

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

TRƯỜNG ĐỨC HÀO
HỒNG HUY HOÀNG

**TRIỂN KHAI ỨNG DỤNG MICROSERVICES THEO
KIẾN TRÚC ZERO-TRUST TRÊN MÔI TRƯỜNG
HYBRID CLOUD**

**Deploying Microservices Application using Zero Trust Architecture
on a Hybrid Cloud Environment**

CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN

TP. HỒ CHÍ MINH, 2026

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

TRƯỜNG ĐỨC HÀO - 22520407

HỒNG HUY HOÀNG - 23520517

KHÓA LUẬN TỐT NGHIỆP
TRIỂN KHAI ỨNG DỤNG MICROSERVICES THEO
KIẾN TRÚC ZERO-TRUST TRÊN MÔI TRƯỜNG
HYBRID CLOUD

**Deploying Microservices Application using Zero Trust Architecture
on a Hybrid Cloud Environment**

CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN

GIẢNG VIÊN HƯỚNG DẪN
TS. NGUYỄN NGỌC TỰ

TP. HỒ CHÍ MINH, 2026

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Nguyễn Ngọc Tự, giảng viên đã tận tình giảng dạy và hướng dẫn em trong suốt quá trình thực hiện đồ án môn học này. Em xin cảm ơn Thầy đã luôn nhiệt huyết, khắt khe trong chuyên môn nhưng cũng rất cởi mở để giúp sinh viên tiếp cận những công nghệ mới nhất. Những bài học từ Thầy không chỉ giúp em hoàn thành đồ án này mà còn là hành trang quý báu cho định hướng nghề nghiệp sau này. Mặc dù đã rất cố gắng, nhưng do giới hạn về thời gian và kinh nghiệm thực tế, đồ án khó tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp quý báu từ Thầy để hoàn thiện hơn. Em xin kính chúc Thầy dồi dào sức khỏe, hạnh phúc và gặt hái được nhiều thành công hơn nữa trong sự nghiệp trồng người.

**Trương Đức Hào
Hồng Huy Hoàng**

MỤC LỤC

TỔNG QUAN ĐỀ TÀI.....	1
1.1. Giới thiệu.....	2
1.2. ZERO TRUST.....	2
1.3. CÁC NGUYÊN TẮC CỐT LÕI.....	3
1.3.1. Never Trust, Always Verify.....	3
1.3.2. Least Privilege Access (Quyền tối thiểu).....	3
1.3.3. Assume Breach (Giả định bị xâm nhập).....	3
1.3.4. Verify Explicitly.....	3
1.4. ĐẶT VẤN ĐỀ.....	4
1.4.1. Thách thức của mô hình bảo mật truyền thống.....	4
1.4.2 Mục tiêu đồ án.....	4
1.5. PHẠM VI ĐỒ ÁN.....	4
1.5.1. Môi trường triển khai.....	4
1.5.2. Công nghệ sử dụng.....	5
1.6. TÀI SẢN CẦN BẢO VỆ.....	5
1.6.1. Dữ liệu.....	5
1.6.2. Người dùng & Quyền.....	5
Chương 2. KIẾN TRÚC GIẢI PHÁP.....	7
2.1. Network Architecture.....	7
2.1.1. Hybrid Cloud Network.....	7
2.2. Sơ đồ kiến trúc tổng quan.....	8
2.3. Chi tiết các thành phần.....	10
2.3.1. Identity Provider (Keycloak).....	10
2.3.2. Policy Decision Point (OPA).....	11
2.3.3. OAuth2-Proxy (Auth Gateway).....	12
2.3.4. WireGuard VPN.....	12
Chương 3. KỊCH BẢN TRIỂN KHAI.....	13
3.1. Thiết lập Hybrid Cloud Network.....	13
3.1.1. OpenStack Infrastructure.....	13
3.1.2. AWS Infrastructure (Terraform).....	14
3.1.3. WireGuard VPN Setup.....	15
3.2. Kubernetes Cluster Setup (K3s).....	16
3.2.1. K3s Master (OpenStack).....	16
3.2.2. K3s Worker (AWS).....	16
3.3. Istio Service Mesh.....	17
3.4. Keycloak & OAuth2-Proxy.....	18
3.4.1. Keycloak Deployment.....	18

3.4.2. Cấu hình Keycloak Logic.....	19
3.5. Application Deployment.....	19
3.5.1. Demo App (FastAPI - Python).....	19
3.5.2. TKB Service (Node.js - AWS).....	21
Chương 4. DEMO VÀ ĐÁNH GIÁ.....	22
4.1 Network.....	22
4.2 Workloads và Microservices Application.....	23
4.3 Attack Simulation Testing.....	28
4.4 Logging và Monitoring.....	29
4.5 Kết luận.....	31
4.5.1. Đóng góp chính của đồ án.....	31
4.5.2. Hạn chế tồn tại.....	31
4.5.3. Hướng phát triển trong tương lai.....	31

