BÁO CÁO ĐỒ ÁN MÔN HỌC

Xây dựng hệ thống giám soát các ứng dụng Microservices

Môn học: NT531

Lớp: P11.MMCL

THÀNH VIÊN THỰC HIỆN (Nhóm 10):

STT	Họ và tên	MSSV
1	Ngô Vũ Minh Đạt	21521935
2	Quách Minh Huy	21522164

Giảng Viên hướng dẫn: GS. Lê Trung Quân

ĐÁNH GIÁ KHÁC:

Tổng thời gian thực hiện	2 tháng
Phân chia công việc	Terraform, Ansible: Quách Minh Huy Kubernetes, Spring app, Helm: Ngô Vũ Minh Đạt
Ý kiến *(nếu có)* + Khó khăn + Đề xuất, kiến nghị	

Phần bên dưới của báo cáo này là báo cáo chi tiết của nhóm thực hiện

Danh sách hình ảnh

A. Mở Đầu

1. Lý do chọn đề tài

- Microservices ngày càng phổ biến vì khả năng giúp xây dựng ứng dụng linh hoạt, mở rộng dễ dàng. Tuy nhiên, kiến trúc này cũng làm tăng độ phức tạp trong việc giám sát và quản lý. Việc theo dõi tình trạng từng dịch vụ, lưu lượng và hành vi yêu cầu là cần thiết để duy trì hiệu suất, độ tin cậy và khả năng mở rộng của toàn hệ thống.
- Giám sát và tracing giúp nhanh chóng xác định các vấn đề hoặc điểm nghẽn trong quá trình xử lý yêu cầu. Từ đó, đội ngũ vận hành có thể kịp thời đưa ra giải pháp nhằm giảm thiểu thời gian ngừng hoạt động, đảm bảo trải nghiệm người dùng ổn định.

2. Mục tiêu của đề tài

- Tìm hiểu kiến trúc và sử dụng các công cụ monitoring như Prometheus, Grafana, Promtail

- Đồ án xây dựng hệ thống giám sát và theo dõi ứng dụng microservices chạy trên một cụm Kubernetes 4 node.
- Mục tiêu của hệ thống là cung cấp khả năng giám sát toàn diện và truy vết (tracing) chi tiết để hỗ trợ phát hiện và khắc phục sự cố nhanh chóng trong môi trường microservices.

B. Chương I. CƠ SỞ LÝ THUYẾT.

1. Kubernetes

a. Tổng quan

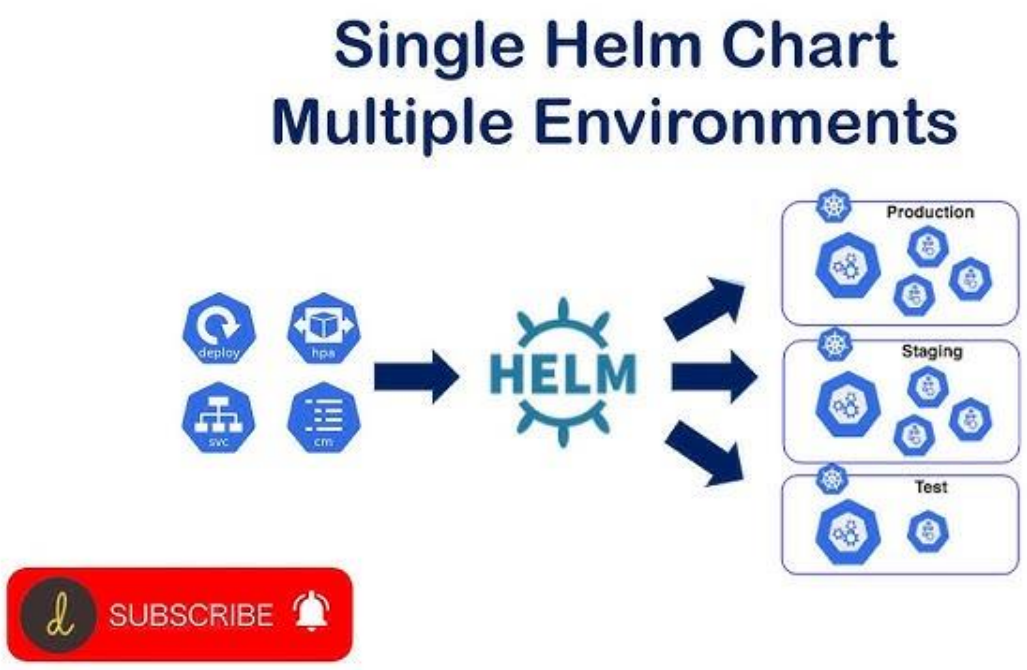
Kubernetes (viết tắt là k8s) là một hệ thống mã nguồn mở được phát triển bởi Google để quản lý và tự động hóa việc triển khai, mở rộng và quản lý các ứng dụng container. Kubernetes giải quyết các thách thức của việc quản lý các ứng dụng phức tạp, đa dạng và có quy mô lớn trong môi trường cloud-native. Điều này bao gồm việc tự động hóa việc triển khai ứng dụng, tự động phục hồi khi có lỗi xảy ra, cân bằng tải động, và quản lý tài nguyên hiệu quả. [1]

b. Các thành phần chính của Kubenertes:

- Master Node: Là trung tâm quản lý của hệ thống, bao gồm các thành phần như API Server, Scheduler và Controller Manager.
- Node (Minion): Là các máy chủ trong cụm Kubernetes, chạy các container và các Agent như Kubelet để giữ liên lạc với Master Node.
- Pods: Là đơn vị nhỏ nhất của Kubernetes, chứa một hoặc nhiều container, chia sẻ mạng và lưu trữ.
- Service: Là một cơ chế để tạo ra một địa chỉ IP ổn định và tải cân bằng cho một nhóm các Pod, cho phép truy cập vào ứng dụng một cách dễ dàng và nhất quán.
- Volume: Là một cách để lưu trữ dữ liệu dùng chung giữa các container trong một Pod hoặc giữa các Pod.
- Namespace: Là một cách để phân chia cụm Kubernetes thành các phân vùng ảo, giúp tổ chức và quản lý tài nguyên một cách hiệu quả.

2. Helm chart

a. Tổng quan



Hình 1 Tổng quan Helm chart

Helm Chart là một công cụ quan trọng trong hệ sinh thái Kubernetes, được thiết kế để quản lý các ứng dụng và dịch vụ trên nền tảng này một cách dễ dàng và hiệu quả. Helm, được biết đến như "trình quản lý gói" cho Kubernetes, giúp đơn giản hóa quá trình triển khai, quản lý và cập nhật các ứng dụng phức tạp trên cụm Kubernetes.

b. Các thành phần chính của Helm chart

- Một Helm Chart là một bộ sưu tập các tệp tin định nghĩa một ứng dụng hoặc dịch vụ trong Kubernetes. Chart chứa tất cả các thông tin cần thiết để triển khai ứng dụng, bao gồm các cấu hình, tài nguyên, và thông tin phụ thuộc (dependencies).
    - Mỗi Chart thường bao gồm một tệp Chart.yaml (chứa metadata như tên, phiên bản, và mô tả), một thư mục templates (chứa các tệp mẫu để tạo ra các tài nguyên Kubernetes), và một tệp values.yaml (chứa các biến cấu hình có thể tùy chỉnh).
- Helm sử dụng các tệp mẫu (templates) để tạo ra các tài nguyên Kubernetes như pod, service, deployment, và ingress. Các tệp mẫu này được viết bằng ngôn ngữ định nghĩa mẫu (Go templates), cho phép người dùng dễ dàng tùy chỉnh cấu hình mà không cần phải viết lại toàn bộ tệp YAML.
- Tệp values.yaml cho phép người dùng xác định các giá trị mặc định cho các biến trong tệp mẫu. Người dùng có thể tùy chỉnh các giá trị này khi triển khai Chart để phù hợp với yêu cầu cụ thể của ứng dụng hoặc môi trường.

3. Promethues

Hình 2 Prometheus Logo

a. Tổng quan

Prometheus là một hệ thống giám sát mã nguồn mở do SoundCloud phát triển vào năm 2012, được thiết kế để theo dõi các hệ thống và ứng dụng phức tạp. Công cụ này cung cấp cái nhìn chi tiết về hiệu suất, độ sẵn sàng, và các vấn đề liên quan đến hệ thống. Prometheus thu thập dữ liệu metric dưới dạng chuỗi thời gian, tức là dữ liệu được lưu với mốc thời gian ghi nhận. Người dùng có thể truy vấn và phân tích dữ liệu này bằng ngôn ngữ truy vấn Prometheus (PromQL) [2].

Tính tới thời điểm hiện tại, Prometheus là một phần mềm hoàn toàn miễn phí và không có phiên bản trả phí nào. Bên dưới là một số tính năng chính của Prometheus:

- Mô hình dữ liệu đa chiều: Dữ liệu metric có thể được chia nhỏ theo ý muốn, dọc theo các chiều labels mà Prometheus cấu hình
- Hỗ trợ nhiều nguồn dữ liệu: Prometheus có thể thu thập dữ liệu từ nhiều nguồn khác nhau, bao gồm các ứng dụng, hệ thống, container và dịch vụ đám mây.
- Hỗ trợ nhiều định dạng: Prometheus hỗ trợ nhiều định dạng dữ liệu metric khác nhau, bao gồm JSON, OpenMetrics và InfluxDB.
- Khả năng mở rộng cao: Prometheus có thể được mở rộng để giám sát một lượng lớn các hệ thống và ứng dụng.

