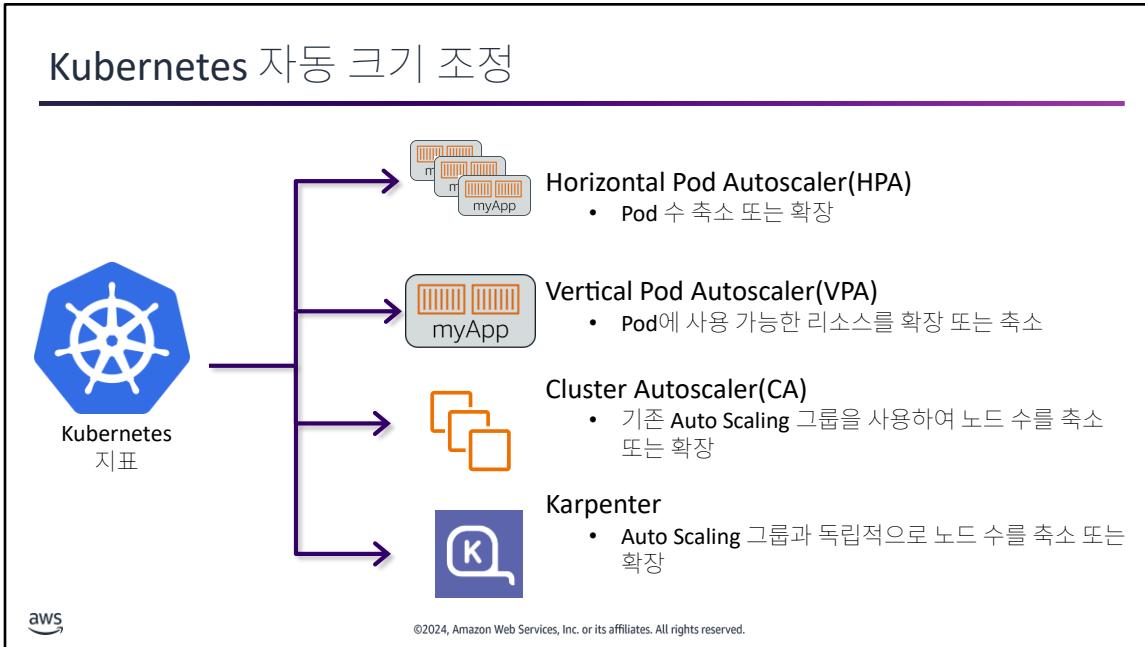
| 수강생용 노트

수평으로 확장 가능한 시스템이란 컴퓨팅 리소스를 추가하거나 제거하여 용량을 늘리거나 줄일 수 있는 시스템입니다. 이는 성능을 향상시키는 유일한 방법이 컴퓨팅 리소스에 더 빠른(또는 더 많은) **CPU**, 메모리 또는 스토리지로 더 많은 리소스를 추가하는 것뿐인 수직 확장 가능한 시스템과는 대조적입니다.

기존 서버의 경우 수평적 크기 조정이 가능한 시스템은 수직적 크기 조정만 가능한 시스템보다 성능이 뛰어난 경우가 많습니다. 클라우드 컴퓨팅과 **Kubernetes**에서는 그렇지 않습니다. **Kubernetes**에는 수직 및 수평으로 크기를 조정할 수 있는 메커니즘이 있습니다.

첫 번째 예에서는 수요가 급증할 때 더 많은 Pod가 배포됩니다(확장). 수요가 감소하면 Pod가 제거됩니다(축소). 두 번째 예에서는 수요가 급증할 때 Pod의 크기(할당된 CPU 및 메모리 리소스)가 증가하고(확장) 수요가 감소하면 크기가 줄어듭니다(축소).

Kubernetes 자동 크기 조정



~Alt text: 자동 크기 조정 도구: Kubernetes 지표는 Horizontal Pod Autoscaler, Vertical Pod Autoscaler, Cluster Autoscaler, Karpenter를 지원합니다.

|수강생용 노트

Kubernetes 지표는 수요 변동에 대응하는 자동 크기 조정에 필요한 데이터를 제공합니다.

Horizontal Pod Autoscaler(HPA)는 Pod 수를 확장하거나 축소합니다. 기본적으로 HPA는 Kubernetes 지표 서버의 지표를 사용합니다. Kubernetes 지표 서버는 CPU 및 RAM과 같은 컴퓨팅 리소스를 모니터링합니다. 또한 대기 시간 같이 다른 지표를 기준으로 Pod 수를 조절하고 싶다면 다른 소스에서 사용자 지정 지표를 사용하도록 HPA를 구성하는 것도 가능합니다.

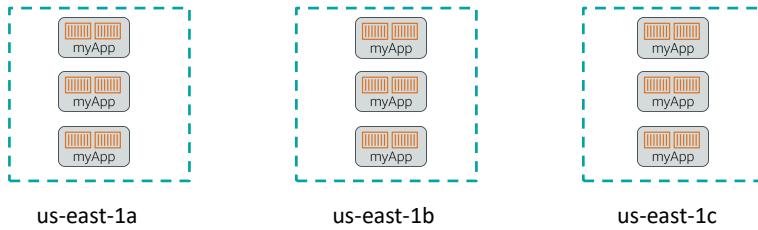
Vertical Pod Autoscaler(VPA)는 Pod에 사용할 수 있는 리소스 수를 조절합니다. VPA는 Kubernetes 지표 서버 및 Prometheus 지표의 지표를 사용합니다.

Cluster Autoscaler(CA)는 클러스터의 현재 상태에 대해 수집한 지표를 기반으로 클러스터의 노드 수를 확장하거나 축소합니다. 예: 클러스터에서 보류 중(즉, 예약 대기 중)인 Pod 수.

CA와 마찬가지로 **Karpenter**는 자동으로 컴퓨팅 리소스를 시작하여 애플리케이션을 실행합니다. **Karpenter**는 예약 불가능인 **Pod** 수와 실행 중인 **Pod**의 현재 컴퓨팅 리소스 사용량을 기반으로 클러스터 노드의 크기를 조정합니다. **CA**와 **Karpenter**의 주요 차이점은 **Karpenter**는 **Amazon EC2 Auto Scaling** 그룹을 사용하지 않고 기존 **ASG**와 독립적으로 적정 규모의 컴퓨팅 리소스를 배포한다는 것입니다. **Karpenter**는 필요에 따라 적정 규모의 노드를 프로비저닝하므로 시간 경과에 따라 통계적으로 노드 믹스를 수평적 및 수직적으로 최적화할 수 있습니다.

자세한 내용은 이 모듈의 **Karpenter** 섹션에서 확인할 수 있습니다.

Horizontal Pod Autoscaler



```
$ kubectl autoscale deployment myapp --cpu-percent=50 --min=3 --max=10
```



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

~Dev notes

~Animation will break on image flattening, please do not group

~all items (except for the code snippet) are marked as decorative. The alt text is located on “Rectangle 2” (the us-east-1a border) for accessibility purposes.

~ Alt text: 각각 1개의 Pod가 있는 3개의 가용성 영역이 표시되어 있습니다. 그런 다음 슬라이드에는 Pod 수를 AZ당 3개로 확장하기 위해 생성되는 추가 Pod가 표시됩니다.

| 강사용 노트

| 클릭 – Pod 수가 AZ당 1개에서 AZ당 3개까지 확장됩니다.

| 수강생용 노트

컨테이너 사용의 이점 중 하나는 애플리케이션을 빠르게 자동으로 크기 조정할 수 있다는 점입니다. 많은 워크로드의 경우 인바운드 연결 요청 또는 작업 대기열 길이와 같은 사용자 정의 지표를 기반으로 애플리케이션 크기 조정을 정의할 수 있는 것이 중요합니다. **Horizontal Pod Autoscaler(HPA)**는 CPU 사용률 또는 **Kubernetes** 지표 서버를 통해 정의한 기타 지표를 기준으로 서비스를 자동으로 스케일 인하거나 스케일 아웃하는 **Kubernetes** 구성 요소입니다.

HPA는 다음을 수행하여 서비스를 개선합니다.

- 과도한 수의 Pod로 인해 낭비될 수 있는 하드웨어 리소스를 해제합니다.
- 서비스 수준 계약을 달성하기 위해 필요에 따라 성능을 높이거나 낮출 수 있습니다.

이 예에서는 HPA 리소스가 생성될 때 50%의 CPU 비율이 지정되었습니다. CPU 사용량이 할당된 컨테이너 리소스의 50%를 초과하면 HPA는 지정된 최소값(3)에서 구성된 최대값(10)으로 스케일 아웃합니다. HPA는 CPU 평균이 목표(50%)보다 낮아질 때까지 스케일 아웃합니다. 로드가 감소하면 Amazon EKS는 최종적으로 최소 개수까지 Pod 수를 되돌립니다.

참고: HPA는 기본적으로 CPU 사용량을 기준으로 크기를 조정하지만 메모리에 따라 크기를 조정하도록 구성할 수도 있습니다. 슬라이드의 예제 명령에서 ‘CPU’를 ‘memory’로 변경하면 이 작업이 완료됩니다.

HPA는 기본적으로 CPU 사용률을 사용하지만 사용자 지정 지표를 사용하도록 구성할

수 있습니다. 예: Amazon CloudWatch Metrics Adapter for Kubernetes(k8s-cloudwatch-adapter)를 사용하는 CloudWatch의 지표. 자세한 내용은 AWS 컴퓨팅 블로그의 'Scaling Kubernetes deployments with Amazon CloudWatch metrics' (<https://aws.amazon.com/blogs/compute/scaling-kubernetes-deployments-with-amazon-cloudwatch-metrics/>)를 참조하십시오.

Amazon EKS 클러스터에서 HPA를 사용하는 방법을 자세히 알아보려면 **Amazon EKS** 사용 설명서의 Horizontal Pod Autoscaler 섹션 (<https://docs.aws.amazon.com/eks/latest/userguide/horizontal-pod-autoscaler.html>)을 참조하십시오.

Vertical Pod Autoscaler

- 문제: 모든 것이 변화합니다.
 - 일일 또는 주간 트래픽 패턴
 - 시간이 지남에 따라 증가하는 사용자 기반
 - 리소스 요구 사항이 다른 애플리케이션 수명 주기 단계

내일도 이것이 정확합니까? →

```
apiVersion: apps/v1
kind: Deployment
...
template:
  metadata:
    labels:
      app: worker
  spec:
    containers:
      - name: worker
        resources:
          requests:
            memory: "100M"
            cpu: "250m"
```



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

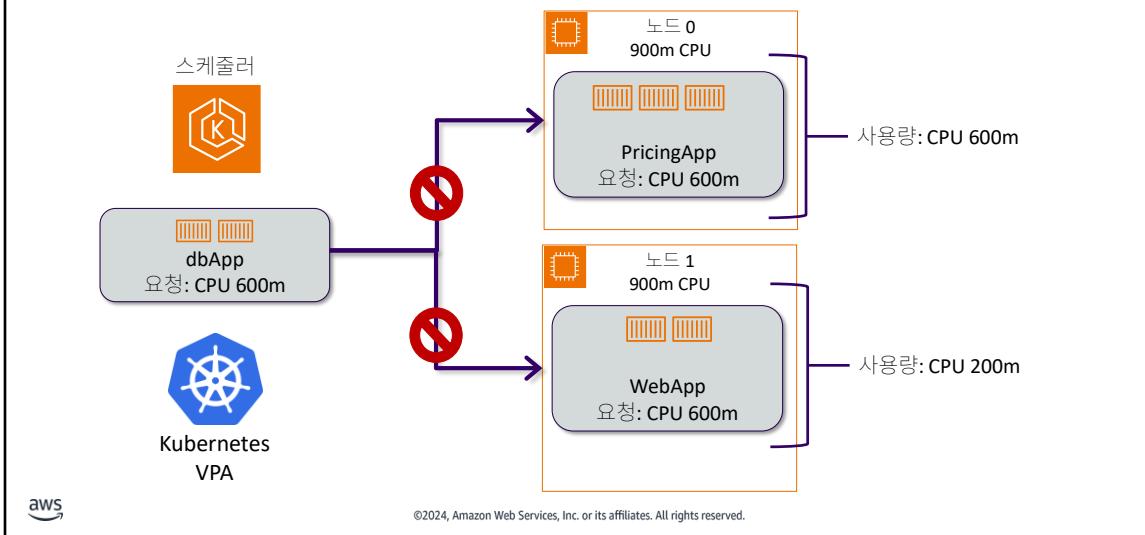
| 수강생용 노트

Kubernetes Vertical Pod Autoscaler는 Pod에 예약된 CPU 및 메모리를 자동으로 조정하여 애플리케이션의 크기를 적절히 조정할 수 있도록 지원합니다. 이러한 조정을 통해 클러스터 리소스 사용률을 개선하고 다른 Pod를 위한 CPU 및 메모리를 확보할 수 있습니다.

이 예는 100MB의 메모리와 250millicpu(1000millicpu = vCPU 코어 1개)에 대한 초기 리소스 요청을 보여줍니다.

Kubernetes 리소스 단위에 대한 자세한 내용은 Kubernetes 온라인 문서의 'Kubernetes의 리소스 단위' (<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#meaning-of-cpu>)를 참조하십시오.

예: Vertical Pod Autoscaler(1/4)



~Alt text

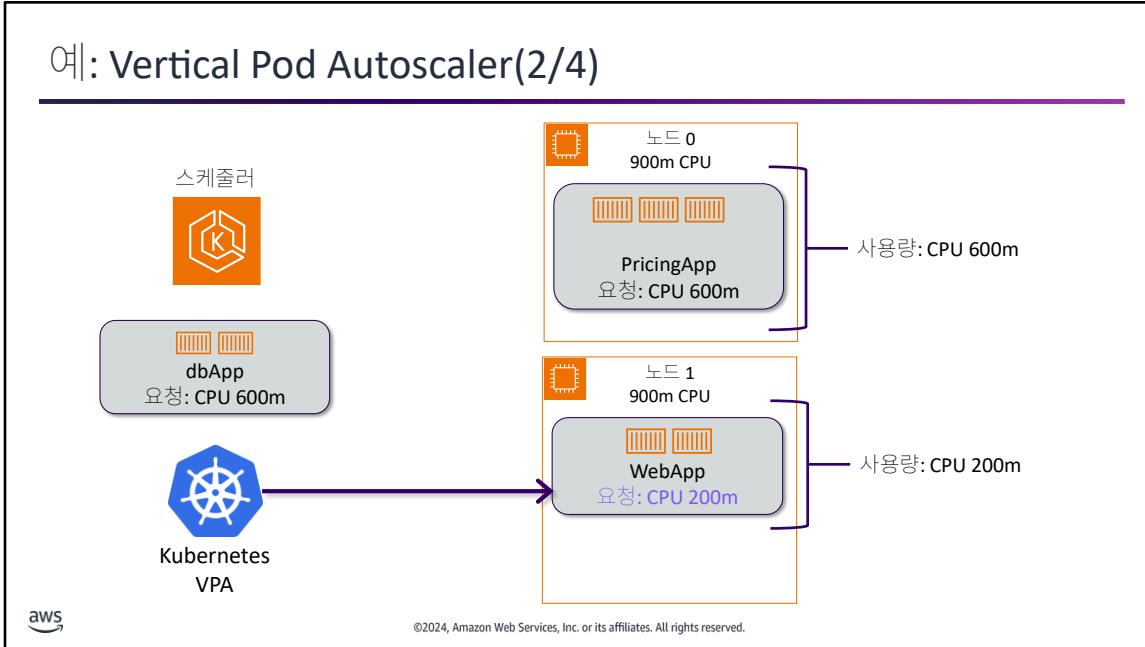
~ 컴퓨팅 리소스가 부족하기 때문에 어떤 노드에서도 Pod를 실행하도록 스케줄링할 수 없습니다.

