Tìm hiểu về kiến trúc của Kubernetes

Trần Thùy Dương

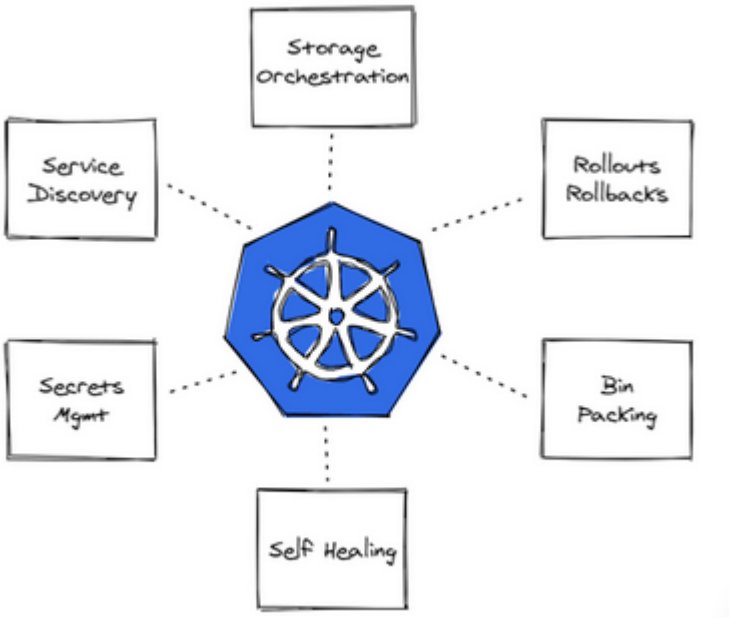
Đại học công nghệ, ĐHQGHN

Mục lục

I. Kubernetes (K8s) là gì

1. Khái niệm chung

Kubernetes (K8s) là một hệ thống mã nguồn mở được thiết kế để để quản lý và tự động hóa việc triển khai, mở rộng và quản lý các container. Được phát triển bởi Google và được chuyển giao cho Cloud Native Computing Foundation (CNCF), Kubernetes đã trở thành một trong những công cụ quản lý container phổ biến nhất trên thị trường hiện nay.



2. Lịch sử hình thành

- **2003:** Google bắt đầu sử dụng một hệ thống quản lý container gọi là Borg để quản lý hàng trăm ngàn container trên hạ tầng của mình.
- **2014:** Google công bố Kubernetes, dự án mã nguồn mở dựa Cannot connect to the Docker daemon
- trên nền tảng của Borg, trong một sự kiện có tên là "Container Management at Scale" tại San Francisco.
- **2015:** Google cùng với sự hỗ trợ từ các ông lớn công nghệ như Microsoft, Red Hat, IBM và Docker, đã chuyển giao Kubernetes cho CNCF.

- **2016:** Kubernetes v1.0 được phát hành, đánh dấu sự ổn định và được sử dụng rộng rãi của nền tảng.
- **2017 - hiện tại:** Kubernetes tiếp tục phát triển và trở thành một phần không thể thiếu trong cả những hạ tầng Cloud công cộng và riêng tư, với nhiều cải tiến và tính năng mới được thêm vào từ các phiên bản mới.