Dễ dàng sử dụng: Prometheus có giao diện người dùng web và CLI dễ sử

dụng.

4. Loki

Hình 3 Loki Logo

a. Tổng quan

Loki là một hệ thống quản lý log mã nguồn mở do Grafana Labs phát triển, được thiết kế để lưu trữ và truy vấn log một cách hiệu quả, đặc biệt trong các môi trường phân tán và hệ thống container như Kubernetes. Loki nổi bật vì đơn giản và tiết kiệm tài nguyên nhờ phương pháp lưu trữ log không sử dụng chỉ mục phức tạp. Được lấy cảm hứng từ Prometheus, Loki chỉ lưu trữ các log cùng với metadata cơ bản (label) từ các ứng dụng hoặc container, giúp tối ưu hoá quá trình lưu trữ và truy vấn mà không đòi hỏi nhiều dung lượng. [3]

5. Promtail

a. Tổng quan

Promtail là một công cụ mã nguồn mở thuộc hệ sinh thái giám sát của Grafana Labs, chủ yếu được sử dụng để thu thập và chuyển tiếp log từ nhiều nguồn khác nhau vào Loki - hệ thống lưu trữ log phi tập trung của Grafana. Promtail hoạt động như một agent chạy trên các máy chủ hoặc trong môi trường container, có nhiệm vụ đọc log từ các nguồn log khác nhau, sau đó gửi log đó đến Loki để lưu trữ và phân tích. Điều này làm cho việc giám sát và xử lý log trở nên dễ dàng và hiệu quả hơn, đặc biệt là khi kết hợp với công cụ hiển thị dữ liệu như Grafana. [4]

6. Terraform

a. 1.1 Tổng quan

Hình 4 Terraform Logo

Terraform là một công cụ mã nguồn mở của HashiCorp, cho phép quản lý cơ sở hạ tầng dưới dạng mã (Infrastructure as Code - IaC). Với Terraform, các tài nguyên được định nghĩa thông qua các file mã, từ đó sẽ tạo ra một đồng nhất kế hoạch thực hiện các thay đổi và triển khai

chúng một cách tự động, giúp tăng cường tính nhất quán và hiệu quả trong quản lý cơ sở hạ tầng. [5]

Một số đặc điểm chính của Terraform bao gồm:

- Khả năng đa nền tảng: Terraform hỗ trợ nhiều nhà cung cấp dịch vụ đám mây và nền tảng cơ sở hạ tầng như AWS, Azure, Google Cloud Platform, và nhiều hơn nữa
- Tự động hóa: Terraform thực hiện tự động hóa việc triển khai, thay đổi và quản lý cơ sở hạ tầng, giúp giảm thiểu lỗi và tiết kiệm thời gian so với cách thực hiện thủ công
- Khả năng mở rộng: Với khả năng mở rộng linh hoạt, Terraform cho phép thay đổi và mở rộng hạ tầng nhanh chóng chỉ bằng cách thay đổi mã cấu hình của nó.

Terraform bao gồm 2 thành phần chính:

- Terraform Core: Đây là thành phần lõi của terraform, nó đọc và xử lý các file cấu hình, quản lý toàn bộ vòng đời hạ tầng. Bao gồm việc lập kế hoạch(plan), triển khai (apply) và quản lý trạng thái (state)
- Terraform Plugins: Mỗi plugin thực hiện các giao tiếp với một cloud provider (Google Cloud) hoặc với các công cụ provisioner (như Bash)

b. 1.2 Kiến trúc hệ thống

1.2.1 Cấu trúc file cấu hình Terraform

a) File provider.tf

```
provider "google" {
  credentials = file("D:/Study/key/norse-case-439506-h4-783c3e3d6177.json")  # Đường dẫn tới file JSON
  project     = "norse-case-439506-h4"  # Đúng ID dự án
  region      = "us-central1"
}
```

Hình 2. File provider.tf

File provider.tf cấu hình kết nối đến GCP thông qua file JSON và project ID. File JSON đã được cấp khi tạo Service Account trong GCP chứa các thông tin xác thực cần thiết để xác định danh tính của tài khoản dịch vụ và các quyền hạn cần thiết để quản lý tài nguyên trên GCP.

b) File firewall.tf

Nó định nghĩa các Rule của Firewall để quản lý quyền truy cập.

```
# Cấu hình SSH Firewall Rule
resource "google_compute_firewall" "allow_ssh" {
  name    = "allow-ssh"
  network = "default"

  allow {
    protocol = "tcp"
    ports    = ["22"]
  }

  source_ranges = ["0.0.0.0/0"]
  target_tags   = ["ssh"]
}
```

Hình 3. Firewall Rule cho SSH

Cấu hình Firewall Rule cho phép lưu lượng qua cổng SSH (port 22) để truy cập vào tài nguyên trên GCP với tags là “ssh”.

```
# Cấu hình HTTP và HTTPS Firewall Rule
resource "google_compute_firewall" "allow_http_https" {
  name    = "allow-http-https"
  network = "default"

  allow {
    protocol = "tcp"
    ports    = ["80", "443"]
  }

  source_ranges = ["0.0.0.0/0"]
  target_tags   = ["k8s"]
}
```

Hình 4. Firewall Rule cho HTTP và HTTPS

Cấu hình Firewall Rule cho phép lưu lượng qua cổng HTTP (port 80) và HTTPS (port 443) với tags “k8s”.

```
# Cấu hình Firewall Rule cho các port của worker nodes
resource "google_compute_firewall" "allow_port-worker" {
  name    = "allow-port-worker"
  network = "default"

  allow {
    protocol = "tcp"
    ports    = ["30080", "30443"]
  }

  source_ranges = ["0.0.0.0/0"]
  target_tags   = ["worker"]
}
```

Hình 5. Firewall Rule cho Worker nodes

Cấu hình Firewall Rule cho phép truy cập vào các dịch vụ thông qua port 30080 và 30043 với giao thức TCP, tags là "worker".

c) File main.tf

Thực hiện tạo các IP tĩnh, instance, và thiết lập lưu trữ trạng thái trên GCS.

```
# Tạo các địa chỉ IP tĩnh cho các node
resource "google_compute_address" "k8s_master_ip" {
  name    = "k8s-master-ip"
  region  = "us-central1"
  address_type = "INTERNAL"
}
resource "google_compute_address" "k8s_worker_ip" {
  count   = 3
  name    = "k8s-worker-${count.index}-ip"
  region  = "us-central1"
  address_type = "INTERNAL"
}
```

Hình 6. Tạo IP tĩnh cho các node

Cấu hình Terraform đã được sử dụng để tạo các địa chỉ IP tĩnh nội bộ (static IP) cho master node và các worker nodes nhằm thuận lợi cho việc kế nối và quản lý

```
# Tạo instance cho master node
resource "google_compute_instance" "k8s_master" {
  name         = "k8s-master"
  machine_type = "e2-medium"
  zone         = "us-central1-a"

  boot_disk {
    initialize_params {
      image = "ubuntu-2204-jammy-v20240927"
    }
  }

  network_interface {
    network       = "default"
    subnetwork    = "default"
    network_ip    = google_compute_address.k8s_master_ip.address  # Gán IP tĩnh cho master node
    access_config {}  # Tạo địa chỉ IP Ephemeral Public
  }

  metadata = {
    ssh-keys = "k8s-admin:${file("D:/Study/id_rsa.pub")}"  # Public key của bạn
  }
```

Hình 7.1 Instance master node

Terraform cấu hình một instance với các thiết lập cần thiết cho master node như: loại máy (e2-medium), hệ điều hành (Ubuntu 2204), gán IP tĩnh, thiết lập khóa SSH để cho phép truy cập từ xa vào master node qua SSH, gán tags Firewall Rule (Hình 7.2).

```
36
37        metadata_startup_script = <<-EOT
38          #!/bin/bash
39          # Tạo user k8s-admin
40          useradd -m -s /bin/bash k8s-admin
41          echo "k8s-admin:admin" | chpasswd
42          usermod -aG sudo k8s-admin
43          echo "k8s-admin ALL=(ALL) ALL" >> /etc/sudoers.d/k8s-admin
44          chmod 0440 /etc/sudoers.d/k8s-admin
45
46          # Cài đặt Python và thư viện Kubernetes
47          sudo apt update
48          sudo apt install -y python3
49          sudo apt install -y python3-pip
50          pip3 install kubernetes
51        EOT
52
53        tags = ["k8s", "ssh"]
54      }
55
```

Hình 7.2 Script startup cho master node

Script này sẽ tạo người dùng quản trị k8s-admin, thiết lập các quyền cần thiết, và cài đặt các công cụ cần thiết như Python và thư viện Kubernetes

```
56    # Tạo instance cho worker nodes
57    resource "google_compute_instance" "k8s_workers" {
58      count        = 3
59      name         = "k8s-worker-${count.index}"
60      machine_type = "e2-medium"
61      zone         = "us-central1-a"
62
63      boot_disk {
64        initialize_params {
65          image = "ubuntu-2204-jammy-v20240927"
66        }
67      }
68
69      network_interface {
70        network       = "default"
71        subnetwork    = "default"
72        network_ip    = google_compute_address.k8s_worker_ip[count.index].address  # Gán IP tĩnh cho worker node
73        access_config {}
74      }
75
76      metadata = {
77        ssh-keys = "k8s-admin:${file("D:/Study/id_rsa.pub")}"
78      }
```

