



NHN Cloud 컨테이너 보안 가이드

NHN Cloud
Container Security Guide



NHN Cloud 컨테이너 보안 가이드

저작권

Copyright NHN Cloud Corp. All rights reserved.

이 문서는 NHN Cloud의 지적 자산이므로 NHN Cloud의 승인 없이 문서를 다른 용도로 임의 변경하여 사용할 수 없습니다. 이 문서는 정보 제공의 목적으로만 제공됩니다. NHN Cloud는 이 문서에 수록된 정보의 완전성과 정확성을 검증하기 위해 노력하였으나, 발생할 수 있는 내용상의 오류나 누락에 대해서는 책임지지 않습니다. 따라서 이 문서의 사용이나 사용 결과에 따른 책임은 전적으로 사용자에게 있으며, NHN Cloud는 이에 대해 명시적 혹은 묵시적으로 어떠한 보증도 하지 않습니다. 관련 URL 정보를 포함하여 이 문서에서 언급한 특정 소프트웨어 상품이나 제품은 해당 소유자의 저작권법을 따르며, 해당 저작권법을 준수하는 것은 사용자의 책임입니다.

NHN Cloud는 이 문서의 내용을 예고 없이 변경할 수 있습니다.

문서 이력

버전	일자	이력 사항
1.0 버전	2024. 7.	NHN Cloud 컨테이너 보안 가이드 1.0 버전 출시
1.1 버전	2024. 8.	신규 CI 적용
1.2 버전	2024. 9.	파드 네트워크 접근 통제 방법 추가
1.3 버전	2025. 1.	외부 취약점 검사 모듈 연동 방식 설명 업데이트

목차

NHN Cloud 컨테이너 보안 가이드 2

저작권 2

문서 이력 2

1 개요 5

1.1 가이드 소개 5

1.2 용어 6

2 컨테이너 8

2.1 정의 8

2.2 컨테이너 주요 요소 개념 9

 2.2.1 주요 요소 및 NHN Cloud 컨테이너 서비스 9

 2.2.2 컨테이너 가상화 10

 2.2.3 컨테이너 이미지와 이미지 레지스트리 11

 2.2.4 컨테이너 런타임 12

 2.2.5 컨테이너 오케스트레이션(Kubernetes) 13

3 컨테이너 보안 위협 18

3.1 컨테이너 보안 위협 정의	18
3.2 컨테이너 보안 위협	19
3.2.1 노드 및 OS, 클라우드 자원의 위협	19
3.2.2 이미지 보안 위협	20
3.2.3 오케스트레이터 및 컨테이너 보안 위협	20
3.2.4 배포 및 운영 위협	21

4 컨테이너 보안 적용 방안 22

4.1 컨테이너 보안 적용 방안 개요	22
4.2 노드와 OS 및 관리 콘솔 보안	23
4.3 이미지와 레지스트리 보안	25
4.4 오케스트레이터 및 컨테이너 보안	32
4.5 모니터링 및 위협 탐지	57

5 컨테이너 보안 아키텍처 62

1 개요

1.1 가이드 소개

NHN Cloud에서는 컨테이너 환경에서 다양한 애플리케이션을 개발, 운영할 수 있도록 NKS(NHN Kubernetes Service), NCR(NHN Container Registry), NCS(NHN Container Service)와 같은 서비스를 제공합니다.

본 가이드에서는 NHN Cloud에서 제공하는 NKS와 NCR을 기반으로 컨테이너 및 Kubernetes 보안 방법을 소개합니다. 가이드에서 안내하는 내용은 다음과 같습니다.

- 컨테이너에 대한 개념과 주요 구성 요소에 대해 설명합니다.
- 컨테이너 환경의 보안 위협을 식별하고 보호 대상을 확인합니다.
- 컨테이너 환경과 컨테이너를 기반으로 서비스하는 워크로드를 보호할 수 있는 방법을 소개합니다.

NHN Cloud의 NKS, NCR 서비스로 생성한 컨테이너 이미지, 파드, 컨테이너 정책, 설정 등은 클라우드 사용자의 운영 관리가 필요한 영역으로 해당 가이드에서 제공하는 방법 외에도 고객의 자체적인 보안 거버넌스에 따라 주기적으로 컨테이너 환경 전반의 보안성 검토와 모니터링을 수행하고, 컨테이너 환경의 보안 위험을 최소화하기 위한 노력이 필요합니다. IaaS, PaaS, SaaS와 같은 클라우드 책임 공유 모델에 대한 내용은 [NHN Cloud 보안 센터](#)에서 추가로 확인하실 수 있습니다.

1.2 용어

컨테이너와 관련된 주요 용어를 설명합니다.

표 1-1 컨테이너 관련 주요 용어

분류	용어	설명
가상화	컨테이너 런타임	컨테이너를 실행하고 관리하는 데 필요한 기능을 제공하는 환경
	네임스페이스	Kubernetes 클러스터 내의 논리적인 공간으로, 리소스의 범위를 구분하여 격리 및 관리
	cgroups	컨트롤 그룹(control groups)의 약자로 리소스(예: CPU, 메모리, 디스크 I/O 등)를 그룹화하고 제한하는 기능을 제공하는 Linux 커널 기술
이미지	Dockerfile	컨테이너 이미지를 빌드하기 위한 텍스트 기반 스크립트 파일
	이미지 레지스트리	컨테이너 이미지를 저장, 관리 및 배포하는 데 사용되는 저장소
	이미지	애플리케이션을 실행하기 위해 필요한 모든 소프트웨어 및 설정이 패키징된 파일
Kubernetes	컨트롤 플레이언	Kubernetes 클러스터의 제어를 담당하는 주요 노드로 클러스터 상태를 관리하고 파드(pod)의 스케줄링을 담당
	워커 노드	Kubernetes 클러스터 내에서 컨테이너가 실행되는 물리 또는 가상 머신. 노드는 클러스터의 구성원으로서 컨테이너를 실행하고 관리
	클러스터	여러 대의 컴퓨터 노드(서버)로 구성된 컨테이너 오케스트레이션 단위를 나타내는 개념
Kubernetes	파드	Kubernetes에서 가장 작은 배포 단위이며 하나 이상의 컨테이너로 구성. 동일한 파드 내의 컨테이너는 동일한 네트워크 네임스페이스와 IP 주소를 공유
	디플로이먼트	애플리케이션을 배포하기 위한 리소스로 레플리카셋을 생성하고 관리하는 역할을 수행. 롤링 업데이트 및 롤백과 같은 배포 전략을 제공
	레플리카셋	파드의 복제본을 유지하고 관리하는 Kubernetes 리소스. 지정된 수의 파드 인스턴스를 유지하고 부하 분산을 위해 사용
Kubernetes	매니페스트(manifest)	Kubernetes 클러스터에서 실행되는 리소스를 정의하는 YAML 또는 JSON 형식의 파일
	시크릿	민감한 데이터(예: 사용자 이름, 비밀번호, API 키 등)를 안전하게 저장하고 관리하는 데 사용되는 객체
	API 서버(엔드포인트)	Kubernetes 클러스터의 중심 컴포넌트이며, 사용자와 클러스터의 다른 부분 및 모든 외부 컴포넌트 간의 상호 작용을 가능하게 하는 역할
서비스	서비스	동일한 서비스를 제공하는 여러 파드를 논리적으로 그룹화한 것으로, 네트워크 트래픽의 로드 밸런싱 및 서비스 디스커버리 역할. 타입으로 노드 포트, 클러스터 IP, 로드 밸런서 등이 있음
	인그레스	클러스터 외부에서 내부로의 HTTP 및 HTTPS 트래픽을 관리하는 Kubernetes 리소스

Docker	Docker	컨테이너 기반 가상화 플랫폼
	Docker 레지스트리	컨테이너 이미지를 저장하고 공유하는 데 사용되는 저장소

2 컨테이너

2.1 정의

컨테이너는 가상화 기술을 이용하여 독립적으로 실행 가능한 애플리케이션을 만들 수 있는 기술 또는 기술의 집합이라고 정의할 수 있습니다. 가볍고 빠르게 실행 가능하며 실행에 필요한 파일, 라이브러리, 환경 설정 등을 이미지라는 형태로 패키징하고 컨테이너 런타임을 통해 가상화되어 실행됩니다.

컨테이너의 주요 특징은 아래와 같습니다.

- 애플리케이션을 격리하고 가상화합니다. 즉, OS 커널상의 자원은 공유하면서 각각의 컨테이너가 독립적으로 실행하는 방식입니다.

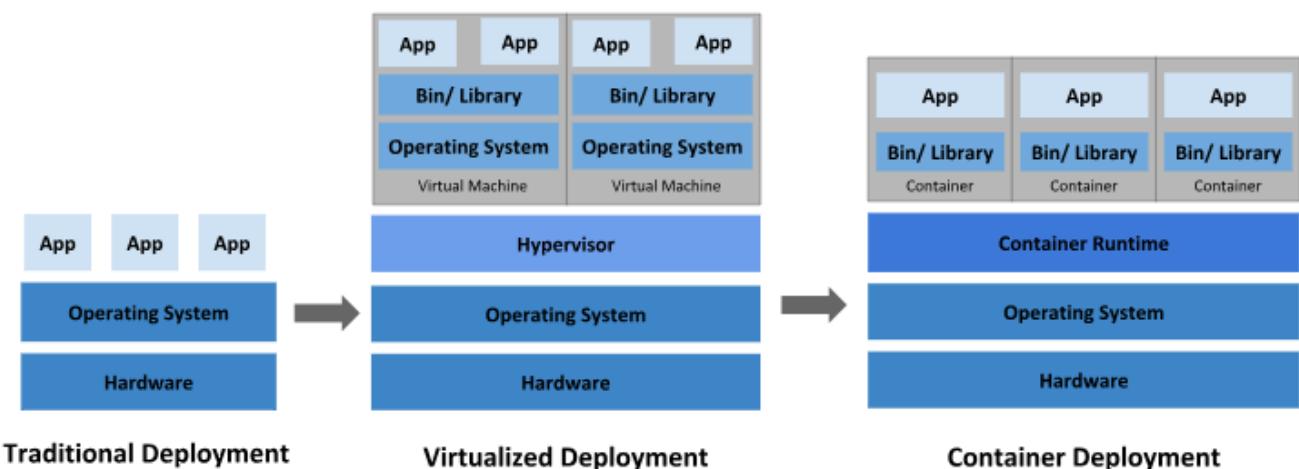


그림 1 가상화 개념

[출처: <https://kubernetes.io/ko/docs/concepts/overview/>]

- 컨테이너 런타임을 통해 OS 독립적으로 실행될 수 있습니다.
- 구동 모듈의 크기가 작고 고밀도화 구성이 가능하여 낮은 사양에서 활용도가 높습니다.
- 프로세스 실행 로직이 단축되어 빠른 구동, 종료가 가능합니다.
- 애플리케이션에 필요한 소프트웨어를 패키징함으로써 일관된 구동을 보장할 수 있습니다.
- 민첩하고 자동화된 배포가 가능하여 오토스케일링 등이 용이합니다.

2.2 컨테이너 주요 요소 개념

2.2.1 주요 요소 및 NHN Cloud 컨테이너 서비스

표 2-1 컨테이너 주요 요소 및 NHN Cloud 서비스

분류	주요 요소 및 기술	NHN Cloud 서비스
컨테이너 가상화 기술	네임스페이스	-
	cgroups	-
이미지	Docker 이미지	-
이미지 저장소	퍼블릭 Docker 허브	NHN Container Registry(NCR)
	프라이빗 레지스트리	NHN Container Registry(NCR)
컨테이너 런타임	Containerd	NHN Kubernetes Service(NKS)에 포함
	CRI-O	-
	RunC	-
	Docker 엔진	-
	Lxc(linux container)	-
오케스트레이션	Kubernetes	NHN Kubernetes Service(NKS)
	서비스 오케스트레이션 서비스	NHN Container Service(NCS)

2.2.2 컨테이너 가상화

가상화는 하나의 시스템을 여러 개의 가상 공간으로 분리 및 격리하여 최적화한 리소스를 효율적으로 사용할 수 있도록 하는 기술입니다. 분리 및 격리 기술을 활용하여 가상 시스템이 서로 독립적으로 실행할 수 있도록 하는 특징을 가졌으며, 하이퍼바이저를 이용한 서버 및 데스크톱 OS 가상화 방식과 애플리케이션을 가상화하여 OS에서 독립적으로 실행하는 방식이 있습니다.

컨테이너는 Linux에서 제공하는 네임스페이스와 cgroups 기술 등을 이용하여 만들어진 애플리케이션 가상화입니다. 네임스페이스는 컨테이너에서 실행된 프로세스가 시스템에서 독립적으로 실행될 수 있도록 프로세스 격리, 네트워크 인터페이스, 파일 시스템 포인트 관리 등을 담당하며, cgroups는 프로세스들이 사용할 수 있는 컴퓨팅 자원(CPU, 메모리, 네트워크, I/O 자원 등)을 제어할 수 있는 기술입니다.

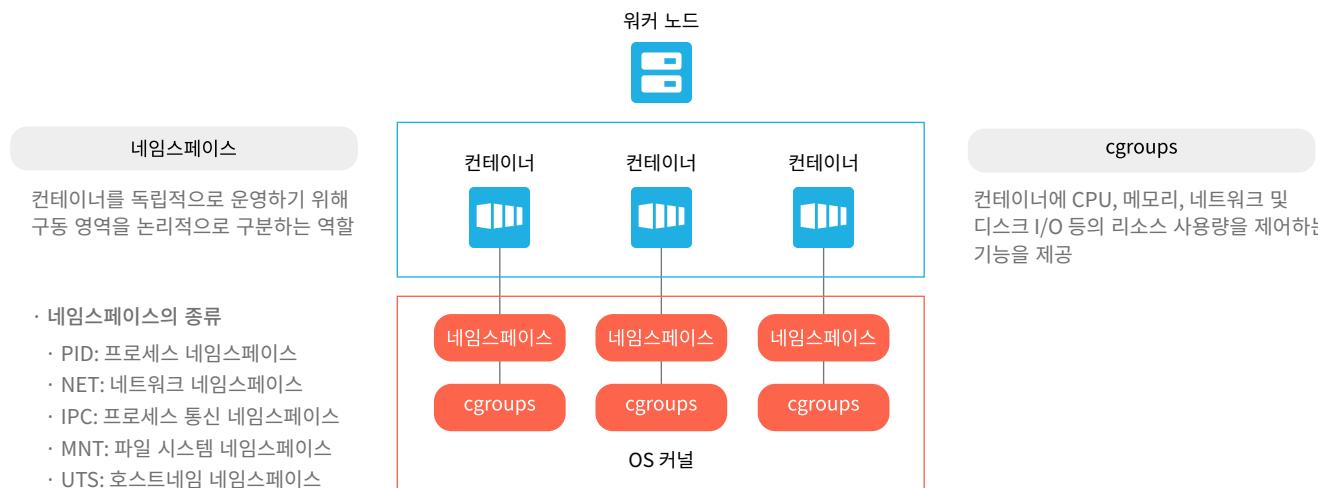


그림 2 컨테이너 가상화 기술

2.2.3 컨테이너 이미지와 이미지 레지스트리

컨테이너 실행을 위해 필요한 실행 파일, 라이브러리, 환경 설정 파일 등을 패키징한 것을 컨테이너 이미지라고 합니다. 컨테이너 이미지는 OCI(open container initiative)에서 이미지 포맷과 실행 환경 스펙 표준을 정의하고 있습니다. 여러 개의 레이어로 구성되어 있으며, 크게는 읽기 전용인 베이스 이미지 레이어와 쓰기 가능한 레이어로 나누어져 있습니다. 컨테이너를 생성, 구동할 수 있는 명령어나 파일 등이 레이어로 구성되어 있고, 컨테이너가 레이어를 공유할 수 있어 디스크 공간을 절약할 수 있으며 업데이트와 변경 작업 등에 유리합니다. 컨테이너 이미지는 Docker를 포함한 여러 도구와 플랫폼에서 생성하고 관리할 수 있습니다. 또한, Docker는 컨테이너 이미지를 저장할 수 있는 퍼블릭 저장소인 Docker Hub를 제공하고 있으며, Docker 레지스트리를 사용하여 프라이빗 저장소를 만들 수도 있습니다.

NHN Cloud는 컨테이너 이미지를 안전하게 저장, 관리하고 배포할 수 있는 NCR 서비스를 제공하고 있습니다.

NCR은 퍼블릭 및 프라이빗 연결을 지원하며 NHN Kubernetes Service(NKS)와 연동해 사용자의 애플리케이션을 손쉽게 컨테이너 환경으로 구축할 수 있도록 지원합니다.

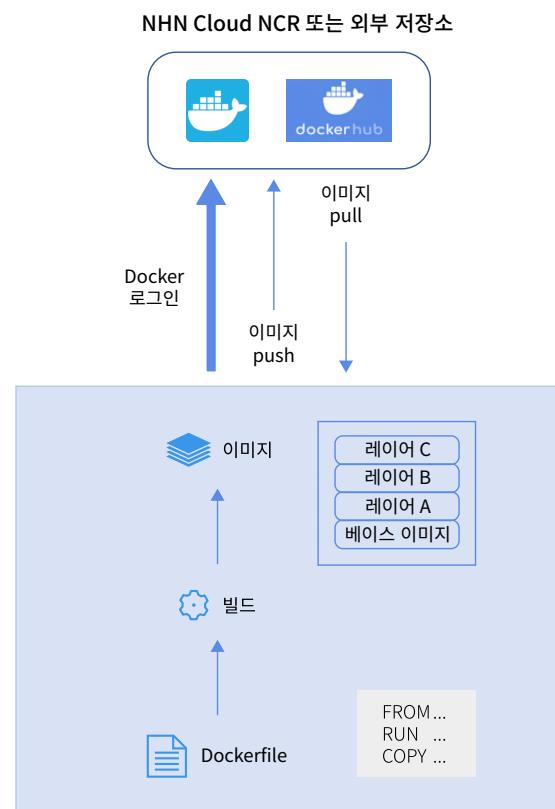


그림 3 컨테이너 이미지와 이미지 저장소

2.2.4 컨테이너 런타임

컨테이너 런타임은 컨테이너 이미지를 기반으로 컨테이너를 생성하고 실행하는 엔진입니다. 컨테이너의 시작과 종료를 담당하고, 컨테이너에 할당되는 컴퓨팅 자원 등을 관리합니다. OCI(open container initiative) 표준을 준수하여 만들어진 런타임은 컨테이너 실행만을 담당하는 저수준 런타임과 이미지 전송 및 관리 기능 등을 포함한 고수준 런타임으로 나뉩니다.

저수준 런타임인 runc는 컨테이너의 네임스페이스와 cgroups 관리 등의 기능을 처리합니다. 한편 고수준 런타임인 containerd나 CRI-O와 같은 프로그램은 이미지 관리, 배포 기능 등을 갖추고 있습니다.

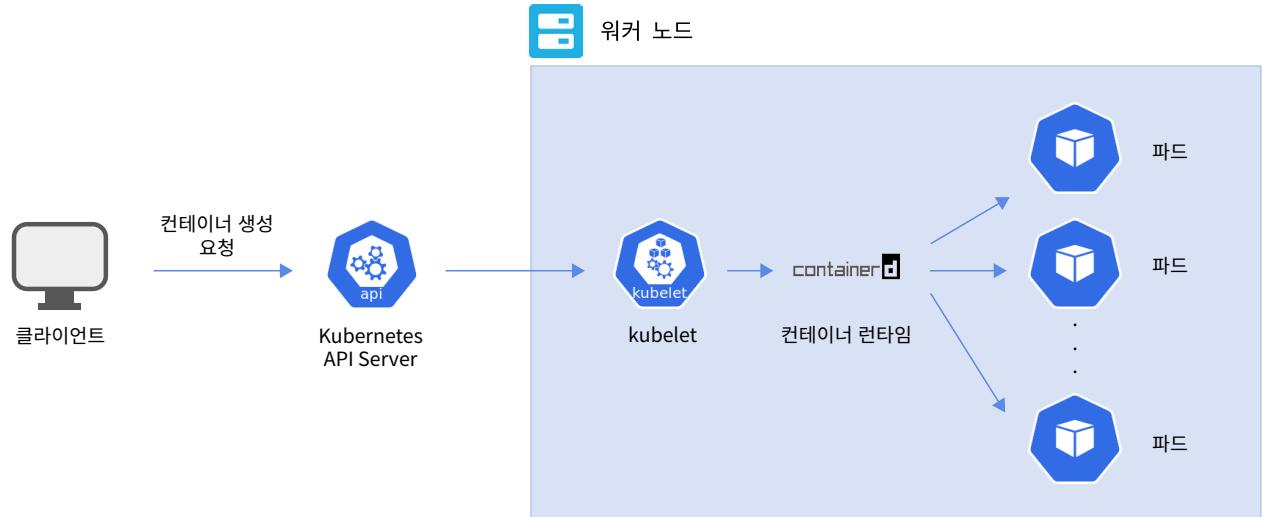


그림 4 컨테이너 런타임 containerd 동작 예시

2.2.5 컨테이너 오케스트레이션(Kubernetes)

컨테이너 오케스트레이션은 대규모 컨테이너들이 안정적으로 운영될 수 있도록 관리의 복잡성을 줄이고 이를 자동화하는 것입니다. 컨테이너 오케스트레이터는 컨테이너 자동 배치와 복제, 컨테이너 그룹에 대한 로드 밸런싱, 장애 복구 등을 할 수 있는 도구를 제공하여 컨테이너 운영의 현실적인 문제를 해결합니다. 대표적인 플랫폼으로 Kubernetes와 Docker Swarm 등이 있습니다.