~

|수강생용 노트

Vertical Pod Autoscaler를 사용하는 예입니다. Pod가 스케줄링되기를 기다리고 있습니다. Pod를 실행할 수 없는 이유는 노드에 Pod를 수용할 CPU 리소스가 부족하기 때문입니다. 그런데 600m의 CPU를 요청한 WebApp Pod가 200m만 사용하고 있습니다.

예: Vertical Pod Autoscaler(2/4)



~Alt text

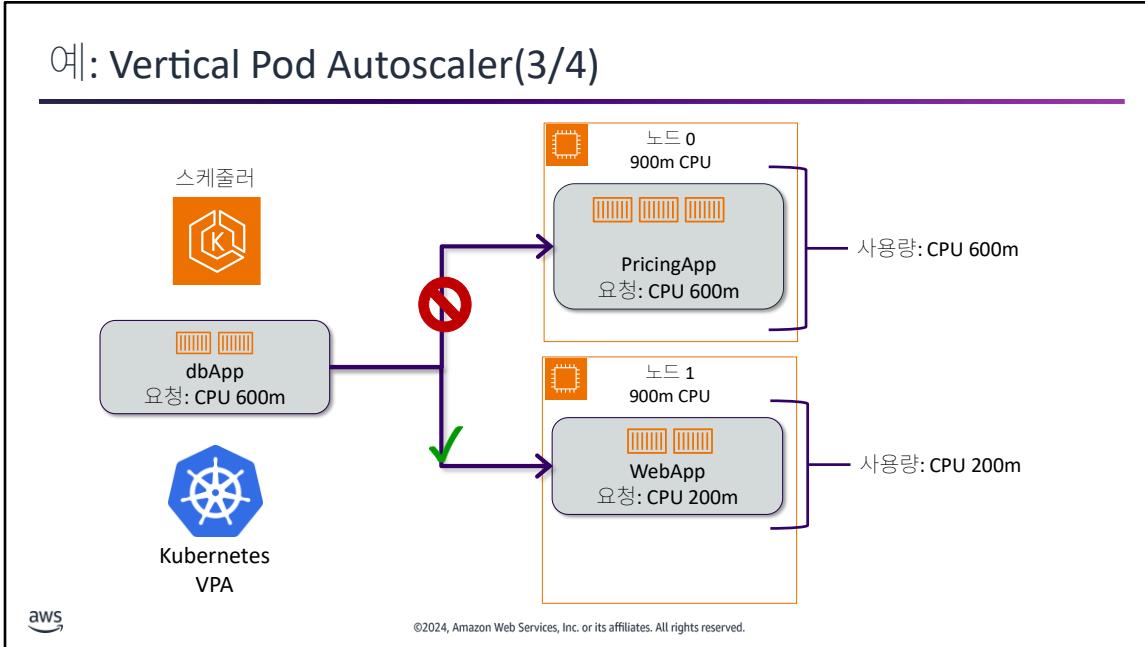
~ Kubernetes VPA는 실제 사용량과 일치하도록 Pod의 리소스 요청을 조정합니다.

~

| 수강생용 노트

Kubernetes Vertical Pod Autoscaler는 실제 사용률에 맞게 WebApp Pod의 리소스 스케줄링을 조정합니다. 리소스 스케줄링을 조정하려면 Pod를 제거하고 다시 스케줄링해야 합니다.

예: Vertical Pod Autoscaler(3/4)



~Alt text

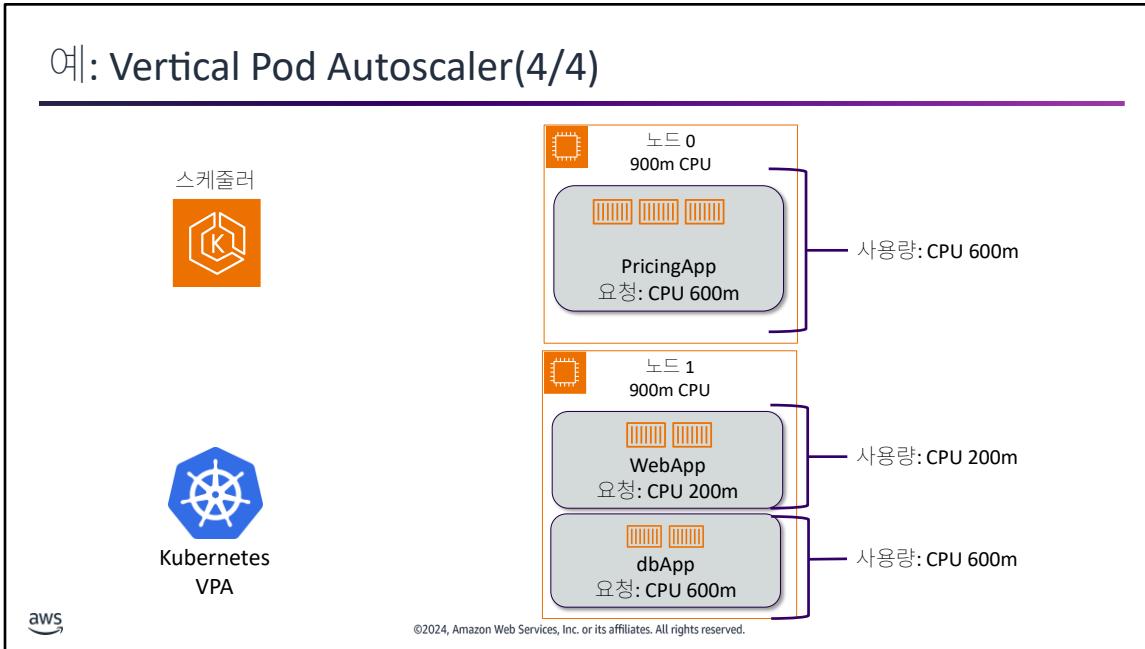
~ 이제 클러스터에는 보류 중인 Pod에 사용할 수 있는 컴퓨팅 용량이 있습니다.

~

|수강생용 노트

이제 스케줄러가 노드 1에서 dbApp Pod를 실행하기에 충분한 리소스를 사용할 수 있습니다.

예: Vertical Pod Autoscaler(4/4)



~Alt text

~ 클러스터에서 모든 **Pod**가 실행 중입니다.

~

| 수강생용 노트

이제 **dbApp Pod**가 노드 1에서 실행 중입니다.

Vertical Pod Autoscaler 구성 요소

VPA는 클러스터에서 3개의 배포로 실행됩니다.

NAME	READY	STATUS	RESTARTS	AGE
aws-node-949vx	1/1	Running	0	122m
aws-node-b4nj8	1/1	Running	0	122m
coredns-6c75b69b98-r9x68	1/1	Running	0	133m
coredns-6c75b69b98-rt9bp	1/1	Running	0	133m
kube-proxy-bkm6b	1/1	Running	0	122m
kube-proxy-hpqm2	1/1	Running	0	122m
metrics-server-8459fc497-kfj8w	1/1	Running	0	83m
vpa-admission-controller-68c748777d-ppspd	1/1	Running	0	7s
vpa-recommender-6fc8c67d85-g1jp1	1/1	Running	0	8s
vpa-updater-786b96955c-bgp9d	1/1	Running	0	8s



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

| 수강생용 노트

Vertical Pod Autoscaler는 클러스터에서 3개의 배포를 실행합니다. Vertical Pod Autoscaler는 3가지 구성 요소로 구성됩니다.

- **Recommender** - 현재 및 과거 리소스 소비를 모니터링하고 컨테이너의 CPU 및 메모리 요청에 대한 권장 값을 제공합니다.
- **Updater** - 올바른 리소스 세트가 있는 관리형 Pod를 확인합니다. 리소스가 잘못 할당된 Pod를 종료하여 컨트롤러가 업데이트된 요청으로 다시 생성할 수 있도록 합니다.
- **Admission Plugin** - 방금 생성되었거나 Updater의 활동으로 인해 컨트롤러에 의해 다시 생성된 새 Pod에 올바른 리소스 요청을 설정합니다.

Vertical Pod Autoscaler는 기본적으로 Kubernetes 지표 서버의 지표를 사용합니다. 그러나 Prometheus를 사용하여 Vertical Pod Autoscaler에 대한 지표를 제공할 수도 있습니다. 자세한 내용은 Amazon EKS 사용 설명서의 'Prometheus를 사용한 제어 영역 지표'(<https://docs.aws.amazon.com/eks/latest/userguide/prometheus.html>)을 참조하십시오.

Vertical Pod Autoscaler 고려 사항

- VPA는 세 가지 모드로 실행될 수 있습니다.
 - Off(권장 사항 전용)
 - Initial(초기화 전용)
 - Auto
- VPA 모범 사례를 따릅니다(목록은 슬라이드 노트 참조).



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

| 수강생용 노트

Vertical Pod Autoscaler는 3가지 모드 중 하나로 실행할 수 있습니다.

- Off(권장 사항 전용)
 - 권장되는 Pod 크기만 게시하며 실제로는 아무것도 변경되지 않습니다.
 - 권장 사항 전용 모드로 시작(`Kubectl get vpa myvpa --output yaml`)
- Initial(초기화 전용)
 - Pod가 생성될 때만 Pod의 크기를 조정합니다.
- Auto
 - 기존 Pod를 재시작하여 Pod 크기를 조정합니다.
 - 자동 모드 사용 시 Pod 중단 예산 정의

Vertical Pod Autoscaler를 사용할 때는 다음과 같은 중요한 제한 사항을 기억하십시오.

최신 정보는 Vertical Pod Autoscaler의 GitHub 리포지토리

(<https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler>)를 참조하십시오.

고려 사항

Vertical Pod Autoscaler를 사용하는 경우 크기 조정을 위해 고려해야 할 많은 옵션, 개선 사항, 모범 사례가 있습니다.

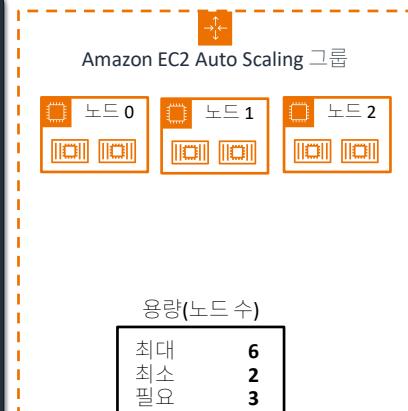
- Vertical Pod Autoscaler에서 최소 및 최대 컨테이너 크기 설정
- 실행 중인 Pod의 업데이트는 Vertical Pod Autoscaler의 실험적 기능입니다. Vertical Pod Autoscaler가 Pod 리소스를 업데이트할 때마다 Pod가 다시 생성됩니다. Pod를 다시 생성하면 실행 중인 모든 컨테이너가 다시 시작됩니다. Pod는 다른 노드에서 다시 생성할 수 있습니다.
- Vertical Pod Autoscaler는 컨트롤러에서 실행되는 Pod를 제거합니다. 이러한 Pod의 경우 자동 모드는 현재 초기 모드와 동일합니다.
- CPU 또는 메모리의 Horizontal Pod Autoscaler와 함께 Vertical Pod Autoscaler를 사용하지 마십시오. 그러나 사용자 정의 및 외부 지표에는 Horizontal Pod

Autoscaler와 함께 Vertical Pod Autoscaler를 사용할 수 있습니다.

- Vertical Pod Autoscaler 승인 컨트롤러는 승인 웹훅입니다. 클러스터에 다른 승인 웹훅을 추가하는 경우 상호 작용 방식과 서로 충돌할 수 있는지 여부를 분석하는 것이 중요합니다. 승인 컨트롤러의 순서는 API 서버의 플래그로 정의됩니다.
- Vertical Pod Autoscaler는 대부분의 메모리 부족 이벤트에 반응하지만 모든 상황에서 그런 것은 아닙니다.
- Vertical Pod Autoscaler 성능은 대규모 클러스터에서 테스트되지 않았습니다.
- Vertical Pod Autoscaler 권장 사항이 사용 가능한 리소스(예: 노드 크기, 사용 가능한 크기, 사용 가능한 할당량)를 초과하여 Pod가 보류될 수 있습니다. 이 문제는 Cluster Autoscaler와 함께 Vertical Pod Autoscaler를 사용하여 부분적으로 해결할 수 있습니다.
- 동일한 Pod와 일치하는 여러 Vertical Pod Autoscaler 리소스에 정의되지 않은 동작이 있습니다.

Cluster Autoscaler 및 Amazon EKS(1/5)

```
$ kubectl get pods -l app=myapp
NAME READY STATUS RESTARTS AGE
myapp-a5cb25d9g4-kpqgm 1/1 Running 0 5d
myapp-a5cb25d9g4-5f6gu 1/1 Running 0 5d
myapp-a5cb25d9g4-zb2x7 1/1 Running 0 4d
myapp-a5cb25d9g4-g9wjb 1/1 Running 0 3d
myapp-a5cb25d9g4-7tkh3 1/1 Running 0 3d
myapp-a5cb25d9g4-zbwf6 1/1 Running 0 3d
```



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

~Alt text

~ 3개의 노드를 실행 중인 Amazon EC2 Auto Scaling 그룹입니다. 각 노드는 2개의 Pod를 실행 중입니다.

~

| 수강생용 노트

Kubernetes Cluster Autoscaler

Kubernetes Cluster Autoscaler는 클러스터를 분석하고 노드 수를 자동으로 조정합니다. 리소스 부족으로 인해 현재 노드에서 스케줄링할 수 없는 Pod가 있는 경우 Cluster Autoscaler는 더 많은 노드를 추가하도록 요청합니다. 상당한 시간 동안 필요하지 않은 노드가 있는 경우(사용률이 낮고 중요한 Pod를 모두 다른 곳으로 이동할 수 있음) Cluster Autoscaler는 노드 그룹을 축소하도록 요청합니다.

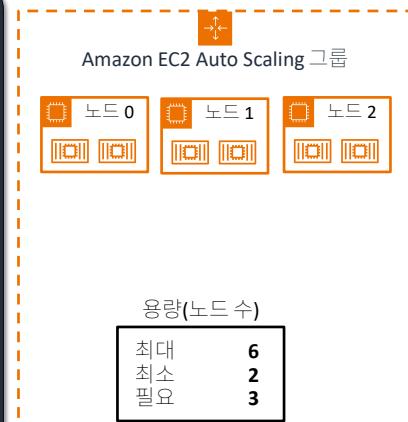
Amazon EKS에서 Cluster Autoscaler는 EC2 Auto Scaling 그룹을 사용하여 확장하거나 축소합니다. 관리형 및 비관리형 노드 그룹은 모두 Cluster Autoscaler와 호환됩니다.

이 예에서는 클러스터의 각 노드에 myapp replicaset에서 2개의 Pod를 실행하기에 충분한 리소스가 있습니다.

Cluster Autoscaler 및 Amazon EKS(2/5)

```
$ kubectl scale deployment myapp --replicas 10
$ kubectl get pods -l app=myapp

NAME READY STATUS RESTARTS AGE
myapp-a5cb25d9g4-kpqgm 1/1 Running 0 5d
myapp-a5cb25d9g4-5f6gu 1/1 Running 0 5d
myapp-a5cb25d9g4-zb2x7 1/1 Running 0 4d
myapp-a5cb25d9g4-g9wjg 1/1 Running 0 3d
myapp-a5cb25d9g4-7tkh3 1/1 Running 0 3d
myapp-a5cb25d9g4-ybwf6 1/1 Running 0 3d
myapp-a5cb25d9g4-vf25u 1/1 Pending 0 2m
myapp-a5cb25d9g4-k68cp 1/1 Pending 0 2m
myapp-a5cb25d9g4-dqy4v 1/1 Pending 0 2m
myapp-a5cb25d9g4-m7gfd 1/1 Pending 0 2m
```



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

~Alt text

~ 3개의 노드를 실행 중인 Amazon EC2 Auto Scaling 그룹입니다. 각 노드는 2개의 Pod를 실행 중입니다.