DANH MỤC HÌNH

Hình 2.1: Kiến trúc Hybrid Cloud

Hình 2.2: Authentication Flow

Hình 4.1: AWS Network

Hình 4.2: Openstack Network

Hình 4.3: Node và Pods đang chạy

Hình 4.4: Policy của OPA

Hình 4.5: SPIRE Workload Identity

Hình 4.6: Keycloak Dashboard

Hình 4.7: User Login

Hình 4.8: Website

Hình 4.9: Sinhvien Login

Hình 4.10: Sinhvien Restricted

Hình 4.11: Giangvien access

Hình 4.12: Test Microservices AWS

Hình 4.13: AWS healthcheck

Hình 4.14: Fake JWT

Hình 4.15: Prevent Fake Token

Hình 4.16: Grafana

Hình 4.17: Security Monitoring

Hình 4.18: Loki

Hình 4.19: Prometheus

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Tên đầy đủ	Giải nghĩa
PEP	Policy Enforcement Point	Điểm thực thi chính sách (Nơi chặn hoặc cho phép traffic đi qua)
RBAC	Role-Based Access Control	Kiểm soát truy cập dựa trên vai trò người dùng
SIEM	Security Information and Event Management	Quản lý sự kiện và thông tin bảo mật
ZTA	Zero Trust Architecture	Kiến trúc Zero Trust (Kiến trúc không tin tưởng)

TÓM TẮT ĐỒ ÁN

Trong bối cảnh điện toán đám mây phát triển mạnh mẽ, mô hình Hybrid Cloud đang trở thành xu hướng nhờ tính linh hoạt và tối ưu chi phí. Tuy nhiên, việc kết hợp giữa hạ tầng On-premises và Public Cloud cũng làm bộc lộ những hạn chế của phương pháp bảo mật vành đai truyền thống, đặc biệt là trước các nguy cơ tấn công nội bộ (insider threats) và di chuyển ngang (lateral movement).

Đồ án môn học này tập trung nghiên cứu và triển khai kiến trúc Zero Trust (ZTA) nhằm giải quyết các thách thức trên. Hệ thống được xây dựng dựa trên nguyên tắc cốt lõi: "Never Trust, Always Verify" (Không bao giờ tin tưởng, luôn luôn xác thực), loại bỏ hoàn toàn khái niệm vùng tin cậy mặc định.

Môi trường thực nghiệm được thiết lập trên hạ tầng Hybrid Cloud kết hợp giữa Private Cloud (OpenStack) và Public Cloud (AWS), kết nối bảo mật qua VPN WireGuard. Giải pháp tích hợp các công nghệ Cloud Native bao gồm: Keycloak (Quản lý định danh), OAuth2-Proxy (Identity-aware Proxy), Istio Service Mesh (Vi phân đoạn và mã hóa mTLS) cùng Open Policy Agent - OPA (Thực thi chính sách phân quyền).

Kết quả thực nghiệm cho thấy hệ thống hoạt động ổn định, bắt buộc xác thực và ủy quyền đối với mọi yêu cầu truy cập. Đồ án đã ngăn chặn thành công các kịch bản tấn công giả lập như giả mạo định danh và truy cập trái phép chéo đám mây, khẳng định tính khả thi của Zero Trust trong việc bảo vệ các hệ thống phân tán.

TỔNG QUAN ĐỀ TÀI

1.1. Giới thiệu

Trong kỷ nguyên số hóa hiện nay, xu hướng chuyển dịch hạ tầng sang Cloud (Điện toán đám mây) là tất yếu. Tuy nhiên, thay vì chọn hoàn toàn Public Cloud hoặc Private Cloud, nhiều doanh nghiệp lựa chọn Hybrid Cloud để tận dụng lợi thế của cả hai: sự bảo mật của hạ tầng tại chỗ (On-premises) và khả năng mở rộng linh hoạt của Public Cloud.

Đồ án này tập trung vào việc giải quyết bài toán bảo mật cốt lõi trong môi trường lai ghép này bằng cách thiết kế và triển khai kiến trúc Zero Trust (ZTA). Hệ thống kết hợp giữa OpenStack (đại diện cho Private Cloud) và AWS Singapore Region (đại diện cho Public Cloud). Mục tiêu là xây dựng một hệ thống bảo mật toàn diện, loại bỏ khái niệm "vùng tin cậy" (trusted zone) truyền thống, thay thế bằng cơ chế xác thực và ủy quyền liên tục cho mọi truy cập, bất kể nguồn gốc từ bên trong hay bên ngoài mạng.