NHN Cloud는 Kubernetes 기반의 NKS 서비스와 서비스 서비스로 컨테이너를 배포, 관리할 수 있는 NCS 서비스를 제공하고 있습니다.

Kubernetes 주요 구성 요소 이해

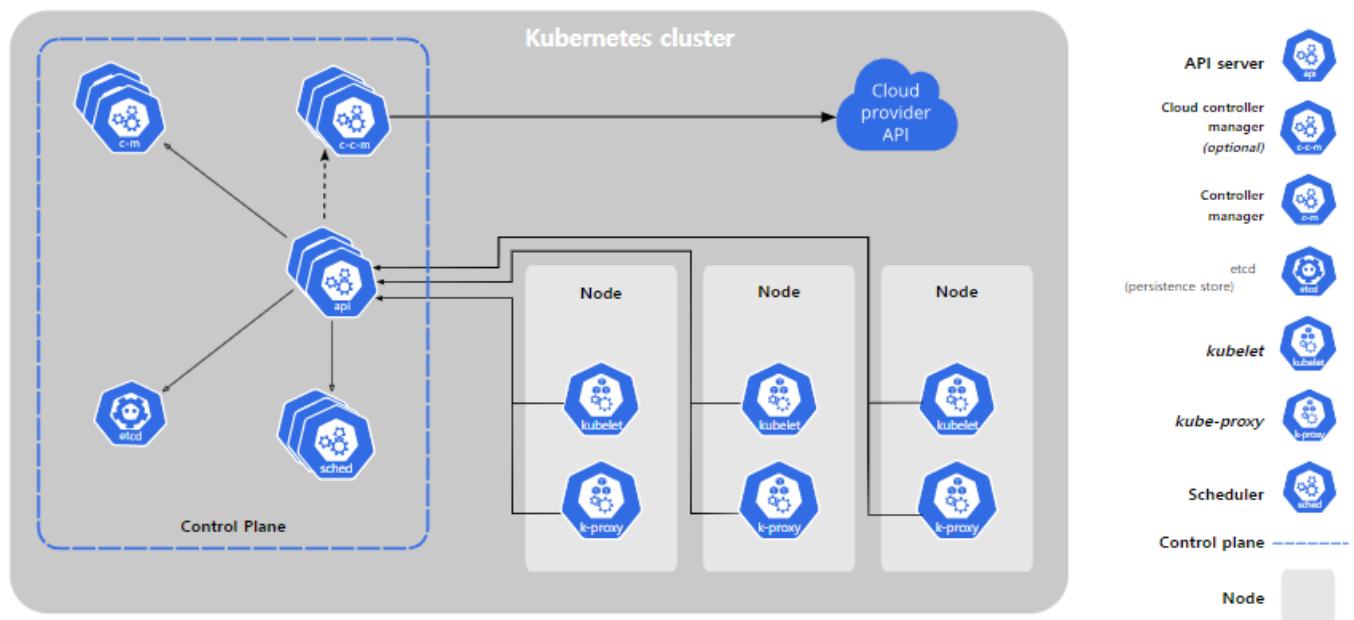


그림 5 Kubernetes 구조

[출처: <https://kubernetes.io/ko/docs/concepts/overview/components/>]

NKS는 NHN Kubernetes Service의 약자입니다. 컨트롤 플레인(control plane)은 사용자에게 인스턴스 형태로 노출되지 않아 보안에 유리하며 관리의 용이성을 제공합니다.

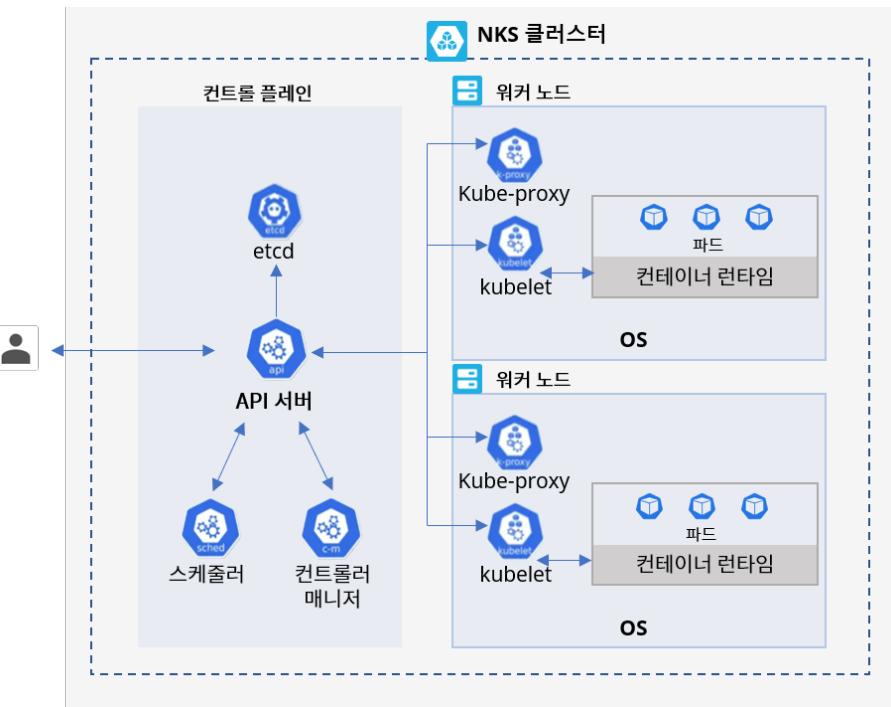


그림 6 NKS의 Kubernetes 구조

표 2-2 Kubernetes 핵심 요소

핵심 요소	설명
컨트롤 플레인	워커 노드와 클러스터 내 파드를 관리
워커 노드	Kubernetes는 컨테이너를 파드 내에 배치하고 노드에서 실행함으로써 워크로드를 구동
클러스터	컨테이너화된 애플리케이션을 실행하는 노드라고 하는 워커 머신의 집합. 모든 클러스터는 최소 한 개의 워커 노드가 존재
파드	Kubernetes에서 생성하고 관리할 수 있는 배포 가능한 가장 작은 컴퓨팅 단위
서비스	실행 중인 애플리케이션을 네트워크 서비스로 노출하는 추상화 방법. Kubernetes는 파드에게 고유한 IP 주소와 파드 집합에 대한 단일 DNS명을 부여하고, 그들 간에 로드 밸런싱을 수행
볼륨	컨테이너 내의 디스크에 있는 파일은 임시적이며, 컨테이너에서 실행될 때 애플리케이션에 적지 않은 몇 가지 문제가 발생하므로 데이터를 저장, 관리할 수 있도록 볼륨을 지원

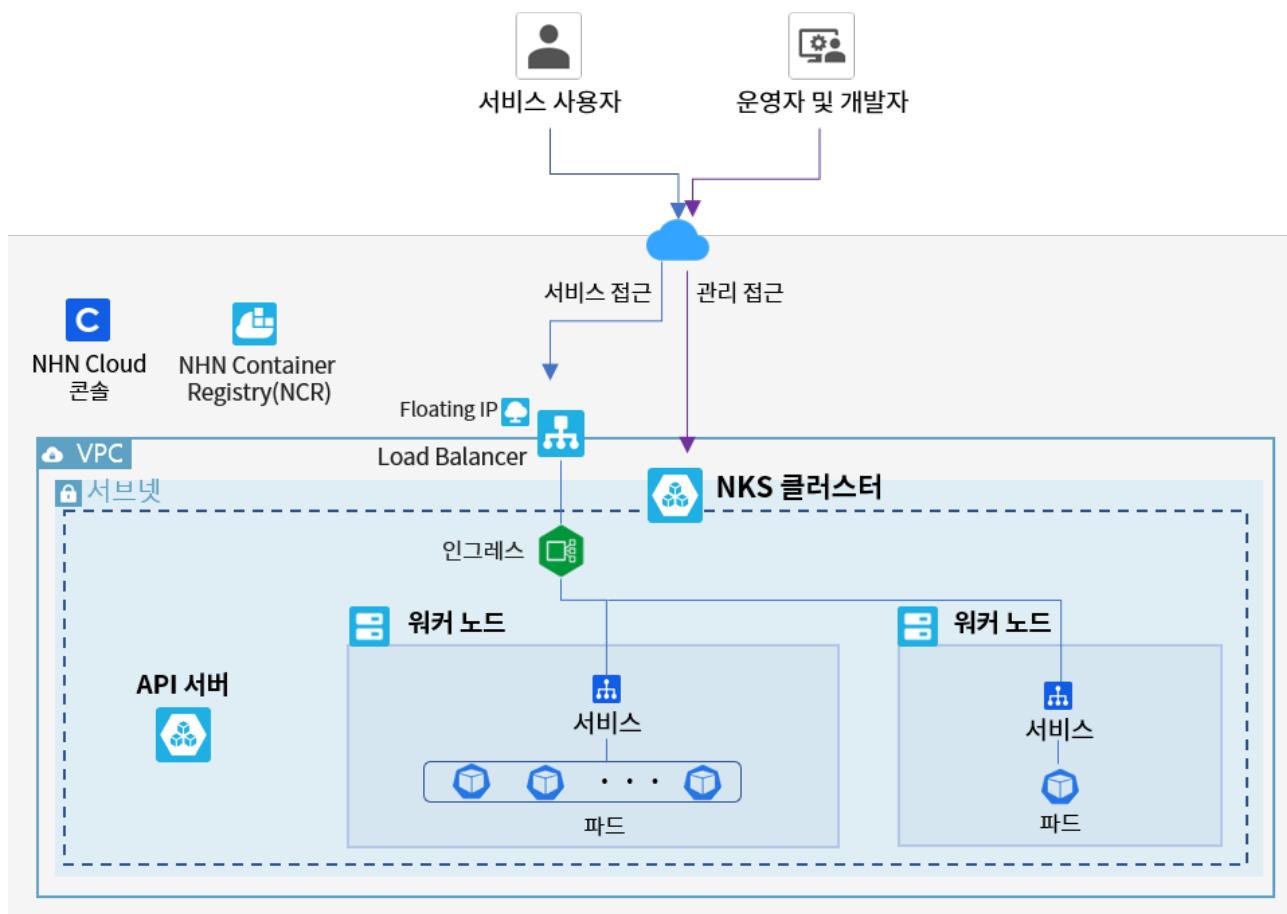
표 2-3 컨트롤 플레인의 주요 요소

주요 요소	역할
API 서버	Kubernetes의 모든 명령과 통신을 위한 API 서버
etcd	모든 클러스터 데이터를 담는 Kubernetes 운영 저장소, 분산형 키/값 오픈 소스로 Kubernetes 클러스터의 상태나 설정 정보를 저장
컨트롤러 매니저	컨트롤러를 생성하고 이를 각 노드에 배포하며 관리하는 역할 노드, 레플리케이션, 엔드포인트, 서비스 어카운트 및 토큰을 관리하는 컨트롤러로 구성
스케줄러	파드, 서비스 등 각 리소스들을 적절한 노드에 할당하는 역할

표 2-4 워커 노드의 주요 요소

주요 요소	역할
파드	Kubernetes의 최소 단위 객체, 클러스터에서 동작하는 컨테이너의 집합으로 단일 프라이머리 컨테이너를 구동하기 위해 셋업
컨테이너 런타임	Kubernetes 파드 내 구성된 컨테이너를 실행하는 기능 담당 Docker, containerd, CRI-O 등
kube-proxy	클러스터 내 각 노드에서 실행되는 네트워크 프락시
kubelet	클러스터 내 각 노드에서 실행되는 에이전트 파드 스펙 정보 관리/운용(생성, 삭제 등)

클라우드의 컨테이너 서비스를 활용한 서비스 구성 이해



클라우드 플랫폼

- 클라우드 콘솔 계정
- 네트워크 서비스
- 스토리지 서비스
- 인스턴스
- Kubernetes 서비스

노드 OS

- 워커 노드 OS

오케스트레이터

- Kubernetes(NKS)

이미지

- 이미지 저장소(NCR)
- 컨테이너 이미지

그림 7 NHN Cloud 컨테이너 서비스 구성

위의 그림과 같이 컨테이너 서비스를 생성하고 실행하기 위한 주요 인프라는 다음과 같습니다.

• 컨테이너 구성을 위한 인프라

- 클라우드 플랫폼: 컨테이너 생성과 실행, 서비스 접속을 위한 다양한 서비스를 제공합니다. 클라우드 콘솔 계정, IaaS 서비스, 컨테이너 관련 PaaS 서비스, 네트워크 서비스 등이 주요한 요소입니다. NHN Cloud의 클라우드 콘솔 계정, Instance, Load Balancer와 같은 서비스가 있습니다.
- 호스트/노드 OS: 컨테이너 생성, 실행 등을 위해 필요한 컴포넌트나 플러그인 등이 구성되는 노드 OS와 관리 접속 등을 위해 사용되는 배스천 서버 등이 해당됩니다.
- 오케스트레이터: 컨테이너를 배포, 관리하고 스케일링 등을 자동화하는 도구입니다. 컨테이너 런타임을 포함하여 배포 관리를 위한 컴포넌트들이 구성되어 있으며 다양한 애드온이 설치될 수 있습니다. NHN Cloud는 NKS와 서비스 방식의 NCS를 제공하고 있습니다.

- 이미지: 컨테이너 생성에 필요한 애플리케이션, 라이브러리, 설정 등이 포함되어 있는 이미지 자체와 이것을 보관할 수 있는 이미지 저장소(레지스트리)가 있습니다. NHN Cloud는 퍼블릭 또는 프라이빗으로 접근하여 이미지를 보관 및 관리할 수 있는 NCR 서비스를 제공하고 있습니다.

- **컨테이너 환경으로 실행되는 워크로드**

- 클러스터: 컨테이너화된 애플리케이션을 실행하는 머신의 집합입니다. 클러스터는 파드와 워커 노드를 관리하는 컨트롤 플레인과 파드가 실행되는 워커 노드를 포함합니다.
- 파드: 하나 이상의 컨테이너를 포함하는 Kubernetes의 가장 작은 배포 단위입니다. 스토리지 및 네트워크를 공유하고 컨테이너를 구동하는 방식을 파드 단위로 가지며 쉽게 생성되고 삭제될 수 있는 특징을 가집니다.
- 서비스: 컨테이너에서의 서비스는 클러스터 내의 특정 파드(pod)에 네트워크 접속을 제공하는 Kubernetes 리소스로, 클러스터 IP(clusterIP), 노드 포트(nodeport), 로드 밸런서(load balancer) 등의 유형이 있습니다. 이를 통해 클러스터 내부 또는 외부에서 네트워크를 통해 애플리케이션에 접근할 수 있습니다.
- 인그레스: 클러스터 내의 서비스를 외부에서 접근할 수 있도록 하는 것으로 주로 HTTP 프로토콜에 대해 부하 분석, SSL/TLS 구성 등을 지원합니다.

- **서비스 이용과 개발/관리**

- 서비스 사용자: 컨테이너화된 애플리케이션에서 제공하는 서비스를 직접 이용하는 불특정 다수 또는 특정 서비스 사용자입니다.
- 개발자: 애플리케이션을 개발하고, 이미지 빌드 및 컨테이너 배포 등을 담당합니다.
- 운영자: 인프라를 운영하고 관리합니다.

3 컨테이너 보안 위협

3.1 컨테이너 보안 위협 정의

컨테이너 보안은 소프트웨어 개발 환경에서 더욱 중요한 주제로 떠오르고 있습니다. 컨테이너 기술의 등장은 애플리케이션 개발 및 배포 프로세스를 빠르고 유연하게 대처할 수 있도록 변화시켰으며, 이로 인해 개발자들은 컨테이너 환경에서의 애플리케이션 개발을 지향하는 추세입니다. 이러한 소프트웨어 개발 환경 변화와 함께 컨테이너 보안의 중요성도 증가하고 있습니다.

컨테이너는 이식성과 확장성이 뛰어나며, 개발 주기를 단축시키고 배포를 효율화하는 등의 이점을 제공합니다. 그러나 이러한 편리함은 보안 관점에서 새로운 도전과 위험을 가져옵니다. 컨테이너 환경에서의 보안 위협은 애플리케이션의 배포 및 실행 과정에서 다양한 형태로 발생할 수 있습니다.

악의적인 공격은 컨테이너 이미지에 취약점을 심어서 시스템에 침투하거나, 컨테이너 간 통신을 스니핑하여 민감한 데이터를 탈취하는 등의 형태로 나타날 수 있습니다. 또한, 컨테이너의 권한 관리를 제대로 하지 않았을 경우 공격자가 스스로의 권한을 상승시켜 시스템 내부에서의 공격을 더 쉽게 수행할 수 있습니다. 더 나아가, 컨테이너 서비스의 거부(DoS) 공격으로 인해 서비스의 가용성이 감소할 수도 있습니다.

안전한 서비스 제공을 위해 컨테이너의 보안은 개발 초기부터 배포 및 운영/관리에 이르기까지 모든 단계에서 고려되어야 합니다. 이를 위해 보안 업데이트를 수시로 진행하고, 네트워크 보안 설정을 강화하며, 권한 관리를 철저히 실시하는 등의 조치가 필요합니다. 또한, 제로 트러스트 아키텍처를 구축하기 위해서도 컨테이너 보안은 중요한 요소입니다.

컨테이너 기술의 발전은 애플리케이션의 개발과 운영을 더욱 효율적으로 만들어 주었지만, 동시에 보안 측면에서의 책임과 주의가 더욱 중요해지고 있는 현실을 직시해야 합니다. 본 문서에서 제시하는 보안 가이드를 통해 안전하고 신뢰할 수 있는 서비스를 제공할 수 있을 것입니다.

3.2 컨테이너 보안 위협

3.2.1 노드 및 OS, 클라우드 자원의 위협

컨테이너는 호스트 OS 환경 위에서 실행됩니다. 수동으로 호스트를 구축하는 경우 물리적인 서버에 설치되는 OS 때문에 발생할 수 있는 위협으로 인해 컨테이너 서비스 또는 워크로드 등에 중대한 영향을 끼칠 수 있으므로 호스트 서버 관리와 OS에 대한 보안 관리가 필요합니다(NHN Cloud에서 제공하는 NCS와 같이 서버리스로 제공되는 컨테이너 서비스 제외).

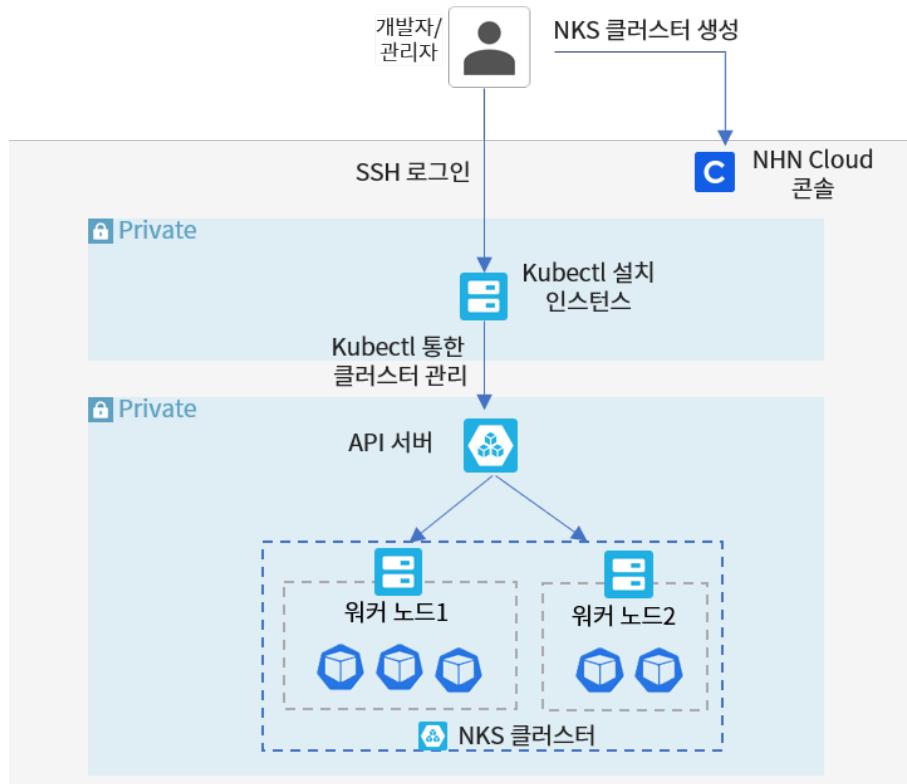


그림 8 NKS 구성 예시

주요 위협은 다음과 같습니다.

- OS 취약점으로 인한 위협
- OS 계정의 문제로 인한 위협
- OS 악성코드 감염
- 부적절한 OS 네트워크 접근 위협
- 클라우드 계정 관리 미흡

3.2.2 이미지 보안 위협

컨테이너는 이미지를 기반으로 생성됩니다. 이때 취약한 이미지를 생성하거나 다운로드한 이미지에 취약점이나 보안 위협이 존재할 경우, 컨테이너가 생성될 때 해당 위험이 이어질 수 있습니다. 따라서 이미지를 구축할 때 취약점을 최소화하고 보안 설정을 강화하는 것이 중요합니다. 이를 통해 컨테이너 환경에서의 보안 위협을 최소화할 수 있습니다.

주요 위협은 다음과 같습니다.

