



Running Containers on Amazon EKS

모듈 1:

Kubernetes 기본 개념



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

| 강사용 노트

| 이 모듈을 완료하는 데는 약 **1시간 40분** 정도 걸립니다. 수강생은 이 모듈 중간에 휴식 시간을 갖는 것이 좋습니다. 또한 모듈을 마치고 점심 식사를 하는 것이 좋습니다.



Running Containers on Amazon EKS

모듈 1 개요

- 컨테이너의 이점
- 컨테이너 오케스트레이션
- **Kubernetes** 내부
- **Pod** 스케줄링
- **kubectl** 유ти리티(데모)
- **Kubernetes** 객체
- 지식 확인 및 활동



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



Running Containers on Amazon EKS

컨테이너의 이점



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

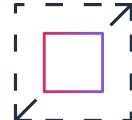
애플리케이션 구축 모범 사례



이동 가능성



클라우드 배포 가능



확장성



지속적 배포



선언형 형식



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

| 수강생용 노트

Amazon Elastic Kubernetes Service(Amazon EKS)에 대해 배우기 전에 이 과정을 수강하게 된 이유를 생각해 보십시오. 아마도 가능한 최선의 방법으로 애플리케이션 및 서비스를 구축, 배포 또는 지원하기를 바라고 이를 위해 **Amazon EKS**를 사용하고 싶을 것입니다. 그러나 컨테이너와 **Amazon EKS**가 애플리케이션의 실행과 관리를 위한 훌륭한 도구인 이유를 알아보기 전에 먼저 몇 가지 일반적인 애플리케이션 구축 모범 사례를 이해해야 합니다.

12 Factor App은 모범 사례를 사용하여 애플리케이션을 구축하는 방법을 다룹니다. 이 방법론은 **Heroku** 플랫폼에서 작업한 숙련된 개발자들이 고안했습니다.

12 Factor App은 다음과 같이 애플리케이션을 구축해야 한다고 명시합니다.

- 프로젝트에 참여하는 신규 개발자의 시간과 비용을 최소화하기 위해 설치 자동화에 선언형을 사용합니다.
- 기본 운영 체제와의 명확한 계약을 통해 실행 환경 간 이동 가능성을 극대화합니다.
- 최신 클라우드 플랫폼에 배포하는 데 적합하여 서버와 시스템을 관리할 필요가 없습니다.
- 도구, 아키텍처 또는 개발 실무에 큰 변화 없이 확장할 수 있습니다.
- 개발 환경과 프로덕션 환경의 차이를 최소화하여 민첩성 극대화를 위한 지속적 배포가 가능합니다.

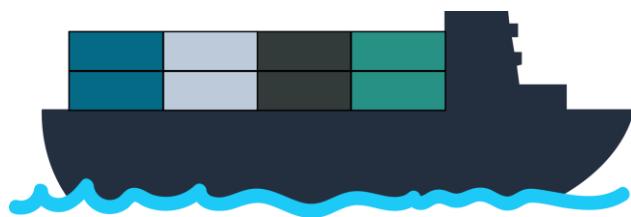
이 과정에서는 다섯 가지 모범 사례를 사용하고 이 방법론을 토대로 애플리케이션을 생성하는 데 **Amazon EKS** 사용이 도움이 되는 이유에 대해 알아봅니다.

12 Factor App은 이 방법론을 달성하는데 필요한 12가지 요소를 제시합니다. **12 Factor App**에 대한 설명은 ‘**12 Factor App**’(<https://12factor.net/>)을 참조하십시오.

이점: 이동 가능성



이동 가능성



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

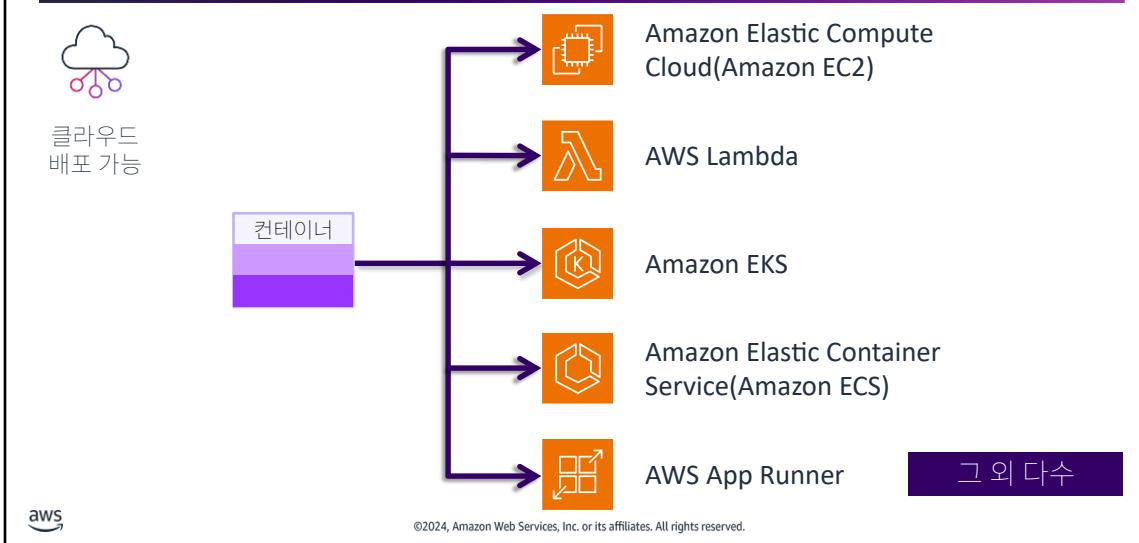
| 수강생용 노트

12 Factor App의 측면에서 컨테이너의 이점을 생각해 봅시다.

실행 환경 간의 이동 가능성

컨테이너는 이동 가능한 애플리케이션 환경입니다. 컨테이너화 플랫폼을 지원하는 모든 운영 체제(**OS**)에서 컨테이너를 사용할 수 있습니다. 또한 서로 다른 종속성과 서로 다른 라이브러리를 가진 여러 애플리케이션 버전을 동시에 실행할 수도 있습니다. 이는 컨테이너를 전혀 변경할 필요 없이 트럭, 기차, 선박, 기타 운송 방식에서 일관되게 운송하는 것과 유사합니다.

이점: 클라우드 배포 가능



~Dev notes

~Alt text: 컨테이너 이미지: AWS 서비스와 통합되는 컨테이너입니다.

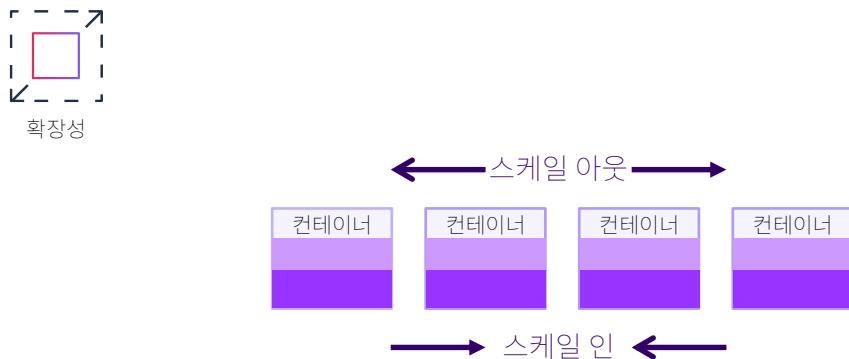
| 수강생용 노트

클라우드 배포 가능

컨테이너는 가벼울 뿐 아니라 애플리케이션을 실행하고 어디서든 조정할 수 있는 일관되고 이동 가능한 소프트웨어 환경을 제공합니다. 그러므로 컨테이너는 최신 클라우드 플랫폼에서 실행하기에 적합합니다. 예시 플랫폼은 조직의 필요 및 요구 사항을 충족하는 방식으로 컨테이너식 애플리케이션을 실행할 수 있습니다. 이 과정의 초점은 **Amazon EKS**이지만, 이 과정에서 제공하는 많은 방법론 및 컨테이너 모범 사례는 다른 클라우드 플랫폼에도 적용할 수 있습니다.

AWS 컨테이너 서비스는 몇 가지 더 있습니다. 보다 상세한 목록은 ‘AWS의 컨테이너’(<https://aws.amazon.com/containers/>)를 참조하십시오.

이점: 확장성



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

~ALT text

~컨테이너 1: 수요 변화에 따라 컨테이너 수가 확장 또는 축소됩니다.

~스케일 아웃: 확장

~스케일 인: 축소

| 강사용 노트 및 애니메이션

|<1> 컨테이너 수가 4개로 확장되고 ‘Scale out’ 텍스트가 나타납니다.

|<2> 컨테이너 수가 3개로 축소되고 ‘Scale in’ 텍스트가 나타나면서 ‘Scale out’ 텍스트가 사라집니다.

| 수강생용 노트

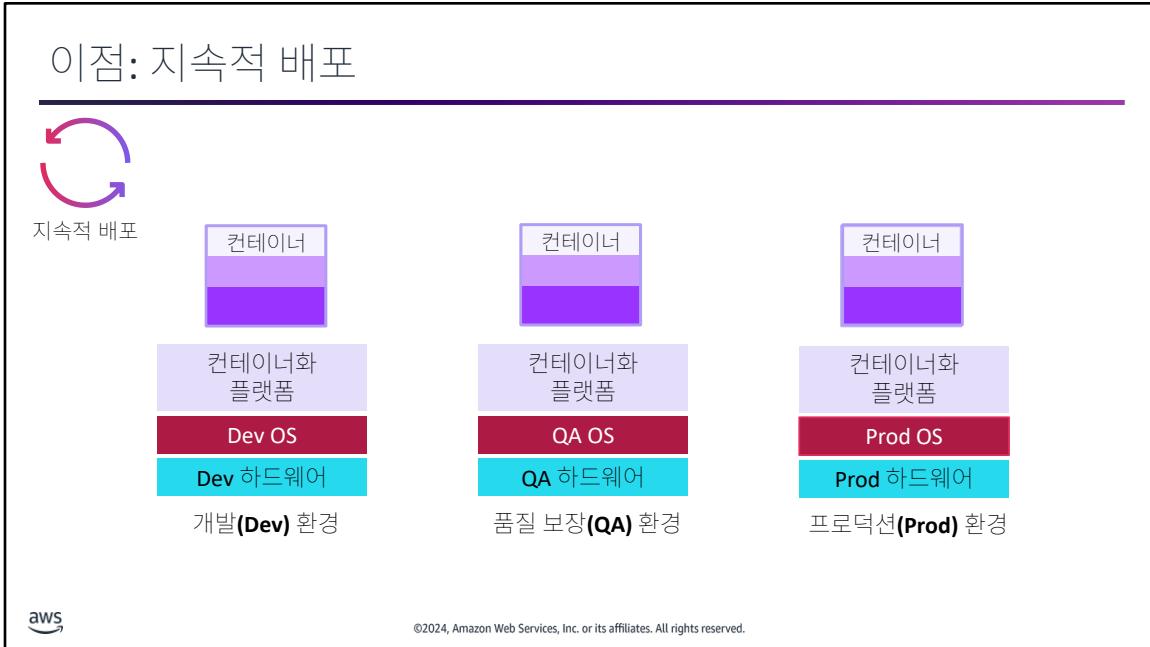
도구, 아키텍처 또는 개발 실무에 큰 변화 없이 확장이 가능합니다.

컨테이너는 크기 조정에 도움이 되며 일관되게 크기가 조정됩니다.

- 원하는 만큼 컨테이너 수를 확장할 수 있습니다.
- 컨테이너를 축소하려면 제거하면 됩니다.

컨테이너는 매우 가볍고 민첩하며 빠르므로 신속하게 효율적으로 시작하고 크기를 조정할 수 있습니다. 또한 리소스 활용도와 효율성도 뛰어나므로 하나의 서버에서 더 많은 컨테이너 크기를 조정할 수 있습니다.

이점: 지속적 배포



| 수강생용 노트

지속적인 배포성

- 지속적 배포는 애플리케이션을 자주 자동 배포하는 방법입니다.
- 컨테이너는 자동화된 빌드, 테스트, 배포 파이프라인과 통합됩니다. 또한 컨테이너는 민첩하고 가볍고 빠르기 때문에 배포와 개발 주기가 빨라집니다.

개발과 프로덕션 간의 차이 최소화

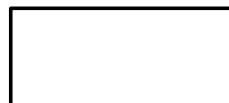
컨테이너의 내용은 컨테이너화 플랫폼을 지원하는 모든 **OS**에서 동일합니다. 동일하기 때문에 개발 환경과 프로덕션 환경의 컨테이너에서 동일한 성능과 동작을 기대할 수 있습니다.

이 주제에서는 지속적 배포에 대해 설명합니다. 이 과정의 후반부에서 지속적 통합(**CI**) 및 지속적 전달(**CD**)을 비롯하여 지속적 배포에 대해 자세히 알아보겠습니다. 차이점과 이점에 대해서도 알아봅니다.