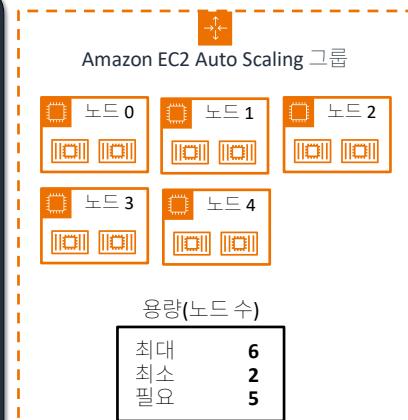
~

|수강생용 노트

이 예에서는 관리자가 Deployment를 확장했습니다. 리소스 부족으로 실행할 수 없는 보류 중 상태의 Pod가 있습니다. Cluster Autoscaler는 기본적으로 10초마다 예약 불가능 Pod를 검색합니다. CA는 스케줄링 불가능 Pod를 감지하면 조치를 취합니다.

Cluster Autoscaler 및 Amazon EKS(3/5)

```
$ kubectl get pods -l app=myapp
NAME READY STATUS RESTARTS AGE
myapp-a5cb25d9g4-kpqgm 1/1 Running 0 5d
myapp-a5cb25d9g4-5f6gu 1/1 Running 0 5d
myapp-a5cb25d9g4-zb2x7 1/1 Running 0 4d
myapp-a5cb25d9g4-g9wjb 1/1 Running 0 3d
myapp-a5cb25d9g4-7tkh3 1/1 Running 0 3d
myapp-a5cb25d9g4-ybwf6 1/1 Running 0 3d
myapp-a5cb25d9g4-vf25u 1/1 Running 0 11s
myapp-a5cb25d9g4-k68cp 1/1 Running 0 11s
myapp-a5cb25d9g4-dqy4v 1/1 Running 0 16s
myapp-a5cb25d9g4-m7gfd 1/1 Running 0 16s
```



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

~Alt text

~ 5개의 노드를 실행 중인 Amazon EC2 Auto Scaling 그룹입니다. 각 노드는 2개의 Pod를 실행 중입니다.

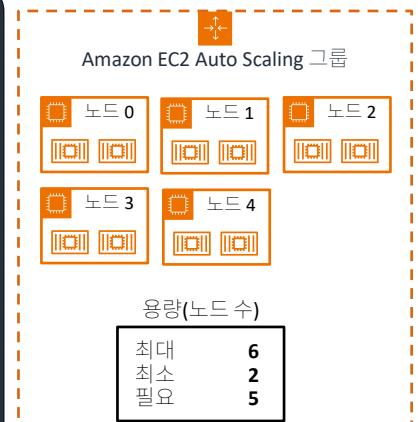
~

|수강생용 노트

이 예에서 Cluster Autoscaler는 더 많은 리소스가 필요한 것을 감지하고 Auto Scaling 그룹을 확장합니다. 이제 모든 Pod를 실행할 수 있는 노드가 충분합니다.

Cluster Autoscaler 및 Amazon EKS(4/5)

```
$ kubectl get pods -l app=myapp
$ kubectl scale deployment myapp --replicas 14
NAME READY STATUS RESTARTS AGE
myapp-a5cb25d9g4-kpqgm 1/1 Running 0 5d
myapp-a5cb25d9g4-5f6gu 1/1 Running 0 5d
myapp-a5cb25d9g4-zb2x7 1/1 Running 0 4d
myapp-a5cb25d9g4-g9wjb 1/1 Running 0 3d
myapp-a5cb25d9g4-7tkh3 1/1 Running 0 3d
myapp-a5cb25d9g4-ybwf6 1/1 Running 0 3d
myapp-a5cb25d9g4-vf25u 1/1 Running 0 4m
myapp-a5cb25d9g4-k68cp 1/1 Running 0 4m
myapp-a5cb25d9g4-dqy4v 1/1 Running 0 4m
myapp-a5cb25d9g4-m7gfd 1/1 Running 0 4m
myapp-a5cb25d9g4-m1f9y 1/1 Pending 0 2m
myapp-a5cb25d9g4-fk6rm 1/1 Pending 0 2m
myapp-a5cb25d9g4-6jf6 1/1 Pending 0 2m
myapp-a5cb25d9g4-skh8x 1/1 Pending 0 2m
```



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

~Alt text

~ 5개의 노드를 실행 중인 Amazon EC2 Auto Scaling 그룹입니다. 각 노드는 2개의 Pod를 실행 중입니다.

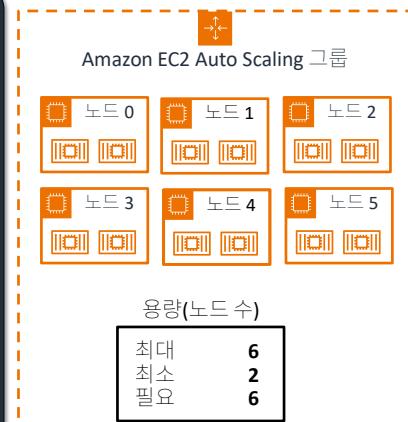
~

|수강생용 노트

이 예에서는 관리자가 배포를 다시 스케일 아웃했습니다.

Cluster Autoscaler 및 Amazon EKS(5/5)

```
$ kubectl get pods -l app=myapp
NAME READY STATUS RESTARTS AGE
myapp-a5cb25d9g4-kpqgm 1/1 Running 0 5d
myapp-a5cb25d9g4-5f6gu 1/1 Running 0 5d
myapp-a5cb25d9g4-zb2x7 1/1 Running 0 4d
myapp-a5cb25d9g4-g9wj 1/1 Running 0 3d
myapp-a5cb25d9g4-7tkh3 1/1 Running 0 3d
myapp-a5cb25d9g4-ybwf6 1/1 Running 0 3d
myapp-a5cb25d9g4-vf25u 1/1 Running 0 14m
myapp-a5cb25d9g4-k68cp 1/1 Running 0 14m
myapp-a5cb25d9g4-dqy4v 1/1 Running 0 14m
myapp-a5cb25d9g4-m7gfd 1/1 Running 0 14m
myapp-a5cb25d9g4-m1f9y 1/1 Running 0 4m
myapp-a5cb25d9g4-fk6rm 1/1 Running 0 4m
myapp-a5cb25d9g4-6j4f6 1/1 Pending 0 12m
myapp-a5cb25d9g4-skh8x 1/1 Pending 0 12m
```



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

~Alt text

~ 6개의 노드를 실행 중인 Amazon EC2 Auto Scaling 그룹입니다. 각 노드는 2개의 Pod를 실행 중입니다.

~

|수강생용 노트

Cluster Autoscaler가 리소스가 더 필요하다고 감지하면 Auto Scaling 그룹을 다시 확장합니다. 이 예에서는 Auto Scaling 그룹의 최대 용량 설정이 6개의 인스턴스로 설정되어 있어 Cluster Autoscaler가 보류 중인 모든 Pod를 실행할 수 있을 만큼 확장되지 않습니다. 이 문제를 해결하려면 사용 가능한 리소스를 늘리거나(예: 최대 인스턴스 수 증가) 워크로드를 줄이거나(예: Pod 수 감소) 해야 합니다.

Cluster Autoscaler 모범 사례

- 자동 검색 설정이 선호됩니다.
- Auto Scaling 그룹에서 최소/최대 그룹 크기를 직접 조정합니다.
- MixedInstancesPolicy를 사용하여 온디맨드와 스팟 인스턴스를 결합할 수 있습니다.
- 클러스터에서 여러 노드 그룹을 사용하는 경우 Expander를 사용하여 크기 조정 결정의 우선순위를 지정합니다.



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

| 수강생용 노트

다음은 Cluster Autoscaling을 사용하기 위한 몇 가지 모범 사례입니다.

- 자동 검색 설정이 선호됩니다(--노드 그룹 자동 검색 플래그).
 - 자동 검색이 설정되면 Cluster Autoscaler(CA)는 대상 Auto Scaling 그룹의 최소값, 최대값 및 원하는 값을 자동으로 결정합니다. 또한 CA는 Auto Scaling 그룹의 원하는 값만 조정합니다. 최소값과 최대값은 변경하지 않습니다. 자동 검색의 대안은 CA에서 Auto Scaling 그룹의 최소값, 최대값 및 원하는 값을 수동으로 구성하는 것입니다.
- Auto Scaling 그룹에서 최소/최대 그룹 크기를 직접 조정합니다.
 - Cluster Autoscaler는 최신 변경 사항을 자동으로 가져옵니다.
 - Cluster Autoscaler에서 최소/최대 구성을 변경해도 Auto Scaling 그룹에는 해당 변경 사항이 적용되지 않습니다.
- MixedInstancesPolicy를 사용하여 온디맨드 인스턴스와 스팟 인스턴스를 결합할 수 있습니다. 각 인스턴스 유형은 RAM 용량 및 vCPU 수가 동일해야 합니다.
 - GitHub에 예가 나와 있습니다
(<https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/MixedInstancePolicy.md>).
- Cluster Autoscaler를 여러 개의 노드 그룹에 사용할 경우 기본 전략은 CA가 무작위로 노드를 추가할 그룹을 선택하는 것입니다. Expander를 사용하여 확장할 노드 그룹을 선택하기 위한 다양한 전략을 구성할 수 있습니다.
 - GitHub에 예가 나와 있습니다
(<https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-expanders>).

Cluster Autoscaler 모범 사례에 대해 자세히 알아보려면 EKS 모범 사례 가이드 설명서의 Kubernetes Cluster Autoscaler 모범 사례(<https://aws.github.io/eks-best-practices/cluster-autoscaling/>)를 참조하십시오.

활동

이 활동에서는 다음 과제를 수행합니다.

- Cluster Autoscaler를 사용하여 축소

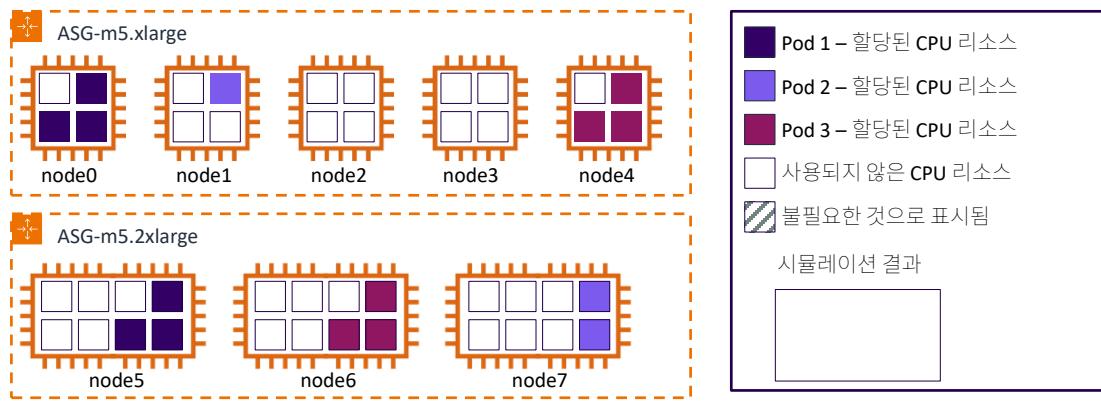


©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

이 활동에서는 Cluster Autoscaler가 현재 사용률을 기준으로 필요한 것보다 더 많은 리소스가 프로비저닝되는 시나리오를 처리하는 방법을 살펴봅니다.

활동: Cluster Autoscaler 축소(1/6)



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

~Alt text

- ~ 6개의 노드가 있는 Auto Scaling Group입니다. 이 중 2개는 사용되지 않습니다.
- ~ 3개의 노드가 있는 Auto Scaling 그룹입니다. 3개 노드 모두 사용되지 않은 컴퓨팅 용량이 있습니다.

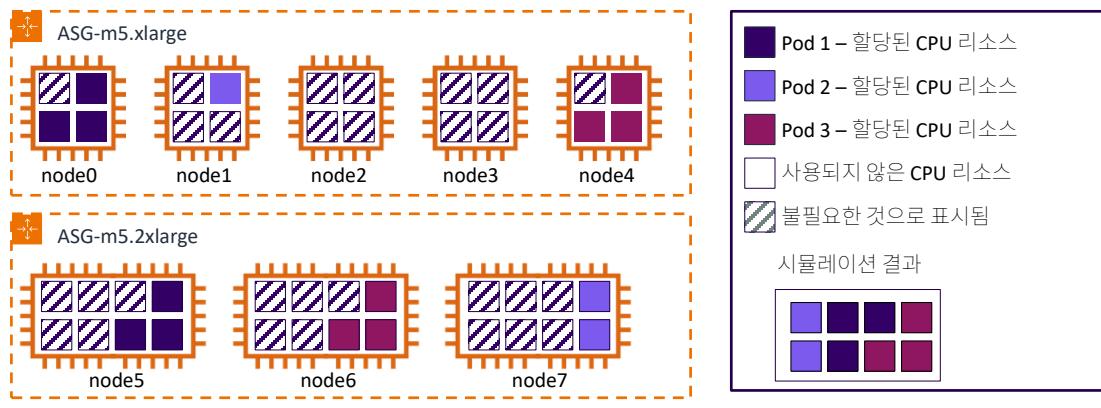
~

|수강생용 노트

이 활동에서는 Cluster Autoscaler가 현재 사용률을 기준으로 필요한 것보다 더 많은 리소스가 프로비저닝되는 시나리오를 처리하는 방법을 살펴봅니다. 특정 노드에는 미사용 용량이 있습니다. 2개의 노드가 Pod를 전혀 지원하지 않습니다. Cluster Autoscaler는 10분 간격으로 일련의 전달을 수행합니다. 효율성을 개선하기 위해 Pod를 통합하고 노드를 종료하는 방법을 결정합니다.

이 활동 전반에 걸쳐 각 Pod는 Kubernetes에서 분할할 수 없는 최소 객체이므로 Cluster Autoscaler의 지표로 사용되지만 Pod 리소스는 컨테이너 수준별로 설정된다는 점에 유념하는 것이 중요합니다.

활동: Cluster Autoscaler 축소(2/6)



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

~Alt text

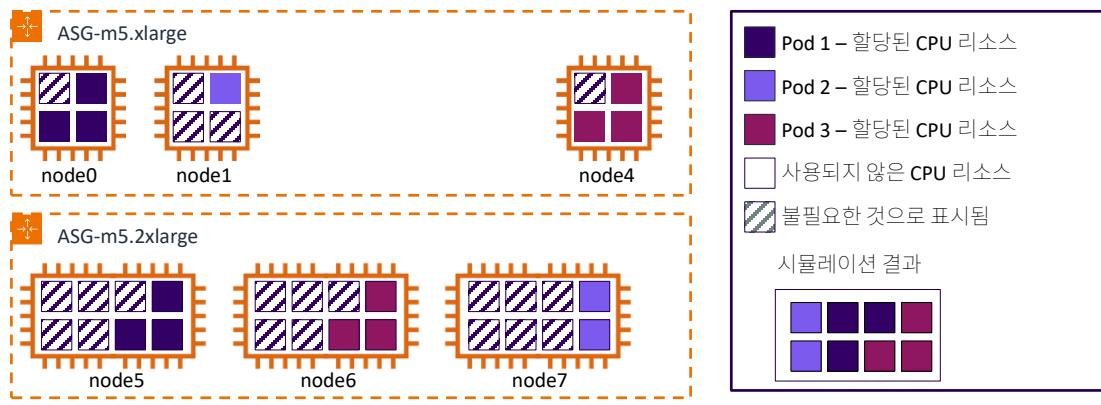
- ~ 6개의 노드가 있는 Auto Scaling Group입니다. 이 중 2개는 사용되지 않습니다.
- ~ 3개의 노드가 있는 Auto Scaling 그룹입니다. 3개 노드 모두 사용되지 않은 컴퓨팅 용량이 있습니다.