- 이미지 레지스트리의 부적절한 접근: 안전하지 않은 레지스트리 노출에 의한 레지스트리 공격, 파괴
- 불충분한 이미지 레지스트리 인증과 권한: 불법 로그인, 중요 이미지 유출
- 취약한 이미지 작성: 패스워드와 같은 기밀 정보의 평문 노출, 베이스 이미지의 취약한 버전, 루트 권한으로 실행
- 취약한 이미지 다운로드: 악성코드 포함, 취약한 버전의 라이브러리 등이 포함되어 있는 이미지

3.2.3 오케스트레이터 및 컨테이너 보안 위협

컨테이너 오케스트레이터는 컨테이너 애플리케이션을 관리하는 핵심 도구로서, 중요한 역할을 합니다. 그러나 악의적인 공격자에 의해 오케스트레이터의 취약점이 악용될 경우, 클러스터 전체가 위험에 노출될 수 있습니다. 오케스트레이터의 잘못된 구성은 보안 문제를 야기할 수 있으며 클러스터 리소스가 불법적인 접근에 노출될 경우 데이터 유출이나 랜섬웨어 감염과 같은 보안 사고가 발생할 수 있습니다.

주요 위협은 다음과 같습니다.

- 취약한 버전의 오케스트레이션 도구(런타임 등) 및 애드온 취약점
- 불충분한 네트워크 접근 통제
- 파드/컨테이너 네트워크 분리 미흡
- 파드 네트워크 보안 위협(DDoS, 웹 등의 네트워크 공격)
- 컨피그맵, 시크릿 관리 소홀
- 애플리케이션 취약점 및 실행 권한(특권 권한으로 실행) 위협

3.2.4 배포 및 운영 위협

컨테이너 기반의 소프트웨어 개발 및 운영 프로세스를 통합하여 개발과 배포를 더욱 빠르고 효율적으로 수행할 수 있지만 개발 및 배포 프로세스상에서 충분한 모니터링 되지 않고 취약한 이미지 등이 자동화되어 배포될 경우 개발, 배포 과정에서 취약점이 더욱 심각해질 수 있습니다.

또한 컨테이너가 실행되는 OS 관리 등을 위해 배스천 서버 또는 시스템 접근제어 서비스를 사용할 경우 배스천 서버, 시스템 접근제어 서버의 OS도 위협의 대상이 될 수 있으며 이러한 서버를 클라우드에서 인스턴스로 생성하여 사용할 경우 클라우드 보안 위협 영역도 관리가 필요합니다.

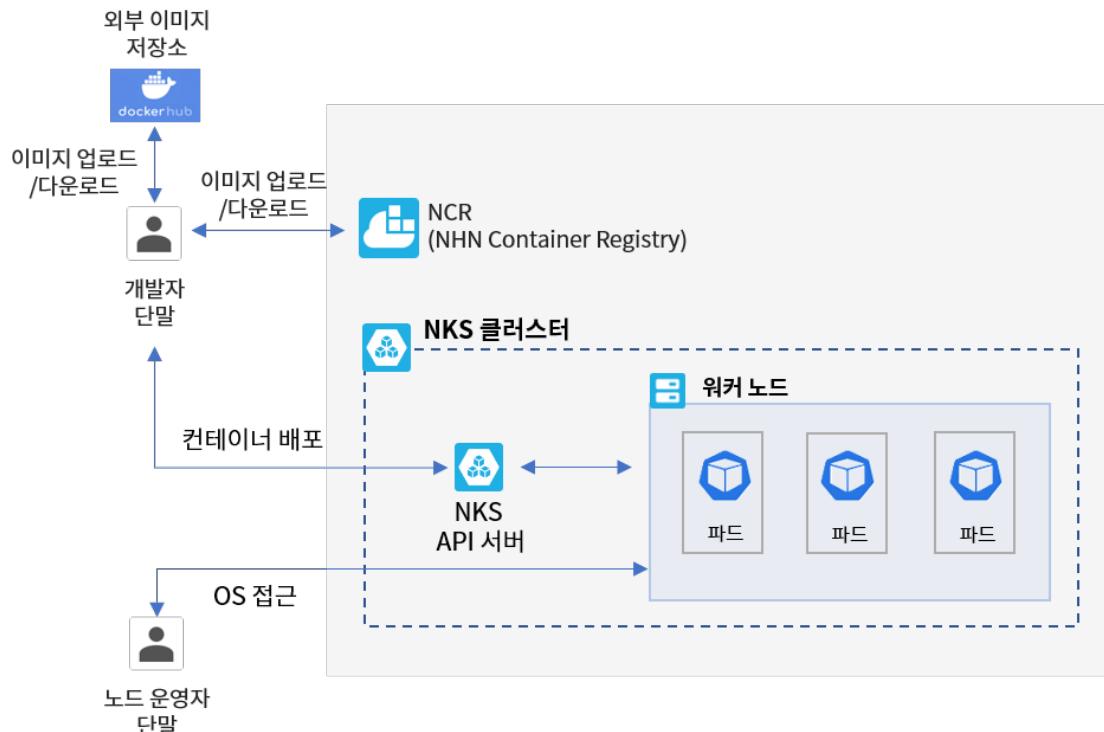


그림 9 배포와 노드 운영 관리

주요 위협은 다음과 같습니다.

- 승인되지 않은 자동화된 배포
- 분리되지 않은 개발과 운영의 컨테이너
- 개발 과정에서 테스트를 위해 생성한 컨테이너
- 미흡한 관리의 컨테이너 배포, 운영 계정

4 컨테이너 보안 적용 방안

4.1 컨테이너 보안 적용 방안 개요

컨테이너 환경에는 다양한 보안 위협이 존재합니다. 본 장에서는 이러한 위협에 효과적으로 대응하기 위한 주요 방안들을 소개하고자 합니다. 취약점 및 패치 관리, 접근제어 및 위협 대응, 컨테이너 이미지 및 레지스트리 보안, 오케스트레이터와 런타임 보안 등 NHN Cloud 컨테이너 서비스 사용 시 적용할 수 있는 전반의 보안 대책을 다룹니다.

4.2 노드와 OS 및 관리 콘솔 보안

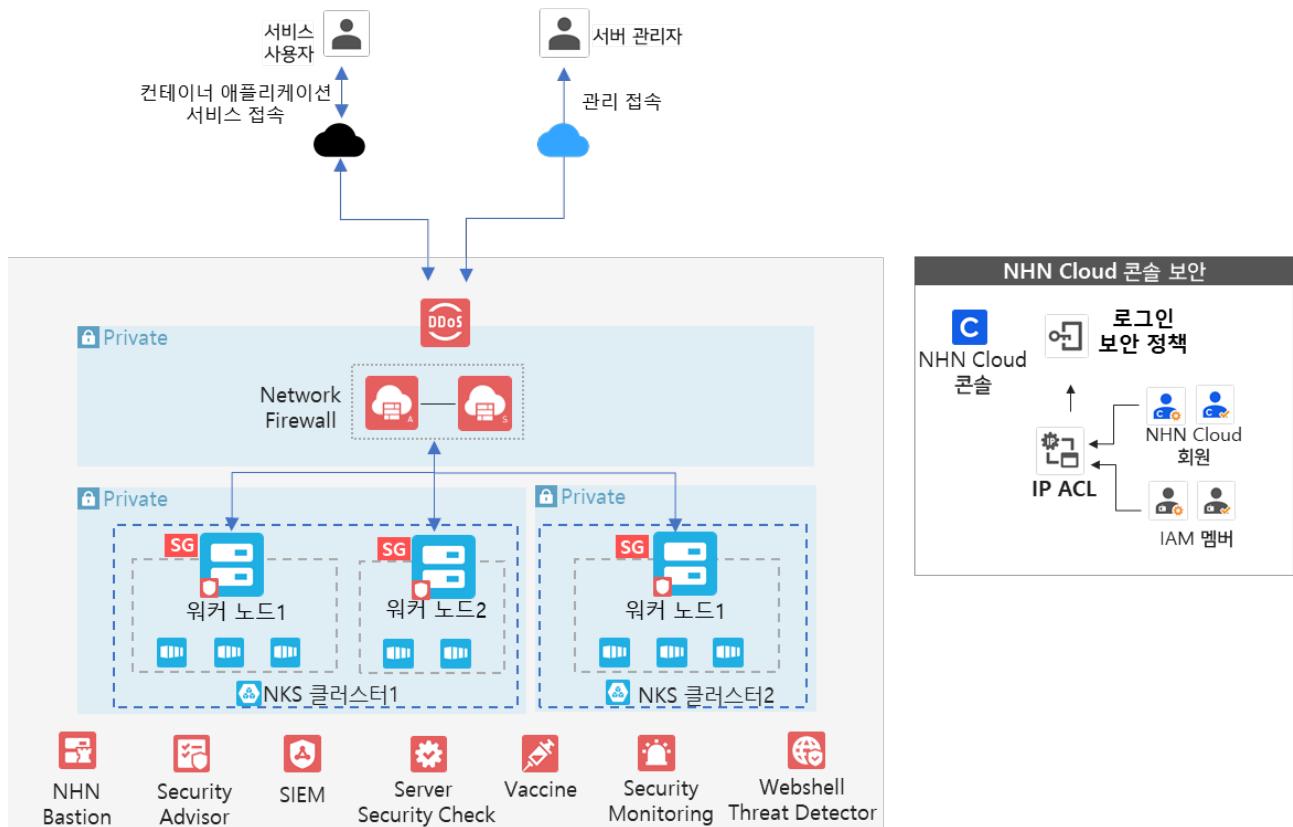


그림 10 노드 구조와 NHN Cloud 콘솔 예시

OS 패치 관리와 취약점 관리

컨테이너는 워커 노드에서 실행되므로 워커 노드의 OS가 취약하지 않도록 관리하는 것이 중요합니다. 패치는 보안 취약점을 해결하고 시스템을 보호하는 데 중요한 역할을 합니다. 정기적으로 패치를 확인하고 필요한 패치를 수동으로 다운로드하여 설치하거나 패치 관리 시스템 등을 통해 패치 관리를 할 수 있습니다.

또한 OS의 정기적인 취약점 진단을 통해 OS의 취약한 설정 등을 정기적으로 검사할 필요가 있습니다. NHN Cloud의 Server Security Check 서비스를 이용하면 시스템의 주요 보안 설정값을 점검하고 조치해 시스템 안전성을 확보하고 잠재적 취약점을 제거할 수 있습니다.

추가로 컨테이너 런타임과 컨테이너 실행 상태, 배포, 운영 등의 보안 강화를 위해 NIST(미국 국립표준기술연구소) SP 800-190 또는 KISA(한국인터넷진흥원) 클라우드 취약점 점검 가이드 등을 활용하여 주기적인 컴플라이언스 검토와 체크리스트 기반 취약점 점검을 수행할 수 있습니다.

※ [NIST 800-190 Application Container Security Guide](#)

※ [KISA 클라우드 취약점 점검 가이드](#)

악성코드 관리

악성코드로 인해 발생할 수 있는 보안 사고를 방지하기 위해 백신과 같은 솔루션을 이용하여 실시간으로 악성코드를 검사하고 주기적인 수동 검사로 서버 OS를 보호할 필요가 있습니다.

OS 접근제어 및 계정 관리

컨테이너가 실행되는 워커 노드의 OS 계정이나 OS 접근은 운영자/관리자에게만 접근을 허용하고 계정을 할당해야 합니다. 개발자, 운영자/관리자, 보안 관리자 등의 업무에 따라 계정을 개별 할당하고 책임과 역할에 따라 권한을 부여합니다. 특히 외부에 노출되어 있는 공인 IP 등을 가지거나 컨테이너를 통해 외부 서비스가 실행되고 있는 서버인 경우 보다 안전한 인증을 위해 2차 인증을 적용하는 등의 보안 조치가 필요합니다.

노드 네트워크 접근제어와 위협 관리

노드의 네트워크 접근제어와 위협 관리 방법은 여러 가지가 있습니다. NHN Cloud의 NKS를 활성화하면 워커 노드는 인스턴스로 실행되므로 NHN Cloud에서 제공하는 Security Groups나 Network ACL과 같은 기본 서비스를 이용하여 IP, 포트 접근통제를 하고 DDoS Guard 서비스 등을 이용하여 네트워크 공격을 차단할 수 있습니다. 또는 NHN Cloud의 보안 서비스인 Network Firewall 구성을 통해 워커 노드에 대한 접근통제와 네트워크 접근 기록을 관리할 수도 있습니다.

다양하고 상세한 네트워크 접근제어와 위협 관리에 대한 방법을 [NHN Cloud 보안 센터 「네트워크 아키텍처 보안 가이드」](#)에서 확인할 수 있습니다.

NHN Cloud 콘솔 보안

NHN Cloud 콘솔은 NKS, NCR과 같은 서비스 이용을 위해 필수적으로 사용하는 인터페이스이며 콘솔을 통해 여러 가상 자원의 생성, 삭제 등이 이루어집니다.

따라서 NHN Cloud 콘솔의 계정 관리는 무엇보다 중요하며 반드시 계정의 2차 인증 적용을 권고합니다. 또한 관리자의 역할과 권한을 IAM을 통해 분리하여 관리하고 Security Advisor와 같은 무료 보안 서비스를 통해 수시로 계정의 보안 거버넌스 설정을 점검하는 것을 권고합니다.

NHN Cloud 콘솔 보안에 대한 상세한 내용은 [NHN Cloud 보안 센터 「콘솔 보안 가이드」](#)에서 확인하실 수 있습니다.

4.3 이미지와 레지스트리 보안

이미지 레지스트리 외부 노출 제한

NHN Cloud NCR은 Public URI와 Private URI를 제공합니다. Private URI는 인터넷 연결이 되지 않는 사설 호스트에서 연결이 가능한 URI로 NHN Cloud Service Gateway를 통해 접근 가능합니다.

폐쇄망에 있는 개발 서버 등에서 NCR에 이미지를 사설망으로 가져오기(pull) 또는 저장하기(push) 할 수 있습니다.

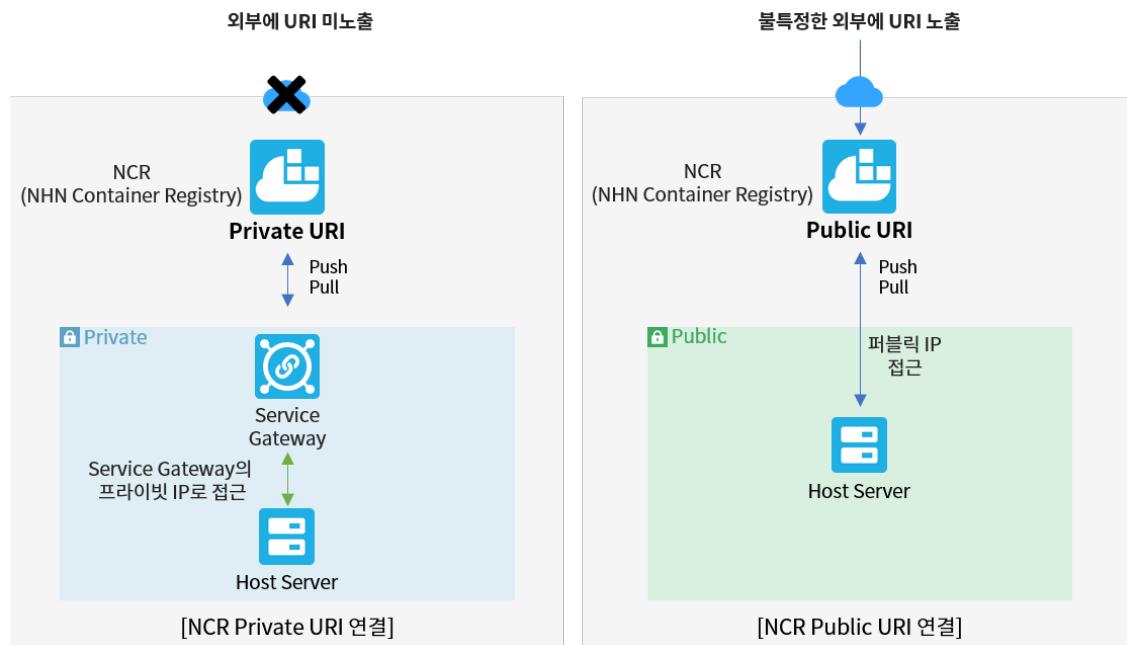


그림 11 NCR Private과 Public URI 연결

NKS와 연동할 경우 Private URI를 통해 워커 노드에 컨테이너 배포가 가능합니다.

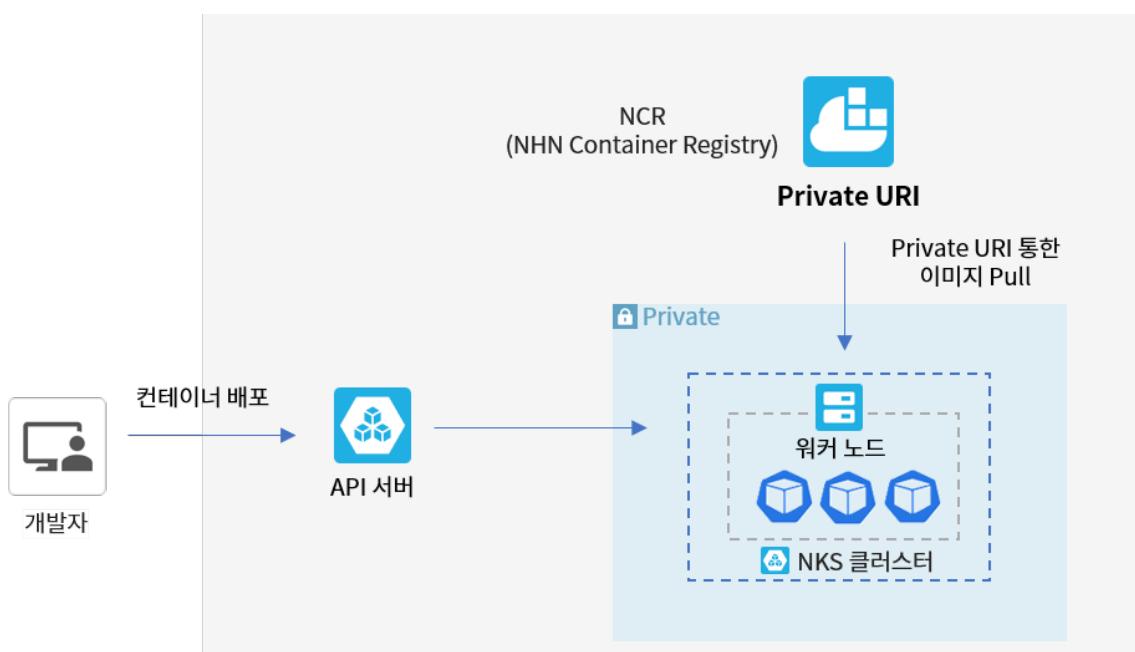


그림 12 NCR의 Private URI를 통한 컨테이너 배포

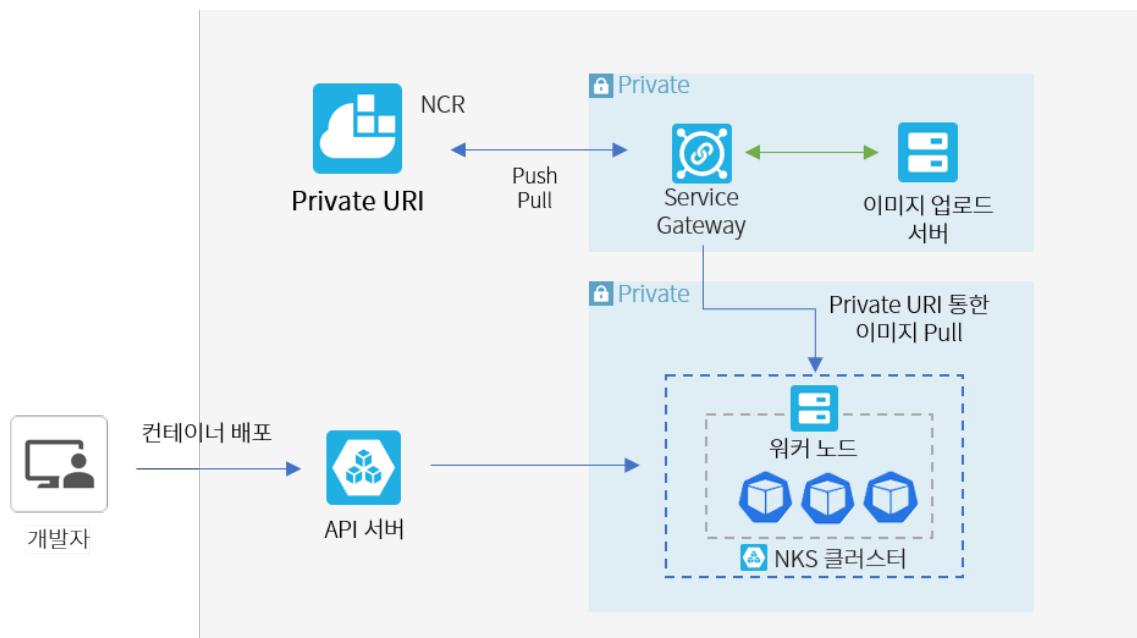


그림 13 NCR의 Private URI과 NKS 구성

이미지 레지스트리의 인증과 권한 관리

NCR은 Docker 명령어를 통해 이미지를 push하거나 pull 할 수 있도록 지원합니다. NKS(NHN Kubernetes Service)와 연동하거나 이미지 업로드 서버 등에서 Docker 명령어로 이미지를 push, pull 하기 위해서는 NCR 인증이 필요합니다.

개발자, 운영자가 Docker에 로그인하기 위해 사용하는 서버 또는 단말의 OS에 먼저 Docker를 설치해야 합니다. Docker 설치 방법은 [NHN Cloud 사용자 가이드](#)를 참고하세요.

설치된 Docker를 통해 NCR에 로그인하기 위해서는 User Access Key와 Secret Access Key가 필요합니다.

