



Running Containers on Amazon EKS

모듈 6: Amazon EKS에서 네트워킹 관리

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

| 수강생용 노트

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



Running Containers on Amazon EKS

모듈 6 개요

- 복습: AWS의 네트워킹
- Amazon EKS의 통신
- Pod 수준 보안 개선
- 서비스를 사용한 로드 밸런싱
- 지식 확인



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



Running Containers on Amazon EKS

복습: AWS의 네트워킹



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

AWS의 네트워킹 개요



Amazon EKS와 통신

IP 공간 관리

통신 인프라 관리



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

~ SEQUOIA CHANGES: Removed sentence about App Mesh from slide notes

| 수강생용 노트

이 모듈에서는 **Amazon Elastic Kubernetes Service(Amazon EKS)**에서 다양한 유형의 통신이 어떻게 작동하는지를 배웁니다. 효과적인 통신을 촉진하기 위해 **AWS**는 여러 구성 요소에 **IP 주소**를 할당합니다. 따라서 **IP 공간 관리**가 중요한 이유와 **IP 공간**을 관리하고 절약하는 방법을 이해해야 합니다.

AWS의 네트워킹 기본 구성 요소

- Amazon Virtual Private Cloud(Amazon VPC)
- 서브넷
- 보안 그룹



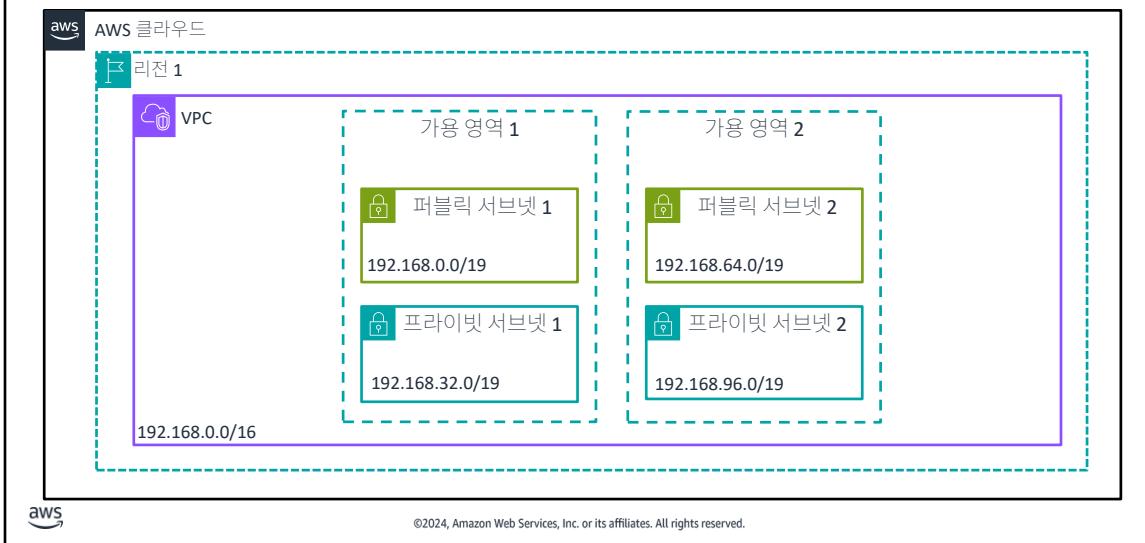
©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

AWS는 네트워크 아키텍처, 네트워크 연결 및 애플리케이션 전송에 도움이 되는 많은 서비스를 제공합니다. **Amazon EKS**를 논의할 때는 다음 네트워킹 구성 요소가 특히 중요합니다.

- **Amazon VPC** - Amazon EKS 클러스터를 배포하는 위치입니다.
- 서브넷 - 클러스터를 생성할 때 지정합니다. 서브넷은 Amazon EKS가 제어 영역과 노드 간 통신에 사용되는 탄력적 네트워크 인터페이스를 배치하는 위치를 결정합니다.
- 보안 그룹 - Amazon EKS 제어 영역과 노드 간에 트래픽이 흐르는 방식을 결정합니다.

Amazon VPC 기본 사항



~dev notes

~Alt text for basic-VPC-diagram-1: 2개의 가용 영역에 걸쳐 있는 Amazon VPC입니다. 각 가용 영역에 퍼블릭 및 프라이빗 서브넷이 하나씩 있습니다.

|수강생용 노트

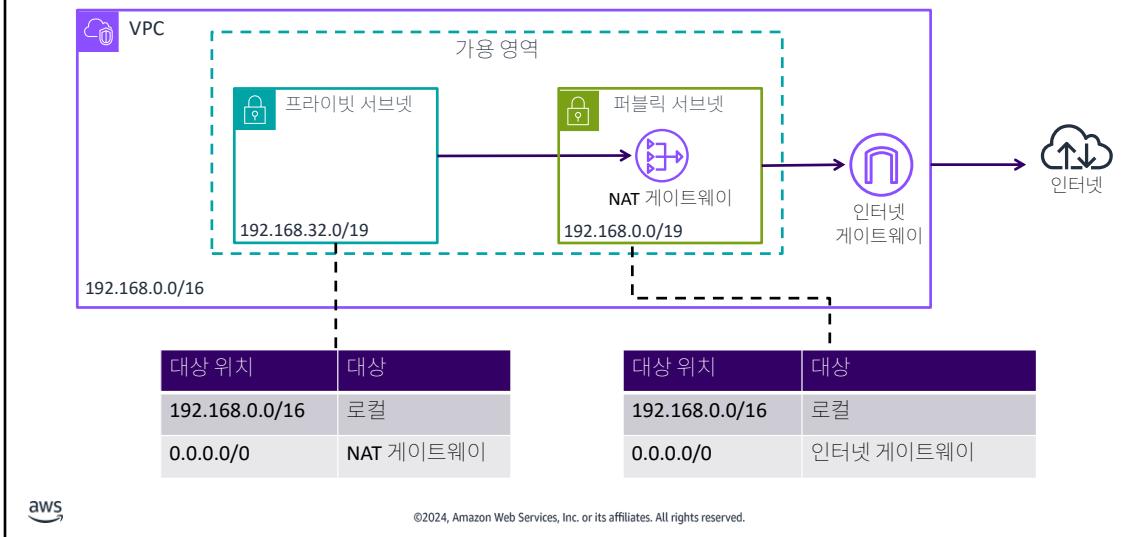
Amazon VPC는 가상 데이터 센터라고 생각하면 됩니다. 사용자가 정의하는 가상 네트워크에서 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스와 같은 AWS 리소스를 시작할 수 있는 AWS 클라우드의 격리된 공간입니다.

Amazon VPC는 단일 AWS 리전에서 생성되지만 해당 리전 내의 모든 가용 영역에 걸쳐 있습니다. Virtual Private Cloud(VPC)을 생성할 때는 VPC의 IPv4 주소 범위를 Classless Inter-Domain Routing(CIDR) 블록으로 지정해야 합니다. 예를 들어 192.168.0.0/16과 같이 지정합니다. 이 주소 범위가 VPC의 기본 CIDR 블록입니다. VPC에 IPv6 CIDR 블록 하나를 할당하고 서브넷에 IPv6 CIDR 블록을 여러 개 할당할 수 있습니다.

VPC를 생성한 후에는 각 가용 영역에 하나 이상의 서브넷을 추가할 수 있습니다.

서브넷을 생성할 때는 VPC CIDR 블록의 하위 집합에 속하는 CIDR 블록을 해당 서브넷에 대해 지정합니다. 각 서브넷은 하나의 가용 영역 내에 모두 상주해야 하며, 다른 영역으로 확장할 수 없습니다.

퍼블릭 및 프라이빗 서브넷



~dev notes

~Alt text for subnets: 프라이빗 서브넷의 트래픽이 퍼블릭 서브넷의 NAT 게이트웨이를 통과합니다. 트래픽은 NAT 게이트웨이에서 인터넷 게이트웨이를 통해 인터넷으로 흐릅니다.

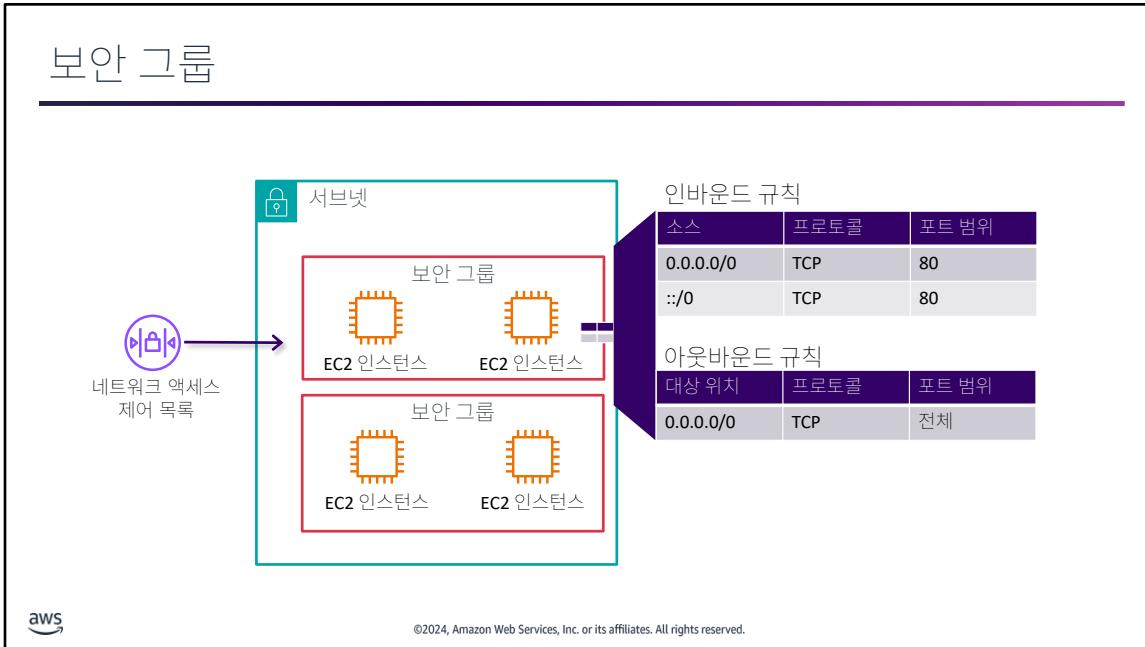
|수강생용 노트

서브넷은 퍼블릭이거나 프라이빗일 수 있습니다. 서브넷의 트래픽이 인터넷 게이트웨이로 라우팅되는 서브넷을 퍼블릭 서브넷이라고 하고, 그렇지 않은 서브넷을 프라이빗 서브넷이라고 합니다.

프라이빗 서브넷의 인스턴스는 네트워크 주소 변환(NAT) 게이트웨이를 사용하여 인터넷 또는 다른 AWS 서비스에 연결할 수 있습니다. 그러나 프라이빗 인스턴스이기 때문에 인터넷에서는 프라이빗 인스턴스와의 연결을 시작할 수 없습니다.

VPC에서 트래픽이 이동할 수 있는 위치를 정의하려면 두 표에 표시된 대로 라우팅 테이블을 사용하십시오. 라우팅 테이블에는 서브넷 또는 게이트웨이의 네트워크 트래픽을 보낼 위치를 결정하는데 사용되는 경로라는 규칙 집합이 포함되어 있습니다.

보안 그룹



| 수강생용 노트

보안 그룹은 인스턴스에 대한 인바운드 및 아웃바운드 트래픽을 제어하는 가상 방화벽 역할을 합니다. 보안 그룹은 서브넷 수준이 아니라 인스턴스 수준에서 작동하므로 인스턴스의 다양한 인터페이스에 다양한 보안 그룹을 할당할 수 있지만, **Amazon EKS**에서는 동일한 인스턴스의 모든 인터페이스가 동일한 보안 그룹을 사용합니다. 각 보안 그룹에 대해 인스턴스에 대한 인바운드 트래픽을 제어하는 규칙과 아웃바운드 트래픽을 제어하는 별도의 규칙 집합을 추가합니다.

보안 그룹과 유사한 규칙으로 네트워크 액세스 제어 목록(**네트워크 ACL**)을 설정하면 **Amazon VPC**에 보안 계층 하나를 더할 수 있습니다. 네트워크 **ACL**은 이 과정의 범위가 아니므로 더 이상 다루지 않겠습니다. 네트워크 **ACL**에 대해 자세히 알아보려면 **Amazon VPC** 사용 설명서의 '**네트워크 ACL**' (<https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html>)을 참조하십시오.