선언형 형식

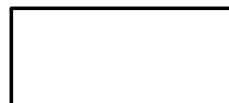
명령형 형식

```
position1 = new Point(100,100);
position2 = new Point(200,100);
position3 = new Point(200,400);
position4 = new Point(100,400);
graphics.setColor(Color.BLACK);
graphics.drawLine(position1, position2);
graphics.drawLine(position2, position3);
graphics.drawLine(position3, position4);
graphics.drawLine(position4, position1);
```



선언형 형식

```
Rectangle:
  position: Point(100,100)
  width: 300
  height: 100
  borderColor: Black
```



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

| 강사용 노트 및 애니메이션

| <1> 클릭하면 선언형 형식이 나타납니다.

| 수강생용 노트

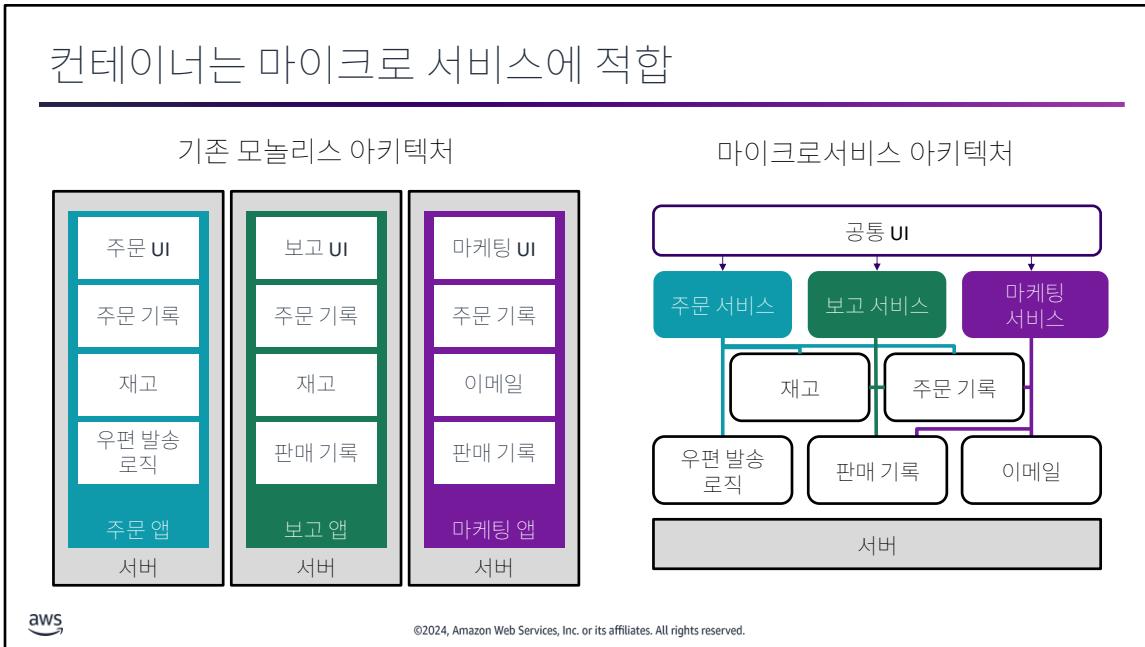
선언형 형식은 결과를 얻는 방법(명령형 형식) 대신, 결과로 원하는 대상을 코딩하는 프로그래밍 방식입니다. 선언형 프로그래밍 언어는 결과로 원하는 대상을 살펴보고, 로직을 구현합니다.

직사각형을 그리는 방법으로 이를 쉽게 이해할 수 있습니다. 명령형 프로그래밍 언어의 경우 “포인트를 만들어 색상을 설정한다. 그런 다음 선을 그려 직사각형을 생성한다.”라고 지정하여 직사각형을 그리는 방법을 설명합니다.

선언형 프로그래밍의 경우 특정 위치, 너비, 높이, 색상 속성의 직사각형을 원한다고 지정합니다. 결과를 얻는 방법 대신 원하는 결과를 설명하는 것입니다. 개발자는 코드를 빠르게 살펴보고 결과를 이해할 수 있습니다.

다음 주제에서는 몇 가지 **Kubernetes** 구성 요소를 살펴보고 선언형 형식을 사용하여 클러스터를 정의하는 방법을 검토합니다.

컨테이너는 마이크로 서비스에 적합



| 수강생용 노트

컨테이너는 확장 가능성과 이동 가능성이 뛰어나며 지속적인 배포가 가능하므로 마이크로서비스 아키텍처에 이상적인 옵션입니다.

마이크로서비스 아키텍처란 무엇입니까?

전통적으로 애플리케이션은 모놀리스 아키텍처로 구축되었습니다. 모놀리스 아키텍처를 사용하면 소프트웨어 애플리케이션의 구성 요소를 단일 플랫폼에서 단일 프로그램으로 구축할 수 있습니다. 애플리케이션은 자체 포함되어 있으며 프로세스 자체에 필요한 모든 태스크를 수행할 수 있습니다.

반대로, 마이크로서비스 아키텍처는 전통적인 모놀리스 아키텍처를 서비스로 실행되고 가벼운 API를 사용하여 통신하는 독립된 구성 요소로 해체합니다. 이러한 마이크로서비스 환경에서는 복원력, 효율성, 전반적인 민첩성이 향상되어 반복 작업을 더욱 신속하게 처리할 수 있습니다.

마이크로서비스 아키텍처를 사용하면 서비스를 기반으로 구성 요소를 분리할 수 있습니다. 모든 마이크로서비스를 컨테이너에 구축할 수 있습니다. 각 마이크로서비스는 별도의 구성 요소이므로 장애를 더 잘 견딜 수 있습니다. 컨테이너를 사용하면 컨테이너에 장애가 발생할 경우 해당 컨테이너를 종료하고 특정 서비스에 새 컨테이너를 신속하게 시작할 수 있습니다. 또한 특정 서비스에 트래픽이 많은 경우 해당 마이크로서비스를 처리하는 컨테이너 수를 확장할 수 있습니다. 이렇게 하면 전체 애플리케이션을 지원하기 위해 서버를 추가로 배포할 필요가 없습니다.

マイクロサービスとコンテイнерは、直線的なバッファードでも適応できます。アプリケーションの他の構成要素に影響を与えることなく、個別のサービスを素早く効率的にアップデートできます。



Running Containers on Amazon EKS

컨테이너 오케스트레이션



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

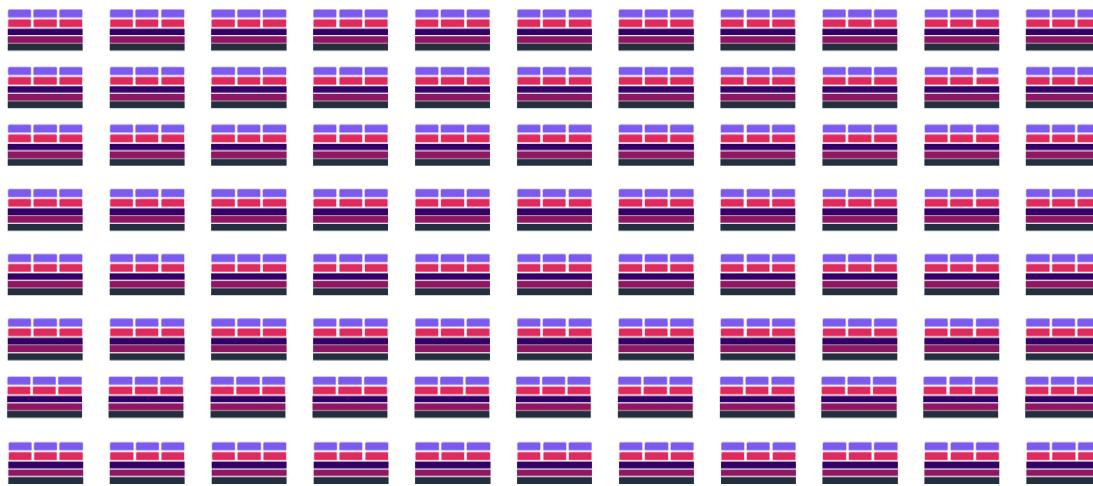


~ALT text

~호스트 그룹: 별도의 애플리케이션 및 라이브러리를 포함한 컨테이너가 여러 개 있는 호스트입니다. 컨테이너화 플랫폼, 운영 체제 및 서버 하드웨어도 표시되어 있습니다.

|수강생용 노트

컨테이너를 사용할 때는 확장성을 고려하는 것이 중요합니다. 단일 호스트에서 하나 또는 두 개의 컨테이너를 실행하는 것은 간단합니다. 하지만 수백 개의 컨테이너가 포함될 수 있는 호스트가 수십 개인 스테이징 및 테스트 환경으로 이동한다면 어떻게 될까요?



수천 개의 컨테이너가 있는 수백 개의 호스트



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

~ALT text

~호스트: 여러 행에 컨테이너가 여러 개 있는 호스트입니다.

| 수강생용 노트

수백 개의 호스트와 수천 개의 컨테이너가 있는 전체 프로덕션 환경을 가정해 보십시오. 그러면 엔터프라이즈 규모의 클러스터 환경을 관리하게 되므로 클러스터를 관리하기가 어려워집니다. 클러스터는 하나의 시스템으로 함께 작동하는 여러 개의 서버나 인스턴스의 집합입니다.

가용성, 복원력, 성능을 극대화하기 위해 컨테이너를 인스턴스에 지능적으로 배치하는 방법이 필요할 것입니다. 다시 말해 시스템 내 모든 것의 상태를 알아야 한다는 뜻입니다. 컨테이너를 적절한 서버에서 제어하고 예약할 수 있도록 모든 컨테이너를 관리하기 위한 소프트웨어가 필요합니다. 각 서버는 클러스터의 노드로 간주되고 서버마다 하드웨어가 다르고, 다른 OS를 실행할 수 있습니다.

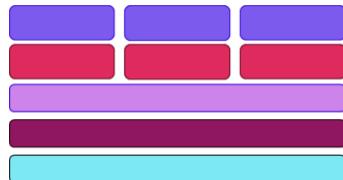
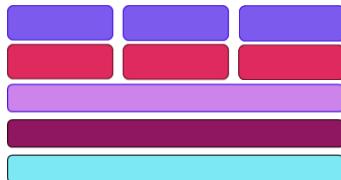
클러스터를 관리하려면 시스템에 대해 다음을 알아야 합니다.

- 메모리 및 포트와 같은 사용 가능한 리소스가 있는 인스턴스
- 컨테이너가 중단되는 시점
- 로드 밸런서로 연결하는 방법
- 지속적 통합(CI) 및 지속적 전달(CD)을 지원하는 방법

즉, 대규모로 컨테이너를 관리하는 방법을 알아야 합니다.

컨테이너 오케스트레이션 도구

- 스케줄링
- 클라우드
- 네트워킹
- 로드 밸런싱
- 스토리지
- 보안
- 모니터링
- 로깅



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

~Dev notes

~Alt text for Hosts – 각각 별도의 애플리케이션 및 라이브러리를 포함하는 여러 컨테이너가 있는 두 개의 호스트입니다.

|수강생용 노트

컨테이너 오케스트레이션 도구(또는 컨테이너 관리 플랫폼)는 컨테이너의 관리, 배포, 크기 조정을 자동화하는 데 사용됩니다. 컨테이너 관리 플랫폼은 클러스터에 있는 모든 컨테이너를 관리하는 데 도움이 되는 모든 도구입니다.

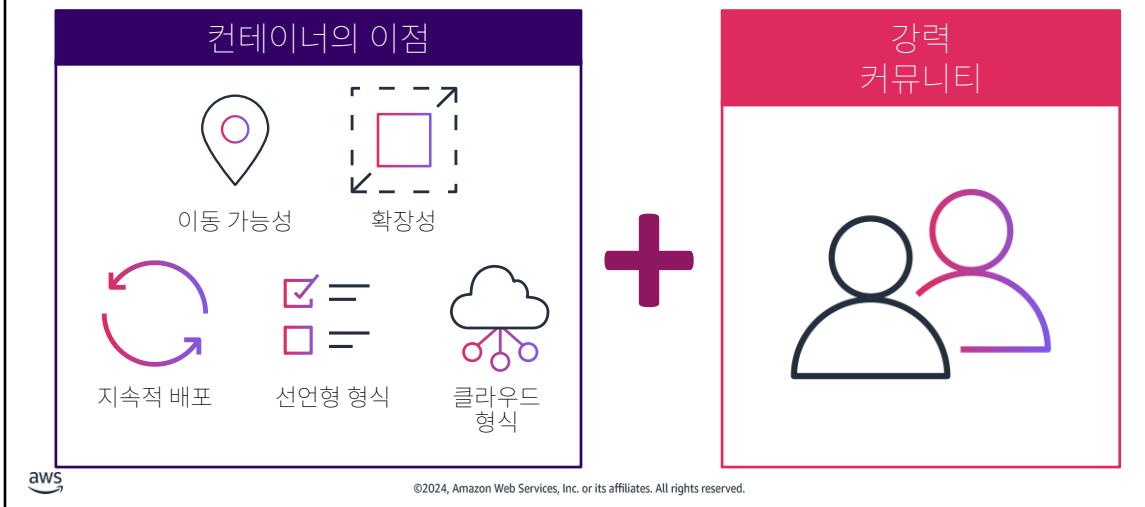
관리 플랫폼마다 컨테이너 시스템의 서로 다른 측면을 관리할 것입니다. 이러한 많은 플랫폼이 관리하는 주요 범주는 다음과 같습니다.

- 컨테이너의 스케줄링 및 배치(적절한 컨테이너 크기 자동 조정)
- 비정상 컨테이너를 자동으로 제거하고 그 위치에 새 컨테이너를 배포하여 컨테이너 자동 복구
- 컨테이너를 배치할 호스트 결정

관리 플랫폼은 트래픽 및 영구 스토리지의 로드 밸런싱을 처리하는 네트워킹 서비스와 같은 기타 서비스 및 클라우드와의 통합도 제공합니다. 이러한 기능을 사용하여 컨테이너가 종료될 때 데이터를 유지합니다. 관리 플랫폼은 시스템에 대한 보안, 모니터링, 로깅을 관리하여 보안 시스템을 유지 관리하는 데 도움을 줍니다. 컨테이너 관리 플랫폼 옵션은 **Amazon Elastic Container Service(Amazon ECS)**, 기본 도구(예: Docker Swarm), 오픈 소스 플랫폼(예: Kubernetes) 등 다양합니다.