User Access Key ID	Secret Access Key	상태	생성 일시	마지막 사용 일시	제생성 일시
1D: h***** 재생성	V***** 재생성	STABLE	2024-03-05 12:15:07 (UTC+09)	2024-04-16 15:43:20 (UTC+09)	2024-03-21 17:33:13 (UTC+09)

그림 14 NCR 로그인 키

User Access Key와 Secret Access Key는 NHN Cloud 회원 계정 > API 보안 설정에서 확인할 수 있으며 콘솔 계정마다 부여됩니다. NCR 로그인에 필요한 인증 정보이므로 반드시 안전한 장소에 보관이 필요하며 가급적 공유를 금지하고 90일마다 변경하는 것을 권장합니다.



그림 15 NCR 로그인

NHN Cloud 콘솔은 NCR 역할을 계정별로 부여할 수 있습니다. 필요한 사용자에게만 역할을 부여하여 컨테이너 이미지에 대한 불필요한 접근을 막고 필요한 작업만 수행할 수 있도록 합니다.

IAM 멤버 등록

ID *

developer 9/20

역할 *

Q NCR

검색

NHN Container Registry(NCR)

NHN Container Registry(NCR) ADMIN
NHN Container Registry(NCR) 서비스 Create(생성), Read(읽기), Update(갱신), Delete(삭제)

NHN Container Registry(NCR) Image Uploader
NHN Container Registry(NCR) 서비스 Read(읽기), 이미지 Create(생성)

NHN Container Registry(NCR) PERMISSION
NHN Container Registry(NCR) 서비스 Enable(활성화), Disable(비활성화)

NHN Container Registry(NCR) VIEWER
NHN Container Registry(NCR) 서비스 Read(읽기)

NHN Container Registry(NCR) Image Uploader x

취소 등록

그림 16 NHN Cloud 콘솔 계정 NCR 역할

이미지 위협 관리

NCR은 이미지 취약점 스캔 기능을 제공합니다. 저장되어 있는 이미지를 스캔할 수 있으며 Push 할 때 자동으로 취약점 스캔을 할 수 있도록 설정할 수 있습니다. 그리고 이미지 취약점 점검 결과 취약점이 발견된 이미지는 취약점 레벨에 따라 Pull을 방지하도록 설정할 수 있습니다.

레지스트리 > nhn-ncr > nhn-ncr/nhn-nginx							
아티팩트 삭제		스캔	스캔 중지	총 2개			
	아티팩트 유형	아티팩트	크기 :	인증	Push 시간 :	Pull 시간 :	취약점
<input checked="" type="checkbox"/>		sha256:a04ebcee	67.26MB		2024-03-22T14:42:04+09:00	2024-04-01T13:48:49+09:00	Critical (total 151 / fix 0)
<input type="checkbox"/>		sha256:fddbaab6	67.26MB		2024-03-22T14:32:33+09:00	2024-03-22T18:29:33+09:00	-

그림 17 NCR 이미지 스캔 기능

The screenshot shows the 'Image Scan Settings' dialog box in the NCR interface. It includes fields for 'Push 시 자동 스캔' (Automated scan on push), 'Pull 방지' (Pull prevention), 'CVE 허용' (CVE Allow), and a '취소' (Cancel) and '확인' (Confirm) button.

레지스트리 이름	상태
nhn-ncr	일반
yeji	일반

이미지 스캔 설정

Push 시 자동 스캔: 설정 (Radio button selected)

Pull 방지: Critical

CVE 허용: 공동 레지스트리 허용 설정 (Radio button selected)

허용 목록: CVE ID

* 설정값은 개별 레지스트리에 적용됩니다.

그림 18 NCR 이미지 스캔 설정

외부 이미지 레지스트리 접근 제한 및 다운로드 시 취약점 검증 절차를 적용하여 위협 요인을 제거할 수 있습니다. 외부 이미지 레지스트리 접근은 최소화하고 필요시 다운로드하되, 다운로드한 이미지는 취약점 점검이 필수로 진행될 수 있도록 프로세스화하는 것도 중요합니다. 관리적 절차, 단계 등을 통해 취약점 점검이 되지 않은 이미지는 적용될 수 없도록 합니다.

취약점 점검을 위한 레지스트리와 서비스 운영을 위해 사용될 이미지가 저장되는 레지스트리를 분리하여 취약점이 있는 이미지가 실 운영 서비스에 적용되지 않도록 운영할 수 있습니다.

The screenshot shows the 'Registry Separation' section of the NCR interface. It lists two registries: 'dev-registry' (selected) and 'prod-registry'. The 'dev-registry' row has 'Public URI' and 'Private URI' columns. The 'prod-registry' row has '사용 안 함' (Not used) in both columns.

레지스트리 이름	Public URI	Private URI
dev-registry	사용 안 함	private-e92f505...
prod-registry	사용 안 함	private-e92f505...

그림 19 레지스트리 분리 운영

NCR에서 제공하는 취약점 스캔 기능을 이용할 경우 아래 그림과 같은 절차, 방법으로 이미지 취약점을 점검할 수 있습니다.



그림 20 NCR 취약점 스캔 기능을 이용한 외부 이미지 취약점 관리 절차 예시

아래는 취약점 검사를 오픈 소스 프로그램 또는 상용 보안 솔루션 등을 이용하여 연동하는 방법의 예시입니다. 그림의 취약점 점검 플랫폼(개발 예시)은 서비스 개발 환경과 내부 승인 절차 등을 고려하여 설계 후 개발이 필요한 부분입니다. NCR 웹 헤더 사용 방법은 [NHN Cloud 사용자 가이드](#)에서 확인 가능합니다. 아래 제시된 그림은 예시이며, 각 환경에 적합한 취약점 검사 모듈과 관리 프로세스를 선택해 적용하세요.

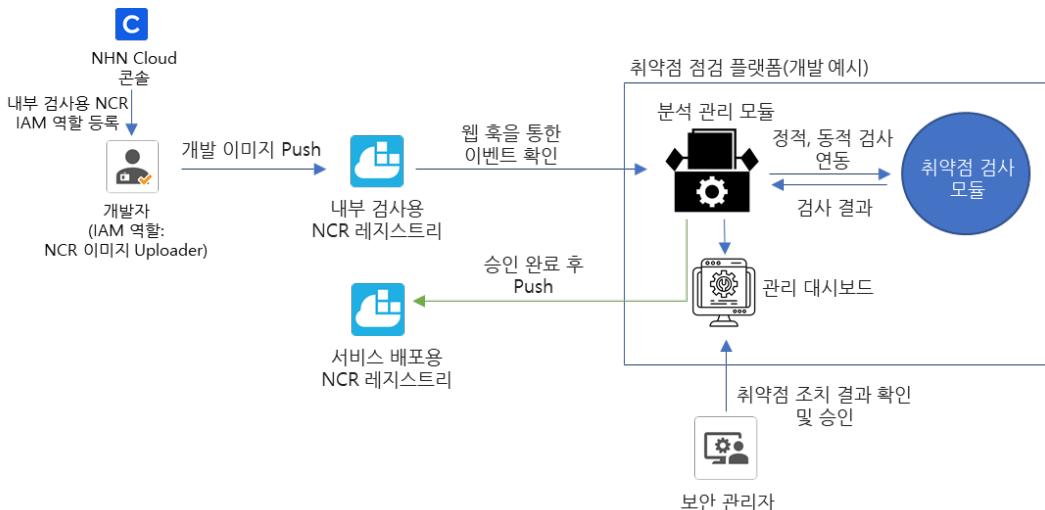


그림 21 외부 취약점 검사 모듈 연동 방식 예시

Dockerfile은 아래 그림과 같이 컨테이너 이미지를 생성(빌드)할 때 필요한 명령어, 설정, 패키지 등이 포함된 파일이므로 불필요한 패키지가 포함되거나 계정의 시크릿(secret) 정보가 포함되지 않도록 하는 것이 중요합니다. 루트(root) 계정 등을 사용하도록 컨테이너 이미지가 생성될 경우 컨테이너 서비스 실행 시 권한 악용 등으로 인한 서비스 취약점 발생 가능성이 높아지므로 가급적 루트 계정으로 실행되도록 하는 명령어를 포함하지 않도록 하고 사용자를 지정하는 것이 좋습니다.

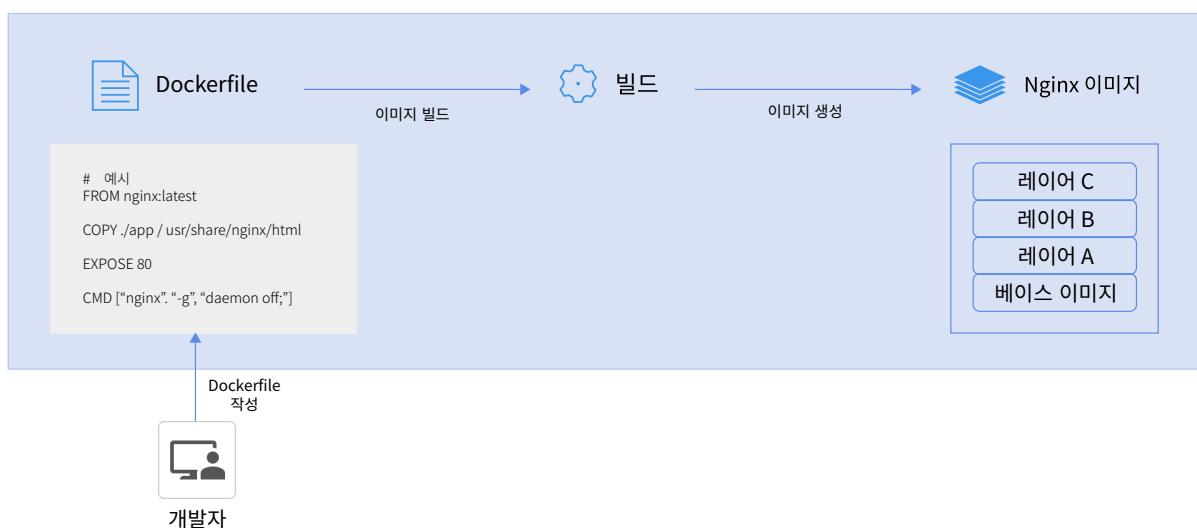


그림 22 Dockerfile을 작성하여 이미지를 생성한 예시

컨테이너 이미지에 서명을 통해 컨테이너 이미지의 신뢰성 관리, 무결성, 출처 확인을 할 수 있습니다. NCR에서는 sigstore/cosign 솔루션으로 서명 기능을 제공하며 서명 여부를 클라우드 콘솔에서 확인할 수 있습니다. 서명 기능을 사용하기 위해서는 sigstore/cosign 솔루션을 설치해야 하며 자세한 방법은 [NHN Cloud 사용자 가이드](#)에서 확인할 수 있습니다.

서명이 검증된 이미지의 경우 NHN Cloud 콘솔 NCR의 아티팩트 목록의 인증 열에서 아티팩트들의 서명 여부를 확인할 수 있습니다.

NHN Container Registry(NCR) > 관리

관리 복제 이미지 캐시 URL

레지스트리 > nhn-ncr > nhn-ncr/mariadb

아티팩트 삭제	스캔	스캔 중지					
<input type="checkbox"/>	아티팩트 유형	아티팩트	크기	인증	Push 시간	Pull 시간	취약점
<input type="checkbox"/>		sha256:87d4a027	117.92MB	X	2024-04-12T14:43:38+09:00	-	-

그림 23 이미지 인증서 확인 예시

이미지 백업 및 삭제 보호

컨테이너 이미지의 정기적인 백업을 통해 장애 복구 또는 랜섬웨어 감염 등에 대비할 수 있습니다. 컨테이너 이미지를 .tar 파일로 저장하여 별도의 저장 공간에 보관할 수도 있으며 NCR에서 제공하는 복제 기능을 이용하여 다른 리전의 레지스트리에 이미지를 보관할 수도 있습니다. NCR의 복제 기능은 자동 복제되도록 설정 가능하므로 조건을 설정하여 자동으로 복제(백업)되도록 할 수 있습니다.

복제 생성

복제 이름: backup

설명:

복제 유형: Push

중복 이미지: 건너뛰기

대상 리전: 한국(평촌)

대상 레지스트리: backupregistry

소스 이미지 이름 필터: 이미지 이름 또는 빙칸(전체)

소스 태그 필터: 태그 이름이 태그 또는 빙칸(전체) | 에 해당하는 아티팩트 이미지 복제

자동 실행:

- 복제가 완료되면 대상 리전의 레지스
- 소스 리전의 이미지 또는 아티팩트가

사용 안 함

이벤트 기반

사용자 설정

취소 확인

그림 24 타 리전으로의 이미지 복제 기능

사용자의 실수 등으로 인해 운영 서비스에서 사용 중인 이미지가 임의로 삭제되는 것을 방지할 수 있도록 NCR은 이미지 보호 기능을 제공하고 있습니다. 이미지가 임의로 삭제되어 유실되는 것을 차단할 수 있습니다.

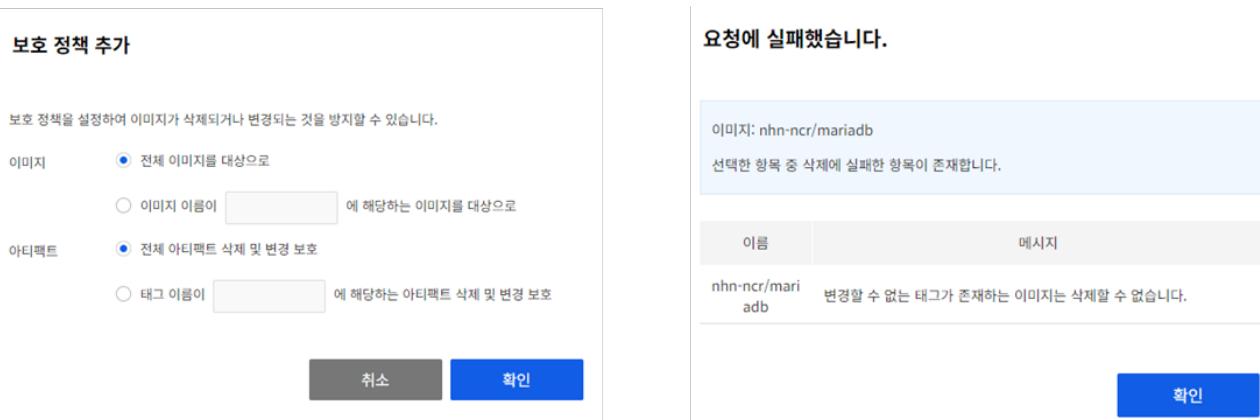


그림 25 이미지 삭제 보호 정책

Docker 등을 통해 만들어진 이미지는 NCR과 같은 레지스트리에 별도 보관하여 관리할 수 있지만 .tar 파일로 변환하여 보관 할 수도 있습니다. Docker가 설치되어 있는 단말 등에서 “# docker save” 명령어를 이용하여 .tar 파일로 변환한 후 별도의 스토리지에 이동 및 보관/관리할 수 있습니다.

오픈 소스 기반 이미지 레지스트리 관리

오픈 소스로 이미지 레지스트리 서버를 직접 구축하여 운영할 경우에도 중요 이미지가 저장되는 서버는 외부에 직접 노출하지 않도록 하고 필요한 경우 방화벽과 같은 네트워크 접근통제 수단을 통해 접근제어를 하는 것을 권장합니다. Harbor와 같은 오픈 소스를 이용할 경우 자체 계정과 권한 관리 기능을 제공하므로 환경에 맞게 계정과 권한을 적절히 관리하는 것이 필요합니다. 또한 오픈 소스를 이용하여 이미지 레지스트리 서버를 구축한 경우에도 패키지의 취약점 관리나 장애 관리 등 자체적인 관리 절차를 통해 레지스트리 인프라를 주의 깊게 관리할 필요가 있습니다.

4.4 오케스트레이터 및 컨테이너 보안

오케스트레이션 도구(Kubernetes 등)의 취약점 관리

Kubernetes와 같은 컨테이너 오케스트레이션 플랫폼은 다양한 런타임 및 플러그인으로 구성되어 있습니다. 이러한 구성 요소들에는 보안 취약점이 지속적으로 발견되고 있어 주의가 필요합니다. 예를 들어 최근 runc 런타임에서는 호스트 파일 시스템에 대한 액세스 권한을 획득할 수 있는 취약점이 발견되었습니다. 이처럼 오케스트레이션 플랫폼의 구성 요소들은 버전에 따라 새로운 취약점에 노출될 수 있습니다. 따라서 보안 담당자는 이러한 취약점을 지속적으로 모니터링하고, 적시에 패치를 적용하는 체계를 갖추어야 합니다.

NHN Kubernetes Service(NKS)는 클라우드 콘솔에서 클러스터 버전 업그레이드 기능을 지원하고 있어 손쉽게 버전을 확인하고 업그레이드를 진행할 수 있습니다.

클러스터 업그레이드 (마스터 구성 요소)

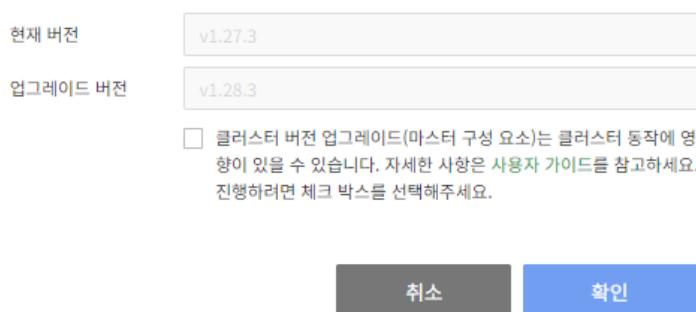


그림 26 클러스터 업그레이드

NKS의 생성 가능한 버전 정책은 4개로 유지되고 서비스 지원이 종료된 버전을 사용하는 클러스터는 NKS의 신규 기능 동작을 보장하지 않습니다. 따라서 사용자는 주기적으로 버전을 확인하고 관리해야 합니다. 자세한 Kubernetes 버전 지원 정책은 [NHN Cloud 사용자 가이드](#)에서 확인할 수 있습니다.

Kubernetes와 관련된 CVE 목록은 [공식 CVE 피드](#)에서 확인할 수 있으며 [한국인터넷진흥원 취약점 정보 공유 사이트](#) 등에서 취약점 공지 사항을 주기적으로 확인하고 필요시 패치를 적용하여 취약점으로 인한 위협에 선제적으로 대응해야 합니다.

워크로드 특징별 파드 분리 구성

컨테이너로 배포될 워크로드의 서비스 성격과 데이터 저장 유무, 외부 접근성 등을 고려하여 컨테이너가 배치되는 VPC, 서브넷, 워커 노드를 분리하여 설계합니다.

예를 들어 외부 서비스의 클러스터와 내부 업무용 클러스터를 분리하여 구성하면 불필요한 접점이 생기지 않고 클러스터 장애나 보안 사고 시 서비스 영향도를 최소화할 수 있습니다.

동일 클러스터 내에서도 웹 서버 컨테이너가 배치되는 워커 노드와 WAS 컨테이너가 배치되는 워커 노드를 분리하여 서로 간의 서비스 네트워크 접근통제뿐만 아니라 외부 위협으로부터 계층적으로 대응할 수 있도록 설계하는 것이 효과적인 위협 대응 방법일 수 있습니다.

네트워크 접근통제 및 위협 모니터링은 네트워크 구성, 분리가 명확하지 않을 경우 대표 IP로만 모니터링 가능하거나 네트워크 접근통제가 되지 않을 수도 있으므로 VPC, 서브넷, 서비스 IP, 파드 IP 등을 고려하여 설계하는 것이 필요합니다. 그리고 오토스케일 등으로 인해 새로운 컨테이너나 노드가 생성될 때마다 일관적으로 보안 정책이 적용될 수 있도록 사전에 충분한 검토 후 워크로드 배치를 설계하는 것이 필요합니다.

- 워크로드 설계 주요 고려 사항

- 서비스 중요도 및 불특정 다수의 외부 접점 여부를 고려한 노드(컨테이너) 식별
- 중요 데이터 저장 노드(컨테이너) 서브넷 등의 분리 구성
- 네트워크 접근통제 및 모니터링 구간 고려한 네트워크 구성
- 오토스케일 고려한 보안 정책 적용 방안(IP 등 고려하여 보안 정책 수립)

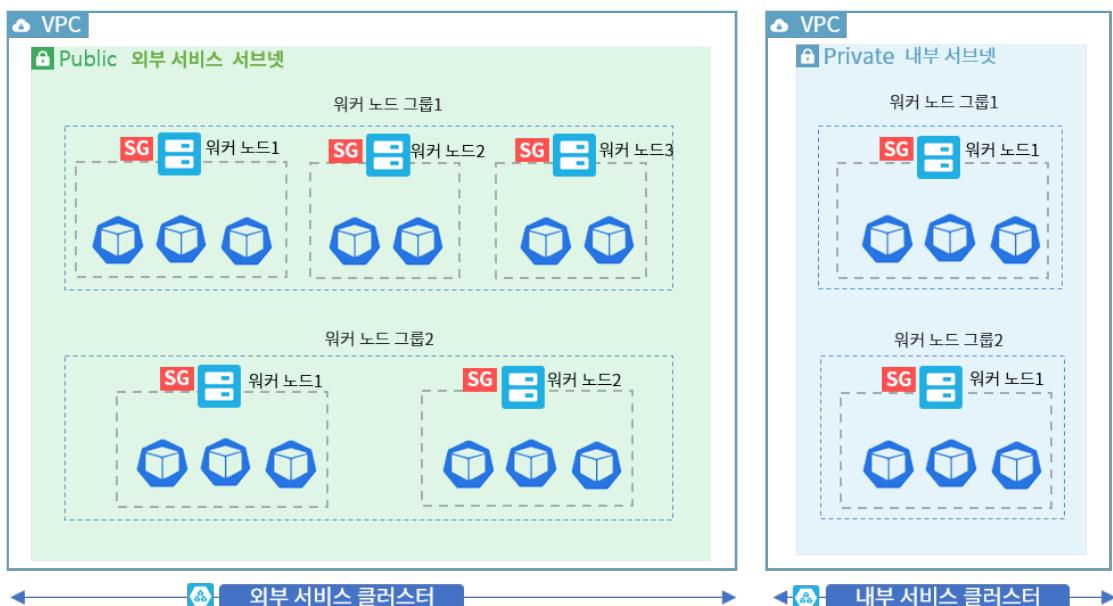


그림 27 워크로드 설계 예시

네임스페이스 분리 구성

컨테이너가 사용하는 네임스페이스를 분리하지 않고 공유하여 사용할 경우 애플리케이션 간의 격리가 되지 않고 리소스 등의 충돌이나 보안 사고 발생 시 다른 컨테이너에 영향을 줄 수 있으므로 컨테이너가 사용하는 네임스페이스를 적절히 분리하여 구성하는 것이 필요합니다.