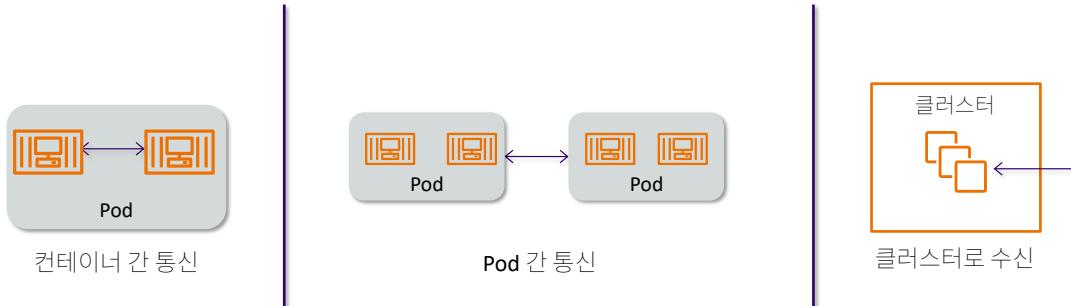
Running Containers on Amazon EKS

Amazon EKS의 통신



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

Amazon EKS의 통신 유형



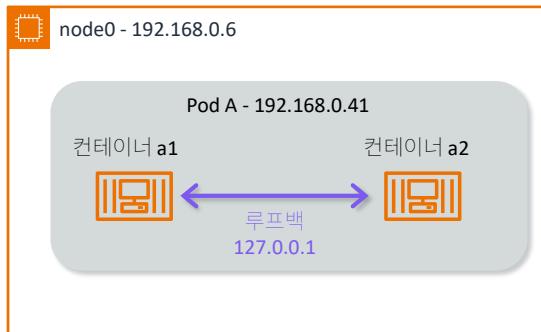
©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

분산 컨테이너 오케스트레이션 시스템용 네트워킹 솔루션을 설계할 때는 다음과 같은 다양한 통신 라인을 고려해야 합니다.

- 컨테이너 간의 **Pod** 간 통신
- 동일한 노드에 있는 **Pod** 또는 다른 노드에 있는 **Pod** 간 통신
- 클러스터 외부로부터 수신 연결

Kubernetes Pod 내 통신



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

| 수강생용 노트

Kubernetes는 다음 측면에서 가상 머신(**VM**)이나 물리적 호스트와 동일한 방식으로 **Pod**를 처리합니다.

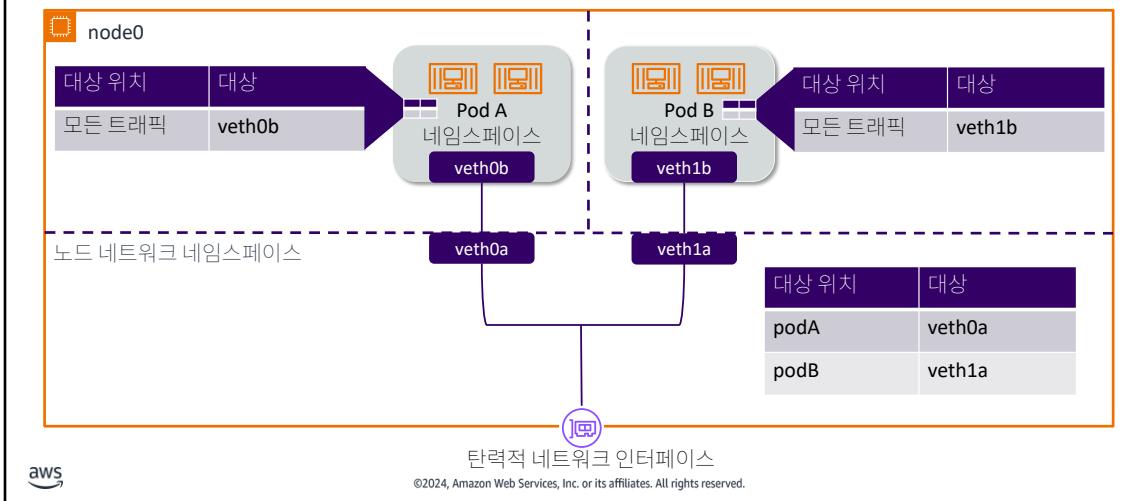
- 포트 할당
- 네트워킹
- 이름 지정
- 서비스 검색
- 로드 밸런싱
- 애플리케이션 구성
- 컨테이너화되지 않은 워크로드에서 마이그레이션

Pod는 **Kubernetes**의 기본 빌딩 블록이므로 **Kubernetes**는 네트워크의 IP 주소를 각 애플리케이션 컨테이너가 아니라 **Pod**에 적용합니다.

Pod의 컨테이너는 네트워크 네임스페이스를 공유하며 **localhost**(즉, 루프백 주소 **127.0.0.1**)를 사용하여 서로 통신할 수 있습니다. 이는 **Pod** 내에 루프백 인터페이스가 있는 예에 나와 있습니다. **Pod**의 컨테이너에서 프로세스 간 통신은 루프백 주소와 지정된 **TCP** 또는 **UDP** 포트 번호를 사용하는 단일 컨테이너의 프로세스 간에 사용되는 것과 동일한 방식을 따릅니다.

Kubernetes 네트워킹에서 컨테이너가 식별하는 **Pod IP** 주소는 네트워크의 모든 엔터티에 대한 **IP** 주소와 동일합니다. 이 예에서는 **Pod**에 IP 주소 **192.168.0.41**이 할당되었습니다. **Pod**의 컨테이너에서 프로세스는 다른 **Pod** 또는 다른 네트워크 엔터티(예: 가상 머신)와 통신하기 위해 이러한 **Pod IP** 주소를 **TCP/IP** 또는 **UDP/IP** 통신을 위한 적절한 포트 번호를 함께 사용합니다. **Pod**의 컨테이너(프로세스)에 대한 인바운드 통신은 적절한 프로토콜 및 포트 번호와 함께 **Pod**의 **IP** 주소로 전달되어야 합니다. 이러한 인바운드 통신에서 포트 번호는 해당 포트에서 수신 대기하는 프로세스를 실행하는 컨테이너를 효과적으로 선택합니다. **Pod**의 컨테이너는 동일한 네트워크 네임스페이스를 공유하므로 동일한 **Pod IP** 주소를 공유할 뿐만 아니라 **TCP** 및 **UDP** 포트 번호 공간도 공유합니다. 따라서 동일한 **Pod**의 컨테이너 간에 포트 충돌이 발생하지 않도록 주의해야 합니다. 이는 일반적으로 올바른 **Pod** 설계의 문제입니다. 즉, **Pod**의 각 컨테이너는 컨테이너별로 서로 다른 소프트웨어를 사용하거나 적어도 포트 번호 사용이 전체 **Pod**에서 고유하도록 서로 다른 구성 사용하여 고유한 목적을 가져야 합니다.

Kubernetes 호스트 내 통신(1/2)



| 수강생용 노트

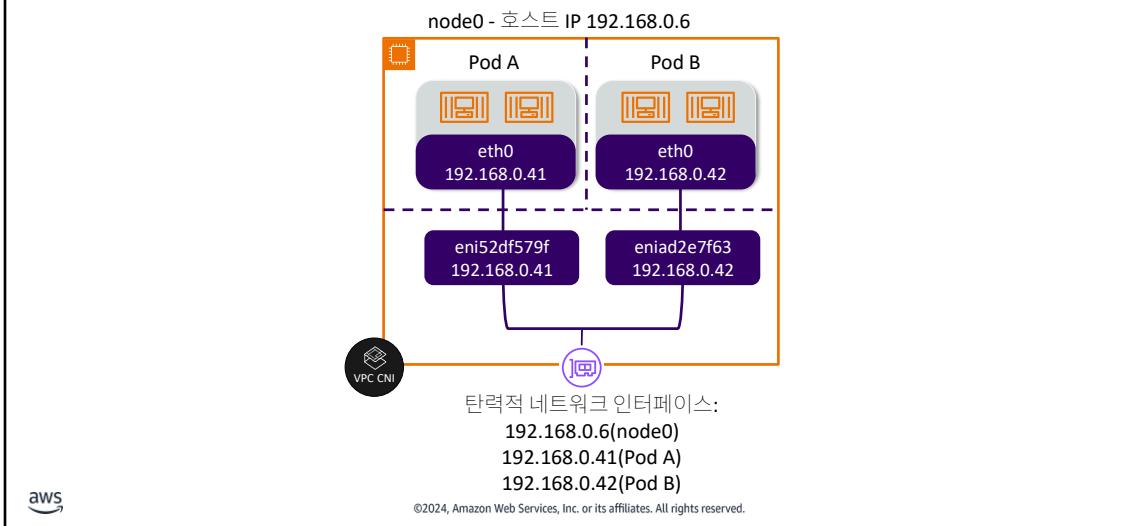
동일한 Pod의 컨테이너는 공유 네트워크 네임스페이스를 통해 통신합니다. Pod 간 통신은 Pod가 네트워크 네임스페이스를 공유하지 않기 때문에 다르게 작동합니다.

각 Pod에 네트워크 네임스페이스가 있을 뿐만 아니라 호스트 노드에도 네트워크 네임스페이스가 있습니다. 각 네트워크 네임스페이스에는 자체 라우팅 테이블이 있습니다. 각 Pod 네임스페이스와 호스트 네임스페이스는 한 쌍의 Linux 가상 이더넷(veth) 디바이스를 통해 연결됩니다. 각 veth 쌍은 기본 호스트 네트워크 네임스페이스와 Pod 네트워크 네임스페이스 간에 터널을 생성합니다. 이 예에서는 각각 자체 veth가 있는 두 개의 Pod를 보여줍니다. Pod A에는 veth0b가 있고, Pod B에는 veth1b가 있습니다. 노드 네트워크 네임스페이스에 해당 veth가 표시되어 있습니다. 즉, 노드의 veth0a는 Pod A의 쌍을 이루는 veth0b로 터널링하고, 노드의 veth1a는 Pod B의 쌍을 이루는 veth1b로 터널링합니다. 노드에 29개의 Pod가 있다면 veth 쌍은 29개가 됩니다. Pod당 1개의 veth가 있고 노드의 29개 veth가 각 Pod와 통신하기 때문입니다. 노드에 737개의 Pod가 있다면 veth 쌍은 737개가 됩니다(Pod당 1개의 veth가 있고 노드에 737개 Pod가 있음).

여기에 사용된 **veth**의 이름은 **Pod**별 및 노드 라우팅 테이블에 대한 이해를 돋기 위해 인위적으로 고유하게 명명된 것입니다. **Pod** 내 통신 항목에서 설명했듯이 동일한 **Pod**에 있는 컨테이너 간 통신은 해당 컨테이너의 적절한 프로세스에 대한 포트 번호 기반 방향에 의존합니다. 각 **Pod**의 기본 경로는 노드 네트워크 네임스페이스의 해당 **veth**로 터널링되는 **Pod**의 **veth**(예: **Pod A**의 경우 **veth0b**, **Pod B**의 경우 **veth1b**)를 통해 다른 대상 위치(동일한 클러스터의 다른 **Pod** 또는 다른 위치)로 향하는 트래픽을 전달합니다. 이 예에서는 노드 네트워크 네임스페이스 내에서 노드 네트워크 네임스페이스의 라우팅 테이블을 더 쉽게 식별할 수 있도록 **veth0a** 및 **veth1a**라는 이름을 사용합니다. 노드의 라우팅 테이블에 있는 대상 위치 ‘**podA**’ 및 ‘**podB**’는 실제로 **Pod**의 IP 주소로 나타나며, 대상 **veth**는 이러한 인위적 이름 **veth0a** 및 **veth1a** 대신 Container Network Interface(CNI)에서 할당한 실제 **veth** 인터페이스 이름을 사용합니다.

호스트에서 **Pod** 간 통신은 이러한 **veth** 쌍 터널과 노드의 라우팅 테이블을 사용하는 노드의 인터넷 프로토콜 드라이버를 통해 발생합니다.

Kubernetes 호스트 내 통신(2/2)



| 수강생용 노트

참고: Amazon EKS를 사용하는 경우 veth의 이름은 이전 예의 ‘veth0a’ 및 ‘veth1a’와 같은 형식이 아니라 ‘eni’로 시작하고 그 뒤에 일련의 문자 및 숫자(16진수)가 오는 형식(예: eni52df579fccd 또는 eniad2e7f63cc2)을 사용합니다.