관리 플랫폼은 컨테이너 기반 워크로드를 아키텍팅할 때 선택하는 중요 항목입니다. 개발 관점에서 컨테이너는 이동 가능성이 뛰어나며 다양한 플랫폼에서 동일하게 잘 실행되어야 합니다. 그렇지만 인프라 관점에서 볼 때 개발 파이프라인의 훈련, 도구, 조직과 같이 컨테이너 오케스트레이션 플랫폼과 관련된 항목을 선택할 수 있습니다.

Kubernetes



| 수강생용 노트

이 과정에서는 AWS에서 네이티브 업스트림 **Kubernetes**를 사용하여 컨테이너식 애플리케이션을 배포하고, 관리하고, 크기를 조정하는 데 **Amazon Elastic Kubernetes Service(Amazon EKS)**가 어떤 도움이 되는지 알아봅니다. 그렇지만 **Amazon EKS**에 대해 알아보기 전에 **Kubernetes**의 기본 개념을 알아야 합니다.

대부분의 컨테이너 오케스트레이션 도구와 마찬가지로 **Kubernetes**는 컨테이너 오케스트레이션 자동화, 크기 조정, 로드 밸런싱, 배포, 스케줄링에 도움이 됩니다. **12 Factor App**의 이동 가능성, 확장성, 지속적 배포를 준수하는 애플리케이션을 구축하기 위한 도구가 함께 제공됩니다.

Kubernetes는 오픈 소스이며, 강력하고 역동적인 커뮤니티의 지원을 받는다는 특징이 있습니다. Linux Foundation의 종속 항목인 **Cloud Native Computing Foundation(CNCF)**은 **Kubernetes**를 유지 관리합니다. CNCF는 **Kubernetes**를 포함해 각 프로젝트가 결정을 내리는 주체, 권한을 소유하는 주체, 프로젝트의 기타 세부 사항을 공개적으로 지정하도록 보장하는 중립 조직입니다. 이를 오픈 거버넌스라고 하며 이를 통해 오픈 소스 커뮤니티와의 신뢰를 형성합니다. CNCF에서는 프로젝트를 담당할 수 없습니다. 오픈 소스 커뮤니티는 오픈 거버넌스를 인정합니다. 이러한 이유로 **Kubernetes**가 GitHub에서 가장 인기 있는 프로젝트 중 하나로 손꼽힙니다. 다른 CNCF 프로젝트에 대해 자세히 알아보려면 CNCF 웹 사이트의 **Graduated and Incubating Projects'** 페이지(<https://www.cncf.io/projects/>)를 참조하십시오.

Kubernetes는 온프레미스 서버 외에도 여러 클라우드 제공업체를 지원합니다. **Kubernetes API**는 AWS 내부와 온프레미스에서 리소스를 추상화하는 데 사용할 수 있는 확장 가능한 단일 계층으로 생각할 수 있습니다. AWS에서 **Kubernetes**를 사용할 경우 AWS 제품 및 서비스의 크기, 성능, 다양한 기능을 활용할 수 있습니다. 온프레미스에서 컨테이너를 배포하기 위해 친숙한 **Kubernetes API**를 동일하게 사용하면서 이 작업을 수행할 수 있습니다.

또한 **Kubernetes**는 설정 자동화에 선언형 형식을 사용합니다. 따라서 새로운 개발자가 프로젝트에 참여하는 데 드는 시간과 비용이 절감됩니다. 다음에는 선언형 형식에 대해 자세히 알아봅니다.

PodSpec으로 Pod 정의

```
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app.kubernetes.io/name: webapp
spec:
  containers:
  - name: nginx
    image: nginx:1.25.1
    ports:
    - containerPort: 80
```



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

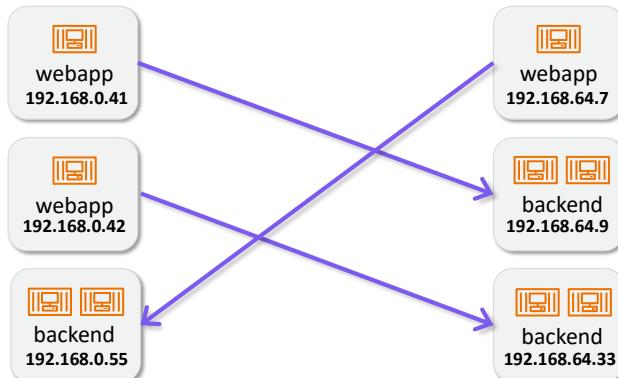
| 수강생용 노트

Pod는 하나 이상의 컨테이너로 구성된 그룹입니다. **PodSpec**은 컨테이너를 실행하는 방법에 대한 사양입니다. **Pod**는 배포, 확장, 복제를 위한 **Kubernetes** 내 기본 빌딩 블록입니다. **Pod** 내의 애플리케이션 컨테이너는 서로 쉽게 통신할 수 있습니다.

Pod는 YAML 기반 **PodSpec** 매니페스트 파일을 사용하여 정의됩니다. YAML 파일은 일반적으로 데이터가 저장되거나 전송되는 구성 파일 및 애플리케이션에 사용됩니다. YAML 콘텐츠는 JSON으로 변환된 후 처리를 위해 **Kubernetes API**로 전송됩니다. JSON은 매니페스트 파일에 허용되는 또 다른 형식입니다. 이 과정에서는 **Kubernetes**가 매니페스트 파일을 표시하는 데 주로 YAML이 사용됩니다.

PodSpec에는 **Pod**용으로 생성될 **Pod**의 이름, 컨테이너, 볼륨이 포함됩니다. 컨테이너는 항상 **Pod**의 일부이며 **Pod** 간에 공유되지 않습니다. 이 **PodSpec** 예는 **nginx** 컨테이너가 하나인 **Pod**를 정의합니다. 이 **PodSpec**은 선언형 형식의 좋은 예입니다. **PodSpec**은 **Kubernetes**에 **Pod** 구축 방식을 지시하는 대신 **Kubernetes**에 원하는 **Pod** 결과를 지시합니다.

Pod: 통신



aws

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

~Dev notes

~Alt text: Pod: 6개 Pod로 구성된 그룹입니다(3개는 webapp, 3개는 백엔드). 각 Pod에는 1개 또는 2개의 컨테이너가 있습니다.

~IP: 각 Pod는 고유한 IP를 갖습니다.

~통신: 각 웹앱은 IP만 알면 각 백엔드와 통신할 수 있습니다.

| 강사용 노트

| 애니메이션 노트

| <1> 고유한 IP가 각 Pod 아래에 나타납니다.

| <2> 화살표가 각 Pod 간의 통신을 표시합니다.

| 수강생용 노트

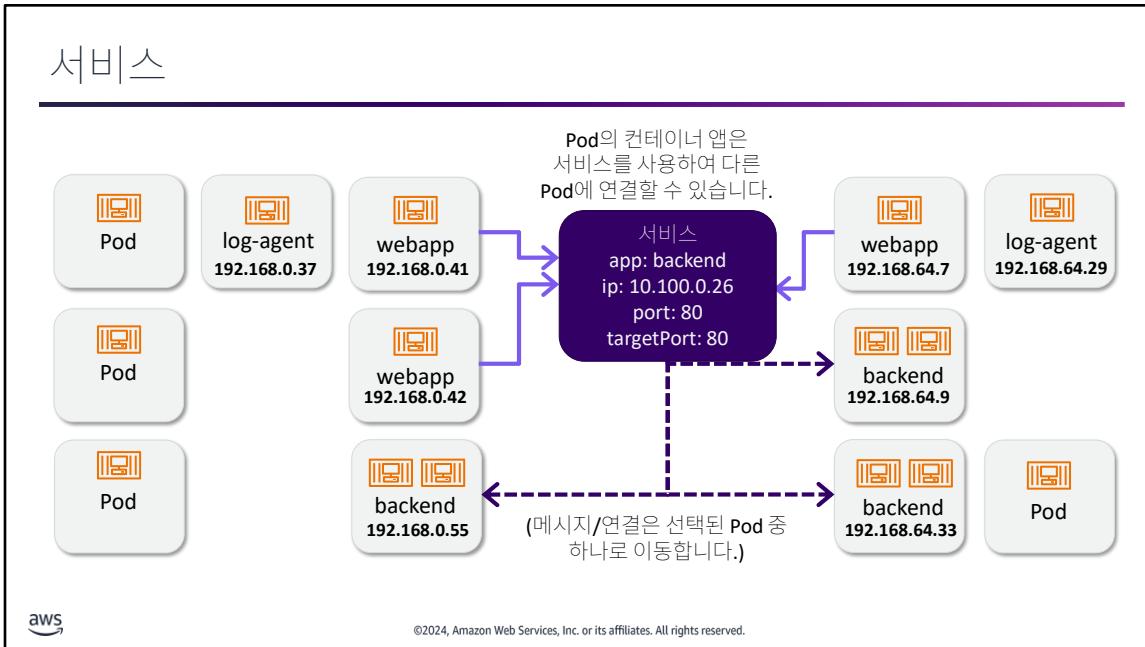
Pod는 하나 이상의 컨테이너로 구성된 그룹입니다. PodSpec은 컨테이너를 실행하는 방법에 대한 사양입니다. Pod는 배포, 확장, 복제를 위한 Kubernetes 내 기본 빌딩 블록입니다. Pod 내의 애플리케이션 컨테이너는 서로 쉽게 통신할 수 있습니다.

여러 Pod에 있는 컨테이너가 서로 통신해야 한다면 어떻게 해야 합니까? 각 Pod에는 Kubernetes 클러스터 내에서 사용되는 고유한 IP 주소가 할당됩니다. 개별 애플리케이션 컨테이너와 마찬가지로 Pod는 내구성 있는 엔터티가 아니라 상대적으로 휘발성이 있는 엔터티로 간주됩니다. 즉, Pod는 이상이 생기면 사라질 수 있고 새로운 Pod가 그 자리를 차지할 수 있습니다.

이 경우 **Pod**는 통신을 어디로 보내야 할지를 어떻게 알 수 있습니까? **Pod**의 할당된 IP 주소는 **Pod**가 사라지면 할당이 취소됩니다. **Pod**를 교체하면 다른 IP 주소가 할당될 가능성이 높습니다.

Kubernetes는 동일한 애플리케이션 구성 요소의 복제본을 구현하는 **Pod** 모음을 참조하기 위한 메커니즘을 구현합니다. 모든 **Pod**에는 IP 주소가 할당됩니다. 서비스와 같은 다른 객체도 마찬가지입니다. 이 과정에서는 이후 모듈에서 IP 관리에 대해 보다 자세히 설명합니다.

서비스



~Dev note

~Do note regroup as it will break animation

~Pods: At first, the same set of six pods is visible

~Communication: A service is shown. The three pods labeled “webapp” are connected to the service with a solid line. The three labeled as “backend” are shown connected with a dotted line.

~Additional Pods: additional pods are shown on either side. Same labeled “pod” and some labeled “log-agent”. More information is available in the notes.

| 강사용 노트

| 이 슬라이드를 사용하여 커뮤니케이션 형태로 서비스를 소개하십시오. 최종 애니메이션은 추가 Pod를 믹스에 추가하여 더 큰 규모의 클러스터를 표시합니다.

| 애니메이션 노트

| <1> 서비스가 나타납니다.

| <2> Pod와 서비스 사이에 통신 화살표 및 텍스트가 나타납니다.

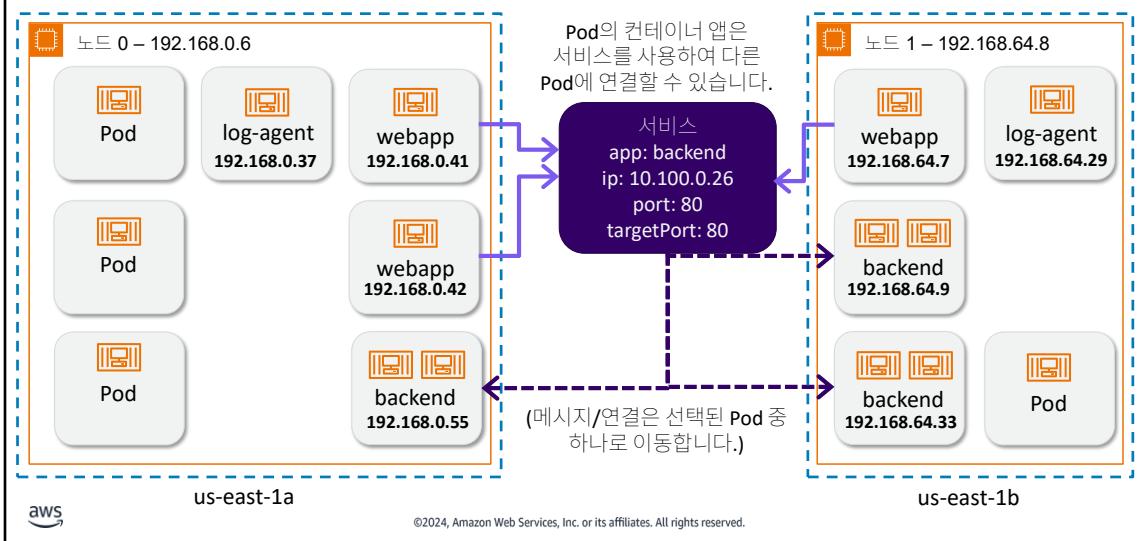
| <3> 추가 Pod가 나타납니다.

| 수강생용 노트

Kubernetes에서 서비스란 여러 Pod의 논리적 모음과 여기에 액세스하는 수단입니다. 서비스는 사용 가능한 Pod 세트로 지속적으로 업데이트되며, 각 Pod IP 주소는 통신의 엔드포인트 역할을 합니다. 각 서비스에는 ClusterIP라고 하는 고유한 가상 IP 주소가 할당됩니다. 해당 값은 클러스터 내에서 고유합니다. 클라이언트는 서비스의 가상 ClusterIP 주소에 연결되고, 서비스는 통신 요청을 서비스 뒤에 있는 엔드포인트 Pod 중 하나로 변환합니다. 이렇게 하면 Pod가 다른 Pod의 주소를 직접 추적할 필요가 없습니다.

