



쿠버네티스의 네임스페이스 및 서비스 관리

쿠버네티스 아키텍처에 대한 깊이 있는 이해를 바탕으로, 네임스페이스와 서비스 관리의 핵심을 탐구합니다. 명령형과 선언형 접근 방식을 비교하며, 실제 적용 사례를 통해 학습합니다.

○ by 환준 남



네임스페이스 개념

논리적 구분

쿠버네티스 클러스터 내 자원을 논리적으로 분리하여 관리합니다. 복잡한 환경에서 효율적인 리소스 관리가 가능해집니다.

기본 네임스페이스

default, kube-system, kube-public 등 기본적으로 제공되는 네임스페이스가 있습니다. 각각 고유한 목적을 가집니다.

자원 격리

네임스페이스별로 독립적인 환경을 제공하여 자원 충돌을 방지하고 보안을 강화합니다.

네임스페이스 비유



독립된 생활 공간

각 집은 네임스페이스를, 거주자는 리소스를 나타냅니다. 독립적인 규칙과 자원 관리가 가능합니다.



고유한 환경

각 네임스페이스는 고유한 설정과 리소스를 가집니다. 이는 집 안의 독특한 인테리어와 소지품에 비유됩니다.



네임스페이스 사용 사례

1

개발 환경 분리

Dev와 Production 환경을 동일 클러스터 내에서 논리적으로 분리합니다. 이를 통해 리소스 충돌을 방지하고 보안을 강화합니다.

2

팀별 자원 할당

각 팀에 별도의 네임스페이스를 할당하여 독립적인 자원 관리가 가능합니다. 팀 간 간섭을 최소화합니다.

3

접근 권한 관리

네임스페이스별로 세분화된 RBAC(Role-Based Access Control)를 적용할 수 있습니다. 보안 정책을 효과적으로 구현합니다.

네임스페이스 간 통신

1

동일 네임스페이스 내 통신

서비스 이름만으로 간단히 자원을 참조할 수 있습니다. 예: 'database-service'