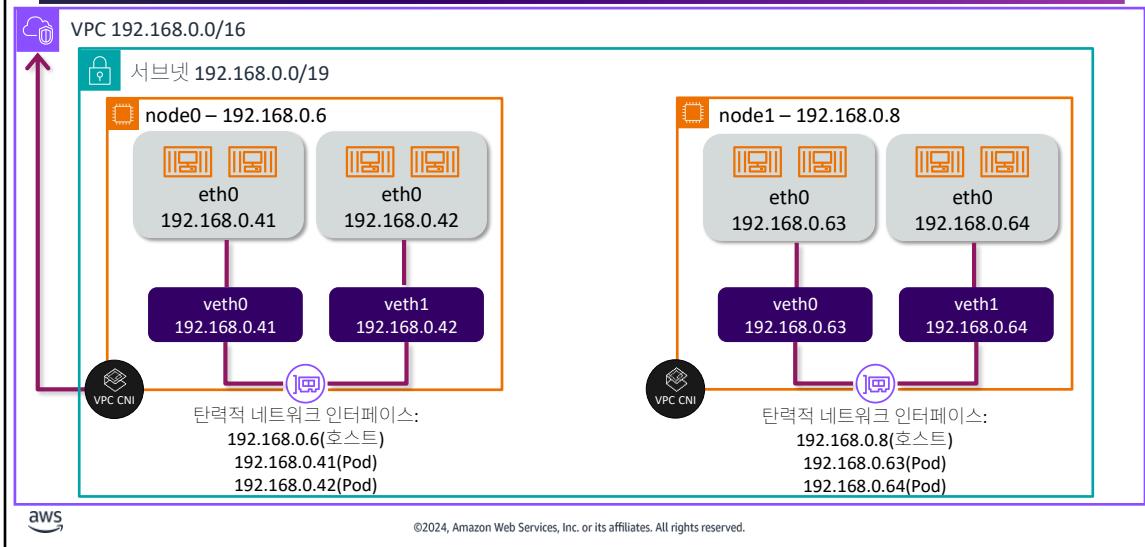
Amazon EKS는 Kubernetes용 Amazon VPC Container Network Interface(CNI) 플러그인을 통해 Amazon VPC 네트워킹을 Kubernetes에 통합합니다. 이 CNI를 사용하면 Kubernetes Pod는 Amazon VPC 네트워크에서처럼 해당 Pod 내에서 동일한 IP 주소를 가질 수 있습니다.

이 CNI 플러그인은 GitHub의 amazon-vpc-cni-k8s 리포지토리 (<https://github.com/aws/amazon-vpc-cni-k8s>)에서 유지 관리되는 오픈 소스 프로젝트입니다. Amazon VPC CNI는 호스트 인스턴스에 여러 네트워크 인터페이스를 프로비저닝하는 Amazon EC2 기능을 활용합니다. 각 인터페이스는 여러 보조 IP 주소를 사용하여 Amazon VPC 풀에서 할당된 여러 IP 주소를 가져옵니다. 이러한 주소는 노드에 할당된 기본 IP 주소와 동일한 VPC 서브넷에 있습니다 (<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html> 및 <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/MultipleIP.html>). 그런 다음 Amazon VPC CNI 에이전트는 이러한 IP 주소를 호스트의 Pod에 배포하고 네트워크 인터페이스를 Pod에 생성된 veth 포트에 연결합니다. 나머지는 Linux 커널에서 관리합니다. 따라서 모든 Pod에는 Amazon VPC에서 할당된 라우팅 가능한 실제 IP 주소가 있으며 다른 Pod, 노드 또는 AWS 서비스와 통신할 수 있습니다.

호스트에서 **CNI**는 기본 라우팅 테이블과 네트워크 인터페이스 라우팅 테이블을 모두 수정합니다. 기본 라우팅 테이블은 트래픽을 **Pod**로 라우팅하는 데 사용됩니다. 각 네트워크 인터페이스에는 발신 **Pod** 트래픽을 라우팅하는 데 사용되는 자체 라우팅 테이블이 있습니다. 각 **Pod**에는 네트워크 인터페이스의 보조 IP 주소가 하나씩 할당됩니다. 예를 들어 **Pod**가 인스턴스 외부로 패킷을 보내는 경우 한 인스턴스에 여러 네트워크 인터페이스가 존재할 수 있으므로 패킷이 올바른 네트워크 인터페이스를 통과하는지 확인해야 합니다. 또한 비대칭 라우팅을 방지하려면 패킷이 올바른 네트워크 인터페이스로 돌아가는지도 확인해야 합니다.

Amazon VPC CNI는 `aws-node`라는 DaemonSet(`kube-system` 네임스페이스에 있음)로 실행됩니다. 따라서 Amazon VPC CNI 애이전트가 클러스터의 각 노드에서 **Pod**로 실행됩니다.

Amazon EKS 호스트 간 통신



| 수강생용 노트

다른 노드의 Pod는 어떻게 통신하는지? 노드 간 통신을 단순화하기 위해 Amazon EKS는 기본적으로 Amazon VPC CNI를 사용합니다. 따라서 모든 Pod에는 Amazon VPC에서 할당된 라우팅 가능한 실제 IP 주소가 있으며 다른 Pod, 노드 또는 AWS 서비스와 통신할 수 있습니다. 기본적으로 모든 Pod는 Amazon EKS 클러스터의 다른 모든 Pod와 통신할 수 있습니다.

서브넷, 라우팅 테이블, 보안 그룹, 네트워크 액세스 제어 목록, VPC 엔드포인트, VPC 피어링, 게이트웨이를 포함하여 Amazon VPC 아키텍처는 Amazon EKS 클러스터에 있는 Pod와의 노드 간 통신에 영향을 미칠 수 있습니다. 여기에는 다른 Amazon EKS 클러스터, Amazon ECS, Amazon EC2 등이 포함될 수 있습니다. 클러스터 설계는 VPC의 서브넷에서 모든 노드 및 Pod에 충분한 양의 IP 주소를 확보해야 합니다. Pod 및 노드가 종료되고 생성되는 업그레이드 또는 기타 운영 이벤트 중에 동시 Pod 및 노드의 전체 인벤토리에 대한 적절한 중복이 VPC 아키텍처에 설계되도록 주의를 기울여야 합니다. 'EKS 모범 사례 가이드'의 'VPC 및 서브넷 고려 사항' (<https://aws.github.io/aws-eks-best-practices/networking/subnets/>)을 참조하십시오.

더 자세한 내용은 다음을 참조하십시오.

- Amazon VPC CNI 오픈 소스 리포지토리: <https://github.com/aws/amazon-vpc-cni-k8s>
- 노드당 ENI 수 및 ENI당 IP 주소 수에 대한 인스턴스 유형별 제한: https://github.com/aws/amazon-vpc-cni-k8s/blob/master/pkg/awsutils/vpc_ip_resource_limit.go
- EC2 인스턴스 유형을 기반으로 노드당 최대 정상 Pod 계산(kube-proxy 및 aws-node 네트워크 에이전트 daemonset Pod 제외) - <https://github.com/aws/amazon-vpc-cni-k8s/blob/master/misc/eni-max-pods.txt>

‘eni-max-pods.txt’ 참조에는 다음 계산이 포함되어 있습니다.

“매핑은 다음 공식을 사용하여 AWS EC2 API에서 계산됩니다.

* 각 ENI의 첫 번째 IP는 Pod에 사용되지 않습니다.

* 호스트 네트워킹(AWS CNI 및 kube-proxy)을 사용하는 Pod의 경우 +2”

노드당 최대 Pod 수 = “ENI 수 * (ENI당 IPv4 수 - 1) + 2”

워크로드의 IP 공간을 관리해야 합니다. AWS는 AWS Organization 외부 네트워크의 IP 주소를 관리하기 위해 Amazon VPC IP Address Manager(IPAM)를 제공합니다. 자세한 내용은 <https://docs.aws.amazon.com/vpc/latest/ipam/enable-integ-ipam-outside-org.html>을 참조하십시오.



Running Containers on Amazon EKS

Pod 수준 보안 개선



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

Pod 네트워크 통신 보안



Amazon VPC
Container Networking
Interface(CNI)
플러그인



- 네트워크 세분화 및 테넌트 격리 구현
- 네트워크 인바운드 및 아웃바운드 규칙 생성
- Pod 또는 Kubernetes 네임스페이스에 네트워크 정책 할당
- 제로 트러스트 네트워크 보안 모델 채택

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

| 수강생용 노트

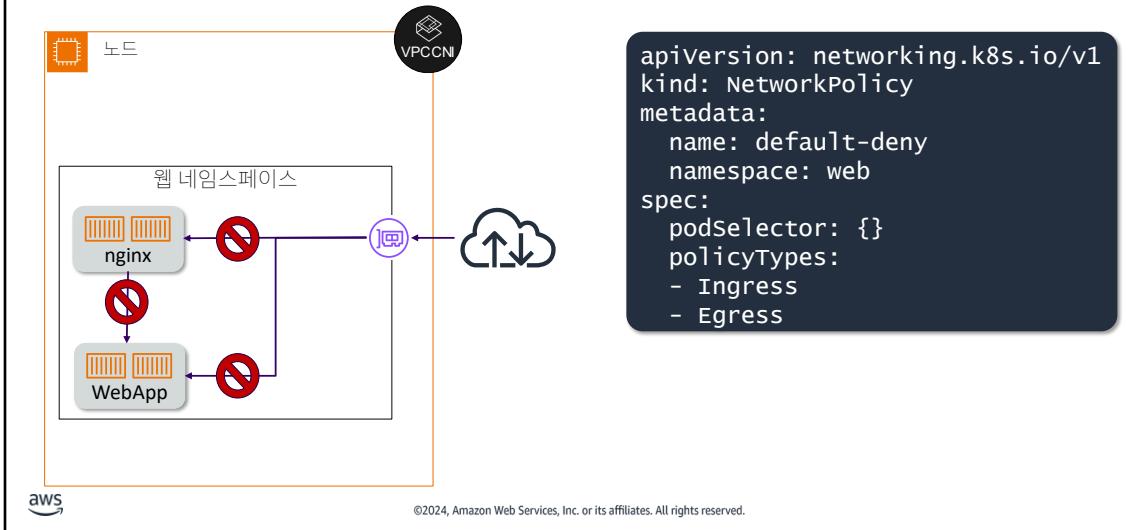
기본적으로 **Kubernetes**는 모든 **Pod**가 제한 없이 서로 통신할 수 있도록 허용합니다. **Kubernetes** 네트워크 정책을 사용하면 **Pod** 간 트래픽 흐름에 대한 규칙을 정의 및 적용할 수 있습니다. 이는 **Pod** 레이블, 네임스페이스, IP 주소, IP 블록(CIDR 범위), 포트와 같은 다양한 기준에 따라 수신 및 송신 네트워크 트래픽 규칙을 지정하여 클러스터를 세분화하고 보호할 수 있는 가상 방화벽 역할을 합니다.. 오늘날까지 고객은 서드 파티 네트워크 정책 플러그인을 사용하여 네트워크 정책을 구현했으며 이로 인해 운영 및 관리 오버헤드가 종종 발생했습니다. **Amazon VPC CNI**의 통합 기능은 클러스터 구성 및 배포를 단순화합니다. 트래픽 흐름을 제한함으로써 크기 조정 문제에 대해 걱정할 필요 없이 더욱 강력한 보안 태세를 달성할 수 있습니다.

Amazon VPC CNI에서 네트워크 정책을 기본 지원하므로 **AWS**에서 **Kubernetes**를 실행할 때 이제 민감한 워크로드를 격리하고 무단 액세스로부터 보호하는 정책을 생성할 수 있습니다. 이러한 세분화된 제어 수준을 통해 승인된 **Pod**만 서로 통신할 수 있도록 보장하는 최소 권한 원칙을 구현할 수 있습니다. 네트워크 정책은 **Amazon Elastic Compute Cloud(Amazon EC2)** 보안 그룹 및 네트워크 액세스 제어 목록과 같이 **Amazon VPC**에서 제공하는 보안 기능을 확장하는 심층 방어 메커니즘을 제공합니다.

Amazon EKS는 VPC CNI 외에도 **Calico** 및 **Cilium**과 같은 서드 파티 솔루션도 지원합니다.

자세한 내용은 <https://aws.amazon.com/blogs/containers/amazon-vpc-cni-now-supports-kubernetes-network-policies/>를 참조하십시오.

