

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG



CÔNG NGHỆ DEVOPS VÀ ỨNG DỤNG
MICROSERVICE DEPLOY ON RKE2, ESXI, CI/CD WITH HA ARCHITECTURE

21522151
21520813

Nguyễn Đoàn Khắc Huy
Lê Ngọc Hân

TP. HỒ CHÍ MINH, 2024

Mục lục

Chương 1.	Mở đầu	7
1.1.	BỐI CẢNH	7
1.2.	Mục tiêu của đề tài	7
Chương 2.	Cơ sở lý thuyết	8
2.1.	Kiến trúc Microservices	8
2.1.1.	Kiến trúc một khối (Monolithic architecture)	8
2.1.2.	Kiến trúc dịch vụ nhỏ (Microservice architecture)	8
2.2.	Sonarqube	9
2.3.	Kubernetes	10
2.3.1.	Giới thiệu	10
2.3.2.	Kiến trúc thành phần chính	10
2.3.3.	Pod	11
2.4.	ArgoCD	11
2.4.1.	Tổng quan	11
2.4.2.	Kiến trúc	12
2.4.3.	Cơ chế hoạt động	13
2.5.	Rancher	14
2.5.1.	Thành phần chính	14
2.5.2.	Chức năng chính	14
2.6.	RKE2	15
2.7.	Prometheus	16
2.7.1.	Kiến trúc	17
2.7.2.	Vai trò của Prometheus trong quản lý K8s	17
2.8.	Grafana	18
2.9.	Gitlab	19
2.10.	Vmware esxi	19
2.11.	Metalb	21
2.12.	Haproxy và Keepalived	22
Chương 3.	Triển khai hệ thống	24
3.1.	Sơ đồ hệ thống	24
3.2.	Sourcecode microservice	26
3.2.1.	Tổng quan application	26
3.2.2.	Application repository	26
3.3.	Deployment Repository	26
3.4.	Jenkins	27
3.5.	Private Registry	29

3.6.	Argo CD và Kubernetes Cluster.....	30
3.7.	SonarQube.....	30
3.8.	Prometheus và Grafana.....	31
3.9.	Triển khai các service trên kubernetes.....	32
3.9.1.	Triển khai mô hình cluster.....	32
3.9.2.	Triển khai ứng dụng lên kubernetes	36

DANH MỤC HÌNH

Figure 2.1.1: Kiến trúc monolithic	8
Figure 2.1.2: Kiến trúc microservice	8
Figure 2.2.1: Sonarqube overview	9
Figure 2.4.1: Kiến trúc của Argo CD	12
Figure 2.4.2: Luồng hoạt động của ArgoCD	13
Figure 2.5.1: Logo Rancher	14
Figure 2.6.1: Logo RKE2	15
Figure 2.6.2: Kiến trúc RKE2	15
Figure 2.7.1: Logo Prometheus.	16
Figure 2.7.2: Kiến trúc Prometheus.	17
Figure 2.8.1: Logo Grafana.	18
Figure 2.9.1: Logo GitLab.	19
Figure 2.10.1 Mô hình cách hoạt động của MetaLB	21
Figure 2.11.1: Mô hình triển khai HA cho Kubernetes	22
Figure 3.1.1: Sơ đồ hệ thống Kubernetes	24
Figure 3.1.2: Mô hình triển khai ESXi và vCenter	25
Figure 3.1.3: Sơ đồ hệ thống	25
Figure 3.2.1: Repo chứa tất cả dự án microservice.	26
Figure 3.4.1: Jenkins server	27
Figure 3.4.2: Jenkinsfile của Testcase1_pipeline.1	28
Figure 3.4.3: Jenkinsfile của Testcase1_pipeline.2	28
Figure 3.4.4: Jenkinsfile của update_manifesh_repo.1	29
Figure 3.4.5 : Jenkinsfile của update_manifesh_repo.2	29
Figure 3.5.1 Registry Harbor.	30
Figure 3.6.1: ArgoCD của đồ án	30
Figure 3.7.1: Kết quả khi kích hoạt sonarQube từ pipeline	31
Figure 3.8.1: Giao diện quản lý tổng cluster.	31
Figure 3.8.2: Giao diện quản lý network của mod-edges.	32
Figure 3.9.1: Số lượng máy ảo sử dụng	35
Figure 3.9.2: Thông số của các máy ảo	35
Figure 3.9.3: Các kubernetes nodes sau khi triển khai.	36
Figure 3.9.4: Rancher dashboard.	36
Figure 3.9.5: Số lượng pod mà đồ án sử dụng	37
Figure 3.9.6: Giao diện web	37

DANH MỤC TỪ VIẾT TẮT

Kí hiệu chữ viết tắt	Chữ viết đầy đủ
API	Application programming interface
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
XML	Extensible Markup Language
ORM	Object-Relational Mapping
YOLO	You Only Look Once
CI/CD	Continuous Integration/ Continuous Deployment
CI/CD	Continuous Integration/ Continuous Delivery

DANH MỤC CÁC THUẬT NGỮ

Tên thuật ngữ	Ý nghĩa
Business logic	Logic nghiệp vụ: là việc chuyển đổi các quy tắc doanh nghiệp thành các đoạn code trong ứng dụng, nó mô tả việc data sẽ được xử lý như thế nào.
Application server	Là Server cung cấp các tài nguyên và dịch vụ cần thiết để chạy ứng dụng
Decomposing	Là quá trình tách các thành phần của ứng dụng có kiến trúc một khối thành các ứng dụng có dịch vụ nhỏ
Unbundling	Là quá trình đóng gói các mỗi các dịch vụ nhỏ bao gồm mã nguồn, các file cấu hình, database thành một đơn vị nguyên khối (self-contained unit)
Production	Là môi trường cho người dùng cuối cùng. Môi trường này cần đảm bảo không được lỗi và tối ưu để đưa đến cho người dùng
Gitlab Runner	Là “lính” đứng lắng nghe lệnh từ Gitlab server để chạy các công việc. Gitlab runner là một phần mềm có thể cài trên bất kì máy nào.
Gitlab Executor	Là chế độ đứng ra thực hiện công việc, tùy thuộc vào quy trình sẽ sử dụng các chế độ khác nhau.
Route	Điều hướng request đến nơi khác.

Chương 1. Mở đầu

1.1. BỐI CẢNH

Trong thời đại hiện tại, sự phát triển nhanh chóng của công nghệ, phần cứng và phần mềm đã thúc đẩy sự phát triển của các ứng dụng thông minh dựa trên tính toán cận biên (edge computing). Để đáp ứng nhu cầu thiết yếu của con người, nhóm của chúng tôi đã quyết định xây dựng một hệ thống cận biên thông minh, tích hợp công nghệ trí tuệ nhân tạo vào quá trình nhận diện. Hệ thống này được thiết kế dựa trên kiến trúc microservices, giúp giải quyết các vấn đề liên quan đến việc sử dụng tài nguyên nhiều hơn trên điện toán đám mây, đồng thời giảm chi phí hoạt động và đảm bảo thời gian thực. Đồng thời, chúng em sẽ tích hợp quy trình triển khai CI/CD tự động tích hợp với Kubernetes (K8s) để đảm bảo việc triển khai và quản lý ứng dụng một cách hiệu quả.

1.2. MỤC TIÊU CỦA ĐỀ TÀI

- Áp dụng kiến thức đã học vào đề tài nhóm thực hiện.
- Xây dựng quy trình Devops CICI Pipeline và Kubernetes.
- Có hệ thống thông báo, logging, đánh giá khi phát hiện bất thường
- Đáp ứng cho nhu cầu doanh nghiệp
- Tối ưu hóa chi phí vận hành
- Tính linh hoạt
- Quản lý và giám sát
- Độ tin cậy và tính sẵn sàng cao
- Hiệu suất và khả năng mở rộng

Chương 2. Cơ sở lý thuyết

2.1. KIẾN TRÚC MICROSERVICES

2.1.1. Kiến trúc một khối (Monolithic architecture)

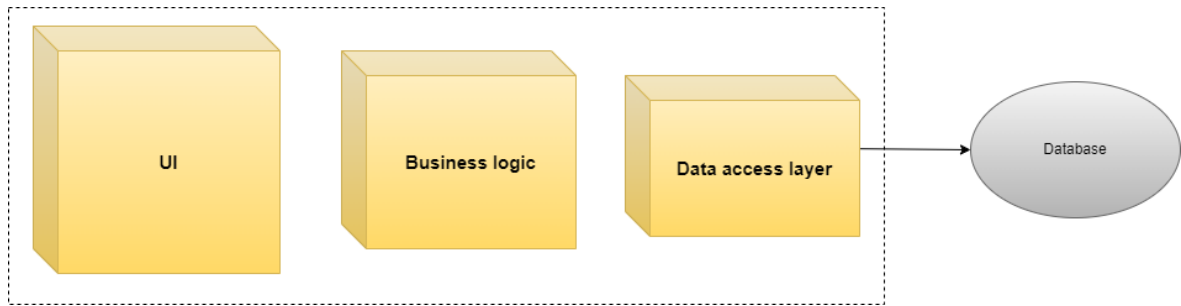


Figure 2.1.1: Kiến trúc monolithic

Kiến trúc một khối là dạng phát triển ứng dụng, và khi triển khai lên thì nó ở 1 dạng khối duy nhất. Tất cả các UI, các logic nghiệp vụ (business logic), các logic để truy cập database sẽ được đóng gói thành 1 ứng dụng duy nhất và sẽ được triển khai trên máy chủ ứng dụng (Application server) [2].

2.1.2. Kiến trúc dịch vụ nhỏ (Microservice architecture)

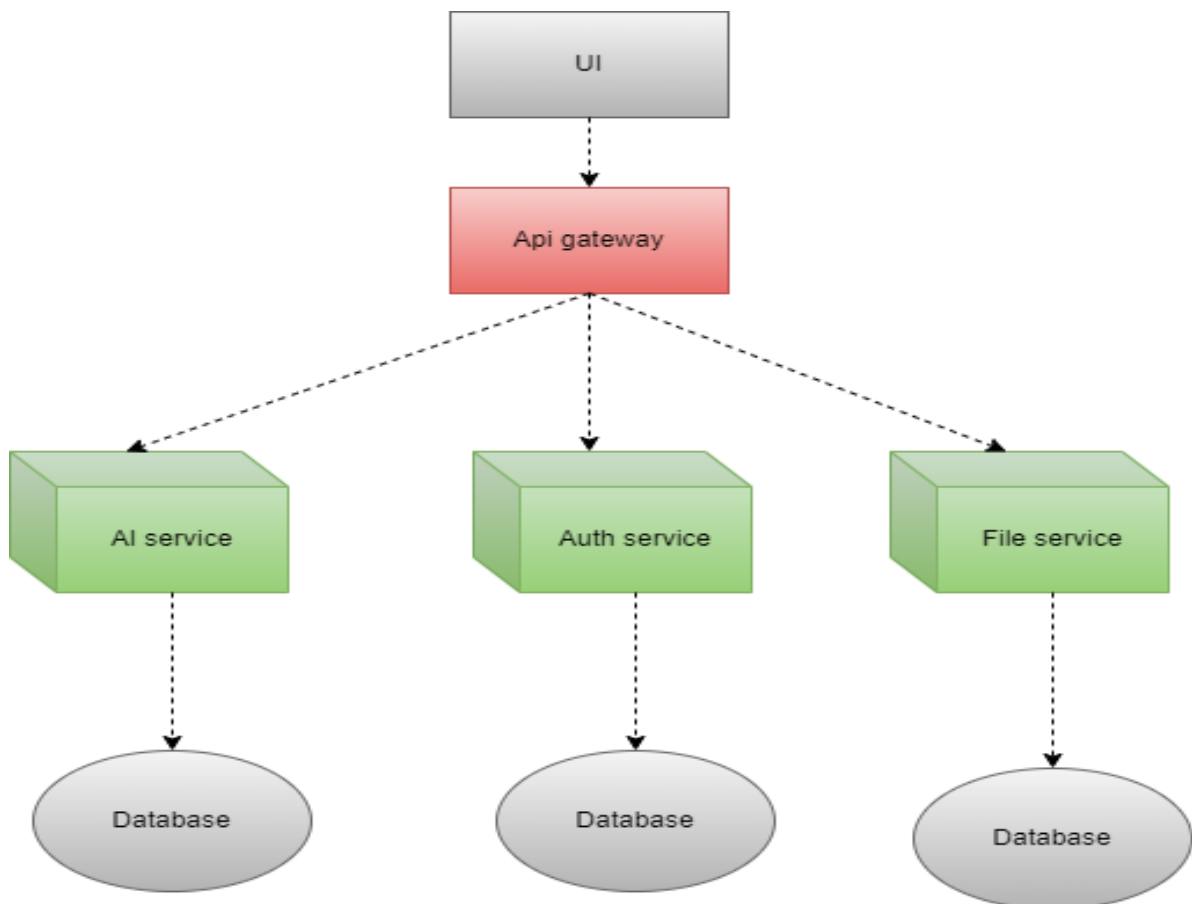


Figure 2.1.2: Kiến trúc microservice

Kiến trúc microservices là một phương pháp thiết kế phần mềm tiên tiến, hướng đến việc phân chia ứng dụng một khối (Monolithic) thành các dịch vụ nhỏ (microservice), độc lập và tự chủ. Mỗi dịch vụ được thiết kế để thực hiện một chức năng cụ thể, sở hữu logic nghiệp vụ riêng biệt và được phát triển, triển khai, vận hành độc lập với các dịch vụ khác [3].

Khi viết ứng dụng microservice là quá trình phân tách (decomposing) và đóng gói (unbundling). Đầu tiên là phân tách các thành phần của ứng dụng có kiến trúc một khối thành các ứng dụng microservice. Thứ 2 là đóng gói các mỗi các microservices bao gồm mã nguồn, các file cấu hình, database thành một đơn vị nguyên khối (self-contained unit) [3].

Kiến trúc microservice giúp xây dựng hệ thống với các tính chất:

- Linh hoạt: Dịch vụ nhỏ, dễ thay đổi, triển khai nhanh, hỗ trợ tích hợp và mở rộng chức năng
- Đàn hồi: Khắc phục sự cố cục bộ, cô lập lỗi, giảm thiểu ảnh hưởng đến toàn bộ hệ thống, đảm bảo hoạt động liên tục.
- Khả năng mở rộng: Phân phối dịch vụ theo chiều ngang, dễ dàng mở rộng hiệu năng và tính năng.

2.2. SONARQUBE

Sonarqube là công cụ kiểm tra chất lượng mã nguồn trong môi trường phát triển phần mềm, Sonarqube hỗ trợ nhiều loại ngôn ngữ, có thể quét và phát hiện các lỗ hổng, bugs lỗi logic và đưa báo cáo đến cho người dùng. Công cụ này có thể cấu hình để tích hợp vào quy trình CI của dự án, giúp đảm bảo code chất lượng nhất trước khi đưa đến cho người dùng cuối.