1.2. ZERO TRUST

Zero Trust không phải là một công nghệ đơn lẻ, mà là một tư duy chiến lược an ninh mạng. Mô hình này hoạt động dựa trên nguyên tắc "Never Trust, Always Verify" (Không bao giờ tin tưởng, luôn luôn xác minh).

Khác với mô hình bảo mật vành đai (perimeter-based security) truyền thống - nơi hệ thống được ví như một "lâu đài có hào nước", mọi thứ bên trong hào nước đều được coi là an toàn và tin cậy. Zero Trust giả định rằng môi trường mạng luôn thù địch và các mối đe dọa có thể xuất hiện từ bất cứ đâu, kể cả bên trong mạng nội bộ.

1.3. CÁC NGUYÊN TẮC CỐT LÕI

1.3.1. Never Trust, Always Verify

Mọi request đến hệ thống, dù xuất phát từ một Pod nằm ngay cạnh hay từ Internet, đều PHẢI đi qua Keycloak để xác thực danh tính. Không có ngoại lệ (exception), không có đường tắt (bypass path). Ngay cả traffic giao tiếp giữa các microservices nội bộ cũng phải có chứng chỉ mTLS hợp lệ do Istio cấp phát.

1.3.2. Least Privilege Access (Quyền tối thiểu)

Việc cấp quyền không dựa trên sự thuận tiện mà dựa trên nhu cầu tối thiểu để thực hiện công việc. OPA (Open Policy Agent) đóng vai trò là "Thẩm phán" đưa ra quyết định authorization dựa trên các luật (Rego policies) nghiêm ngặt.

Ví dụ: Role giangvien chỉ có quyền GET vào API giảng viên, không được phép truy cập cơ sở dữ liệu hệ thống. Default policy luôn là DENY ALL.

1.3.3. Assume Breach (Giả định bị xâm nhập)

Hệ thống được thiết kế với tư duy bi quan: giả định rằng kẻ tấn công đã vượt qua tường lửa và đang ở trong mạng nội bộ. Do đó:

Traffic phải được mã hóa ngay cả trong internal network (sử dụng mTLS của Istio).

Kết nối giữa 2 cloud phải đi qua đường hầm bảo mật WireGuard VPN.

Sử dụng Network Policies để hạn chế sự di chuyển ngang (Lateral Movement) của kẻ tấn công.

1.3.4. Verify Explicitly

Hệ thống không tin tưởng các thông tin header do người dùng gửi lên. Các headers quan trọng dùng để định danh như x-forwarded-user, x-forwarded-groups phải được thiết lập bởi OAuth2-Proxy (một thành phần

tin cậy) sau khi đã verify JWT token với Keycloak. Mọi header tự phát từ client sẽ bị loại bỏ (stripped).

1.4. ĐẶT VẤN ĐỀ

1.4.1. Thách thức của mô hình bảo mật truyền thống

Vấn đề	Mô tả chi tiết	Hệ quả thực tế
Lateral Movement	Kẻ tấn công sau khi chiếm được 1 máy chủ web (web server) có thể scan và SSH sang database server trong cùng mạng LAN.	Một điểm yếu nhỏ có thể làm lộ toàn bộ dữ liệu hệ thống (Data Breach).
Credential Theft	Sử dụng mật khẩu hoặc khóa API dài hạn (long-lived) dễ bị lộ lọt.	Attacker có thể ám thầm sử dụng quyền của user hợp lệ trong thời gian dài mà không bị phát hiện.
Insider Threats	Nhân viên nội bộ có quyền truy cập quá mức cần thiết hoặc lạm dụng quyền hạn.	Dữ liệu nhạy cảm bị rò rỉ vô tình hoặc cố ý bởi chính người trong tổ chức.
Cloud Challenges	Tài nguyên trên Cloud có địa chỉ IP động, thay đổi liên tục, làm cho firewall dựa trên IP trở nên vô dụng.	Khả năng giám sát (Visibility) bị hạn chế, khó kiểm soát luồng dữ liệu.

1.4.2 Mục tiêu đồ án

STT	Mục tiêu cụ thể	Công nghệ giải quyết
1	Identity-first Access: Chuyển từ bảo mật dựa trên IP sang dựa trên định danh người dùng.	Keycloak (OIDC/OAuth2)
2	Micro-segmentation: Chia nhỏ mạng thành các vùng an toàn cực nhỏ (tới mức từng Pod).	Istio Service Mesh
3	Encryption Everywhere: Mã hóa dữ liệu trên đường truyền (data-in-transit) ở mọi nơi.	mTLS (nội bộ) + WireGuard (liên mạng)
4	Policy-as-Code: Quản lý chính sách bảo mật tập trung, linh hoạt và có thể kiểm thử.	OPA/Rego
5	Observability: Giám sát toàn diện ai đang làm gì, ở đâu.	Prometheus, Grafana, Loki

1.5. PHẠM VI ĐO ÁN

1.5.1. Môi trường triển khai

Hệ thống được xây dựng trên mô hình Hybrid Cloud thực tế:

Private Cloud: OpenStack Lab (Triển khai K3s Master Node).