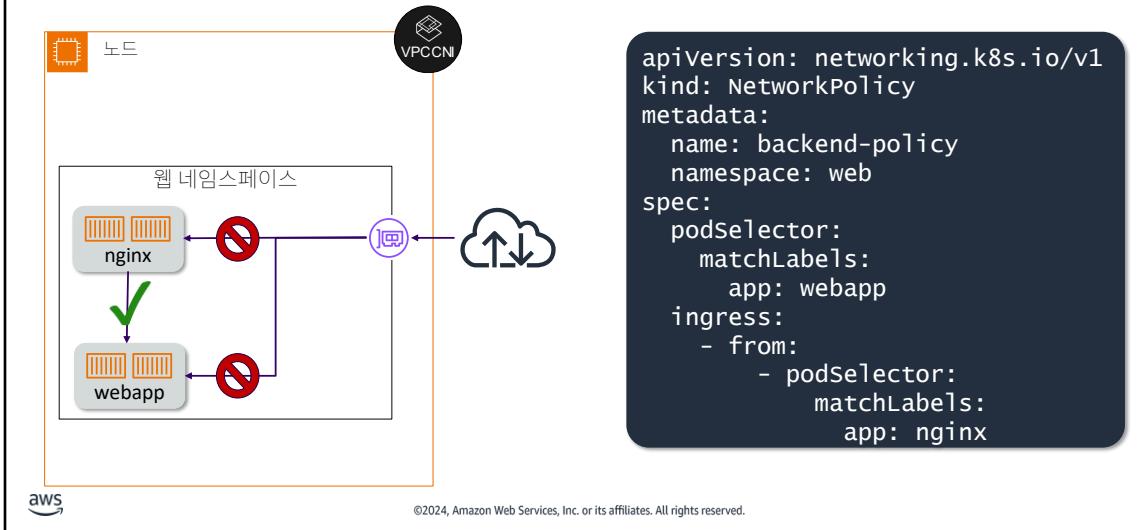
네트워크 보안 정책 적용(1/3)



| 수강생용 노트

VPC CNI 플러그인은 **kube-system** 네임스페이스에서 **DaemonSet**로 실행됩니다.
제한적인 네트워크 정책이 웹 공간에 적용되어 소스에서 **Web** 네임스페이스의 **Pod**에
대한 송수신이 허용되지 않습니다. 안전하지만 서비스 작동이 허용되지 않습니다.

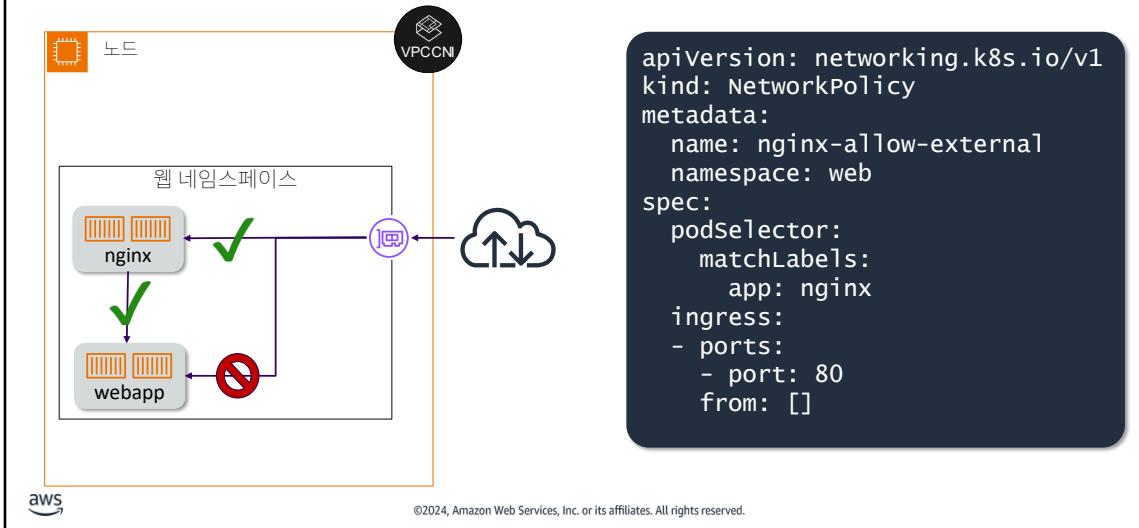
네트워크 보안 정책 적용(2/3)



| 수강생용 노트

이 예에서는 프런트엔드 **nginx Pod**에서 트래픽을 수신하기 위해 **webapp** 백엔드 **Pod**를 여는 네트워크 정책을 보여 줍니다. 이제 워크로드를 실행하는 **Pod**가 한 개 이상의 방향으로 통신할 수 있지만 외부에서의 연결은 여전히 거부됩니다.

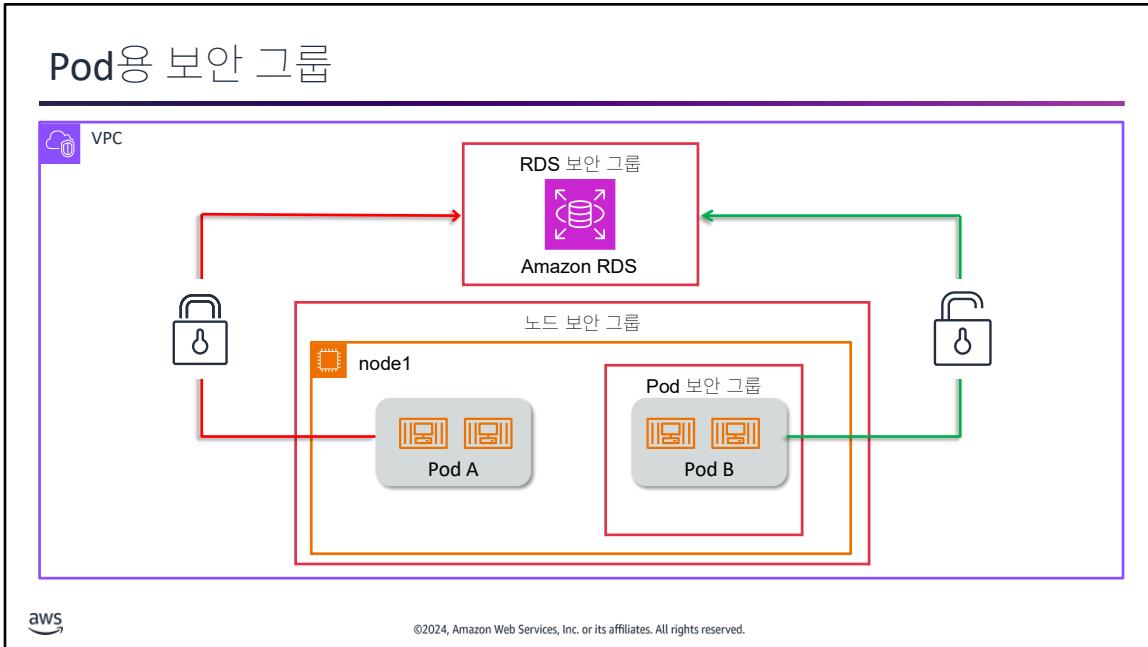
네트워크 보안 정책 적용(3/3)



| 수강생용 노트

이 예에서는 **port 80**의 **nginx Pod**에 대한 연결을 허용하는 세 번째 네트워크 정책을 추가합니다. 이제 외부 트래픽은 프런트엔드 서비스에 도달할 수 있지만 백엔드 Pod에는 도달할 수 없습니다. 이는 일반적으로 바람직한 구성입니다. 이러한 네트워크 정책을 AWS 및 Kubernetes가 제공하는 다른 보안 기능과 결합하면 잠재적인 악용을 차단하는 ‘심층 방어’를 제공할 수 있습니다.

Pod용 보안 그룹



| 수강생용 노트

컨테이너식 애플리케이션은 클러스터 내에서 실행되는 다른 서비스뿐만 아니라 **외부 AWS** 서비스에 액세스해야 하는 경우가 자주 있습니다. **AWS**에서는 서비스 간 네트워크 수준 액세스 제어가 흔히 **EC2** 보안 그룹을 통해 수행됩니다. **VPC CNI** 플러그인(이 모듈의 뒷부분에서 설명)을 사용하면 **Pod**용 보안 그룹을 통해 이를 단순화할 수 있습니다. **Pod**용 보안 그룹은 공유 컴퓨팅 리소스에서 다양한 네트워크 보안 요구 사항이 있는 애플리케이션을 실행하여 네트워크 보안 규정 준수를 달성합니다. **Pod** 간 트래픽 및 **Pod**에서 외부 **AWS** 서비스로의 트래픽까지 포괄하는 네트워크 보안 규칙은 **EC2** 보안 그룹을 사용하여 단일 위치에서 정의하고 **Kubernetes** 네이티브 API를 사용하여 애플리케이션에 적용할 수 있습니다. **Pod** 수준에서 보안 그룹을 적용한 후 애플리케이션 및 노드 그룹 아키텍처를 단순화할 수 있습니다.

Pod용 보안 그룹을 사용하면 공유 컴퓨팅 리소스에서 다양한 네트워크 보안 요구 사항이 있는 애플리케이션을 실행하여 컴퓨팅 효율성을 향상시킬 수 있습니다. **Pod** 간 및 **Pod**에서 외부 **AWS** 서비스와 같은 여러 유형의 보안 규칙을 **EC2** 보안 그룹을 사용하여 단일 위치에서 정의하고 **Kubernetes** 네이티브 API를 사용하여 워크로드에 적용할 수 있습니다. 아래 이미지는 **Pod** 수준에서 적용되는 보안 그룹과 이를 통해 애플리케이션 배포 및 노드 아키텍처를 단순화하는 방법을 보여줍니다. 이제 **Pod**가 **Amazon RDS** 데이터베이스에 액세스할 수 있습니다.

VPC CNI에서 `ENABLE POD ENI=true`를 설정하여 Pod용 보안 그룹을 활성화할 수 있습니다. 보안 그룹이 활성화되면 EKS에서 관리하는 제어 영역에서 실행되는 VPC 리소스 컨트롤러가 ‘aws-k8s-trunk-eni’라는 트렁크 인터페이스를 생성하고 노드에 연결합니다. 트렁크 인터페이스는 인스턴스에 연결된 표준 네트워크 인터페이스 역할을 합니다. 트렁크 인터페이스를 관리하려면 Amazon EKS 클러스터에 제공되는 클러스터 역할에 `AmazonEKSVPCResourceController` 관리형 정책을 추가해야 합니다.

Pod용 보안 그룹에 대한 자세한 내용은 <https://docs.aws.amazon.com/eks/latest/userguide/security-groups-for-pods.html>을 참조하십시오.

VPC 리소스 컨트롤러에 대한 자세한 내용은 <https://github.com/aws/amazon-vpc-resource-controller-k8s>를 참조하십시오.



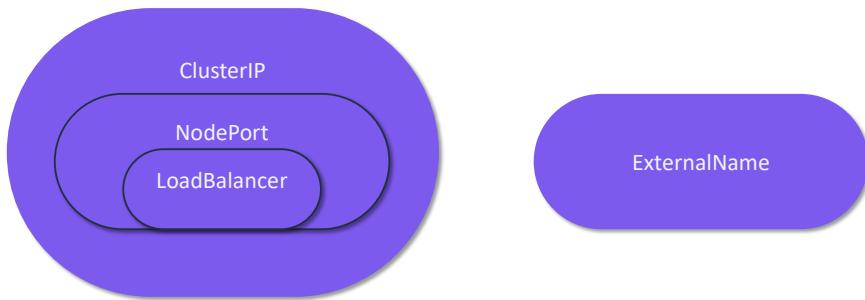
Running Containers on Amazon EKS

서비스를 사용한
로드 밸런싱



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

Kubernetes 서비스 유형



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

| 수강생용 노트

Pod와 컨테이너가 통신하는 방식을 이해했으므로 통신이 서비스에서 어떻게 작동하는지 알아보겠습니다.

서비스는 **Pod**가 사라지고 다른 **IP** 주소로 새 **Pod**가 생성되는 문제를 해결합니다. 임시 **Pod**의 **IP** 주소와 통신을 시도하는 대신 서비스의 **IP** 주소와 통신하십시오. 서비스는 **Pod** 상태로 계속 업데이트되어 정상 **Pod**로 전송됩니다.

서비스는 일관된 **IP** 주소 및 포트를 **Pod** 그룹의 진입점으로 제공합니다. **ExternalName**을 제외하고, 각 서비스 유형에는 서비스가 존재하는 동안 변경되지 않는 **IP** 주소, 정규화된 도메인 이름 및 포트가 있습니다. 내부 또는 외부 클라이언트는 서비스 **IP** 및 포트에 연결하는 방법으로 **Pod** 그룹에서 실행되는 애플리케이션에 도달할 수 있습니다. 그 후 이 연결은 해당 서비스를 지원하는 **Pod** 중 하나로 라우팅됩니다.

Pod 트래픽을 서비스의 **DNS** 도메인 이름을 통해 라우팅하면 애플리케이션 구성이 단순화됩니다. 기본 **DNS** 이름 지정 체계는 `<service name>.<namespace>.svc.cluster.local`입니다. 서비스 객체와 동일한 네임스페이스에 있는 **Pod**는 해당 <서비스 이름>만 참조할 수 있습니다. 서비스 객체가 다른 네임스페이스에 있는 경우 <서비스 이름>.<네임스페이스>를 사용할 수 있습니다.

Kubernetes는 Pod 상태를 추적하고 적절한 Pod로 트래픽을 전송하기 위해 다음과 같은 4가지 유형의 서비스를 사용합니다.

