OpenShift Container Platform 4.19

네트워킹 개요

OpenShift Container Platform의 기본 네트워킹 개념 및 일반 작업 이해

Last Updated: 2025-12-10

OpenShift Container Platform 4.19 네트워킹 개요

OpenShift Container Platform의 기본 네트워킹 개념 및 일반 작업 이해

Legal Notice

Abstract

이 문서에서는 OpenShift Container Platform 내의 핵심 네트워킹 개념, 기본 아키텍처 및 일반적인 네트워킹 작업에 대한 소개를 제공합니다.

Table of Contents

1장. 네트워킹 이해

OpenShift Container Platform의 네트워킹 기본 사항을 이해하면 클러스터 내에서 효율적이고 안전한 통신이 보장되며 효과적인 네트워크 관리에 필수적입니다. 사용자 환경에서 네트워킹의 핵심 요소에는 포드와 서비스의 통신 방법, IP 주소의 역할, 서비스 검색을 위한 DNS 사용 등이 포함됩니다.

1.1. OPENSHIFT 컨테이너 플랫폼의 네트워킹

OpenShift Container Platform은 클러스터 내의 다양한 구성 요소 간, 그리고 외부 클라이언트와 클러스터 간의 원활한 통신을 보장합니다. 네트워킹은 다음 핵심 개념 및 구성 요소를 사용합니다.

- 포드 간 통신
- 서비스
- DNS
- Ingress
- 네트워크 제어
- 로드 밸런싱

1.1.1. 네트워킹 서비스의 일반적인 관행

OpenShift Container Platform에서 서비스는 여러 개의 Pod가 해당 서비스를 제공하더라도 클라이언트가 사용할 수 있는 단일 IP 주소를 생성합니다. 이러한 추상화를 통해 클라이언트에 영향을 주지 않고 원활한 확장, 내결함성 및 롤링 업그레이드가 가능해집니다.

네트워크 보안 정책은 클러스터 내의 트래픽을 관리합니다. 네트워크 제어를 통해 네임스페이스 관리자는 포드에 대한 수신 및 송신 규칙을 정의할 수 있습니다. 네트워크 관리 정책을 사용하면 클러스터 관리자가 네임스페이스 정책을 설정하고, 네임스페이스 정책을 재정의하거나, 아무것도 정의되지 않은 경우 기본 정책을 설정할 수 있습니다.

송신 방화벽 구성은 포드에서 나가는 트래픽을 제어합니다. 이러한 구성 설정은 승인된 통신만 이루어지도록 보장합니다. 유입 노드 방화벽은 유입 트래픽을 제어하여 노드를 보호합니다. 또한, Universal Data Network는 클러스터 전반의 데이터 트래픽을 관리합니다.

1.1.2. 네트워킹 기능

OpenShift Container Platform은 다양한 네트워킹 기능과 향상된 기능을 제공합니다. 이러한 기능과 향상된 기능은 다음과 같습니다.

- Ingress Operator 및 Route API: OpenShift Container Platform에는 Ingress Controller API를 구현하는 Ingress Operator가 포함되어 있습니다. 이 구성 요소는 고급 라우팅 구성과 부하 분산을 지원하는 HAProxy 기반 Ingress 컨트롤러를 배포하고 관리하여 클러스터 서비스에 대한 외부 액세스를 활성화합니다. OpenShift Container Platform은 Route API를 사용하여 업스트림 Ingress 객체를 경로 객체로 변환합니다. 경로는 OpenShift Container Platform의 네트워킹에 따라 다르지만 타사 Ingress 컨트롤러를 사용할 수도 있습니다.
- 강화된 보안: OpenShift Container Platform은 송신 방화벽 및 수신 노드 방화벽과 같은 고급 네트워크 보안 기능을 제공합니다.
    - 송신 방화벽: 송신 방화벽은 클러스터 내의 포드에서 나가는 트래픽을 제어하고 제한합니다. 포드가 통신할 수 있는 외부 호스트 또는 IP 범위를 제한하는 규칙을 설정할 수 있습니다.

- 수신 노드 방화벽: 수신 노드 방화벽은 수신 방화벽 운영자가 관리하며 노드 수준에서 방화벽 규칙을 제공합니다. 클러스터 내의 특정 노드에 이 방화벽을 구성하여 해당 노드에 도달하기 전에 들어오는 트래픽을 필터링함으로써 노드를 위협으로부터 보호할 수 있습니다.

참고

OpenShift Container Platform은 또한 네트워크 정책, 관리자 네트워크 정책, 보안 컨텍스트 제약(SCC)과 같은 서비스를 구현하여 포드 간 통신을 보호하고 액세스 제어를 시행합니다.

- 역할 기반 액세스 제어(RBAC): OpenShift Container Platform은 Kubernetes RBAC를 확장하여 네트워크 리소스에 액세스하고 관리할 수 있는 사람에 대한 보다 세부적인 제어를 제공합니다. RBAC는 클러스터 내에서 보안과 규정 준수를 유지하는 데 도움이 됩니다.

- 다중 테넌시 지원: OpenShift Container Platform은 여러 사용자와 팀이 리소스를 격리하고 안전하게 유지하면서 동일한 클러스터를 공유할 수 있도록 다중 테넌시 지원을 제공합니다.

- 하이브리드 및 멀티 클라우드 기능: OpenShift Container Platform은 온프레미스, 클라우드 및 멀티 클라우드 환경에서 원활하게 작동하도록 설계되었습니다. 이러한 유연성 덕분에 조직은 다양한 인프라에 컨테이너화된 애플리케이션을 배포하고 관리할 수 있습니다.

- 관찰성 및 모니터링: OpenShift Container Platform은 네트워크 문제를 관리하고 해결하는 데 도움이 되는 통합된 관찰성 및 모니터링 도구를 제공합니다. 이러한 도구에는 네트워크 메트릭과 로그에 대한 역할 기반 액세스가 포함됩니다.

- 사용자 정의 네트워크(UDN): UDN을 사용하면 관리자가 네트워크 구성을 사용자 정의할 수 있습니다. UDN은 향상된 네트워크 격리 및 IP 주소 관리를 제공합니다.

- 송신 IP: 송신 IP를 사용하면 네임스페이스 내의 포드에서 발생하는 모든 송신 트래픽에 대해 고정된 소스 IP 주소를 할당할 수 있습니다. Egress IP는 외부 서비스에 대한 일관된 소스 IP 주소를 보장함으로써 보안과 액세스 제어를 개선할 수 있습니다. 예를 들어, 포드가 특정 IP 주소에서만 트래픽을 허용하는 외부 데이터베이스에 액세스해야 하는 경우 액세스 요구 사항을 충족하도록 해당 포드에 대한 송신 IP를 구성할 수 있습니다.

- 송신 라우터: 송신 라우터는 클러스터와 외부 시스템 간의 브리지 역할을 하는 포드입니다. 송신 라우터를 사용하면 포드의 트래픽을 다른 목적으로 사용되지 않는 특정 IP 주소를 통해 라우팅할 수 있습니다. 송신 라우터를 사용하면 액세스 제어를 시행하거나 특정 게이트웨이를 통해 트래픽을 라우팅할 수 있습니다.