2

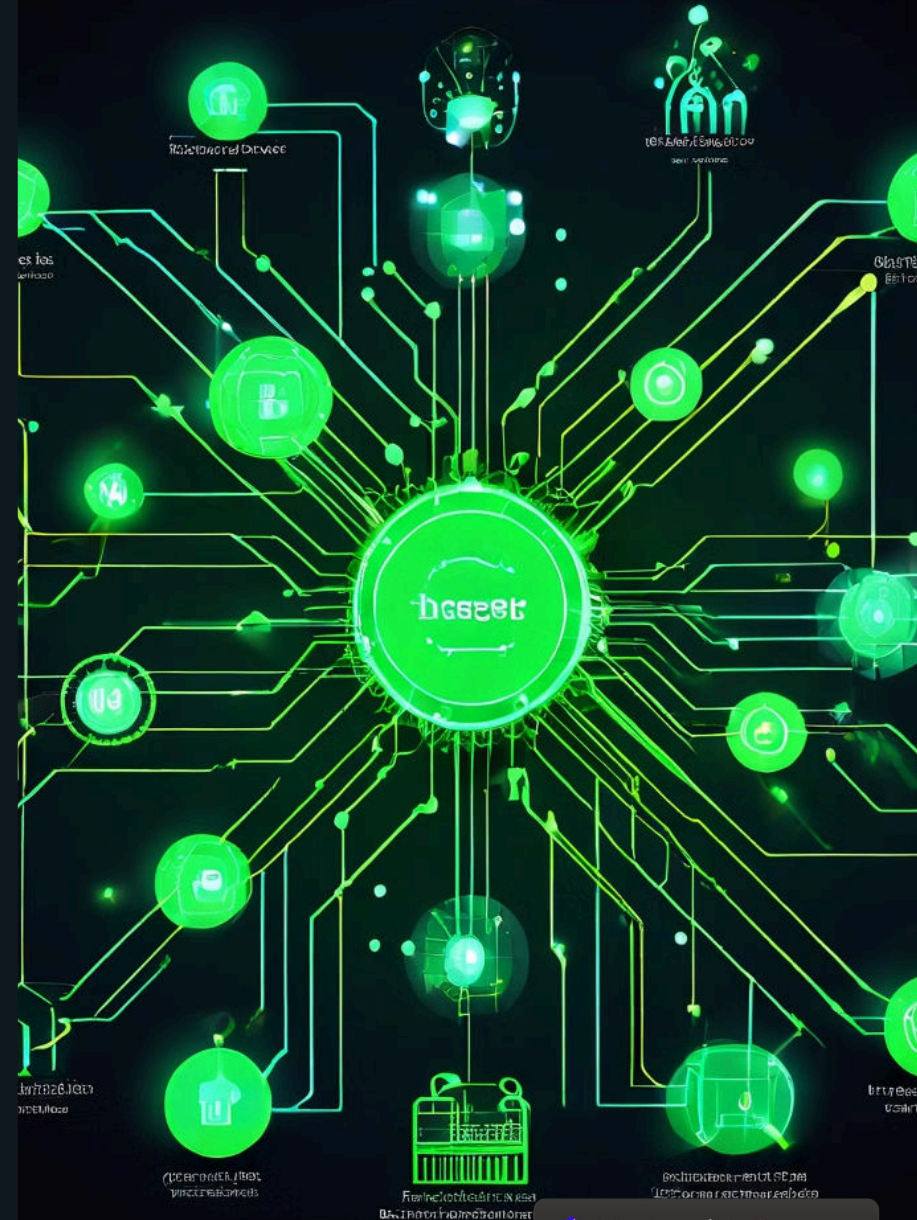
다른 네임스페이스 참조

FQDN(Fully Qualified Domain Name) 형식을 사용합니다. 예: 'servicename.namespace.svc.cluster.local'

3

네트워크 정책 적용

필요에 따라 네임스페이스 간 통신을 제한하거나 허용할 수 있습니다. 세밀한 네트워크 제어가 가능합니다.



네임스페이스 관련 명령어

명령어	설명
<code>kubectl get pods</code>	기본 네임스페이스의 파드 조회
<code>kubectl get pods --namespace=kube-system</code>	특정 네임스페이스의 파드 조회
<code>kubectl create namespace <name></code>	새로운 네임스페이스 생성
<code>kubectl delete namespace <name></code>	네임스페이스 삭제 (주의 요망)

쿠버네티스 서비스 유형

NodePort

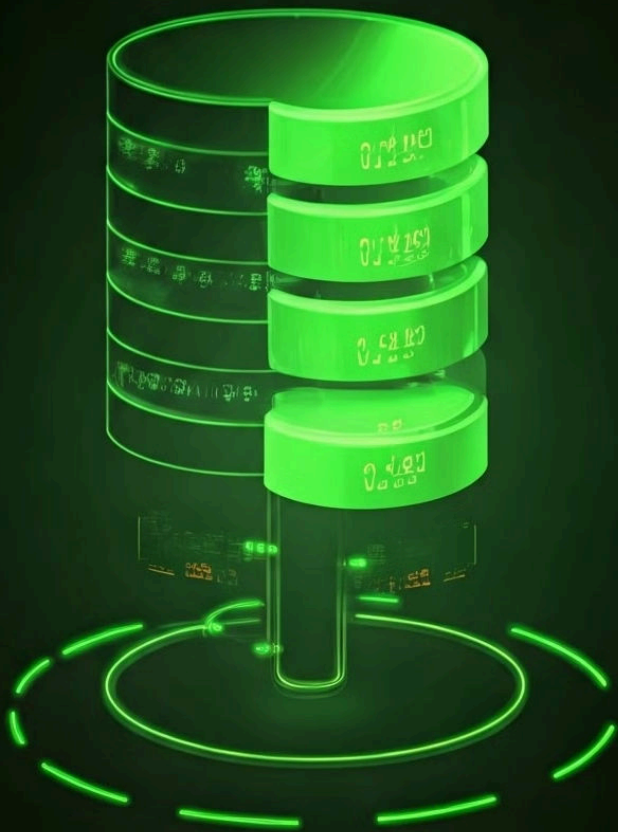
노드의 특정 포트를 통해 외부에서 접근 가능합니다. 개발 및 테스트 환경에서 주로 사용됩니다.

ClusterIP

클러스터 내부에서만 접근 가능한 서비스입니다. 내부 마이크로서비스 간 통신에 적합합니다.

LoadBalancer

클라우드 제공업체의 로드 밸런서를 사용하여 외부 접근을 제공합니다. 프로덕션 환경에 적합합니다.