이 예에서는 프런트엔드 **webapp Pod**가 백엔드 애플리케이션에 액세스해야 합니다. 고가용성 스테이트리스 앱을 위한 백엔드 **Pod** 모음의 경우 어떤 **Pod**와 통신하는지가 중요한 것이 아니고, 프런트엔드 **webapp**이 백엔드 애플리케이션의 인스턴스에 안정적으로 연결할 수 있어야 합니다. 이 서비스는 포트 **80**에서 요청된 애플리케이션에 대한 연결 요청을 분산합니다. 그런 다음 애플리케이션을 '**app=backend**' 레이블이 있는 모든 **Pod**에 매핑합니다. 이 서비스는 그 **Pod**에 구성된 대상 포트(이 예에서는 **80**)를 사용합니다.

노드 및 가용 영역



~Dev notes

~Node_AZ: the same diagram used in the previous slide is shown, but with nodes drawn over two sets of pods. Each node is shown to be located within a different availability zone, us-east-1a and us-east-1b.

| 수강생용 노트

Kubernetes 설치를 클러스터라고 합니다. 각 Kubernetes 클러스터에는 컨테이너의 Pod를 실행하는 노드가 포함되어 있습니다. Kubernetes 클러스터 개념에 대한 설명은 Kubernetes 설명서 페이지의 ‘Overview’ (<https://kubernetes.io/docs/concepts/overview/>)를 참조하십시오.

Pod의 컨테이너는 항상 공동 배치되고, 한 노드에서 실행되도록 함께 예약되고, 해당 Pod의 수명 주기 전체에서 함께 관리됩니다. 즉, 한 Pod에 있는 여러 컨테이너를 노드 간에 분할할 수 없습니다. 그러나 다른 Pod를 클러스터의 다른 노드에 예약할 수 있습니다. 특정 Pod를 실행하도록 예약할 노드를 결정하는 규칙은 여러 가지 있으며, 이에 대해서는 다른 섹션에서 다룹니다.

각 노드에는 다음을 비롯한 특정 용량이 있습니다.

- CPU 용량 및 (일부의 경우) GPU 용량
- 메모리 용량
- 최대 Pod 수

물론 각 노드의 기술적 세부 사항에는 관련 스토리지 용량, 입력/출력 처리량, 네트워크 대역폭 용량도 포함될 수 있습니다. Kubernetes 노드에 대한 설명은 Kubernetes 설명서 페이지의 ‘Nodes’(<https://kubernetes.io/docs/concepts/architecture/nodes/>)를 참조하십시오.

Kubernetes 설명서에는 ‘Considerations for large clusters’ (<https://kubernetes.io/docs/setup/best-practices/cluster-large/>)도 포함되어 있습니다. 여기에는 컨테이너, Pod, 노드당 Pod 및 클러스터 내 노드에 대해 Kubernetes 커뮤니티에서 권장하는 최대 개수가 포함됩니다. 동일한 클러스터에서 이러한 최대 개수를 모두 달성할 수는 없지만 최소 공통 제한 요소 아키텍처가 권장됩니다.

각 Kubernetes 노드는 물리적 또는 가상 장애 도메인을 나타냅니다. 또한 특정 노드가 호스팅되는 가용 영역도 장애 도메인을 나타냅니다. Pod 사양에는 동일한 노드 또는 가용 영역에서 예약할 수 있는 Pod와 다른 노드 또는 가용 영역에 분산될 수 있는 Pod를 결정하는 데 사용할 수 있는 Kubernetes 스케줄러에 대한 제안이 포함될 수 있습니다.

따라서 논리적 고가용성을 제공하기 위해 각 애플리케이션 구성 요소(앱, 구성 요소 또는 마이크로서비스)에 대해 여러 개의 복제본 Pod를 사용하는 방식을 채택하고 해당 Pod 모음에 대한 요청을 배포, 전달, 로드 밸런싱하는 서비스를 사용할 수 있습니다. 다중 노드, 다중 가용 영역, Pod 예약에 대한 적절한 힌트를 포함하도록 클러스터 인프라를 신중하게 설계하면 이러한 고가용성을 가상 및 물리적 영역으로 확장할 수 있습니다.



Running Containers on Amazon EKS

Kubernetes 내부



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

제어 영역 및 데이터 영역

제어 영역: 클러스터를 관리



제어 영역 노드

데이터 영역: 애플리케이션을 실행



노드



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

~ALT Text

~제어 영역: 제어 영역 노드가 있는 제어 영역입니다.

~데이터 영역: 노드가 있는 데이터 영역입니다.

|수강생용 노트

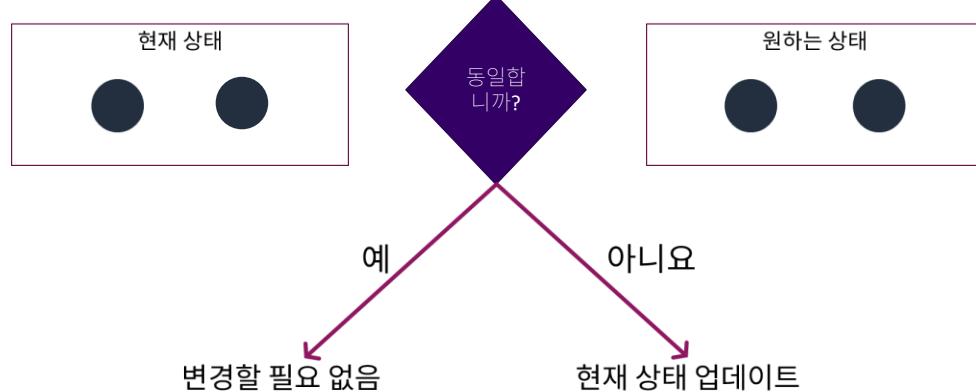
Kubernetes 클러스터를 실행하는 시스템을 노드라고 하며, 노드는 제어 영역 및 데이터 영역으로 구성됩니다.

- 제어 영역 구성 요소는 클러스터에 관해 스케줄링과 같은 전역 결정을 내립니다. 또한 클러스터 이벤트를 감지하고 대응합니다(예를 들어, 배포의 복제본 필드가 만족스럽지 않을 때 새 Pod 시작).
- 데이터 영역은 컨테이너식 애플리케이션을 실행하는 노드로 구성됩니다.

여러 제어 영역 노드와 여러 노드를 유지할 수 있습니다.

컨트롤러

제어 루프



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

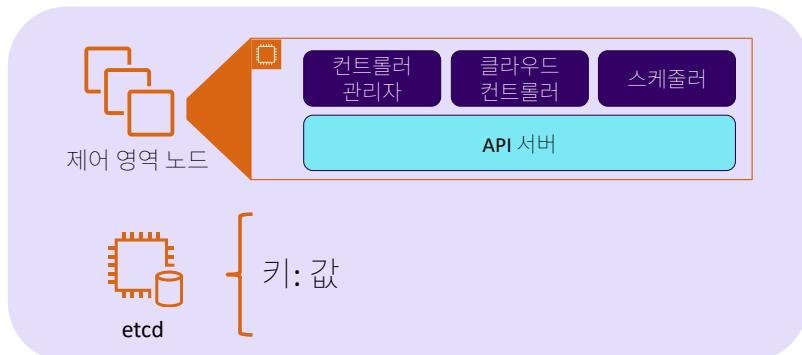
~ALT text

- ~원하는 상태: 원하는 상태가 2개의 원으로 표시되어 있습니다.
 - ~현재 상태: 현재 상태는 1개의 원으로 표시되어 있습니다. 또 다른 원은 제어 루프를 거친 후에 애니메이션으로 표시됩니다.
 - ~제어 루프 – 예: 현재 상태 및 원하는 상태가 같으면 변경이 필요하지 않습니다.
 - ~제어 루프 – 아니요: 현재 상태 및 원하는 상태가 같지 않으면 현재 상태를 업데이트합니다.
- | 강사용 노트 및 애니메이션
- | <1> 현재 상태가 원하는 상태와 같지 않으면 컨트롤러는 일치하도록 현재 상태를 업데이트합니다.
- | <2> 현재 상태가 원하는 상태와 같으면 아무 작업도 일어나지 않습니다.
- | 수강생용 노트
- Kubernetes 제어 영역의 사양에 대해 알아보기 전에 Kubernetes 컨트롤러를 이해해야 합니다. 컨트롤러는 현재 상태가 원하는 상태와 같은지 확인합니다. 이를 제어 루프라고 합니다.

현재 상태가 원하는 상태와 같지 않으면 컨트롤러는 일치하도록 현재 상태를 업데이트합니다. 현재 상태가 원하는 상태와 같으면 아무 작업도 일어나지 않습니다. 이는 오븐 온도 컨트롤러와 비슷합니다. 오븐 온도 컨트롤러는 지속적으로 오븐의 온도가 원하는 온도와 같은지 확인합니다. 온도가 같지 않으면 컨트롤러는 적절하게 오븐 히터를 켜거나 끄도록 안내합니다.

이 모듈의 뒷부분에서 배포, **ReplicaSet**, **DaemonSet**와 같은 일부 컨트롤러에 대해 학습합니다.

제어 영역



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

~ALT text

- ~제어 영역 노드: **Kubernetes** 컨트롤러 관리자, 클라우드 컨트롤러, 스케줄러, **API** 서버를 표시하는 제어 영역 노드입니다.
- ~etcd: **etcd**는 중요 클러스터 데이터 및 상태가 저장되는 고가용성 키 값 저장소입니다.
 - | 강사용 노트
 - | 클라우드 컨트롤러는 기본 클라우드 공급업체와 상호 작용하는 특정 컨트롤러입니다.
 - | 스케줄러는 새로 생성된 **Pod**가 실행될 노드를 선택합니다.
 - | API 서버는 **Kubernetes API**를 노출하며 **Kubernetes** 제어 영역의 프런트엔드 역할을 합니다.
 - | 수강생용 노트
제어 영역 구성 요소는 클러스터의 기본 제어 지점입니다. 제어 영역 구성 요소에는 **Kubernetes** 컨트롤러 관리자, 클라우드 컨트롤러, 스케줄러, **API** 서버가 포함됩니다.

Kubernetes 컨트롤러 관리자는 컨트롤러라는 핵심 **Kubernetes** 제어 루프를 실행하고 클러스터 이벤트를 감지하고 이에 대응합니다. 컨트롤러는 제어 루프를 실행하여 클러스터의 상태를 감시하고, 클러스터의 현재 상태가 원하는 상태인지 확인합니다. 즉, 컨트롤러는 선언형 코드를 확인하고, 클러스터가 지정된 상태가 되는지 확인합니다.

클라우드 컨트롤러는 기본 클라우드 공급업체와 상호 작용하는 특정 컨트롤러입니다.

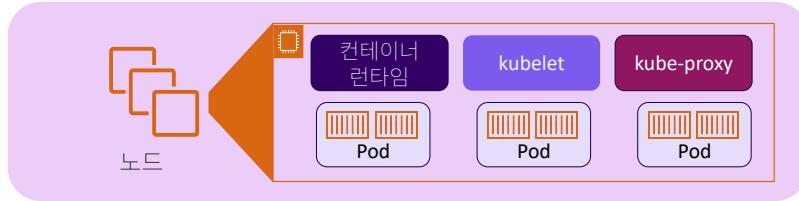
스케줄러는 새로 생성된 **Pod**가 실행될 노드를 선택합니다.

API 서버는 **Kubernetes API**를 노출하며 **Kubernetes** 제어 영역의 프런트엔드 역할을 합니다. **API** 서버는 클러스터에서 제어 영역으로의 모든 통신을 처리합니다. 다른 제어 영역 구성 요소는 원격 서비스를 노출하도록 설계되지 않았습니다. **API** 서버는 수평적으로 확장하여 필요에 따라 더 많은 인스턴스를 배포할 수 있도록 설계되었습니다. 제어 영역은 다음 구성 요소를 실행할 수도 있습니다.

- 이름 확인을 위한 추가 기능(예: **KubeDNS**)
- 대시보드(웹 기반 사용자 인터페이스)
- 네이티브 및 서드 파티를 모두 포함하는 기타 서비스(모니터링, 로깅)

Kubernetes 제어 영역 노드 외에 **Kubernetes**용 핵심 지속성 계층인 **etcd**도 실행해야 합니다. **etcd**는 고가용성 분산 키 값 저장소입니다. 여기에 중요한 클러스터 데이터와 상태가 저장됩니다. 제어 영역을 지원하는데 필요한 인스턴스 수를 줄이기 위해 제어 영역 구성 요소와 **etcd**를 같은 인스턴스에 공동 배치할 수 있습니다. 이러한 모든 서비스가 **Kubernetes**의 제어 영역을 구성합니다.

데이터 영역



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

~ALT text

노드: 컨테이너 런타임, kubelet, kube-proxy 등 Pod를 실행하는 데 필요한 서비스가 있는 여러 Pod가 포함된 노드입니다.

| 강사용 노트 및 애니메이션

| <1> 노드는 컨테이너 런타임도 실행합니다. Kubernetes는 Docker 엔진 및 containerd와 같은 인기 있는 런타임을 포함하는 여러 런타임을 지원합니다.

Kubernetes v1.24 이후부터 EKS는 기본적으로 containerd를 실행합니다.

| <2> Kubelet은 노드에서 실행되는 기본 에이전트입니다. 다음에는 Kubelet에 대해 자세히 알아봅니다.

| <3> 각 노드에는 Pod 외에 네트워킹에 도움이 되는 kube-proxy가 포함됩니다. 호스트에서 네트워크 규칙을 유지 관리하고, 필요할 수 있는 모든 연결을 전달합니다.