1.2. 노드, 클라이언트 및 클러스터와의 네트워킹

노드는 클러스터에서 제어 평면 구성 요소, 워크로드 구성 요소 또는 둘 다를 실행할 수 있는 머신입니다. 노드는 물리적 서버이거나 가상 머신입니다. 클러스터는 컨테이너화된 애플리케이션을 실행하는 노드의 집합입니다. 클라이언트는 클러스터와 상호 작용하는 도구와 사용자입니다.

1.2.1. 노드란 무엇인가요?

노드는 컨테이너화된 애플리케이션을 실행하는 물리적 또는 가상 머신입니다. 노드는 포드를 호스팅하고 애플리케이션을 실행하기 위한 메모리, 스토리지 등의 리소스를 제공합니다. 노드는 포드 간의 통신을 가능하게 합니다. 각 포드에는 IP 주소가 할당됩니다. 동일 노드 내의 포드는 이러한 IP 주소를 사용하여 서로 통신할 수 있습니다. 노드는 포드가 클러스터 내의 서비스를 검색하고 통신할 수 있도록 하여 서비스 검색을 용이하게 합니다. 노드는 네트워크 트래픽을 포드 간에 분산하여 효율적인 부하 분산과 애플리케이션의 높은 가용성을 보장합니다. 노드는 내부 클러스터 네트워크와 외부 네트워크 사이에 브리지 역할을 하여 외부 클라이언트가 클러스터에서 실행되는 서비스에 액세스할 수 있도록 합니다.

1.2.2. 클러스터 이해

클러스터는 컨테이너화된 애플리케이션을 실행하기 위해 함께 작동하는 노드의 집합입니다. 이러한 노드에는 제어 평면 노드와 컴퓨팅 노드가 포함됩니다.

1.2.3. 외부 고객 이해

외부 클라이언트는 클러스터 내부에서 실행되는 서비스 및 애플리케이션과 상호 작용하는 클러스터 외부의 모든 엔터티입니다. 외부에는 최종 사용자, 외부 서비스, 외부 장치가 포함될 수 있습니다. 최종 사용자는 브라우저나 모바일 기기를 통해 클러스터에 호스팅된 웹 애플리케이션에 액세스하는 사람들입니다. 외부 서비스는 클러스터 내 서비스와 상호 작용하는 다른 소프트웨어 시스템이나 애플리케이션으로, 종종 API를 통해 이루어집니다. 외부 장치는 사물 인터넷(IoT) 장치와 같이 클러스터 서비스와 통신해야 하는 클러스터 네트워크 외부의 모든 하드웨어입니다.

1.3. 네트워킹 개념 및 구성 요소

OpenShift Container Platform의 네트워킹은 몇 가지 핵심 구성 요소와 개념을 사용합니다.

- 포드와 서비스는 쿠버네티스에서 배포 가능한 가장 작은 단위이며, 서비스는 포드 세트에 대해 안정적인 IP 주소와 DNS 이름을 제공합니다. 클러스터의 각 포드에는 고유한 IP 주소가 할당됩니다. 포드는 어떤 노드에 있든 상관없이 IP 주소를 사용하여 다른 포드와 직접 통신합니다. 포드 IP 주소는 포드가 파괴되고 생성될 때 변경됩니다. 서비스에는 고유한 IP 주소도 할당됩니다. 서비스는 서비스를 제공할 수 있는 포드와 연결됩니다. 서비스 IP 주소에 접근하면 해당 서비스를 지원하는 포드 중 하나로 트래픽을 전송하여 포드에 안정적으로 접근할 수 있는 방법이 제공됩니다.

- Route 및 Ingress API는 클러스터 내의 서비스로 HTTP, HTTPS 및 TLS 트래픽을 라우팅하는 규칙을 정의합니다. OpenShift Container Platform은 기본 설치의 일부로 Route 및 Ingress API를 제공하지만, 타사 Ingress 컨트롤러를 클러스터에 추가할 수 있습니다.

- CNI(Container Network Interface) 플러그인은 Pod 간 통신을 가능하게 하기 위해 Pod 네트워크를 관리합니다.

- 클러스터 네트워크 운영자(CNO) CNO는 클러스터의 네트워킹 플러그인 구성 요소를 관리합니다. CNO를 사용하면 Pod 네트워크 CIDR, 서비스 네트워크 CIDR 등의 네트워크 구성을 설정할 수 있습니다.

- DNS 운영자는 클러스터 내의 DNS 서비스를 관리하여 DNS 이름으로 서비스에 접근할 수 있도록 합니다.

- 네트워크 제어는 포드가 서로 통신하고 다른 네트워크 엔드포인트와 통신할 수 있는 방법을 정의합니다. 이러한 정책은 트래픽 흐름을 제어하고 포드 통신에 대한 규칙을 시행하여 클러스터의 보안을 유지하는 데 도움이 됩니다.

- 부하 분산은 네트워크 트래픽을 여러 서버에 분산시켜 안정성과 성능을 보장합니다.

- 서비스 검색은 서비스가 클러스터 내에서 서로를 찾아 통신할 수 있는 메커니즘입니다.

- Ingress Operator는 OpenShift Container Platform Route를 사용하여 라우터를 관리하고 클러스터 서비스에 대한 외부 액세스를 활성화합니다.

추가 리소스

- 네트워크 정책 정의

1.4. 포드가 통신하는 방식

포드는 통신에 IP 주소를 사용하고, 포드나 서비스의 IP 주소를 검색하기 위해 동적 이름 시스템(DNS)을 사용합니다. 클러스터는 어떤 통신이 허용되는지 제어하는 다양한 정책 유형을 사용합니다. 포드는 포드 간 통신, 서비스 간 통신, 이렇게 두 가지 방식으로 통신합니다.

1.4.1. 포드 간 통신

포드 간 통신은 클러스터 내에서 포드가 서로 통신할 수 있는 기능입니다. 이는 마이크로서비스와 분산 애플리케이션의 기능에 매우 중요합니다.

클러스터의 각 포드에는 다른 포드와 직접 통신하는 데 사용하는 고유한 IP 주소가 할당됩니다. 포드 간 통신은 포드가 데이터를 교환하거나 협업적으로 작업을 수행해야 하는 클러스터 내부 통신에 유용합니다. 예를 들어, Pod A는 Pod B의 IP 주소를 사용하여 Pod B에 직접 요청을 보낼 수 있습니다. 포드는 NAT(네트워크 주소 변환) 없이 플랫 네트워크에서 통신할 수 있습니다. 이를 통해 서로 다른 노드의 포드 간에 원활한 통신이 가능해집니다.

1.4.1.1. 예: 포드 간 통신 제어

여러 개의 Pod가 있는 마이크로서비스 기반 애플리케이션에서 프런트엔드 Pod는 백엔드 Pod와 통신하여 데이터를 검색해야 합니다. 포드 간 통신을 직접 또는 서비스를 통해 사용하면 이러한 포드 간에 효율적으로 정보를 교환할 수 있습니다.