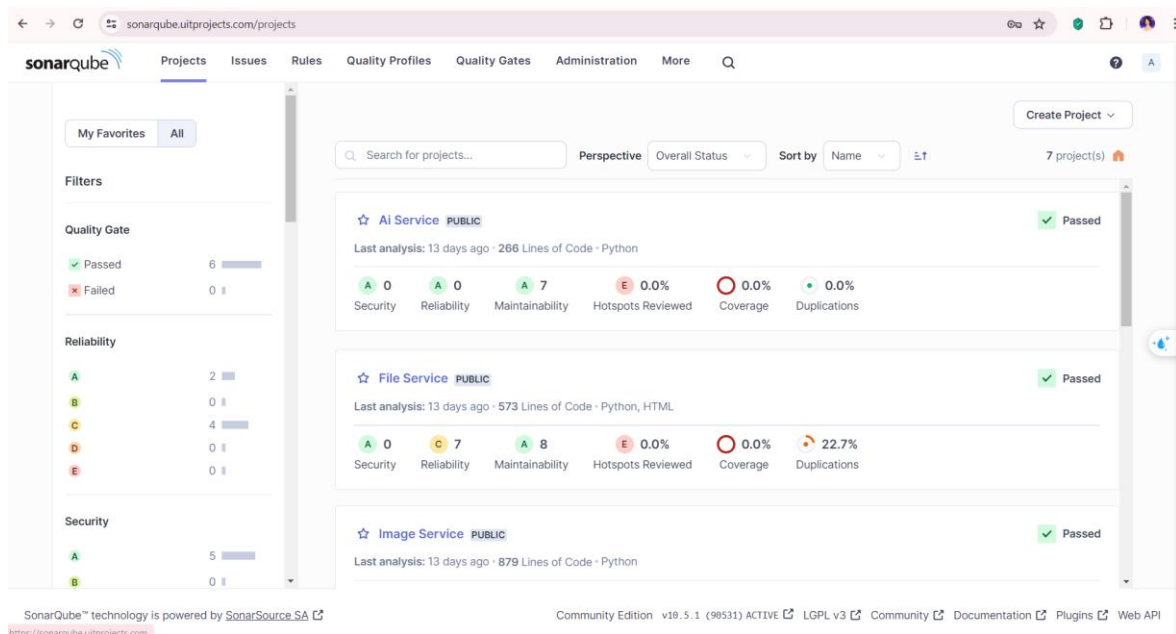


Figure 2.2.1: Sonarqube overview

2.3. KUBERNETES

2.3.1. Giới thiệu

Kubernetes là một hệ thống orchestration container mã nguồn mở để tự động hóa việc triển khai, mở rộng và quản lý phần mềm. Được thiết kế ban đầu bởi Google, dự án hiện do một cộng đồng những người đóng góp trên toàn thế giới bảo trì và thương hiệu được nắm giữ bởi Cloud Native Computing Foundation.

2.3.2. Kiến trúc thành phần chính

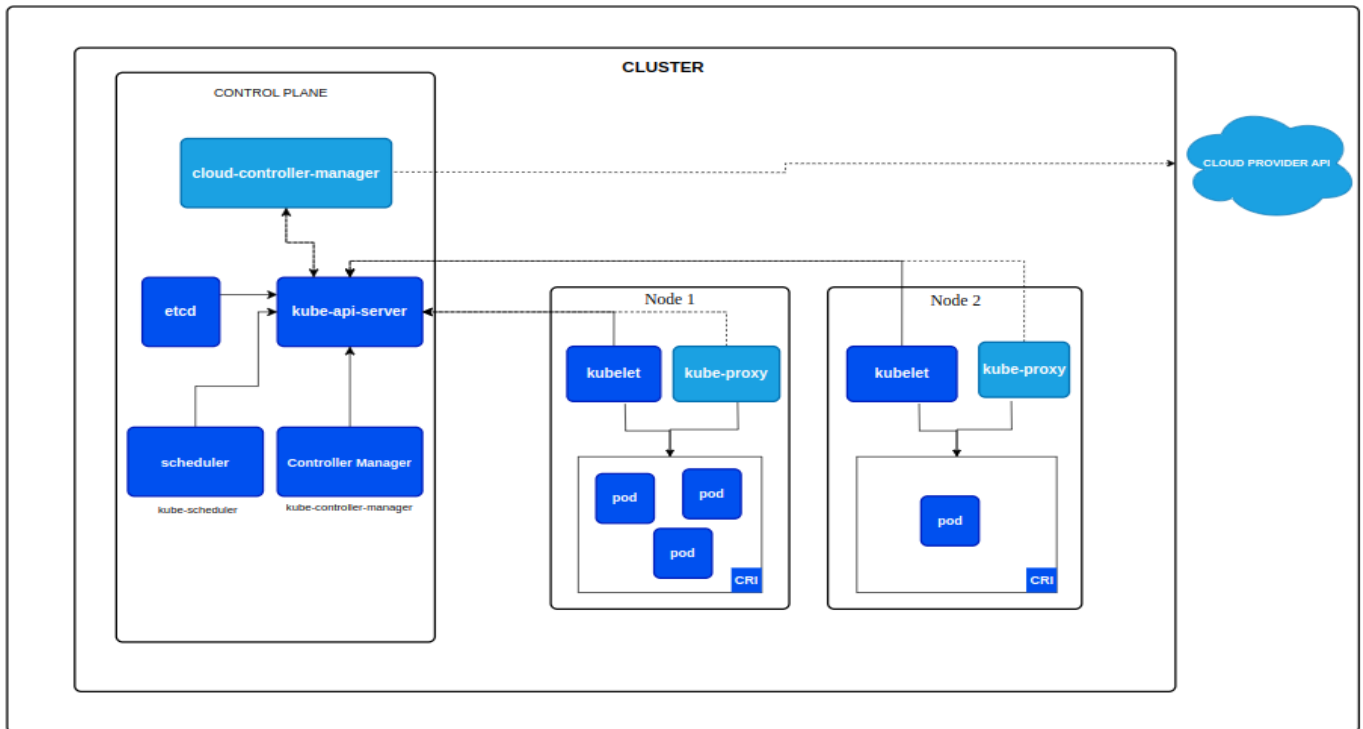


Figure 2.3.: Kiến trúc Kubernetes

2.3.2.1. Control plane

- **kube-api-server:** là interface chính, đại diện cho control plane và cluster.
- **Etcd:** là interface chính, đại diện cho control plane và cluster.
- **kube-scheduler:** đảm nhận việc scheduling (tiến trình mà chọn một node trong cluster để chạy container).
- **kube-controller-manager:** có trách nhiệm quản lý các utility (tiện ích) trên k8s.
- **cloud-controller-manager(option):** cung cấp interface giữa k8s và các mô hình cloud (AWS, Azure, Google Cloud).

2.3.2.2. Node

- Node là các máy đang chạy hay đó là nơi chứa các container được quản lý bởi control plane.

- **Kubelet:** có nhiệm vụ liên lạc với control plane và chắc chắn rằng container đang được chạy trên node mà control plane chỉ định.
- **container runtime:** là một phần mềm chạy container, không phải là một phần của k8s, nhưng để chạy container trên mỗi node thì k8s cần có container runtime.
- **kube-proxy:** đảm nhận nhiệm vụ liên quan đến cung cấp mạng giữa các container và services trong cluster.

2.3.3. Pod

- Pod là đơn vị nhỏ nhất trong Kubernetes, 1 Pod có thể chứa 1 hay nhiều container hoạt động đồng thời. Tuỳ thuộc vào tính chất dự án mà có thể chọn 1 pod chạy với nhiều container hoặc một container. Thường là sẽ là một do dễ quản lí và kiểm soát phần logging dễ hơn khi 1 container bị down. [7]
- Các container trong cùng một pod sẽ share cùng một tài nguyên, cùng môi trường, cùng địa chỉ IP. Có thể cấu hình để giới hạn tài nguyên sử dụng của pod. [7]
- Container: là các image đã được push lên Dockerhub hoặc private registry.
- Về việc triển khai các pod này lên node nào sẽ được tự động chọn bởi thành phần Schedule, Pod có thể ở node 1 khi khởi tạo, nhưng có thể bị phá huỷ và triển khai tự động lại trên Node 2 [8].

2.4. ARGOCD

2.4.1. Tổng quan

- ArgoCD là một mã nguồn mở container-native workflow engine phục vụ việc deploy service trên Kubernetes.
- ArgoCD được triển khai trên Kubernetes như một Kubernetes CRD (Custom Resource Definition)
- ArgoCD là một công cụ dễ sử dụng cho phép các nhóm phát triển triển khai và quản lý các ứng dụng mà không cần phải tìm hiểu nhiều về Kubernetes và không cần toàn quyền truy cập vào hệ thống Kubernetes.

2.4.2. Kiến trúc

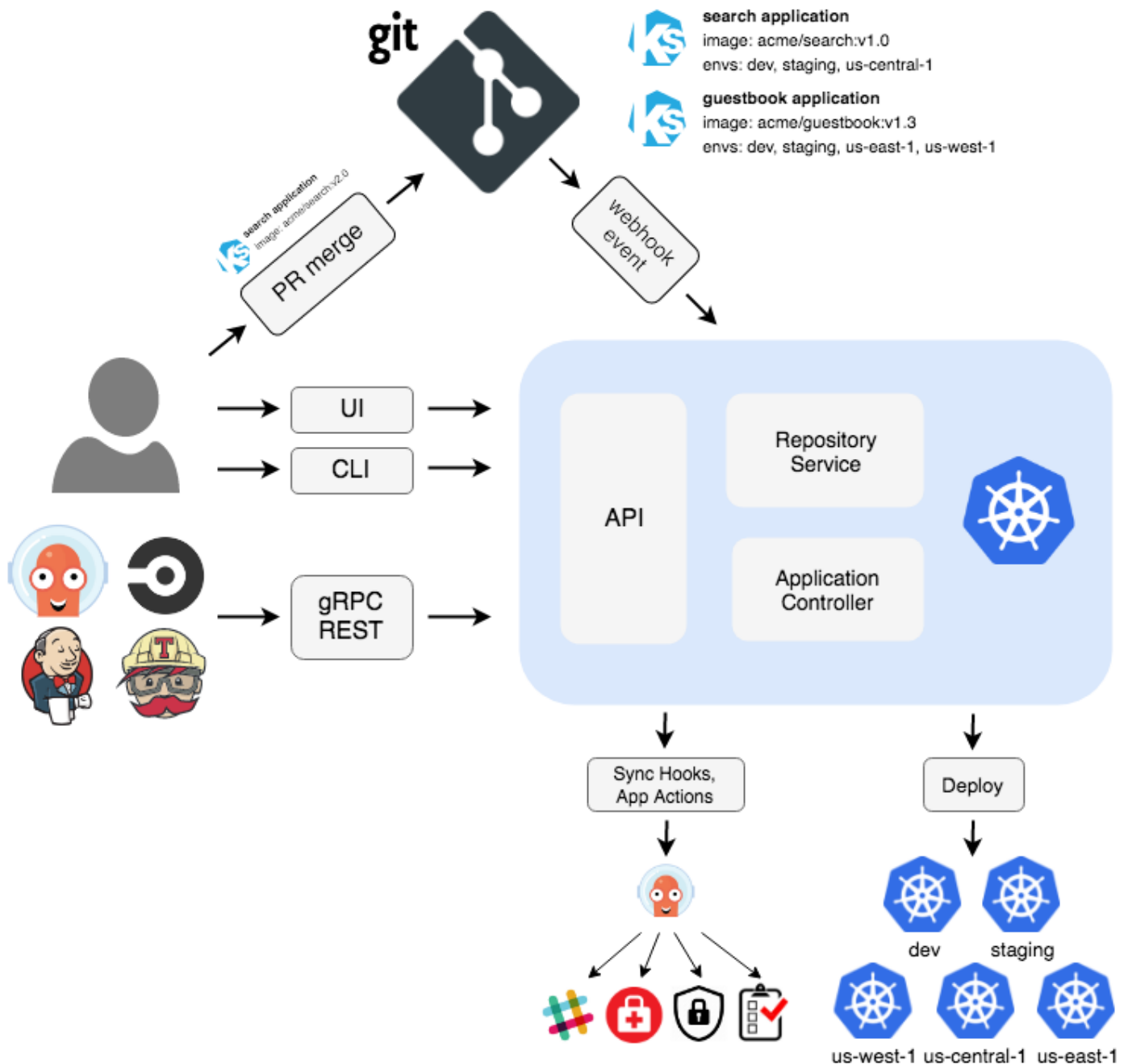


Figure 2.4.1: Kiến trúc của Argo CD

ArgoCd có 3 thành phần chính là Api server, Repository server, Application controller

2.4.2.1. API Server

Là một gRPC/REST server cung cấp các APIs, các APIs này được sử dụng bởi Argocd Web UI, CLI, và các hệ thống CI/CD khác tích hợp. API Server có thể thực hiện các chức năng như: Quản lý ứng dụng và báo cáo trạng thái, thực hiện các hành động như sync, rollback hoặc các tác vụ do người dùng định nghĩa quản lý các credential của repository và cluster (được lưu trữ như thành phần Secret trong Kubernetes), có hỗ trợ xác thực, ủy quyền và lắng nghe các sự kiện WebHook

2.4.2.2. Repository Server

Là một internal service nó duy trì một local cache của git repository giữ các application manifests. Nó chịu trách nhiệm tạo ra và trả về các kubernetes manifests dựa vào các thông tin đầu vào như repository

URL, revision (commit, tag, branch), application path (đường dẫn tới thư mục chứa manifest của ứng dụng được khai báo) và template specific settings: parameters, helm values.yaml

2.4.2.3. Application Controller

Là một kubernetes controller nó tiếp tục thực hiện việc theo dõi trạng thái của ứng dụng đang chạy (trên kubernetes) và so sánh với trạng thái mong muốn (được định nghĩa trong git repo) . Nó phát hiện tình trạng OutOfSync của service và tùy chọn thực hiện các hành động sửa chữa, nó chịu trách nhiệm cho việc gọi bất kỳ user-defined hooks cho các lifecycle events (PreSync, Sync, PostSync)

2.4.3. Cơ chế hoạt động

Luồng xử lý của ArgoCD được chia thành 3 giai đoạn, mỗi giai đoạn được xử lý bởi một thành phần được liệt kê ở trên. 3 giai đoạn của gồm:

2.4.3.1. Lấy source manifest từ remote git repo

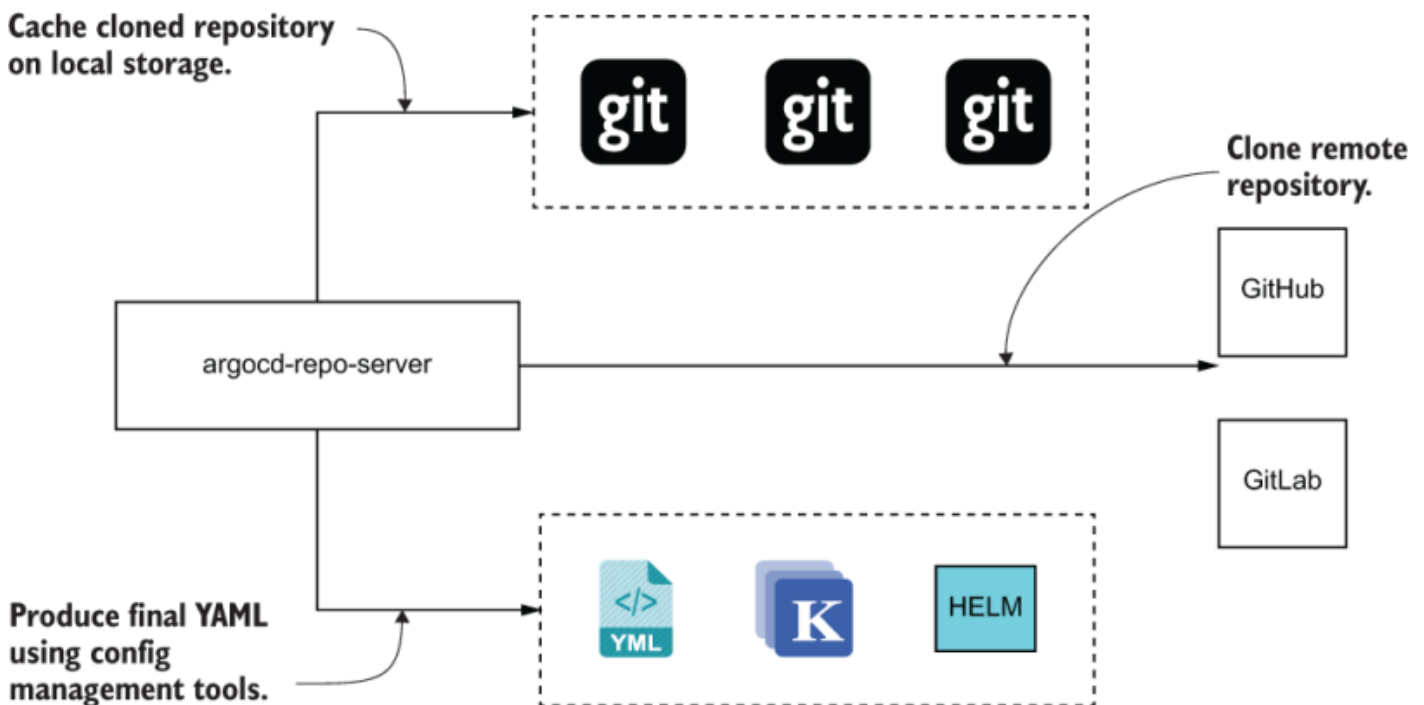


Figure 2.4.2: Luồng hoạt động của ArgoCD

Argocd lấy thông tin file manifest từ remote git repo thông qua bước git pull. Sau đó lưu thông tin này vào local storage của ArgoCD repository server. Tạo ra manifest phục vụ cho Argocd applicatoin controller sử dụng ở giai đoạn tiếp theo.

2.5. RANCHER



Figure 2.5.1: Logo Rancher

Rancher là một công cụ nguồn mở cung cấp giao diện nền web để quản lý các container, bao gồm triển khai ứng dụng, quản lý tài nguyên, theo dõi, giám sát tình trạng chung của cluster... Đơn giản hóa việc quản lý K8S.

2.5.1. Thành phần chính

Rancher server: trung tâm của Rancher cluster bao gồm các thành phần như etcd, authentication proxy, API Rancher server, cluster control. Chức năng chính là cho phép người dùng quản lý, giám sát cung cấp các cluster khác thông qua UI.

Rancher K8S Engine: tạo các RKE cluster.

Cluster control : chịu trách nhiệm thiết lập các liên lạc an toàn giữa Rancher server và từng cluster.

Authentication Proxy: xác thực người gọi bằng các dịch vụ xác thực từ bên thứ 3 sau đó chuyển tiếp lệnh đến cluster thích hợp.

Tác nhân node: thực hiện một số thao tác trên cluster do Rancher khởi chạy.

2.5.2. Chức năng chính

- Triển khai và quản lý các Kubernetes cluster trên nhiều nền tảng, bao gồm các dịch vụ cloud được quản lý (EKS, AKS, GKE), các nhà cung cấp khác (DOKS, LKE, ACK, CCE, OKE, TKE) và on-premises.

- Tạo trình điều khiển tùy chỉnh để hỗ trợ hầu như mọi nền tảng Kubernetes hiện có.
- Cung cấp và cài đặt Kubernetes on-premises hoặc trên cloud.
- Thực thi bảo mật cấp doanh nghiệp bằng bảng điều khiển trung tâm.
- Hỗ trợ Active Directory, LDAP và SAML.
- Quản lý tất cả các Kubernetes cluster từ một giao diện duy nhất.

2.6. RKE2



Figure 2.6.1: Logo RKE2

RKE2, còn được gọi là RKE Chính phủ, là phiên bản tiếp theo của Rancher Kubernetes Engine, phân phối Kubernetes thể hệ mới tập trung vào bảo mật và tuân thủ cho các tổ chức thuộc Chính phủ Hoa Kỳ. Nó được gọi là RKE2 vì đây là phiên bản tiếp theo của Rancher Kubernetes Engine dành cho các trường hợp sử dụng trung tâm dữ liệu. Bản phân phối chạy độc lập hoặc tích hợp vào Rancher. Tính năng cung cấp tự động các cụm RKE2 mới có sẵn trong Rancher v2.6+[11].

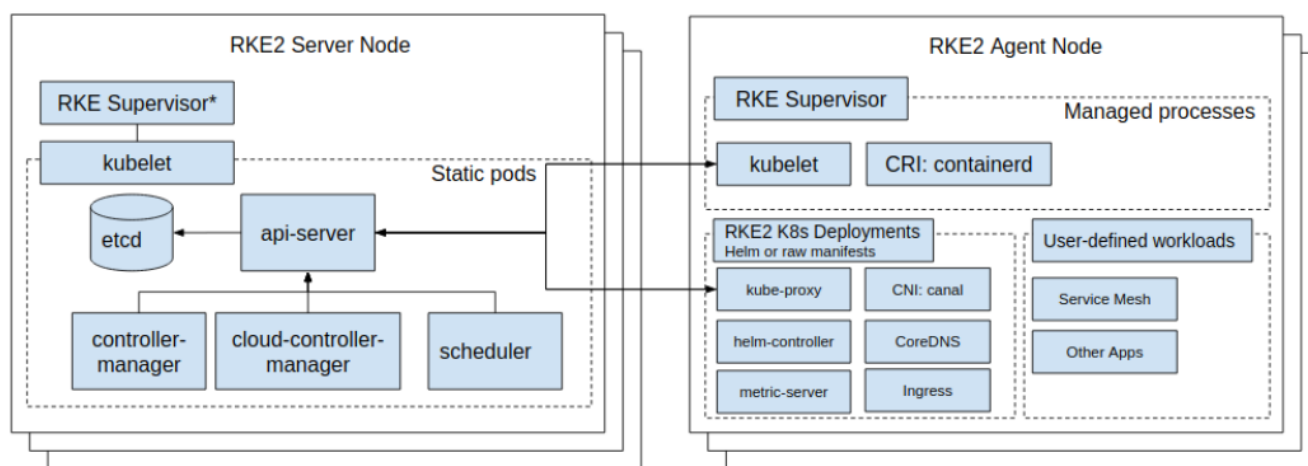


Figure 2.6.2: Kiến trúc RKE2

Cung cấp các tính năng sau:

- Thiết lập **mặc định và tùy chọn cấu hình** cho phép cụm vượt qua CIS Kubernetes Benchmark với sự can thiệp tối thiểu của người vận hành. CIS Kubernetes Benchmark là một bộ hướng

dẫn thực hành bảo mật được công nhận rộng rãi dành cho các cụm Kubernetes. RKE2 đi kèm với các cấu hình mặc định giúp bạn dễ dàng đáp ứng các yêu cầu bảo mật này.

- **Cho phép tuân thủ FIPS 140-2.** FIPS 140-2 là tiêu chuẩn của Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (NIST) dành cho các mô-đun mật mã. RKE2 được xây dựng để hỗ trợ tiêu chuẩn này, giúp bạn đáp ứng các yêu cầu về bảo mật dữ liệu nghiêm ngặt.
- Hỗ trợ **chính sách SELinux** và thực thi nhãn **Multi-Category Security (MCS)**. SELinux là một cơ chế kiểm soát truy cập bắt buộc (MAC) trên Linux giúp cải thiện bảo mật hệ thống. MCS là một tính năng SELinux cho phép gán nhãn bảo mật nhiều mức độ cho các đối tượng, tăng cường khả năng kiểm soát truy cập hơn nữa.
- Quét thường xuyên các thành phần để **tìm lỗ hổng bảo mật (CVE)** bằng trivy trong quy trình xây dựng của chúng. Trivy là một trình quét lỗ hổng mã nguồn containerized. Bằng cách tích hợp trivy, RKE2 đảm bảo các thành phần của nó được cập nhật và vá lỗi kịp thời.

2.7. PROMETHEUS



Figure 2.7.1: Logo Prometheus.

Prometheus là một bộ công cụ theo dõi, giám sát và cảnh báo hệ thống có mã nguồn mở được xây dựng ban đầu bởi SoundCloud. Prometheus có khả năng thu thập và lưu trữ các dữ liệu metric từ các ứng dụng, hệ thống và cơ sở hạ tầng IT thông qua kênh trực tiếp hoặc dịch vụ Pushgateway trung gian và lưu trữ ở các local máy chủ. Prometheus hỗ trợ rất nhiều bộ template giám sát với các mã nguồn mở, giúp việc triển khai và cấu hình giám sát nhanh chóng, hiệu quả. Đi kèm với Prometheus thì cần có Alert Manager để có thể phát hiện được các vấn đề của hệ thống [9].

2.7.1. Kiến trúc

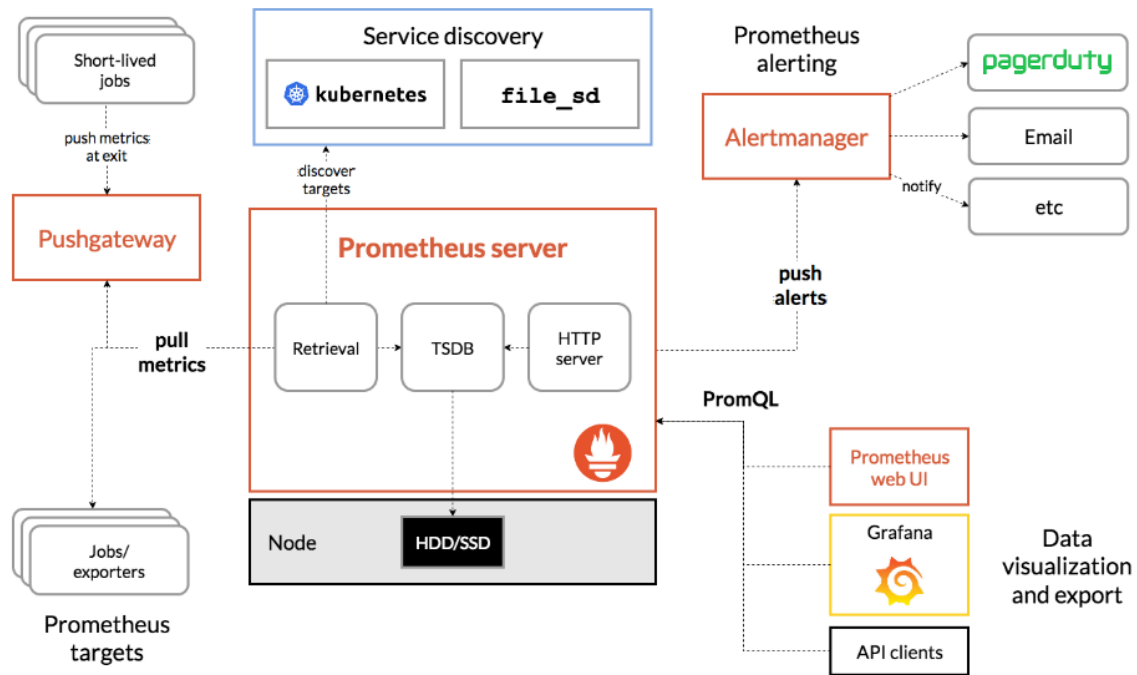


Figure 2.7.2: Kiến trúc Prometheus.

- Thành phần chính là Prometheus server:
 - + Time series Database: lưu trữ thông tin metrics của đối tượng giám sát.
 - + Data Retrieval Worker: lấy thông tin metric từ các đối tượng giám sát và lưu vào database.
 - + HTTP Server API: tiếp nhận các truy vấn lấy dữ liệu được lưu ở DB, hiển thị dữ liệu lên dashboard.
- Thư viện cho các ứng dụng.
- Push Gateway Prometheus: hỗ trợ các đối tượng có thời gian thực hiện ngắn, giúp các metric đẩy về pushgateway rồi về Prometheus server khi Prometheus không thể chủ động lấy dữ liệu.
- Alert Manager: quản lý, xử lý các cảnh báo.

2.7.2. Vai trò của Prometheus trong quản lý K8s

- Giám sát hiệu suất của các K8s cluster và ứng dụng chạy trên K8s.
- Theo dõi tài nguyên CPU, memory, storage của các node trong cluster.
- Giám sát sức khỏe của các pod và service.
- Phát hiện các vấn đề về hiệu suất và đưa ra cảnh báo.

2.8. GRAFANA



Figure 2.8.1: Logo Grafana.

Grafana là một nền tảng để xây dựng các analytics và monitoring hay nói cách khác grafana là một vizualizer hiển thị các metric dưới dạng các biểu đồ hoặc đồ thị, được tập hợp lại thành một dashboard có tính tùy biến cao hỗ trợ theo dõi tình trạng của node một cách dễ dàng hơn. Đây là một nền tảng mã nguồn mở có tính ứng dụng cao trên bất kì lĩnh vực nào có thể thu thập dữ liệu theo dòng thời gian [10].

Vai trò của Grafana trong quản lí K8S:

- Trực quan hóa các dữ liệu của Prometheus.
- Giám sát hiệu suất thu thập dữ liệu từ nhiều nguồn khác nhau trong cụm Kubernetes, bao gồm:
 - + Metrics: CPU, RAM, ổ đĩa, mạng, v.v.
 - + Logs: Kubernetes logs, application logs, v.v.
 - + Events: Kubernetes events, pod events, node event,...
- Phân tích sự cố giúp xác định nguyên nhân của các vấn đề.

2.9. GITLAB



Figure 2.9.1: Logo GitLab.

GitLab là một nền tảng DevOps toàn diện được ứng dụng rộng rãi trong việc quản lý mã nguồn và quy trình phát triển phần mềm. Nền tảng này cung cấp một môi trường tích hợp, hỗ trợ hiệu quả các hoạt động như quản lý kho lưu trữ Git, theo dõi thay đổi, kiểm soát phiên bản, kiểm tra mã, quản lý vấn đề, triển khai tự động và giám sát hiệu suất.