Hình 8.1 Instance các worker node

Tương tự như ở master node nhưng thêm thành phần là ‘count = 3’ đại diện cho việc tạo 3 worker nodes

```
79
80      metadata_startup_script = <<-EOT
81        #!/bin/bash
82        # Tạo user k8s-admin
83        useradd -m -s /bin/bash k8s-admin
84        echo "k8s-admin:admin" | chpasswd
85        usermod -aG sudo k8s-admin
86        echo "k8s-admin ALL=(ALL) ALL" >> /etc/sudoers.d/k8s-admin
87        chmod 0440 /etc/sudoers.d/k8s-admin
88      EOT
89
90      tags = ["k8s", "ssh", "worker"]
91    }
```

Hình 8.2 Script startup cho worker nodes

Script này sẽ tạo người dùng quản trị k8s-admin, thiết lập các quyền cần thiết cho worker nodes. Bên cạnh đó còn thêm tags worker để mở port 30080 và 30443.

```
93    # Cấu hình lưu trữ trạng thái trên GCS
94    terraform {
95      backend "gcs" {
96        bucket  = "k8s_bucket_group10_nt531"  # Tên bucket đã tạo
97        prefix  = "terraform/state"
98      }
99    }
```

Hình 9. Lưu trữ trạng thái trên GCS

Terraform sử dụng file trạng thái (state file) để lưu trữ trạng thái hiện tại của hạ tầng, nó sẽ được lưu trên Google Cloud Storage (GCS) với bucket.

d) File load_balancer.tf

Cấu hình load balancer để phân phối lưu lượng

```
# Health Check cho HTTP
resource "google_compute_health_check" "http_health_check" {
  name                = "k8s-http-health-check"
  check_interval_sec = 5
  timeout_sec         = 5

  http_health_check {
    port = 30080
  }
}

# Health Check cho HTTPS
resource "google_compute_health_check" "https_health_check" {
  name                = "k8s-https-health-check"
  check_interval_sec = 5
  timeout_sec         = 5

  https_health_check {
    port = 30443
  }
}
```

Hình 10. Tạo Health Check

Để đảm bảo các dịch vụ trên các cổng HTTP và HTTPS hoạt động ổn định, ta cần thiết lập Health Check để kiểm tra tình trạng của các dịch vụ này. Health Check giúp load balancer theo dõi trạng thái của các worker nodes và phân phối lưu lượng đến các node đang hoạt động tốt

- port = 30080 cho HTTP: Kiểm tra tình trạng HTTP trên cổng 30080.
- port = 30443 cho HTTPS: Kiểm tra tình trạng HTTPS trên cổng 30443.

```
# Tạo Instance Group cho worker nodes
resource "google_compute_instance_group" "k8s_workers_group" {
  name  = "k8s-workers-group"
  zone  = "us-central1-a"

  named_port {
    name = "http"
    port = 30080
  }

    named_port {
    name = "https"
    port = "30443"
  }

  instances = google_compute_instance.k8s_workers[*].self_link
}
```

Hình 11. Tạo Istance Group

```
# Backend Service cho HTTP
resource "google_compute_backend_service" "k8s_http_backend" {
  name          = "k8s-http-backend"
  protocol      = "HTTP"
  port_name     = "http"
  health_checks = [google_compute_health_check.http_health_check.self_link]
  backend {
    group = google_compute_instance_group.k8s_workers_group.self_link
  }
}

# Backend Service cho HTTPS
resource "google_compute_backend_service" "k8s_https_backend" {
  name          = "k8s-https-backend"
  protocol      = "HTTPS"
  port_name     = "https"
  health_checks = [google_compute_health_check.https_health_check.self_link]
  backend {
    group = google_compute_instance_group.k8s_workers_group.self_link
  }
}
```

Hình 12. Tạo Backend Service cho HTTP và HTTPS

Backend Service là thành phần quan trọng trong cấu hình Load Balancer, chịu trách nhiệm phân phối lưu lượng đến các worker nodes. Ta cấu hình 2 Backend Service riêng biệt cho HTTP và HTTPS, đảm bảo xử lý đúng giao thức và thực hiện kiểm tra tình trạng (Health Check). Mỗi Backend có một Health Check liên kết và Instance Group để phân phối lưu lượng

```
63    # Tạo Managed SSL Certificate
64    resource "google_compute_managed_ssl_certificate" "my_certificate" {
65      name = "k8s-managed-ssl-cert"
66      managed {
67        domains = ["nt531.com"]
68      }
69    }
70
71    # URL Map để định tuyến đến Backend Service cho HTTP
72    resource "google_compute_url_map" "k8s_http_url_map" {
73      name            = "k8s-http-url-map"
74      default_service = google_compute_backend_service.k8s_http_backend.self_link
75    }
76
77    # URL Map để định tuyến đến Backend Service cho HTTPS
78    resource "google_compute_url_map" "k8s_https_url_map" {
79      name            = "k8s-https-url-map"
80      default_service = google_compute_backend_service.k8s_https_backend.self_link
81    }
82
83    # Target HTTP Proxy
84    resource "google_compute_target_http_proxy" "k8s_http_proxy" {
85      name    = "k8s-http-proxy"
86      url_map = google_compute_url_map.k8s_http_url_map.self_link
87    }
88
89    # Target HTTPS Proxy
90    resource "google_compute_target_https_proxy" "k8s_https_proxy" {
91      name             = "k8s-https-proxy"
92      url_map          = google_compute_url_map.k8s_https_url_map.self_link
93      ssl_certificates = [google_compute_managed_ssl_certificate.my_certificate.self_link]
94    }
```

Hình 13. Tạo Managed SSL Certificate, URL Map, và các HTTP/HTTPS Proxy

Terraform được sử dụng để cấu hình Managed SSL Certificate, URL Map, và các HTTP/HTTPS Proxy. Tạo một SSL Certificate do Google quản lý để mã hóa lưu lượng HTTPS dùng cho HTTPS Proxy.

```
96    # Global Forwarding Rule cho HTTP
97    resource "google_compute_global_forwarding_rule" "http_forwarding_rule" {
98      name        = "k8s-http-forward"
99      target      = google_compute_target_http_proxy.k8s_http_proxy.self_link
100     port_range  = "80"
101     ip_protocol = "TCP"
102   }
103
104   # Global Forwarding Rule cho HTTPS
105   resource "google_compute_global_forwarding_rule" "https_forwarding_rule" {
106     name        = "k8s-https-forward"
107     target      = google_compute_target_https_proxy.k8s_https_proxy.self_link
108     port_range  = "443"
109     ip_protocol = "TCP"
110   }
```

Hình 14. Tạo Forwarding

Forwarding Rule giúp định tuyến lưu lượng từ Internet đến các dịch vụ chạy trên hệ thống thông qua Load Balancer. Cấu hình 2 Global Forwarding Rule: một cho HTTP (cổng 80) và một cho HTTPS (cổng 443), giúp phân phối lưu lượng đến các proxy HTTP và HTTPS tương ứng. Sau đó Proxy sẽ chuyển tiếp các yêu cầu HTTP và HTTPS đến URL Map (HTTPS sử dụng chứng chỉ SSL Certificat). URL Map sẽ định tuyết các yêu cầu HTTP và HTTPS đến Backend tương ứng.

e) File output.tf

Xuất các địa chỉ IP để tiện cho việc kết nối.

```
1   # Output địa chỉ IP public của load balancer cho HTTP
2   output "load_balancer_http_ip" {
3     description = "The external IP address of the HTTP load balancer"
4     value       = google_compute_global_forwarding_rule.http_forwarding_rule.ip_address
5   }
6
7   # Output địa chỉ IP public của load balancer cho HTTPS
8   output "load_balancer_https_ip" {
9     description = "The external IP address of the HTTPS load balancer"
10    value       = google_compute_global_forwarding_rule.https_forwarding_rule.ip_address
11  }
12
13  # Output cho IP công khai của các worker nodes
14  output "worker_nodes_public_ips" {
15    value = google_compute_instance.k8s_workers[*].network_interface[0].access_config[0].nat_ip
16    description = "Public IP addresses of the worker nodes"
17  }
```

Hình 15. Output các Ip Public

Hiển thị các IP public của worker nodes và load balancer để thuận tiện cho việc kết nối.

7. Google Cloud Platform (GCP)

a. 2.1 Tổng quan

Hình 5 GCP

Google Cloud Platform (GCP) là một nền tảng cung cấp các dịch vụ điện toán đám mây do Google phát triển, cho phép doanh nghiệp và cá nhân triển khai, quản lý và mở rộng các ứng dụng và dịch vụ trực tuyến. GCP cung cấp một bộ công cụ đa dạng, mạnh mẽ, hỗ trợ nhiều ngôn ngữ lập trình và dễ dàng tích hợp với các dịch vụ khác [6]

Trong đồ án này nhóm sử dụng một số dịch vụ của GCP như:

- Google Compute Engine (GCE): cung cấp khả năng tạo và quản lý các máy ảo (VM) trên GCP
- Google Cloud Storage: được sử dụng để lưu trữ trạng thái (state) của Terraform
- Google Cloud VPC (Virtual Private Cloud): hỗ trợ việc quản lý và kiểm soát luồng lưu lượng mạng cho các tài nguyên trong cùng một mạng nội bộ và với các tài nguyên bên ngoài
- Google Cloud Firewall Rules: được sử dụng để thiết lập các quy tắc cho phép hoặc chặn các lưu lượng truy cập dựa trên giao thức, địa chỉ IP, và cổng
- Google Cloud Load Balancer: giúp phân phối lưu lượng truy cập đến các node để đảm bảo tính sẵn sàng và khả năng mở rộng
- Google Managed SSL Certificate: Mã hóa lưu lượng HTTPS.