~

|수강생용 노트

Cluster Autoscaler는 첫 번째 전달에서 미사용 용량을 식별하여 불필요한 용량으로 표시합니다. 또한 시뮬레이션을 실행하여 효율성을 개선하기 위해 Pod를 통합할 수 있는 방법을 결정합니다.

활동: Cluster Autoscaler 축소(3/6)



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

~Alt text

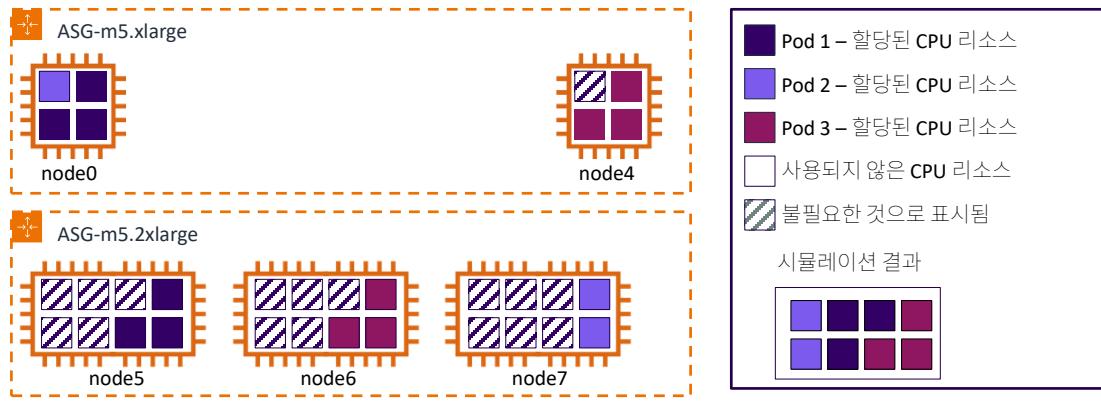
- ~ 6개의 노드가 있는 Auto Scaling Group입니다. 이 중 2개는 사용되지 않습니다.
- ~ 3개의 노드가 있는 Auto Scaling 그룹입니다. 3개 노드 모두 사용되지 않은 컴퓨팅 용량이 있습니다.

~

|수강생용 노트

다음 전달에서 Cluster Autoscaler는 어떤 Pod도 지원하지 않는 노드를 중지합니다. 인스턴스의 크기와 인스턴스에서 실행 중인 Pod의 CPU 요청으로 인해 ASG-m5.xlarge 그룹에서는 통합이 불가능합니다. CA는 빈 노드를 한 번에 최대 10개까지 종료하지만, Pod를 실행 중인 노드만 한 번에 하나씩 종료합니다.

활동: Cluster Autoscaler 축소(4/6)



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

~Alt text

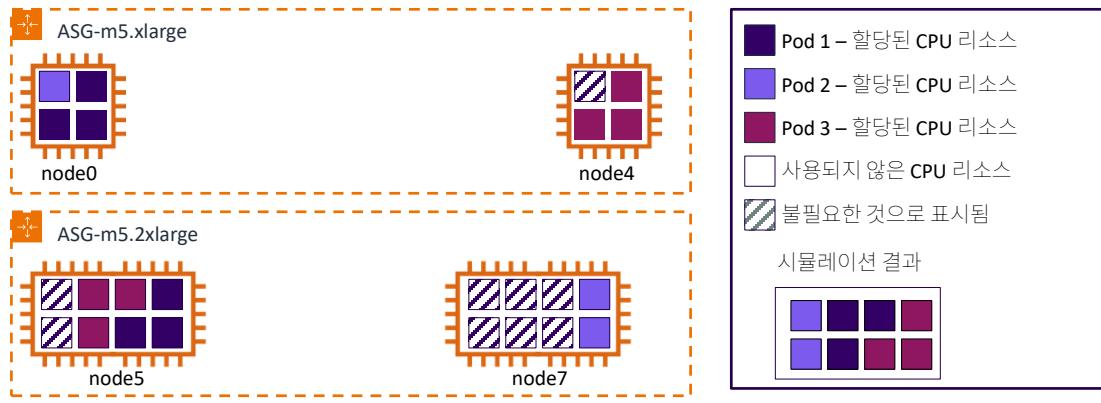
- ~ 6개의 노드가 있는 Auto Scaling Group입니다. 이 중 2개는 사용되지 않습니다.
- ~ 3개의 노드가 있는 Auto Scaling 그룹입니다. 3개 노드 모두 사용되지 않은 컴퓨팅 용량이 있습니다.

~

|수강생용 노트

다음 단계에서 Cluster Autoscaler는 node1에서 실행 중인 Pod를 제거하고 node0에서 Pod를 다시 시작한 후 node1을 종료합니다.

활동: Cluster Autoscaler 축소(5/6)



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

~Alt text

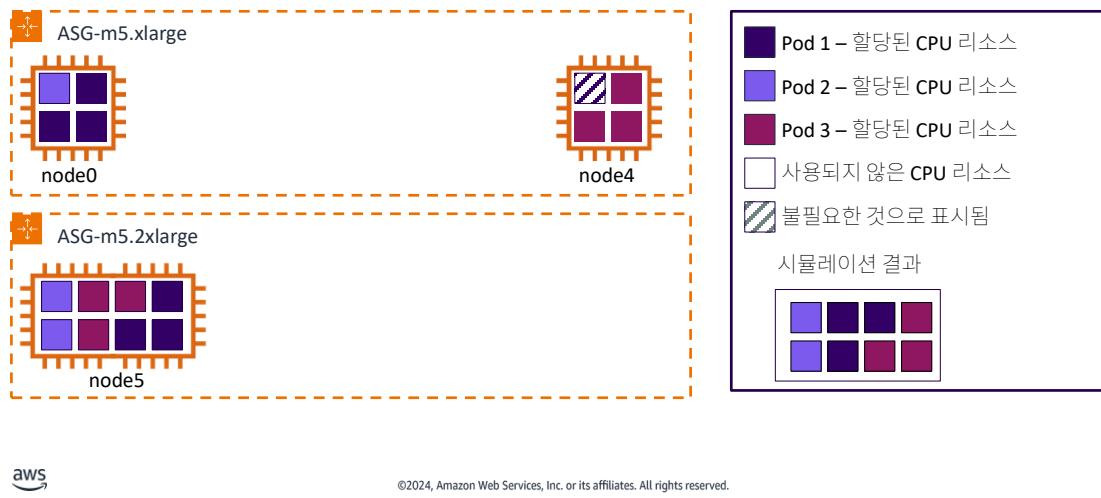
- ~ 6개의 노드가 있는 Auto Scaling Group입니다. 이 중 2개는 사용되지 않습니다.
- ~ 3개의 노드가 있는 Auto Scaling 그룹입니다. 3개 노드 모두 사용되지 않은 컴퓨팅 용량이 있습니다.

~

|수강생용 노트

다음 단계에서 Cluster Autoscaler는 node6에서 실행 중인 Pod를 제거하고 node5에서 Pod를 다시 시작한 후 node6을 종료합니다.

활동: Cluster Autoscaler 축소(6/6)



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

~Alt text

- ~ 6개의 노드가 있는 Auto Scaling Group입니다. 이 중 2개는 사용되지 않습니다.
- ~ 3개의 노드가 있는 Auto Scaling 그룹입니다. 3개 노드 모두 사용되지 않은 컴퓨팅 용량이 있습니다.

~

|수강생용 노트

다음 단계에서 Cluster Autoscaler는 node7에서 실행 중인 Pod를 제거하고 node5에서 Pod를 다시 시작한 후 node7을 종료합니다. 이는 리소스 요구 사항과 사용 가능한 용량을 고려할 때 가능한 가장 효율적인 구성입니다. 원래 실행하던 노드보다 5개 적은 수의 노드를 실행하여 비용 이점을 실현할 수 있습니다.

Karpenter

- AWS에서 관리하는 오픈 소스 Cluster Autoscaler
- 적정 규모의 노드로 Kubernetes 클러스터의 크기를 자동으로 조정
- 프로비저너 및 제약 조건을 사용하여 크기 조정 요구 사항을 충족
- Auto Scaling 그룹을 사용하지 않음



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

| 수강생용 노트

Karpenter는 애플리케이션 가용성과 클러스터 효율성을 향상하는 데 도움이 되는 오픈 소스 고성능 Kubernetes Cluster Autoscaler입니다. Karpenter는 스케줄링 불가능 Pod를 처리하기 위해 노드를 추가하고, 해당 노드에서 Pod를 스케줄링하고, 필요하지 않은 노드를 제거합니다. Karpenter는 Kubernetes 프로젝트의 Cluster Autoscaler와 유사하게 작동합니다. 그러나 Karpenter는 기존 Auto Scaling 그룹의 다른 노드를 사용하는 대신 적정 규모의 컴퓨팅 리소스를 시작합니다. 이러한 차이점 덕분에 클러스터의 노드 전체에서 효율적인 리소스 사용이 보장됩니다. 명확히 설명하면 Karpenter는 기존 노드 그룹 및 Fargate Pod와 함께 작동하지만 클러스터 크기 조정을 위해 사전 구성된 노드 그룹이 필요하지 않습니다.

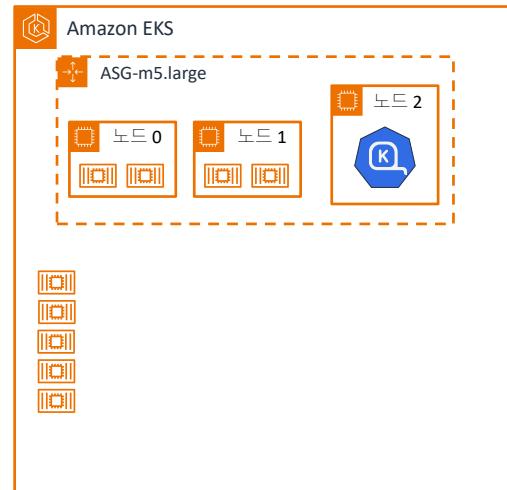
기본적으로 Karpenter는 EC2 인스턴스를 모든 유형 중에서 선택하고 노드를 임의의 가용 영역에 배포합니다. 클러스터 사용자는 프로비저너와 제약 조건을 통해 클러스터 및 애플리케이션 크기 조정 요구 사항에 맞게 Karpenter를 조정할 수 있습니다. 클러스터 관리자는 프로비저닝 작업을 지정하기 위해 프로비저너를 사용하여 Karpenter를 구성합니다. 예를 들어, 프로비저너를 설정하여 클러스터에 추가되는 EC2 인스턴스의 유형을 제한하거나 사용되지 않는 노드가 클러스터에서 제거되는 시간 제한을 설정할 수 있습니다.

애플리케이션 개발자의 경우 제약 조건은 Karpenter에게 애플리케이션의 컴퓨팅 요구 사항을 전달하는 수단을 제공합니다. 제약 조건이라는 용어는 Pod 사양에 포함된 모든 Kubernetes 스케줄링 요구 사항을 나타냅니다. 예를 들어, Pod 사양은 특정 레이블이 있는 노드에서만 실행되도록 nodeSelector를 지정할 수 있습니다. 동일한 Pod 사양에는 노드에 애플리케이션에서 사용할 수 있는 충분한 메모리 및 CPU 용량이 있는지 확인하기 위한 컴퓨팅 리소스 요청도 포함될 수 있습니다. Pod가 스케줄링 불가능인 경우 Karpenter는 요구 사항에 맞는 컴퓨팅 리소스를 생성합니다. Karpenter는 노드를 프로비저닝하기 전에 스케줄링 제약 조건을 분석한 다음 명시된 요구 사항과 일치하는 노드를 선택합니다. 지정된 제약 조건이 프로비저너의 제약 조건을 벗어나는 경우 Pod는 스케줄링되지 않은 상태로 유지됩니다.

Karpenter에 대해 자세히 알아보려면 Karpenter 설명서(<https://karpenter.sh/docs>)를 참조하십시오.

Karpenter 사용(1/4)

```
kind: Deployment
...
spec:
  replicas: 5
...
  containers:
    - name: myapp
      image: repo.corp/myapp:1.0
      resources:
        requests:
          cpu: 1
```



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

~Alt text

~ Karpenter를 실행하는 Amazon EKS 클러스터입니다. 5개 Pod가 스케줄링
불가능입니다.

~

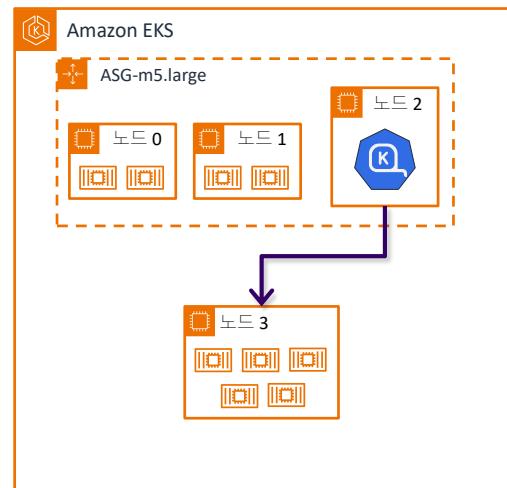
| 수강생용 노트

이 예에서는 배포 YAML이 사용 가능한 용량이 없는 Amazon EKS 클러스터에 배포됩니다. 배포에는 각각 1개의 CPU 코어를 요청하는 5개의 복제본 Pod가 필요합니다. 배포가 되면 kube-scheduler는 Pod 상태를 스케줄링 불가능으로 설정합니다. 그러므로 Karpenter는 5개의 Pod를 수용하기 위해 사용할 수 있는 최선의 Amazon EC2 인스턴스를 계산하고 새 노드를 추가한 다음 새 노드에서 실행되도록 Pod를 스케줄링합니다.

이 예에서는 프로비저너에 기본 구성이 있습니다. 이는 Karpenter가 모든 가용 영역에서 사용 가능한 최상의 EC2 인스턴스를 배포한다는 것을 의미합니다.

Karpenter 사용(2/4)

```
kind: Deployment
...
spec:
  replicas: 5
...
  containers:
    - name: myapp
      image: repo.corp/myapp:1.0
      resources:
        requests:
          cpu: 1
```



aws

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

~Alt text

~ Karpenter는 적절한 EC2 인스턴스를 시작하고 Pod를 예약합니다.