Một vài điểm nổi bật của GitLab:

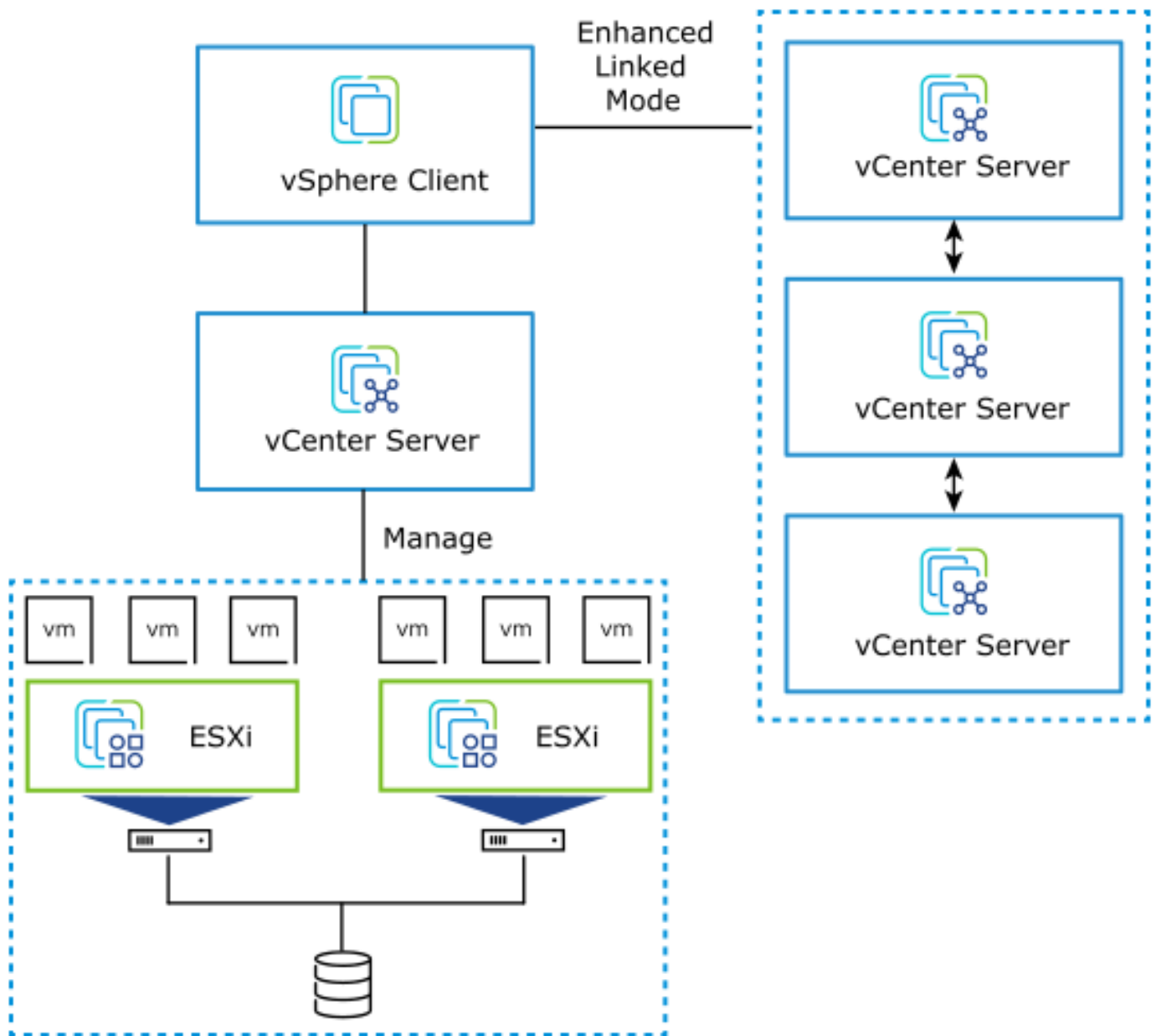
- Lưu trữ kho lưu trữ Git an toàn và hiệu quả.
- Theo dõi và giám sát chi tiết mọi thay đổi trong mã nguồn.
- Quản lý nhánh (branch) và hợp nhất (merge) linh hoạt, hỗ trợ tối ưu cho quy trình phát triển.
- Hệ thống CI/CD tự động, giúp đảm bảo quy trình phát triển liên tục và tích hợp.
- Unit testing: kiểm tra tự động các đơn vị code, đảm bảo tính chính xác và độ tin cậy.
- Theo dõi và quản lý các lỗi, yêu cầu tính năng và nhiệm vụ phát triển.
- Gán nhiệm vụ, phân quyền và theo dõi tiến độ công việc.
- Trao đổi và thảo luận về các vấn đề liên quan đến dự án.

Lợi ích khi sử dụng GitLab:

- Nâng cao hiệu quả phát triển: GitLab giúp tối ưu hóa quy trình phát triển phần mềm, thúc đẩy sự cộng tác và tăng tốc độ triển khai dự án.
- Tăng cường chất lượng phần mềm: Các công cụ kiểm tra mã và quy trình CI/CD tự động giúp đảm bảo chất lượng cao cho sản phẩm phần mềm.
- Cải thiện khả năng cộng tác: GitLab cung cấp môi trường cộng tác hiệu quả, hỗ trợ làm việc nhóm và chia sẻ thông tin thuận lợi.
- Tăng cường bảo mật: GitLab áp dụng các biện pháp bảo mật tiên tiến, bảo vệ mã nguồn và dữ liệu của an toàn.

2.10. VMWARE ESXI

VMware ESXi (Elastic Sky X Integrated) là một nền tảng ảo hóa dạng hypervisor được thiết kế để cung cấp các dịch vụ ảo hóa mạnh mẽ và linh hoạt cho các trung tâm dữ liệu. Được phát triển bởi VMware, ESXi là một trong những giải pháp phổ biến nhất trong lĩnh vực ảo hóa, được sử dụng rộng rãi trong các doanh nghiệp lớn và nhỏ. Sau đây là một cái nhìn tổng quát về VMware ESXi.



Hình 2.10.1: Mô hình kiến trúc hệ thống ảo hóa của VMware

VMware ESXi là một hypervisor loại 1, nghĩa là nó chạy trực tiếp trên phần cứng mà không cần một hệ điều hành cơ sở. Điều này giúp giảm thiểu chi phí về tài nguyên và cải thiện hiệu suất. ESXi được thiết kế để quản lý các máy ảo (VMs), cho phép một máy chủ vật lý duy nhất có thể chạy nhiều hệ điều hành và ứng dụng đồng thời.

- **Các tính năng nổi bật của VMware ESXi:**

1. **Hiệu suất và Khả năng mở rộng:** ESXi cung cấp hiệu suất cao và khả năng mở rộng mạnh mẽ, có thể hỗ trợ hàng trăm VM trên một máy chủ vật lý duy nhất. Các tính năng như vMotion cho phép di chuyển các VM giữa các máy chủ mà không gây gián đoạn.
2. **Bảo mật:** ESXi đi kèm với các tính năng bảo mật tích hợp như mã hóa VM, chứng thực mạnh mẽ và khả năng kiểm soát truy cập chi tiết, giúp bảo vệ dữ liệu và tài nguyên của bạn khỏi các mối đe dọa.

3. **Quản lý tập trung:** Với vCenter Server, ESXi cung cấp khả năng quản lý tập trung cho toàn bộ môi trường ảo hóa, cho phép quản lý dễ dàng và hiệu quả hơn. Bạn có thể theo dõi và kiểm soát tất cả các VM và tài nguyên từ một giao diện duy nhất.
4. **Khả năng Tích hợp:** ESXi tích hợp tốt với nhiều công cụ và ứng dụng khác nhau, bao gồm các giải pháp sao lưu, phục hồi và các hệ thống lưu trữ.
5. **Lợi ích của việc sử dụng VMware ESXi**

- **Tối ưu hóa tài nguyên:** Với ESXi, bạn có thể tận dụng tối đa tài nguyên phần cứng của mình bằng cách chạy nhiều VM trên cùng một máy chủ vật lý.
- **Giảm chi phí:** Việc hợp nhất các máy chủ vật lý thành các VM giúp giảm chi phí về phần cứng, năng lượng và không gian.
- **Tính linh hoạt:** ESXi cung cấp khả năng linh hoạt cao, cho phép triển khai nhanh chóng các môi trường ảo hóa mới mà không cần phải mua thêm phần cứng.

VMware ESXi là một giải pháp ảo hóa mạnh mẽ và đáng tin cậy, cung cấp nhiều tính năng và lợi ích cho các doanh nghiệp. Từ việc cải thiện hiệu suất và bảo mật cho đến quản lý tập trung và tích hợp dễ dàng, ESXi là một lựa chọn tuyệt vời cho những ai muốn tối ưu hóa hạ tầng IT của mình.

2.11. METALB

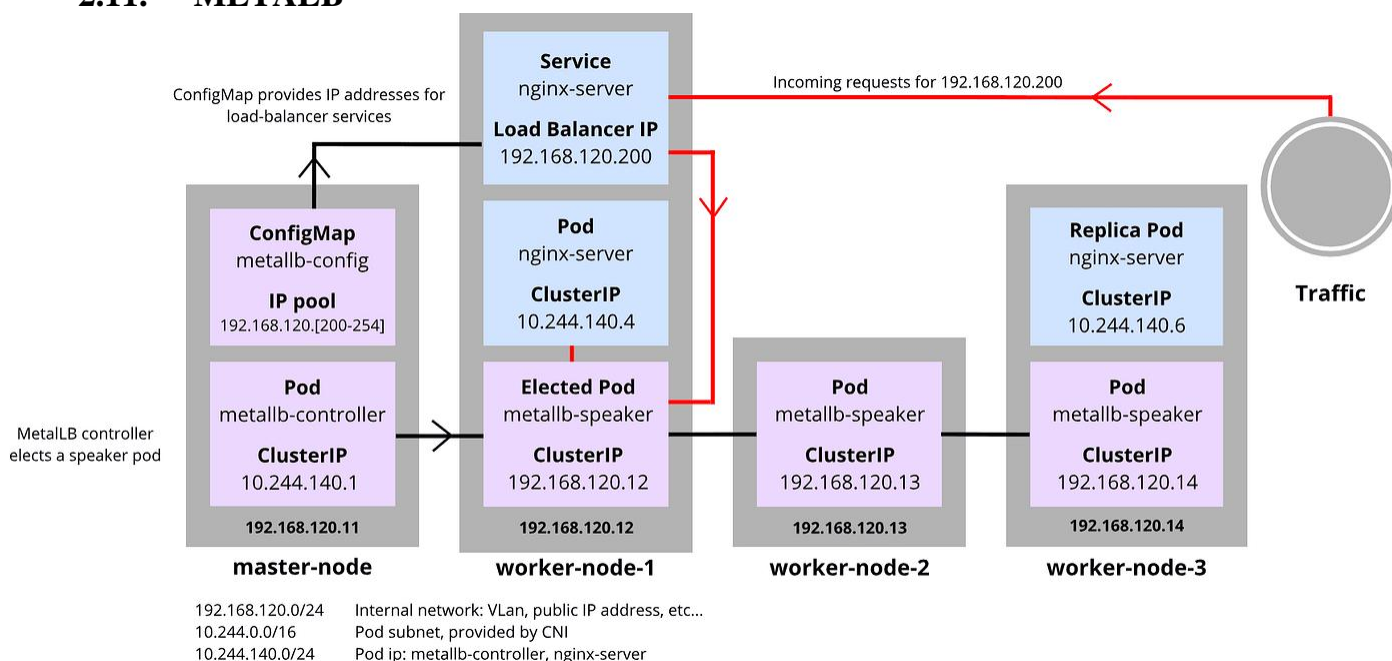


Figure 2.11.1 Mô hình cách hoạt động của MetalLB

- MetalLB là một giải pháp tải trọng mạng (Load Balancer) dành cho các cluster Kubernetes chạy trên các máy chủ vật lý, sử dụng các giao thức định tuyến tiêu chuẩn. Đây là một dự án mở, được phát triển bởi cộng đồng và được sử dụng rộng rãi trong các môi trường Kubernetes không phải trên các nền tảng đám mây như AWS, GCP hay Azure.
- MetalLB được thiết kế để cung cấp các dịch vụ LoadBalancer cho Kubernetes khi chạy trên các máy chủ vật lý. Khác với các nền tảng đám mây, Kubernetes không cung cấp sự hỗ trợ nào cho các dịch vụ LoadBalancer trên các cluster vật lý. MetalLB giúp giải quyết vấn đề này bằng cách tích hợp với thiết bị mạng hiện có, giúp các dịch vụ ngoài mạng trên các cluster vật lý hoạt động mượt mà hơn.
- **Các tính năng nổi bật của MetalLB**

- + **Tích hợp với giao thức định tuyến tiêu chuẩn:** MetalLB sử dụng các giao thức như BGP (Border Gateway Protocol) và ARP (Address Resolution Protocol) để quản lý tải trọng mạng.
- + **Khả năng mở rộng:** MetalLB có khả năng mở rộng cao, hỗ trợ nhiều máy chủ và dịch vụ cùng lúc.