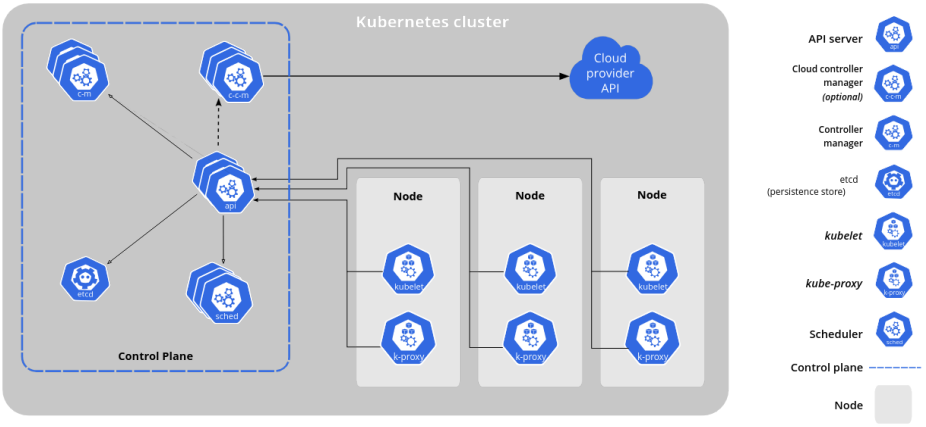
3. Công dụng

- **Continuous Delivery (Phát Triển Liên Tục):** Kubernetes cung cấp môi trường linh hoạt cho continuous delivery, cho phép các nhà phát triển triển khai và cập nhật các ứng dụng một cách nhanh chóng và tự động, giúp tăng tốc quá trình phát triển phần mềm.
- **Infrastructure Automation (Tự Động Hóa Hạ Tầng):** Kubernetes giúp tự động hóa việc triển khai, mở rộng và quản lý hạ tầng, giảm thiểu sự can thiệp thủ công và tối ưu hóa hiệu suất của hạ tầng IT.
- **Cloud Migration (Di Dời Lên Cloud):** Kubernetes cung cấp cách tiếp cận linh hoạt và hỗ trợ cho việc di dời các ứng dụng lên Cloud, giúp tận dụng các lợi ích của môi trường Cloud như tính mở rộng, tính sẵn sàng và tính linh hoạt.
- **Multi-cloud Deployment (Triển Khai Đa Đám Mây):** Kubernetes cho phép triển khai ứng dụng trên nhiều nền tảng Cloud khác nhau một cách dễ dàng và hiệu quả, giúp tối ưu hóa tính linh hoạt và sẵn sàng của hạ tầng.
- **Serverless Architecture (Kiến Trúc Không Cần Server):** Kubernetes hỗ trợ kiến trúc serverless thông qua việc triển khai và quản lý các container một cách tự động và linh hoạt, giúp giảm thiểu tài nguyên không cần thiết và tối ưu hóa chi phí vận hành.

II. Kiến trúc của K8S

1. Kubernetes Cluster

Để tìm hiểu sâu hơn về kiến trúc của Kubernetes, trước hết ta hãy nói về cấu trúc của một cụm Kubernetes (Kubernetes Cluster).



Khi thực hiện deploy trên Kubernetes, ta sẽ có một cluster. Một Kubernetes cluster là tập hợp của các máy worker hay còn gọi là nodes, dùng để chạy các container được đóng gói. Mỗi cluster sẽ có tối thiểu một nodes

Các worker nodes sẽ host các **Pods** dùng để khởi tạo ứng dụng. Control Plane sẽ quản lý các worker nodes và pods trong cluster. Trong môi trường Production, control plane thường được chạy qua nhiều máy tính và một cluster thường chạy nhiều nodes cung cấp khả năng chịu lỗi (fault tolerance) và tải cao (high availability)

a. Control Plane Components

Control Plane Components tạo ra chỉ định thực hiện đối với cluster như lên lịch và phản hồi tới các sự kiện của cluster.

Control Plane Components có thể chạy trên bất kỳ thiết bị nào trong cluster. Tuy nhiên để đơn giản hóa, lệnh cài đặt thường chạy toàn bộ control plane trên cùng một máy và không chạy container người dùng trên thiết bị đó

Control Plane Components bao gồm các phần có thể kể đến như: kube-apiserver, kube-scheduler, etcd, kube-controller-manager, cloud-controller-manager

- **kube-apiserver:** là một thành phần của Control Plane giúp expose các Kubernetes API, là phần frontend của Kubernetes control plane.
- **etcd:** là một cơ sở dữ liệu phân tán lưu trữ trạng thái của Kubernetes cluster, chịu trách nhiệm lưu trữ dữ liệu cấu hình, cung cấp các cặp key-value, đảm bảo tính nhất quán và khả dụng của dữ liệu.
- **kube-scheduler:** là một thành phần chịu trách nhiệm phân bổ pod đến các node trong cluster, là bộ điều khiển vòng lặp liên tục theo dõi các pod chưa được lên lịch và đưa ra quyết định nên đặt pod nào trên node nào. Nó có các chức năng chính là phân bổ pod, cân bằng tải và tối ưu hóa vị trí giữa các pod.
- **kube-controller-manager:** là thành phần điều khiển duy trì trạng thái mong muốn muốn của cluster. Thường các controller là các tiến trình riêng biệt, gồm một vài loại như node controller, job controller, EndpointSlice controller, ServiceAmount controller

- **cloud-controller-manager:** chịu trách nhiệm quản lý tương tác giữa các kubernetes cluster và các nhà cung cấp dịch vụ đám mây cơ bản. Nó cũng có các controller như node controller, service controller, …

b. Node Components

Node Components sẽ chạy trên mỗi node, đảm bảo việc chạy các pod và cung cấp môi trường Kubernetes runtime, bao gồm kubelet, kube-proxy, và container runtime.