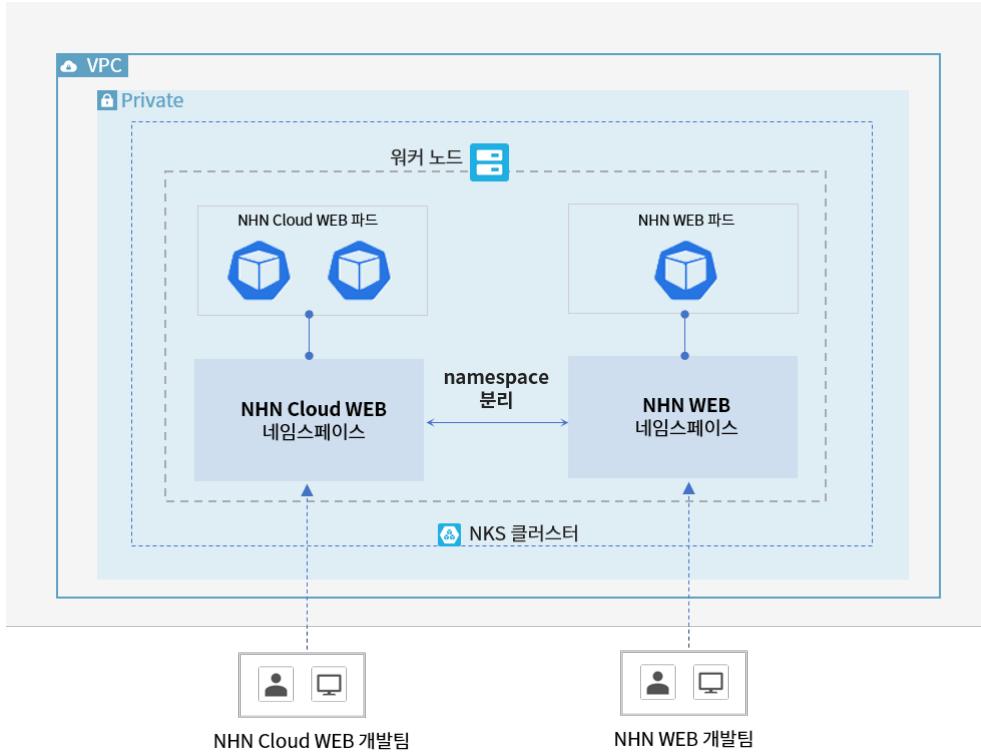


그림 28 네임스페이스 분리 구성의 예시

노드, 파드의 네트워크 접근제어 및 API 서버 접근제어

네트워크 접근제어는 외부에서 내부로의 접근뿐만 아니라 노드 간의 트래픽, 내부에서 외부로 향하는 트래픽 등 환경과 상황에 따라 네트워크 보안 정책 적용을 다층적으로 적용하는 것이 바람직합니다. 컨테이너 서비스의 네트워크 구성은 아래와 같이 다양한 방법으로 구성할 수 있습니다.

NKS 클러스터 최초 생성 시 네트워크 구성의 예시입니다. VPC와 서브넷, 파드 네트워크, Kubernetes 서비스 네트워크 사용자가 설정한 IP 대역으로 만들어진 예시입니다.

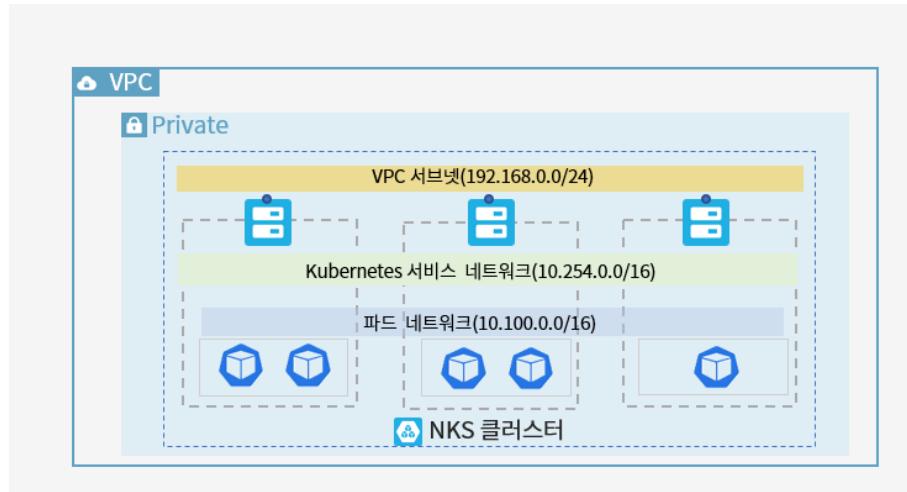


그림 29 NKS 클러스터 최초 생성 시 네트워크 예시

최초 구성에 클러스터 IP 탑으로 10.254.235.35:8080 서비스를 생성한 예시입니다. 클러스터 내부에서만 접근 가능하며 Kubernetes가 지원하는 기본적인 형태의 서비스입니다. 셀렉터로 지정된 파드들로 트래픽이 분배됩니다.

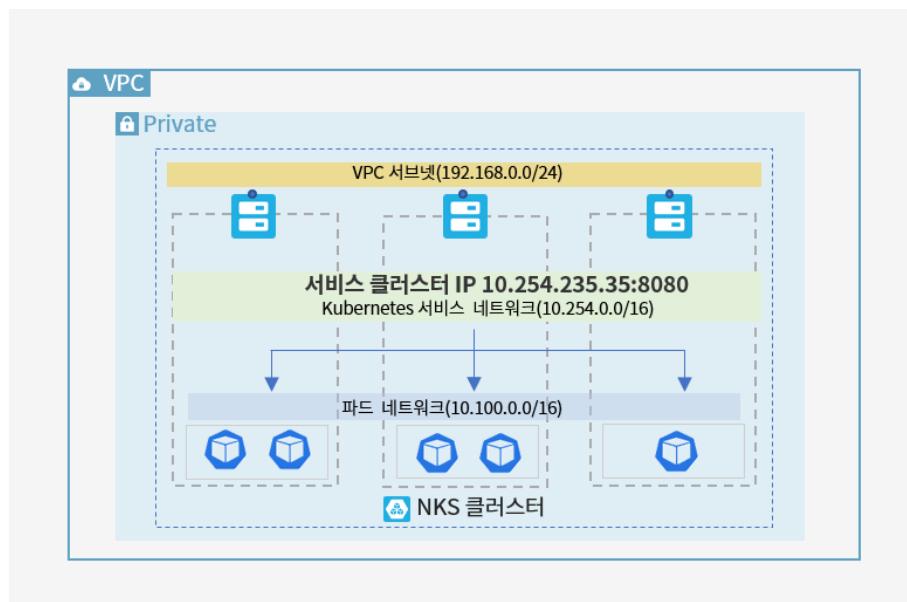


그림 30 NKS 클러스터의 클러스터 IP 구성 예시

아래는 서비스(service) 생성을 위해 작성된 YAML 파일 예시입니다. 아래와 같이 YAML 파일로 작성 후 kubectl 명령어 (`$ kubectl -f [파일명]`)를 통해 클러스터에 서비스를 생성할 수 있습니다.

- 서비스 작성 예시

```
# nhn-nginx-service.yaml 작성 예시
apiVersion: v1
kind: Service
metadata:
  name: nhn-nginx-service
spec:
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 80
  selector:
    app: nhn-nginx
```

외부에서 연결 가능한 서비스의 한 유형으로 노드 포트를 생성하여 구성한 예시입니다. 외부에서 노드 IP의 특정 포트 (<NodeIP>:<NodePort>)로 들어오는 트래픽은 서비스를 통해 파드로 전달됩니다.

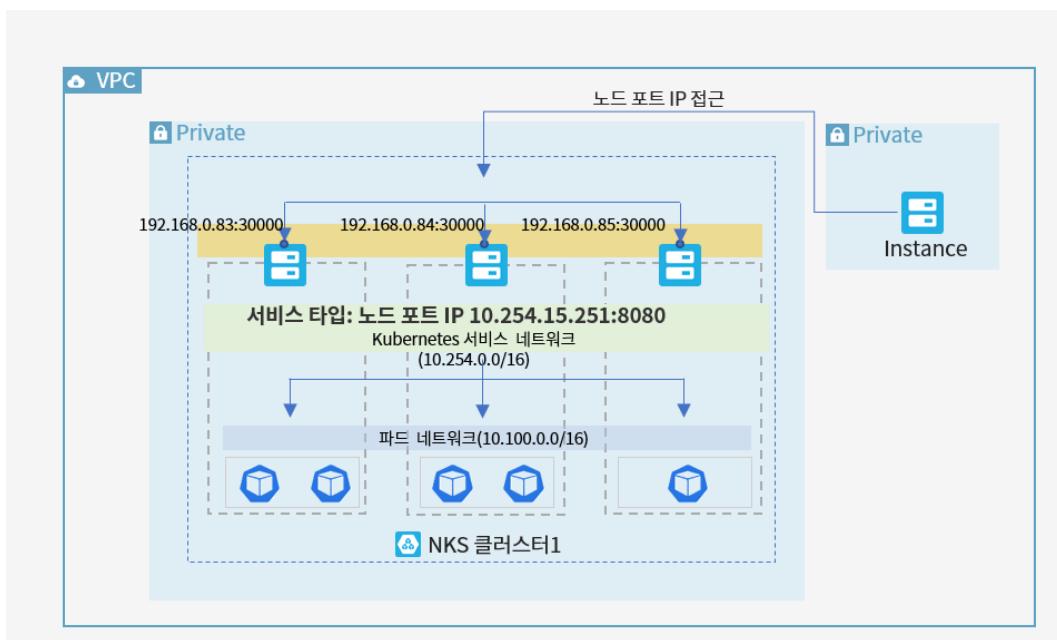


그림 31 노드 포트 구성 예시

NHN Cloud와 같은 CSP(cloud service provider)에서 제공하는 로드 밸런서를 이용하여 서비스를 구성하는 방법입니다. 외부에서 유입된 트래픽은 로드 밸런서와 연결된 서비스를 통해 파드로 전달됩니다.

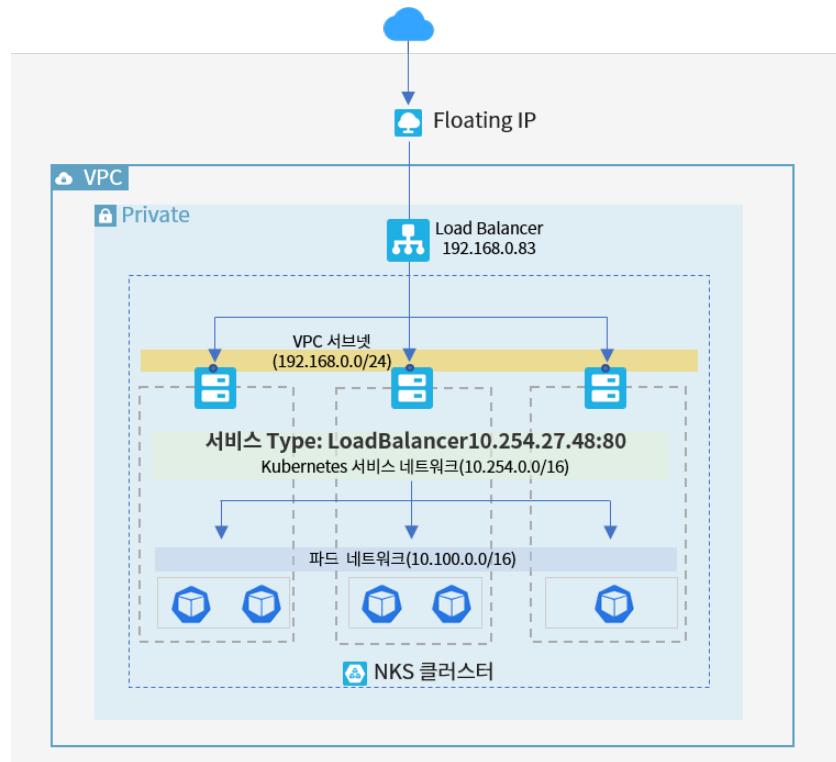


그림 32 로드 밸런서 구성 예시

아래 구성은 인그레스를 설치하여 도메인 주소로 파드를 연결할 수 있는 구성 예시입니다. 인그레스를 설치하여 여러 개의 도메인으로 트래픽을 분산할 수 있습니다.

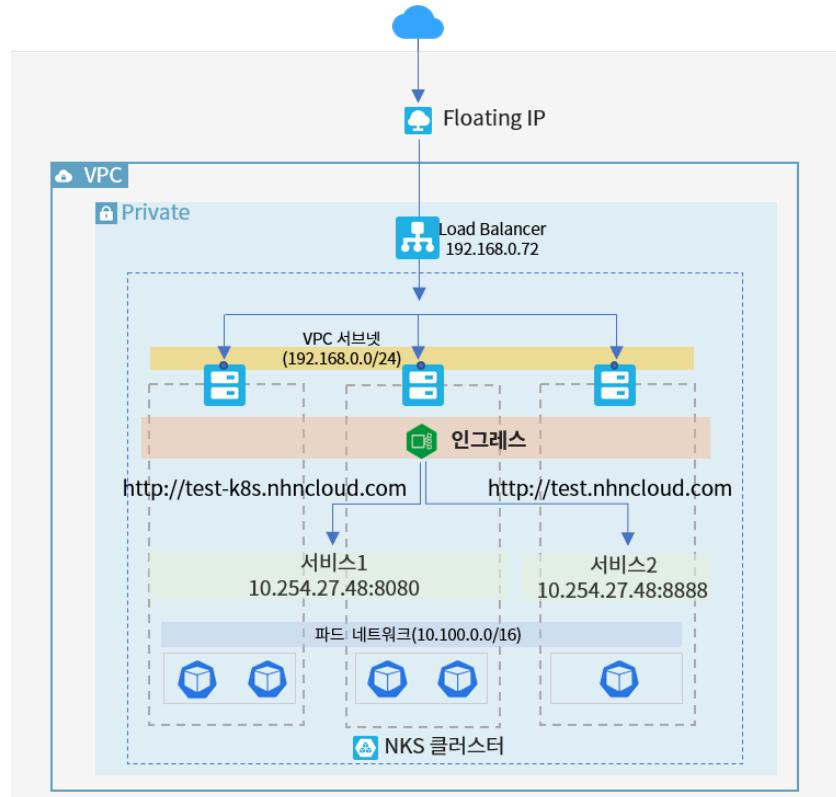


그림 33 NGINX 인그레스 구성 예

• Load Balancer IP 접근제어 기능을 활용한 접근통제

NHN Cloud에서 제공하는 Load Balancer는 IP 접근제어를 설정하여 특정 IP만 접근을 허용하거나 차단할 수 있습니다.

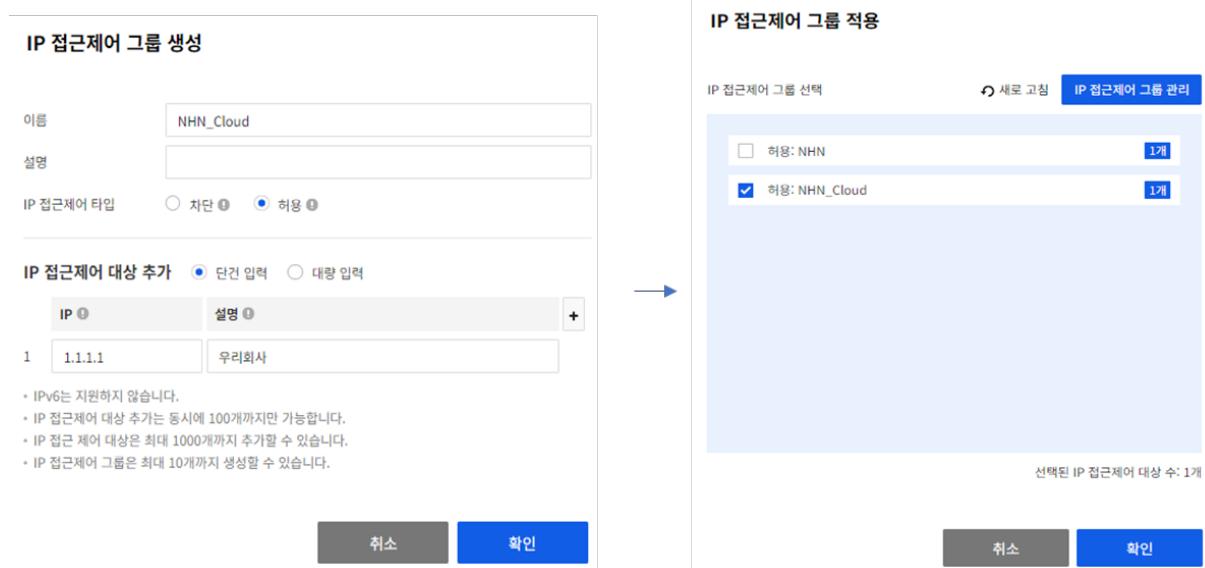


그림 34 NHN Cloud Load Balancer의 IP 접근제어 기능 설정

- Network Firewall, Security Groups을 활용한 워커 노드 간의 접근 통제

워커 노드는 OS가 설치되어 있는 파드 등이 배포되는 호스트입니다. NKS 클러스터 생성 시 NHN Cloud 인스턴스로 생성이 되므로 워커 노드 간의 통신은 NHN Cloud의 Security Groups, Network Firewall 등으로 네트워크 접근통제가 가능합니다.

NKS 클러스터 생성 시 워커 노드에는 Security Groups이 디폴트로 생성되어 파드 생성 및 관리 등에 필요한 원격지와의 통신 정책이 적용됩니다. 그 외 필요한 원격 접속이나 서비스 통신 등에 대해서는 새로운 Security Groups을 적용하거나 Network Firewall을 통해 제어할 수 있습니다.

NHN Cloud의 Network Firewall 서비스 구성은 [NHN Cloud 사용자 가이드](#)나 [네트워크 아키텍처 보안 가이드](#)에서 상세히 확인할 수 있습니다.