2.12. HAProxy VÀ KEEPALIVED

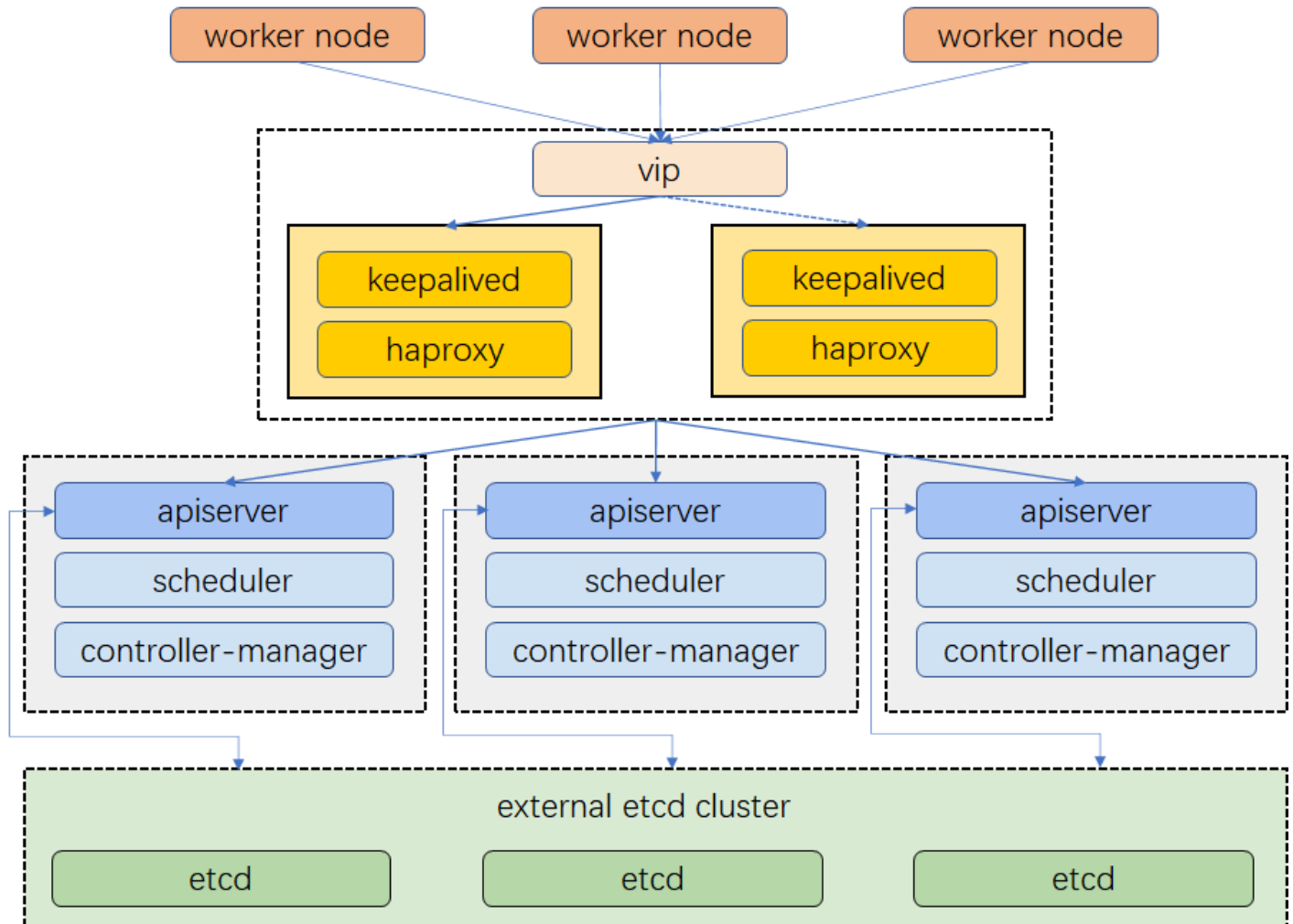


Figure 2.12.1: Mô hình triển khai HA cho Kubernetes

- HAProxy (High Availability Proxy) là một công cụ mã nguồn mở, được sử dụng rộng rãi như một proxy và tải trọng mạng (Load Balancer) trên các hệ thống Linux. HAProxy được thiết kế để cung cấp hiệu suất cao và đáng tin cậy, giúp tối ưu hóa việc phân phối lưu lượng mạng giữa nhiều máy chủ khác nhau.
- HAProxy là một giải pháp proxy và tải trọng mạng mạnh mẽ, được sử dụng trong các hệ thống lớn như GitHub, Twitter và Stack Overflow. Nó hỗ trợ cả giao thức TCP và HTTP, cho phép phân phối lưu lượng mạng một cách hiệu quả và linh hoạt.
- **Các Tính Năng Nổi Bật của HAProxy**
 - + **Hiệu Suất Cao:** HAProxy có hiệu suất cao và nhẹ, tiêu thụ ít tài nguyên, giúp giảm tải cho các máy chủ chính.
 - + **Khả Năng Mở Rộng:** HAProxy có khả năng mở rộng cao, hỗ trợ hàng ngàn kết nối đồng thời mà vẫn duy trì được hiệu suất tốt.
 - + **Bảo Mật:** HAProxy cung cấp nhiều tính năng bảo mật, bao gồm khả năng kiểm tra và ngăn chặn các mối đe dọa từ mạng.
 - + **Hỗ Trợ SSL:** HAProxy tích hợp tốt với SSL, cho phép xử lý các kết nối HTTPS an toàn và bảo mật.

- Keepalived là một dịch vụ daemon trên Linux được sử dụng để cung cấp tính năng High Availability (HA) và Load Balancing thông qua việc quản lý các IP virtual (VIP). Nó hoạt động dựa trên giao thức VRRP (Virtual Router Redundancy Protocol) để đảm bảo rằng nếu máy chủ chính bị lỗi, một máy chủ phụ tạm thời sẽ nhận giữ VIP và tiếp tục phục vụ yêu cầu.
- **Các Tính Năng Nổi Bật của Keepalived**
 - + **Tính năng failover:** Keepalived có khả năng tự động chuyển IP virtual sang máy chủ phụ trong trường hợp máy chủ chính bị lỗi.
 - + **Load Balancing:** Ngoài việc quản lý failover, Keepalived cũng có thể phân phối tải trọng giữa nhiều máy chủ để tối ưu hóa hiệu suất.
 - + **Tích hợp với HAProxy:** Keepalived thường được sử dụng kết hợp với HAProxy để tạo ra một hệ thống HA và Load Balancing mạnh mẽ.

Chương 3. Triển khai hệ thống

3.1. SƠ ĐỒ HỆ THỐNG

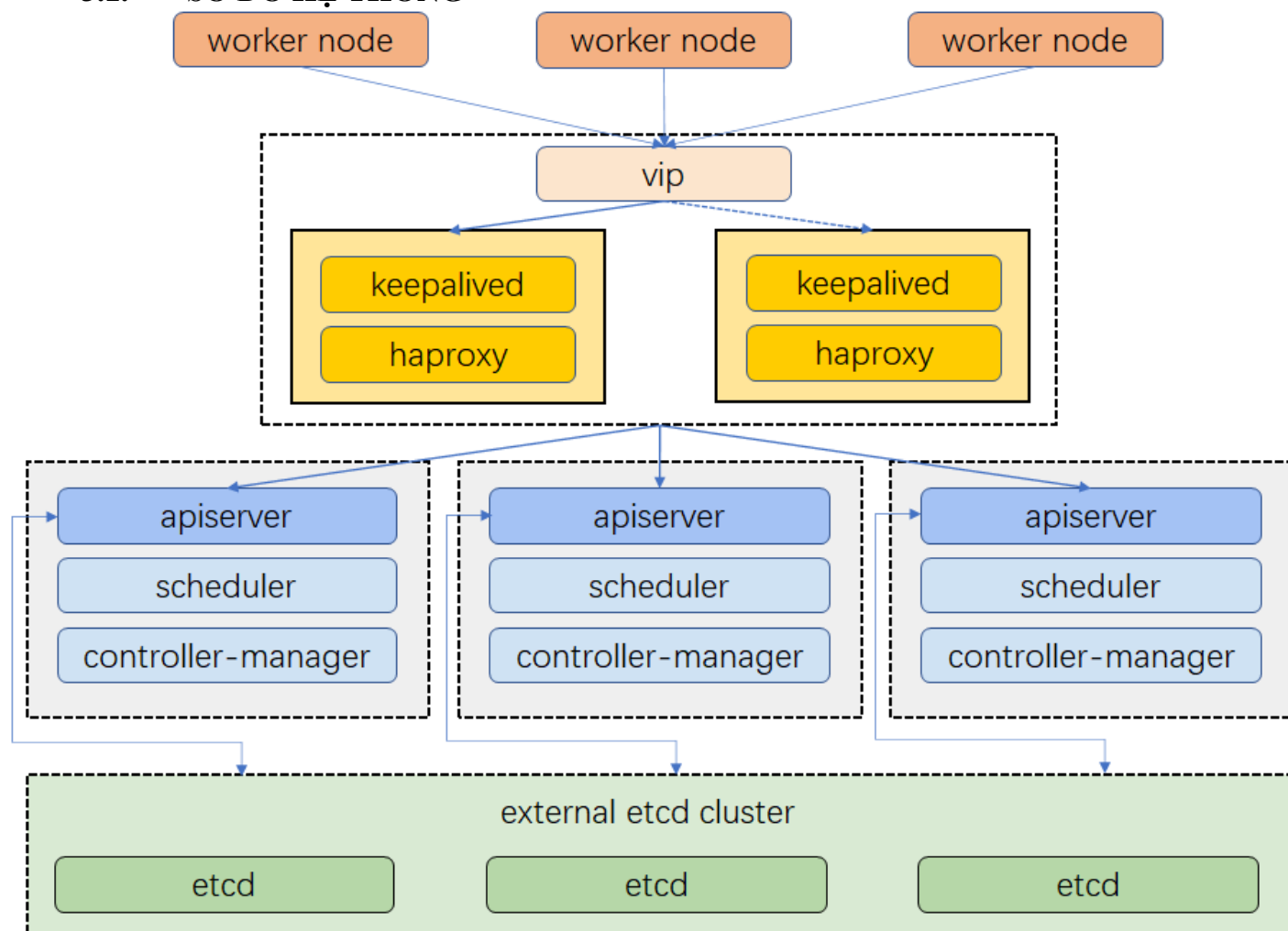


Figure 3.1.1: Sơ đồ hệ thống Kubernetes

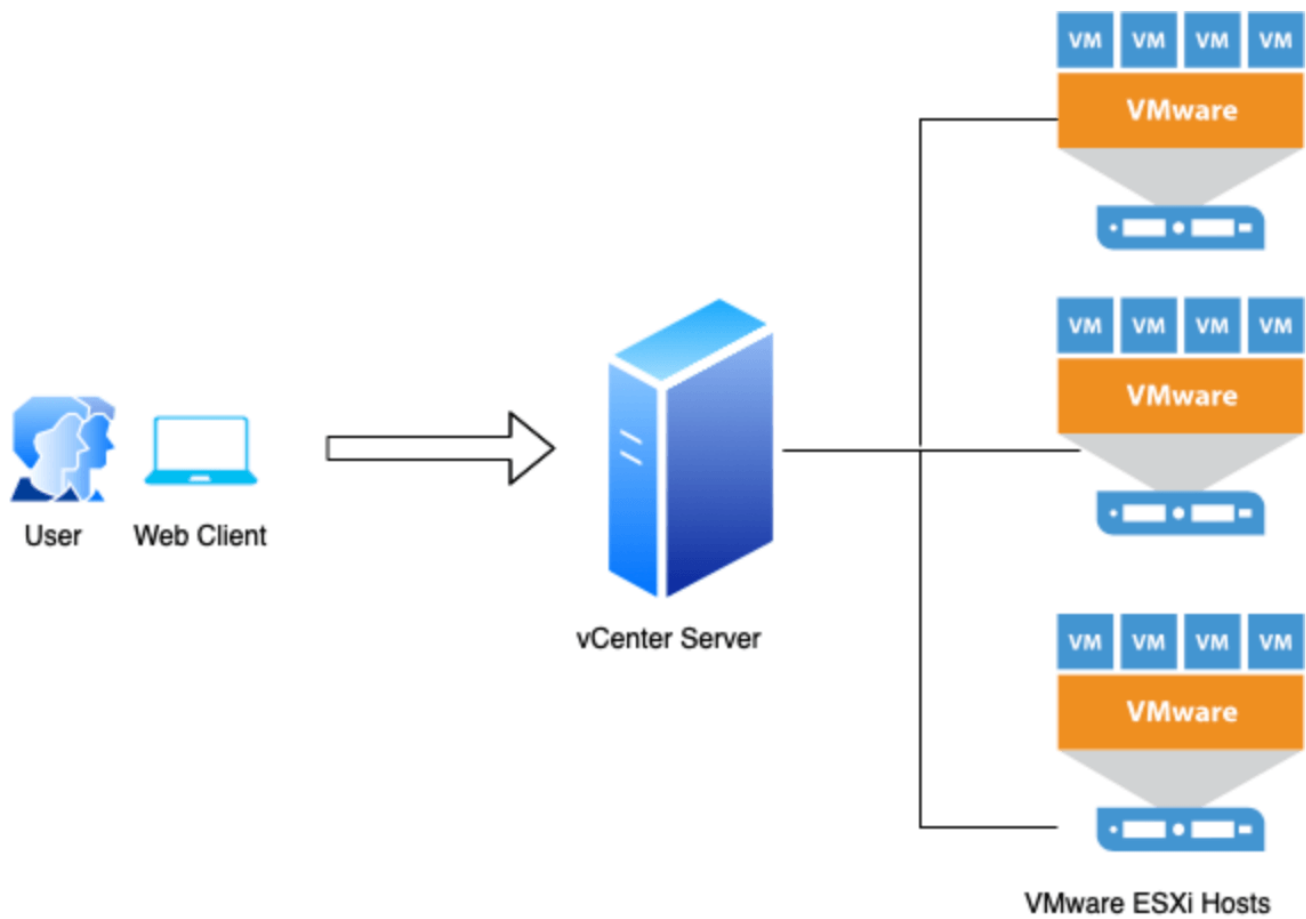


Figure 3.1.2: Mô hình triển khai ESXi và vCenter

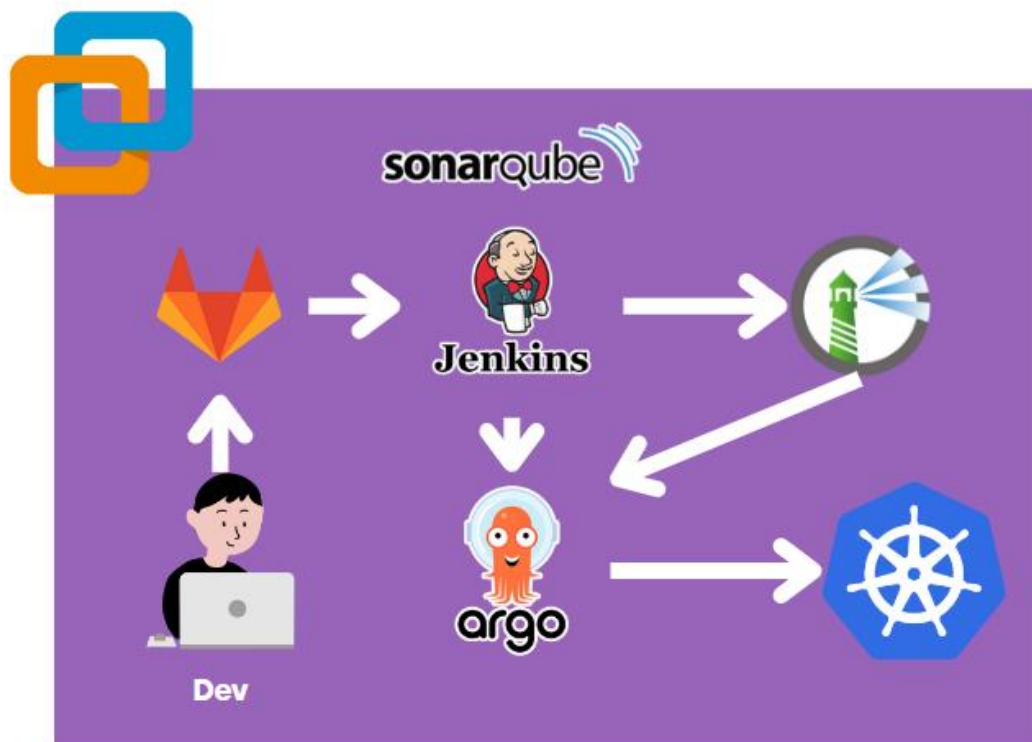


Figure 3.1.3: Sơ đồ hệ thống.