포드 간 통신을 제어하고 보호하기 위해 네트워크 제어를 정의할 수 있습니다. 이러한 제어는 레이블과 선택기를 기반으로 포드가 서로 상호 작용하는 방식을 지정하여 보안 및 규정 준수 요구 사항을 강화합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-some-pods
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: backend
      ports:
        - protocol: TCP
          port: 80
```

1.4.2. 서비스-포드 통신

서비스-포드 통신은 서비스가 적절한 포드로 트래픽을 안정적으로 라우팅할 수 있도록 보장합니다. 서비스는 논리적인 포드 세트를 정의하고 IP 주소 및 DNS 이름과 같은 안정적인 엔드포인트를 제공하는 객체입니다. Pod IP 주소는 변경될 수 있습니다. 서비스는 IP 주소가 변경되더라도 애플리케이션 구성 요소에 액세스할 수 있는 일관된 방법을 제공하기 위해 포드 IP 주소를 추상화합니다.

서비스-포드 통신의 주요 개념은 다음과 같습니다.

- 엔드포인트: 엔드포인트는 서비스와 연결된 포드의 IP 주소와 포트를 정의합니다.
- 선택기: 선택기는 키-값 쌍과 같은 레이블을 사용하여 서비스가 타겟으로 삼아야 할 객체 집합을 선택하기 위한 기준을 정의합니다.

- 서비스: 서비스는 포드 세트에 대한 안정적인 IP 주소와 DNS 이름을 제공합니다. 이러한 추상화를 통해 개별 포드가 아닌 다른 구성 요소가 서비스와 통신할 수 있습니다.
- 서비스 검색: DNS는 서비스를 검색 가능하게 만듭니다. 서비스가 생성되면 DNS 이름이 지정됩니다. 다른 포드는 이 DNS 이름을 발견하고 이를 사용하여 서비스와 통신합니다.
- 서비스 유형: 서비스 유형은 서비스가 클러스터 내부 또는 외부에 노출되는 방식을 제어합니다.
  - ClusterIP는 내부 클러스터 IP에서 서비스를 제공합니다. 이는 기본 서비스 유형이며 클러스터 내부에서만 서비스에 접근할 수 있도록 합니다.
  - NodePort는 각 노드의 IP에서 정적 포트로 서비스를 노출하여 외부 트래픽이 서비스에 액세스할 수 있도록 합니다.
  - LoadBalancer는 클라우드 공급자의 로드 밸런서를 사용하여 서비스를 외부에 노출합니다.

서비스는 트래픽을 수신해야 하는 포드를 식별하기 위해 선택기를 사용합니다. 선택기는 포드의 레이블을 일치시켜 어떤 포드가 서비스에 속하는지 판별합니다. 예: 선택기 **app: myapp이** 있는 서비스는 라벨이 **app: myapp** 인 모든 포드로 트래픽을 라우팅합니다.

엔드포인트는 서비스 선택기와 일치하는 포드의 현재 IP 주소를 반영하도록 동적으로 업데이트됩니다. OpenShift Container Platform은 이러한 엔드포인트를 유지 관리하고 서비스가 트래픽을 올바른 Pod로 라우팅하도록 보장합니다.

통신 흐름은 Kubernetes의 서비스가 트래픽을 적절한 Pod로 라우팅할 때 발생하는 일련의 단계와 상호작용을 말합니다. 서비스-포드 통신의 일반적인 통신 흐름은 다음과 같습니다.

- 서비스 생성: 서비스를 생성할 때 서비스 유형, 서비스가 수신하는 포트, 선택기 레이블을 정의합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

- DNS 확인: 각 포드에는 다른 포드가 서비스와 통신하는 데 사용할 수 있는 DNS 이름이 있습니다. 예를 들어, 서비스 이름이 **my-app** 네임스페이스에 **my-service** 인 경우 해당 DNS 이름은 **my-service.my-app.svc.cluster.local** 입니다.
- 트래픽 라우팅: 포드가 서비스의 DNS 이름에 요청을 보내면 OpenShift Container Platform은 해당 이름을 서비스의 ClusterIP로 확인합니다. 그런 다음 서비스는 엔드포인트를 사용하여 트래픽을 선택기와 일치하는 포드 중 하나로 라우팅합니다.
- 부하 분산: 서비스는 기본적인 부하 분산도 제공합니다. 이들은 선택기와 일치하는 모든 포드에 들어오는 트래픽을 분산합니다. 이를 통해 트래픽이 너무 많아 어떤 포드에도 과부하가 걸리지 않습니다.

1.4.2.1. 예: 서비스-포드 통신 제어

클러스터는 프런트엔드와 백엔드라는 두 가지 구성 요소로 구성된 마이크로서비스 기반 애플리케이션을 실행합니다. 프런트엔드는 백엔드와 통신하여 데이터를 가져와야 합니다.

프로세스

1. 백엔드 서비스를 만듭니다.

```
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

2. 백엔드 포드를 구성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: backend-pod
  labels:
    app: backend
spec:
  containers:
    - name: backend-container
      image: my-backend-image
      ports:
        - containerPort: 8080
```

3. 프런트엔드 커뮤니케이션을 구축합니다.
   프런트엔드 포드는 이제 DNS 이름 **backend.default.svc.cluster.local** 을 사용하여 백엔드 서비스와 통신할 수 있습니다. 이 서비스는 트래픽이 백엔드 포드 중 하나로 라우팅되도록 보장합니다.

서비스-포드 통신은 포드 IP 관리의 복잡성을 추상화하고 클러스터 내에서 안정적이고 효율적인 통신을 보장합니다.

1.5. 지원되는 로드 밸런서

부하 분산은 들어오는 네트워크 트래픽을 여러 서버로 분산하여 단일 서버에 너무 많은 부하가 걸리지 않도록 보장함으로써 클러스터의 상태와 효율성을 유지합니다. 로드 밸런서는 로드 밸런싱을 수행하는 장치입니다. 이들은 클라이언트와 서버 사이에서 중개자 역할을 하며, 사전 정의된 규칙에 따라 트래픽을 관리하고 전달합니다.

OpenShift Container Platform은 다음 유형의 로드 밸런서를 지원합니다.

- 클래식 로드 밸런서(CLB)
- 탄력적 로드 밸런싱(ELB/NLB)
- 네트워크 로드 밸런서(NLB)

- 애플리케이션 로드 밸런서

ELB는 AWS 라우터의 기본 로드 밸런서 유형입니다. CLB는 자체 관리 환경의 기본값입니다. NLB는 AWS(ROSA)의 Red Hat OpenShift Service의 기본값입니다.

중요

애플리케이션 앞에는 ALB를 사용하지만 라우터 앞에는 사용하지 마세요. ALB를 사용하려면 AWS Load Balancer Operator 추가 기능이 필요합니다. 이 연산자는 모든 Amazon Web Services(AWS) 지역이나 모든 OpenShift Container Platform 프로필에서 지원되지 않습니다.

1.5.1. 로드 밸런서 구성

클러스터를 설치하는 동안 기본 로드 밸런서 유형을 정의할 수 있습니다. 설치 후에는 클러스터 설치 시 정의한 글로벌 플랫폼 구성에 포함되지 않은 특정 방식으로 동작하도록 인그레스 컨트롤러를 구성할 수 있습니다.

1.5.1.1. 기본 로드 밸런서 유형을 정의합니다.

클러스터를 설치할 때 사용할 로드 밸런서 유형을 지정할 수 있습니다. 클러스터 설치 시 선택한 로드 밸런서 유형은 전체 클러스터에 적용됩니다.

이 예제에서는 AWS에 배포된 클러스터에 대한 기본 로드 밸런서 유형을 정의하는 방법을 보여줍니다. 다른 지원되는 플랫폼에도 이 절차를 적용할 수 있습니다.