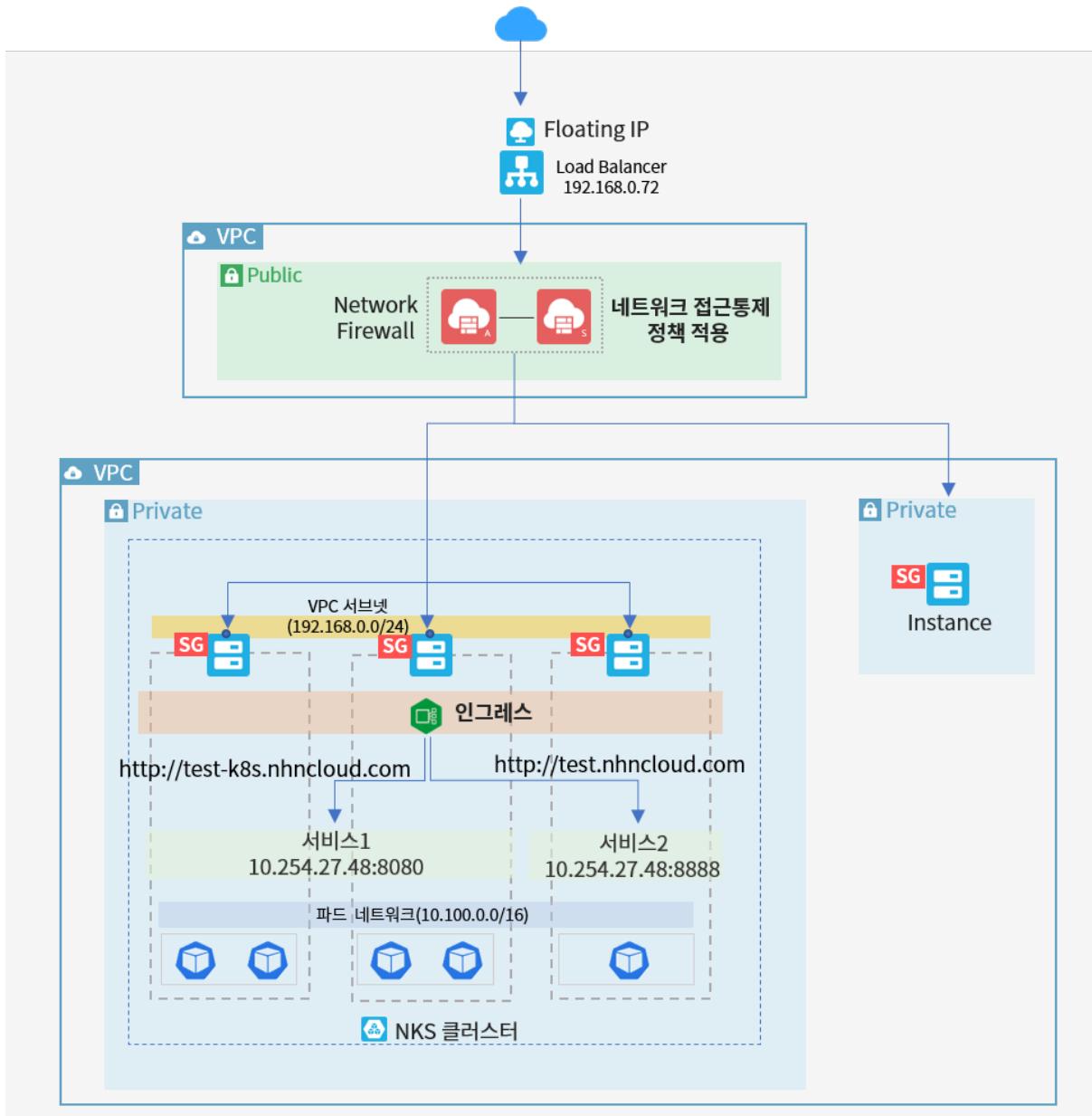


그림 35 NHN Cloud Network Firewall, Security Groups의 인스턴스 접근통제 적용 예시

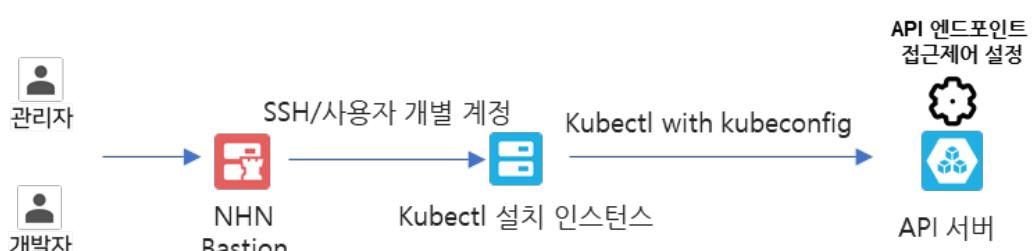
• API 서버 접근통제

API 서버는 NKS에서는 OS 형태로 노출되는 인스턴스는 아니지만 관리자나 개발자가 API 서버에 kubectl 명령어를 사용하여 외부 접근이 가능한 서버이며, 모든 배포 및 관리는 API 서버를 통해 이루어지므로 API 서버 접근을 필요한 클라이언트에게만 허용하는 것이 필요합니다. NKS는 API 엔드포인트 접근제어 기능을 통해 특정 IP의 접근을 허용하거나 차단할 수 있습니다. 그리고 개발자 및 운영자는 클러스터의 리소스를 생성, 삭제, 관리를 위해 kubectl을 통해 API 서버와 통신합니다. 따라서 Kubectl 명령어를 통해 API 서버와 통신하는 클라이언트를 최소화하고 클라이언트의 활동(명령어 등)을 모니터링할 수 있도록 구성하는 것이 좋습니다.

아래 그림의 구성에서는 Kubernetes API 서버는 kubectl이 설치된 인스턴스의 IP 주소에서만 접근을 허용하여 보안을 강화할 수 있습니다. 그러나 특정 명령어 차단이나 모니터링에는 한계가 있습니다.



NHN Cloud가 제공하는 NHN Bastion을 통해 API 서버 접근통제를 적용하여 API 서버로 전달되는 명령어 통제 기능을 이용하여 관리자나 개발자가 API 서버에 전달하는 명령어를 제한하고 모니터링할 수 있습니다.



API 엔드포인트 접근 제어 설정 변경

사용	<input checked="" type="radio"/> 사용	<input type="radio"/> 사용 안 함						
접근 제어 타입	<input type="radio"/> 차단	<input checked="" type="radio"/> 허용						
대상 추가	<input checked="" type="radio"/> 단건 입력	<input type="radio"/> 대량 입력						
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">IP ⓘ</td> <td style="padding: 5px;">설명</td> <td style="text-align: right; padding: 5px;"><button style="border: none; background-color: transparent; font-size: small;">+</button></td> </tr> <tr> <td style="padding: 5px;">1.1.1.1</td> <td style="padding: 5px;"></td> <td style="text-align: right; padding: 5px;"><button style="border: none; background-color: transparent; font-size: small;">-</button></td> </tr> </table>			IP ⓘ	설명	<button style="border: none; background-color: transparent; font-size: small;">+</button>	1.1.1.1		<button style="border: none; background-color: transparent; font-size: small;">-</button>
IP ⓘ	설명	<button style="border: none; background-color: transparent; font-size: small;">+</button>						
1.1.1.1		<button style="border: none; background-color: transparent; font-size: small;">-</button>						

* IPv6는 지원하지 않습니다.
* IP 접근제어 대상 추가는 동시에 100개까지만 가능합니다.

취소 **확인**

그림 38 NKS API 엔드포인트 접근제어 설정

• 인그레스 네트워크 접근제어 설정

1. IP 제한 설정

NHN Cloud의 Network Firewall이나 Load Balancer의 IP 제어 기능 외 인그레스 리소스의 애너테이션(annotation)을 설정하여 IP 제한을 적용할 수 있습니다. 인그레스 YAML 파일 예시는 아래와 같습니다. 아래와 같이 NGINX 인그레스 서비스를 작성하여 배포할 경우 특정 IP에서만 접근을 허용할 수 있습니다. 또는 “nginx.ingress.kubernetes.io/blacklist-source-range”를 사용할 경우 특정 IP를 접근 차단하도록 설정할 수도 있습니다.

```
# Ingress IP control Test Yaml 예시
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/whitelist-source-range: "1.1.1.1/32" # 허용할 IP 대역 설정
spec:
  rules:
  - host: test-k8s.nhncloud.com
    http:
      paths:
      - backend:
          service:
            name: nhn-nginx-nodeport-service
            port:
              number: 8080
      pathType: ImplementationSpecific
```

2. 도메인별 IP 제한 설정

위의 방법은 인그레스가 서비스하는 모든 도메인에 대해 IP 접근제어를 설정하지만, 인그레스 애너테이션에 스니펫을 사용하여 도메인별로 서로 다른 IP 제어 정책을 적용할 수 있습니다.

아래는 서비스별(nhn, nhncloud)로 IP 제한 설정을 다르게 적용한 예시입니다. 아래 “# Ingress Domain access control Test yaml” 코드의 굵은 글씨로 표시된 부분은 클라이언트 IP 1.1.1.1에서 nhn.com에 접근할 경우 허용하고 그 외 IP 접근 시 차단(HTTP Forbidden 메시지 표시)하도록 설정한 것이며 nhncloud.com 접근에 대해서는 IP 제한을 설정하지 않은 예시입니다.

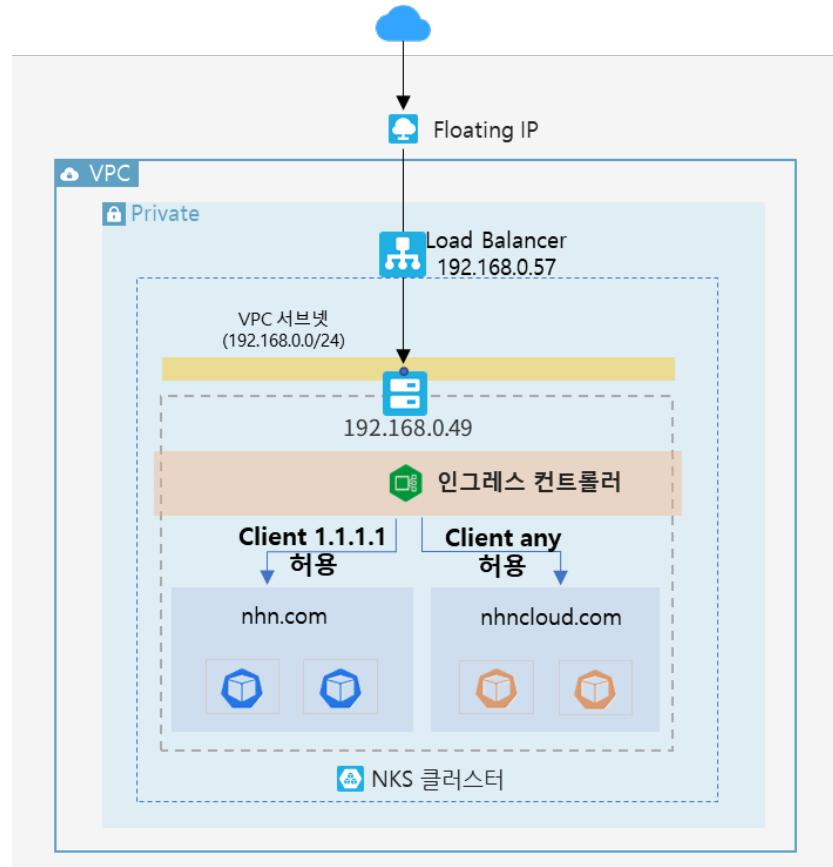


그림 39 도메인별 접근제어 구성

```

# Ingress Domain access control Test yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/configuration-snippet: | # snippet을 이용한 IP 접근제어 설정
      access_by_lua_block {
        local host = ngx.var.host
        local allowed_ips = {
          ["nhn.com"] = "1.1.1.1" # 클라이언트 IP 1.1.1.1만 nhn.com 접근 허용함
        }

        local client_ip = ngx.var.remote_addr
        if allowed_ips[host] and allowed_ips[host] ~= client_ip then
          ngx.exit(ngx.HTTP_FORBIDDEN)
        end
      }
    spec:
      rules:
        - host: nhn.com
          http:
            paths:
              - backend:
                  service:
                    name: nhn-service
                    port:
                      number: 80
                pathType: ImplementationSpecific
        - host: yeji.com
          http:
            paths:
              - backend:
                  service:
                    name: nhncloud-service
                    port:
                      number: 80
                pathType: ImplementationSpecific

```

해당 예시는 nginx-ingress를 사용할 경우 적용할 수 있는 예시이며 nginx ingress controller에서 snippet을 사용 가능하도록 설정해 주어야 합니다.

“kubectl describe configmap [configmap 이름] -n [nginx ingress controller namespace]” 명령어를 사용하여 snippet이 true로 설정되어 있는지 확인하고 false로 되어 있을 경우 “kubectl edit configmap [configmap 이름] -n [nginx ingress controller namespace]” 명령어를 통해 true로 변경합니다.

• 파드와 파드의 네트워크 접근통제

기본으로는 모든 파드의 모든 통신이 허용된 상태입니다. Kubernetes의 네트워크 정책을 이용할 경우 파드의 인바운드(ingress) 또는 아웃바운드(egress) 네트워크 정책을 적용할 수 있습니다. 네트워크 정책은 네트워크 플러그인(예: Calico)에 의해 구현되며 정책 적용은 YAML 파일 등으로 작성하여 kubectl 명령어로 적용시킬 수 있습니다.

CNI(container network interface)에 따라 네트워크 정책이 지원되지 않는 경우가 있으니 NKS에서 생성한 클러스터의 CNI를 확인하는 것이 필요합니다. NKS에서 네트워크 정책이 지원되는 CNI는 Calico(칼리코)입니다. NKS의 CNI 관련 정보는 [NHN Cloud 사용자 가이드](#)에서 확인 가능하며 네트워크 정책 설정 방법 등은 [Kubernetes 공식 사이트](#)에서 상세한 내용을 확인할 수 있습니다.

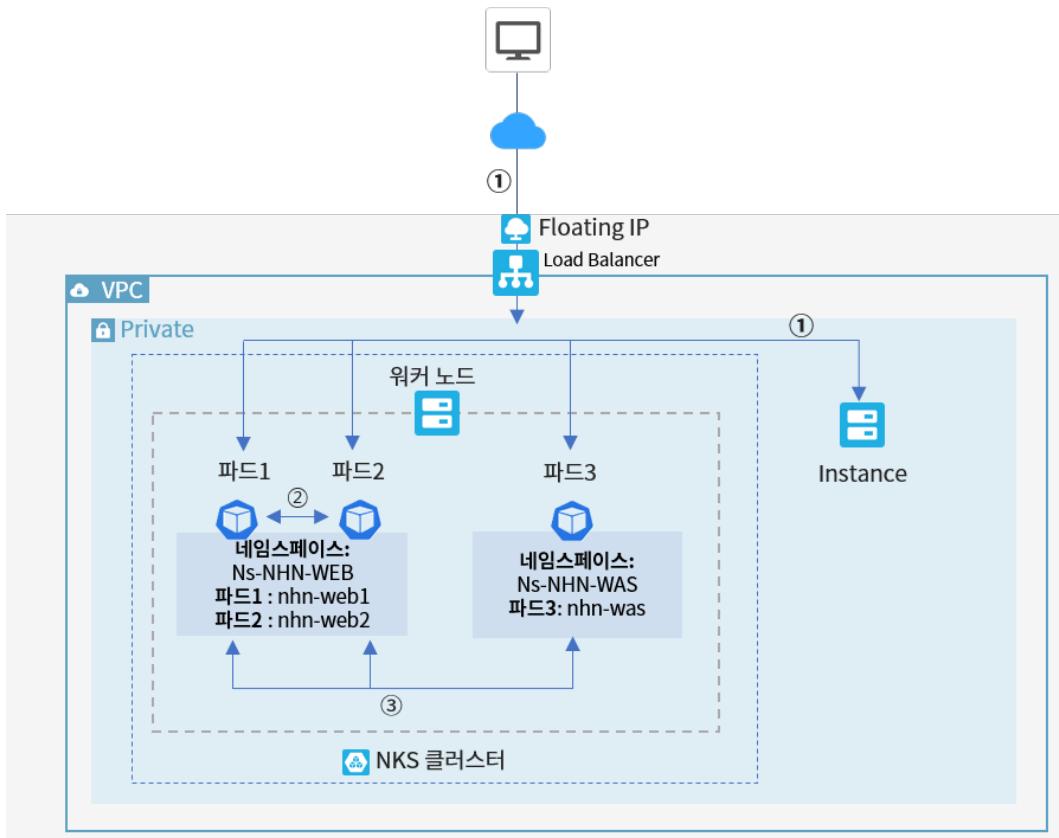


그림 40 네트워크 정책 적용 구성 예시

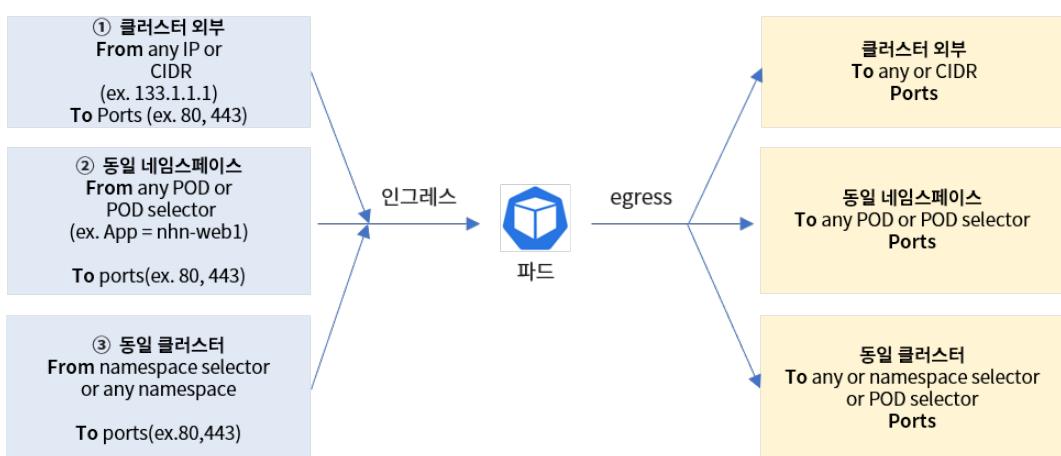


그림 41 네트워크 정책 적용 예시

- 파드 네트워크 정책 예시 - 외부 특정 IP 허용 YAML 파일 작성 예시

```
# POD network policy 작성 예시
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-policy
spec:
  podSelector: nhn-web
  policyTypes:
    - Ingress
  ingress:
    - from:
      - ipBlock:
          cidr: 1.1.1.1/32
    ports:
      - port: 80
```

NKS 클러스터 계정 인증과 권한/명령어 관리

개발자 또는 관리자는 클러스터 내의 리소스를 생성하거나 삭제, 관리를 위해 Kubernetes kubectl 도구를 이용하여 API 서버에 접근하거나 애플리케이션을 할당된 계정으로 파드 형태로 배포합니다.

클러스터의 보안과 계정으로 인한 침해 사고 방지를 위해 계정의 적절한 인증과 권한 관리는 매우 중요합니다.

Kubernetes에서 계정은 인증, 권한, 승인 컨트롤러(admission controller)라는 과정의 계정 관리 프로세스를 제공합니다. 인증 과정을 통해 계정을 식별한 후 사전에 정의된 역할과 권한을 할당하고, 승인 컨트롤러를 활용하여 세부적인 권한 제어를 수행할 수 있습니다.

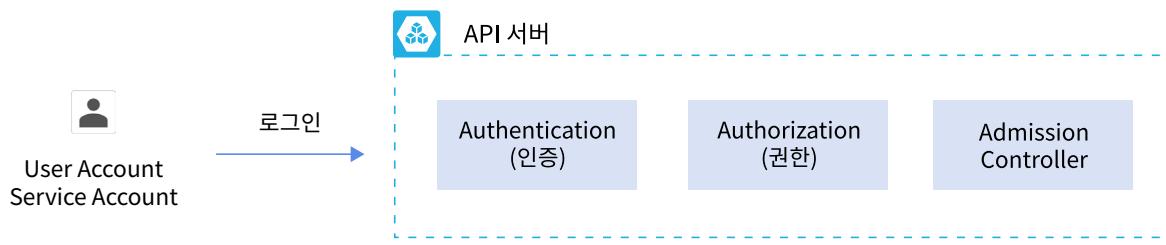


그림 42 Kubernetes의 인증과 권한 과정

• Kubernetes 인증 관리

NKS 클러스터 접근을 위해서는 클러스터의 kubeconfig 파일이 필요합니다. Kubeconfig 파일은 NKS에서 클러스터를 생성한 후 다운로드할 수 있습니다. Kubernetes는 여러 다양한 인증 방식(토큰 방식, X.509 인증서 기반 인증, OAuth 연동 인증, 웹훅 모드 등)을 제공하고 있으며 NKS에서 제공하는 kubeconfig를 사용하면 kubeconfig에 저장된 계정과 인증서 데이터를 이용하여 인증 후 API 서버에 접근할 수 있습니다.

Kubeconfig에 대한 사용 방법은 [NHN Cloud 사용자 가이드](#)에서 확인 가능하며 Kubernetes의 추가 상세한 인증 관련 내용은 [Kubernetes 공식 홈페이지](#)에서 확인할 수 있습니다.

Kubeconfig 파일에는 클러스터 정보와 인증을 위한 정보 등이 포함되어 있으므로 유출 또는 분실하지 않도록 주의하여 보관하는 것이 필요합니다.

+ 클러스터 생성		클러스터 삭제	클러스터 오너 변경	현재 목록 내에서 필터링 됩니다.	필터		
		클러스터 이름	노드 수	Kubernetes 버전	kubeconfig 파일	작업 상태	k8s API 상태
<input checked="" type="checkbox"/>		k8s-advanced-cluster	3	v1.27.3	다운로드	●	●

그림 43 kubeconfig 파일 다운로드

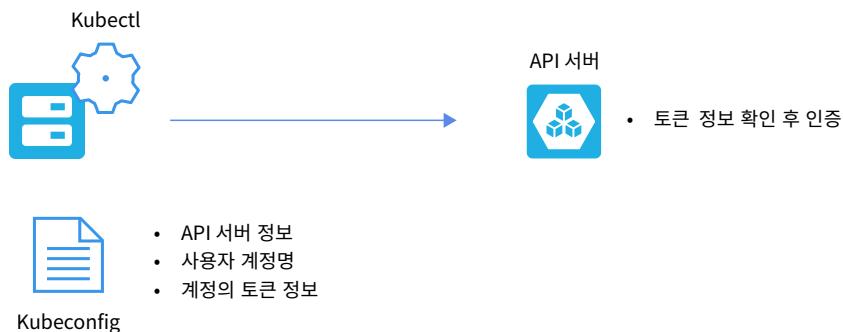


그림 44 kubeconfig 파일을 이용한 API 서버 접근

NKS에서 제공하는 Kubeconfig 파일에 Default 계정 외 사용자를 추가하여 x.509 방식의 인증을 통해 API 서버에 접근할 수 있습니다. 예를 들어 사용자 어카운트(user account)를 다음과 같은 방법으로 인증 등록할 수 있습니다.

Linux 서버에 openssl이 설치되어 있다고 가정하고 jane이라는 계정으로 x.509 인증을 사용하여 API 서버에 접근하는 예를 살펴보겠습니다.

3. Linux가 설치되어 있는 서버에서 openssl을 이용하여 개인 키를 생성합니다.

```
# openssl genrsa -out jane.key 2048
```

4. 인증서 생성 요청서(.csr 파일)를 작성합니다.