3.2. SOURCECODE MICROSERVICE

3.2.1. Tổng quan application

- **Tên dự án:** Sock Shop
- Tên dự án: Mô phỏng một ứng dụng thương mại điện tử (e-commerce) được xây dựng dựa trên microservices.
- **Chức năng chính:** Hiện thị cách các microservices phối hợp để cung cấp tính năng như quản lý người dùng, giỏ hàng, đặt hàng, thanh toán và theo dõi dịch vụ.
- **Công nghệ sử dụng:**
 - + Containers: Sử dụng Docker để đóng gói từng dịch vụ.
 - + Orchestration: Hỗ trợ triển khai trên Kubernetes.
 - + Giao tiếp dịch vụ: REST API và message queues để trao đổi dữ liệu giữa các microservices.

3.2.2. Application repository

Đồ án sử dụng Gitlab làm nơi lưu trữ source.

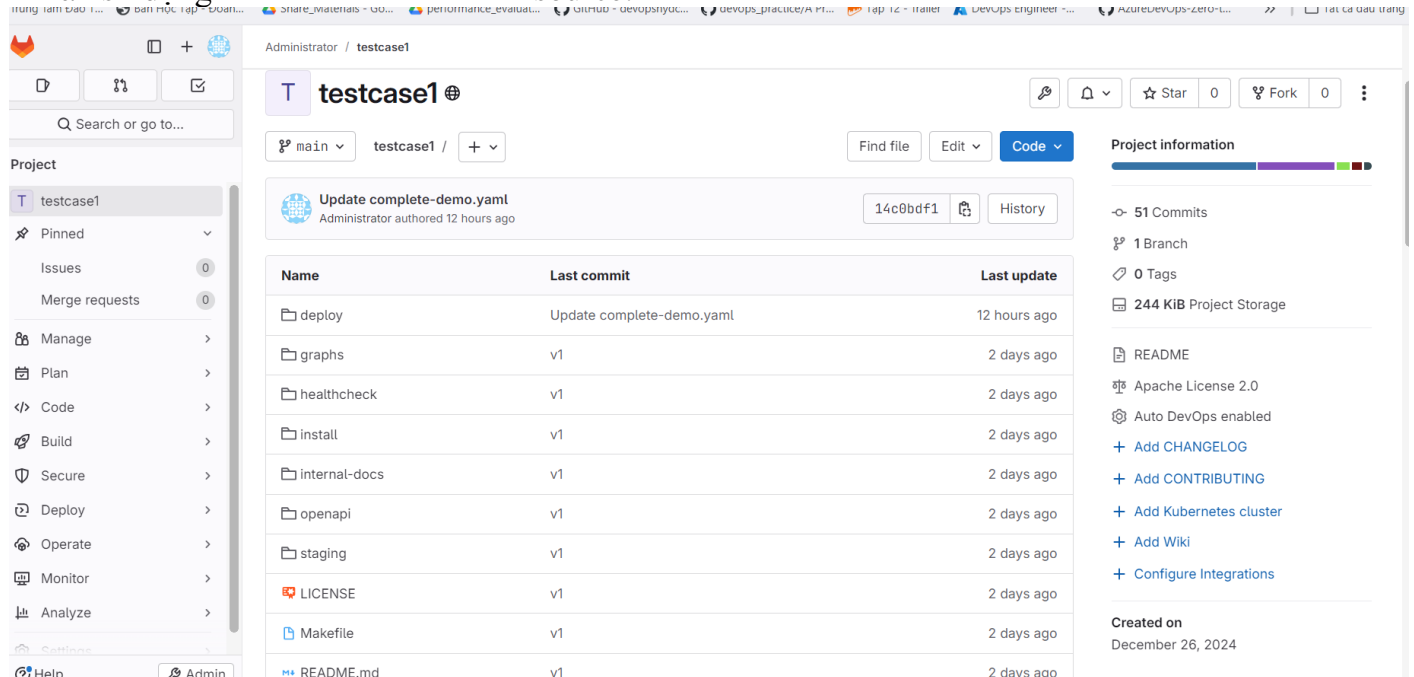


Figure 3.2.1: Repo chứa tất cả dự án microservice.

3.3. DEPLOYMENT REPOSITORY

- Repo tập chung chứa các file cấu hình liên quan đến việc triển khai trên Kubernetes.

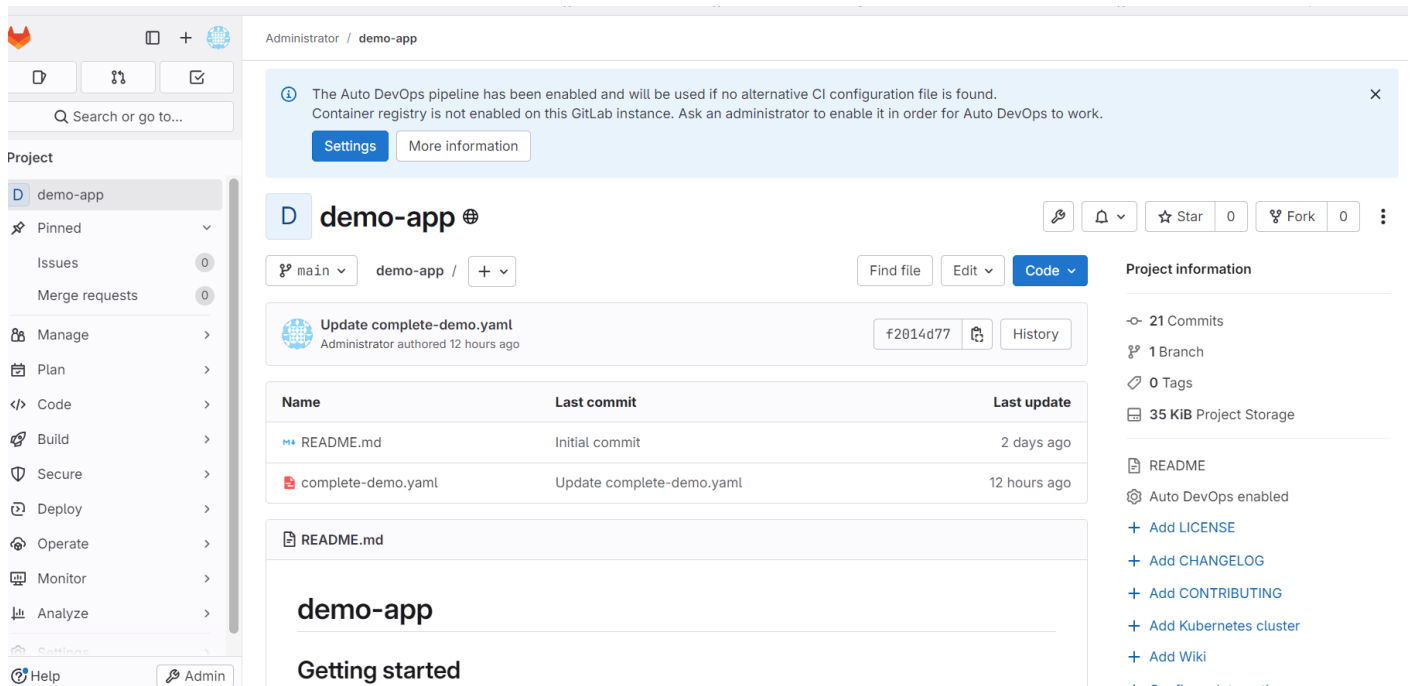


Figure 3.3. : Repo chứa file cấu hình Kubernetes của tất cả microservices

3.4. JENKINS

Nơi quá trình CI được triển khai thực hiện các công việc đã được cài đặt sẵn bằng Jenkinsfile. Quá trình này được kích hoạt tự động khi có tín hiệu push code từ developer lên repo đã được liên kết với Jenkins bằng webhook.

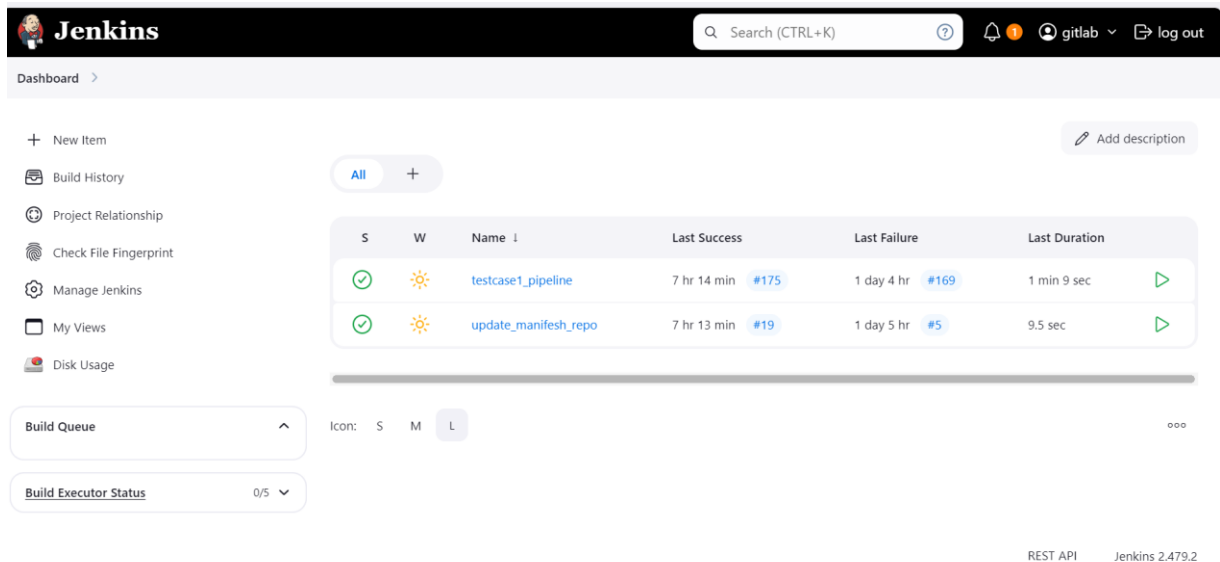


Figure 3.4.1: Jenkins server .

- Trong đồ án này, chúng em cấu hình 2 Jenkinsfile cho 2 jobs gồm:
 - + Testcase1_pipeline: nơi liên kết với repo microservice application thực hiện quá trình checkout repo tiếp theo kiểm tra source code bằng SonarQube sau đó thực hiện build và push các image lên private registry(Harbor) và cuối cùng là thực hiện trigger qua pipeline thứ 2 để kích hoạt quá trình CD.

```

def images= ""
pipeline {
    agent any
    tools {
        maven 'mvn'
    }
    environment {
        //gitlab
        REPO_URL = 'http://10.30.182.20/root/testcase1.git'
        //harbor
        HARBOR_URL = 'harbor.securityzone.vn'
        IMAGE_GROUP = 'testcase1'
        TAG = 'latest'
        SONAR_URL='http://10.30.182.22'
        SCANNER_HOME= tool 'sonar-scanner'
    }
    stages{
        stage('Check Repo') {
            steps {
                script {
                    def exitCode = sh(script: "git ls-remote $REPO_URL", returnStatus: true)
                    if (exitCode != 0) {
                        error("Repository does not exist!")
                    }
                }
            }
        }
        stage('Clone Repo') {
            steps {
                git branch: 'main', url: "${REPO_URL}"
            }
        }
        stage('Static Code Analysis') {
            steps {
                sh '''
                    #SCANNER_HOME /bin/sonar-scanner -X \

```

Figure 3.4.2: Jenkinsfile của Testcase1_pipeline.1

```

-Dsonar.projectName=group20\
-Dsonar.java.binaries=. \
-Dsonar.projectKey=group20 \
...
    }
}

stage('Build and Push Docker Images') {
    steps {
        script {
            sh 'docker compose -f ./deploy/docker-compose/docker-compose.yml pull'
            images = sh(script: "docker compose -f ./deploy/docker-compose/docker-compose.yml config | grep 'image:' | awk '{print \$2}'", returnStdout: true).trim()
            images.split('\n').each {
                image -> def imageName = image.split(":")[0]
                def imageTag = "${HARBOR_URL}/${IMAGE_GROUP}/${imageName}:${BUILD_NUMBER}"
                sh "docker tag ${image} ${imageTag}"
                sh "docker push ${imageTag}"
            }
        }
    }
}

stage('Trigger ManifestUpdate') {
    steps {
        echo "Update manifestjob"
        build job: 'update_manifest_repo',
            parameters: [
                string(name: 'DOCKERTAG', value: "${BUILD_NUMBER}"),
                string(name: 'images', value: "${images}")
            ]
    }
}

```

Figure 3.4.3: Jenkinsfile của Testcase1_pipeline.2

- + update_manifest_repo: Sau khi pipeline đầu tiên thực hiện thành công và kích hoạt trigger triển khai pipeline thứ 2. Pipeline này có nhiệm vụ cập nhật image mới nhất sau khi được push thành công lên Harbor vào file manifest cấu hình triển khai Kubernetes nhằm kích hoạt quá trình CD của ArgoCD.

```
jenkinsfile2
1 pipeline {
2   agent any
3   environment {
4     HARBOR_URL = 'harbor.securityzone.vn'
5     IMAGE_GROUP = 'testcase1'
6     TAG = 'latest'
7   }
8   stages {
9     stage('Checkout') {
10      steps {
11        sh 'echo passed 1'
12        git branch: 'main', changelog: false, poll: false, url: 'http://10.30.182.20/root/demo-app'
13      }
14    }
15
16    stage('Update Deployment File and Commit to Git') {
17      environment {
18        GITLAB_REPO_NAME = "testcase1"
19        GITLAB_USER_NAME = "root"
20        GITLAB_PASSWORD = "Khongcho1."
21        GIT_REPO_URL = "http://10.30.182.20/root/demo-app"
22      }
23      steps {
24        script {
25          catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE'){
26
27            echo "${images}"
28            images.split('\n').each { image ->
29              def imageName = image.split(":")[0]
30              def imageTagex = image.split(":")[1]
31              echo "Updating image tag for ${imageName}"
32
33              def imageTag = "${HARBOR_URL}/${IMAGE_GROUP}/${imageName}:${DOCKERTAG}"
34              //sh "sed -i 's|image: ${HARBOR_URL}/${IMAGE_GROUP}/${imageName}:${imageTagex}|image: ${imageTag}|g' complete-demo.yaml"
35              sh "sed -i 's+${HARBOR_URL}/${IMAGE_GROUP}/${imageName}.+${imageTag}+g' complete-demo.yaml"
36            }
37          }
38        }
39      }
40    }
41  }
42}
```

Figure 3.4.4: Jenkinsfile của update_manifest_repo.1

```

36      }
37    }
38  }
39}

sh'cat complete-demo.yaml'
withCredentials([usernamePassword(credentialsId: 'gitlab-user', usernameVariable: 'GIT_USERNAME', passw
sh '''
  git config --global user.name "root"
  git config --global user.email "jenkins@ci.com"
  git add .
  git commit -m "Update image tags for ${DOCKERTAG}"
  git push http://$GIT_USERNAME:$GIT_PASSWORD@10.30.182.20/root/demo-app HEAD:main
  ...
''')
}
```

Figure 3.4.5 : Jenkinsfile của update_manifest_repo.2

3.5. PRIVATE REGISTRY

Để có thể lưu trữ các Docker image một cách riêng tư. Nhóm sử dụng Opensource Harbor để dựng ra một nơi lưu trữ các bản image.