IP Internal: 10.0.1.185

Floating IP (Public): 172.10.0.190

Public Cloud: AWS Singapore Region (Triển khai K3s Worker Node).

WireGuard Gateway IP: 18.143.117.69 (Elastic IP AWS)

1.5.2. Công nghệ sử dụng

Danh mục	Công nghệ	Mục đích sử dụng chi tiết
Container Orchestrator	K3s	Phiên bản Kubernetes nhẹ, phù hợp cho Edge/Hybrid computing, dễ dàng kết nối qua VPN.
Service Mesh	Istio + Envoy	Quản lý traffic, thực thi mTLS tự động, thu thập metrics và logs.
Identity Provider	Keycloak	Quản lý người dùng tập trung, cấp phát JWT token theo chuẩn OIDC.
Auth Gateway	OAuth2-Proxy	Reverse Proxy đứng trước ứng dụng, chịu trách nhiệm tương tác với Keycloak để xác thực session.
Policy Engine	OPA	Đưa ra quyết định Authorization độc lập với code của ứng dụng (Decoupled Authz).
VPN Tunnel	WireGuard	Tạo đường hầm kết nối Layer 3 an toàn, hiệu năng cao giữa OpenStack và AWS.
IoT	Terraform	Tự động hóa việc khởi tạo hạ tầng trên AWS (VPC, EC2, Security Groups).
Monitoring	Prometheus Stack	Thu thập metrics hệ thống và logs ứng dụng để phục vụ audit và troubleshooting.

1.6. TÀI SẢN CẦN BẢO VỆ

1.6.1. Dữ liệu

Trong hệ thống giáo dục giả lập này, các tài sản dữ liệu được phân loại như sau:

Loại dữ liệu	Mức độ nhạy cảm	Yêu cầu bảo vệ
Thông tin định danh	Cao	Mã hóa lưu trữ (at-rest) và truyền tải (in-transit).
Thông tin xác thực	Rất cao	Sử dụng Token có thời hạn ngắn (Short-lived tokens), không lưu password plaintext.
Dữ liệu học vụ (Điểm, TKB)	Cao	Access control chặt chẽ theo role (Giảng viên xem được tất cả, SV chỉ xem của mình).
Cấu hình hệ thống	Critical	Không lưu cứng (hardcode) trong source code, quản lý qua K8s Secrets.

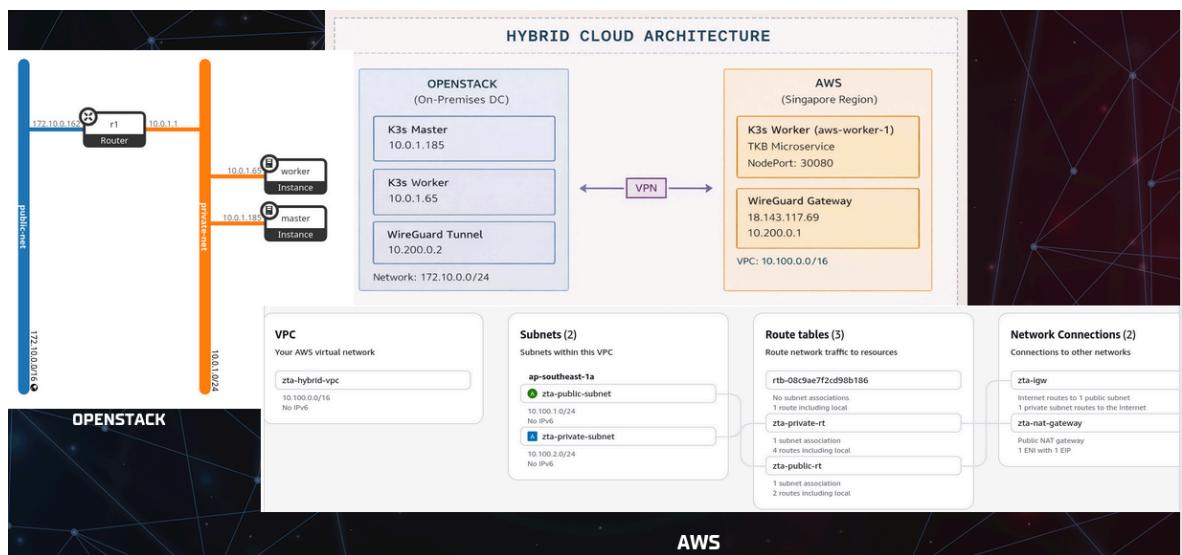
1.6.2. Người dùng & Quyền

Username	Password	Group	Quyền hạn (Permissions)
gv1	gv1	giangvien	Full access to /api/giangvien, /api/sinhvien, /api/tkb.
sv1	sv1	sinhvien	Restricted access. Chỉ được vào /api/sinhvien, /api/tkb.