아래 명령어는 jane.key 파일을 이용하여 jane.csr을 작성하는 것입니다.

```
# openssl req -new -key jane.key -subj "/CN=jane" -out jane.csr
```

5. NKS API 서버에 인증서 생성을 요청합니다.

1) 인증서 생성 요청을 위한 YAML 파일을 작성합니다.

```
# BASE64_CSR=$(cat jane.csr | base64 | tr -d '\n')
```

```
# vi csr.yaml
```

```

apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: jane
spec:
  groups:
    - system:authenticated
  request: ${BASE64_CSR}
  signerName: kubernetes.io/kube-apiserver-client
  usages:
    - client auth

```

2) kubectl 명령어로 YAML 파일을 적용합니다.

```
# kubectl apply -f csr.yaml
```

6. API 서버에서 CSR 요청 상태를 확인하고 승인합니다.

1) CSR 요청 상태를 확인(Pending은 승인 또는 거부를 기다리는 상태이며, 승인되면 Approved, Issued로 상태가 변경됩니다.)합니다.

```
# kubectl get csr
```

2) CSR 요청을 승인합니다.

```
# kubectl certificate approve jane
```

7. 승인 받은 CSR을 CRT 형식으로 변환합니다.

```
# kubectl get csr jane.csr -o jsonpath='{.status.certificate}' | base64 --decode >jane.crt
```

8. kubeconfig(kubectl 클라이언트의 OS가 Linux인 경우 ~/.kube/config 파일에 해당됩니다.)에 jane의 계정 정보를 추가합니다.

```
# vi ~/.kube/config
```

```

apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: *****
  server: https://apiserver:6443
  name: toast-k8s-advanced-cluster
contexts:
- context:
  cluster: toast-k8s-advanced-cluster
  user: admin
  name: default
- context:
  cluster: toast-k8s-advanced-cluster
  user: jane
  name: jane
current-context: default
kind: Config
preferences: {}
users:
- name: admin
  user:
    client-certificate-data: *****
    client-key-data: *****
- name: jane
  user:
    client-certificate: /파일 경로/jane.crt
    client-key: /파일 경로/jane.key

```

9. 사용자를 전환하고 kubectl을 실행하면 jane에 할당되어 있는 역할에 따라 명령어 실행 결과가 도출됩니다.

```
# kubectl config use-context jane (계정 전환 명령어)
```

• Kubernetes 클러스터를 위한 계정과 역할 관리

계정 인증 이후 계정을 실행할 수 있는 역할을 관리하는 것은 모든 IT 시스템에서 매우 중요한 요소입니다. 최소 권한 및 직무 분리 원칙 등을 기반으로 역할을 할당하여 권한의 오남용을 막고 작업의 실수나 악의적인 행위로부터 위험을 감소시킬 수 있습니다.

NKS에서는 RBAC(role-based access control)를 사용하여 계정의 권한을 관리할 수 있습니다. RBAC 방식을 사용하면 룰(role)과 클러스터룰(clusterrole)을 통해 Kubernetes 리소스에 대한 액세스 권한을 정의하고 제어할 수 있습니다.

클러스터룰은 클러스터 수준의 리소스(예: 노드, 네임스페이스)에 대한 작업 권한을 부여하고 클러스터룰 바인딩으로 계정과 매핑하여 권한을 관리할 수 있습니다. 룰은 특정 네임스페이스의 리소스(예: 파드, 서비스)에 대한 작업(예: 생성, 읽기, 수정, 삭제) 권한을 부여하고 룰바인딩을 통해 계정에 역할을 부여할 수 있습니다.

따라서 서비스 개발자나 서비스 관리자 등으로 직무를 분리하여 클러스터룰, 룰 등을 통해 계정별로 권한이 필요한 사용자에게 할당할 수 있습니다.

Kubernetes에서는 아래와 같이 두 종류의 계정을 지원합니다. 사람이 사용하는 사용자 어카운트(user account)와 애플리케이션(프로세스)이 사용하는 서비스 어카운트(service account)입니다. 두 계정은 관리 정책이나 감사 방법이 다를 수 있으므로 Kubernetes에서는 분리해서 생성, 관리하는 것을 권장하고 있습니다.

표 4-1 계정의 종류

계정의 종류	설명
사용자 어카운트 (user account)	사람을 위한 계정으로 Kubernetes 내에 정의되지 않으며 클러스터의 모든 네임스페이스에서 고유해야 합니다. 사용자 어카운트는 Kubernetes 외부에서 생성 관리합니다.
서비스 어카운트 (service account)	네임스페이스에 정의하는 계정으로 Kubernetes에서 생성 관리하는 계정입니다. 파드에서 실행되는 프로세스에 대한 ID를 관리하는 것으로 애플리케이션 프로세스용입니다.

Kubernetes에서는 클러스터 전역에 적용 가능한 클러스터롤과 네임스페이스에 정의하는 룰을 제공합니다.

표 4-2 룰(role)의 종류

룰(role)의 종류	설명
룰	네임스페이스를 지정하여 권한을 부여하고 룰바인딩(rolebinding)을 통해 계정에 역할을 부여합니다.
클러스터룰	룰과 달리 네임스페이스를 지정하지 않고 클러스터 단위로 역할을 부여할 수 있는 방법으로 클러스터룰바인딩(clusterrolebinding)을 통해 계정에 역할을 부여합니다.

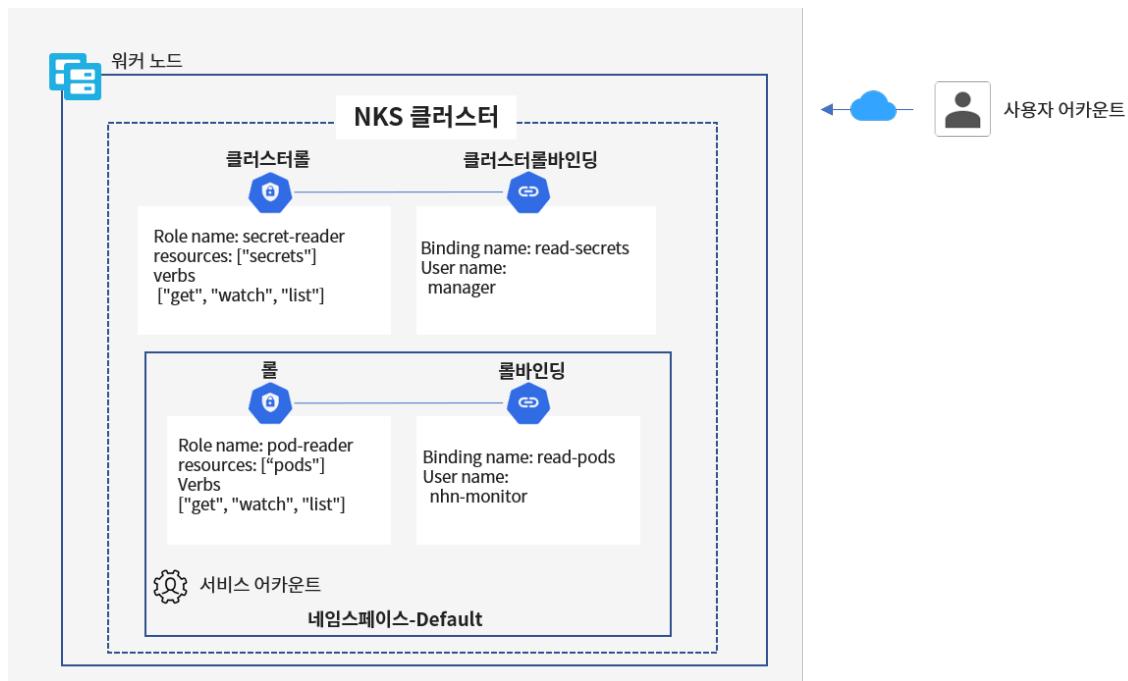


그림 45 계정과 역할 정의 예시

표 4-3 RBAC 정의 예시

종류	네임 스페이스	대상 리소스	권한	바인딩 종류	바인딩에 속한 계정	설명
클러스터롤	N/A	시크릿	get, watch, list	클러스터 롤바인딩	manager	manager 계정은 kubectl 명령어를 통해 시크릿 리소스를 “get, watch, list” 할 수 있습니다. 즉 시크릿의 정보를 확인할 수 있는 권한이 부여된 것입니다.
룰 (pod-reader)	Default	파드	get, watch, list	롤바인딩	nhn- monitor	nhn-monitor 계정은 kubectl 명령어를 통해 파드를 “get, watch, list”하여 실행된 파드 정보를 확인할 수 있습니다.

아래는 룰/롤바인딩, 클러스터롤/클러스터롤바인딩을 생성하기 위해 작성하는 YAML 파일 예시이며 작성된 YAML 파일을 kubectl 명령어(# kubectl -f [파일명])를 통해 클러스터에 적용할 수 있습니다.

- 룰 예시

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"] #권한을 작성
```

[출처: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>]

- 롤바인딩 예시

```

apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
  # You can specify more than one "subject"
  - kind: User
    name: jane # "바인딩에 포함할 계정
    apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role
  name: pod-reader # 바인드 할 Role name
  apiGroup: rbac.authorization.k8s.io

```

[출처: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>]

- 클러스터롤 예시

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
  - apiGroups: [""]
    #
    # at the HTTP level, the name of the resource for accessing Secret
    # objects is "secrets"
    resources: ["secrets"]
    verbs: ["get", "watch", "list"]

```

[출처: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>]

- 클러스터롤바인딩 예시

```

apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to read secrets in any namespace.
kind: ClusterRoleBinding
metadata:
  name: read-secrets
subjects:
- kind: user
  name: manager # Name is case sensitive
apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io

```

[출처: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/>]

그 외, Kubernetes를 직접 구축하여 사용할 경우 ABAC(attribute-based access control) 방식이나 노드 권한 등의 권한 제어 방식을 사용할 수 있습니다. 노드 권한 부여는 kubelet에 의해 이루어지는 요청에 대해 권한을 부여하는 것이며 ABAC는 속성 기반의 권한 부여 방식입니다. 권한에 대한 추가 상세한 내용은 [Kubernetes 공식 홈페이지](#)에서 확인할 수 있습니다.

- 승인 컨트롤러(admission controller)

승인 컨트롤러는 Kubernetes API 서버 요청을 가로채 객체를 변경하거나 요청을 거부할 수 있는 기능입니다. 예를 들어 승인 컨트롤러 플러그인 중 MutatingAdmissionWebhook을 이용하여 파드에 자동으로 특정 서비스 어카운트를 할당할 수 있습니다. 파드 배포 시 서비스 어카운트를 지정하지 않으면 디폴트 계정이 자동으로 할당되는데 이것을 다른 서비스 어카운트로 할당하도록 변경하는 방식입니다. 승인 컨트롤러를 사용하는 방법은 [Kubernetes 공식 홈페이지](#)에서 보다 상세하게 확인할 수 있습니다.

NKS에서 제공하는 클러스터 버전과 클러스터 생성 시점에 따라 승인 컨트롤러에 적용되는 플러그인의 종류가 다르므로 버전 정보는 [NHN Cloud 사용자 가이드](#)에서 확인하세요.

- Kubectl 명령어 통제(솔루션 사용)

개발자 또는 관리자가 Kubectl을 통해 파드 배포, 업데이트, 삭제, 관리 등을 할 때 보안 솔루션을 이용하여 명령어를 모니터링하거나 통제하는 방법도 있습니다. 시스템 접근제어 솔루션과 같은 보안 솔루션은 사용자별 계정을 발급하고 계정별로 사용 가능한 명령어를 제어하거나 모니터링할 수 있으므로 클러스터롤, 룰 등과 함께 적용하여 사용자의 실수나 권한 오남용 등의 보안 사고를 방지하거나 사후 추적을 위해 활용할 수 있습니다.

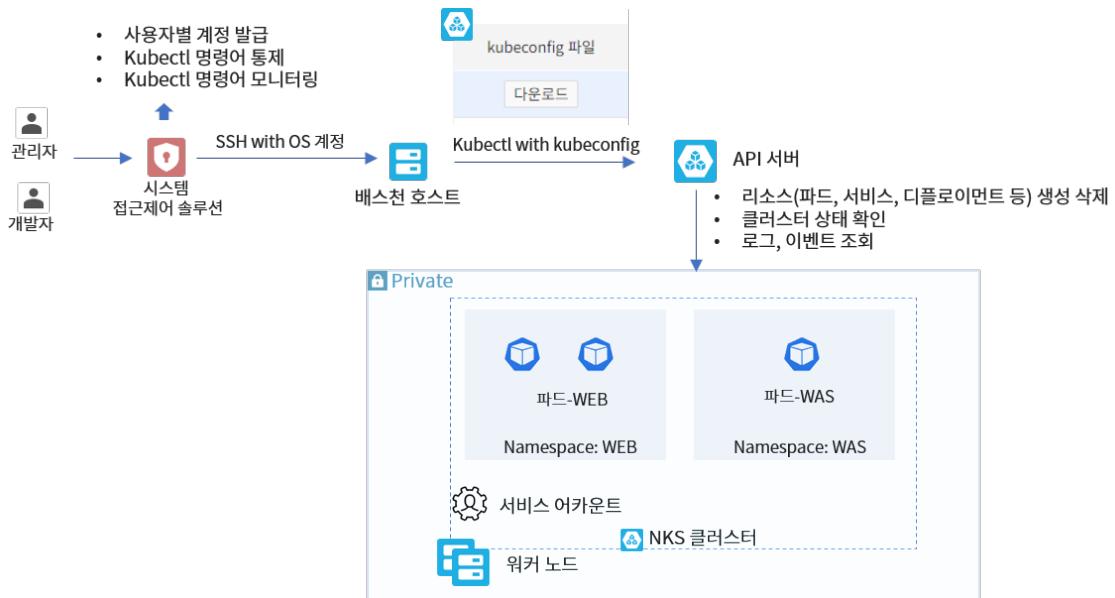


그림 46 Kubectl의 클러스터 접근 및 명령어 제어 방법 예시

컨테이너 배포 계정과 자원 관리

• 컨테이너 실행 계정, 권한 관리

컨테이너에서 실행되는 프로세스에 부여되는 권한은 최소 권한 원칙에 따라 제한되어야 합니다. 불필요한 권한이 부여되면 컨테이너 이스케이프나 호스트 시스템 침해 등의 공격이 발생할 수 있습니다. 또한 특권 모드(privileged mode) 실행은 엄격히 제한되어야 합니다. 특권 모드로 실행되는 컨테이너는 호스트 시스템의 모든 리소스와 커널 기능에 액세스할 수 있게 됩니다. 따라서 반드시 필요한 경우가 아니라면 특권 모드 실행을 금지하는 것이 좋습니다.

파드 배포 시 특권 모드를 명시적으로 금지하거나 실행 계정을 지정하는 것으로, 권한으로 인한 보안 사고가 발생하는 것을 방지할 수 있습니다. Kubernetes 매니페스트에 SecurityContext를 적절히 정의하여 컨테이너 프로세스의 권한을 제한하고 특권 모드 실행을 방지할 수 있습니다.

SecurityContext에서는 다음과 같은 주요 설정이 가능합니다.

- 실행 사용자(runAsUser) 및 그룹(runAsGroup) 지정으로 최소 권한 원칙 적용
- 권한 있는 실행(privileged) 여부 설정
- Linux 커널 기능(capabilities) 제한
- SELinux/AppArmor 프로파일 할당 등

- securityContext의 runAsUser 사용 예시

```
apiVersion: v1
kind: Pod
metadata:
  name: nhn-security-test
spec:
  securityContext:
    runAsUser: 1000
  containers:
    - name: nhn-nginx
      image: nhn-nginx:1.0
      ports:
        - containerPort: 80
  imagePullSecrets:
    - name: <your-image-pull-secret-name>
```

추가 정보 및 상세한 작성법은 Kubernetes 공식 홈페이지의 [파드 시큐리티 스탠다드](#) 또는 [Security Context 관련 페이지](#) 등에서 확인이 가능합니다.

- 컨테이너 자원 관리

파드 및 컨테이너가 무제한으로 컴퓨팅 자원을 소비하게 되면 해당 노드의 전체 리소스가 고갈되어 다른 워크로드의 실행에 영향을 미칠 수 있습니다. 이로 인해 전체 클러스터의 안정성과 가용성이 저하될 수 있습니다. 따라서 워크로드 간 성능 설정을 통해 특정 워크로드의 리소스 소비가 다른 워크로드의 성능 저하를 유발하지 않도록 해야 합니다. 또한 리소스 고갈 공격을 방지하기 위해서도 적절한 리소스 제한이 필요합니다. 이를 위해 관리자는 각 워크로드의 요구 사항을 평가하여 CPU, 메모리 등에 대한 리소스 제한을 설정해야 합니다. 이러한 제한 사항은 컨테이너 매니페스트의 리소스 필드에 정의하여 파드를 배포할 수 있습니다.

- resources requests, limits가 적용된 예시

```

apiVersion: v1
kind: Pod
metadata:
  name: nhn-resource-test
spec:
  containers:
    - name: nhn-nginx
      image: nhn-nginx:1.0
      ports:
        - containerPort: 80
      resources:
        requests:
          memory: "100Mi"
        limits:
          memory: "200Mi"
      args: [--vm, "1", "--vm-bytes", "150M", "--vm-hang", "1"]
  imagePullSecrets:
    - name: <your-image-pull-secret-name>

```

상세한 내용 및 추가 정보는 [Kubernetes 공식 홈페이지](#)에서 확인할 수 있습니다.

• 불필요 패키지 미포함

컨테이너에는 불필요하거나 원격 접속을 허용하는 등의 패키지가 포함되지 않도록 하는 것이 좋습니다. 패키지에 포함된 원격 접속 도구 등으로 인해 보안 사고가 발생할 수 있으므로 Dockerfile, 컨테이너 등에 SSH 등이 포함되어 있는지 검사하거나 배포가 되었다면 원격 접속 네트워크 IP, 포트를 차단하는 등의 조치를 취하는 것이 필요할 수 있습니다. 이를 위해 이미지 취약점 진단 도구를 활용하거나 서비스 네트워크 접속제어 등을 적용할 수 있으며 자세한 방법은 [4.3 이미지와 레지스트리 보안](#)과 [4.4 오케스트레이터 및 컨테이너 보안의 노드, 파드의 네트워크 접근제어 및 API 서버 접근제어](#)의 내용을 참조하실 수 있습니다.

• 시크릿(secrets) 관리

시크릿은 암호, 토큰, 키와 같은 중요한 데이터를 포함하는 Kubernetes 오브젝트입니다. 이러한 민감한 정보를 파드나 컨테이너 이미지에 직접 포함하는 것은 위험할 수 있습니다. 대신 시크릿을 활용하여 이러한 인증 관련 정보를 안전하게 관리할 수 있습니다. 파드에서는 시크릿을 볼륨으로 마운트하거나 환경 변수로 사용할 수 있습니다. 또한 시크릿은 주기적으로 변경하고, 용도 및 계정마다 서로 다른 시크릿을 사용하는 것이 좋습니다. 보안을 한층 더 강화하기 위해 RBAC 권한을 이용하여 시크릿을 조회, 생성 및 변경할 수 있는 권한을 최소화하여 계정에 부여해야 합니다. 또는 시크릿을 통한 인증 정보 관리가 어렵다고 생각될 경우 x.509와 같은 인증서 기반 인증 방식을 사용하는 것도 고려해 볼 수 있습니다. 마지막으로 시크릿의 데이터를 암호화하여 저장하는 것도 좋은 방법입니다. Kubernetes에서 시크릿을 보다 안전하게 사용하고 관리하는 모범 사례는 [Kubernetes 공식 홈페이지](#)에서 확인할 수 있습니다.

NHN Cloud는 etcd에 저장되는 시크릿을 AES/CBC 방식으로 암호화하여 저장하고 있습니다.

아래는 NKS와 NCR 연동 시 레지스트리 접근을 위한 인증을 위해 시크릿을 생성하여 사용하는 경우의 예시입니다.

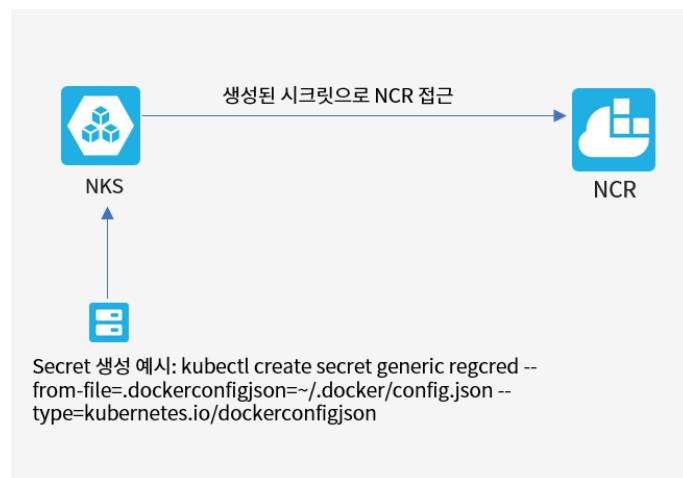


그림 47 NKS의 시크릿을 이용한 NCR 접근

4.5 모니터링 및 위협 탐지

실행 상태 모니터링

NHN Cloud의 NKS 메뉴에서 제공하는 클라우드 콘솔을 통해 자원을 모니터링하고 관리할 수 있습니다
또는 Kubernetes에서 제공하는 웹 대시보드를 구성하여 리소스의 상태나 오류 정보 등을 확인할 수 있습니다.