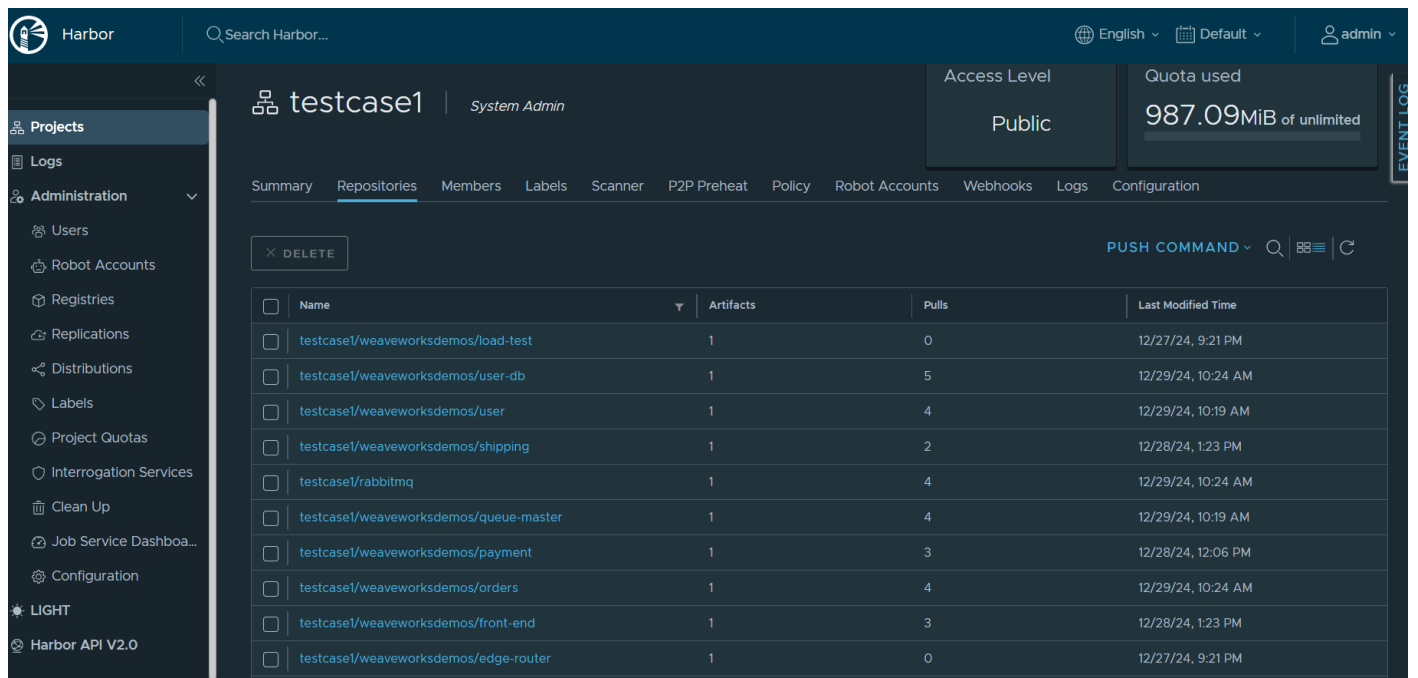


Figure 3.5.1 Registry Harbor.

3.6. ARGO CD VÀ KUBERNETS CLUSTER

Đối với quy trình triển khai, Nhóm sử dụng công cụ ArgoCD. Cài đặt Argocd như là một thành phần của Kubernetes cluster, sau đó cấu hình ArgoCD theo dõi Deployment Repository với thông tin Credential Gitlab phù hợp. Sau đó Argocd tự động triển khai với toàn bộ file cấu hình k8s được định bên trong folder dev.

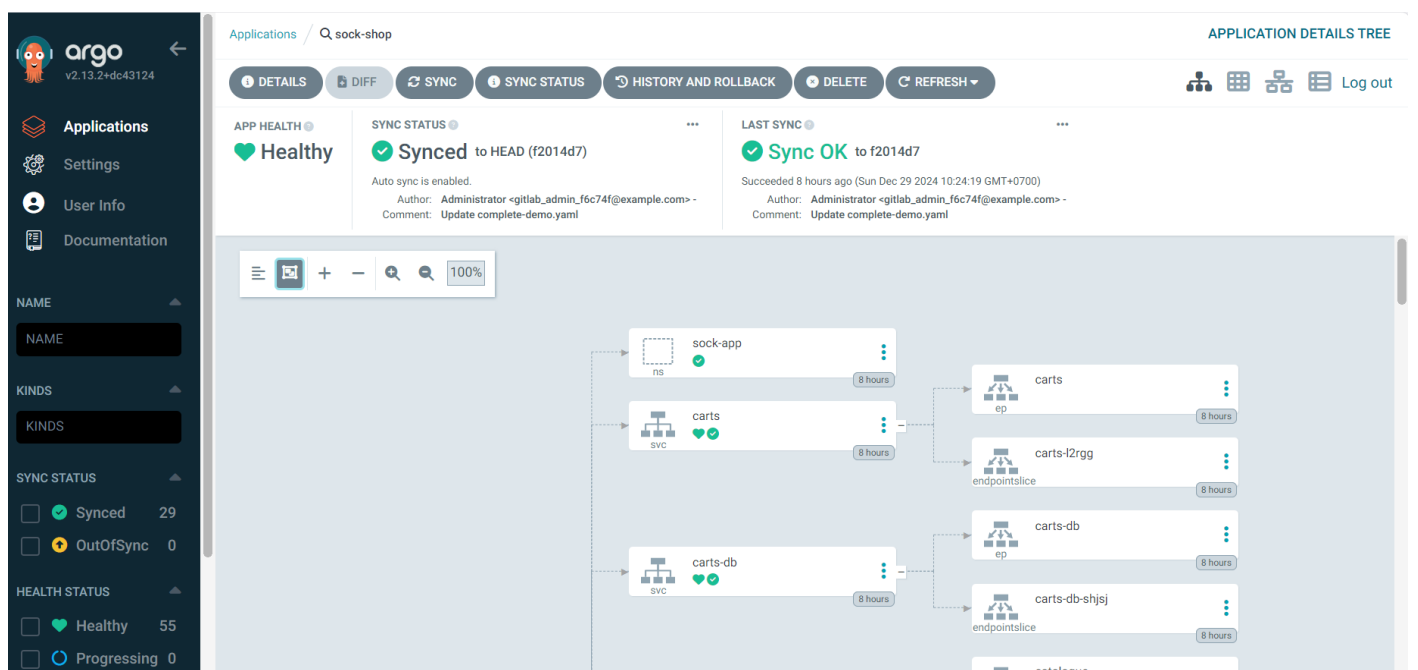


Figure 3.6.1: ArgoCD của đồ án.

3.7. SONARQUBE

Sau khi thực hiện checkout repo, pipeline sẽ tự động kích hoạt quá trình kiểm tra source trên sonarQube qua sonarQube server đã được integrating trước đó.

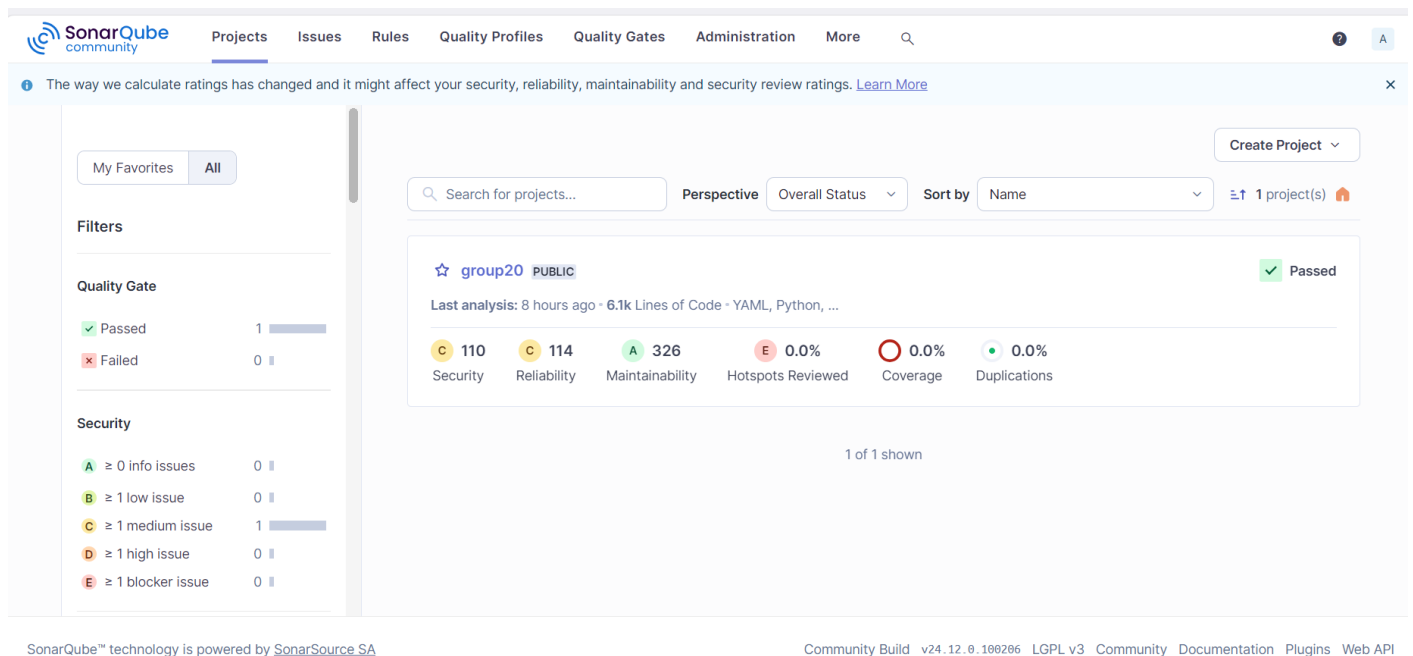


Figure 3.7.1: Kết quả khi kích hoạt sonarQube từ pipeline.

3.8. PROMETHEUS VÀ GRAFANA

Để quản lý hệ thống một cách trực quan và luôn sẵn sàng cho những tình huống xấu nhất như việc bị tấn công mạng DDos, các dịch vụ bị lỗi, RAM hoặc CPU bị sử dụng quá mức. Cần một công cụ quản lý hạ tầng và ứng dụng nhóm chọn sử dụng Prometheus để thu thập dữ liệu và trực quan hóa dữ liệu bằng Grafana gọi chung là Monitoring.

- Một số chức năng mà nhóm sử dụng:
 - + Quản lý các tài nguyên của cluster.

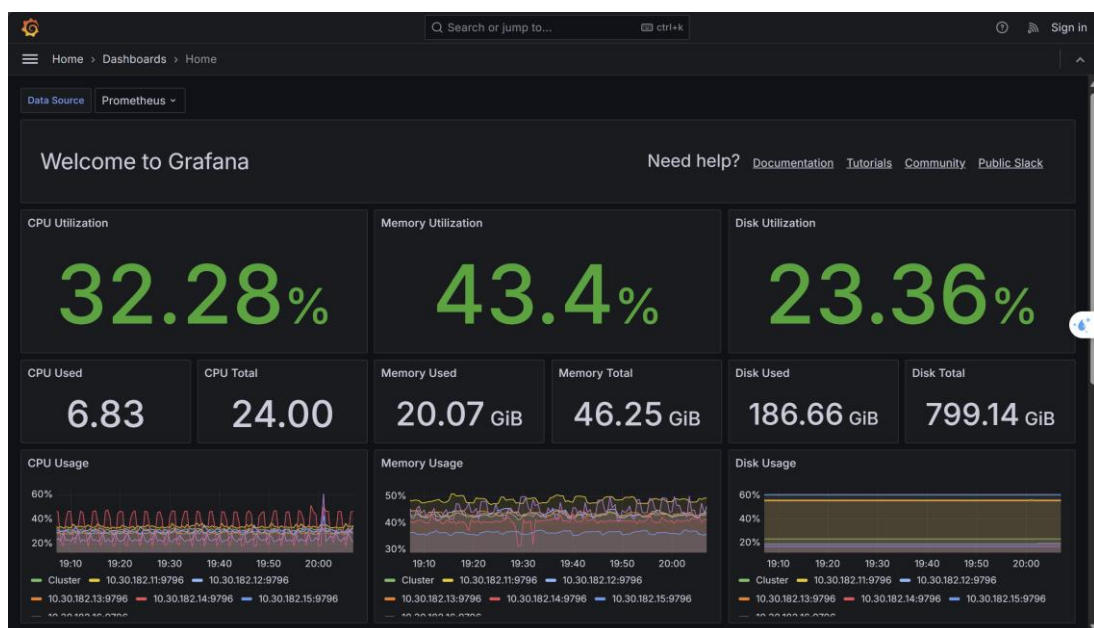


Figure 3.8.1: Giao diện quản lý tổng cluster.

+ Quản lý network của ứng dụng.

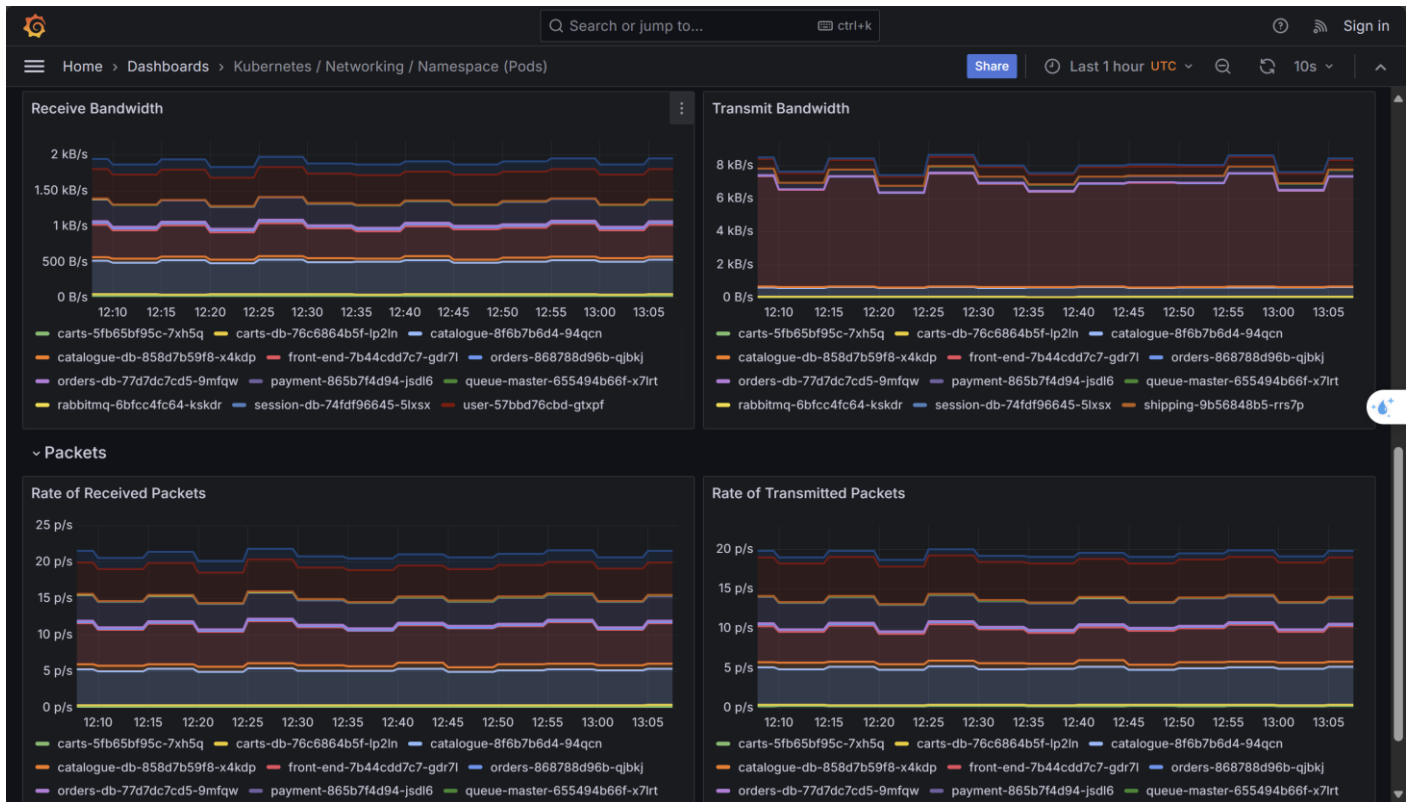


Figure 3.8.2: Giao diện quản lý network của mod-edges.