~

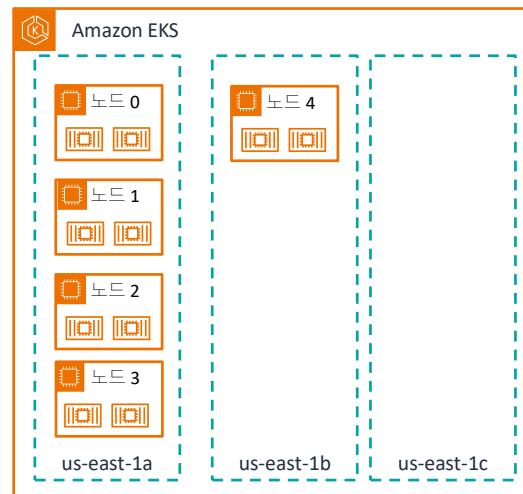
|수강생용 노트

Karpenter는 5개의 Pod를 수용하기 위해 사용할 수 있는 최선의 Amazon EC2 인스턴스를 계산하고 새 노드를 추가한 다음 새 노드에서 실행되도록 Pod를 스케줄링합니다. 이 예에서는 Karpenter가 5개의 Pod에 1개의 m5.2xlarge EC2 인스턴스 유형을 선택했습니다.

Karpenter 사용(3/4)

```
kind: Deployment
...
spec:
  replicas: 6
  ...
  containers:
    - name: myapp
      image: repo.corp/myapp:1.0
      resources:
        requests:
          cpu: 1
  topologySpreadConstraints:
    - maxSkew: 2
      topologyKey: kubernetes.io/hostname
      whenUnsatisfiable: DoNotSchedule
      ...

```



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

~Dev notes

~ Animation will break when flattening the image.

~

~Alt text

~ Karpenter가 3개의 노드를 시작하고 의도한 대로 각 노드에서 2개의 Pod가 실행되도록 스케줄링합니다.

~

|수강생용 노트

Karpenter는 사용자 지정 워크로드 요구 사항을 충족할 수 있는 유연성을 제공합니다. 이전 예에서 Karpenter는 배포를 수용하기 위해 클러스터에 1개의 대형 노드를 추가했습니다. 단일 노드 솔루션의 문제점은 단일 장애 지점입니다. 노드가 다운되면 5개 Pod를 모두 사용할 수 없기 때문입니다. 이 문제를 완화하기 위해 Karpenter는 Pod에 대한 분배 제약 조건과 같은 고급 Kubernetes 스케줄링 요구 사항을 지원합니다. 토플로지 분배 제약 조건에 대해 자세히 알아보려면 **Kubernetes** 설명서의 **Pod Topology Spread Constraints** 섹션(<https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/>)을 참조하십시오.

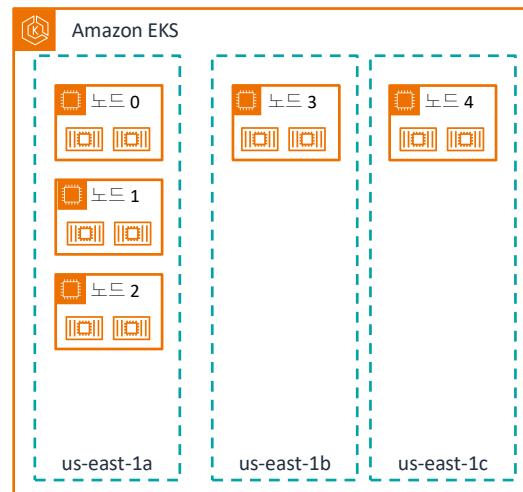
왼쪽의 배포 YAML은 6개의 복제본을 지정하고 단일 노드에 2개의 Pod를 실행하는 Pod 분배 제약 조건을 지정합니다. Karpenter는 임의의 가용 영역에 적정 규모의 노드를 3개 배포한 다음 각 노드에서 실행되도록 2개의 Pod를 스케줄링합니다.

Karpenter 사용(4/4)

```
kind: Deployment
...
spec:
  replicas: 6
  ...
  resources:
    requests:
      cpu: 1
  topologySpreadConstraints:
    - maxSkew: 2
      topologyKey: kubernetes.io/hostname
      whenUnsatisfiable: DoNotSchedule
      ...
    - maxSkew: 1
      topologyKey:
        topology.Kubernetes.io/zone
      whenUnsatisfiable: DoNotSchedule
```



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



~Dev notes

~ Animation will break when flattening the image.

~

~Alt text

~ Karpenter가 각 AZ에 1개씩 3개의 노드를 시작하고 의도한 대로 각 노드에서 2개의 Pod가 실행되도록 스케줄링합니다.

~

| 수강생용 노트

이 예에서는 Pod를 전체 가용 영역에 균등하게 분배하는 것을 포함하여 이전 예를 확장합니다. 배포 YAML은 AZ당 2개의 Pod만 스케줄링하도록 명시하는 추가 Pod 제약 조건을 제공합니다. Karpenter는 3개의 AZ에 적정 규모의 EC2 인스턴스를 추가한 다음 각 노드에서 2개의 Pod가 실행되도록 스케줄링합니다.

Karpenter가 Pod 예약을 위해 지원하는 Kubernetes 기능에는 `nodeAffinity` 및 `nodeSelector`가 포함됩니다. 또한 Karpenter는 `PodDisruptionBudget`, `topologySpreadConstraints`, Pod 간 선호도 및 비선호도를 지원합니다. Pod 배포에 제약 조건을 추가할 수 있는 다른 방법에 대해 자세히 알아보려면 Karpenter 설명서의 ‘Scheduling’(<https://karpenter.sh/docs/concepts/scheduling/>)을 참조하십시오.

Karpenter 모범 사례

- 변동이 심한 수요 또는 다양한 컴퓨팅 요구 사항에 사용
- Pod 예약에 계층화된 제약 조건을 사용
- 중요한 워크로드에 대해 **do-not-evict** 주석을 사용



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

| 수강생용 노트

Karpenter는 시기에 따라 수요가 급증하거나 컴퓨팅 요구 사항이 다양한 워크로드가 있는 Kubernetes 클러스터에 가장 적합합니다. 정적이고 일관된 워크로드는 관리형 노드 그룹과 Auto Scaling 그룹에 더 적합하며, 둘 모두 Cluster Autoscaler의 요구 사항입니다. 동일한 Kubernetes 클러스터에서 Karpenter와 Cluster Autoscaler를 모두 사용하는 것은 권장하지 않습니다. 예측할 수 없는 클러스터 크기 조정 이벤트가 발생할 수 있습니다.

Karpenter의 계층화된 제약 조건 모델을 사용하면 프로비저너와 Pod 배포 제약 조건을 복잡하게 결합하여 Pod 스케줄링을 위한 최선의 일치 기준을 구현할 수 있습니다.

Karpenter는 Pod 및 노드 선호도를 포함한 Kubernetes 고급 스케줄링 옵션을 지원합니다. 예를 들어, 계층화된 제약 조건을 사용하면 워크로드가 대상 워크로드와 동일한 가용 영역에 배포되도록 보장하여 교차 AZ 트래픽을 배제할 수 있습니다.

Pod가 여전히 실행 중인지 여부에 관계없이 생성 이후 지정된 시간(수명)이 경과하면 노드를 종료하도록 프로비저너를 구성할 수 있습니다. Karpenter 프로비저닝 노드에서 장기 실행 배치 작업 또는 스테이트풀 애플리케이션과 같은 중요한 워크로드를 실행 중인 경우 노드의 유지 시간(TTL)이 만료되어 인스턴스가 종료된다면 워크로드가 중단됩니다. Pod에 **karptener.sh/do-not-evict** 주석을 추가하면 Karpenter는 Pod가 종료되거나 **do-not-evict** 주석이 제거될 때까지 노드를 보존합니다.

Karpenter 모범 사례에 대해 자세히 알아보려면 EKS 모범 사례 가이드 설명서의 Karpenter 모범 사례(<https://aws.github.io/eks-best-practices/karpenter/>)를 참조하십시오.



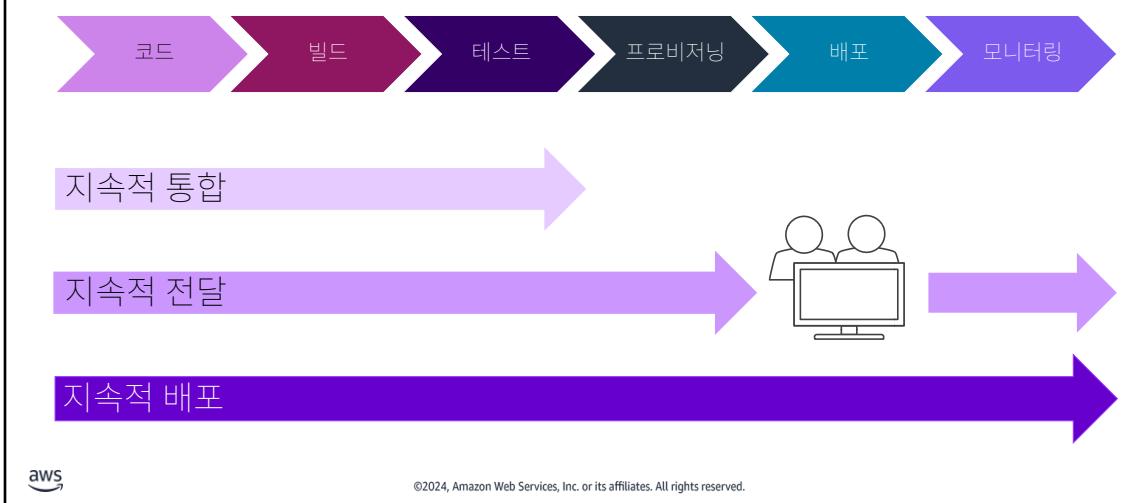
Running Containers on Amazon EKS

Amazon EKS에 지속적 배포



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

릴리스 프로세스 단계 - 지속적 통합(CI) 및 지속적 전달(CD) 파이프라인



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

~ALT text

~릴리스 프로세스의 여러 단계를 보여주는 다이어그램으로 노트에 자세히 설명되어 있습니다.

|수강생용 노트

이 슬라이드에는 **6**단계 파이프라인(코딩, 구축, 테스트, 프로비저닝, 배포, 모니터링)을 보여주는 다이어그램이 포함되어 있습니다. 화살표는 처음 **3**단계를 수행하는 지속적 통합을 나타냅니다. 두 번째 화살표는 **6**가지 단계를 모두 수행하는 지속적 전달을 나타내며 **Deployment** 전 단계는 수동 승인을 위한 단계입니다. 세 번째 화살표는 **6**가지 단계를 모두 수행하는 지속적 배포를 나타냅니다.

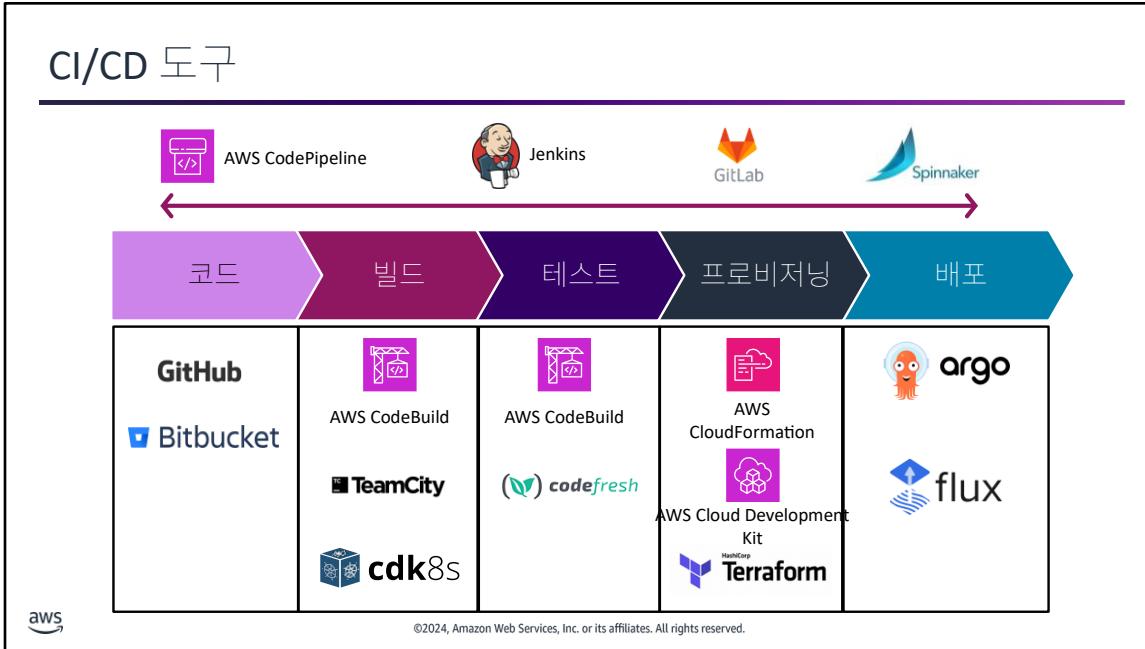
릴리스 프로세스의 단계

릴리스 프로세스의 단계는 참조 프레임으로 사용됩니다. 다음은 개발 내에서 파이프라인과 그 역할에 관한 개요입니다. 지속적 통합(CI)을 사용하면 개발자는 Git과 같은 버전 관리 시스템을 사용하여 공유 리포지토리에 커밋하는 경우가 많습니다. 각 커밋 전에 개발자는 통합하기 전에 추가 검증 계층으로 코드에서 로컬 단위 테스트를 실행할 수 있습니다. 지속적 통합 서비스는 새로운 코드 변경 사항에 대한 단위 테스트를 자동으로 구축하고 실행하여 모든 오류를 즉시 표시합니다.

지속적 전달과 지속적 배포 비교

지속적 전달과 지속적 배포의 차이점은 프로덕션 업데이트에 대한 수동 승인의 존재 여부입니다. 지속적 배포의 경우 명시적 승인 없이 프로덕션으로 자동 업데이트됩니다. 지속적 전달은 전체 소프트웨어 릴리스 프로세스를 자동화합니다. 커밋되는 모든 개정 버전은 업데이트를 빌드, 테스트, 단계별로 진행하는 자동화된 흐름을 시작합니다. 개발자는 라이브 프로덕션 환경에 배포하는 최종 결정을 시작합니다.

CI/CD 도구



~ SEQUOIA CHANGES: Removed CodeCommit

~ALT text

~Deployment 파이프라인의 각 단계에 매핑되는 CI/CD 도구의 디스플레이입니다.
자세한 내용은 노트에 있습니다.

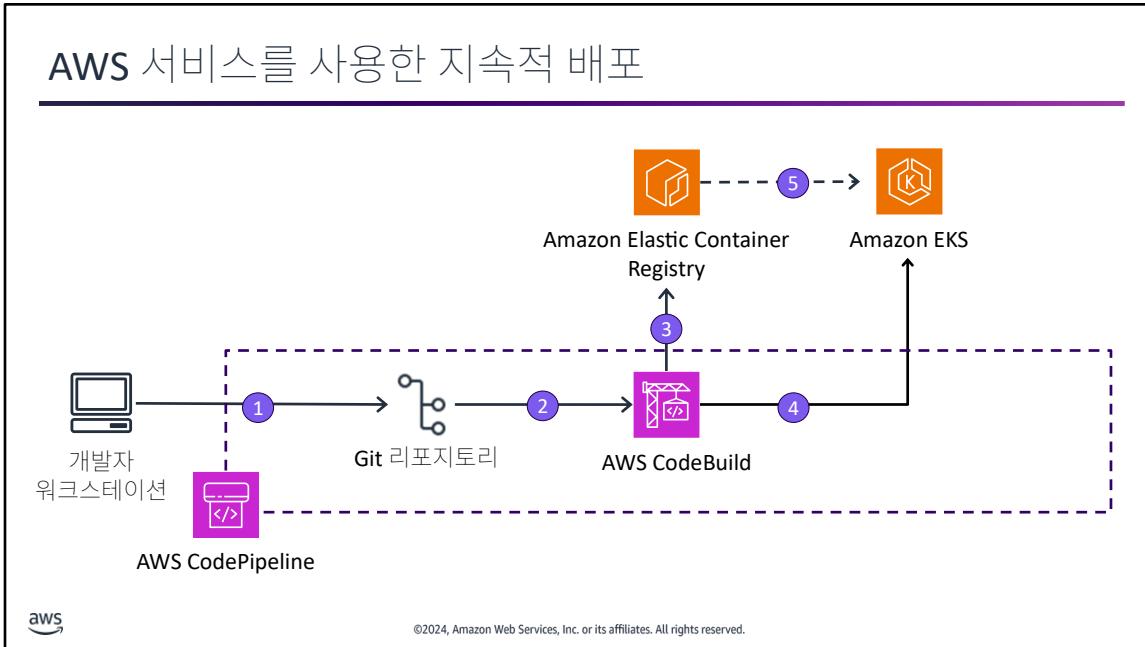
| 수강생용 노트

Kubernetes 환경에서는 DevOps 방법론과 CI/CD 파이프라인을 사용하는 것이 일반적입니다. 사용할 수 있는 다양한 도구 중에는 프로세스의 어느 한 단계에 집중하는 도구도 있고 여러 단계에 걸쳐 있는 도구도 있습니다. 다음은 일반적으로 사용되는 다른 CI/CD 제품 외에 AWS 서비스의 몇 가지 예입니다.