- **ClusterIP** - 클러스터 내부에서만 사용됩니다.
- **NodePort** - 정적 포트(NodePort)에 있는 각 노드의 IP에 노출되며 <NodeIP>:<NodePort>를 요청하여 클러스터 외부에서 액세스할 수 있습니다. NodePort 서비스는 NodePort를 사용하여 자동으로 생성되는 ClusterIP 서비스에 내부적으로 연결됩니다.
- **LoadBalancer** - 클라우드 제공업체의 로드 밸런서를 이용해 외부적으로 노출됩니다. LoadBalancer 서비스는 자동으로 생성되는 NodePort 및 ClusterIP에 모두 연결됩니다.
- **ExternalName** - CNAME 레코드를 해당 외부 DNS 이름 값과 함께 반환하여 내부 클러스터 DNS 이름에 맵핑됩니다.

예: ClusterIP 서비스 정의

```
apiVersion: v1
kind: Service
metadata:
  name: backend
  namespace: prod
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
```



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

```
- name: https
  protocol: TCP
  port: 443
  targetPort: 9377
```

서비스는 모든 수신 포트를 대상 포트에 매핑할 수 있습니다.

| 수강생용 노트

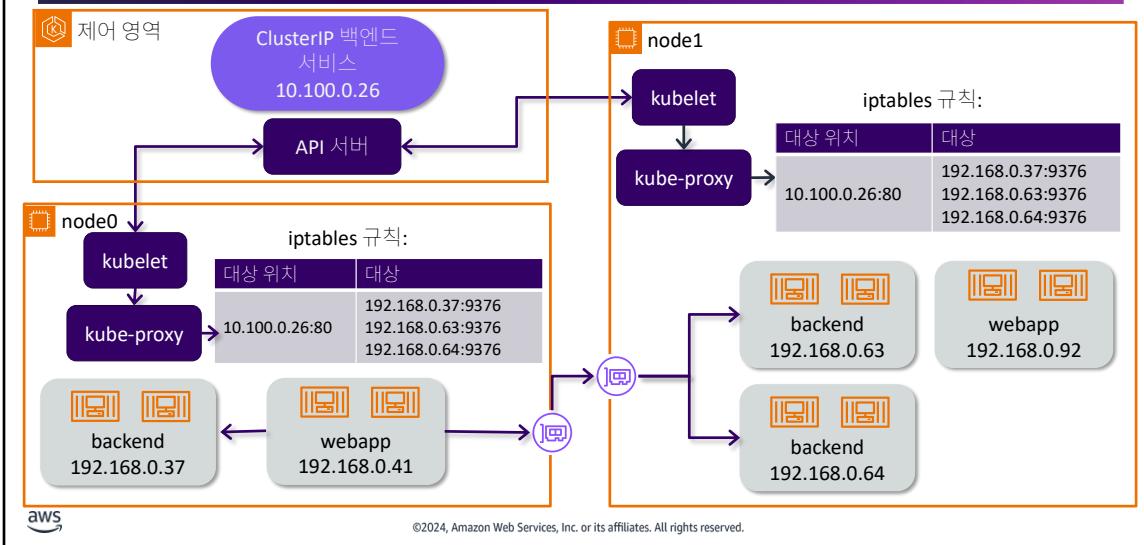
이 예에서는 ClusterIP 서비스의 YAML 매니페스트를 보여줍니다.

사양에서 가장 중요한 필드는 **selector**로, 이 서비스의 엔드포인트 역할을 하는 Pod를 결정합니다. 이 경우 레이블이 **app=backend**인 모든 Pod가 이 서비스의 일부가 됩니다. 이 사양 예에는 포트 2개가 표시됩니다. 포트 80은 컨테이너 포트 9376에 매핑되고 포트 443은 컨테이너 포트 9377에 매핑됩니다. **selector** 레이블과 일치하는 모든 Pod는 서비스의 일부가 됩니다. 서비스의 일부인 Pod의 레이블을 ‘**backend**’에서 다른 값으로 변경하면 서비스는 요청을 해당 Pod로 라우팅하지 않습니다.

Pod는 DNS 도메인 이름인 **backend.prod.svc.cluster.local**을 사용하여 네트워크 트래픽을 이 서비스로 보낼 수 있습니다. Kubernetes는 이 이름을 자동으로 등록하여 서비스의 ClusterIP 주소를 확인합니다. 예를 들어 **backend.prod.svc.cluster.local**은 **10.100.0.26**으로 확인됩니다. 그런 다음 Pod의 컨테이너가 해당 주소로 메시지 또는 연결 요청을 보내면 통신은 **10.100.0.26** 포트 80에서 포트 9376의 Pod 중 하나로 NAT됩니다. 또는, 요청이 **10.100.0.26** 포트 443으로 전달되는 경우 이는 Pod 중 하나의 IP 주소에 있는 포트 9377로 NAT되어 효과적으로 로드 밸런싱됩니다.

서비스는 모든 수신 포트를 **targetPort**에 매핑할 수 있습니다. 편의상 기본적으로 **targetPort**는 **port** 필드와 동일한 값으로 설정됩니다.

ClusterIP 서비스



| 수강생용 노트

ClusterIP 서비스는 클러스터 내에서만 액세스할 수 있습니다. 클러스터 서비스 IP 주소 범위의 IP 주소는 ClusterIP 주소에 할당됩니다. 클러스터의 서비스 ClusterIP 범위는 VPC 및 해당 서브넷의 주소 범위와 다릅니다. 노드 및 Pod에는 VPC의 서브넷에서 할당된 IP 주소가 있는 반면, 서비스 객체에는 이 목적을 위해 명시적으로 할당된 다양한 주소 범위에서 고유한 IP 주소가 할당됩니다. 예를 들어 클러스터의 VPC CIDR 범위는 192.168.0.0/16이고 클러스터의 서브넷 중 하나는 192.168.0.0/19일 수 있습니다. 노드는 192.168.0.6 및 192.168.0.8과 같은 값을 가질 수 있으며, Pod는 동일한 서브넷의 Amazon VPC CNI에 의해 192.168.0.41, 192.168.0.63, 192.168.0.64와 같은 주소가 할당될 수 있습니다. 클러스터의 서비스 CIDR은 VPC CIDR 범위와 겹치지 않습니다. 예를 들어 Kubernetes 서비스 CIDR은 10.100.0.0/16일 수 있습니다. ‘백엔드’ 서비스의 ClusterIP 주소에는 해당 범위에서 10.100.0.26과 같은 고유한 값이 할당됩니다.

다음과 같은 AWS CLI 명령을 사용하여 클러스터의 Kubernetes 서비스 CIDR 범위를 쿼리할 수 있습니다.

```
aws eks describe-cluster --name dev-cluster --region us-west-2 --query  
"cluster.kubernetesNetworkConfig"  
결과는 다음과 같을 수 있습니다.  
{  
    "serviceIpv4Cidr": "10.100.0.0/16",  
    "ipFamily": "ipv4"  
}
```

Kubernetes에서 서비스 객체를 선언할 때 ‘selector’를 지정합니다. 이 selector는 **Kubernetes** 제어 영역에서 레이블이 해당 selector와 일치하는 Pod를 쿼리하는데 사용됩니다. 특히 서비스 컨트롤러는 서비스 객체와 동일한 이름 및 **Kubernetes** 네임스페이스를 사용하여 클러스터에 엔드포인트 객체를 생성합니다. 제어 영역의 엔드포인트 컨트롤러는 서비스의 ‘selector’와 일치하는 후보 Pod에 대한 쿼리를 수행하고 Pod IP 주소 목록을 필터링하여 준비 상태인 Pod의 하위 집합을 선택합니다. 엔드포인트 컨트롤러는 제어 영역에서 엔드포인트 객체를 업데이트합니다.

예를 들어 스케일 아웃 및 스케일 인을 통해 서비스 selector와 일치하는 Pod가 생성 및 종료되고 해당 Pod가 준비 상태가 되거나 준비 상태가 중지되면 제어 영역에서 엔드포인트 객체가 업데이트됩니다. 이러한 업데이트가 발생할 때 제어 평면은 서비스의 엔드포인트를 클러스터의 모든 노드에 분산합니다. 각 노드의 kubelet은 서비스의 현재 엔드포인트 목록을 수신하면 이러한 서비스-엔드포인트 매핑의 처리를 위임합니다. 클러스터는 kube-proxy라는 DaemonSet을 실행합니다. 그러면 클러스터의 각 노드에 kube-proxy Pod가 생성됩니다.

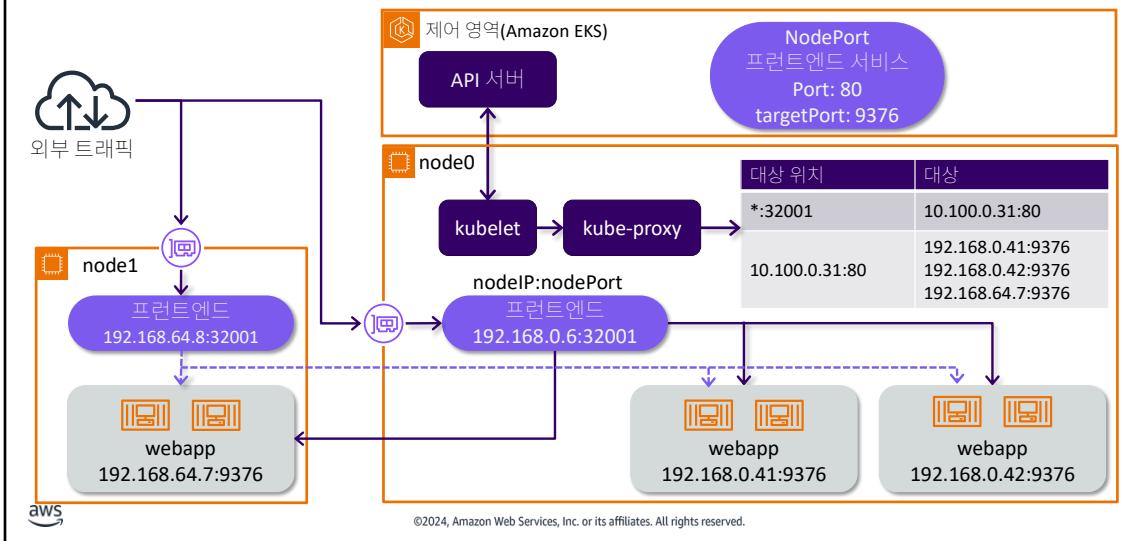
kube-proxy는 Pod 간 통신의 데이터 경로에 있지 않음에 유념해야 합니다. 또한 로드 밸런싱을 직접 구현하지도 않습니다. 대신, kube-proxy는 노드의 운영 체제 커널(예: Linux 또는 Windows)을 구성합니다. Kube-proxy는 iptables 모드 또는 IPVS 모드에서 작동하도록 구성할 수 있습니다. Amazon EKS의 기본값은 iptables 모드입니다. iptables 대신 IPVS를 사용하도록 클러스터 및 해당 노드를 구성할 수 있습니다. 이를 통해 클러스터 내 로드 밸런싱을 위해 구성할 수 있는 특정 옵션을 사용하여 Pod로 가는 트래픽의 분산을 더 강력하게 제어할 수 있습니다.

Kube-proxy는 선언적 ClusterIP 서비스 구성을 사용하고 ClusterIP 트래픽을 적절한 Pod IP로 리디렉션하는 해당 노드 iptables 규칙을 생성합니다. EC2 인스턴스는 kube-proxy를 EKS 추가 기능에 의해 제어되는 DaemonSet로 구현합니다. Fargate 노드에서는 자동으로 관리되며 노드에서 실행되는 프로세스로 숨겨집니다. Pod의 컨테이너가 서비스의 ClusterIP 주소로 통신을 전송하면 커널은 준비 상태인 엔드포인트 Pod 중 하나로 네트워크 주소 변환(NAT)을 수행합니다. 따라서 ClusterIP 서비스는 클러스터의 해당 서비스에 대해 선택된 Pod 세트 전체에서 서비스로 가는 트래픽을 로드 밸런싱합니다.

이 ClusterIP 서비스 유형은 프런트엔드 Pod가 백엔드 Pod에 도달하거나 로드 밸런서 역할을 하는 데 도움이 됩니다.

[이미지 설명: ClusterIP 서비스가 생성됩니다. API 서버는 각 kubelet에 알립니다. kubelet은 kube-proxy에 iptables 규칙을 업데이트하도록 지시합니다. webapp Pod는 ClusterIP 서비스를 사용하여 백엔드 Pod와 통신합니다. 설명 끝]

NodePort 서비스



| 수강생용 노트

NodePort 서비스는 각 노드에서 포트를 연다는 점을 제외하면 **ClusterIP** 서비스와 유사합니다. 포트를 열면 **ClusterIP**를 사용하여 클러스터 내부에서 서비스에 액세스할 수 있습니다. 로드 밸런서와 같은 외부 구성 요소는 **NodePort**의 노드에 직접 연결할 수 있습니다.