- **kubelet:** là một tác nhân trên mỗi node trong Kubernetes cluster, chịu trách nhiệm quản lý vòng đời các pod trên node đó, theo dõi sức khỏe các pod, khởi chạy và quản lý container, thu thập thông tin các node. Kubelet có thể được cấu hình thông qua các cờ lệnh và tệp cấu hình để đáp ứng yêu cầu cụ thể và mức sử dụng tài nguyên của cluster.

- **kube-proxy:** là thành phần chịu trách nhiệm cân bằng tải lưu lượng tới các dịch vụ trong cluster. Nó đảm bảo rằng lưu lượng được phân phối đồng đều đến các pod của dịch vụ và hỗ trợ nhiều giao thức mạng.

- **container runtime:** chịu trách nhiệm tạo, chạy và quản lý vòng đời các container, được sử dụng để chạy các pod. Một số container runtime phổ biến có thể kể đến như docker, containerd, CRI-O, …

Một số khái niệm node quan trọng là:

- **Master nodes** chứa control plane (tầng điều khiển) của cluster Kubernetes, có thể bao gồm một hoặc nhiều node để đảm bảo tính khả dụng và khả năng mở rộng. Master nodes chịu trách nhiệm điều phối chính cho toàn bộ cluster và host các component:
    - etcd : lưu trữ thông tin của toàn bộ cluster
    - API server
    - Bộ lập lịch
    - Bộ quản lý controller (quản lý các controller Kubernetes)
    - Bộ quản lý controller đám mây (cung cấp logic của hạ tầng đám mây cho bộ quản lý controller)
    - Thường có các yêu cầu đặc biệt về mặt phần cứng, bởi tầm quan trọng của các node này đối với sự sống còn của toàn bộ hệ thống

- **Worker nodes** chịu trách nhiệm chạy các ứng dụng hoặc workload được triển khai trên cluster Kubernetes, cung cấp các tài nguyên tính toán như CPU, bộ nhớ, mạng, bộ lưu trữ cho việc triển khai các container

  Các worker node cấu thành data plane - nơi các container có thể chạy và kết nối vào mạng, thường bao gồm

    - kubelet : quản lý các node và đảm bảo các Pod cũng như các container bên trong chạy theo đúng yêu cầu của API server

- Một môi trường để chạy container: Docker, containerd,...

c. Workload, Pods

Workload là ứng dụng chạy bên trong Kubernetes. Bất kể khi chạy ứng dụng trong một hay nhiều component với nhau, trên Kubernetes chúng chạy trên cùng một tập pod. Trong K8s, Pod là biểu diễn các containers đang chạy trong cluster.

Pod là một thành phần nhỏ nhất mà bạn có thể tạo và quản lý trong Kubernetes. Một pod là một hay nhiều các container cần làm việc với nhau và chia sẻ tài nguyên bộ nhớ và mạng.

Thông thường, chúng ta sẽ không tạo Pod trực tiếp, thay vào đó, Workload sẽ phụ trách việc tạo ra các workload resources. Các workloads resources này dùng để quản lý tập các pod phía dưới. Chúng thực hiện cấu hình các controllers để đảm bảo các pods hoạt động và quản lý đúng cách.

Kubernetes cung cấp nhiều loại workloads có sẵn để có thể quản lý các tài nguyên như sau:

- **Deployment và Replicaset:** phù hợp để quản lý một ứng dụng không trạng thái (stateless application) workload trên cluster, nơi mà các pod trong Deployment có thể hoán đổi lẫn nhau và thay thế lúc cần thiết.
- **StatefulSet:** dùng để chạy một hay nhiều Pod cần theo dõi trạng thái. Ví dụ, nếu workloads thực hiện lưu data liên tục, bạn có thể chạy một StatefulSet ghép các pod với một PersistentVolume. Các chương trình chạy trong Pod có thể được thay thế với các Pod có cùng StatefulSet nhằm tăng khả năng phục hồi.
- **Job và CronJob:** cung cấp nhiều cách khác nhau để định nghĩa công việc và dừng. Bạn có thể dùng Job để định nghĩa một công việc để chạy hoàn thành và CronJob để chạy một công việc nhiều lần được lên lịch.