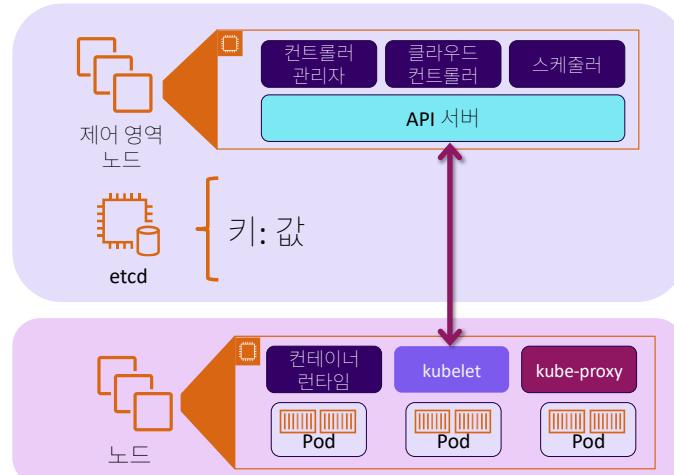
| 수강생용 노트

노드는 Kubernetes 런타임 환경을 지원합니다. Kubernetes의 작업자 시스템을 노드라고 합니다. 노드는 Pod를 실행하는 데 필요한 서비스를 포함하고, 제어 영역 구성 요소에서 관리됩니다.

- 컨테이너 런타임 - 노드는 컨테이너 런타임도 실행합니다. Kubernetes는 Docker 엔진 및 containerd와 같은 인기 있는 런타임을 비롯하여 여러 런타임을 지원합니다. Kubernetes v1.24 이후부터 Amazon EKS는 기본적으로 containerd를 실행합니다.
- **Kubelet** – 노드에서 실행되는 기본 에이전트입니다. 다음에는 kubelet에 대해 자세히 알아봅니다.

- **Kube-proxy** – 각 노드에는 Pod 외에 네트워킹에 도움이 되는 **kube-proxy**가 포함됩니다. 호스트에서 네트워크 규칙을 유지 관리하고, 필요할 수 있는 모든 연결을 전달합니다.

제어 영역과 데이터 영역 통신



aws

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

~ALT text

~제어 영역: 제어 영역 노드 및 **etcd**로 표현되는 제어 영역입니다. 이 제어 영역은 **API** 및 **kubelet**을 통해 데이터 영역과 통신합니다.

~데이터 영역: 노드 및 **Pod**가 있는 데이터 영역입니다. 이 데이터 영역에도 제어 영역 API와 통신하는 **kubelet**이 있습니다.

| 수강생용 노트

제어 영역과 데이터 영역이 함께 작동할 때 제어 영역 노드는 컨트롤러와 스케줄러를 사용하여 노드의 상태가 원하는 상태가 되도록 합니다. 제어 영역과 노드 간의 통신은 API 서버를 통해 **kubelet**으로 전달됩니다.

Kubelet은 **Pod**에 올바른 컨테이너가 실행되도록 하고 그 컨테이너가 정상 상태가 되도록 합니다. **Kubelet**은 다음 활동을 수행합니다.

1. (제어 영역의 API 서버 또는 로컬 구성 파일에 의해) 해당 노드에 할당된 **Pod**를 감시합니다.
2. 필요한 볼륨 탑재
3. **Secret** 다운로드
4. 컨테이너 런타임을 사용하여 **Pod**의 컨테이너 실행
5. 요청된 컨테이너 활성 프로브 또는 상태 검사 주기적 실행
6. 노드의 상태를 시스템의 나머지 부분에 다시 보고



Running Containers on Amazon EKS

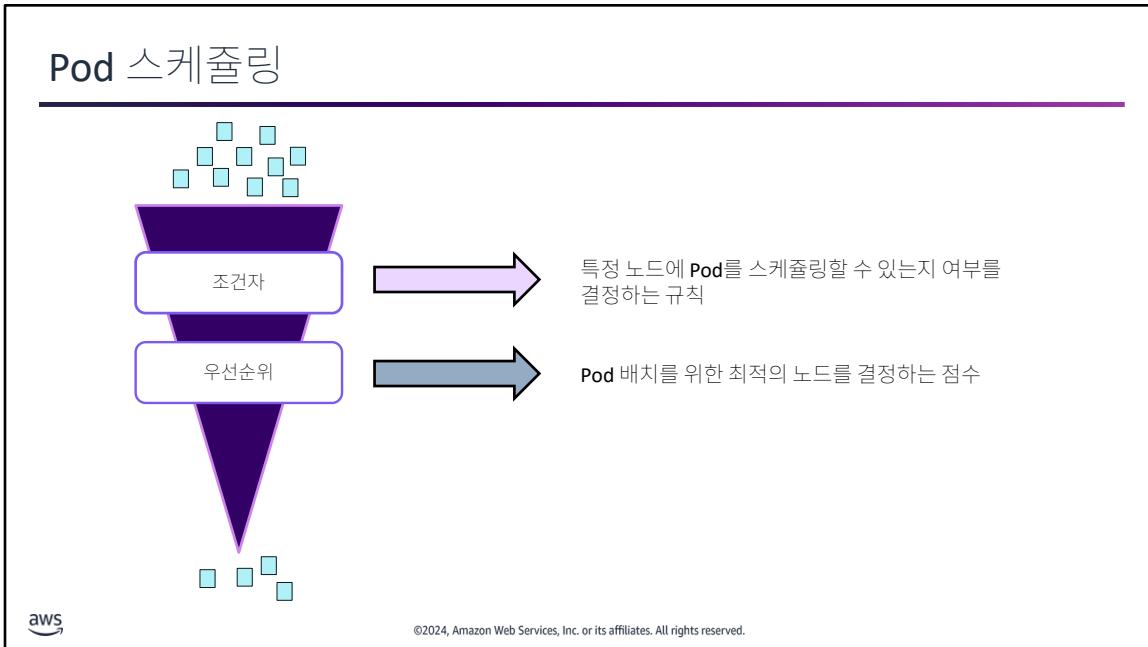
Pod 스케줄링

조건자 및 우선순위를 사용하여
Pod 스케줄링



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

Pod 스케줄링



| 수강생용 노트

Kubernetes 스케줄러를 사용하여 **Pod**를 스케줄링할 수 있습니다. 이 스케줄러는 **Pod**에 필요한 리소스를 확인하고 그 정보를 사용하여 스케줄링 결정에 영향을 줍니다. 스케줄러는 필터를 사용하여 **Pod** 배치에 부적격한 노드를 제외한 다음, 점수 체계에 따라 최종 노드를 결정합니다.

조건자는 부적격 노드를 결정합니다. 이 부적격은 볼륨, 리소스 또는 토플로지 요구 사항으로 인해 발생할 수 있습니다.

우선순위는 필터링되지 않은 각 노드의 점수를 계산합니다. 그런 다음 스케줄러는 점수가 가장 높은 노드에 **Pod**를 배치합니다.

필터를 모두 적용한 후에 사용 가능한 노드가 남아 있지 않으면 스케줄링에 실패합니다.

조건자: 리소스 요구 사항

Container A	Container B
requests:	requests:
cpu: 400m	cpu: 600m
memory: 600Mi	memory: 600Mi
limits:	limits:
cpu: 800m	cpu: 800m
memory: 800Mi	memory: 800Mi
Pod resources total	
requests:	
cpu: 1000m	
memory: 1200Mi	
limits:	
cpu: 1600m	
memory: 1600Mi	



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

• 리소스 요청

- 스케줄러가 할당 가능한 용량이 충분히 남아 있는 후보 노드를 선택하기 위해 사용

• 리소스 한도

- Pod의 컨테이너가 메모리 한도를 초과하는 경우 CPU 속도 제한 또는 Pod 제거를 적용하는데 사용

| 수강생용 노트

리소스 필터를 사용하는 경우 스케줄러는 Pod에 필요한 리소스가 어떤 노드에 있는지 고려합니다. 여기에는 CPU, 메모리, 디스크 공간, 사용 가능한 포트 등이 포함됩니다.

이 예에서는 두 컨테이너가 하나의 Pod에 속합니다. 두 가지 설정인 requests와 limits는 컨테이너의 리소스 사용률을 제어합니다. Pod의 각 컨테이너에서 requests 파라미터를 사용하여 할당된 리소스 수를 지정할 수 있습니다. 이러한 리소스는 컨테이너에 대해 보장됩니다. 즉, 이 설정은 스케줄러가 컨테이너의 Pod를 배치하는 위치에 영향을 미칩니다.

limits 파라미터는 Pod가 실행된 후에 리소스에 대한 유한한 한도를 정의합니다. 실행 중인 컨테이너는 리소스 사용량을 원래 요청량에서 정의된 한도까지 버스트할 수 있습니다. 컨테이너가 메모리 한도를 초과하면 스케줄러는 컨테이너를 종료합니다. 컨테이너가 한도보다 많은 CPU를 사용하는 경우 컨테이너의 프로세스가 제한됩니다.

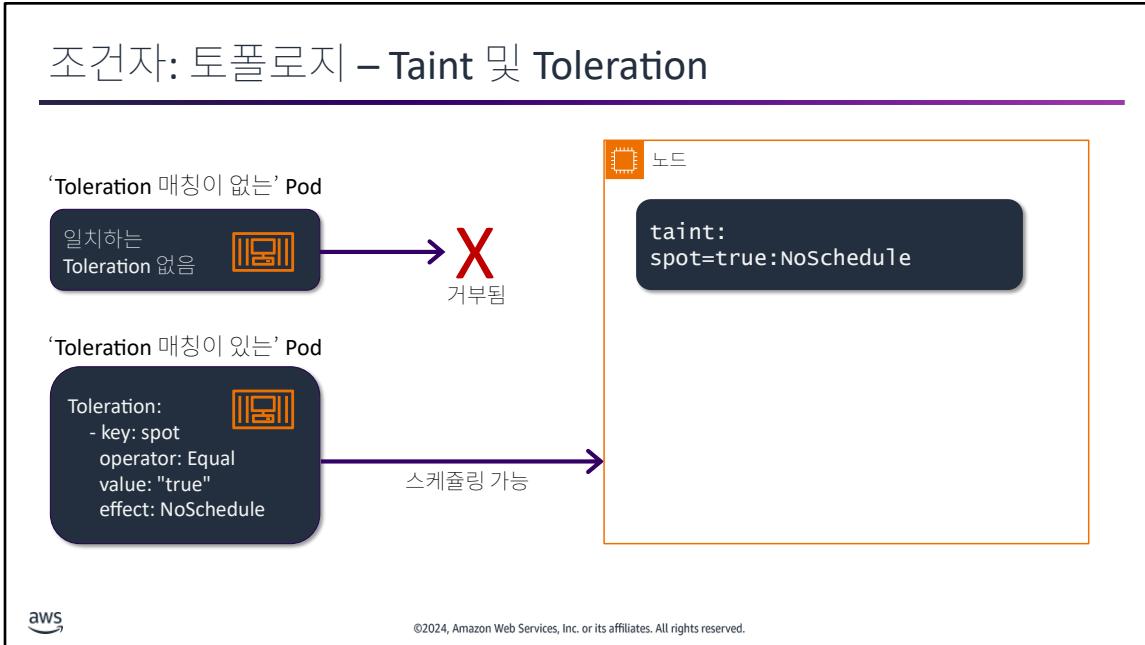
리소스 요구 사항은 컨테이너 레벨에서 정의되므로 모든 컨테이너에 대해 요청된 리소스의 합계에 따라 Pod의 리소스가 정의됩니다. 예약을 결정할 때 스케줄러는 리소스의 수를 사용합니다. 노드는 오버프로비저닝될 수 없으므로 클러스터에 Pod에 대한 충분한 리소스가 없으면 예약에 실패합니다.

limits 및 requests의 CPU는 CPU 단위로 측정됩니다. limits 및 requests의 메모리는 바이트 단위로 측정됩니다. 이 경우 컨테이너 A에는 400millipu(또는 400millicores) 및 600mebibyte가 있습니다.

이 예에서 2개의 컨테이너에서 요청된 CPU 합계의 단위는 **1,000** 단위이며 이는 전체 가상 CPU(vCPU) 1개에 해당합니다. 이 Pod를 배치할 노드를 선택할 때 스케줄러가 사용하는 값입니다. 실행될 경우 Pod는 제거 정책이 적용되기 전에 최대 **1,600** 단위로 버스트되도록 허용됩니다.

자세한 내용은 kubernetes 설명서 페이지의 ‘Resource Management for Pods and Containers’ 섹션에서 ‘Requests and limits’ (<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits>)를 참조하십시오.

조건자: 토플로지 – Taint 및 Toleration



| 수강생용 노트

볼륨 및 리소스 제약 조건을 충족한 후에는 스케줄러가 Pod 배치를 미세 조정하도록 설정된 제약 조건을 고려합니다. 사용자는 노드 수준과 Pod 수준에서 스케줄링 제약 조건을 설정할 수 있습니다. **Taint** 및 **Toleration**은 특정 노드에 배치할 수 있는 노드를 제어하기 위해 함께 작동하는 설정입니다.

Taint는 Pod 배치를 방지하는 노드의 속성입니다. 테인트된 노드는 해당 **Taint**를 허용하는 Pod만 수락합니다. 노드를 테인트하려면 키=값 쌍(예: `spot=true`)을 지정한 다음, **Taint**가 고려되는 경우를 정의하는 작업을 추가합니다. 이 예에서는 스케줄링 중에 **Taint**를 따르도록 스케줄러가 구성됩니다. Pod에 일치하는 **Toleration**이 없으면 이 노드에서 Pod가 스케줄링되지 않습니다.