Chương 2. KIẾN TRÚC GIẢI PHÁP

2.1. Network Architecture

2.1.1. Hybrid Cloud Network



Hình 2.1: Kiến trúc Hybrid Cloud

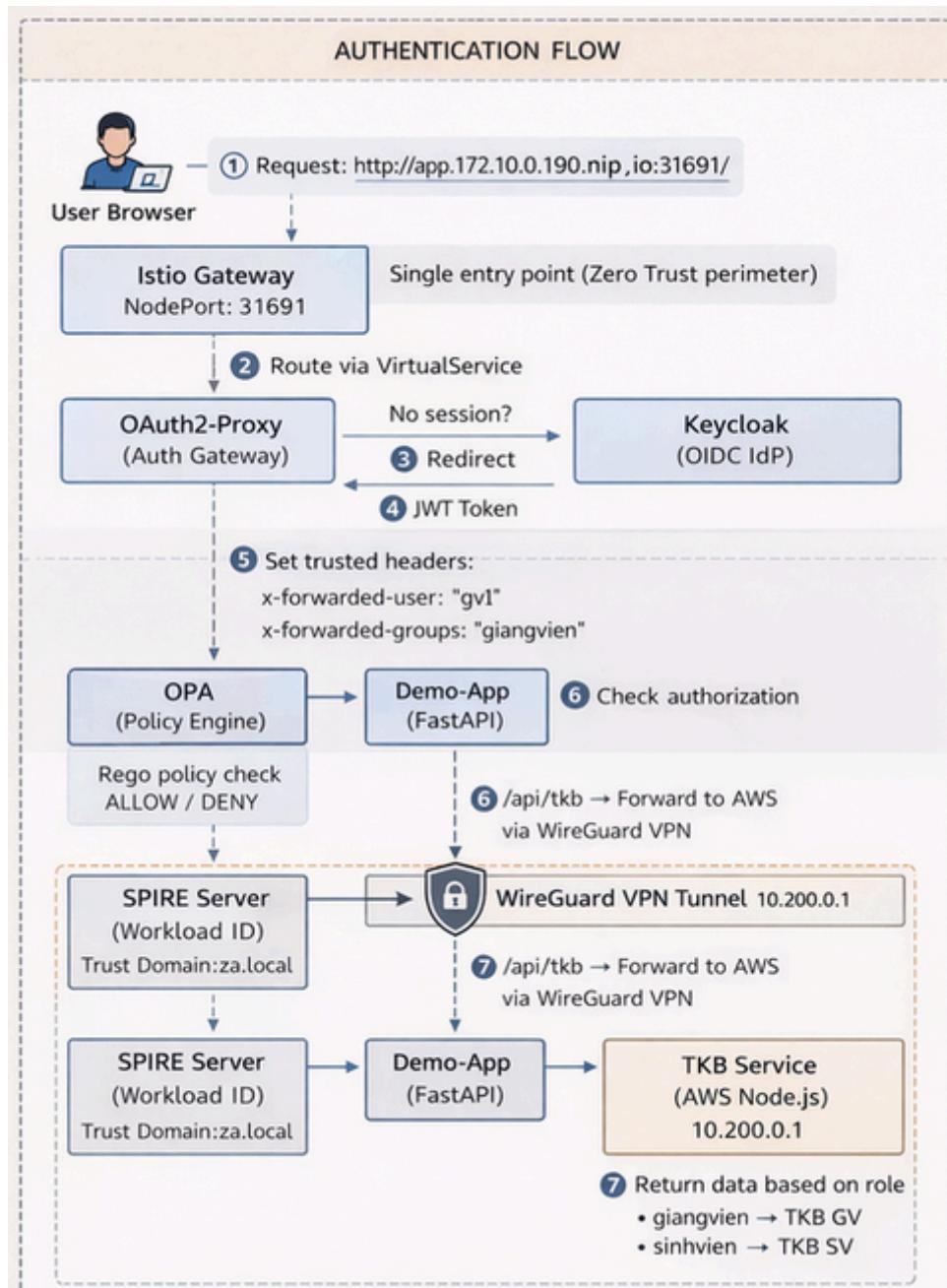
Để giải quyết bài toán kết nối giữa hai môi trường mạng không đồng nhất và nằm sau NAT, đồ án sử dụng giải pháp WireGuard VPN để thiết lập một đường hầm mã hóa (Encrypted Tunnel) an toàn.