2. Kubernetes Networking

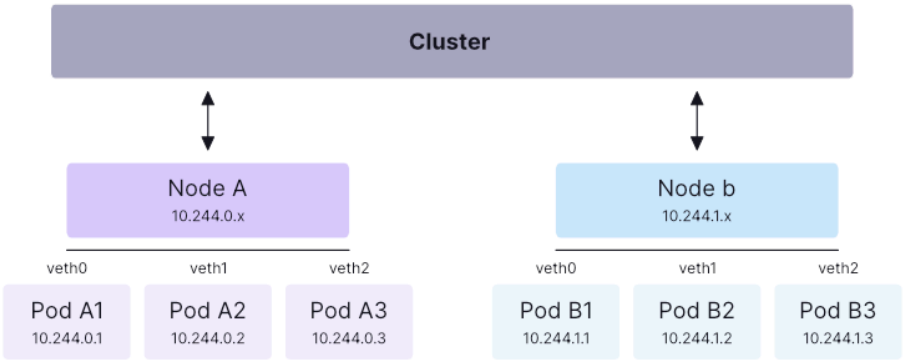
2.1.Định nghĩa

2.1.1. Khái niệm

K8S Networking là cơ chế để các thành phần bên trong và bên ngoài các cluster như Nodes, Pods, Services, và các traffic bên ngoài có thể giao tiếp được với nhau.

Do K8S là một hệ thống phân tán, nó sử dụng một lớp mạng ảo hóa (virtual overlay network) để cung tạo ra kiến trúc flat, cho phép các thiết bị trong mạng (nodes, host, endpoints) có thể kết nối với nhau trực tiếp mà không cần các lớp trung gian hay

mức định tuyến phức tạp. Dưới đây là ví dụ về một Kubernetes networking diagram:



2.1.2. Sự khác nhau giữa physical/VM networking và K8S networking

K8S networking vẫn áp dụng các nguyên tắc mạng cơ bản như trong mô hình mạng truyền thống. Tuy nhiên, trong môi trường mạng vật lý hoặc VM, ta thường phải cấu hình các thiết lập như địa chỉ IP, mở cổng firewall, cấu hình DNS thủ công cho mỗi thiết bị mới, trong khi đó, Kubernetes cung cấp tất cả các chức năng này một cách tự động.

Do đó, K8S networking sẽ giảm thiểu sự phức tạp và thời gian cần thiết để học và triển khai các giải pháp mạng mới. Lập trình viên và quản trị viên không cần phải hiểu cách mạng được triển khai, mà chỉ cần tập trung vào triển khai và quản lý tài nguyên trong Kubernetes.

2.1.3. Sự khác nhau giữa Docker Networking và K8S networking

Kubernetes sử dụng kiến trúc flat để phù hợp với kiến trúc phân tán. Tất cả các pods đều có thể giao tiếp với nhau dù chúng được triển khai trên các node vật lý khác nhau.

Mặc định Docker sử dụng mô hình mạng dựa trên bridge network để kết nối tất cả các container vào mạng của host. Ta cũng có thể tạo ra các mạng khác cho các containers sử dụng, bao gồm bridge, host (trực tiếp share network stack của host) và overlay (hệ thống phân tán thông qua Nodes yêu cầu môi trường Swarm)

Nhìn chung, Docker và Kubernetes networking cũng có các tính năng tương tự nhau, nhưng mỗi môi trường được tối ưu hóa cho mục đích sử dụng riêng. Docker tập trung vào mạng dựa trên một node duy nhất và sử dụng bridge network để đơn giản hóa mạng, trong khi K8S là một hệ thống phân tán sử dụng mạng overlay để hỗ trợ điều này.

2.2. Kiến trúc K8S Networking

2.2.1. Kiến trúc phẳng (Flat structure) của K8S

Trong Kubernetes, tất cả các Pods đều được gán địa chỉ IP riêng. Điều này giúp ta không cần tạo link giữa các Pods và không cần map container ports tới host ports.

Đồng thời, tất cả các pods trong cùng 1 nodes hoặc trên các nodes khác nhau có thể giao tiếp với nhau mà không cần sử dụng Network Address Translation (NAT).

Các Pods được gán namespace và interface riêng cung cấp môi trường mạng độc lập cho các container bên trong pods. Tất cả giao tiếp giữa các pods sẽ thông qua những interface được gán này.

Trong K8S, mỗi Node có một namespace riêng. Tuy nhiên để để cho phép các container và các service trong một cluster giao tiếp được với nhau cũng như với bên ngoài cluster, mạng ở mức cluster phải ánh xạ các Node-level namespace này để đảm bảo traffic được định tuyến chính xác qua các node.

2.2.2. Cơ chế cấp phát địa chỉ Pod IP

Kubernetes allocates địa chỉ IP tới các pods sử dụng hệ thống Classless Inter-Domain Routing (CIDR). Mỗi Pod trong Kubernetes sẽ được gán một địa chỉ IP từ range của CIDR đã chỉ định cho cluster .