Toleration은 Pod가 테인트된 노드에서 실행될 수 있도록 지정해주는 Pod의 속성입니다. **Toleration**은 특정 **Taint**와 일치해야 합니다. 이 예에서 **Toleration**이 이전에 설정된 **Taint**와 일치하는 특정 Pod에 적용됩니다. 이제 스케줄러는 테인트된 노드에 이 Pod를 배치하도록 허용되지만 다른 제약 조건에 따라 다른 위치에서 스케줄링할 수도 있습니다.

조건자: 토플로지 – 선호도

```
apiVersion: v1
kind: Pod
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "node.kubernetes.io/instance-type"
                operator: In
                values: ["r4.large", "r4.xlarge"]
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
              - key: "topology.kubernetes.io/zone"
                operator: In
                values:
                  - "us-east-1a"
```



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

| 수강생용 노트

경우에 따라 **Pod**가 특정 노드에서 스케줄링되도록 해야 할 수 있습니다. **Pod**에 특정 하드웨어 리소스(예: 그래픽 처리 장치(**GPU**) 또는 솔리드 스테이트 드라이브(**SSD**))가 필요하다고 가정해 보겠습니다. **Pod**가 특정 노드 또는 인스턴스 유형에서 실행되도록 하려면 선호도 설정을 사용하면 됩니다. 선호도 설정은 **Taint** 및 **Toleration**과 함께 사용할 경우 올바른 **Toleration**이 있는 **Pod**만 노드로 스케줄링할 수 있도록 합니다.

nodeAffinity 설정을 사용하여 **Pod**가 일부 인스턴스 유형에서만 실행되도록 할 수 있습니다. 노드 선호도로는 **requiredDuringSchedulingIgnoredDuringExecution** 및 **preferredDuringSchedulingIgnoredDuringExecution**이라는 두 가지 유형이 있습니다. 이러한 선호도를 각각 ‘하드’ 및 ‘소프트’로 간주할 수 있습니다. 하드는 노드에서 **Pod**를 스케줄링하기 위해 충족해야 하는 규칙을 지정합니다. 소프트는 기본 설정을 지정하며 스케줄러에서는 이를 적용하려고 하지만 보장되지는 않습니다. 이름의 ‘**IgnoredDuringExecution**’ 부분은 런타임에 노드의 레이블이 변경되어 **Pod**의 선호도 규칙이 더 이상 충족되지 않을 경우 **Pod**가 노드에서 계속 실행됨을 의미합니다.

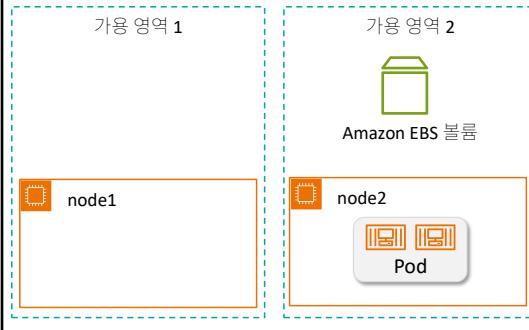
이 예에서는 **Pod**가 **r4.large** 또는 **r4.xlarge** 인스턴스에서 반드시 실행되도록 지정합니다. 스케줄러는 가용 영역 **us-east-1a**에서 이 **Pod** 세트를 실행하려고 합니다. 그렇지만 가능하지 않은 경우 이 **Pod** 세트를 다른 위치에서 실행할 수 있습니다.

또한 **Pod** 수준에서 선호도 및 비선호도를 지정하여 다른 **Pod**에 상대적으로 **Pod**를 배치하는 방식을 제어할 수도 있습니다. 예를 들어, 비선호도 스케줄링을 통해 **Pod**가 같은 노드에 위치하지 않도록 할 수 있습니다. 스케줄러가 특정 **Pod**에서 단일 장애 지점을 생성하지 않게 하려면 이 작업을 수행하십시오.

자세한 내용은 Kubernetes 설명서 페이지에서 다음 자료를 참조하십시오.

- ‘Affinity and anti-affinity’ (<https://kubernetes.io/docs/concepts/scheduling-eviction/assign-Pod-node/#affinity-and-anti-affinity>)
- ‘Well-Known Labels, Annotations and Taints’ (<https://kubernetes.io/docs/reference/labels-annotations-taints/>)

조건자: 볼륨 요구 사항



```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - name: app
      image: centos
      command: ["/bin/sh"]
      args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 5; done"]
  volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: ebs-claim
```



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

| 수강생용 노트

볼륨 필터는 **Pod**의 볼륨 요구 사항을 확인하고 호환되는 노드를 결정합니다. 예를 들어 한 가용 영역의 **Amazon Elastic Block Store(Amazon EBS)** 볼륨은 다른 가용 영역의 노드에 연결할 수 없습니다. 마찬가지로 **Pod**에는 노드에 이미 탑재된 특정 볼륨이 필요할 수 있습니다. 이 경우 그 **Pod**를 동일한 노드에 배치해야 합니다.



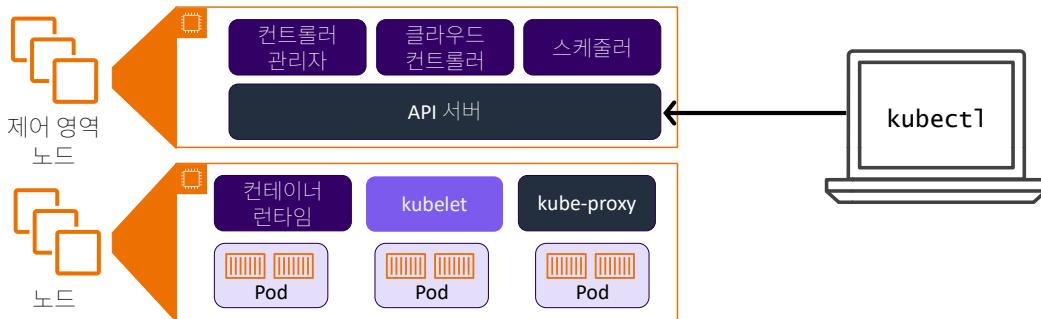
Running Containers on Amazon EKS

kubectl utility



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

Kubectl 도구



```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-4293833666-20vr8	1/1	Running	0	2m
nginx-4293833666-3gzfw	1/1	Running	0	2m
nginx-4293833666-7nbih	1/1	Running	0	2m

aws

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

~ALT text

~제어 영역 노드: 컨트롤러 관리자, 클라우드 컨트롤러, 스케줄러, API 서버가 있는 제어 영역입니다.

~데이터 영역: 노드 및 Pod가 있는 데이터 영역입니다. 컨테이너 런타임, kubelet, kube-proxy도 표시되어 있습니다.

~kubectl 컴퓨터: Kubectl은 제어 영역에서 Kubernetes API 서버와 통신합니다.

| 강사용 노트 및 애니메이션

| <1> 클러스터의 모든 Pod를 보기 위한 kubectl get 명령이 나타납니다.

| 시간이 허락한다면 이 슬라이드 이후에 데모를 하는 것이 좋습니다.

| 데모용(다음 슬라이드를 숨김 해제할 수도 있음)

| 시간이 허락한다면 데모를 진행합니다(시간이 충분하면 이 슬라이드의 숨기기를 해제).

| 데모 요구 사항: Kubernetes 클러스터 2개, 호환 가능한 kubectl 버전, Bash 명령줄 액세스 권한

| 셸 환경에서 kubectl을 사용하여 KUBECONFIG 변수를 클러스터에 액세스하기 위한 참조 지점으로 설정하는 방법을 시연합니다.

| ~/.kube/ 디렉터리에서 여러 구성 파일을 생성하고 별칭을 사용하여 참조할 kubectl의 컨텍스트를 설정하는 방법을 시연합니다.

|

| 출처:

| <https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/>

| <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/eks/update-kubeconfig.html>

| 수강생용 노트

kubectl을 사용하여 제어 영역 노드와 통신할 수 있습니다. **Kubectl**은 특정 클러스터에서 **Kubernetes API** 서버와 통신하기 위한 명령줄 인터페이스(**CLI**)입니다. 컨테이너 오케스트레이션과 관리를 위한 다목적 도구인 **Kubectl**은 **Kubernetes** 클러스터 관리자를 제어합니다. **Kubectl**은 리소스를 생성하고, 클러스터와 리소스에 대한 세부 정보를 확인하고, 문제 해결 도구에 액세스하는 명령을 제공합니다. **Kubectl** 명령은 리소스를 롤아웃, 조정 및 자동 조정하는 데 사용합니다.

kubectl 명령을 실행하려면 다음 구문을 사용하십시오.

kubectl [명령] [유형] [이름] [플래그]

- 명령 - 수행할 작업을 지정합니다.
- 유형 - 리소스 유형을 지정합니다.
- 이름 - 리소스의 이름을 지정합니다.
- 플래그 - 선택적 플래그를 지정합니다.

이 예에서는 **kubectl get** 명령을 사용하여 클러스터에 있는 모든 **Pod**를 확인합니다.

사용자 정의 리소스를 사용하면 **kubectl** 명령을 사용하여 **Kubernetes** 리소스와 동일한 방식으로 리소스를 읽고 쓸 수 있습니다. 예를 들어 **Pod**의 현재 상태를 가져오는 것과 동일한 방법으로 사용자 정의 리소스의 현재 상태를 가져올 수 있습니다. **kubectl api-resources** 명령을 실행하면 짧은 이름, **API** 그룹, 네임스페이스 지정 여부, 종류를 비롯해 지원되는 모든 리소스 유형 목록이 출력됩니다.



Running Containers on Amazon EKS

Kubernetes 객체



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

네임스페이스(1/2)

클러스터



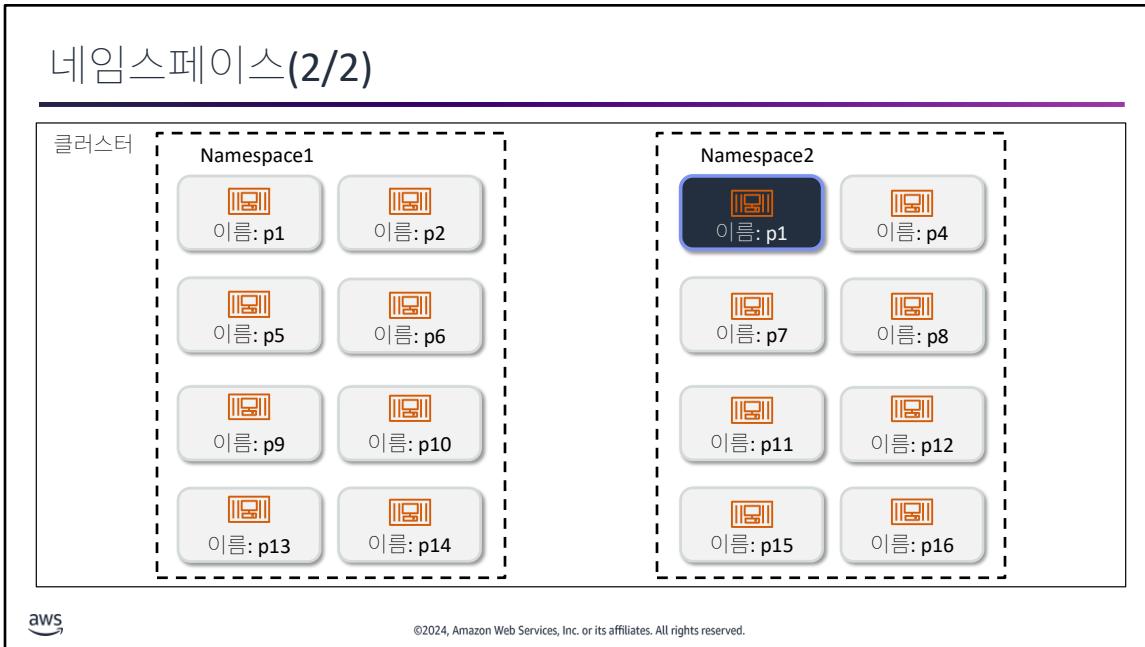
©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

Kubernetes의 또 다른 기본 객체는 **Namespace**입니다.

일반적으로 동일한 클러스터 안의 리소스 이름들은 고유해야 합니다. 이 경우에는 다른 Pod가 이미 그 이름을 갖고 있으므로 강조 표시된 Pod의 이름을 'p1'으로 지정할 수 없습니다.

네임스페이스(2/2)



| 수강생용 노트

하지만 **Namespace**에서는 동일한 물리적 클러스터가 지원하는 가상 클러스터를 생성할 수 있습니다. 물리적 클러스터는 서로 다른 **Namespace**에 있는 한 동일한 이름의 리소스를 지닐 수 있습니다.

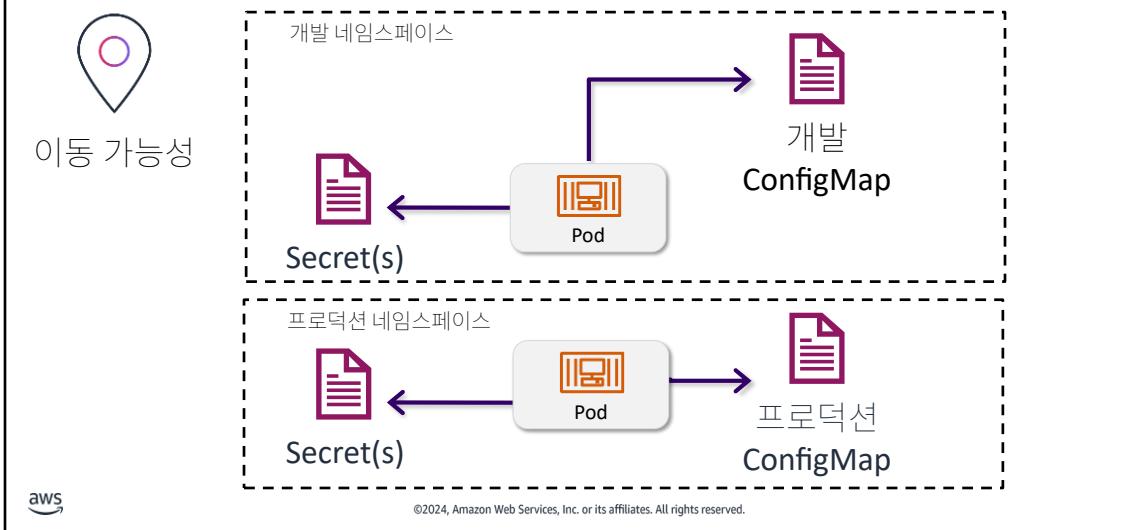
네임스페이스는 여러 팀이나 프로젝트에서 동일한 클러스터를 사용하는 경우에 특히 유용합니다. **Kubernetes** 네임스페이스를 멀티 테넌트 클러스터의 한 구성 요소로 사용할 수 있습니다. **Kubernetes** 역할 기반 액세스 제어(**RBAC**)와 결합하면 네임스페이스는 더 강력한 다중 팀, 멀티 테넌트 클러스터를 제공합니다. 예를 들어 동일한 **Kubernetes** 클러스터 내에 별도의 논리적 클러스터를 제공하기 위한 네임스페이스 ‘app1-prod’, ‘app1-test’, ‘app1-dev-branch-x’ 및 ‘app1-dev-branch-y’가 있을 수 있습니다. 네임스페이스 내의 객체 이름은 폴더의 파일과 마찬가지로 다른 네임스페이스의 이름과 겹칠 수 있습니다. 그러므로 테넌트는 다른 테넌트가 수행하는 작업을 고려하지 않고도 리소스 이름을 지정할 수 있습니다.

다중 테넌트 클러스터 모범 사례에 대한 자세한 내용은 **Kubernetes** 설명서의 ‘**Multi-tenancy**(<https://kubernetes.io/docs/concepts/security/multi-tenancy/>)’를 참조하십시오.

네임스페이스는 **Pod**가 어떤 노드에서 실행되는지에 영향을 주지 않습니다. 그렇게 되지 않도록 특정 예약 규칙을 설정하지 않는 한 여러 네임스페이스의 **Pod**가 동일한 노드에 존재할 수 있습니다.

참고: 가상 클러스터를 생성할 수 있음을 언급할 때 이는 ‘가상 Kubernetes 클러스터(또는 vCluster)’를 의미하지는 않는다는 점에 유의하십시오. 전혀 다른 주제입니다.

ConfigMap과 Secret



| 수강생용 노트

앞서 **PodSpec**을 사용하여 **Pod**에서 인스턴스화해야 할 컨테이너 이미지를 정의하는 방법에 대해 살펴보았습니다. 이동성 모범 사례 따르기: 개발이나 프로덕션을 위해 컨테이너 이미지를 변경하지 않고도 **Pod**가 일관되게 실행될 수 있는지 확인하십시오. 이 작업은 애플리케이션 코드에서 구성 데이터를 분리함으로써 수행할 수 있습니다.

Kubernetes에서는 **ConfigMap**을 사용하여 수행할 수 있습니다. **ConfigMap**은 데이터를 키-값 쌍으로 저장하는 API 객체입니다. 그러면 **Pod**가 **ConfigMap**의 키 값을 사용할 수 있습니다. **ConfigMap** 데이터는 환경 변수, 명령줄 인수 또는 볼륨의 구성 파일로 사용할 수 있습니다. **ConfigMap**은 기밀 데이터 이외의 데이터만 저장하도록 되어 있습니다.

Secret은 민감한 기밀 정보가 포함된 Kubernetes 객체입니다. AWS 자격 증명과 같은 모든 기밀 데이터는 **Secret**에 저장되어야 합니다. **ConfigMap** 객체와 마찬가지로 **Secret**에 보관된 데이터는 **Pod**에 탑재되거나 환경 변수로 노출되는 데이터 볼륨을 통해 액세스할 수 있습니다. **Secret** 객체를 보호하려면 클러스터에 대한 추가 구성이 필요합니다.

ConfigMap 예 및 사용

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  labels:
    app.kubernetes.io/name: fluentbit
data:
  fluent-bit.conf: |
    [SERVICE]
    Parsers_File parsers.conf
    [INPUT]
    Name tail
    Tag kube./*
    Path var/log/containers/*.log
    Parser docker
    DB /var/log/flb_kube.db
  ...
  ...
```

```
apiVersion: v1
kind: Pod
metadata:
  name: Pod-example
spec:
  containers:
    - name: Pod-name
      image: Fluent/fluent-bit
      volumeMounts:
        - name: fb-config
          mountPath: "/config"
          readOnly: true
  volumes:
    - name: fb-config
      configMap:
        name: fluent-bit-config
```