2.2 Kiến trúc hệ thống

2.2.1 Một số dịch vụ phổ biến của GCP

a) Compute Engine

Vai trò: Được sử dụng để tạo các instance máy ảo cho các node bao gồm master node và các worker nodes.

Cấu hình: Terraform tạo các instance với các cấu hình xác định và địa chỉ IP cố định với các tags Firewall Rule.

b) Cloud Storage

Vai trò: Lưu trữ trạng thái của Terraform, giúp quản lý và đồng bộ hóa các thay đổi trong cấu hình hạ tầng.

Cấu hình: Terraform sử dụng Cloud Storage làm backend để lưu file trạng thái (state file)

c) Virtual Private Cloud (VPC) và Firewall

Vai trò: VPC cung cấp mạng nội bộ cho các node, và các quy tắc tường lửa (Firewall) kiểm soát lưu lượng truy cập vào hệ thống.

Cấu hình:

- VPC tạo ra một mạng nội bộ cô lập để các node trong hệ thống giao tiếp với nhau một cách an toàn
- Firewall Rules được thiết lập để chỉ cho phép lưu lượng hợp lệ, như truy cập SSH từ bên ngoài vào master node và các kết nối HTTP/HTTPS đến worker nodes

d) Cloud Load Balancing:

Vai trò: Cân bằng tải lưu lượng HTTP và HTTPS đến các worker nodes

Cấu hình: Terraform cấu hình Load Balancer để phân phối đều lưu lượng đến các worker nodes, đảm bảo hệ thống duy trì hiệu suất cao. Load Balancer cũng sử dụng Health Check để giám sát tình trạng của các nodes và chỉ định hướng lưu lượng đến những nodes đang hoạt động tốt.

C. Thiết kế hệ thống

1. Phân tích kỹ thuật

Mô hình giám sát được xây dựng cần đạt được các yêu cầu sau:

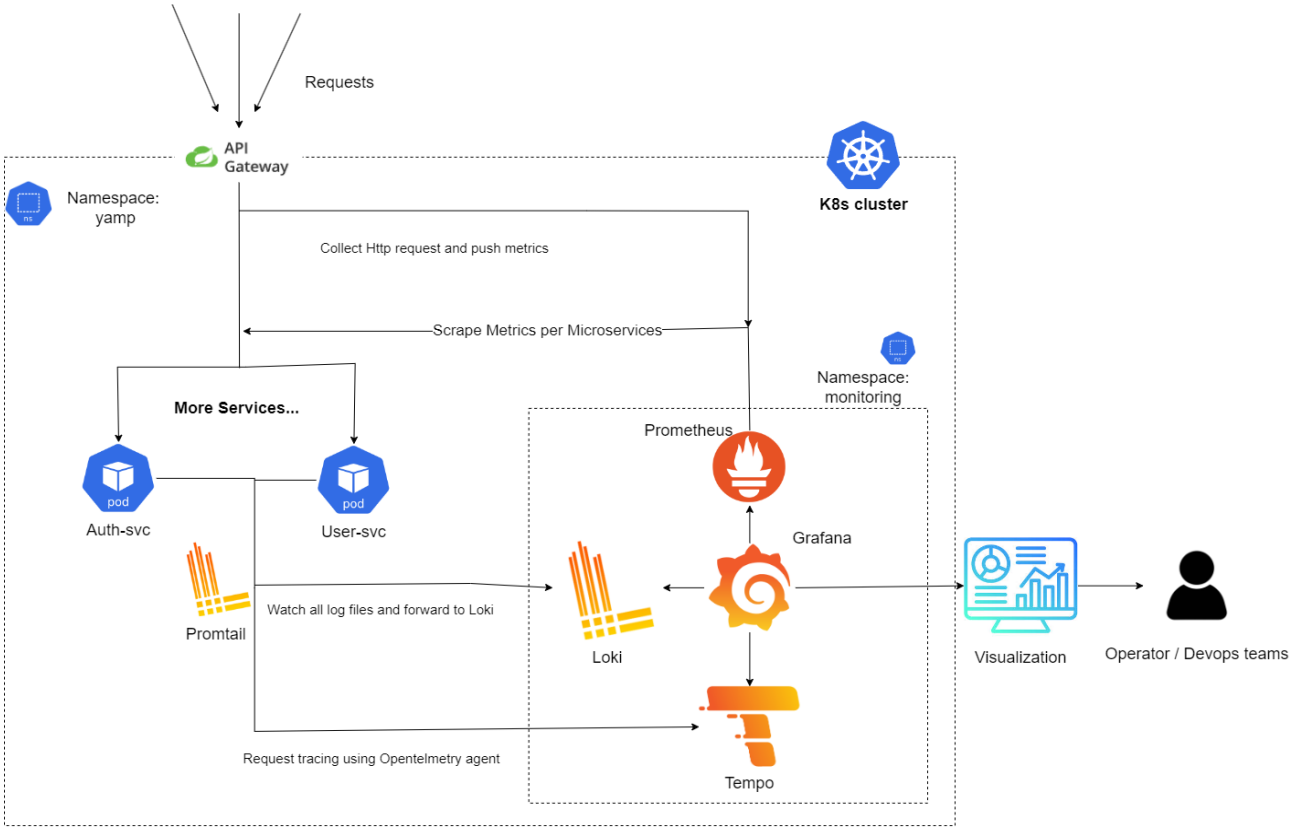
- **Giám sát hiệu năng hệ thống Kubernetes và các microservices:**
  - Hệ thống cần khả năng theo dõi các chỉ số hiệu năng của cụm Kubernetes như CPU, bộ nhớ, dung lượng lưu trữ và trạng thái của từng node và pod trong thời gian thực.
  - Theo dõi các chỉ số từ các dịch vụ microservices như auth-svc, gateway-svc, và user-svc, đảm bảo các dịch vụ hoạt động ổn định, phát hiện kịp thời các vấn đề.
- **Tập trung hóa log và tracing**:
  - Hệ thống cần thu thập và lưu trữ log từ các microservices và Kubernetes cluster để giúp các đội ngũ vận hành dễ dàng tra cứu và xử lý lỗi. Log từ các container và hệ thống phải được chuyển đến một nơi tập trung để dễ quản lý.
  - Hỗ trợ tracing request thông qua OpenTelemetry và Tempo để có thể theo dõi chi tiết các yêu cầu từ đầu đến cuối, cho phép tìm ra nguyên nhân của sự cố và tối ưu hóa hiệu năng.
- **Sử dụng công nghệ phù hợp với quy mô và tính phức tạp**:
  - Các công cụ Prometheus, Grafana, Tempo, Loki và Promtail cần được sử dụng để thiết lập môi trường giám sát thống nhất, cho phép dễ dàng truy vấn, phân tích và hiển thị các dữ liệu metric và log.
  - Cấu hình ServiceMonitor và sử dụng Prometheus Operator để tự động theo dõi các thay đổi trong cluster, cho phép hệ thống dễ dàng mở rộng theo nhu cầu.
- **Hiển thị và phân tích dữ liệu trực quan**:
  - Grafana cần được cấu hình để trực quan hóa dữ liệu log và metric, giúp người dùng dễ dàng nắm bắt tình trạng của hệ thống. Dashboard cần tích hợp metric từ Prometheus và log từ Loki để người dùng có cái nhìn tổng quan và chi tiết về tình trạng hệ thống.
  - Hỗ trợ người dùng truy xuất và phân tích các trace để xác định các bước thực thi của yêu cầu, từ đó tối ưu hóa quy trình xử lý.
  - Cấu hình sharding và replication cho Loki và Prometheus để đảm bảo dữ liệu luôn sẵn sàng, đồng thời tăng cường khả năng chịu lỗi và khôi phục dữ liệu.
- **Khả năng tìm kiếm và phân tích log mạnh mẽ**:
  - Loki kết hợp cùng Promtail phải cho phép gắn nhãn log theo metadata từ Kubernetes như tên namespace, tên pod, và tên container để người dùng dễ dàng tìm kiếm log liên quan đến các dịch vụ cụ thể.

- Đảm bảo khả năng tìm kiếm log nhanh chóng và hiệu quả mà không cần chỉ mục phức tạp, nhờ vào việc tích hợp nhãn và sử dụng ngôn ngữ truy vấn của Grafana.