Ngoài ra, nhiều plugin mạng trong K8S cũng hỗ trợ các chức năng quản lí địa chỉ IP (IP address Management -IPAM) cho phép gán thủ công các địa chỉ IP, prefix và pool.

2.2.3. Cách thức hoạt động của DNS trong Kubernetes cluster

K8S clusters hỗ trợ dịch vụ DNS, trong đó CoreDNS là một trong những dịch vụ cung cấp DNS phổ biến nhất. Khi triển khai các ứng dụng và Service trên K8S, K8S tự động gán các tên DNS cho Pods và Service theo định dạng sau:

- Pod:pod-ip-address.pod-namespace-name.pod.cluster-domain.example (ví dụ: 10.244.0.1.my-app.svc.cluster.local)
- Service:service-name.service-namespace-name.svc.cluster-domain.example (ví dụ: database.my-app.svc.cluster.local)

Cơ chế này cho phép các ứng dụng và dịch vụ trong Kubernetes tương tác với nhau thông qua tên DNS thay vì sử dụng các địa chỉ IP trực tiếp. Việc sử dụng DNS là lựa chọn phổ biến hơn vì dễ nhớ và ổn định, trong khi địa chỉ IP của một Service có thể thay đổi nếu Service đó được xóa và thay thế.

2.2.4. Isolate K8S networking bằng Network Policies

Kubernetes mặc định cho phép tất các các pods có thể tương tác được với nhau, điều này có thể gây ra các rủi ro về an toàn thông tin.

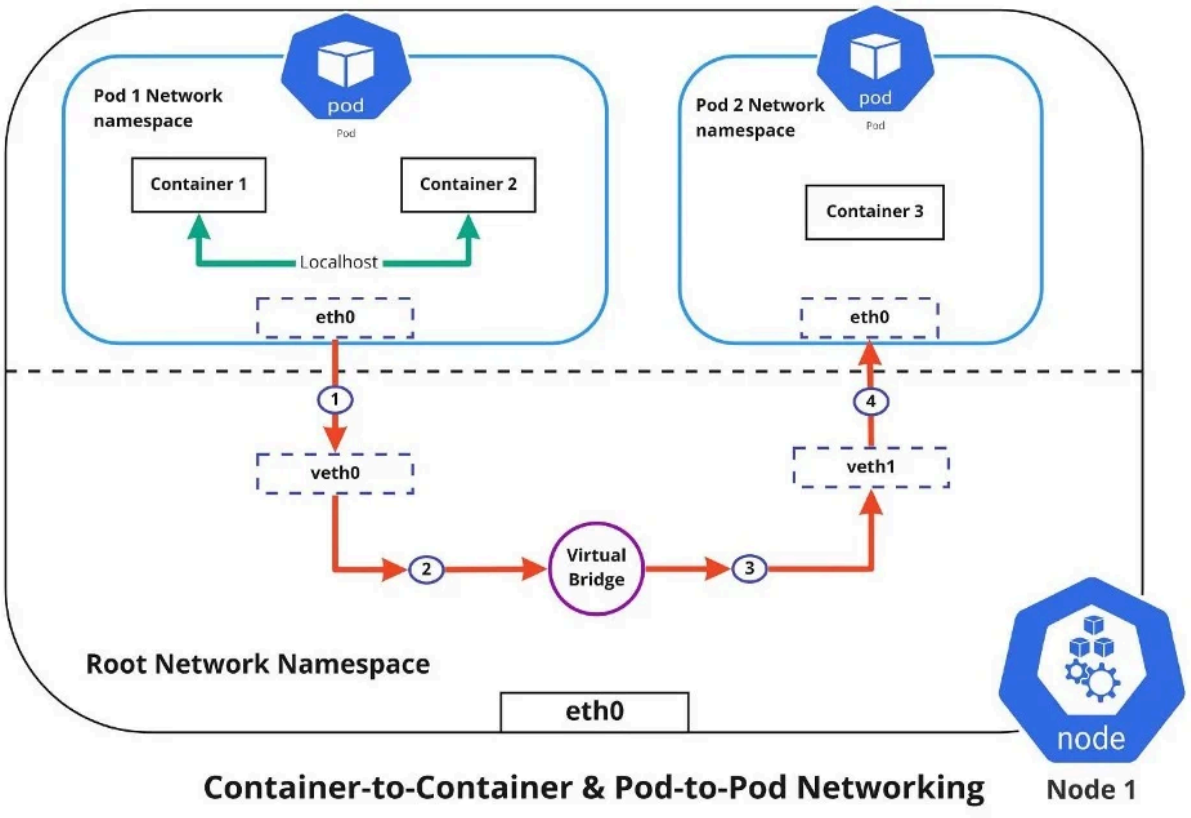
K8S Network Policy là các API object cho phép xác định các routes được phép ingress và egress tới pods.