각 **NodePort** 서비스의 경우, **nodePort** 키-값 쌍을 사용하여 특정 포트를 지정하지 않은 한 노드 포트 범위(기본값: 30000~32767)에서 무작위로 선택된 포트가 해당 서비스의 각 포트 항목에 할당됩니다. 해당 포트는 각 노드에서 열립니다. **kube-proxy**는 트래픽을 노드 포트에서 서비스의 클러스터 IP로 전달한 다음 Pod로 전달하는 **iptables** 규칙을 업데이트합니다. 이는 사실상 이중 변환입니다.

외부 네트워킹 트래픽(즉, 클러스터 네트워크 외부의 트래픽)은 특정 포트의 모든 단일 EC2 인스턴스로 전송될 수 있습니다. 이 예의 프런트엔드 서비스는 포트 32001에서 수신 대기하고 클러스터 내 어디에 있든 webapp Pod 중 하나에 요청을 로드 밸런싱합니다. 이는 첫 번째 노드에서 로컬로 실행되는 webapp Pod에도 불구하고 처음에 하나의 EC2 노드에서 처리된 수신 트래픽이 다른 EC2 노드로 라우팅될 수 있다는 의미입니다. 예를 들어, 외부 트래픽이 IP 주소가 192.168.0.60이고 TCP 포트 번호가 32001인 node0으로 전달되는 경우 node0의 iptables는 먼저 nodeIP:nodePort 192.168.0.6:32001의 주소를 서비스의 clusterIP:port 10.100.0.31:80으로 변환한 다음, 준비 상태인 선택된 엔드포인트의 podIP:targetPort 중 하나로 변환합니다. 이는 node0의 192.168.0.41:9376, node0의 192.168.0.42:9376 또는 node1의 192.168.64.7:9376 중 하나일 수 있습니다. 그런 다음 iptables는 트래픽을 무작위로 선택된 대상 위치로 전달합니다. 마찬가지로, 외부 트래픽이 IP 주소가 192.168.64.8이고 TCP 포트 번호가 32001인 node1로 전달되는 경우 node1의 iptables는 먼저 nodeIP:nodePort 192.168.64.7:32001의 주소를 서비스의 clusterIP:port 10.100.0.31:80으로 변환한 다음, 준비 상태인 선택된 엔드포인트의 podIP:targetPort 중 하나로 변환합니다. 이는 node0의 192.168.0.41:9376, node0의 192.168.0.42:9376 또는 node1의 192.168.64.7:9376 중 하나일 수 있습니다.

[이미지 설명: Kubernetes API 서버는 노드의 kubelet에 NodePort 서비스 구성 정보를 알립니다. kubelet은 노드에서 iptables 규칙을 수정하도록 kube-proxy에 지시합니다. 외부 클러스터 트래픽은 노드의 열린 포트를 통해 선택된 Pod 중 하나로 라우팅됩니다. 설명 끝]

예: NodePort 서비스 정의

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: NodePort
  selector:
    app: webapp
  ports:
  - name: http
    port: 80
    targetPort: 9376
```



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

| 강사용 노트

| 멀티 테넌시 클러스터에서 충돌을 방지하기 위해 동적 포트 할당이 선택되었습니다
수강생이 "왜 **NodePort** 값이 표시되지 않습니까?"라고 질문하는 경우 멀티 테넌시
클러스터에서 충돌을 피하기 위해 동적 포트 할당이 선택되었음을 언급하십시오.
서비스 유형이 **NodePort**이고 해당 서비스의 사양에서 **nodePort** 값을 명시적으로
설정하지 않은 경우 **Kubernetes** 제어 평면은 생성 시간에 사용되지 않는 포트를
자동으로 할당합니다. 자세한 내용은

<https://kubernetes.io/blog/2023/05/11/nodeport-dynamic-and-static-allocation/> 및
<https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport>를
참조하십시오.

| 수강생용 노트

이 예에서는 **NodePort** 서비스의 YAML 파일을 보여줍니다. 이 값에 **nodePort** 값(예:
spec.ports.nodePort)을 지정할 수도 있지만, 더 나은 방법은 **Kubernetes**가 사용자 대신
노드 포트를 할당하도록 허용하여 **NodePort** 서비스 전체에서 포트 충돌을 방지하는
것입니다. 노드 포트 값을 수동으로 할당하려면 포트 충돌을 완화하기 위해 지속적인
노드 포트 관리가 필요합니다.

Helm을 사용하여 **Kubernetes** YAML을 생성하는 경우 **Helm** 차트의 **values.yaml**
파일에서 **service.type** 값을 변경할 수 있습니다. **Helm** 차트에 기본
templates/service.yaml을 사용하면 해당 값이 서비스 생성에 사용됩니다. 예를 들어,
templates/service.yaml의 다음 스니펫이 있습니다.

```
spec:  
type: {{ .Values.service.type }}  
ports:  
- port: {{ .Values.service.port }}  
  targetPort: http  
  protocol: TCP  
  name: http
```

`values.yaml` 파일이 다음을 포함하는 경우:

```
service:  
type: NodePort  
port: 8080
```

(‘helm template’, ‘helm install’ 또는 ‘helm upgrade’를 통해) 다음을 생성합니다.

```
spec:  
type: NodePort  
ports:  
- port: 8080  
  targetPort: http  
  protocol: TCP  
  name: http
```

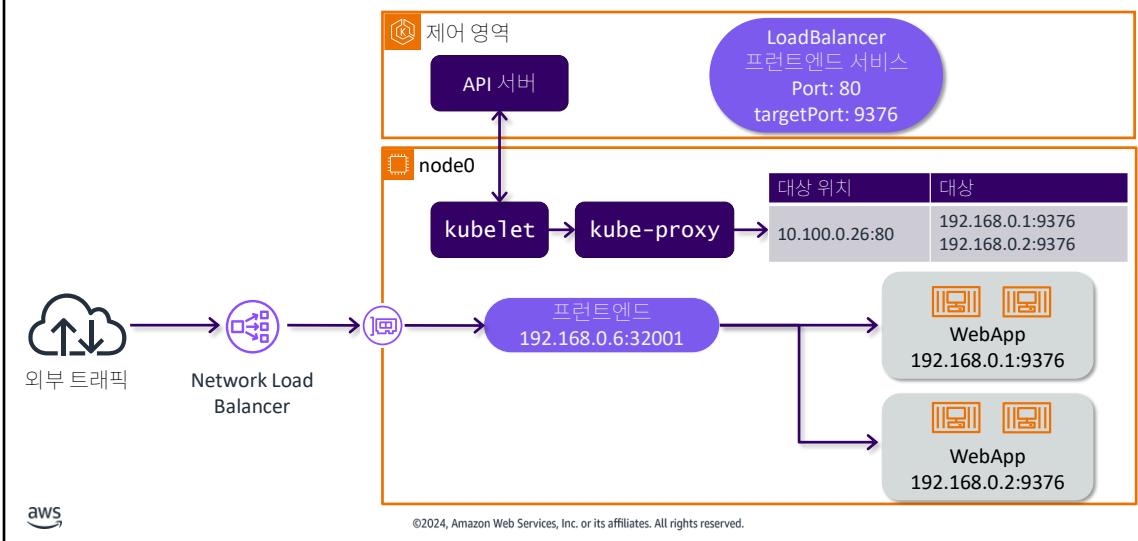
이 Helm 차트 예는 처음 **NodePort** 예와 거의 반대되는 시나리오를 나타냅니다. 이전 예는 **nodeIP:nodePort**에서 **clusterIP:80**으로 매핑된 다음 **podIP:9376** 중 하나로 매핑되었습니다. 이 Helm 차트 예는 **nodeIP:nodePort**에서 **clusterIP:8080**으로 매핑된 다음 **podIP:80** 중 하나로 매핑됩니다.

다음 시퀀스를 기억하십시오.

- **nodeIP:nodePort => clusterIP:port => podIP:targetPort**

또한 서비스에 지정된 **targetPort**는 선택된 Pod의 컨테이너 중 하나의 **ports** 섹션에 있는 **containerPort**와 일치해야 합니다. 이는 **NodePort** 서비스뿐만 아니라 **ClusterIP** 서비스에도 적용됩니다.

LoadBalancer 서비스 – 인스턴스 모드



~Alt text

~ 외부 클러스터 트래픽은 Network Load Balancer를 거쳐 NodePort 서비스로 라우팅됩니다.

~

| 수강생용 노트

LoadBalancer 서비스는 모든 노드 앞에 로드 밸런서를 추가하여 NodePort 서비스를 확장합니다. Amazon EKS에서는 Kubernetes가 로드 밸런서를 요청하고 모든 노드를 등록한다는 의미입니다. 로드 밸런서는 지정된 서비스의 Pod가 실행되고 있는 위치를 감지하지 않습니다. 따라서 모든 노드가 로드 밸런서에 백엔드 인스턴스로 추가됩니다. 기본적으로 Network Load Balancer는 LoadBalancer 유형 서비스에 사용됩니다. 즉, 요청을 수신 대기하는 모든 인스턴스 앞에 Network Load Balancer가 있다는 의미입니다. 로드 밸런서는 노출된 포트를 통해 이를 노드로 라우팅합니다.

LoadBalancer 서비스는 인스턴스 모드와 IP 모드 중 하나로 사용할 수 있습니다.

이 슬라이드에서는 인스턴스 모드에 대해 설명합니다. 단, 인스턴스 모드는 AWS Fargate와 같은 서비스와 호환되지 않는다는 점을 알고 있어야 합니다. 이 경우 IP 모드를 사용해야 합니다. IP 모드에 대해서는 이 모듈의 뒷부분에서 설명합니다.

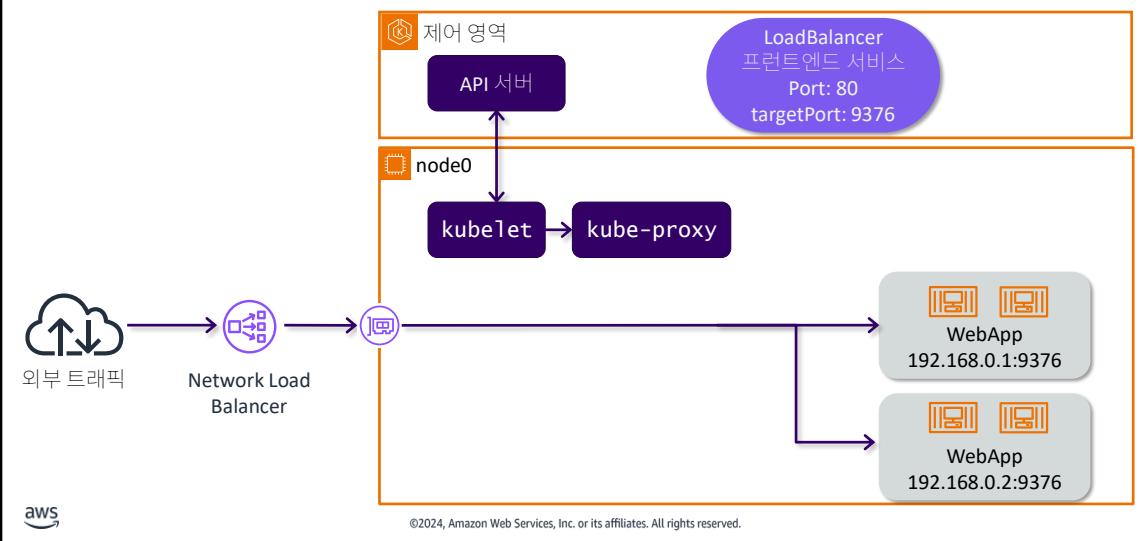
인스턴스 모드에서는 인터넷에서 전송된 요청은 먼저 **Network Load Balancer**에 도착합니다. 그러면 **Network Load Balancer**는 이 요청을 특정 포트의 **EC2** 인스턴스 중 하나로 전달합니다. 이 예의 프런트엔드 **NodePort** 서비스는 포트 **32001**에서 수신 대기하고 백엔드 **Pod** 중 하나에 요청을 전달합니다. 그러면 여기에서 요청이 처리됩니다.

일반적인 **Kubernetes** 배포에서는 내부 **Kubernetes** 트래픽에 오버레이 네트워크를 사용합니다. 이로 인해 각 **Pod**는 외부에서 연결할 수 없습니다. 이러한 유형의 네트워킹에서는 **Pod**에 직접 액세스할 수 없으므로 인스턴스 모드를 사용해야 합니다. 노드에 직접 호스팅된 **NodePort**만 외부 네트워킹을 사용하여 액세스할 수 있습니다. **EKS** 버전(**VPC CNI**)은 **VPC**와 동일한 네트워크에서 **Pod**를 호스팅하므로 **AWS**의 모든 것에 액세스할 수 있습니다. **Calico**와 같은 서드 파티 **CNI**를 사용하는 것이 이러한 상황이 발생할 수 있는 예입니다.

Kubernetes CNI 추가 기능의 목록은 **Kubernetes** 설명서의 ‘**Networking and Network Policy**’(<https://kubernetes.io/docs/concepts/cluster-administration/addons/#networking-and-network-policy>)를 참조하십시오.

Fargate를 사용하는 경우 인스턴스 모드를 사용할 수 없으며 대신 **IP** 모드를 사용해야 합니다.

LoadBalancer 서비스 – IP 모드



~Alt text

~ 외부 클러스터 트래픽은 Network Load Balancer를 거쳐 Pod로 직접 라우팅됩니다.

~

| 수강생용 노트

LoadBalancer 서비스는 IP 모드로도 설정할 수 있습니다. IP 모드를 사용하면 로드 밸런서가 NodePort 서비스 없이도 IP 대상이 있는 Pod에 직접 연결할 수 있습니다. IP 모드는 NodePort 매핑으로 인해 발생할 수 있는 지터를 줄일 수 있으므로 유용합니다.

Network Load Balancer 대신, 수신 등을 위해 Application load balancer를 사용할 수도 있습니다. 이에 대해서는 이후의 슬라이드에서 설명합니다.

보안 그룹이 업데이트되고 애플리케이션에 사용되는 모든 로드 밸런서와 연결되어 있는지 확인하는 것이 중요합니다. 이 작업은 AWS CLI 또는 콘솔을 사용하여 수행할 수 있습니다.

자세한 내용은

<https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-update-security-groups.html>을 참조하십시오.

콘솔을 사용하여 언제든지 로드 밸런서와 연결된 보안 그룹을 업데이트할 수 있습니다.

Amazon EC2 콘솔을 엽니다.

Load Balancers를 선택합니다.

로드 밸런서를 선택합니다.

Security 탭에서 **Edit**를 선택합니다.

보안 그룹을 로드 밸런서와 연결하려면 보안 그룹을 선택합니다. 보안 그룹 연결을 제거하려면 보안 그룹에 대한 **X** 아이콘을 선택합니다.

Save changes를 선택합니다.

또한 언제라도 AWS CLI **set-security-groups** 명령을 사용할 수 있습니다.

Amazon EKS 보안 그룹 요구 사항 및 고려 사항은

<https://docs.aws.amazon.com/eks/latest/userguide/sec-group-reqs.html>을 참조하십시오.

예: LoadBalancer 서비스 정의

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
spec:
  selector:
    app: webapp
  ports:
  - port: 80
    targetPort: 9376
    protocol: TCP
  type: LoadBalancer
  loadBalancerClass: service.k8s.aws/nlb
```