```
apiVersion: v1
kind: Network
metadata:
  name: cluster
platform:
  aws: 1
    lbType: classic 2
```

1 **플랫폼** 키는 클러스터를 배포한 플랫폼을 나타냅니다. 이 예제에서는 **aws** 를 사용합니다.

2 **lbType** 키는 로드 밸런서 유형을 나타냅니다. 이 예제에서는 **Classic** Load Balancer를 사용합니다.

1.5.1.2. Ingress Controller에 대한 로드 밸런서 동작 지정

클러스터를 설치한 후에는 Ingress Controller를 구성하여 서비스가 외부 네트워크에 노출되는 방식을 지정할 수 있습니다. 이를 통해 로드 밸런서의 설정과 동작을 더 효과적으로 제어할 수 있습니다.

참고

Ingress Controller에서 로드 밸런서 설정을 변경하면 설치 시 지정한 로드 밸런서 설정이 재정의될 수 있습니다.

```
apiVersion: v1
kind: Network
metadata:
  name: cluster
```

```
endpointPublishingStrategy:
  loadBalancer: 1
    dnsManagementPolicy: Managed
    providerParameters:
      aws:
        classicLoadBalancer: 2
          connectionIdleTimeout: 0s
        type: Classic
      type: AWS
    scope: External
  type: LoadBalancerService
```

1 `loadBalancer` 필드는 로드 밸런서 구성 설정을 지정합니다.

2 **classicLoadBalancer** 필드는 로드 밸런서를 **classic** 으로 설정하고 AWS의 CLB에 대한 특정 설정을 포함합니다.

1.6. 도메인 이름 시스템(DNS)

도메인 이름 시스템(DNS)은 www.example.com과 같은 사용자에게 친숙한 도메인 이름을 네트워크의 컴퓨터를 식별하는 IP 주소로 변환하는 데 사용되는 계층적이고 분산된 명명 시스템입니다. DNS는 서비스 검색과 이름 확인에 중요한 역할을 합니다.

OpenShift Container Platform은 DNS 이름을 통해 서비스에 접근할 수 있도록 내장 DNS를 제공합니다. 기본 IP 주소가 변경되더라도 안정적인 통신을 유지하는 데 도움이 됩니다. 포드를 시작하면 서비스 이름, IP 주소, 포트에 대한 환경 변수가 자동으로 생성되어 포드가 다른 서비스와 통신할 수 있습니다.

1.6.1. 주요 DNS 용어

- CoreDNS: CoreDNS는 DNS 서버이며 서비스와 포드에 대한 이름 확인을 제공합니다.
- DNS 이름: 서비스에는 네임스페이스와 이름을 기반으로 DNS 이름이 지정됩니다. 예를 들어, **기본** 네임스페이스에 있는 **my-service** 라는 서비스의 DNS 이름은 **my-service.default.svc.cluster.local** 입니다.
- 도메인 이름: 도메인 이름은 **example.com** 과 같이 웹사이트와 서비스에 접속하는 데 사용되는 인간 친화적인 이름입니다.
- IP 주소: IP 주소는 통신에 IP를 사용하는 컴퓨터 네트워크에 연결된 각 장치에 할당된 숫자 라벨입니다. IPv4 주소의 예는 **192.0.2.1** 입니다. IPv6 주소의 예는 **2001:0db8:85a3:0000:0000:8a2e:0370:7334** 입니다.
- DNS 서버: DNS 서버는 DNS 레코드를 저장하는 특수 서버입니다. 이러한 기록은 도메인 이름을 IP 주소에 매핑합니다. 브라우저에 도메인 이름을 입력하면 컴퓨터는 DNS 서버에 접속하여 해당 IP 주소를 찾습니다.
- 해결 프로세스: DNS 쿼리가 DNS 확인자에게 전송됩니다. 그런 다음 DNS 확인자는 일련의 DNS 서버에 연결하여 도메인 이름과 연결된 IP 주소를 찾습니다. 리졸버는 **<namespace>.svc.cluster.local** , **svc.cluster.local** , **cluster.local** 과 같은 일련의 도메인을 포함하는 이름을 사용하려고 시도합니다. 이 과정은 첫 번째 매치에서 멈춥니다. IP 주소는 브라우저로 반환된 후 해당 IP 주소를 사용하여 웹 서버에 연결됩니다.

1.6.2. 예: DNS 사용 사례

이 예에서 프런트엔드 애플리케이션은 한 포드 세트에서 실행되고 백엔드 서비스는 다른 포드 세트에서 실행됩니다. 프런트엔드 애플리케이션은 백엔드 서비스와 통신해야 합니다. 백엔드 포드에 안정적인 IP 주소와 DNS 이름을 제공하는 서비스를 만듭니다. 프런트엔드 포드는 개별 포드 IP 주소가 변경되더라도 이 DNS 이름을 사용하여 백엔드 서비스에 액세스합니다.

백엔드 포드에 대한 서비스를 생성하면 프런트엔드 포드가 백엔드 서비스와 통신하는 데 사용할 수 있는 안정적인 IP 및 DNS 이름인 **backend-service.default.svc.cluster.local이** 제공됩니다. 이렇게 설정하면 개별 포드 IP 주소가 변경되더라도 통신이 일관되고 안정적으로 유지됩니다.

다음 단계에서는 DNS를 사용하여 백엔드 서비스와 통신하도록 프런트엔드 포드를 구성하는 방법의 예를 보여줍니다.

1. 백엔드 서비스를 만듭니다.

   a. 백엔드 포드를 배포합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  labels:
    app: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend-container
          image: your-backend-image
          ports:
            - containerPort: 8080
```

   b. 백엔드 포드를 노출하는 서비스를 정의합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

2. 프런트엔드 포드를 생성합니다.

   a. 프런트엔드 포드를 정의합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  labels:
    app: frontend
  spec:
    replicas: 3
    selector:
    matchLabels:
      app: frontend
  template:
  metadata:
    labels:
      app: frontend
  spec:
    containers:
      - name: frontend-container
        image: your-frontend-image
        ports:
          - containerPort: 80
```

b. 클러스터에 Pod 정의를 적용합니다.

```
$ oc apply -f frontend-deployment.yaml
```

3. 백엔드와 통신하도록 프런트엔드를 구성합니다.
프런트엔드 애플리케이션 코드에서 백엔드 서비스의 DNS 이름을 사용하여 요청을 보냅니다. 예를 들어, 프런트엔드 애플리케이션이 백엔드 포드에서 데이터를 가져와야 하는 경우 애플리케이션에 다음 코드를 포함할 수 있습니다.