Từ những yêu cầu trên, nhóm đưa ra hướng tiếp cận của mình như sau:

- **Phân tích yêu cầu và thiết kế kiến trúc tổng thể**:
  - Bắt đầu bằng việc phân tích chi tiết các yêu cầu giám sát, logging và tracing. Dựa trên các yêu cầu đã đưa ra, thiết kế một kiến trúc tổng thể cho hệ thống giám sát, đảm bảo rằng tất cả các thành phần đều được tích hợp một cách hiệu quả.
  - Xác định vị trí và vai trò của từng thành phần như Prometheus, Grafana, Loki, Tempo, và các ứng dụng microservices. Phác thảo một sơ đồ kiến trúc hệ thống để hình dung cách mà các thành phần này tương tác với nhau.
- **Cài đặt và cấu hình các thành phần giám sát**:
  - **Cài đặt Prometheus**: Triển khai Prometheus trên cụm Kubernetes bằng cách sử dụng Helm charts. Cấu hình ServiceMonitor để tự động theo dõi các microservices, thu thập các metric cần thiết.
  - **Triển khai Grafana**: Cài đặt Grafana và cấu hình nó để kết nối với Prometheus và Loki. Tạo dashboard đầu tiên để trực quan hóa dữ liệu metric và log.
  - **Cài đặt Loki**: Thiết lập Loki để thu thập log từ các nguồn khác nhau, sử dụng Promtail để thu thập log từ các pod và container trong cụm Kubernetes.
  - **Triển khai Tempo**: Cài đặt Tempo để hỗ trợ tracing request từ các microservices, giúp theo dõi và phân tích hành trình của các yêu cầu.
- **Cấu hình và tích hợp OpenTelemetry**:
  - Cài đặt OpenTelemetry Java Agent cho các ứng dụng Spring, cấu hình để nó tự động thu thập trace và gửi đến Tempo. Đảm bảo rằng Java Agent có thể can thiệp vào bytecode của ứng dụng để thu thập dữ liệu một cách hiệu quả.
  - Tích hợp các nhãn (label) cho log và trace để dễ dàng phân loại và tìm kiếm thông tin, đảm bảo người dùng có thể truy vấn log và trace theo các tiêu chí cụ thể.
- **Tạo dashboard và báo cáo**:
  - Phát triển các dashboard trong Grafana để hiển thị các chỉ số và log, giúp người dùng theo dõi trạng thái hệ thống một cách trực quan.
  - Thiết lập các cảnh báo trong Prometheus để thông báo cho đội ngũ vận hành khi có sự cố xảy ra, đảm bảo kịp thời xử lý các vấn đề phát sinh.

2. Sơ đồ hệ thống



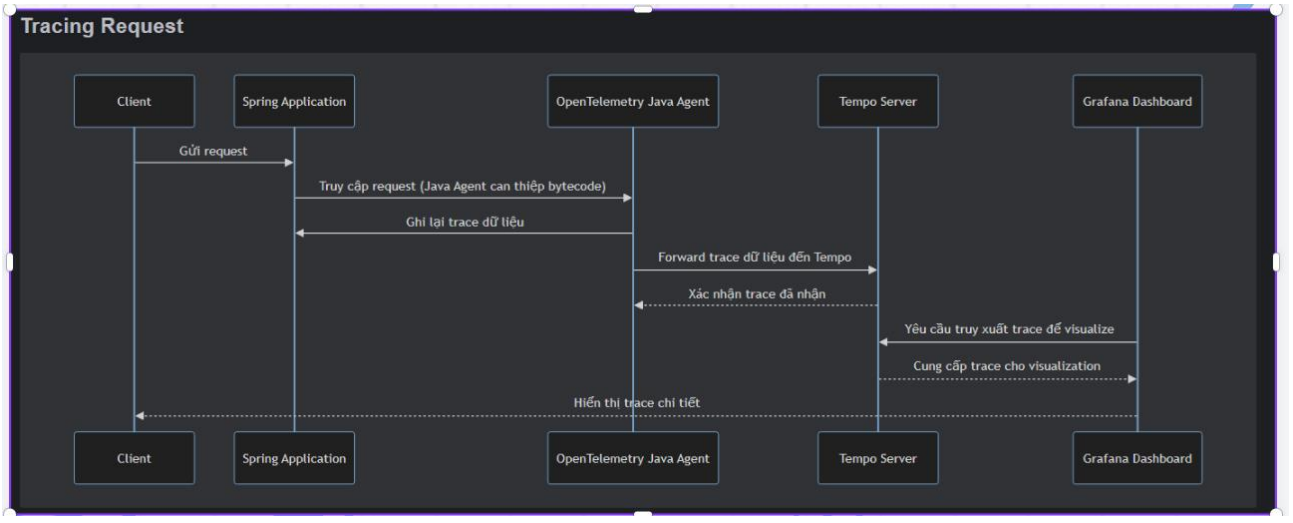
Hình 6. Sơ đồ thiết kế hệ thống

Sơ đồ thiết kế cho hệ thống của đề tài được minh họa bởi hình 1, các thành phần chính của hệ thống gồm

- Các máy ảo chạy trên GCP, một máy ảo chạy Master node, 3 máy ảo chạy worker nodes
- Cụm Kubernetes được dựng bằng tay, không sử dụng GKE (Google Kubernetes Engine)
- Cài đặt hệ thống Monitoring và ứng dụng trên môi trường Kubernetes

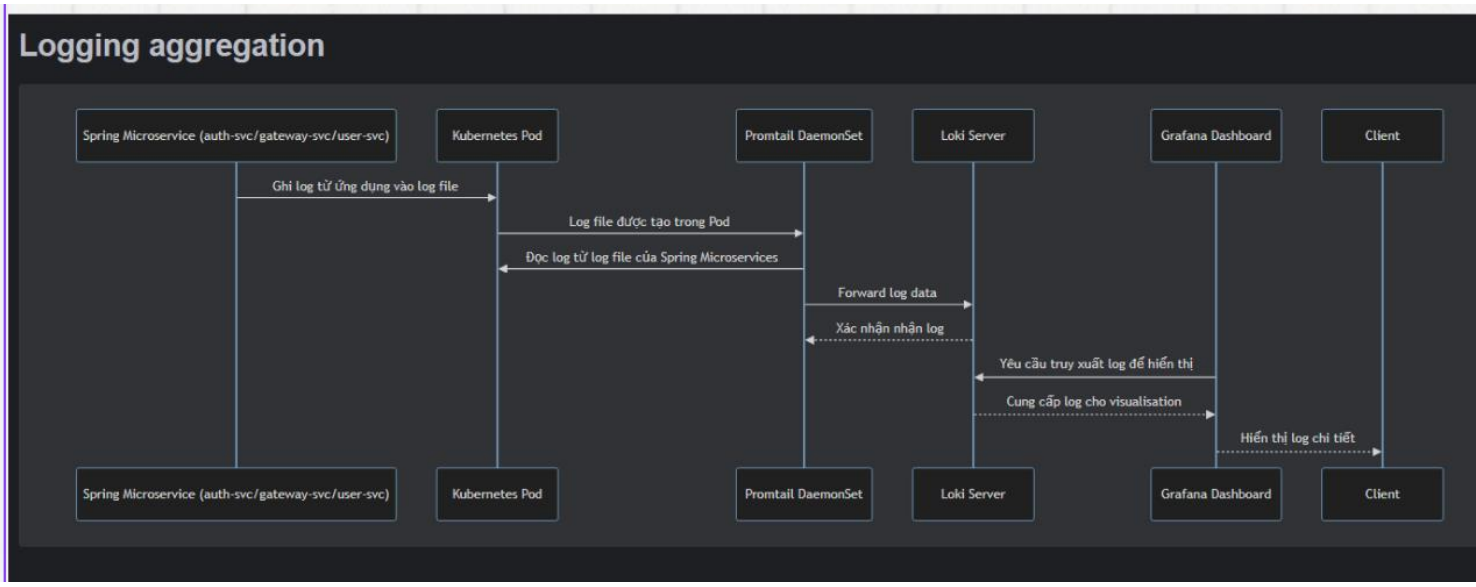
a. Một vài luồng xử lí như sau

Tracing Request



1. Client gửi yêu cầu đến Spring Application: Client khởi tạo một yêu cầu HTTP tới ứng dụng Spring, đây là điểm bắt đầu của quá trình tracing.
2. Spring Application tiếp nhận yêu cầu và OpenTelemetry Java Agent can thiệp: Ứng dụng Spring được cài đặt với Java Agent của OpenTelemetry, cho phép can thiệp vào mã bytecode để ghi lại dữ liệu tracing. Agent tự động thu thập thông tin về yêu cầu.
3. Java Agent ghi lại trace dữ liệu và chuyển tiếp đến Tempo: OpenTelemetry Agent ghi lại chi tiết của request trace bao gồm thông tin như thời gian bắt đầu, kết thúc, trạng thái và các metadata khác. Agent sau đó chuyển tiếp dữ liệu này tới máy chủ Tempo để lưu trữ.
4. Tempo xác nhận trace đã nhận từ Agent: Sau khi nhận được trace, Tempo gửi thông báo xác nhận lại cho Agent để đảm bảo rằng dữ liệu đã được lưu trữ thành công.
5. Grafana truy xuất trace từ Tempo để visualize: Khi người dùng cần xem thông tin chi tiết về trace, Grafana sẽ gửi yêu cầu tới Tempo để truy xuất các trace tương ứng.
6. Tempo cung cấp trace cho Grafana để visualization: Tempo trả lại dữ liệu trace cho Grafana, cho phép Grafana thực hiện quá trình hiển thị và trực quan hóa trace.
7. Grafana hiển thị chi tiết trace cho Client: Cuối cùng, dữ liệu trace sẽ được trình bày trên dashboard của Grafana, cho phép người dùng xem xét và phân tích chi tiết quá trình xử lý yêu cầu trong ứng dụng.