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

| 수강생용 노트

표시된 것처럼 ConfigMap 및 Pod 예는 YAML 파일에 정의됩니다. Pod 예에서는 Pod가 읽기 전용 볼륨을 통해 ConfigMap 객체의 데이터를 사용하는 방법을 보여 줍니다. ConfigMap은 기밀이 아닌 데이터를 키-값 쌍에 저장합니다.

Secret 예 및 사용

```
apiVersion: v1
kind: Secret
metadata:
  name: accountLogin
type: Opaque
data:
  username: bXl1c2vybmFtZQo=
  password: bXlwYXNzd29yZAo=
```

```
apiVersion: v1
kind: Pod
metadata:
  name: Pod-example
spec:
  containers:
    - name: logging-container
      image: fluent/fluent-bit
      volumeMounts:
        - name: login-credentials
          mountPath: "/credentials"
          readOnly: true
  volumes:
    - name: login-credentials
      secret:
        secretName: accountLogin
```



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

| 수강생용 노트

표시된 것처럼 Secret 및 Pod 예는 YAML 파일에 정의됩니다. Pod 예에서는 Pod가 읽기 전용 볼륨을 통해 Secret 객체의 데이터를 사용하는 방법을 보여 줍니다.

Secret은 민감한 정보를 키-값 쌍에 저장합니다. Secret의 **data** 필드에 있는 모든 키의 값이 base64로 인코딩된 문자열이어야 합니다. 특정 시나리오에서는 **data** 필드 대신 **stringData** 필드를 사용해야 할 수 있습니다. 이 필드를 사용하여 Secret YAML 매니페스트에 base64로 인코딩되지 않은 문자열을 바로 추가할 수 있습니다. **stringData** 필드의 데이터는 Secret을 생성하거나 업데이트할 때 인코딩됩니다.

워크로드가 Pod를 생성 및 관리



Job 및 CronJob

- **Job** 워크로드는 Pod를 완료 시까지 한번 실행합니다.
- **CronJob** 워크로드는 작업을 특정 시간 또는 반복 일정으로 예약합니다.

DaemonSet

- **DaemonSet**는 각 노드에서 Pod를 실행합니다.
- 노드 에이전트 소프트웨어(로그 집계자)에 사용됩니다.
- AWS Fargate에서는 지원되지 않습니다.

StatefulSet

- **StatefulSet** 워크로드는 고유한 식별자 및 순서가 지정된 수명 주기로 Pod를 실행합니다.
- **StatefulSet**는 상태 앱 구성 요소에 대해 PersistentVolume과 잘 연동됩니다.

Deployment 및 ReplicaSet

- **ReplicaSet** 워크로드는 특정 수의 Pod를 실행합니다.
- **Deployment**은 ReplicaSets에 선언형 업데이트를 제공합니다.

aws

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

| 수강생용 노트

Kubernetes 워크로드는 하나 이상의 Pod를 생성하고 해당 Pod의 수명 주기를 관리합니다. 자세한 설명은 Kubernetes 설명서의 'Workloads' (<https://kubernetes.io/docs/concepts/workloads/>)를 참조하십시오.

참고: Pod를 0개 생성하는 워크로드를 생성할 수 있습니다. 그러나 1개 이상이 일반적인 사용 사례입니다.

베어 Pod보다는 워크로드를 생성하는 것이 좋습니다. 작업의 경우 '베어 Pod'와 작업의 차이점이 <https://kubernetes.io/docs/concepts/workloads/controllers/job/#bare-Pods>에 설명되어 있습니다.

각 종류의 워크로드에는 해당 워크로드 종류의 객체 동작을 제어하는 컨트롤러가 Kubernetes 제어 영역에 있습니다. 이러한 워크로드 컨트롤러 동작은 관리 중인 Pod가 삭제되는 경우 교체 Pod를 자동 생성하는 등의 필수 기능을 관리합니다. 이는 Kubernetes 컨테이너 오케스트레이션의 가장 중요하고 강력한 기능 중 하나입니다.

Job 및 CronJob

```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
  labels:
    app.kubernetes.io/name: my-job
spec:
  template:
    metadata:
      labels:
        app.kubernetes.io/name: my-Pod
    spec:
      containers:
        - name: my-container
          image: busybox
          command:
            - date
```

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: my-cron-job
spec:
  schedule: '*/1 * * * *'
  jobTemplate:
    metadata:
      labels:
        app.kubernetes.io/name: my-job
    spec:
      template:
        metadata:
          labels:
            ...
        spec:
          containers:
            - ...
```



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

| 강사용 노트

| 참조: Kubernetes Domain Starter: <https://catalog.us-east-1.prod.workshops.aws/workshops/5f769b4f-8b4a-4308-81df-4af63687fa44/en-US/021-running-at-scale#workload-resource-types>

또는 Kubernetes 설명서: <https://kubernetes.io/docs/concepts/workloads/>

| 수강생용 노트

이 예의 CronJob을 사용하면 작업이 1분마다 실행됩니다.

자세한 내용은 다음 페이지를 참조하십시오.

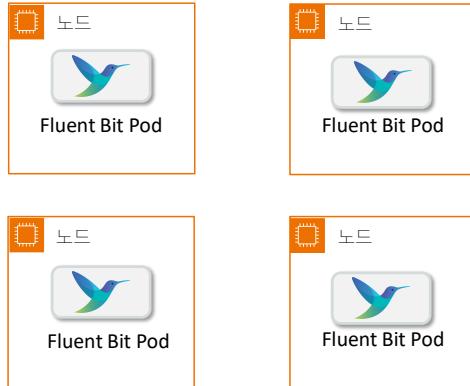
- Kubernetes Domain Starter의 ‘워크로드 리소스 유형’(<https://catalog.us-east-1.prod.workshops.aws/workshops/5f769b4f-8b4a-4308-81df-4af63687fa44/en-US/021-running-at-scale#workload-resource-types>)
- Kubernetes 설명서의 ‘Workloads’
(<https://kubernetes.io/docs/concepts/workloads/>)

DaemonSets: 노드 에이전트용 워크로드

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  labels:
    app: fluent-bit
spec:
  selector:
    matchLabels:
      app: fluent-bit
  template:
    metadata:
      labels:
        app: fluent-bit
    spec:
      containers:
        - name: fluent-bit
          image:"cr.fluentbit.io/fluent/fluent-bit:2.0.5"
[...]
```



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



~ALT Text

~노드: Fluent Bit Pod가 연결된 노드 클러스터입니다.

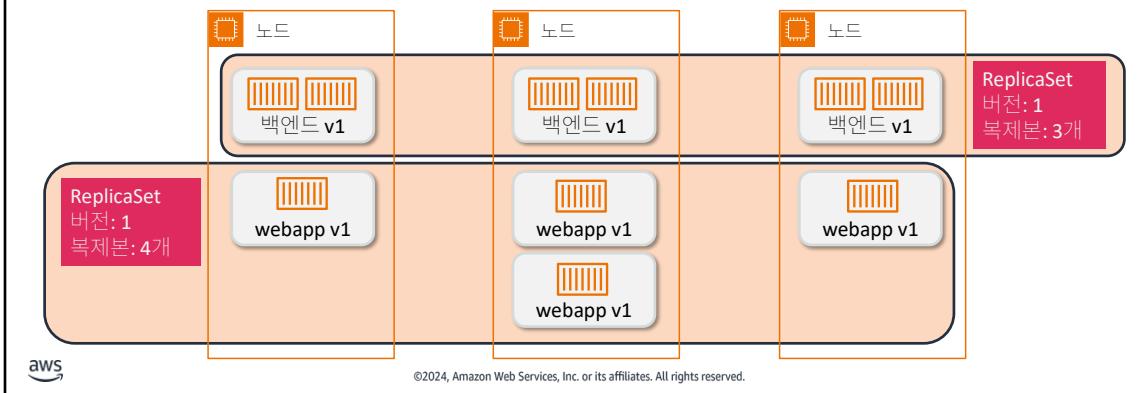
| 수강생용 노트

DaemonSet는 모든 노드가 요청된 **Pod**의 사본을 갖도록 합니다. 이는 애플리케이션 코드를 실행하는 **Pod**에 일반적인 지원 기능(예: 보안, 로깅, 모니터링, 백업)을 제공하려고 하는 경우에 유용합니다. 예를 들어, 노드에서 로그를 수집해야 하는 경우 **DaemonSet**를 실행할 수 있습니다. 이렇게 하면 **Fluent Bit**와 같은 로깅 대문 **Pod**가 모든 노드에서 실행됩니다. **DaemonSet**를 삭제하면 노드 전반에서 생성된 모든 **Pod**도 삭제됩니다.

DaemonSet 대신 **init** 스크립트를 사용하는 경우처럼 다른 방식으로 이러한 대문을 실행할 수 있습니다. 하지만 **DaemonSet**를 사용하면 다음을 수행할 수 있습니다.