```
fetch('http://backend-service.default.svc.cluster.local/api/data')
  .then(response => response.json())
  .then(data => console.log(data));
```

1.7. 네트워크 제어

네트워크 제어는 포드가 서로 통신하고 다른 네트워크 엔드포인트와 통신할 수 있는 방법에 대한 규칙을 정의합니다. 네트워크 제어는 네트워크 수준에서 구현되어 허용된 트래픽만 포드 간에 흐르도록 보장합니다. 이는 트래픽 흐름을 제한하고 무단 액세스를 방지하여 클러스터 보안에 도움이 됩니다.

- 관리자 네트워크 정책(ANP): ANP는 클러스터 범위의 사용자 정의 리소스 정의(CRD)입니다. 클러스터 관리자는 ANP를 사용하여 클러스터 수준에서 네트워크 정책을 정의할 수 있습니다. 일반 네트워크 정책 개체를 사용하여 이러한 정책을 재정의할 수 없습니다. 이러한 정책은 클러스터 전체에 걸쳐 엄격한 네트워크 보안 규칙을 시행합니다. ANP는 관리자가 클러스터에 들어오고 나가는 트래픽을 제어할 수 있도록 유입 및 유출 규칙을 지정할 수 있습니다.

- 유출 방화벽: 유출 방화벽은 클러스터에서 나가는 유출 트래픽을 제한합니다. 이 방화벽을 사용하면 관리자는 클러스터 내부에서 포드가 액세스할 수 있는 외부 호스트를 제한할 수 있습니다. 특정 IP 범위, DNS 이름 또는 외부 서비스에 대한 트래픽을 허용하거나 거부하도록 송신 방화벽 정책을 구성할 수 있습니다. 이를 통해 외부 리소스에 대한 무단 액세스를 방지하고 허용된 트래픽만 클러스터에서 나갈 수 있도록 보장합니다.

- 수신 노드 방화벽: 수신 노드 방화벽은 클러스터 내 노드에 대한 수신 트래픽을 제어합니다. 이 방

화벽을 통해 관리자는 어떤 외부 호스트가 노드에 연결을 시작할 수 있는지 제한하는 규칙을 정의합니다. 이를 통해 노드를 무단 액세스로부터 보호하고 신뢰할 수 있는 트래픽만 클러스터에 도달할 수 있습니다.

1.8. 경로 및 진입

경로와 유입은 모두 애플리케이션을 외부 트래픽에 노출하는 데 사용됩니다. 그러나 그 목적은 약간 다르고, 기능도 다릅니다.

1.8.1. 라우트

경로는 외부 클라이언트가 이름을 통해 서비스에 도달할 수 있도록 호스트 이름에서 서비스를 노출하는 OpenShift Container Platform 리소스에 특화되어 있습니다.

경로는 호스트 이름을 서비스에 매핑합니다. 경로 이름 매핑을 통해 외부 클라이언트가 호스트 이름을 사용하여 서비스에 액세스할 수 있습니다. 경로는 서비스로 향하는 트래픽에 대한 부하 분산을 제공합니다. 경로에 사용된 호스트 이름은 라우터의 IP 주소로 확인됩니다. 그런 다음 경로는 트래픽을 적절한 서비스로 전달합니다. 클라이언트와 서비스 간 트래픽을 암호화하기 위해 SSL/TLS를 사용하여 경로를 보호할 수도 있습니다.

1.8.2. Ingress

Ingress는 부하 분산, SSL/TLS 종료, 이름 기반 가상 호스팅을 포함한 고급 라우팅 기능을 제공하는 리소스입니다. Ingress에 대한 몇 가지 주요 사항은 다음과 같습니다.

- HTTP/HTTPS 라우팅: Ingress를 사용하여 클러스터 내 서비스로 HTTP 및 HTTPS 트래픽을 라우팅하기 위한 규칙을 정의할 수 있습니다.
- 부하 분산: NGINX나 HAProxy와 같은 Ingress Controller는 사용자가 정의한 규칙에 따라 트래픽 라우팅과 부하 분산을 관리합니다.
- SSL/TLS 종료: SSL/TLS 종료는 백엔드 서비스로 전달하기 전에 들어오는 SSL/TLS 트래픽을 해독하는 프로세스입니다.
- 여러 도메인 및 경로: Ingress는 여러 도메인과 경로에 대한 트래픽 라우팅을 지원합니다.

1.8.3. 경로와 Ingress 비교

경로는 진입보다 더 많은 유연성과 고급 기능을 제공합니다. 이를 통해 복잡한 경로 시나리오에 적합한 경로가 만들어집니다. 특히 기본적인 외부 접근 요구 사항의 경우 경로를 설정하고 사용하는 것이 더 간단합니다. Ingress는 보다 간단하고 직접적인 외부 접근을 위해 자주 사용됩니다. 경로는 고급 라우팅과 SSL/TLS 종료가 필요한 보다 복잡한 시나리오에 사용됩니다.

1.8.4. 예: 웹 애플리케이션을 노출하기 위한 경로 및 인그레스 구성

OpenShift Container Platform 클러스터에서 웹 애플리케이션이 실행되고 있습니다. 외부 사용자가 애플리케이션에 접근할 수 있도록 하려고 합니다. 애플리케이션은 특정 도메인 이름을 통해 접근할 수 있어야 하며, 트래픽은 TLS를 사용하여 안전하게 암호화되어야 합니다. 다음 예제에서는 웹 애플리케이션을 외부 트래픽에 안전하게 노출하기 위해 경로와 유입을 모두 구성하는 방법을 보여줍니다.

1.8.4.1. 경로 구성

1. 새 프로젝트를 생성합니다.

```
$ oc new-project webapp-project
```

2. 웹 애플리케이션을 배포합니다.

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git --name=webapp
```

3. 경로를 사용하여 서비스를 노출합니다.

```
$ oc expose svc/webapp --hostname=webapp.example.com
```

4. TLS를 사용하여 경로를 보호합니다.

   a. 인증서와 키를 사용하여 TLS 비밀번호를 생성합니다.

   ```
   $ oc create secret tls webapp-tls --cert=path/to/tls.crt --key=path/to/tls.key
   ```

   b. TLS 비밀번호를 사용하도록 경로를 업데이트합니다.

   ```
   $ oc patch route/webapp -p '{"spec":{"tls":
   {"termination":"edge","certificate":"path/to/tls.crt","key":"path/to/tls.key"}}}'
   ```

1.8.4.2. 인그레스 구성

1. 인그레스 리소스를 생성합니다.
   클러스터에 Ingress Controller가 설치되어 실행 중인지 확인하세요.

2. 웹 애플리케이션에 대한 서비스를 만듭니다. 아직 만들어지지 않았다면 애플리케이션을 서비스로 공개합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
  namespace: webapp-project
spec:
  selector:
    app: webapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

3. 인그레스 리소스를 생성합니다.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  namespace: webapp-project
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
```

```
- host: webapp.example.com
  http:
    paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: webapp-service
            port:
              number: 80
```

4. TLS를 사용하여 침투를 보호하세요.

   a. 인증서와 키를 사용하여 TLS 비밀번호를 생성합니다.

   ```
   $ oc create secret tls webapp-tls --cert=path/to/tls.crt --key=path/to/tls.key -n webapp-project
   ```

   b. TLS 비밀번호를 사용하도록 ingress 리소스를 업데이트합니다.

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: webapp-ingress
     namespace: webapp-project
   spec:
     tls: (1)
       - hosts:
           - webapp.example.com
         secretName: webapp-tls (2)
     rules:
       - host: webapp.example.com
         http:
           paths:
             - path: /
               pathType: Prefix
               backend:
                 service:
                   name: webapp-service
                   port:
                     number: 80
   ```

   (1) **TLS** 섹션은 TLS 설정을 지정합니다.