3.9. TRIỂN KHAI CÁC SERVICE TRÊN KUBERNETES

3.9.1. Triển khai mô hình Kubernetes

- Đồ án triển khai mô hình cluster, sử dụng các dịch vụ máy ảo của Openstack.
- Tài nguyên mà đồ án sử dụng:

+ GITLAB

- Tên: SYS_POP_LAB_GITLAB_182.20
- Hệ điều hành: Ubuntu server 24.04 LTS
- RAM: 8GB
- VCPUs: 4 VCPU
- Disk: 60GB
- IP: 10.30.182.20

+ HARBOR

- Tên: SYS_POP_LAB_HARBOR_182.40
- Hệ điều hành: Ubuntu server 24.04 LTS
- RAM: 8GB
- VCPUs: 4 VCPU

- Disk: 60GB
- IP: 10.30.182.40
- + **JENKINS**
 - Tên: SYS_POP_LAB_JENKINS_182.30
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.30
- + **HAProxy1**
 - Tên: SYS_POP_LAB_RKE2_HAProxy1_182.17
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.17
- + **HAProxy2**
 - Tên: SYS_POP_LAB_RKE2_HAProxy2_182.18
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.18
- + **Master1**
 - Tên: SYS_POP_LAB_RKE2_Master1_182.11
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.11
- + **Master2**
 - Tên: SYS_POP_LAB_RKE2_Master2_182.12
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.12
- + **Master3**

- Tên: SYS_POP_LAB_RKE2_Master3_182.13
- Hệ điều hành: Ubuntu server 24.04 LTS
- RAM: 8GB
- VCPUs: 4 VCPU
- Disk: 60GB
- IP: 10.30.182.13
- + **Worker1**
 - Tên: SYS_POP_LAB_RKE2_Worker1_182.14
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.14
- + **Worker2**
 - Tên: SYS_POP_LAB_RKE2_Worker2_182.15
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.15
- + **Worker3**
 - Tên: SYS_POP_LAB_RKE2_Worker3_182.16
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.16
- + **SONARQUBE**
 - Tên: SYS_POP_LAB_SONARQUBE_182.22
 - Hệ điều hành: Ubuntu server 24.04 LTS
 - RAM: 8GB
 - VCPUs: 4 VCPU
 - Disk: 60GB
 - IP: 10.30.182.22



```

SYS_POP_LAB_GITLAB_182.20
SYS_POP_LAB_HARBOR_182.40
SYS_POP_LAB_JENKINS_182.30
SYS_POP_LAB_RKE2_HAProxy1_182.17
SYS_POP_LAB_RKE2_HAProxy2_182.18
SYS_POP_LAB_RKE2_Master1_182.11
SYS_POP_LAB_RKE2_Master2_182.12
SYS_POP_LAB_RKE2_Master3_182.13
SYS_POP_LAB_RKE2_Worker1_182.14
SYS_POP_LAB_RKE2_Worker2_182.15
SYS_POP_LAB_RKE2_Worker3_182.16
SYS_POP_LAB_SONARQUBE_182.22
  
```

Figure 3.9.1: Số lượng máy ảo sử dụng.

VM Hardware




CPU	4 CPU(s), 75 MHz used
Memory	8 GB, 0 GB memory active
Hard disk 1	60 GB Thin Provision ⓘ vsanDatastore
Network adapter 1	Lab-KHuy_182 (connected) 00:50:56:89:b9:bd
CD/DVD drive 1	Disconnected  ▾
Compatibility	ESXi 8.0 U2 and later (VM version 21)

Figure 3.9.2: Thông số của các máy ảo

- Sau khi chuẩn bị tài nguyên và môi trường, kết nối các tài nguyên trên thành một cluster kubernetes với RKE2.

```

root@master1:/home/ubuntu# kubectl get node
NAME          STATUS    ROLES          AGE      VERSION
master1       Ready    control-plane,etcd,master  6d21h   v1.31.4+rke2r1
master2       Ready    control-plane,etcd,master  6d21h   v1.31.4+rke2r1
master3       Ready    control-plane,etcd,master  6d20h   v1.31.4+rke2r1
worker1       Ready    <none>         6d21h   v1.31.4+rke2r1
worker2       Ready    <none>         6d21h   v1.31.4+rke2r1
worker3       Ready    <none>         6d21h   v1.31.4+rke2r1

```

Figure 3.9.3: Các kubernetes nodes sau khi triển khai.

- Sử dụng kubernetes rancher web dashboard.

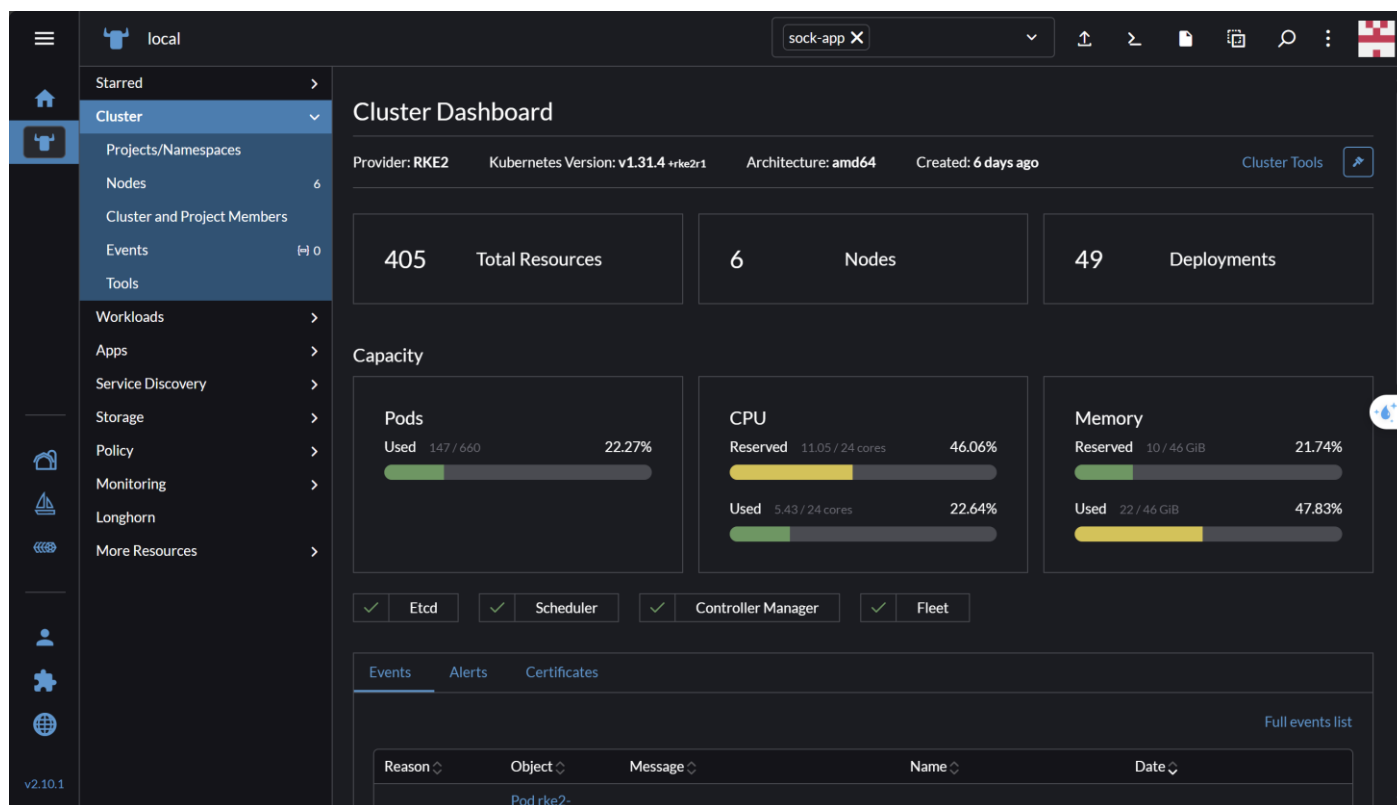


Figure 3.9.4: Rancher dashboard.

3.9.2. Triển khai ứng dụng lên kubernetes

- Sau khi triển khai ứng dụng

State	Name	Namespace	Image	Ready	Restarts	IP	Node	Age
Running	carts-5fb65bf95c-7xh5q	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/carts:175	1/1	0	10.42.3.132	worker2	9 hours
Running	carts-db-76c6864b5f-lp2ln	sock-app	harbor.securityzone.vn/testcase1/mongo:175	1/1	0	10.42.8.101	master3	9 hours
Running	catalogue-8f6b7b6d4-94qcn	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/catalogue:175	1/1	0	10.42.3.131	worker2	9 hours
Running	catalogue-db-858d7b59f8-x4kdp	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/catalogue:175	1/1	0	10.42.1.113	master2	9 hours
Running	front-end-7b44cdd7c7-gdr7l	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/front-end:175	1/1	0	10.42.0.115	master1	9 hours
Running	orders-868788d96b-qjbjk	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/orders:175	1/1	0	10.42.0.116	master1	9 hours
Running	orders-db-77d7dc7cd5-9mfqw	sock-app	harbor.securityzone.vn/testcase1/mongo:175	1/1	0	10.42.8.102	master3	9 hours
Running	payment-865b7f4d94-jsdl6	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/payment:175	1/1	0	10.42.7.245	worker1	9 hours
Running	queue-master-655494b66f-x7lrt	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/queue-master:175	1/1	0	10.42.8.103	master3	9 hours
Running	rabbitmq-6bfcc4fc64-kskdr	sock-app	harbor.securityzone.vn/testcase1/rabbitmq:175	2/2	0	10.42.3.134	worker2	9 hours
Running	session-db-74fdf96645-5lxsx	sock-app	redis:alpine	1/1	0	10.42.7.240	worker1	9 hours
Running	shipping-9b56848b5-rrs7p	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/shipping:175	1/1	0	10.42.7.246	worker1	9 hours
Running	user-57bbd76cbd-gtxpf	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/user:175	1/1	0	10.42.3.133	worker2	9 hours
Running	user-db-bb9bd5d74-p4hf9	sock-app	harbor.securityzone.vn/testcase1/weaveworksdemo/s/user-db:175	1/1	0	10.42.5.69	worker3	9 hours

Figure 3.9.5: Số lượng pod mà đồ án sử dụng.

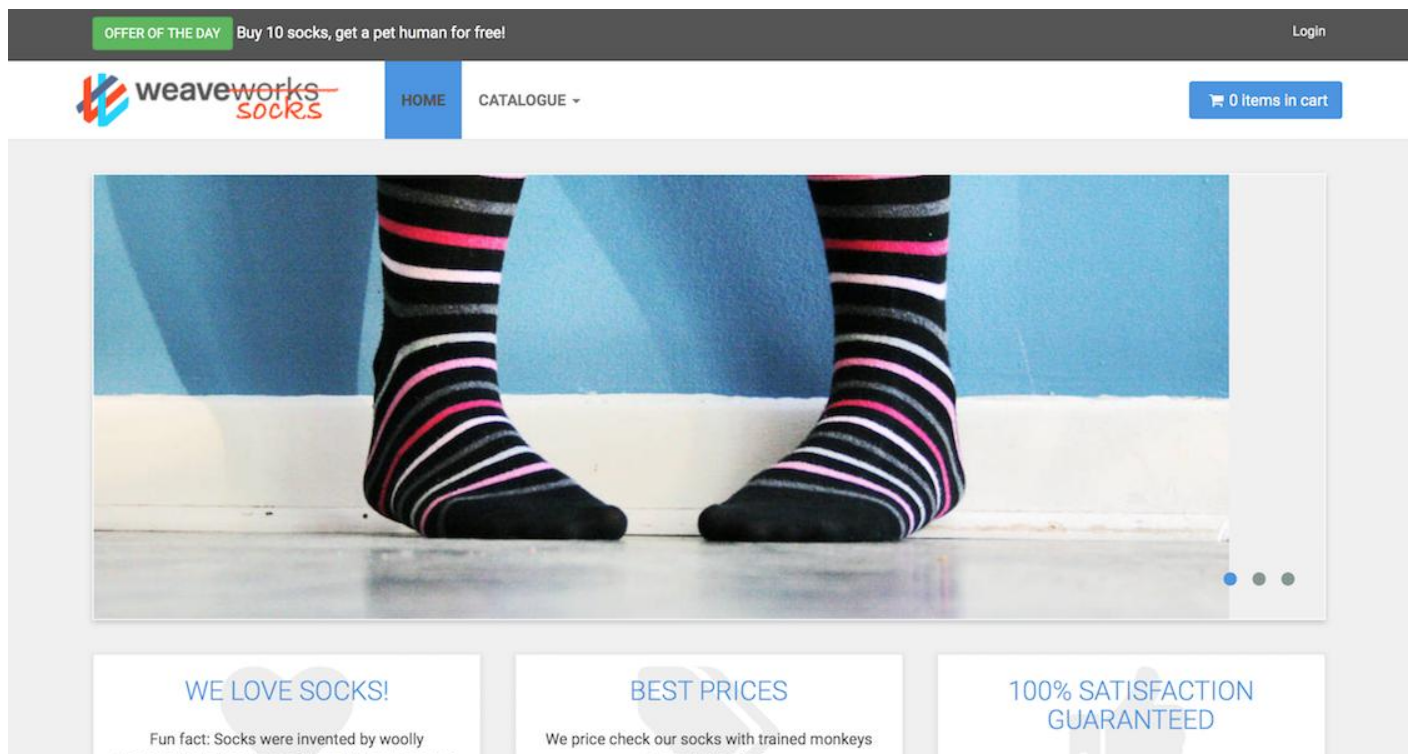


Figure 3.9.6: Giao diện web