Ví dụ dưới đây xác định một policy block các trafic tới Pods được gán nhãn app-component: databases, trừ khi origin Pod được gán nhãn app-component: api.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: database-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
```

2.3. Các loại Kubernetes Networking

2.3.1. Container to container networking



Container-to-container network được xử lý thông qua Pod network namespace. Network namespace là cơ chế cho phép tạo ra các network interface và bảng định tuyến riêng tách biệt với phần còn lại của hệ thống.

Mỗi pod có một namespace riêng và các Container bên trong pod đó chia sẻ cùng một địa chỉ IP. Tất cả các tương tác giữa các container sẽ xảy ra thông qua localhost bởi vì chúng đều là một phần của cùng namespace (Thể hiện bởi đường màu xanh ở hình trên).

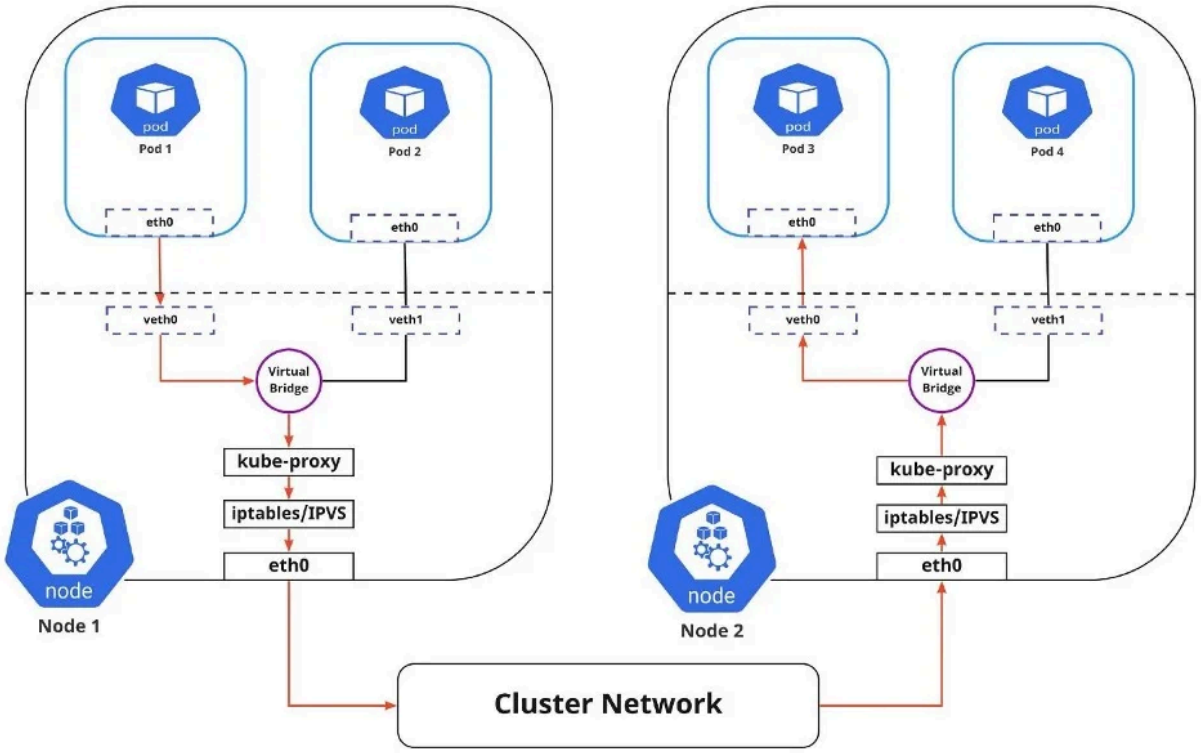
2.3.2. Pod-To-Pod networking

Trong K8S, mỗi node đều có CIDR range được gán cho IP của các pods. Điều này đảm bảo mỗi pod nhận một địa chỉ IP duy nhất trong cluster đó. Khi một pod mới được tạo, địa chỉ IP sẽ không bao giờ bao giờ bị trùng. Không giống như container-to-container networking, việc giao tiếp giữa Pod với Pod sử dụng địa chỉ IP thật, dù 2 Pod này nằm cùng trong 1 node hay 2 node khác nhau trong cluster.

Mô hình trên chỉ ra rằng để các pods có thể giao tiếp được với nhau thì traffic phải đi giữa Pod network namespace và Root network namespace (Mặc định thì Linux sẽ gán mọi process tới root network namespace). Kết nối giữa Pod network namespace và Root network namespace được thực hiện bởi virtual ethernet device hoặc một

veth pair (veth0 to Pod namespace 1 và veth1 to Pod namespace 2 trong biểu đồ trên). Một virtual network bridge kết nối giữa các virtual interface này cho phép traffic chạy giữa chúng sử dụng giao thức Address Resolution Protocol (ARP).

2.3.3. Pod-To-Service networking