   (2) **secretName** 필드는 TLS 인증서와 키가 포함된 Kubernetes 비밀의 이름입니다.

1.9. 보안 및 교통 관리

관리자는 **ClusterIP** , **NodePort** , **LoadBalaner** 와 같은 서비스 유형과 **Ingress** , **Route** 와 같은 API 리소스를 사용하여 애플리케이션을 외부 트래픽에 노출하고 네트워크 연결을 보호할 수 있습니다. Ingress Operator와 CNO(Cluster Network Operator)는 이러한 서비스와 리소스를 구성하고 관리하는 데 도움이 됩니다. Ingress Operator는 하나 이상의 Ingress Controller를 배포하고 관리합니다. 이러한 컨트롤러는 클러스터 내부의 서비스로 외부 HTTP 및 HTTPS 트래픽을 라우팅합니다. CNO는 Pod 네트워크, 서비스 네트워크, DNS를 포함한 클러스터 네트워크 구성 요소를 배포하고 관리합니다.

1.9.1. 애플리케이션 노출

ClusterIP는 클러스터 내부의 내부 IP에 서비스를 노출시켜 클러스터 내의 다른 서비스에서만 클러스터에 접근할 수 있도록 합니다. NodePort 서비스 유형은 각 노드의 IP에 있는 정적 포트에서 서비스를 제공합니다. 이 서비스 유형은 외부 트래픽이 서비스에 액세스하는 것을 허용합니다. 로드 밸런서는 일반적으로 MetalLB를 사용하는 클라우드 또는 베어 메탈 환경에서 사용됩니다. 이 서비스 유형은 외부 트래픽을 서비스로 라우팅하는 외부 부하 분산 장치를 제공합니다. 베어메탈 환경에서 MetalLB는 VIP와 ARP 공지 또는 BGP 공지를 사용합니다.

Ingress는 부하 분산, SSL/TLS 종료, 이름 기반 가상 호스팅과 같은 서비스에 대한 외부 액세스를 관리하는 API 객체입니다. NGINX나 HAProxy와 같은 Ingress Controller는 Ingress API를 구현하고 사용자 정의 규칙에 따라 트래픽 라우팅을 처리합니다.

1.9.2. 연결 보안

Ingress 컨트롤러는 백엔드 서비스로 전달하기 전에 들어오는 SSL/TLS 트래픽을 해독하기 위해 SSL/TLS 종료를 관리합니다. SSL/TLS 종료는 애플리케이션 포드에서 암호화/복호화 프로세스의 부담을 덜어줍니다. TLS 인증서를 사용하여 클라이언트와 서비스 간 트래픽을 암호화할 수 있습니다. **cert-manager** 와 같은 도구를 사용하면 인증서 배포 및 갱신을 자동화하여 인증서를 관리할 수 있습니다.

SNI 필드가 있는 경우 경로는 TLS 트래픽을 포드로 전달합니다. 이 프로세스를 통해 TCP를 실행하는 서비스가 HTTP/HTTPS뿐만 아니라 TLS를 사용하여 노출될 수 있습니다. 사이트 관리자는 인증서를 중앙에서 관리하고 애플리케이션 개발자가 허가 없이도 개인 키를 읽을 수 있도록 허용할 수 있습니다.

Route API는 클러스터 관리 인증서를 사용하여 라우터-포드 트래픽을 암호화할 수 있도록 합니다. 이를 통해 외부 인증서는 중앙에서 관리되고 내부 인증서는 암호화된 상태로 유지됩니다. 애플리케이션 개발자는 자신의 애플리케이션에 대한 고유한 개인 키를 받습니다. 이러한 키는 포드에 비밀 키로 장착될 수 있습니다.

네트워크 제어는 포드가 서로 및 다른 네트워크 엔드포인트와 통신할 수 있는 방법에 대한 규칙을 정의합니다. 이렇게 하면 클러스터 내의 트래픽 흐름을 제어하여 보안이 강화됩니다. 이러한 제어는 네트워크 플러그인 수준에서 구현되어 포드 간에 허용된 트래픽 흐름만 보장됩니다.

역할 기반 액세스 제어(RBAC)는 클러스터 내의 리소스에 누가 액세스할 수 있는지 권한을 관리하고 제어합니다. 서비스 계정은 API에 액세스하는 포드에 대한 ID를 제공합니다. RBAC를 사용하면 각 Pod가 수행할 수 있는 작업을 세부적으로 제어할 수 있습니다.

1.9.3. 예: 애플리케이션 노출 및 연결 보안

이 예에서 클러스터에서 실행되는 웹 애플리케이션은 외부 사용자가 액세스해야 합니다.

1. 서비스를 만들고 요구 사항에 맞는 서비스 유형을 사용하여 애플리케이션을 서비스로 노출합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: my-web-app
spec:
  type: LoadBalancer
  selector:
    app: my-web-app
  ports:
  - port: 80
    targetPort: 8080
```

2. HTTP/HTTPS 트래픽을 관리하고 서비스로 라우팅하기 위해 **Ingress** 리소스를 정의합니다.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: mywebapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-web-app
            port:
              number: 80
```