Tại OpenStack: Gateway được cấu hình trên một Instance chuyên biệt hoặc tích hợp trực tiếp vào Master Node, đóng vai trò điểm cuối của đường hầm.

Tại AWS: Một WireGuard Gateway được triển khai tại Public Subnet để tiếp nhận kết nối từ OpenStack. Thông qua cấu hình Route Table trên AWS, toàn bộ lưu lượng từ các Worker Node trong mạng nội bộ (Private Subnet) muốn giao tiếp với Master Node đều được định tuyến đi qua Gateway này thay vì đi qua Internet công cộng.

Cơ chế này giúp tạo ra một mạng lớp phủ (Overlay Network) liền mạch, cho phép các thành phần của Kubernetes giao tiếp với nhau như trong cùng một mạng LAN, đảm bảo tính bảo mật và độ trễ thấp.

2.2. Sơ đồ kiến trúc tổng quan



Hình 2.2: Authentication Flow

Kịch bản chi tiết khi một Giảng viên tại Việt Nam truy cập hệ thống để lấy Thời khóa biểu lưu tại AWS Singapore:

- ① USER → GET /api/tkb (Gửi request từ trình duyệt)



② ISTIO GATEWAY (OpenStack) → Nhận request, terminte TLS, Route vào bên trong Cluster.



③ OAUTH2-PROXY → Chặn request lại. Kiểm tra Session. Nếu OK, inject header user/group.



④ DEMO APP (OpenStack Pod) → Nhận request, đọc business logic. Thấy cần gọi sang TKB Service.

Thực hiện gọi: <http://10.200.0.1:30080/api/tkb>



⑤ WIREGUARD INTERFACE (OpenStack VM) → Thấy đích đến là 10.200.x.x, đóng gói packet, mã hóa.



INTERNET (Encrypted Tunnel UDP 51820)



⑥ AWS EC2 INSTANCE → Nhận gói tin UDP tại port 51820. WireGuard giải mã.



⑦ TKB SERVICE (AWS Pod) → Nhận request HTTP (đã giải mã). Kiểm tra header role. Trả về JSON.



⑧ RESPONSE: Dữ liệu đi ngược lại quy trình

⑦→⑥→⑤→④→③→②→①.

Tổng thời gian (Latency): ~58ms

2.3. Chi tiết các thành phần

2.3.1. Identity Provider (Keycloak)

Keycloak đóng vai trò là Single Source of Truth (Nguồn chân lý duy nhất) cho việc quản lý danh tính. Trong đồ án này, Keycloak được cấu hình như một OIDC Provider.

Cấu trúc Realm:

Realm: zta (Vùng quản lý định danh riêng biệt cho đồ án)

```
└── Clients
    └── demo-app
        ├── Type: Confidential (Yêu cầu Client Secret để trao đổi token)
        └── Protocol: OpenID Connect
    └── Users
        ├── gv1 → Mapped to Group: giangvien
        └── sv1 → Mapped to Group: sinhvien
    └── Protocol Mapper
        └── groups claim: Ánh xạ Group của user vào trong payload của JWT Token
```

Cấu trúc JWT Token giải mã:

```
```json
{
 "iss": "http://keycloak.../realms/zta",
 "preferred_username": "gv1",
 "groups": ["giangvien"],
 "exp": 1704534000,
```

```
"iat": 1704533700
```

```
}
```

Giải thích: Token này chứa thông tin user và nhóm, có thời hạn sử dụng (exp). Các dịch vụ phía sau sẽ tin tưởng thông tin này vì nó được ký bởi Keycloak.

### 2.3.2. Policy Decision Point (OPA)

OPA tách biệt logic phân quyền ra khỏi code của ứng dụng. Điều này giúp developer chỉ cần tập trung vào business logic, còn security logic do OPA quản lý.

Luật Rego mẫu:

```
```rego
```

```
package zta.authz
```

Mặc định: Từ chối tất cả

```
default allow := false
```

Định nghĩa ma trận phân quyền

```
role_permissions := {
```

```
    "giangvien": ["/api/giangvien", "/api/sinhvien", "/api/tkb"],
```

```
    "sinhvien": ["/api/sinhvien", "/api/tkb"]
```

```
}
```

Logic cho phép

```
allow {
```

```
    # Lấy danh sách roles từ token
```

```
    some role in input.user.roles
```

```
    # Kiểm tra path hiện tại có nằm trong danh sách cho phép của role đó không
```

```
    input.request.path in role_permissions[role]  
}
```

2.3.3. OAuth2-Proxy (Auth Gateway)

Thành phần này bảo vệ các ứng dụng không có khả năng tự xác thực.

Quy trình Header Flow:

Request arrives: User gửi request đến ứng dụng.