NodePort

1

포트 매핑

노드의 포트와 파드의 포트를 매핑하여 외부 접근을 허용합니다. 30000-32767 범위의 포트를 사용합니다.

2

구성 예시

Voting 앱은 31000 포트, Result 앱은 31001 포트에 설정하여 외부에서 접근 가능하게 합니다.

3

사용 시나리오

개발 환경이나 소규모 클러스터에서 간단한 외부 접근이 필요할 때 유용합니다.

ClusterIP



내부 접근

클러스터 내부에서만 접근 가능한 서비스 유형입니다. 외부로부터 격리된 환경을 제공합니다.



백엔드 서비스

데이터베이스나 내부 API 서버와 같은 백엔드 서비스에 적합합니다. 보안성이 향상됩니다.



마이크로서비스

마이크로서비스 아키텍처에서 서비스 간 통신에 이상적입니다. 네트워크 복잡성을 줄입니다.



LoadBalancer

1

외부 로드 밸런서

클라우드 제공업체의 로드 밸런서를 자동으로 프로비저닝합니다. 고가용성과 확장성을 제공합니다.

2

단일 엔드포인트


외부 사용자에게 단일 IP 주소나 도메인을 제공합니다. 접근성과 관리 용이성이 향상됩니다.

3

트래픽 분산

들어오는 트래픽을 여러 파드에 균등하게 분산시킵니다. 시스템의 안정성과 성능을 개선합니다.






쿠버네티스 관리: 명령 형 vs 선언형 접근

쿠버네티스 관리에는 두 가지 주요 접근 방식이 있습니다: 명령형과 선언형. 이 강의에서는 각 방식의 특징, 장단점, 그리고 실제 적용 방법을 살펴봅니다. 효율적인 쿠버네티스 관리를 위한 핵심 전략을 습득하실 수 있습니다.

○ by 환준 남



쿠버네티스 관리: 명령 형 vs 선언형 접근

쿠버네티스 관리에는 두 가지 주요 접근 방식이 있습니다: 명령형과 선언형. 이 강의에서는 각 방식의 특징, 장단점, 그리고 실제 적용 방법을 살펴봅니다. 효율적인 쿠버네티스 관리를 위한 핵심 전략을 습득하실 수 있습니다.

○ by 환준 남

명령형 접근 방식

1 **kubectl run**

새로운 파드나 디플로이먼트를 빠르게 생성합니다. 개발 단계에서 유용합니다.

2 **kubectl create**

다양한 쿠버네티스 리소스를 생성합니다. 복잡한 구성이 필요 없을 때 적합합니다.

3 **kubectl expose**

기존 리소스를 서비스로 노출시킵니다. 빠른 테스트에 적합합니다.

4 **kubectl edit**

실행 중인 리소스의 구성을 직접 수정합니다. 긴급한 변경에 유용합니다.



선언형 접근 방식

YAML 파일 사용

객체의 기대 상태를 상세히 정의합니다.
버전 관리가 용이하며 재사용성이 높습니다.

kubectl apply

정의된 상태로 객체를 생성하거나 업데이트합니다. 일관된 관리가 가능합니다.

GitOps 지원

인프라를 코드로 관리할 수 있어 CI/CD 파이프라인과의 통합이 용이합니다.



kubectl apply 명령어의 작동 방식

1

로컬 구성 파일 분석

사용자가 정의한 YAML 파일의 내용을 파악합니다.

2

라이브 객체 상태 확인

클러스터에 실제 존재하는 객체의 현재 상태를 조회합니다.

3

마지막 적용 구성 비교

이전에 적용된 구성과 현재 구성을 비교합니다.

4

변경 사항 적용

필요한 변경만을 클러스터에 적용합니다.

선언형 접근 방식의 장점

변경 사항 추적

Git과 같은 버전 관리 시스템을 통해 구성 변경 이력을 쉽게 관리할 수 있습니다.

일괄 처리

여러 객체의 구성을 하나의 YAML 파일로 관리하여 복잡한 애플리케이션 배포가 용이합니다.

자동화

시스템이 현재 상태와 목표 상태를 비교하여 필요한 작업을 자동으로 수행합니다.

일관성

모든 환경에서 동일한 구성을 사용하여 개발, 테스트, 프로덕션 간 일관성을 유지합니다.

Kubernetes