3. 보안되고 암호화된 연결을 보장하려면 Ingress에 TLS를 구성하세요.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  tls:
  - hosts:
    - mywebapp.example.com
    secretName: my-tls-secret
  rules:
  - host: mywebapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-web-app
            port:
              number: 80
```

1.9.4. 서비스 유형과 API 리소스 중에서 선택

서비스 유형과 API 리소스는 애플리케이션을 노출하고 네트워크 연결을 보호하는 데 다양한 이점을 제공합니다. 적절한 서비스 유형이나 API 리소스를 활용하면 애플리케이션이 노출되는 방식을 효과적으로 관리하고 내부 및 외부 클라이언트 모두에게 안전하고 안정적인 액세스를 보장할 수 있습니다.

OpenShift Container Platform은 다음과 같은 서비스 유형과 API 리소스를 지원합니다.

- 서비스 유형

    - **ClusterIP** 는 내부적으로만 노출되도록 설계되었습니다. 설정이 쉽고 클러스터 내 서비스에 액세스하기 위한 안정적인 내부 IP 주소를 제공합니다. **ClusterIP** 는 클러스터 내 서비스 간 통신에 적합합니다.

    - **NodePort는** 각 노드의 IP에서 정적 포트로 서비스를 노출하여 외부 액세스를 허용합니다. 설정이 간단하고 개발 및 테스트에 유용합니다. **NodePort** 는 클라우드 공급자의 로드 밸런서가 필요 없이 간단한 외부 액세스에 적합합니다.

    - **LoadBalancer는** 트래픽을 여러 노드에 분산하기 위해 외부 로드 밸런서를 자동으로 프로비저닝합니다. 안정적이고 가용성이 높은 액세스가 필요한 프로덕션 환경에 이상적입니다.

    - **ExternalName은** 서비스를 외부 DNS 이름에 매핑하여 클러스터 외부의 서비스에 해당 서비스의 DNS 이름을 사용하여 액세스할 수 있도록 합니다. 외부 서비스나 레거시 시스템을 클러스터와 통합하는 데 유용합니다.

    - 헤드리스 서비스는 안정적인 **ClusterIP를** 제공하지 않고 Pod IP 목록을 반환하는 DNS 이름입니다. 이 기능은 상태 저장 애플리케이션이나 개별 포드 IP에 직접 액세스해야 하는 시나리오에 이상적입니다.

- api-resources

    - **Ingress는** 부하 분산, SSL/TLS 종료, 이름 기반 가상 호스팅을 지원하는 등 HTTP 및 HTTPS 트래픽 라우팅에 대한 제어 기능을 제공합니다. 서비스만 사용하는 것보다 더 유연하며 여러 도메인과 경로를 지원합니다. 복잡한 라우팅이 필요한 경우에는 **Ingress가** 이상적입니다.

    - **Route는 Ingress** 와 비슷하지만 TLS 재암호화 및 패스스루를 포함한 추가 기능을 제공합니다. 서비스를 외부에 노출하는 과정이 간소화됩니다. **Route는** 통합 인증서 관리와 같은 고급 기능이 필요할 때 가장 적합합니다.

외부 트래픽에 서비스를 노출하는 간단한 방법이 필요한 경우 **Route** 나 **Ingress가** 가장 좋은 선택일 수 있습니다. 이러한 리소스는 네임스페이스 관리자나 개발자가 관리할 수 있습니다. 가장 쉬운 방법은 경로를 만들고, 외부 DNS 이름을 확인한 다음, 외부 DNS 이름을 가리키는 CNAME을 갖도록 DNS를 구성하는 것입니다.

HTTP/HTTPS/TLS의 경우 **Route** 또는 **Ingress로** 충분합니다. 그 외의 것은 더 복잡하며 클러스터 관리자가 포트에 액세스할 수 있는지 또는 MetalLB가 구성되어 있는지 확인해야 합니다. **LoadBalancer** 서비스는 클라우드 환경이나 적절하게 구성된 베어 메탈 환경에서도 사용할 수 있는 옵션입니다.

2장. 호스트에 액세스

배스천 호스트(Bastion Host)를 생성하여 OpenShift Container Platform 인스턴스에 액세스하고 SSH(Secure Shell) 액세스 권한으로 컨트롤 플레인 노드에 액세스하는 방법을 알아봅니다.

2.1. 설치 관리자 프로비저닝 인프라 클러스터에서 AMAZON WEB SERVICES의 호스트에 액세스

OpenShift Container Platform 설치 관리자는 OpenShift Container Platform 클러스터에 프로비저닝된 Amazon EC2(Amazon Elastic Compute Cloud) 인스턴스에 대한 퍼블릭 IP 주소를 생성하지 않습니다. OpenShift Container Platform 호스트에 SSH를 사용하려면 다음 절차를 따라야 합니다.

프로세스

1. **openshift-install** 명령으로 생성된 가상 프라이빗 클라우드(VPC)에 SSH로 액세스할 수 있는 보안 그룹을 만듭니다.

2. 설치 관리자가 생성한 퍼블릭 서브넷 중 하나에 Amazon EC2 인스턴스를 생성합니다.

3. 생성한 Amazon EC2 인스턴스와 퍼블릭 IP 주소를 연결합니다.
   OpenShift Container Platform 설치와는 달리, 생성한 Amazon EC2 인스턴스를 SSH 키 쌍과 연결해야 합니다. 이 인스턴스에서 사용되는 운영 체제는 중요하지 않습니다. 그저 인터넷을 OpenShift Container Platform 클러스터의 VPC에 연결하는 SSH 베스천의 역할을 수행하기 때문입니다. 사용하는 AMI(Amazon 머신 이미지)는 중요합니다. 예를 들어, RHCOS(Red Hat Enterprise Linux CoreOS)를 사용하면 설치 프로그램과 마찬가지로 Ignition을 통해 키를 제공할 수 있습니다.

4. Amazon EC2 인스턴스를 프로비저닝한 후 SSH로 연결할 수 있는 경우 OpenShift Container Platform 설치와 연결된 SSH 키를 추가해야 합니다. 이 키는 베스천 인스턴스의 키와 다를 수 있지만 반드시 달라야 하는 것은 아닙니다.

참고

SSH 직접 액세스는 재해 복구 시에만 권장됩니다. Kubernetes API가 응답할 때는 권한 있는 Pod를 대신 실행합니다.

5. **oc get nodes**를 실행하고 출력을 확인한 후 마스터 노드 중 하나를 선택합니다. 호스트 이름은 **ip-10-0-1-163.ec2.internal**과 유사합니다.

6. Amazon EC2에 수동으로 배포한 베스천 SSH 호스트에서 해당 컨트롤 플레인 호스트에 SSH로 연결합니다. 설치 중 지정한 것과 동일한 SSH 키를 사용해야 합니다.