Validate Session: OAuth2-Proxy kiểm tra cookie session.

Nếu KHÔNG có: Redirect user sang Keycloak Login Page.

Nếu CÓ: Kiểm tra tính hợp lệ, refresh token nếu cần.

Sanitize Headers: Xóa bỏ (STRIP) các header x-forwarded-* do user gửi lên để tránh giả mạo.

Inject Headers: Trích xuất thông tin từ JWT Token và set lại vào header x-forwarded-user, x-forwarded-groups.

Forward: Chuyển request đã được làm sạch và dán nhãn định danh xuống backend.

2.3.4. WireGuard VPN

Sử dụng giao thức ChaCha20-Poly1305 để mã hóa, WireGuard tạo ra một "mạng LAN ảo" giữa OpenStack và AWS.

OpenStack (10.200.0.2) UDP Port 51820 AWS (10.200.0.1)

Ưu điểm của WireGuard so với OpenVPN là chạy trong kernel space, cho hiệu năng cao hơn và độ trễ thấp hơn, phù hợp cho môi trường Hybrid Cloud.

Chương 3. KỊCH BẢN TRIỂN KHAI

3.1. Thiết lập Hybrid Cloud Network

3.1.1. OpenStack Infrastructure

Cấu hình mạng nền tảng cho phía Private Cloud.

1. Tạo Network ngoại vi (Provider Network) để ra Internet

```
openstack network create provider-net --external
```

```
openstack subnet create --network provider-net --subnet-range 172.10.0.0/24  
provider-subnet
```

2. Tạo Network nội bộ (Tenant Network) cho các VM giao tiếp

```
openstack network create tenant-net
```

```
openstack subnet create --network tenant-net --subnet-range 10.0.1.0/24  
tenant-subnet
```

3. Tạo Router ảo để kết nối 2 mạng trên

```
openstack router create zta-router
```

```
openstack router set --external-gateway provider-net zta-router
```

```
openstack router add subnet zta-router tenant-subnet
```

4. Khởi tạo VM K3s Master

```
openstack server create --flavor m1.large --image ubuntu-22.04  
--network tenant-net --key-name mykey k3s-master
```

5. Gán Floating IP để truy cập từ ngoài

```
openstack floating ip create provider-net
```

```
openstack server add floating ip k3s-master 172.10.0.190
```

3.1.2. AWS Infrastructure (Terraform)

Sử dụng Terraform để đảm bảo tính nhất quán của hạ tầng (Infrastructure as Code).

Khởi tạo VPC riêng biệt

```
resource "aws_vpc" "zta_vpc" {  
cidr_block = "10.100.0.0/16"  
enable_dns_hostnames = true  
tags = { Name = "zta-hybrid-vpc" }  
}
```

Security Group cho phép WireGuard hoạt động

```
resource "aws_security_group" "wireguard" {  
name = "zta-wireguard-sg"  
vpc_id = aws_vpc.zta_vpc.id
```

Quan trọng: Mở port UDP 51820 cho kết nối VPN

```
ingress {  
from_port = 51820  
to_port = 51820  
protocol = "udp"  
cidr_blocks = ["0.0.0.0/0"] # Chấp nhận kết nối từ mọi IP (hoặc giới hạn IP  
OpenStack)  
}
```

Cho phép traffic nội bộ từ VPN Tunnel và OpenStack Network đi vào

```
ingress {  
from_port = 0
```

```
to_port = 0  
protocol = "-1"  
cidr_blocks = ["172.10.0.0/24", "10.0.1.0/24", "10.42.0.0/16"]  
}  
}
```

3.1.3. WireGuard VPN Setup

Thiết lập đường hầm bảo mật. Bước này yêu cầu tạo cặp khóa Public/Private Key trên mỗi node.

Cấu hình tại OpenStack Node:

Tạo key

```
wg genkey | tee privatekey | wg pubkey > publickey
```

File cấu hình wg0.conf

```
cat > /etc/wireguard/wg0.conf << 'EOF'
```

[Interface]

```
PrivateKey = <PRIVATE_KEY_CỦA_OPENSTACK>
```

Address = 10.200.0.2/24

ListenPort = 51820

[Peer]

```
PublicKey = <PUBLIC_KEY_CỦA_AWS>
```

AllowedIPs định nghĩa routing: Những IP nào sẽ được đẩy qua tunnel này?

Bao gồm: IP Tunnel của AWS, VPC AWS, và Pod Network của K3s trên AWS

AllowedIPs = 10.200.0.1/32, 10.100.0.0/16, 10.42.3.0/24

Endpoint = 18.143.117.69:51820

```
PersistentKeepalive = 25
```

```
EOF
```

Kích hoạt interface

```
systemctl enable --now wg-quick@wg0
```

Cấu hình tại AWS Node: (Tương tự, đảo ngược Peer và Endpoint).