- **코드:** Bitbucket, GitHub
- **빌드:** AWS CodeBuild, TeamCity, CDK for Kubernetes(cdk8s)
- **테스트:** AWS CodeBuild 및 Codefresh
- **프로비저닝:** AWS CloudFormation, AWS SDK, Terraform
- **배포:** Argo CD 및 Flux

일부 소스 프로젝트 및 APN 파트너 제품에 대한 자세한 내용은 Amazon EKS 파트너 웹 사이트(<https://aws.amazon.com/eks/partners/>)를 참조하십시오.

AWS 서비스를 사용한 지속적 배포



~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository

~Alt text

~ AWS 서비스를 사용하는 CI/CD 파이프라인입니다. 자세한 내용은 노트에 있습니다.

~

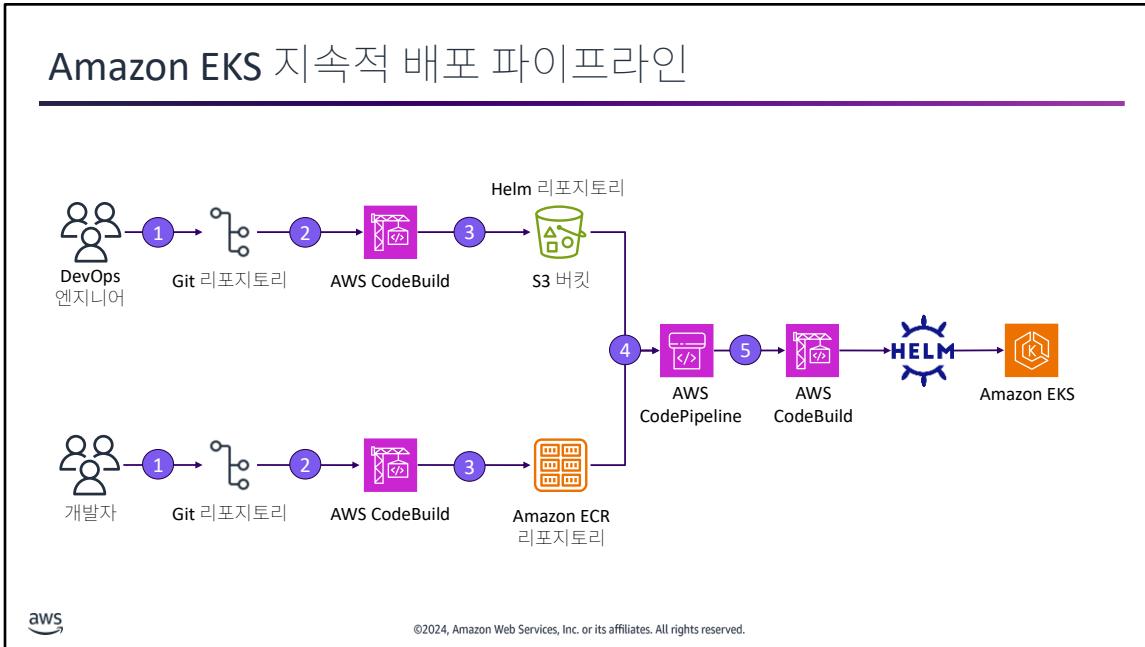
| 수강생용 노트

다음은 다른 AWS 서비스와 Amazon EKS를 통합한 예입니다. 이 예는 Kubernetes와 AWS를 함께 사용하여 컨테이너 기반 애플리케이션용 완전관리형 지속적 배포 파이프라인을 생성합니다. 이 접근 방식에서는 AWS 개발자 도구를 활용하여 소스 코드, 빌드, 파이프라인을 관리합니다.

1. 개발자는 코드를 **Git** 리포지토리로 커밋합니다.
개발자는 제안된 프로덕션 코드 변경 사항을 검토하기 위해 풀 요청을 생성합니다. 풀 요청이 **Git** 리포지토리의 기본 브랜치에 병합되면 **AWS CodePipeline**은 자동으로 브랜치에 대한 변경 사항을 감지하고 파이프라인을 통해 코드 변경 사항을 처리합니다.
2. **AWS CodeBuild**는 코드 변경 사항과 종속성을 패키징하고 컨테이너 이미지를 빌드합니다. 옵션으로 다른 파이프라인 단계도 **AWS CodeBuild**를 사용하여 코드와 패키지를 테스트합니다.
3. 빌드 단계, 테스트 단계 또는 2가지 단계 모두 성공한 후 컨테이너 이미지는 **Amazon ECR**로 푸시됩니다.
4. 배포 매니페스트 업데이트가 완료되면 **AWS CodeBuild**는 **Kubernetes API**를 호출하여 **Kubernetes** 애플리케이션 배포의 이미지를 업데이트합니다.
5. **Kubernetes**는 **Amazon ECR**에 지정된 컨테이너 이미지와 일치하도록 애플리케이션 배포에서 **Pod**의 롤링 업데이트를 수행합니다.

이제 파이프라인이 라이브 상태가 되어 **Git** 리포지토리의 기본 브랜치에 대한 변경 사항에 응답합니다. 이 파이프라인도 완전히 확장할 수 있습니다. 테스트 수행 단계를 추가하거나 코드가 프로덕션 클러스터로 전달되기 전에 스테이징 환경에 배포하는 단계를 추가할 수 있습니다.

Amazon EKS 지속적 배포 파이프라인



~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository

| 수강생용 노트

이 다이어그램은 DevOps 엔지니어가 Helm 차트 및 Kubernetes 클러스터 구성을 유지 관리하는 이중 파이프라인을 보여줍니다. 이 이중 파이프라인을 통해 애플리케이션 개발자는 애플리케이션이 실행될 인프라에 대한 걱정 없이 코드 작성에 집중할 수 있습니다.

개발자 파이프라인

1. 개발자는 애플리케이션 코드를 Git 리포지토리에 커밋하고 풀 요청을 생성하여 프로덕션 코드에 대해 제안된 변경 사항을 검토합니다.
2. AWS CodeBuild는 코드 변경 사항과 종속성을 패키징하고 테스트를 실행하고 컨테이너 이미지를 구축합니다.
3. 빌드 단계 또는 테스트 단계가 성공한 후 Docker 이미지는 Amazon ECR로 푸시됩니다. 이제 이미지를 배포의 소스로 사용할 수 있습니다.

DevOps 엔지니어 파이프라인

1. DevOps 엔지니어는 Helm 차트를 Git 리포지토리에 커밋하고 풀 요청을 생성하여 제안된 변경 사항을 검토합니다.
2. AWS CodeBuild는 차트에서 `helm lint`, `kubeval`, `kubetest` 같은 일련의 테스트를 호출합니다.
3. 테스트가 완료되면 AWS CodeBuild는 차트를 Amazon Simple Storage Service(Amazon S3) 버킷에서 호스팅되는 Helm 리포지토리로 푸시합니다.

Deployment

4. Helm 차트와 차트가 사용할 컨테이너 이미지가 준비된 후 AWS CodePipeline이 소스 코드와 구성을 AWS CodeBuild에 푸시했습니다.
5. AWS CodeBuild는 Helm을 호출하여 Deployment를 테스트하고 오류 발생 시 롤백합니다. 테스트가 완료되면 AWS CodeBuild는 `helm upgrade --install` 명령을 사용하여 애플리케이션을 배포합니다.



Running Containers on Amazon EKS

GitOps 및 Amazon EKS



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

GitOps 원칙



Git



- 단일 정보 소스로서의 Git
- 명령형 대신 선언형
- 승인된 변경 사항이 자동으로 적용됨
- 일관된 상태를 보장하는 소프트웨어 에이전트

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

| 수강생용 노트

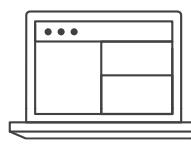
Weaveworks가 대중화한 용어인 **GitOps**는 컨테이너식 애플리케이션의 **Deployment**을 관리하는 모범 사례 중에서 엄선한 것입니다. 이러한 모범 사례는 컨테이너식 애플리케이션이 실행되는 클러스터 인프라에도 적용됩니다. 핵심 원칙은 **Git**이 사용자 환경에 대한 단일 정보 소스가 된다는 것이며, 이는 애플리케이션 **Deployment** 구성이 **Git**에 배포된다는 것을 의미합니다.

GitOps의 다른 원칙은 다음과 같습니다.

- 모든 것이 소스 제어에 있음 - 인프라 및 애플리케이션 구성이 전적으로 소스 제어에서 표현됩니다. 소스 제어는 원하는 시스템 상태에 대한 단일 정보 소스입니다.
- 명령적이기보다 선언적 - 설치 명령을 실행하는 대신 원하는 인프라 및 애플리케이션 상태를 설명합니다. 구성에 대한 선언적 정의는 일련의 설치 지침보다 비교하고 이해하고 설명하기 쉽습니다.
- 승인된 변경 사항을 자동으로 적용 - 원하는 상태로 지속적으로 수렴합니다. 자동 복구 및 자동화된 배포를 촉진하는 도구를 수용합니다.
- 일관된 상태를 보장하기 위한 소프트웨어 에이전트 - **Argo** 또는 **Flux**와 같은 소프트웨어 에이전트는 클러스터에서 실행 중인 애플리케이션의 원하는 상태가 **Git** 리포지토리에 저장된 것과 일치하는지 확인하기 위해 자동으로 수정 조치를 취할 수 있습니다.

GitOps에 대한 자세한 내용은 Weaveworks 웹 사이트의 'What is GitOps' (<https://www.weave.works/blog/what-is-gitops-really>)를 참조하십시오.

Git은 전체 변경 내역을 추적



누가

무엇을

언제

왜

검토



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

| 수강생용 노트

Git은 훌륭한 단일 정보 소스입니다. 인프라 및 애플리케이션 코드에 대한 전체 변경 내역을 추적합니다. 이 기능은 훌륭한 감사 및 보고 도구입니다. 풀 요청은 모든 변경 사항에 대해 동료 검토를 시행한다는 의미입니다.

하나의 진입점에서 하나의 파이프라인으로 변경 사항을 강제로 적용하면 변경 사항을 제어하기가 더 쉽습니다. 승인된 풀 요청과 필수 보안 테스트를 요구하면 파이프라인이 유일한 통로인 **DevSecOps** 환경이 조성됩니다.

개발자가 이미 **Git**을 사용하고 있음

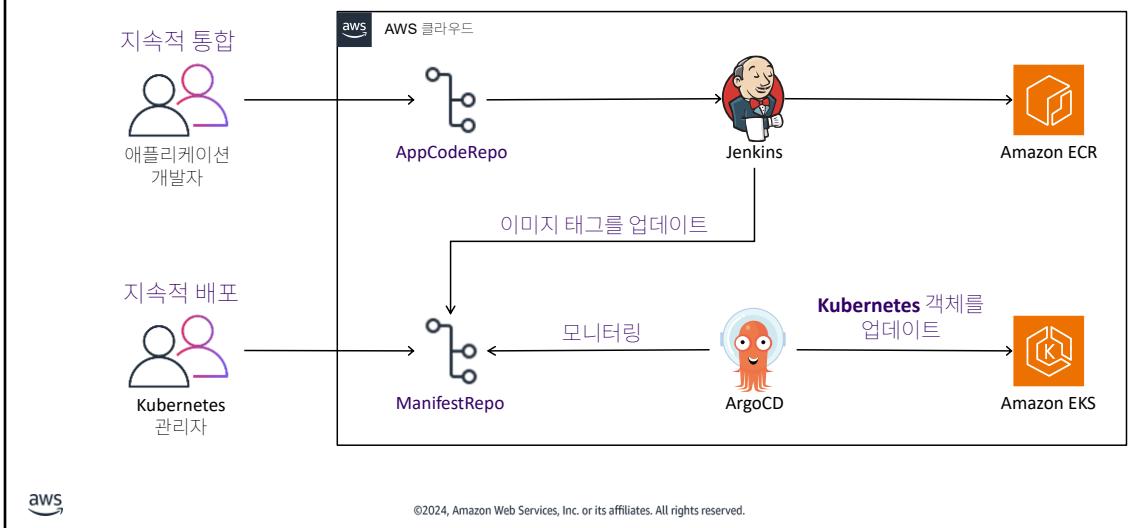


©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

별도의 도구와 워크플로를 사용하는 개발 및 운영(**DevOps**) 대신 **GitOps**를 사용하면 개발 및 운영이 원활하게 함께 작동하는데 도움이 됩니다.

GitOps 예: Amazon EKS 및 Argo CD



~ SEQUOIA CHANGES: Replaced CodeCommit icon with generic Git icon

| 수강생용 노트

이 예는 이 모듈 앞부분에서 설명한 **Amazon EKS** 지속적 배포 파이프라인과 유사합니다. 주요 차이점은 명령적이 아닌 선언적이라는 것입니다. 일련의 명령을 실행하는 대신 **GitOps** 환경은 **Git** 기반 리포지토리에서 애플리케이션 및 지원 코드형 인프라의 원하는 상태를 유지합니다.

Amazon EKS 클러스터에서 실행되는 애플리케이션의 상태 또는 **Amazon EKS** 클러스터의 상태가 **Git**에서 원하는 상태와 다르다고 가정해 보겠습니다. 이 상태를 흔히 ‘구성 드리프트’라고 합니다. 일반적으로 이를 수정하려면 수동 작업이 필요합니다. 하지만 **GitOps** 환경에서는 구성 드리프트가 자동으로 수정됩니다. 예를 들어 애플리케이션 버전이 **Git**의 정식 버전과 동기화되지 않은 경우 배포가 자동으로 실행되어 올바른 버전을 배포합니다. 이 배포는 개발자가 **Git**에서 지정한 내용에 따라 새 버전으로의 업그레이드 또는 이전 버전으로의 롤백일 수 있습니다. 배포 유형과 무관하게 전체 프로세스가 자동화됩니다.

이 예에서는 **Argo**를 사용합니다. **Argo**는 **Git**를 지속적으로 모니터링하여 변경 사항을 감지합니다. 변경 사항이 발견되면 **ArgoCD**는 **Amazon EKS** 클러스터에서 **Kubernetes** 객체를 업데이트합니다.

GitOps 프로세스는 기존 CI/CD 파이프라인에 비해 몇 가지 이점이 있습니다.