```
$ ssh -i <ssh-key-path> core@<master-hostname>
```

3장. 네트워킹 대시보드

네트워킹 메트릭은 OpenShift Container Platform 웹 콘솔의 대시보드에서 **Observe → Dashboards** 에서 볼 수 있습니다.

3.1. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator가 설치되어 있으면 **대시보드** 드롭다운 목록에서 **Netobserv** 대시보드를 선택하여 네트워크 트래픽 메트릭 대시보드를 볼 수 있습니다. 이 **대시보드** 에서 사용할 수 있는 메트릭에 대한 자세한 내용은 네트워크 관찰 메트릭 대시보드를 참조하세요.

3.2. 네트워킹 및 OVN-KUBERNETES 대시보드

대시보드에서 일반 네트워킹 지표와 OVN-Kubernetes 지표를 모두 볼 수 있습니다.

일반적인 네트워킹 지표를 보려면 대시 **보드** 드롭다운 목록에서 **네트워킹/Linux 하위 시스템 통계**를 선택하세요. 대시보드에서 **네트워크 활용도** , **네트워크 포화도** , **네트워크 오류** 등의 네트워킹 지표를 볼 수 있습니다.

OVN-Kubernetes 메트릭을 보려면 **대시보드** 드롭다운 목록에서 **네트워킹/인프라**를 선택하세요. 다음 OVN-Kuberenetes 메트릭을 볼 수 있습니다: **네트워킹 구성** , **TCP 지연 프로브** , **제어 평면 리소스** 및 **작업자 리소스** .

3.3. INGRESS 운영자 대시보드

대시보드에서 Ingress Operator가 처리하는 네트워킹 지표를 볼 수 있습니다. 여기에는 다음과 같은 측정 항목이 포함됩니다.

- 수신 및 발신 대역폭
- HTTP 오류율
- HTTP 서버 응답 지연

이러한 Ingress 메트릭을 보려면 **대시보드** 드롭다운 목록에서 **네트워킹/Ingress**를 선택하세요. 다음 카테고리에 대한 Ingress 메트릭을 볼 수 있습니다: **경로당 상위 10개** , **네임스페이스당 상위 10개** , **샤드당 상위 10개** .

4장. CIDR 범위 정의

클러스터가 OVN-Kubernetes를 사용하는 경우 CIDR(Classless Inter-Domain Routing) 서브넷 범위에 대해 겹치지 않는 범위를 지정해야 합니다.

중요

OpenShift Container Platform 4.17 이상 버전의 경우 클러스터는 IPv4의 경우 **169.254.0.0/17**을 , IPv6의 경우 **fd69::/112**를 기본 마스커레이드 서브넷으로 사용합니다. 사용자는 이러한 범위를 피해야 합니다. 업그레이드된 클러스터의 경우 기본 마스커레이드 서브넷에는 변경 사항이 없습니다.

작은 정보

클러스터 생성 중에 CIDR 범위를 설정하기 전에 Red Hat OpenShift Network Calculator를 사용하여 네트워킹 요구 사항을 결정할 수 있습니다.

계산기를 사용하려면 Red Hat 계정이 있어야 합니다.

다음 서브넷 유형은 OVN-Kubernetes를 사용하는 클러스터에 필수입니다.

- 조인: 조인 스위치를 사용하여 게이트웨이 라우터를 분산 라우터에 연결합니다. 조인 스위치는 분산 라우터의 IP 주소 수를 줄입니다. OVN-Kubernetes 플러그인을 사용하는 클러스터의 경우 전용 서브넷의 IP 주소가 조인 스위치에 연결된 모든 논리적 포트에 할당됩니다.
- 마스커레이드: 로드 밸런서가 라우팅 결정을 내린 후, 노드에서 헤어핀 트래픽으로 동일한 노드로 전송되는 동일한 소스 및 대상 IP 주소에 대한 충돌을 방지합니다.
- 트랜짓: 트랜짓 스위치는 클러스터의 모든 노드에 걸쳐 있는 일종의 분산 스위치입니다. 교통 교환기는 서로 다른 구역 사이의 교통을 라우팅합니다. OVN-Kubernetes 플러그인을 사용하는 클러스터의 경우 전용 서브넷의 IP 주소가 전송 스위치에 연결된 모든 논리적 포트에 할당됩니다.

참고

클러스터의 조인, 마스커레이드, 전송 CIDR 범위를 설치 후 작업으로 변경할 수 있습니다.

OpenShift Container Platform 4.14 이상 버전의 기본 네트워크 공급자인 OVN-Kubernetes는 내부적으로 다음 IP 주소 서브넷 범위를 사용합니다.

- **V4JoinSubnet**: **100.64.0.0/16**
- **V6JoinSubnet**: **fd98::/64**
- **V4TransitSwitchSubnet**: **100.88.0.0/16**
- **V6TransitSwitchSubnet**: **fd97::/64**
- **defaultV4MasqueradeSubnet**: **169.254.0.0/17**
- **defaultV6MasqueradeSubnet**: **fd69::/112**

중요

이전 목록에는 IPv4 및 IPv6 주소 서브넷의 조인, 전송 및 위장이 포함되어 있습니다. 클러스터가 OVN-Kubernetes를 사용하는 경우 클러스터나 인프라의 다른 CIDR 정의에 이러한 IP 주소 서브넷 범위를 포함하지 마세요.

추가 리소스

- 조인 서브넷 또는 전송 서브넷 구성에 대한 자세한 내용은 OVN-Kubernetes 내부 IP 주소 서브넷 구성을 참조하세요.

4.1. 기계 CIDR

머신 클래스리스 도메인 간 라우팅(CIDR) 필드에서는 머신이나 클러스터 노드의 IP 주소 범위를 지정해야 합니다.

참고

클러스터를 생성한 후에는 머신 CIDR 범위를 변경할 수 없습니다.

기본값은 **128** 입니다. 이 범위는 연결된 네트워크와 충돌해서는 안 됩니다.

추가 리소스

- CNO(Cluster Network Operator) 구성

4.2. 서비스 CIDR

서비스 CIDR 필드에서는 서비스의 IP 주소 범위를 지정해야 합니다. 범위는 작업량을 수용할 만큼 충분히 커야 합니다. 주소 블록은 클러스터 내부에서 접근하는 외부 서비스와 겹치면 안 됩니다. 기본값은 **172.30.0.0/16**입니다.

4.3. 포드 CIDR

Pod CIDR 필드에서는 Pod의 IP 주소 범위를 지정해야 합니다.

Pod CIDR은 **clusterNetwork** CIDR 및 cluster CIDR과 동일합니다. 범위는 작업량을 수용할 만큼 충분히 커야 합니다. 주소 블록은 클러스터 내부에서 접근하는 외부 서비스와 겹치면 안 됩니다. 기본값은 128 입니다. 클러스터 설치 후 범위를 확장할 수 있습니다.

추가 리소스

- CNO(Cluster Network Operator) 구성
- 클러스터 네트워크 범위 구성

4.4. 호스트 접두사

hostPrefix 매개변수에서는 개별 머신에 예약된 포드에 할당된 서브넷 접두사 길이를 지정해야 합니다. 호스트 접두사는 각 머신의 Pod IP 주소 풀을 결정합니다.

예를 들어, 호스트 접두사가 **/23** 으로 설정된 경우 각 머신에는 Pod CIDR 주소 범위에서 **/23** 서브넷이 할당됩니다. 기본값은 **/23** 이며, 이를 통해 510개의 클러스터 노드와 노드당 510개의 Pod IP 주소를 허용합니다.

clusterNetwork.cidr 매개변수를 **10.128.0.0/16** 으로 설정하는 또 다른 예를 살펴보겠습니다. 이 경우 클러스터의 전체 주소 공간이 정의됩니다. 이렇게 하면 클러스터에 65536개의 IP 주소 풀이 할당됩니다. 그런 다음 **hostPrefix** 매개변수를 **/23** 으로 설정하면 클러스터의 각 노드에 서브넷 슬라이스가 정의됩니다. 여기서 **/23** 슬라이스는 **/16** 서브넷 네트워크의 서브넷이 됩니다. 이를 통해 각 노드에 512개의 IP 주소가 할당되고, 그 중 2개의 IP 주소는 네트워킹 및 브로드캐스팅 목적으로 예약됩니다. 다음 계산 예시에서는 이러한 IP 주소 수치를 사용하여 클러스터에 대해 생성할 수 있는 최대 노드 수를 결정합니다.

```
65536 / 512 = 128
```

Red Hat OpenShift Network Calculator를 사용하면 클러스터의 최대 노드 수를 계산할 수 있습니다.

4.5. 호스팅된 제어 평면의 CIDR 범위

OpenShift Container Platform에 호스팅된 제어 평면을 배포하려면 다음과 같은 필수 CIDR(Classless Inter-Domain Routing) 서브넷 범위를 사용하세요.

- **v4InternalSubnet** : 100.65.0.0/16(OVN-Kubernetes)
- **clusterNetwork** : 10.132.0.0/14(pod 네트워크)
- **serviceNetwork**: 172.31.0.0/16

OpenShift Container Platform CIDR 범위 정의에 대한 자세한 내용은 "CIDR 범위 정의"를 참조하세요.