Các pods trong K8S có thể scale up hoặc scale down rất linh hoạt. Chúng có thể được tạo lại nếu ứng dụng bị crash hoặc một node xảy ra sự cố. Điều này có thể dẫn đến thay đổi địa chỉ IP của Pod, đây là một trong những khó khắn của K8S networking.

Kubernetes đã giải quyết vấn đề trên bằng cách sử dụng Service function. K8S Service quản lý trạng thái của một tập hợp các pods, cho phép theo dõi thay đổi địa chỉ IP các pods. Các Services được gán một single virtual IP cho một nhóm các Pod IP. Bất kì traffic nào tới virtual IP của Service sẽ được định tuyến tới các Pods đã được gán với virtual IP đó. Điều này cho phép tập các Pods được gán với một Service thay đổi bất kỳ lúc nào thì client chỉ cần biết virtual IP của Service.

Khi tạo một K8S Service, một virtual IP mới hay còn gọi là cluster IP sẽ được tạo cùng. Các traffic tới virtual IP sẽ được load-balanced tới tập các Pods phía sau được gán với service đó.

Để có thể thực hiện load balancing bên trong cluster, Kubernetes dựa trên framework networking được xây dựng bên trong Linux là netfiler. Netfilter là một framework cung cấp bởi Linux cho phép tùy chỉnh các function liên quan đến network. Netfilter cung cấp các function cho Network address Translation, port translation, cũng như cung cấp khả năng chặn các gói tin đến các địa chỉ cụ thể.

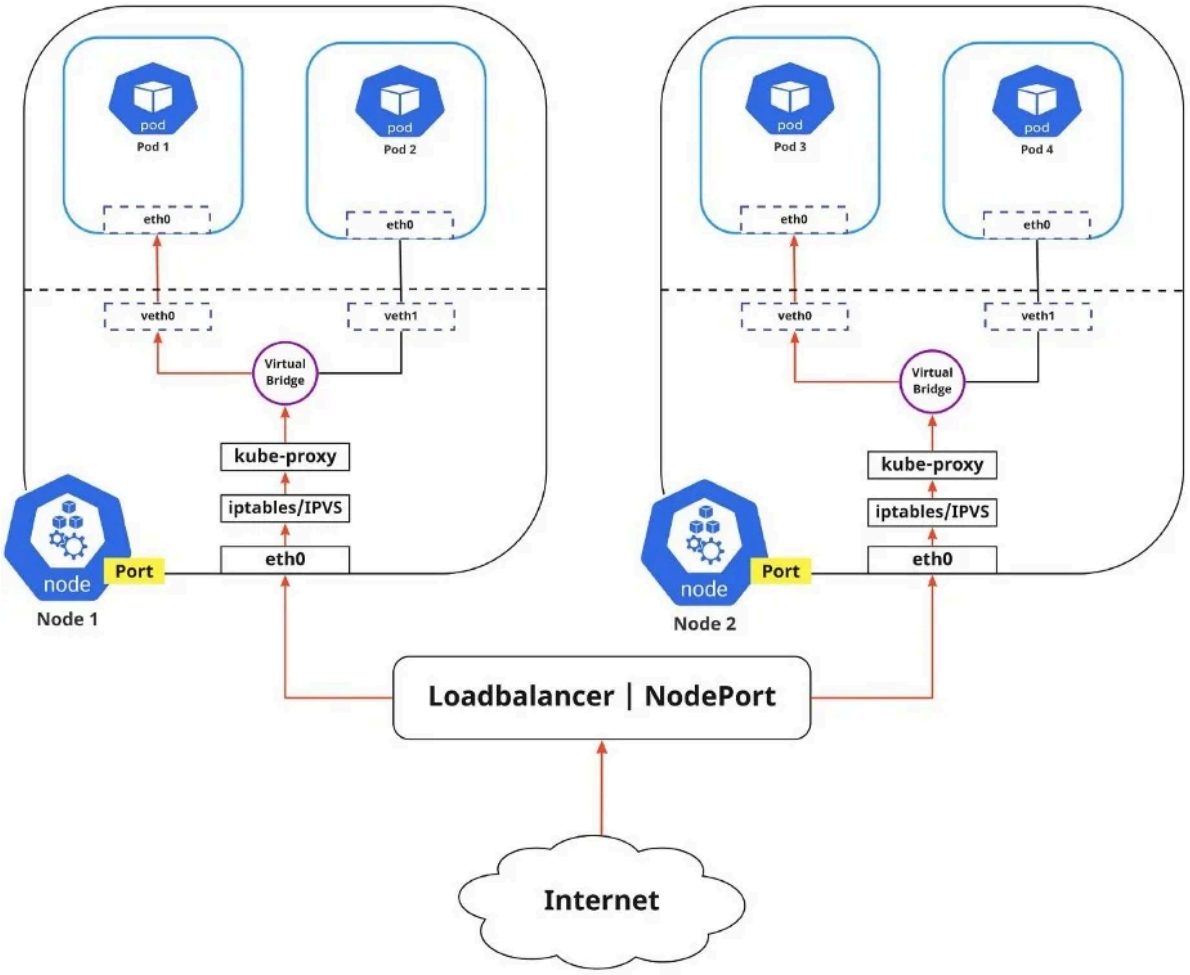
Cân bằng tải trong cluster có thể triển khai theo 2 cách base trên Netfilter:

- **IPTABLE:** là một chương trình user-space cung cấp một bảng xác định các rule cho việc transforming packets sử dụng framework Netfilter. Trong Kubernetes, iptables rules được cấu hình bởi kube-proxy controller theo dõi K8S API Server để xem xét các thay đổi. Khi Service có sự thay đổi hoặc cập nhật virtual IP của Service hoặc IP của một pod, những traffic có đích là virtual IP sẽ được iptable rules thay đổi đích thành một IP pods ngẫu nhiễn từ tập các Pods có sẵn hoặc tập Pods được chọn. Khi Pods up hoặc down, iptables rules cũng sẽ cập nhật và phản hồi sự thay đổi trạng thái này của cluster.