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

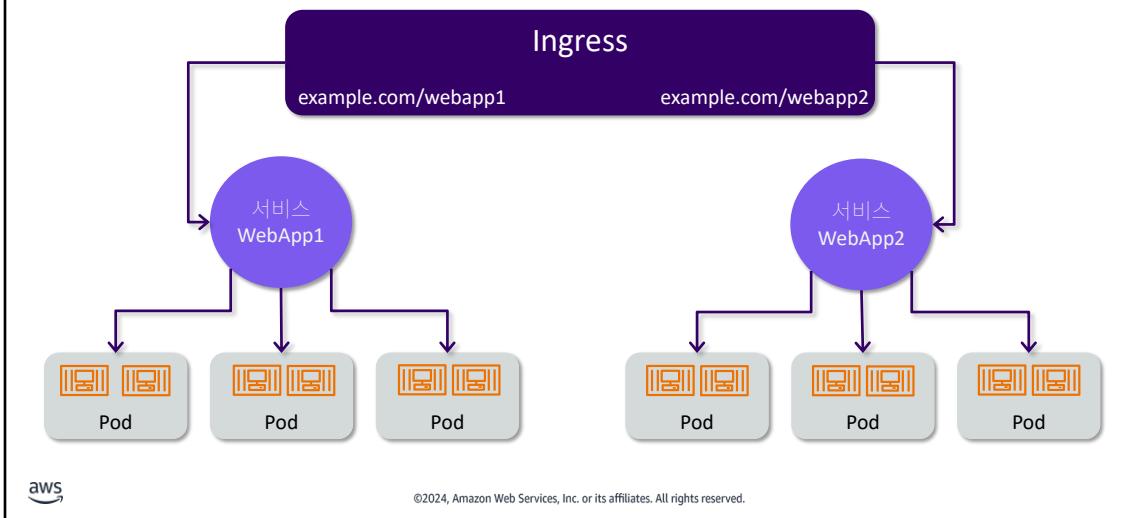
| 수강생용 노트

이 예에서는 인스턴스 모드를 사용하여 **LoadBalancer** 서비스를 정의하는 **YAML** 매니페스트를 보여줍니다. 내부적으로 트래픽은 대상 포트 **9376**에서 앱 레이블이 **webapp**과 동일한 **Pod**로 라우팅됩니다.

v2.2.0 릴리스부터 인터넷 연결 Network Load Balancer를 생성하려면 서비스에서 **service.beta.Kubernetes.io/aws-load-balancer-scheme: INTERNET** 주석을 사용해야 합니다.

NodePort 유형 서비스 예와 마찬가지로 **Helm**을 사용하여 **Kubernetes YAML**을 생성하는 경우 **Helm** 차트의 **values.yaml** 파일에서 **NodePort**(또는 **ClusterIP**) 대신 '**LoadBalancer**' 값을 갖도록 **service.type** 값을 변경할 수 있습니다.

Ingress



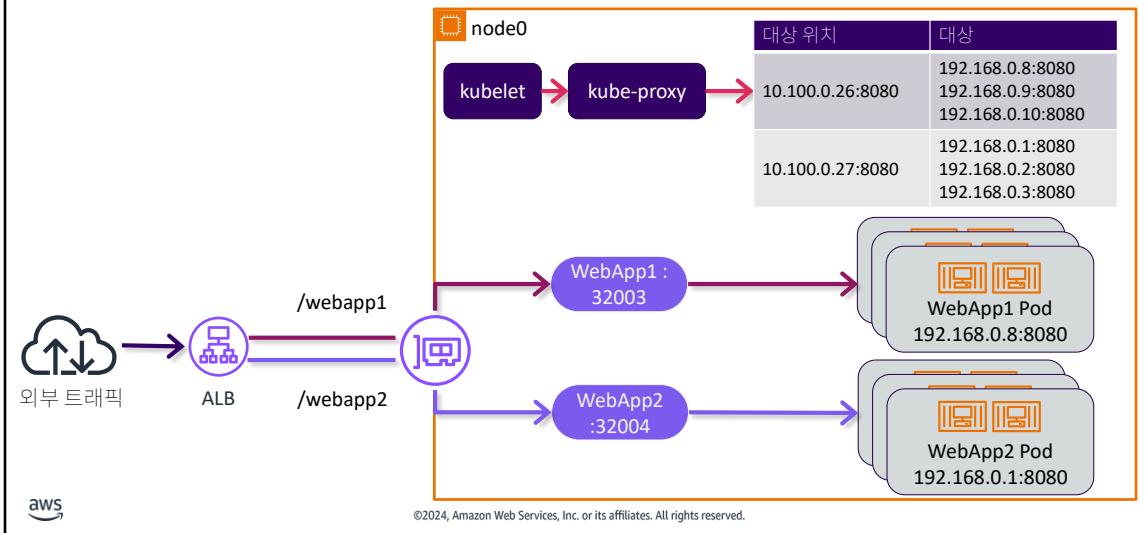
| 수강생용 노트

Kubernetes Ingress 객체를 사용하면 사용하는 로드 밸런서의 수를 줄일 수 있습니다. **Ingress** 객체는 클러스터 외부에 있는 **HTTP** 및 **HTTPS** 경로를 서비스에 노출하고 트래픽 규칙을 정의합니다. **Ingress** 객체는 일반적으로 로드 밸런서를 통해 수식 규칙 및 요청을 이행하는 **Ingress 컨트롤러**를 사용합니다.

Ingress 객체 및 컨트롤러를 사용하면 서비스당 로드 밸런서 하나에서 **Ingress**당 로드 밸런서 하나로 전환하고 여러 서비스로 라우팅할 수 있습니다. 트래픽은 경로 기반 라우팅을 사용하여 적절한 서비스로 라우팅할 수 있습니다.

Kubernetes를 사용하면 **AWS Load Balancer Controller**, **Envoy** 컨트롤러, **NGINX** 컨트롤러 등 여러 가지 **Ingress** 컨트롤러 옵션을 사용할 수 있습니다.

AWS Load Balancer Controller



| 슬라이드 노트

AWS Load Balancer Controller는 Kubernetes 클러스터용 AWS Elastic Load Balancer를 관리하는 컨트롤러입니다. 이러한 로드 밸런서는 Kubernetes 수신 객체를 생성하는 경우 AWS Application Load Balancers(ALB) 또는 Kubernetes 서비스 유형 LoadBalancer를 생성하는 경우 AWS NLB가 될 수 있습니다.

컨트롤러를 사용하면 하나의 ALB를 공유하는 여러 Kubernetes 수신 규칙을 가질 수 있습니다.

이 예에서는 AWS Load Balancer Controller를 사용하는 Kubernetes 클러스터를 보여줍니다. 요청이 이루어진 경우 Application Load Balancer로 라우팅됩니다. Application Load Balancer는 경로에 따라 적절한 대상 그룹을 선택합니다. 그런 다음 로드 밸런서는 이 요청을 포트 32003 또는 32004의 EC2 인스턴스 중 하나로 전달합니다. kube-proxy 명령은 트래픽을 포트에서 서비스의 클러스터 IP로 전달한 다음 Pod로 전달하는 iptables 규칙을 업데이트합니다. WebApp1 서비스는 포트 32003에서 수신 대기하고 요청을 WebApp1 Pod 중 하나로 로드 밸런싱합니다. WebApp2 서비스는 포트 32004에서 수신 대기하고 요청을 WebApp2 Pod 중 하나로 로드 밸런싱합니다. 그러면 Pod가 요청을 처리합니다.

자세한 내용은 **Amazon EKS** 사용 설명서의 ‘AWS Load Balancer Controller’ (<https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html>)를 참조하십시오.

예: Ingress 객체 정의

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp
  namespace: prod
  annotations:
    alb.ingress.kubernetes.io/subnet
    s: subnet-0061ab916d8e0f34f
spec:
  ingressClassName: alb
  rules:
  - http:
    paths:
```

```
- path: /webapp1
  backend:
    service:
      name: webapp1
      port:
        number: 32003
- path: /webapp2
  backend:
    service:
      name: webapp2
      port:
        number: 32004
```



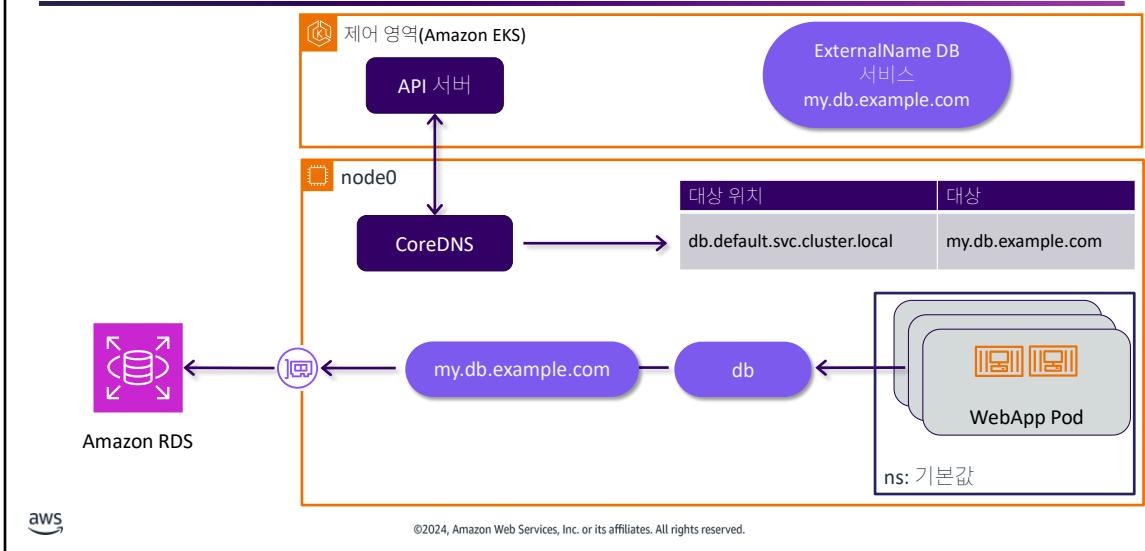
©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트

이 예에서는 Ingress 객체에 대한 YAML 파일을 보여줍니다.

Ingress 객체에는 어떤 수신 경로가 어떤 기존 NodePort 서비스로 이동하는지를 지정하는 **path** 필드가 있습니다. 이 경우 **/webapp1** 경로는 포트 **32003**의 **Webapp1** 서비스로 이동합니다. **/webapp2** 경로는 포트 **32004**의 **Webapp2** 서비스로 이동합니다.

ExternalName 서비스



| 수강생용 노트

마지막 Kubernetes Service 유형은 ExternalName 서비스입니다.

ExternalName 서비스 유형은 클러스터 외부의 리소스에 연결하는 데 사용합니다. 예를 들어 **db**라는 서비스를 생성하고 이를 **Amazon Relational Database Service(Amazon RDS)** 엔드포인트에 연결할 수 있습니다. 데이터베이스에 액세스해야 하는 Pod가 **db** 서비스에 연결되며, **Amazon RDS** 엔드포인트를 반환합니다. ExternalName 서비스는 CNAME DNS 레코드와 유사하게 작동합니다. 따라서 이 서비스 유형은 앞에서 다른 서비스 객체처럼 어떤 종류의 로드 밸런싱 기능도 제공하지 않습니다.

이 유형의 서비스는 다른 환경(예: Dev, QA, Prod 환경의 데이터베이스)에서 외부 서비스를 가리킬 수 있는 동일한 내부 엔드포인트를 사용하는 데 유용합니다. 또한 ExternalName 서비스는 외부 리소스(예: 리팩터링된 모놀리스 애플리케이션)를 Kubernetes 클러스터로 마이그레이션하려는 경우에 유용합니다. 외부 리소스를 클러스터로 마이그레이션한 후 서비스 유형을 ClusterIP로 업데이트합니다. 애플리케이션은 동일한 엔드포인트를 계속 사용할 수 있습니다.

예: ExternalName 서비스 정의