- 보안이 더 뛰어납니다.
클러스터에서 실행되는 **Kubernetes** 컨트롤러(**Argo**)가 배포를 처리하기 때문입니다. 클러스터에 컨트롤러가 있으면 CI 도구에 클러스터 API 자격 증명을 제공할 필요가 없습니다.
- 문제로부터 복구하기가 더 쉽습니다.
클러스터의 일부 또는 전부에 장애가 발생하거나 공격자 또는 나쁜 결정으로 인해 손상된다면 어떻게 됩니까? 클러스터의 올바른 상태가 **Git**에 저장되므로 기존 파이프라인보다 복구하기가 더 쉽습니다.

이 슬라이드의 예는 **2단계** 파이프라인을 보여줍니다. 애플리케이션 개발자는 새 버전의 애플리케이션을 개발하면 코드를 **Git** 리포지토리에 체크인하고 이미지에 태그를 지정합니다. 그러면 단위 테스트를 실행하는 **Jenkins** 파이프라인이 트리거되고, 테스트가 성공하면 이미지를 **Amazon ECR**에 푸시하고, **Kubernetes** 관리자에게 알립니다. 또한 **Jenkins**는 이미지 태그를 **Kubernetes** 관리자가 매니페스트를 유지 관리하는 두 번째 **Git** 프로덕션 준비 리포지토리로 업데이트합니다. **ArgoCD**는 **Kubernetes** 매니페스트 **CodeCommit** 리포지토리를 지속적으로 모니터링하여 변경 사항을 찾습니다. 변경 사항이 발견되면 **ArgoCD**는 **Kubernetes API** 서버를 업데이트합니다. 그러면 새 버전을 클러스터에 배포하는 프로세스가 시작됩니다.

이 예에서는 **Jenkins**와 **ArgoCD**를 사용하지만 **Jenkins** 대신 **AWS CodePipeline**, **ArgoCD** 대신 **Flux**와 같이 다른 도구를 사용하여 유사한 워크플로를 구축할 수 있습니다. 구현 세부 사항은 다를 수 있지만 동일한 개념이 적용됩니다. 이 모듈을 마치고 실습에서 사용하게 되므로 이 예에서는 **Jenkins**와 **ArgoCD**를 선택합니다.

활동



aws

이 활동에서는 다음 과제를 수행합니다.

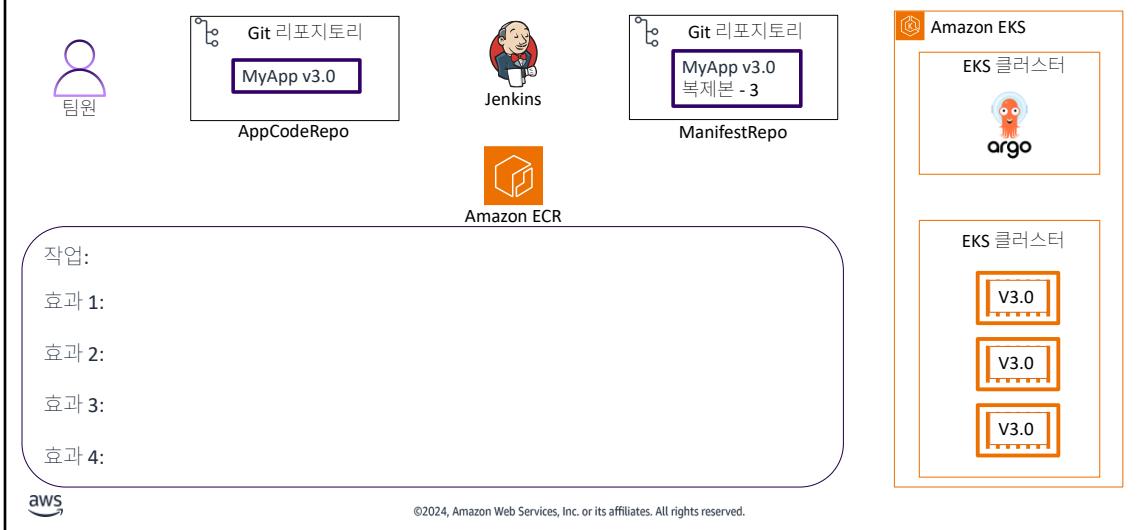
- GitOps 환경에서 시나리오 검토
- 수행한 작업의 효과를 설명

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

| 수강생용 노트

이 활동에서는 GitOps 환경에서 3가지 시나리오를 검토합니다. 수행한 작업의 효과를 설명합니다.

시나리오 1: 애플리케이션 코드 변경



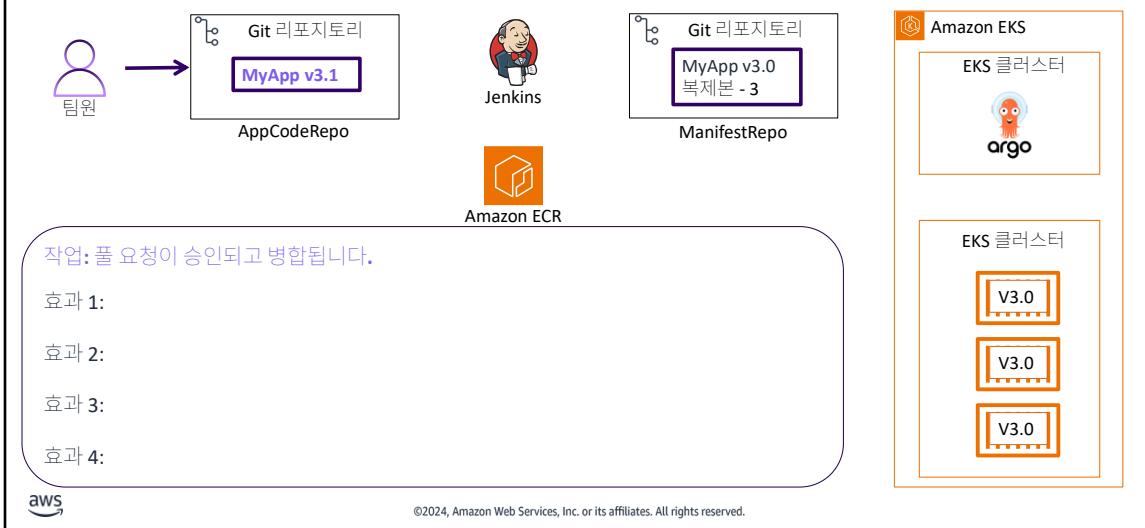
~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

이 다이어그램은 이 활동을 위한 설정을 보여줍니다. 여기에는 2개의 AWS CodeCommit 리포지토리가 있습니다. AppCodeRepo라는 리포지토리에는 애플리케이션 코드와 Jenkinsfile이 저장됩니다. ManifestRepo라는 두 번째 리포지토리에는 Kubernetes 매니페스트가 저장합니다. 현재 두 리포지토리 모두 'MyApp v3.0'에 대한 매니페스트가 있습니다. Jenkins와 Amazon ECR은 나중에 사용됩니다. 2개의 Amazon EKS 클러스터가 있습니다. 한 클러스터에는 Argo CD가 있고 다른 클러스터에는 V3.0으로 레이블이 지정된 컨테이너가 3개 있습니다.

이제 애플리케이션 코드를 변경하면 어떤 일이 발생하는지 살펴보겠습니다.

시나리오 1: 애플리케이션 코드 변경(1/5)

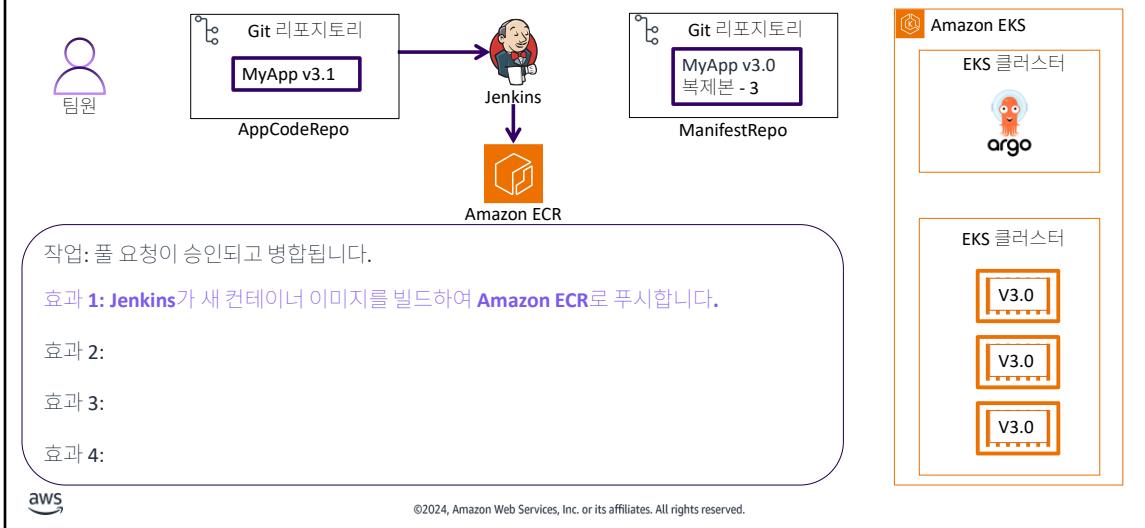


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

1단계에서는 한 팀원이 AppCodeRepo에서 매니페스트를 업데이트합니다. 풀 요청이 승인되고 병합되면 매니페스트에 'MyApp v3.1'이라는 레이블이 지정됩니다.

시나리오 1: 애플리케이션 코드 변경(2/5)

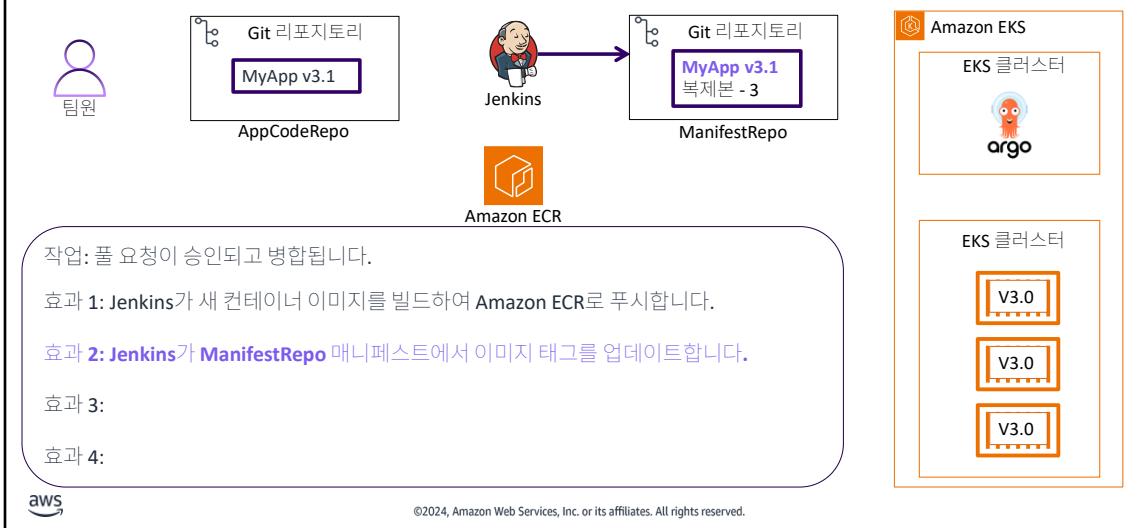


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

이제 Jenkins가 작업을 실행하고 새 컨테이너 이미지를 빌드하여 Amazon ECR로 푸시합니다.

시나리오 1: 애플리케이션 코드 변경(3/5)

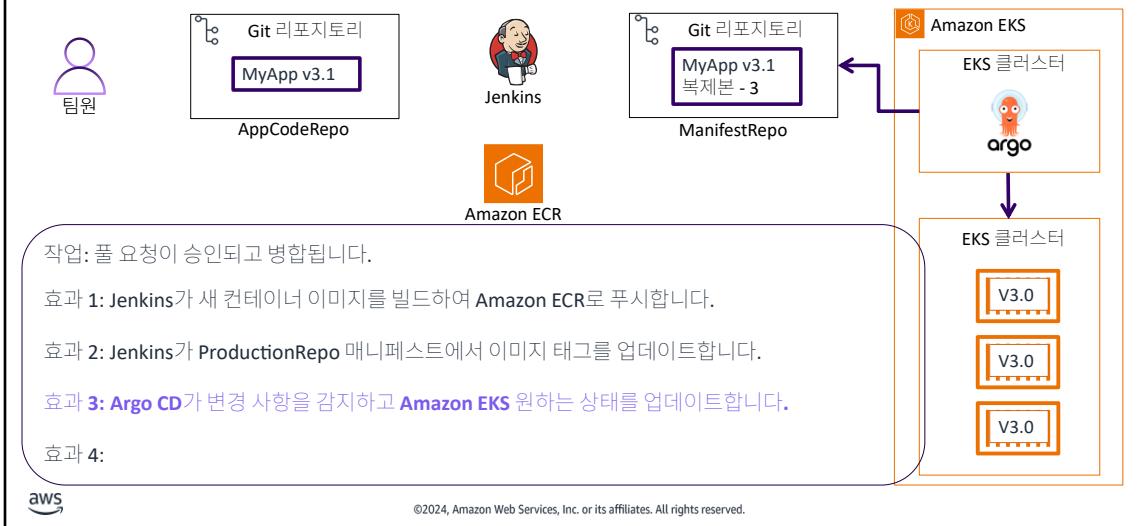


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

그런 다음 Jenkins가 ManifestRepo 매니페스트에서 이미지 태그를 업데이트합니다.
이제 ManifestRepo의 매니페스트는 'MyApp v3.1'로 표시됩니다.

시나리오 1: 애플리케이션 코드 변경(4/5)

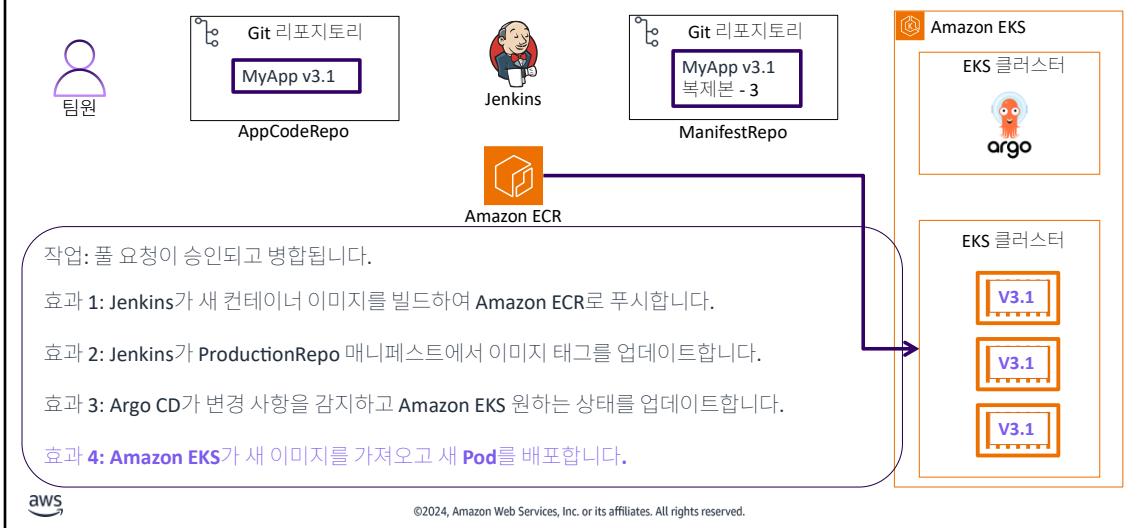


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

Argo CD는 모니터링을 통해 ManifestRepo 매니페스트에 대한 변경 사항을 감지하고 Amazon EKS 원하는 상태를 업데이트합니다.