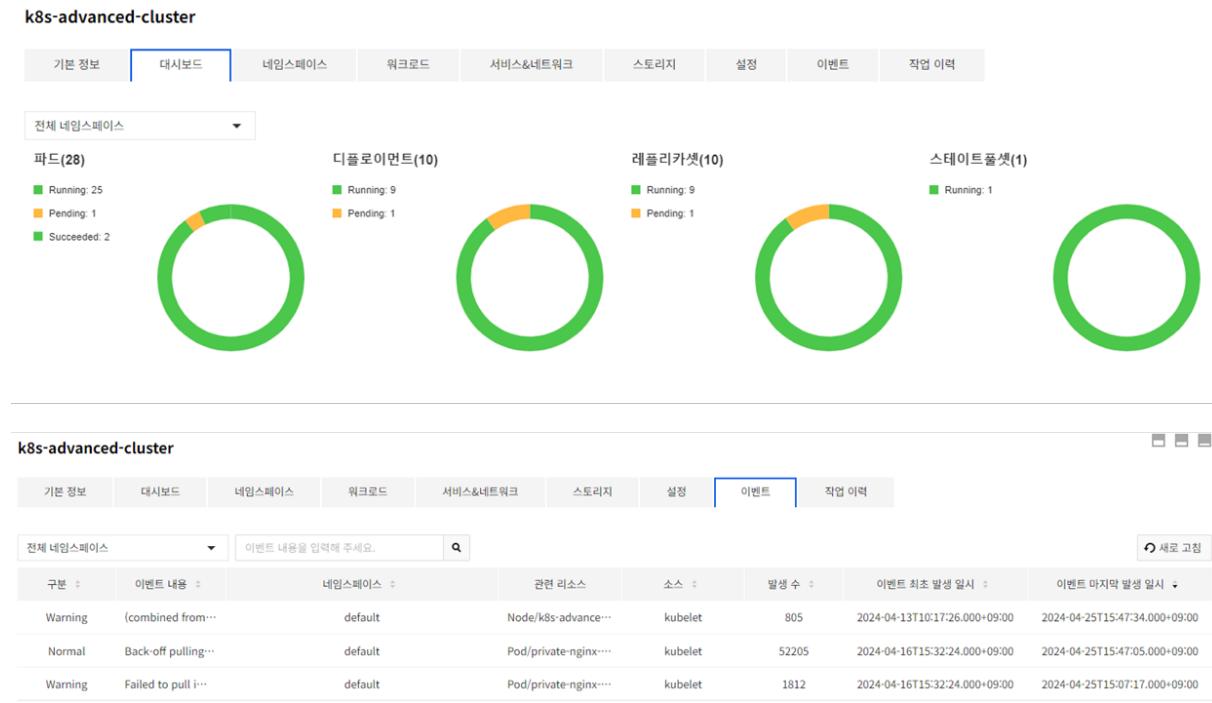


그림 48 NKS 대시보드 예시

컨테이너 로그 관리

NKS, NCR과 관련된 로그를 CloudTrail에서 확인할 수 있습니다. 클러스터, 파드, 서비스 어카운트 등이 생성되고 삭제된 로그를 확인할 수 있습니다. 자세한 이벤트 목록은 [NHN Cloud 사용자 가이드](#)에서 상세 확인이 가능합니다.

CloudTrail <small>BETA</small>								
프로젝트: (클라우드보안실)이예자-L...		소스: 전체	서비스: 전체	이벤트: 전체	아이디 검색	로그 내용 검색(2글자 이상)	URL & Appkey	사용자 가이드
시간	사용자	IP	이벤트	소스	서비스	프로젝트		
2024-03-21 10:41:04			시크릿 수정	API	기본 인프라 서비스			
2024-03-21 10:41:04			시크릿 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			서비스어카운트 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			클러스터풀바인딩 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			클러스터풀바인딩 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			클러스터풀바인딩 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			클러스터풀 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			클러스터풀 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			클러스터풀 생성	API	기본 인프라 서비스			
2024-03-21 10:41:04			서비스어카운트 생성	API	기본 인프라 서비스			
2024-03-21 10:41:01			클러스터풀 수정	API	기본 인프라 서비스			
2024-03-21 10:41:01			컨피그맵 생성	API	기본 인프라 서비스			
2024-03-21 10:41:01			클러스터풀 수정	API	기본 인프라 서비스			
2024-03-21 10:41:01			클러스터풀 수정	API	기본 인프라 서비스			
2024-03-21 10:41:01			컨피그맵 생성	API	기본 인프라 서비스			
2024-03-21 10:41:01			클러스터풀 수정	API	기본 인프라 서비스			

그림 49 NKS 이벤트 예시

로그는 SIEM 등을 활용하여 로그 관리 및 통합 모니터링 체계를 구축함으로써 내/외부 위협에 대한 모니터링 체계를 통해 위협에 대한 가시성을 더욱 높일 수 있습니다. SIEM에서는 사전 정의된 룰셋과 임계치를 기반으로 로그 데이터를 실시간으로 모니터링하고 상관 분석을 통해 보다 지능적인 위협 대응이 가능합니다.

Resource Watcher 활용

NHN Cloud의 조직 내 서비스에서 생성된 모든 리소스에 리소스 태그와 그룹을 이용하여 리소스별 발생한 이벤트를 관리하고 알림을 설정할 수 있습니다.

예를 들어 API 서버의 IP 접근제어 변경이 발생할 경우 설정된 콘솔 사용자에게 이메일이나 웹훅 기능을 통해 메시지를 전달 할 수 있습니다.

The screenshot shows the NHN Cloud Resource Watcher configuration page. At the top, there's a section for '알림 수정' (Modify Alert) with fields for '이름' (Name) set to 'API_IP_Access_Event' and '설명' (Description) set to '알림 설명 입력'. Below this is an '이벤트' (Event) selection section where '이벤트 선택' (Select Event) is selected. A dropdown menu shows '기본 인프라 서비스 > 클러스터 API 엔드포인트 IP 접근 제어 변경 완료' and '기본 인프라 서비스 > 클러스터 API 엔드포인트 IP 접근 제어 변경 시작'. A search bar and a list of events are shown, with '클러스터' (Cluster) selected. The event list includes '예산 추가', '예산 삭제', '예산 수정', '거버넌스 설정 변경', and 'IAM 로그인'. Below the event list is a navigation bar with buttons for '«', '<', '1', '2', '3', '4', '5', '>', and '»'. The next section is '알림 수신 대상' (Recipient Settings), which lists '조직 멤버>NHN Cloud 회원' and '(Email)' as the recipient. It includes a search bar and a table for selecting recipients by group or individual. The table columns are '구분' (Category), '이름' (Name), 'ID' (ID), '이메일' (Email), and checkboxes for 'Email' and 'SMS'. One row has the 'Email' checkbox checked. The final section at the bottom is '웹훅' (Webhook), with input fields for '웹훅 주소' (Webhook URL) and '비밀 키' (Secret Key), and a '추가' (Add) button.

그림 50 Resource Watcher 알림 설정

The screenshot shows the Resource Watcher interface with the following details:

- Header:** Resource Watcher (BETA), URL & Appkey, 사용자 가이드
- Left sidebar:** 리소스, 리소스 그룹, 리소스 태그, 알림
- Top navigation:** 알림 상태: 전체, 알림 검색, + 알림 생성, 알림 삭제
- Table header:** 알림명, 상태, 알림 설명, 최종 수정 일시, 알림 이력, 상세보기, 수정
- Table row:** API_IP_Access_Event, 활성화, 2024-05-20T15:33:52+09:00, [보기], [보기], [수정]
- Page controls:** 총 1개 목록 표시: 20개
- Sub-table header:** API_IP_Access_Event > 알림 이력 (ACTIVE)
- Sub-table filters:** 프로젝트명: 전체, 소스: 전체, 알송 결과: 전체
- Sub-table header:** 리소스별 일자, 일정 발생 일자: 2022-06-08 10:21 ~ 2024-05-20 23:59, 직접 입력
- Sub-table columns:** 알림 발생 일자, 프로젝트명, 리소스명, 서비스, 이벤트, 소스, 알송 결과
- Sub-table rows:**
 - 2024-05-20T15:36:55+09:00, 기본 인프라 서비스, 기밀 인프라 서비스, 풀어스터 API 엔드포인트 IP 접근 차단 변경 완료, ADMIN_CONSOLE, 성공
 - 2024-05-20T15:36:53+09:00, 기본 인프라 서비스, 기본 인프라 서비스, 풀어스터 API 엔드포인트 IP 접근 차단 변경 시작, ADMIN_CONSOLE, 성공
- Page controls:** 총 2개 목록 표시: 20개

그림 51 Resource Watcher 알림 내역

네트워크 공격 대응 및 모니터링

컨테이너로 운영하는 서비스도 온프레미스나 클라우드 IaaS 서비스를 활용하여 서비스를 제공할 때와 같이 네트워크를 통한 위협 대응을 위해 DDoS 공격 대응 정책을 적용하고 웹 방화벽 서비스를 활용하여 공격에 대응하도록 구성하는 것이 필요합니다. NHN Cloud는 DDoS Guard, WEB Firewall 서비스 등을 통해 이러한 위협, 공격에 대응할 수 있는 보안 서비스를 제공하고 있습니다. NHN Cloud의 Security Monitoring과 같은 서비스를 이용할 경우, 발생하는 보안 이벤트에 대해 상시 모니터링 및 관제 서비스 등을 제공 받을 수 있습니다.

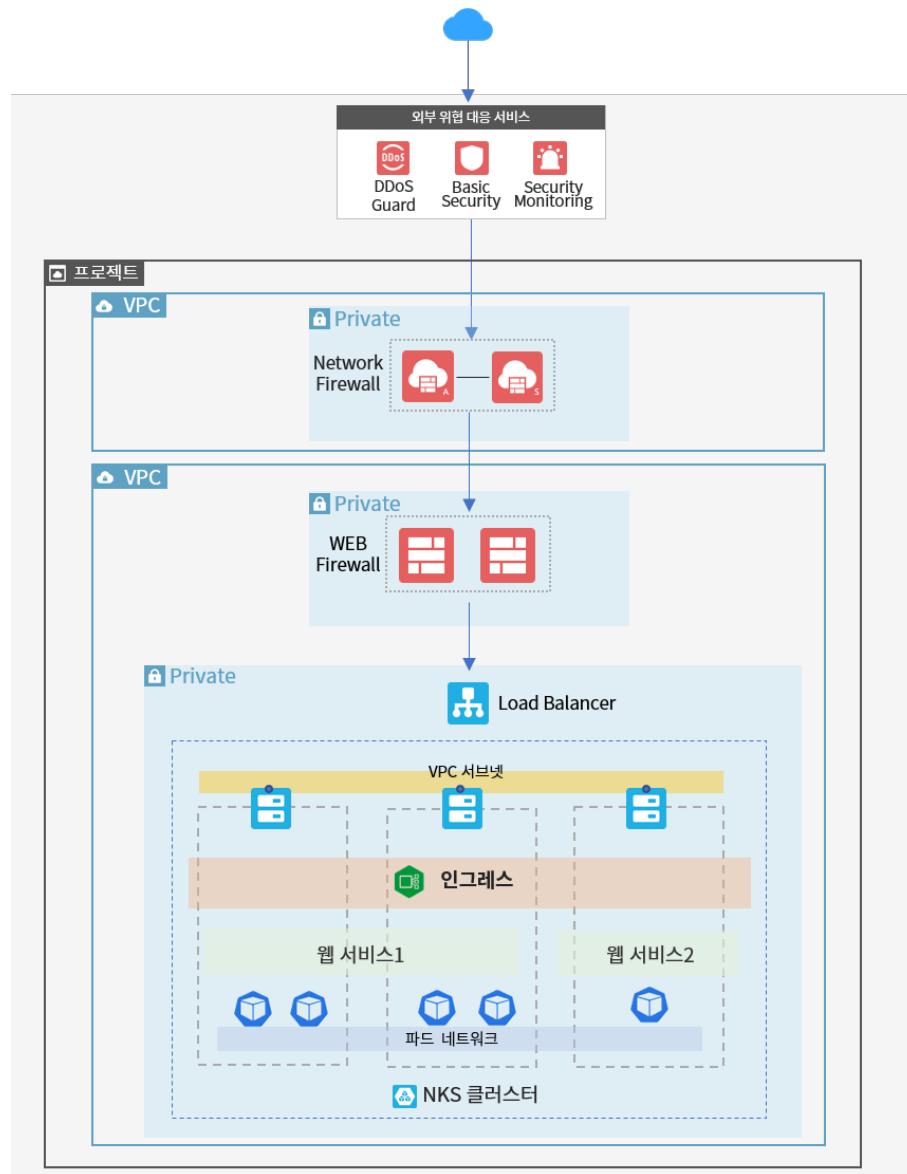


그림 52 네트워크 위협 대응 구성 예시

애플리케이션 및 컨테이너 런타임 위협 대응

Kubernetes에서 제공하는 여러 보안 기능과 NHN Cloud의 보안 서비스를 통해 안전한 컨테이너 서비스 환경을 구성할 수 있습니다. 추가로 다양한 공격 형태, 유형 등을 모두 탐지하고 모니터링하기 위해 컨테이너 보안을 위한 전용 솔루션을 이용하여 다양한 보안 위협으로부터 애플리케이션과 데이터를 보호하는 것이 필요할 수 있습니다.

대표적으로 CWPP(cloud workload protection) 및 CNAPP(cloud native application protection platform)로 분류되는 컨테이너 보안 솔루션이 있습니다. 클라우드 네이티브 애플리케이션 아키텍처의 복잡성과 동적인 특성으로 인해 기존 솔루션으로는 위협 대응에 어려움이 있을 수 있으며 이러한 클라우드 네이티브 보안 솔루션을 이용하여 컨테이너 런타임의 전반적인 보안 제어와 가시성 확보를 통해 보안을 강화하는 것이 필요합니다.

CWPP 솔루션은 컨테이너 관련 취약점 및 악성코드를 모니터링하거나 클러스터 내의 파드 간 통신을 제어하고 모니터링하여 컨테이너 워크로드에 대한 전체적인 보안 제어를 가능하게 하며 CNAPP는 개발에서부터 컨테이너 애플리케이션의 전반적인 보안의 통합 관리가 가능한 솔루션입니다.

이러한 3rd party 솔루션은 NHN Cloud의 마켓플레이스에서 상품으로 제공하고 있으며 상세한 정보는 [NHN Cloud 마켓플레이스](#)의 Security 카테고리에서 확인할 수 있습니다.

5 컨테이너 보안 아키텍처

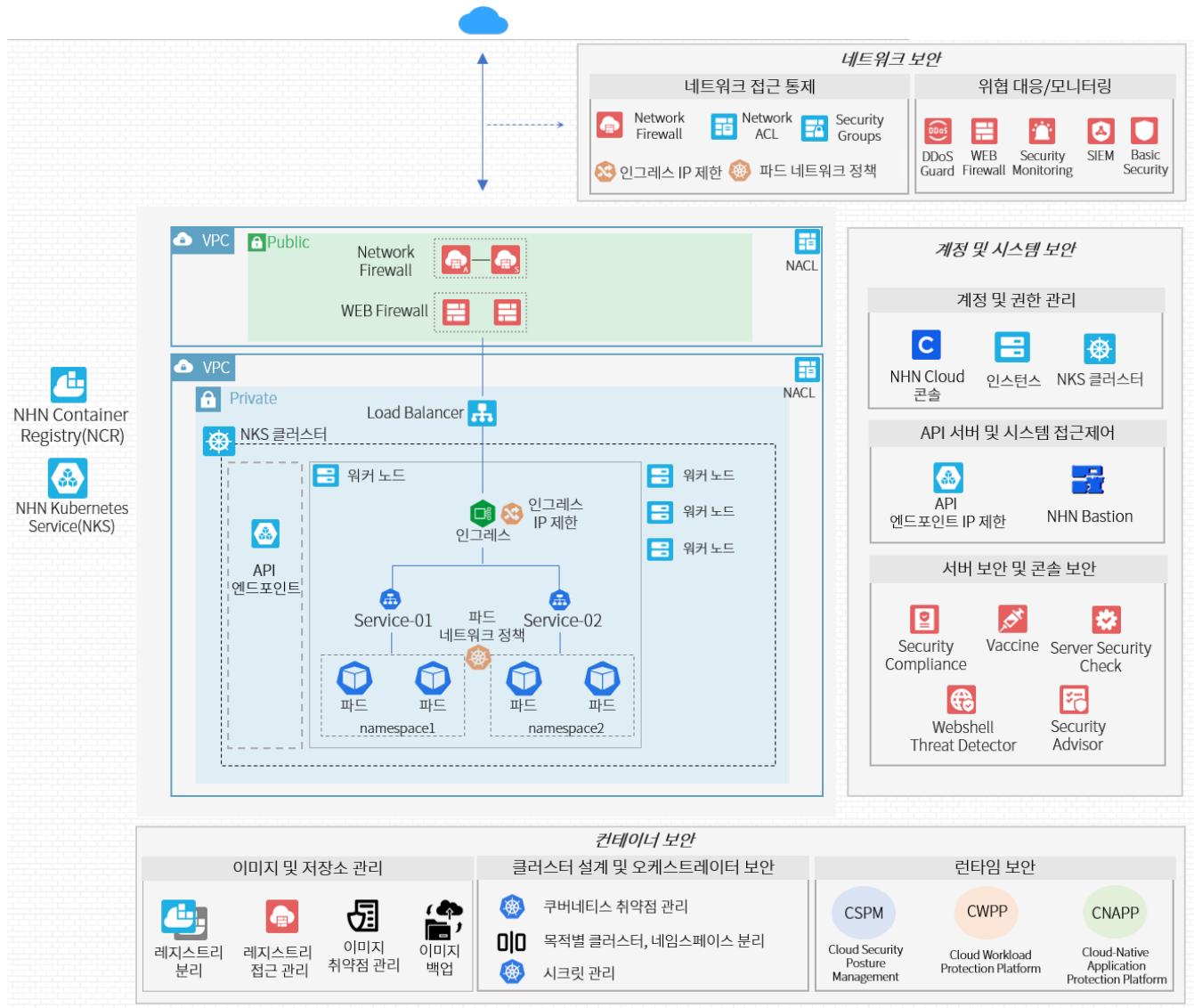


그림 53 컨테이너 보안 아키텍처

네트워크 보안

- 컨테이너 서비스에서도 인스턴스 기반 애플리케이션과 마찬가지로 네트워크 보안은 중요합니다. 외부 네트워크로부터의 잠재적 위협에 대응하기 위해 상시 모니터링과 IP 주소 기반 접근통제 정책을 수립해야 합니다. 특히 마이크로서비스 간 통신을 위하여 네트워크 세그먼트를 분리하고 액세스를 제어해야 합니다.
- 네트워크 접근통제는 NHN Cloud에서 제공하는 Network Firewall과 같은 보안 서비스를 이용하여 적용할 수 있으며 노드(인스턴스) 간의 제어는 Security Groups 등을 통해 구현할 수 있습니다. 파드나 서비스 간의 네트워크 접근제어는 파드 네트워크 정책이나 인그레스에서 IP 제한 설정을 적용하여 좀 더 세분화된 네트워크나 네임스페이스 통제 정책을 적용할 수 있습니다.

계정 및 시스템 보안

- 계정 관리는 정보 보안에 있어 핵심적인 요소입니다. 권한 있는 계정의 오남용을 방지하고, 최소 권한 원칙을 적용하여 내/외부의 위협으로부터 시스템과 애플리케이션을 보호할 수 있습니다.
- 계정은 크게 NHN Cloud 콘솔의 계정, 노드 인스턴스의 OS 계정, NKS 클러스터 관련 계정으로 나눌 수 있습니다. 각각의 환경에서 제공하는 계정과 인증 기능에 따라 필수 인력에게만 계정과 권한을 할당하고, 애플리케이션별 세분화된 계정 정책을 적용하는 것이 필요합니다. 또한 권한 분리, 계정 라이프사이클 관리, 정기적인 권한 검토 등의 계정 관리 프로세스를 통해 계정 관리 체계를 구축하는 것을 권장합니다.

컨테이너 보안

- 컨테이너 환경에서 안전한 서비스를 제공하기 위해서는 설계에서부터 컨테이너로 실행되는 애플리케이션 이미지 등의 관리와 Kubernetes와 같은 오케스트레이션 보안이 필수입니다.
- 서비스, 개발 목적에 따라 클러스터 또는 네임스페이스 등의 리소스 격리를 통해 장애나 보안 이슈로 인한 서로 간의 영향을 최소화할 수 있습니다. 예를 들어 개발 환경과 운영 환경의 클러스터를 분리하거나 웹과 DB의 네임스페이스를 분리하는 방법이 있습니다.
- 컨테이너로 실행되는 애플리케이션은 이미지 빌드 단계부터 운영 환경까지 전 라이프사이클에 걸쳐 다양한 보안 위협에 노출될 수 있습니다. 따라서 이미지 생성 시 안전한 기반 이미지 선택, 최신 업데이트 적용, 최소 권한 원칙 적용 등이 필요합니다. 또한 컨테이너 이미지 레지스트리, 배포, 실행 환경에 이르기까지 지속적인 취약점 평가와 모니터링이 필요합니다. 이를 통해 안전한 컨테이너 기반 애플리케이션 운영 환경을 구축할 수 있습니다.
- 컨테이너 환경에서는 오케스트레이션 도구 자체의 보안 취약점 관리도 필수적입니다. Kubernetes와 같은 오케스트레이션 플랫폼에 새로운 취약점이 발견되면 이를 적시에 업데이트해야 합니다. 그리고 시크릿과 같은 데이터의 관리 정책을 수립하고 RBAC, 감사 로그 등의 적절한 보안 통제를 적용해야 합니다.



엔에이치엔클라우드

13487 경기도 성남시 분당구 대왕판교로645번길 16 NHN 플레이뮤지엄

고객 센터: 1588-7967 | 이메일: support@nhncloud.com

©NHN Cloud Corp. All rights reserved.