- **IPVS:** Từ phiên bản 1.11 Kubernetes cung cấp một option thứ hai cho load-balancing là IP Virtual Server (IPVS). Nó cũng được xây dựng dựa trên netfilter và triển khai transport-layer load balancing tương tự như của Linux kernel. IPVS được tích hợp với LVS (Linux Virtual Server), chạy một host và thực hiện như một LB phía trước một cluster của server thật.

Mô hình trên chỉ ra traffic flow sẽ đi từ Pod 1 tới Pod 3 thông qua một Service tới một node khác (được đánh dấu màu đỏ). Gói tin sẽ được di chuyển tới virtual bridge và tới default route (eth0) bởi vì ARP chạy trên bridge sẽ không hiểu Service. Sau đó package sẽ được filter bởi iptables, sử dụng các rules đã được xác định trong node bởi kube-proxy.

2.3.4. External-to-Service networking



Các phần trước đã tìm hiểu về flow của traffic bên trong một cluster. Tuy nhiên, vẫn còn một khía cạnh khác của K8S networking là expose một ứng dụng ra môi trường bên ngoài.

Có 2 cách để expose một ứng dụng ra external network:

- **Egress:** Sử dụng tính năng này khi muốn định tuyến lưu lượng truy cập từ Kubernetes Service ra Internet. Trong trường hợp này, iptables thực hiện NAT nguồn, do đó lưu lượng truy cập dường như đến từ Node chứ không phải Pod.
- **Ingress:** Đây là lưu lượng traffic đi vào từ external world tới Service. Ingress cũng cho phép block một số kết nối cụ thể sử dụng các rule cho các kết nối.

III. Kết luận

Trong bài nghiên cứu này, chúng ta đã thực hiện một số nghiên cứu về Kubernetes (K8s) và Kubernetes networking.

Tổng kết lại, Kubernetes là một công cụ mạnh mẽ cho việc triển khai và quản lý các ứng dụng container. Tuy nhiên, việc triển khai và cấu hình có thể đòi hỏi một số kiến

thức về hệ thống và mạng lưới. Điều này cũng tạo ra cơ hội để nghiên cứu và phát triển thêm các ứng dụng và giải pháp phù hợp với nhu cầu của từng tổ chức.

Với các công nghệ liên quan ngày càng phát triển, Kubernetes cung cấp một nền tảng vững chắc để các nhà phát triển và quản trị viên có thể triển khai, mở rộng và quản lý các ứng dụng một cách hiệu quả trên môi trường đám mây.