시나리오 1: 애플리케이션 코드 변경(5/5)

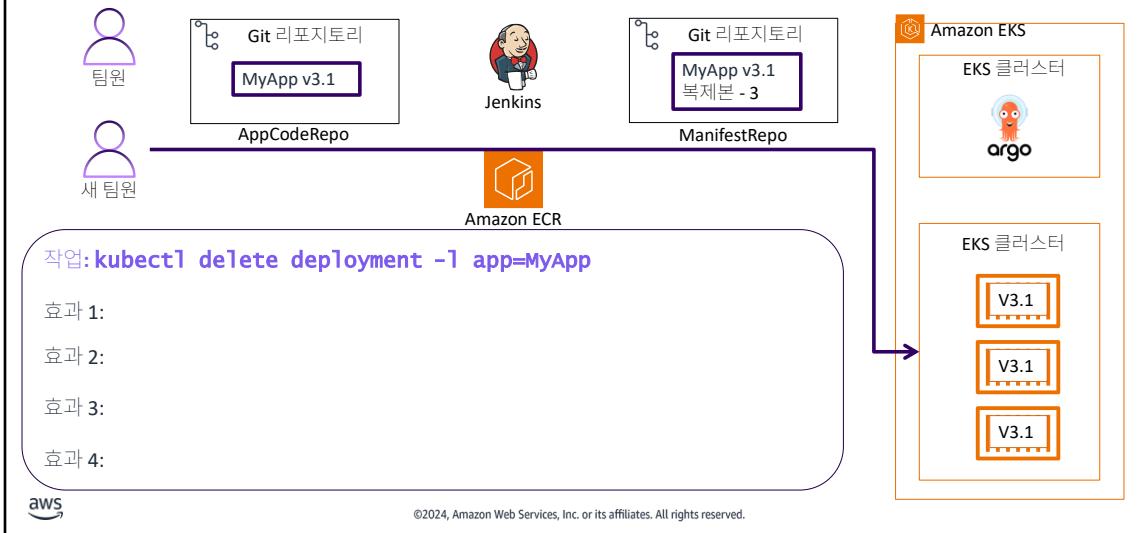


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

이제 Amazon EKS가 Amazon ECR에서 새 이미지를 가져와 새 Pod를 배포합니다. EKS 클러스터의 컨테이너 3개는 이제 v3.1로 표시됩니다.

시나리오 2: 무단 구성 변경(1/5)



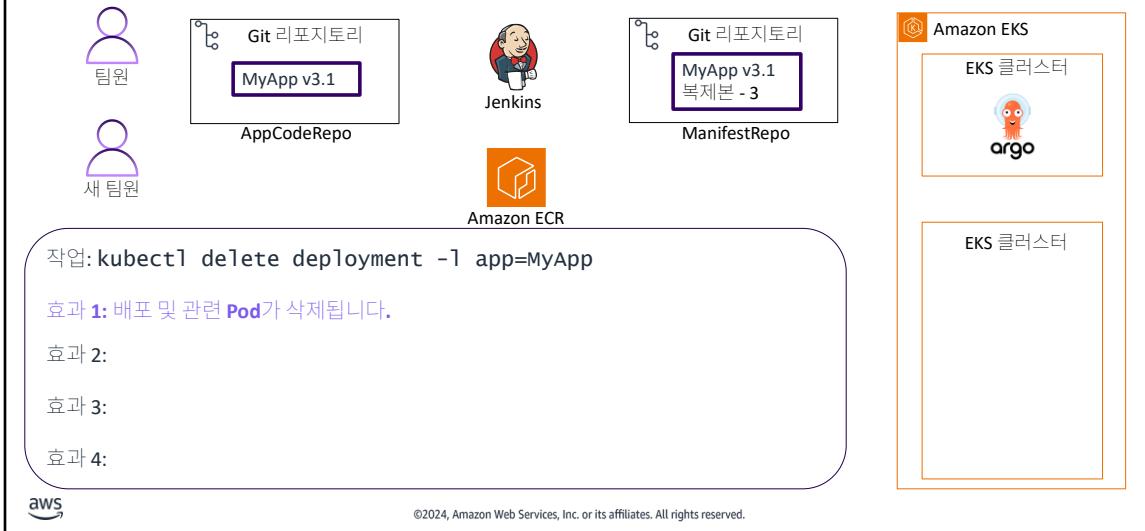
~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

이 다이어그램에는 마지막 활동의 끝부분과 동일한 설정이 표시되어 있습니다. 새 팀원이 EKS 클러스터에 다음 작업을 수행하는 것으로 표시됩니다.

Kubectly delete deployment -1 app=MyApp

시나리오 2: 무단 구성 변경(2/5)

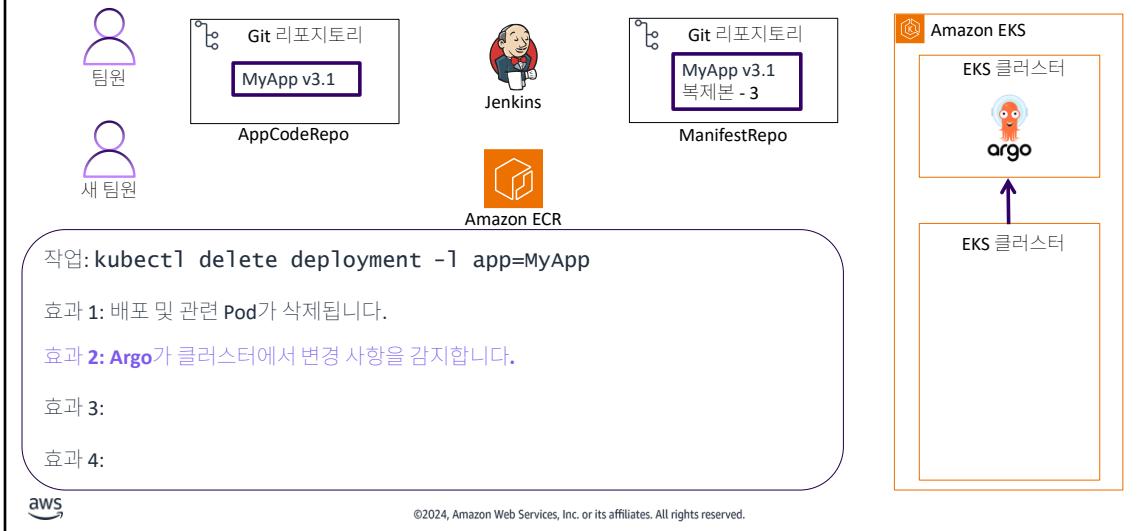


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

|수강생용 노트

이제 배포 및 관련 Pod가 삭제되었습니다. EKS 클러스터는 비어 있는 것으로 표시됩니다.

시나리오 2: 무단 구성 변경(3/5)

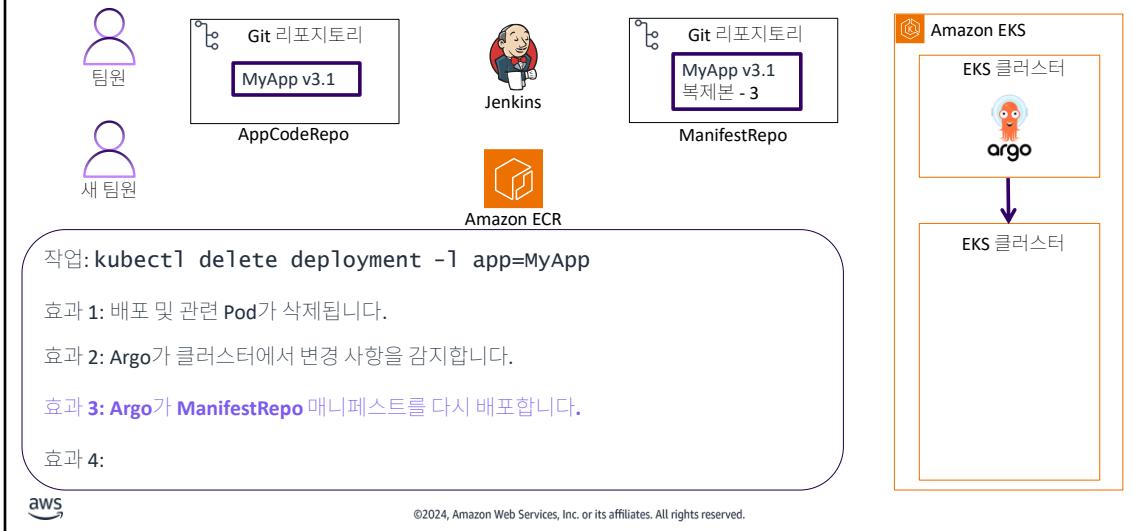


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

Argo는 EKS 클러스터를 모니터링하는 동안 클러스터에 대한 변경 사항을 감지합니다.

시나리오 2: 무단 구성 변경(4/5)

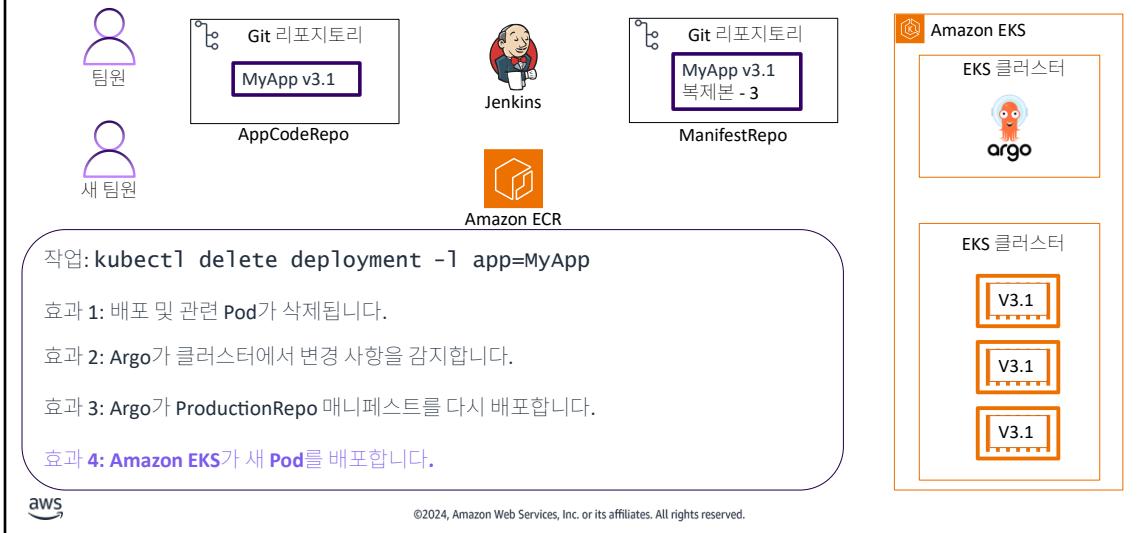


~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

Argo CD가 ManifestRepo를 다시 배포합니다.

시나리오 2: 무단 구성 변경(5/5)



~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content and notes

| 수강생용 노트

Amazon EKS는 새 Pod를 EKS 클러스터에 배포합니다. 3개의 컨테이너가 클러스터 내에 표시되고 모두 v3.1을 실행합니다.

모듈 요약



이 모듈에서 학습한 내용:

- 애플리케이션 배포를 위한 워크플로는 환경이 성장함에 따라 크기를 조정할 수 있어야 합니다.
- 지속적 통합과 지속적 배포는 밀접하게 연결되어 있지만 별개의 프로세스입니다.
- **GitOps** 방법론 챕터는 클러스터 관리를 간소화 및 자동화합니다.

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

| 강사용 노트

| 수강생용 노트

이 슬라이드는 이 강의의 주요 사항을 요약한 것입니다.



Running Containers on Amazon EKS

실습 3: 지속적 배포 및 GitOps



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

~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide notes

| 강사용 노트

| 이 실습을 완료하는 데는 약 **1시간 30분** 정도 걸립니다.

| 수강생용 노트

실습 3에서는 2개의 파이프라인을 생성합니다. 한 파이프라인은 **Git** 리포지토리, **Jenkins** 및 **Amazon ECR**를 사용하여 컨테이너 이미지를 빌드하고 저장합니다. 또 다른 파이프라인은 다른 **Git** 리포지토리, **Argo CD** 및 **Amazon EKS**를 사용하여 컨테이너 이미지를 **Amazon EKS** 클러스터에 지속적으로 배포합니다.

실습 3



aws

애플리케이션 배포

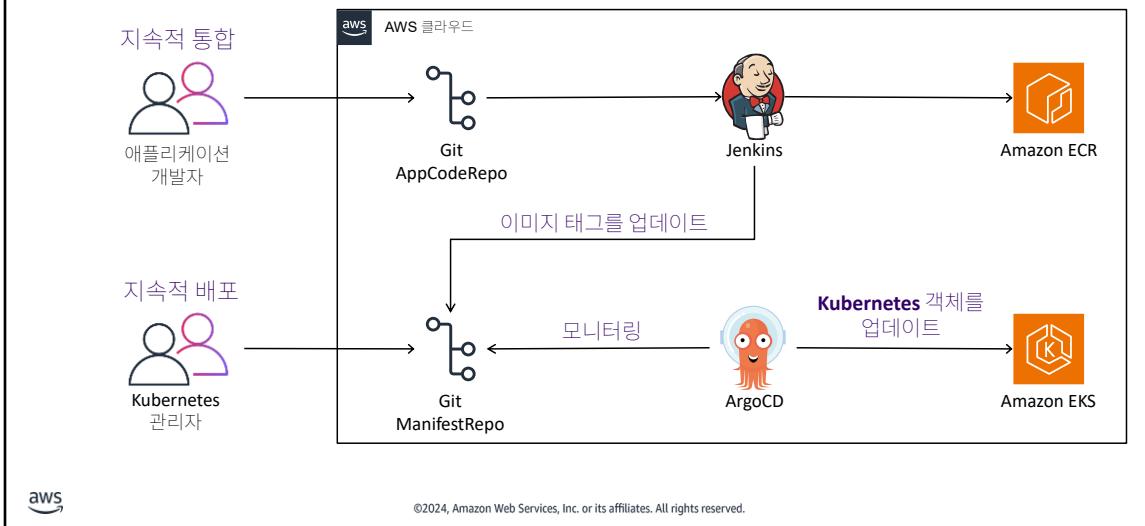
이 실습에서는 다음 과제를 수행합니다.

- **GitOps** 모범 사례에 따라 애플리케이션 및 매니페스트 구성을 저장하기 위해 **Git** 리포지토리를 생성 및 사용합니다.
- **Jenkins**를 사용하여 지속적 통합 파이프라인을 구성합니다.
- **Argo CD**를 배포합니다.
- **Argo CD**를 사용하여 지속적 배포 파이프라인을 구성합니다.

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

~ SEQUOIA CHANGES: Replaced CodeCommit with Git repository in slide content

실습 3 아키텍처 다이어그램



~ SEQUOIA CHANGES: Replaced CodeCommit with Git in slide content



감사합니다.



수정 사항이나 피드백 또는 기타 질문이 있으십니까?

<https://support.aws.amazon.com/#/contacts/aws-training>에서 문의해 주십시오. 모든 상표는 해당 소유자의 자산입니다.

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

| 강사용 노트

| 수강생용 노트