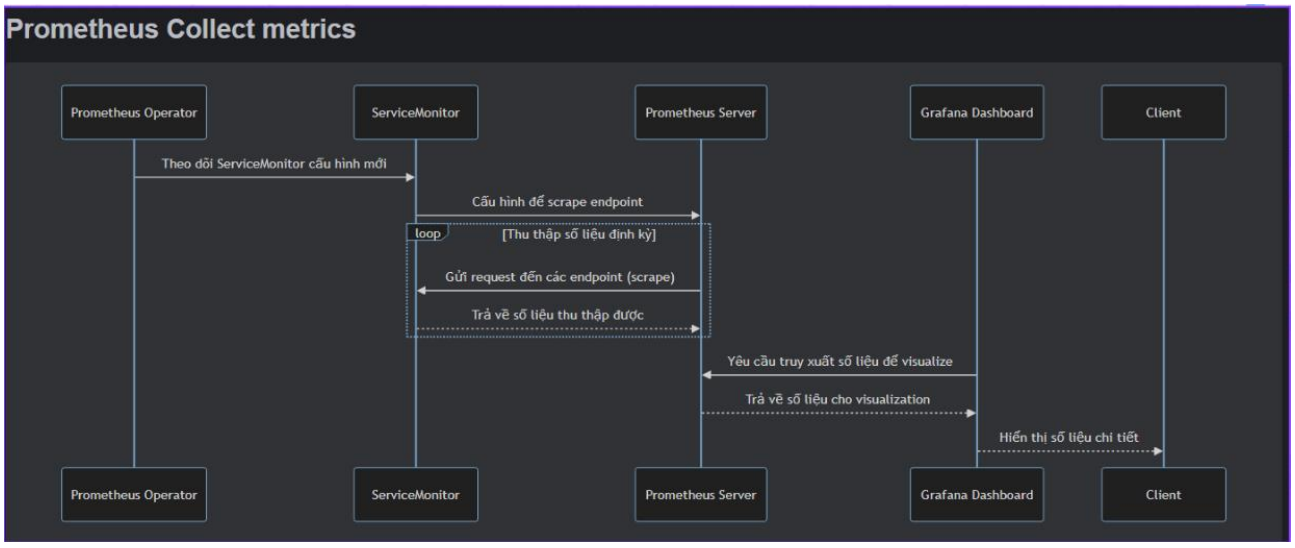
Logging



1. Spring Microservice ghi log vào log file trong Pod: Các dịch vụ Spring (auth-svc, gateway-svc, user-svc) chạy trên Kubernetes và ghi lại thông tin log vào file trong mỗi Pod để lưu trữ các sự kiện và thông tin xử lý.
2. Log file được tạo và lưu trữ trong Pod: File log của ứng dụng Spring được lưu trữ trực tiếp trong môi trường Kubernetes Pod.
3. Promtail đọc log từ file log trong Pod: DaemonSet Promtail, chạy trên cụm Kubernetes, đọc trực tiếp các log từ file log của mỗi Pod, đảm bảo mọi dữ liệu log đều được thu thập.

4. Promtail forward log data tới Loki: Sau khi thu thập, Promtail chuyển tiếp dữ liệu log từ các Pod đến máy chủ Loki để lưu trữ tập trung.
5. Loki xác nhận nhận log từ Promtail: Loki gửi xác nhận đến Promtail để đảm bảo rằng log đã được nhận và lưu trữ thành công.
6. Grafana yêu cầu truy xuất log từ Loki để visualization: Khi cần hiển thị log, Grafana sẽ gửi truy vấn tới Loki để lấy dữ liệu log tương ứng.
7. Loki cung cấp log cho Grafana để trực quan hóa: Loki trả về dữ liệu log để Grafana có thể thực hiện hiển thị.
8. Grafana hiển thị log chi tiết cho Client: Grafana hiển thị các log chi tiết cho người dùng, giúp theo dõi và phân tích các sự kiện hệ thống.

Metrics



1. Prometheus Operator theo dõi các cấu hình mới từ ServiceMonitor: Prometheus Operator kiểm tra cấu hình ServiceMonitor để nhận biết các endpoint mới cần thu thập số liệu (metrics).
2. ServiceMonitor cấu hình Prometheus để scrape endpoint: ServiceMonitor xác định các endpoint từ dịch vụ cần thu thập số liệu, như thông tin về tài nguyên, trạng thái và hiệu suất.
3. Prometheus gửi request đến các endpoint để thu thập số liệu: Trong một vòng lặp định kỳ, Prometheus gửi yêu cầu HTTP tới các endpoint được cấu hình trong ServiceMonitor để thu thập các số liệu từ dịch vụ.
4. ServiceMonitor trả về số liệu thu thập được cho Prometheus: Các endpoint phản hồi lại Prometheus với các dữ liệu về số liệu đã thu thập, giúp Prometheus duy trì một cơ sở dữ liệu cập nhật về hiệu suất và trạng thái của các dịch vụ.
5. Grafana yêu cầu truy xuất số liệu từ Prometheus để visualization: Grafana gửi yêu cầu tới Prometheus để lấy dữ liệu, phục vụ cho việc hiển thị trực quan trên dashboard.
6. Prometheus trả về số liệu để Grafana hiển thị: Prometheus cung cấp dữ liệu đã thu thập cho Grafana, cho phép nó hiển thị các số liệu về hiệu suất và trạng thái của hệ thống.
7. Grafana hiển thị chi tiết số liệu cho người dùng: Cuối cùng, Grafana hiển thị các biểu đồ và số liệu chi tiết cho người dùng để theo dõi và phân tích hiệu suất của hệ thống.

D. Triển khai hệ thống

1. Xây dựng hạ tầng trên Google Cloud Platform (GCP)

Để thuận tiện cho việc xây dựng và quản lý hạ tầng trên GCP, nhóm sử dụng công cụ Terraform

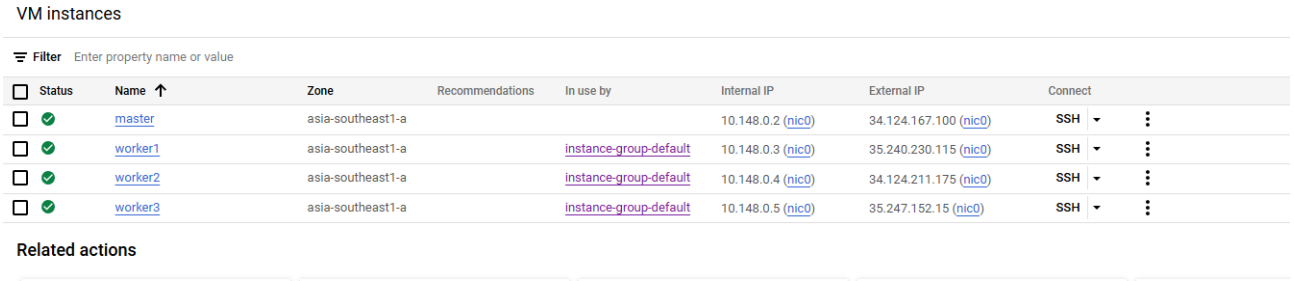
```
akira@Legion:/mnt/d/code/NT531/tf$ tree .
.
├── main.tf
├── modules
│   ├── firewall.tf
│   ├── instances
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── load_balancer.tf
│   ├── output.tf
│   └── provider.tf
└── secrets
    ├── auth-gcp.json
    ├── id-rsa.pub
    └── private-key

3 directories, 11 files
```

Hình 7 Cấu trúc thư mục Terraform

Giải thích:

- main.tf: file quản lí các **resource block**, chứa các cấu hình về máy ảo, tường lửa, …
- variables.tf: file quản lí các **variable block**, chứa biến toàn cục được sử dụng.
- provider.tf: file quản lí các **provider block** và **terraform block**, chứa thông tin liên quan đến phiên bản, nhà cung cấp, xác thực, …

Sau khi đã cấu hình xong, sử dụng lệnh *terraform plan* để xem lại cấu hình cũng như kiểm tra lỗi và sử dụng lệnh *terraform apply* để hoàn tất cấu hình. Dưới đây là kết quả sau khi đã hoàn tất cấu hình:

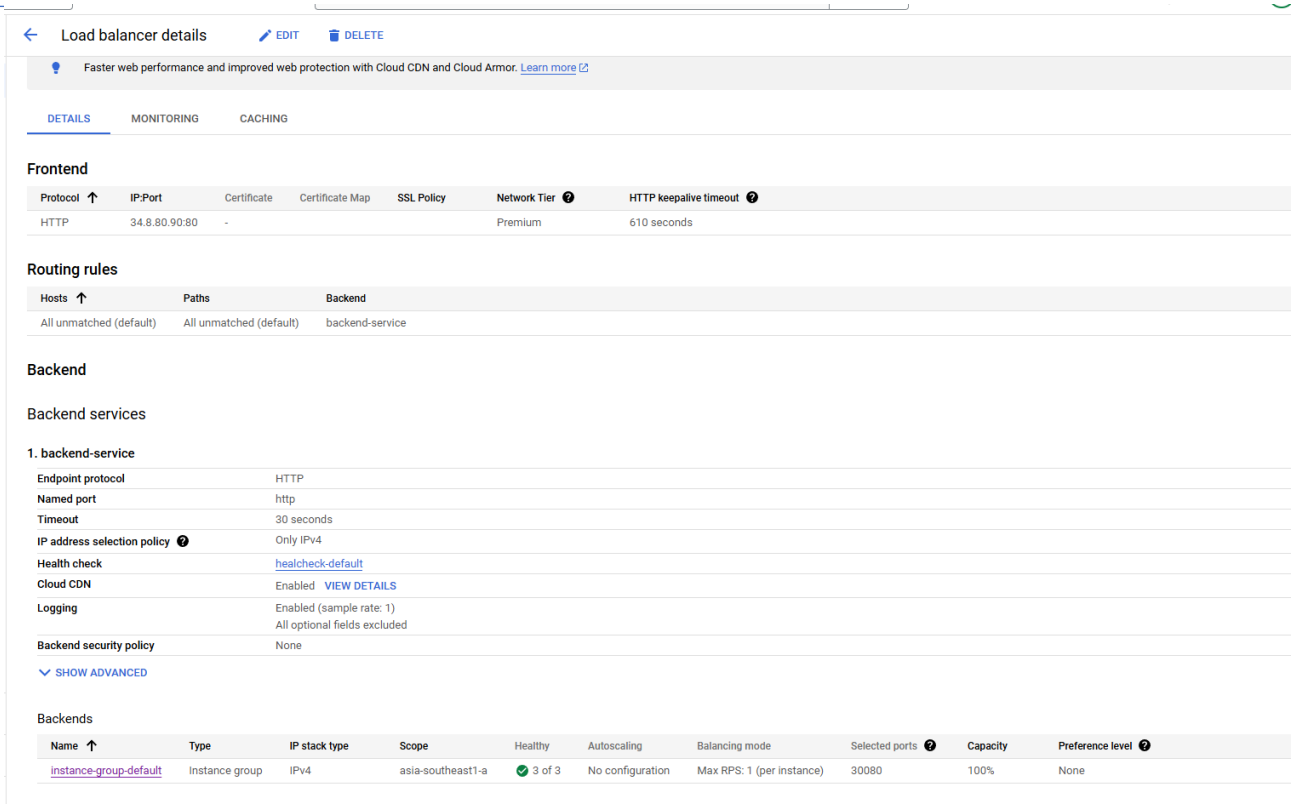

VM instances

Filter Enter property name or value

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	master	asia-southeast1-a			10.148.0.2 (nic0)	34.124.167.100 (nic0)	SSH
✓	worker1	asia-southeast1-a		instance-group-default	10.148.0.3 (nic0)	35.240.230.115 (nic0)	SSH
✓	worker2	asia-southeast1-a		instance-group-default	10.148.0.4 (nic0)	34.124.211.175 (nic0)	SSH
✓	worker3	asia-southeast1-a		instance-group-default	10.148.0.5 (nic0)	35.247.152.15 (nic0)	SSH

Related actions

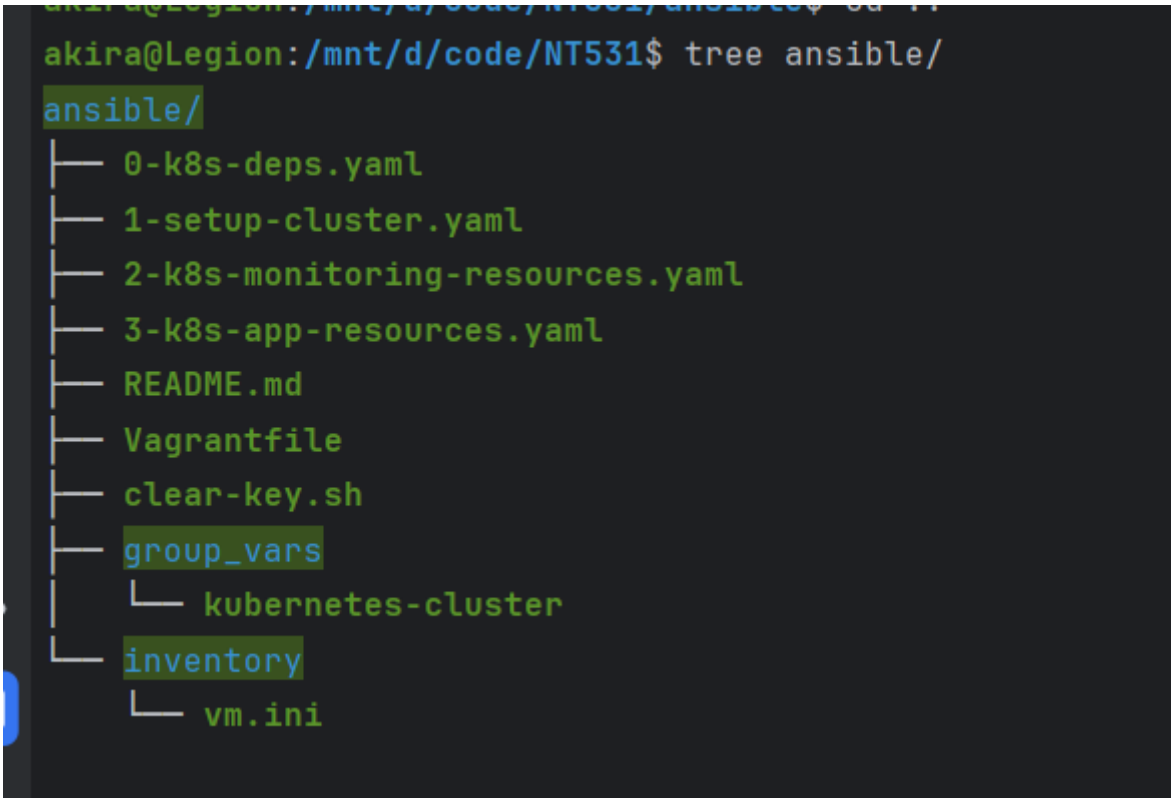
Hình 8 Các máy ảo GCP



Hình 9 Cấu hình Loadbalancer

2. Dựng cụm Kubernetes

Để thuận tiện cho việc cài đặt cụm, nhóm sử dụng dựng thông qua công cụ Ansible



Hình 10 Cấu trúc thư mục Ansible

Hình ảnh sau khi cài đặt thành công

```
PS C:\Users\ngovu> kubectl cluster-info
Kubernetes control plane is running at https://34.124.167.100:6443
CoreDNS is running at https://34.124.167.100:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\ngovu> kubectl get node
NAME      STATUS   ROLES           AGE     VERSION
master    Ready    control-plane   6d10h   v1.29.9
worker1   Ready    <none>          6d10h   v1.29.9
worker2   Ready    <none>          4d10h   v1.29.10
worker3   Ready    <none>          4d10h   v1.29.10
PS C:\Users\ngovu>
```

Hình 11 Kiểm tra cluster

```
PS C:\Users\ngovu> kubectl get pod --all-namespaces
NAMESPACE         NAME                                                READY   STATUS    RESTARTS        AGE
kong              kong-controller-588b484c8b-4lcs8                    1/1     Running   5 (12h ago)     4d1h
kong              kong-gateway-d7f654c4f-757tr                        1/1     Running   2 (12h ago)     4d1h
kube-flannel      kube-flannel-ds-7j7cw                               1/1     Running   8 (12h ago)     6d10h
kube-flannel      kube-flannel-ds-cqqcn                               1/1     Running   3 (12h ago)     4d10h
kube-flannel      kube-flannel-ds-j2pph                               1/1     Running   3 (12h ago)     4d10h
kube-flannel      kube-flannel-ds-zdvvg                               1/1     Running   10 (12h ago)    6d10h
kube-system       coredns-76f75df574-8pq9l                            1/1     Running   2 (12h ago)     4d2h
kube-system       coredns-76f75df574-qrj4z                            1/1     Running   2 (12h ago)     4d2h
kube-system       etcd-master                                         1/1     Running   8 (12h ago)     4d2h
kube-system       kube-apiserver-master                               1/1     Running   8 (12h ago)     4d2h
kube-system       kube-controller-manager-master                      1/1     Running   7 (12h ago)     4d2h
kube-system       kube-proxy-2fkhf                                    1/1     Running   2 (12h ago)     4d1h
kube-system       kube-proxy-m6q8s                                    1/1     Running   2 (12h ago)     4d1h
kube-system       kube-proxy-t62qc                                    1/1     Running   2 (12h ago)     4d1h
kube-system       kube-proxy-xfs5q                                    1/1     Running   2 (12h ago)     4d1h
kube-system       kube-scheduler-master                               1/1     Running   8 (12h ago)     4d2h
longhorn-system   csi-attacher-57689cc84b-7qjfv                       1/1     Running   26 (12h ago)    4d10h
longhorn-system   csi-attacher-57689cc84b-jg7vg                       1/1     Running   34 (12h ago)    5d23h
longhorn-system   csi-attacher-57689cc84b-mcp79                       1/1     Running   30 (8h ago)     4d10h
longhorn-system   csi-provisioner-6c78dcb664-fbmv2                    1/1     Running   35 (12h ago)    5d23h
longhorn-system   csi-provisioner-6c78dcb664-kbbfq                    1/1     Running   27 (12h ago)    4d10h
longhorn-system   csi-provisioner-6c78dcb664-mrxs4                    1/1     Running   36 (12h ago)    5d23h
longhorn-system   csi-resizer-7466f7b45f-5wgr7                        1/1     Running   28 (12h ago)    4d10h
longhorn-system   csi-resizer-7466f7b45f-ntkw4                        1/1     Running   26 (12h ago)    4d10h
longhorn-system   csi-resizer-7466f7b45f-p75zj                        1/1     Running   35 (12h ago)    5d23h
longhorn-system   csi-snapshotter-58bf69fbd5-7rb77                    1/1     Running   34 (12h ago)    5d23h
longhorn-system   csi-snapshotter-58bf69fbd5-dqmsv                    1/1     Running   28 (12h ago)    4d10h
longhorn-system   csi-snapshotter-58bf69fbd5-z2b7m                    1/1     Running   26 (12h ago)    4d10h
longhorn-system   engine-image-ei-5cefaf2b-hzqx6                      1/1     Running   3 (12h ago)     4d10h
longhorn-system   engine-image-ei-5cefaf2b-wtqfn                      1/1     Running   3 (12h ago)     4d10h
longhorn-system   engine-image-ei-5cefaf2b-wwpc2                      1/1     Running   6 (12h ago)     5d23h
longhorn-system   instance-manager-13aed4d1d132e443a48c07ff8e4fff12   1/1     Running   0               12h
longhorn-system   instance-manager-4b89c35c2ac91fe2b1c9292be5bae6c2   1/1     Running   0               12h
longhorn-system   instance-manager-f1f39d716d6c2f6284439b2e80b18752   1/1     Running   0               12h
longhorn-system   longhorn-csi-plugin-499mn                           3/3     Running   62 (12h ago)    4d10h
longhorn-system   longhorn-csi-plugin-ql5mh                           3/3     Running   82 (12h ago)    5d23h
longhorn-system   longhorn-csi-plugin-ztdxt                           3/3     Running   51 (12h ago)    4d10h
longhorn-system   longhorn-driver-deployer-7449f56699-tb9tv           1/1     Running   14 (12h ago)    4d10h
longhorn-system   longhorn-manager-5rnjg                              1/1     Running   19 (12h ago)    4d10h
longhorn-system   longhorn-manager-ndzrx                              1/1     Running   7 (12h ago)     5d23h
longhorn-system   longhorn-manager-nlbzv                              1/1     Running   8 (12h ago)     4d10h
longhorn-system   longhorn-ui-655b65f7f9-5qqlk                        1/1     Running   8 (12h ago)     4d10h
longhorn-system   longhorn-ui-655b65f7f9-fhbg9                        1/1     Running   11 (12h ago)    5d23h
```