```
apiVersion: v1
kind: Service
metadata:
  name: db
  namespace: prod
spec:
  type: ExternalName
  externalName: my.db.example.com
```



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

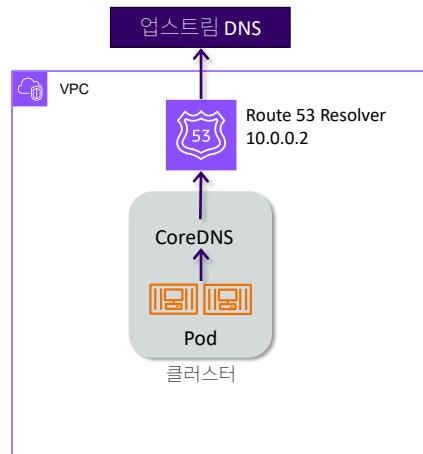
| 수강생용 노트

ExternalName 서비스에 대한 YAML 파일의 예입니다.

ExternalName 서비스에는 서비스가 매핑되는 위치를 지정하는 **externalName** 필드가 있습니다. 이 예에서 `db.prod.svc.cluster.local`의 ExternalName DNS 항목은 `my.db.example.com`으로 확인됩니다.

Kubernetes에서 DNS 확인

- **CoreDNS 추가 기능:** EKS 클러스터 내부 리소스의 DNS 이름을 확인
- **VPC용 기본 Route 53 Resolver:** VPC 내부에 있지만 클러스터 외부에 있는 리소스를 확인
- **업스트림 DNS 이름 서버:** VPC 외부 리소스를 확인



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

| 수강생용 노트

CoreDNS는 Kubernetes 클러스터 DNS 역할을 할 수 있는 유연하고 확장 가능한 DNS 서버입니다. Amazon EKS 클러스터를 하나 이상의 노드로 시작하면 클러스터에 배포된 노드 수에 관계없이 CoreDNS 이미지의 복제본 2개가 기본적으로 배포됩니다. CoreDNS Pod는 클러스터의 모든 Pod에 대한 이름 확인을 제공합니다.

자세한 내용은 <https://docs.aws.amazon.com/eks/latest/userguide/managing-coredns.html>을 참조하십시오.

지식 확인 1

클러스터 외부에서 액세스할 수 있는 서비스는 무엇입니까?

보기	응답
A	ClusterIP
B	NodePort
C	LoadBalancer
D	B와 C 모두



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

지식 확인 1: 정답은 D입니다.

클러스터 외부에서 액세스할 수 있는 서비스는 무엇입니까?

보기	응답
A	ClusterIP
B	NodePort
C	LoadBalancer
D	B와 C 모두

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

| 수강생용 노트

정답은 D입니다. **NodePort** 서비스는 각 노드에서 포트를 열어 클러스터 외부에서의 액세스를 허용합니다. **LoadBalancer** 서비스는 모든 노드 앞에 로드 밸런서를 추가하여 **NodePort** 서비스를 확장합니다.

지식 확인 2

VPC Container Network Interface란 무엇입니까?

보기	응답
A	가상 네트워크 카드를 나타내는 VPC 내의 논리적 네트워킹 구성 요소
B	프라이빗 서브넷의 인스턴스를 인터넷에 연결할 수 있도록 하는 방법
C	Pod 집합에서 실행 중인 애플리케이션을 노출하는 Kubernetes 객체
D	Kubernetes Pod가 VPC 네트워크에서처럼 해당 Pod 내에서 동일한 IP 주소를 갖게 하는 플러그 인

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

|수강생용 노트

지식 확인 2: 정답은 D입니다.

VPC Container Network Interface란 무엇입니까?

보기	응답
A	가상 네트워크 카드를 나타내는 VPC 내의 논리적 네트워킹 구성 요소
B	프라이빗 서브넷의 인스턴스를 인터넷에 연결할 수 있도록 하는 방법
C	Pod 집합에서 실행 중인 애플리케이션을 노출하는 Kubernetes 객체
D	Kubernetes Pod가 VPC 네트워크에서처럼 해당 Pod 내에서 동일한 IP 주소를 갖게 하는 플러그인

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

| 수강생용 노트

정답은 D입니다. Amazon EKS는 Kubernetes용 Container Network Interface(CNI) 플러그인을 사용하여 VPC 네트워킹을 Kubernetes에 통합합니다. CNI 플러그인은 Kubernetes Pod가 VPC 네트워크에서처럼 해당 Pod 내에서 동일한 IP 주소를 가질 수 있게 합니다.

활동



aws

이 활동에서는 다음 과제를 수행합니다.

- 트래픽이 라우팅되는 방법 확인

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

| 수강생용 노트

활동: 각 시나리오에 대해 트래픽이 첫 번째 위치에서 두 번째 위치로 라우팅되는 방식을 결정합니다.

노드 내 Pod 간: 경로

경로: 동일한 노드의 Pod 간

메서드:



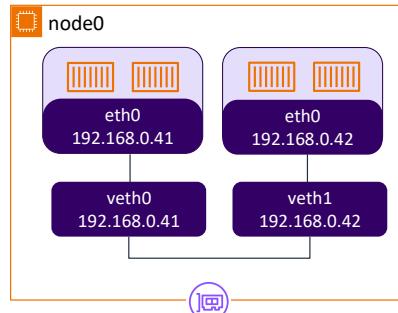
©2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

| 수강생용 노트
동일한 노드에 있는 두 Pod는 어떻게 통신합니까?

노드 내 Pod 간: 메서드

경로: 동일한 노드의 Pod 간

메서드: Linux 가상 이더넷 디바이스(veth)



aws

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

~ALT text

~node 0: veth 터널을 보여주는 아키텍처로, 노트에 자세히 설명되어 있습니다.

| 수강생용 노트

호스트의 Pod 간 통신에서는 veth 터널을 사용합니다. 각 노드는 컨테이너에 대한 네트워크 범위를 가져오고 각 Pod는 해당 범위의 IP 주소를 가져옵니다. 이제 동일한 호스트의 컨테이너가 서로 통신할 수 있습니다.

노드 간 Pod 간: 경로

경로:

다른 노드의 Pod 간

메서드:



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

| 수강생용 노트

다른 노드의 두 Pod는 어떻게 통신합니까?

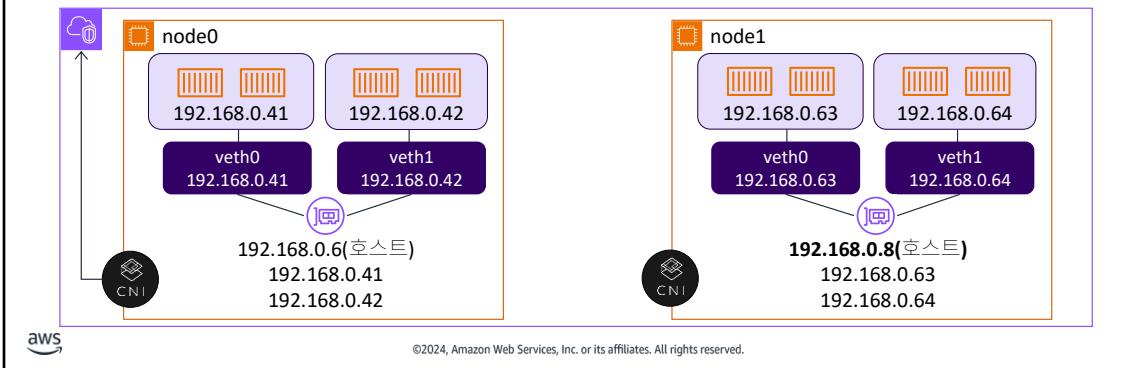
노드 간 Pod와 Pod: 메서드

경로:

다른 노드의 Pod 간

메서드:

Amazon VPC CNI 플러그 인



~ALT text

~VPC: VPC CNI 플러그 인을 보여주는 아키텍처로, 노트에 자세히 설명되어 있습니다.

| 수강생용 노트

노드 간 통신을 위해 Amazon EKS는 Kubernetes용 Amazon VPC CNI 플러그 인을 통해 Amazon VPC 네트워킹을 Kubernetes에 통합합니다. 이 CNI 플러그 인은 Kubernetes Pod가 VPC 네트워크에서와 마찬가지로 Pod 내 동일한 IP 주소를 보유할 수 있게 합니다.

서비스에 대한 외부 트래픽: 경로

경로: 특정 서비스의 Pod에 대한 외부 트래픽

메서드:



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

| 수강생용 노트

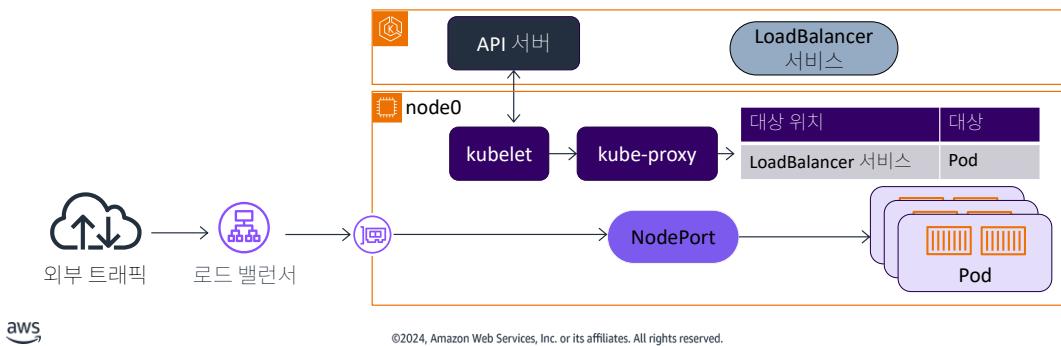
외부 트래픽은 특정 애플리케이션(특정 서비스의 Pod)과 어떻게 통신합니까?

서비스에 대한 외부 트래픽: 메서드

경로: 특정 서비스의 Pod에 대한 외부 트래픽

메서드:

LoadBalancer 서비스



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

~ALT text

~LoadBalancer 서비스를 보여주는 아키텍처로, 노트에 자세히 설명되어 있습니다.

| 수강생용 노트

인스턴스 모드에서 AWS LoadBalancer는 트래픽을 클러스터 내의 모든 EC2 인스턴스로 라우팅합니다. AWS LoadBalancer는 노드의 프라이빗 IP를 대상으로 하고 Kubernetes LoadBalancer 서비스 내에 지정된 NodePort에서 트래픽을 전달합니다. 각 개별 노드의 kube-proxy는 NodePort 포트에서 서비스 뒤에 있는 Pod로 트래픽을 전달하도록 설정합니다.

모듈 요약



aws

이 모듈에서 학습한 내용:

- Amazon EKS는 CNI를 사용하여 서로 다른 노드에 있는 Pod가 통신할 수 있도록 합니다.
- 네트워크 정책을 사용하면 Pod 수준에서 액세스를 세부적으로 조정할 수 있습니다.
- Kubernetes 서비스는 kube-proxy를 사용하여 노드의 운영 체제 커널에서 동일한 애플리케이션이 있는 Pod로 트래픽을 라우팅하는 규칙을 설정합니다.

©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

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