- 다른 애플리케이션처럼 대문에 대한 로그를 모니터링하고 관리합니다.
- 비슷한 **Kubernetes** 도구 세트 및 API 클라이언트의 에코시스템을 유지 관리합니다.
- 애플리케이션 컨테이너에서 대문을 구분합니다.
- 불가피한 상황(예: 노드 장애) 때문에 삭제된 **Pod**를 대신합니다.

ReplicaSet: 스테이트리스 앱 복제본용 워크로드



~개발자 노트

~ReplicaSet1: 2개의 복제본이 2개의 별도 노드에 있는 ReplicaSet입니다.

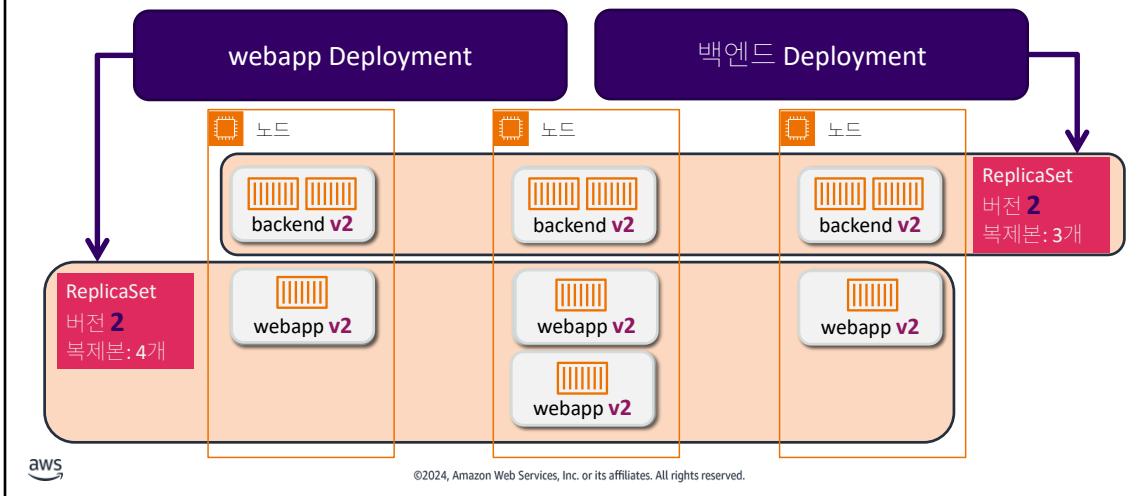
~ReplicaSet2: 4개의 복제본이 3개의 별도 노드에 있는 두 번째 ReplicaSet입니다.

|수강생용 노트

Pod의 휘발성을 이해했으니, 중복성과 복원력에 대해서도 생각해 보십시오. Pod 하나가 아니라 Pod의 복제본이 여러 개 필요한 경우가 종종 있을 것입니다.

ReplicaSet는 특정 시점에 특정 개수의 Pod 복제본이 실행되도록 합니다. 사용자가 ReplicaSet를 생성하고 관리할 수 있지만, 배포를 통해 ReplicaSet를 관리하는 것이 더 일반적입니다.

Deployment 및 ReplicaSet(앱 업그레이드용)



~Dev notes

~Deployment: A deployment is commonly used to manage ReplicaSets.

| 강사용 노트 및 애니메이션

| <1> 한 번 클릭하면 앱 업그레이드 과정이 표시됩니다.

| 수강생용 노트

Deployment는 ReplicaSet를 소유하고 관리합니다. 사용자가 해당 Deployment에 원하는 상태를 기술합니다. 그러면 이 Deployment가 실제 상태를 제어된 비율로 원하는 상태로 변경합니다. Deployment를 사용하면 Pod가 생성하는 ReplicaSet를 관리하는 것에 대해 걱정할 필요가 없습니다. Deployment를 여러 개의 Pod를 한 번에 명령적 방식이 아닌 선언적 방식으로 조작하는 방법이라고 생각하십시오.

이 예에서는 Deployment가 복제본을 버전 1에서 버전 2로 업데이트하는 데 사용됩니다.

예: Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
  labels:
    app: webapp
spec:
  replicas: 4
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
```

```
spec:
  containers:
    - name: webapp
      image: webapp:22.04
      command: ["echo"]
      args: ["Hello World"]
      ports:
        - containerPort: 80
```



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

| 수강생용 노트

다른 **Kubernetes** 객체와 마찬가지로 **Deployment**는 선언형 형식으로 작성됩니다. 이 예에서는 **4개의 webapp 이미지 복제본**을 배포합니다. **template** 필드의 **app: ubuntu** 레이블이 **selector** 레이블과 일치합니다. **Deployment**은 레이블을 사용하여 제어할 **Pod**를 식별합니다. 레이블이 일치하지 않으면 **Deployment**가 대상 **Pod**를 관리하지 못합니다.

예: StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: "mysql"
  replicas: 2
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
```

```
spec:
  containers:
    - name: mysql
      image: mysql
      env:
        - name: MYSQL_ALLOW_EMPTY_PASSWORD
          value: "true"
```

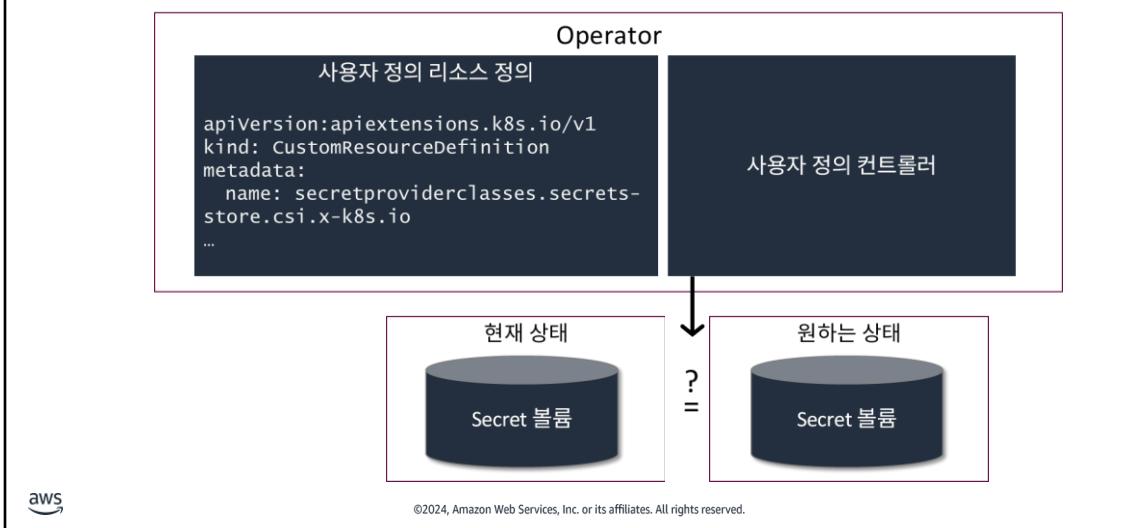


©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

다른 **Kubernetes** 객체와 마찬가지로 **StatefulSet**는 선언형 형식으로 작성됩니다. 이 예에서는 **2개의 mysql** 이미지 복제본을 배포합니다. **template** 필드의 **app: mysql** 레이블이 **selector** 레이블과 일치합니다. **StatefulSet**은 레이블을 사용하여 제어할 **Pod**를 식별합니다. 레이블이 일치하지 않으면 **StatefulSet**가 대상 **Pod**를 관리하지 못합니다.

사용자 정의 리소스



~ALT text

~**Operator**: 사용자 정의 리소스 정의 및 사용자 정의 컨트롤러가 있는 작업자입니다.

~원하는 상태: **Secret** 볼륨이 있는 원하는 상태입니다.

~현재 상태: **Secret** 볼륨이 있는 현재 상태입니다.

| 수강생용 노트

Kubernetes가 정의하는 리소스(예: Pod, 배포) 이외에 사용자 정의 리소스도 생성할 수 있습니다. 사용자 정의 리소스 정의(CRD)를 사용하여 사용자 정의 리소스를 정의합니다.

어떤 유형의 CRD를 정의할 수 있습니까? 사용자 정의 리소스는 데이터베이스와 같은 새 객체일 수도 있고 Kubernetes 기본 리소스의 조합일 수도 있습니다. 이러한 사용자 정의 리소스는 Kubernetes API를 확장할 수 있습니다. Kubernetes 리소스와 마찬가지로 사용자 정의 리소스는 사용자 정의 컨트롤러로 제어할 수 있습니다. 사용자 정의 컨트롤러는 클러스터의 제어 영역에서 실행되지 않습니다. 대신, 클러스터의 데이터 영역의 Pod에서 실행됩니다.

Operator는 CRD 및 사용자 정의 컨트롤러를 사용하여 애플리케이션 태스크를 관리하고 자동화하는 Kubernetes의 소프트웨어 확장입니다. Kubernetes 기본 객체를 수동으로 업데이트하지 않고 **operator**를 사용하는 것이 좋습니다.



Running Containers on Amazon EKS

지식 확인



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

지식 확인 1

Kubernetes에서 복원력 크기 조정의 기본 단위는 무엇입니까?

보기	응답
A	컨테이너
B	Pod
C	태스크
D	ReplicaSet

지식 확인 1: 정답은 B입니다.

Kubernetes에서 복원력 크기
조정의 기본 단위는
무엇입니까?

보기	응답
A	컨테이너
B	Pod
C	태스크
D	ReplicaSet

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

|수강생용 노트

Pod.

이유는 무엇입니까?

Pod는 배포, 확장, 복제를 위한 Kubernetes 내 기본 빌딩 블록입니다.

지식 확인 2

Kubernetes 서비스란 무엇입니까?

보기	응답
A	여러 Pod의 논리적 모음과 여기에 액세스하는 수단
B	여러 Pod의 논리적 모음과 이를 예약하는 수단
C	모든(또는 일부) 노드가 Pod의 복사본을 실행하도록 지원하는 도구
D	특정 시간에 실행되는 안정적인 Pod 복제본 집합을 유지 관리하는 도구



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

지식 확인 2: 정답은 A입니다.

Kubernetes 서비스란 무엇입니까?

보기	응답
A	여러 Pod의 논리적 모음과 여기에 액세스하는 수단
B	여러 Pod의 논리적 모음과 이를 예약하는 수단
C	모든(또는 일부) 노드가 Pod의 복사본을 실행하도록 지원하는 도구
D	특정 시간에 실행되는 안정적인 Pod 복제본 집합을 유지 관리하는 도구

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

| 수강생용 노트

여러 Pod의 논리적 모음과 여기에 액세스하는 수단

이유는 무엇입니까?

Kubernetes에서 서비스란 여러 Pod의 논리적 모음과 여기에 액세스하는 수단입니다. 서비스는 사용 가능한 Pod의 집합으로 계속 업데이트되므로 Pod가 다른 Pod를 추적할 필요가 없습니다.

활동



이 활동에서는 다음 과제를 수행합니다.

- 수행하려는 작업을 제어하는 구성을 선택하십시오.

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

| 강사용 노트

| 수강생용 노트

활동: kubectl 명령 1

kubectl 명령: \$ kubectl create namespace my-ns

출력:



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

활동: kubectl 명령 1 출력

kubectl 명령: \$ kubectl create namespace my-ns

출력: namespace/my-ns created



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

활동: kubectl 명령 2

kubectl 명령: \$ kubectl get namespace

출력:



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

활동: kubectl 명령 2 출력

kubectl 명령: \$ kubectl get namespace

출력:

NAME	STATUS	AGE
default	Active	9m12s
kube-node-lease	Active	9m14s
kube-public	Active	9m14s
kube-system	Active	9m14s
my-ns	Active	5s



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

활동: kubectl 명령 3

kubectl 명령: \$ kubectl apply -f nginx-deploy.yaml

출력:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  namespace: my-ns
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:...
```



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

활동: kubectl 명령 3 출력

kubectl 명령: \$ kubectl apply -f nginx-deploy.yaml

출력: deployment.apps/nginx-deploy created



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

활동: kubectl 명령 4

kubectl 명령: \$ kubectl get Pods

출력:



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

활동: kubectl 명령 4 출력

kubectl 명령: \$ kubectl get Pods

출력: No resources found in default namespace



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

| 수강생용 노트:

kubectl get 명령은 리소스를 나열합니다. 기본적으로 기본 네임스페이스에 있는 리소스를 나열합니다. 이 경우에는 기본 네임스페이스의 **Pod**를 나열합니다. 하지만 이 **Pod**는 **my-ns** 네임스페이스로 배포되었기 때문에 사용자에게는 **Pod**가 없습니다.

활동: kubectl 명령 5

kubectl 명령: \$ kubectl get Pods -n my-ns

출력:



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

활동: kubectl 명령 5 출력

kubectl 명령: \$ kubectl get Pods -n my-ns

출력:

NAME	READY	STATUS	RESTARTS	AGE
nginx-deploy-7f796fdfd9-244vs	0/1	Pending	0	4s
nginx-deploy-7f796fdfd9-1v7ts	0/1	Pending	0	4s



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

| 수강생용 노트:

kubectl get 명령은 리소스를 나열합니다. **-n** 플래그를 추가함에 따라 이 명령은 **my-ns** 네임스페이스에 있는 **Pod**를 나열합니다. 배포를 통해 생성된 **Pod 2개**로, 모두 보류 상태입니다.

모듈 요약



이 모듈에서 학습한 내용:

- 컨테이너 오케스트레이션 도구
 - 클러스터의 모든 컨테이너를 관리하는데 도움이 되는 도구
- **Pod**
 - 하나 이상의 컨테이너로 구성된 그룹
- **Kubernetes** 제어 영역
 - 제어 영역 노드 및 `etcd` 지속성 계층을 포함함
- 데이터 영역
 - Pod를 실행하는 노드를 포함함
- `kubectl`
 - Kubernetes API 서버와 통신하기 위한 CLI

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

| 강사용 노트

| 수강생용 노트



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.