Hình 12 Kiểm tra các pod của hệ thống

3. Cài đặt

1. Tiến hành cài đặt Prometheus, Grafana, Loki, Promtail, Tempo thông qua các Helm chart sau

helm repo add grafana https://grafana.github.io/helm-charts
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm upgrade --install loki grafana/loki-stack # cài đặt cả Promtail và Loki lên cụm
helm upgrade –install kube-prometheus-stack prometheus-community/kube-prometheus-stack -n monitoring --create-namespace --version 65.3.2
helm install tempo grafana/tempo

```
PS C:\Users\ngovu> kns monitoring
Context "kubernetes-admin@kubernetes" modified.
PS C:\Users\ngovu> kubectl get pod -o wide
NAME                                                     READY   STATUS    RESTARTS      AGE     IP             NODE      NOMINATED NODE   READINESS GATE
S
alertmanager-kube-prometheus-stack-alertmanager-0        2/2     Running   4 (12h ago)   4d2h    10.244.1.52    worker2   <none>           <none>
kube-prometheus-stack-grafana-54499ff47f-h8xtr           3/3     Running   6 (12h ago)   4d2h    10.244.3.37    worker3   <none>           <none>
kube-prometheus-stack-kube-state-metrics-86b64f5db8-8tbdb 1/1    Running   3 (12h ago)   4d2h    10.244.3.36    worker3   <none>           <none>
kube-prometheus-stack-operator-95b65986-76kgm            1/1     Running   2 (12h ago)   4d2h    10.244.1.48    worker2   <none>           <none>
kube-prometheus-stack-prometheus-node-exporter-7d89m     1/1     Running   2 (12h ago)   4d2h    10.148.0.2     master    <none>           <none>
kube-prometheus-stack-prometheus-node-exporter-ft92g     1/1     Running   2 (12h ago)   4d2h    10.148.0.5     worker3   <none>           <none>
kube-prometheus-stack-prometheus-node-exporter-m2xfg     1/1     Running   3 (12h ago)   4d2h    10.148.0.3     worker1   <none>           <none>
kube-prometheus-stack-prometheus-node-exporter-rrcwc     1/1     Running   2 (12h ago)   4d2h    10.148.0.4     worker2   <none>           <none>
loki-stack-0                                             1/1     Running   0             6h59m   10.244.3.67    worker3   <none>           <none>
loki-stack-promtail-dp587                                1/1     Running   0             6h59m   10.244.0.15    master    <none>           <none>
loki-stack-promtail-k5jvk                                1/1     Running   0             6h59m   10.244.2.143   worker1   <none>           <none>
loki-stack-promtail-szxfl                                1/1     Running   0             6h59m   10.244.3.68    worker3   <none>           <none>
loki-stack-promtail-vq2mr                                1/1     Running   0             6h59m   10.244.1.70    worker2   <none>           <none>
prometheus-kube-prometheus-stack-prometheus-0            2/2     Running   4 (12h ago)   4d2h    10.244.1.50    worker2   <none>           <none>
tempo-0                                                  1/1     Running   0             9h      10.244.3.52    worker3   <none>           <none>
PS C:\Users\ngovu>
```

Hình 13 Kết quả sau khi install monitoring stack

2. Cài đặt các ứng dụng Java Spring cũng thông qua helm chart, source code, và helm folder sẽ được đính kèm ở chương E

```
PS C:\Users\ngovu> kns yamp-dev
Context "kubernetes-admin@kubernetes" modified.
PS C:\Users\ngovu> kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS      AGE     IP             NODE      NOMINATED NODE   READINESS GATES
auth-svc-67d8f4d667-4n2s2           1/1     Running   6 (8h ago)    8h      10.244.3.57    worker3   <none>           <none>
auth-svc-67d8f4d667-jfwcn           1/1     Running   5 (8h ago)    8h      10.244.1.65    worker2   <none>           <none>
gateway-svc-549f5dc946-lqwdt        1/1     Running   0             6h54m   10.244.1.72    worker2   <none>           <none>
gateway-svc-549f5dc946-rf9tt        1/1     Running   0             6h56m   10.244.3.70    worker3   <none>           <none>
kafka-svc-controller-0              1/1     Running   0             8h      10.244.3.54    worker3   <none>           <none>
kafka-svc-controller-1              1/1     Running   0             8h      10.244.2.141   worker1   <none>           <none>
kafka-svc-controller-2              1/1     Running   0             8h      10.244.1.60    worker2   <none>           <none>
redis-svc-master-0                  1/1     Running   0             8h      10.244.1.62    worker2   <none>           <none>
user-svc-65ff9595c8-bmsl6           1/1     Running   0             7h26m   10.244.1.68    worker2   <none>           <none>
user-svc-65ff9595c8-rh9fz           1/1     Running   0             7h28m   10.244.3.64    worker3   <none>           <none>
yamp-infra-postgresql-0             1/1     Running   0             8h      10.244.1.61    worker2   <none>           <none>
PS C:\Users\ngovu> helm ls
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                   APP VERSION
auth-svc        yamp-dev        4               2024-10-29 13:53:48.1266789 +0700 +07   deployed        auth-svc-0.0.1          0.0.1
gateway-svc     yamp-dev        2               2024-10-29 15:23:33.0700137 +0700 +07   deployed        gateway-svc-0.0.1       0.0.1
user-svc        yamp-dev        6               2024-10-29 14:51:04.0500708 +0700 +07   deployed        user-svc-0.0.1          0.0.1
yamp-infra      yamp-dev        1               2024-10-29 13:29:22.9646289 +0700 +07   deployed        yamp-infra-0.0.1        0.0.1
PS C:\Users\ngovu>
```

E. Kết luận

Đồ án này đã cung cấp cho nhóm một cái nhìn toàn diện về cách triển khai, cấu hình và quản lý một cụm Kubernetes trên nền tảng máy ảo, với việc tích hợp cân bằng tải và theo dõi hiệu năng hệ thống. Việc xây dựng hệ thống bao gồm thiết lập **Master Node** và các **Worker Nodes** để xử lý ứng dụng, cũng như cấu hình **Load Balancer** nhằm chuyển tiếp lưu lượng từ bên ngoài vào hệ thống Kubernetes, giúp tăng cường hiệu quả và khả năng mở rộng của hệ thống.

Qua quá trình thực hiện, nhóm đã nắm vững:

- **Cách thiết lập và cấu hình cụm Kubernetes** từ đầu đến cuối, bao gồm việc triển khai trên các máy ảo và quản lý lưu lượng với Load Balancer.
- **Cơ chế điều phối và phân chia nhiệm vụ giữa các thành phần của Kubernetes** như Master Node và Worker Nodes, giúp hệ thống hoạt động ổn định và hiệu quả.
- **Kiến thức về cân bằng tải và cấu hình mạng** với khả năng chuyển tiếp lưu lượng giữa các cổng của Load Balancer và Worker Nodes, tối ưu hóa cho cả các kết nối HTTP và HTTPS.

- **Kỹ năng về quản lý, giám sát và mở rộng ứng dụng trong môi trường container**. Việc cấu hình Prometheus và Grafana để giám sát cũng giúp tôi hiểu sâu hơn về quản trị hiệu năng và khả năng phát hiện sự cố.

Các kết quả thực hiện được lưu ở đường dẫn sau:

- Demo: https://drive.google.com/file/d/1uvjyYnjdOweRjWU1TTFE5oE0VH2kUp6-/view?usp=sharing
- Github Spring sourcecode, helm chart: https://github.com/ngodat0103/NT531.git

F. Tài liệu tham khảo

[1] "Overview | Kubernetes," Kubernetes, [Online]. Available: https://kubernetes.io/docs/concepts/overview/. [Accessed 30 10 2024].

[2] "Overview | Prometheus," [Online]. Available: https://prometheus.io/docs/introduction/overview/. [Accessed 30 10 2024].

[3] G. L. |. G. L. Documentation. [Online]. Available: https://grafana.com/docs/loki/latest/. [Accessed 30 10 2024].

[4] "Promtail agent | Grafana Loki," [Online]. Available: https://grafana.com/docs/loki/latest/send-data/promtail/. [Accessed 30 10 2024].

[5] "What is Terraform," [Online]. Available: https://developer.hashicorp.com/terraform/intro. [Accessed 30 10 2024].

[6] "Google Cloud Overview," [Online]. Available: https://cloud.google.com/docs/overview. [Accessed 30 10 2024].