3.2. Kubernetes Cluster Setup (K3s)

3.2.1. K3s Master (OpenStack)

Cài đặt Master Node, tắt Traefik mặc định để dùng Istio sau này.

```
```bash
```

```
curl -sfL https://get.k3s.io | sh -s - server
```

```
--disable traefik --disable servicelb
```

```
--tls-san 172.10.0.190
```

```
--flannel-iface wg0 # Sử dụng interface VPN cho Pod Network
```

```
...
```

#### 3.2.2. K3s Worker (AWS)

Tham gia vào cluster thông qua đường hầm VPN.

```
```bash
```

Quan trọng: node-ip phải là IP VPN để Master có thể gọi lại Worker

```
curl -sfL https://get.k3s.io | K3S_URL=https://10.200.0.2:6443
```

```
K3S_TOKEN=<TOKEN_LAY_TU_MASTER> sh -s - agent
```

```
--node-name aws-worker-1
```

```
--node-ip 10.200.0.1
```

```
--flannel-iface wg0
```

```

Verify kết nối:

```
```bash
```

```
kubectl get nodes -o wide
```

Kết quả mong đợi:

NAME	STATUS	INTERNAL-IP	ROLES
------	--------	-------------	-------

```
master Ready 10.0.1.185 control-plane
```

```
aws-worker-1 Ready 10.200.0.1 <none> <-- IP này phải là IP VPN
```

```

### 3.3. Istio Service Mesh

Cài đặt Istio để quản lý traffic và bảo mật mTLS.

```
```bash
```

1. Cài đặt Istio profile demo

```
istioctl install --set profile=demo -y
```

2. Bật sidecar injection cho namespace demo

```
kubectl create namespace demo
```

```
kubectl label namespace demo istio-injection=enabled
```

3. Kích hoạt mTLS STRICT mode

Đây là bước quan trọng nhất của Zero Trust ở lớp mạng.

Nó bắt buộc mọi traffic giữa các pod phải được mã hóa và xác thực.

```
cat <<EOF | kubectl apply -f -
```

```
apiVersion: security.istio.io/v1beta1
```

```
kind: PeerAuthentication
```

```
metadata:  
  name: default  
  namespace: demo  
  
spec:  
  mTLS:  
    mode: STRICT  
  
EOF  
  
```
```

### 3.4. Keycloak & OAuth2-Proxy

#### 3.4.1. Keycloak Deployment

Triển khai Keycloak bằng container trong K8s.

```
'''yaml

apiVersion: apps/v1

kind: Deployment

metadata:
 name: keycloak

 namespace: demo

spec:
 template:
 spec:
 containers:
 - name: keycloak

 image: quay.io/keycloak/keycloak:22.0
```

```
args: ["start-dev"] # Chạy chế độ Dev cho Lab
env:
- name: KEYCLOAK_ADMIN
 value: "admin"
- name: KEYCLOAK_ADMIN_PASSWORD
 valueFrom:
 secretKeyRef:
 name: keycloak-secret
 key: admin-password
 ...
```

### 3.4.2. Cấu hình Keycloak Logic

Các bước thực hiện trên giao diện Admin Console:

Create Realm: zta

Create Client: demo-app

Access Type: confidential

Valid Redirect URIs: http://app.172.10.0.190.nip.io:31691/oauth2/callback

Create Users: gv1 và sv1.

Create Groups: giangvien và sinhvien.

Add Protocol Mapper: Tạo mapper tên groups để đẩy thông tin Group vào trường groups trong JWT Token.

## 3.5. Application Deployment

### 3.5.1. Demo App (FastAPI - Python)

Ứng dụng đóng vai trò Gateway, kiểm tra Header và điều hướng.

```
```python
from fastapi import FastAPI, Request, HTTPException
import httpx

app = FastAPI()

@app.get("/api/giangvien")
async def giangvien_only(request: Request):
    # Lấy thông tin role từ Header do OAuth2-Proxy inject vào
    groups = request.headers.get("x-forwarded-groups", "")

@app.get("/api/tkb")
async def get_tkb(request: Request):
    # Gọi sang AWS Service qua đường hầm VPN
    # Forward toàn bộ identity headers để AWS Service biết ai đang gọi
    async with httpx.AsyncClient() as client:
        response = await client.get(
            "http://10.200.0.1:30080/api/tkb",
            headers={
                "x-forwarded-user": request.headers.get("x-forwarded-user"),
                "x-forwarded-groups": request.headers.get("x-forwarded-groups")
            }
        )
        return response.json()
```
```

### 3.5.2. TKB Service (Node.js - AWS)

Microservice chạy trên AWS, mô phỏng dữ liệu nhạy cảm.

```
```javascript
```

```
const express = require('express');

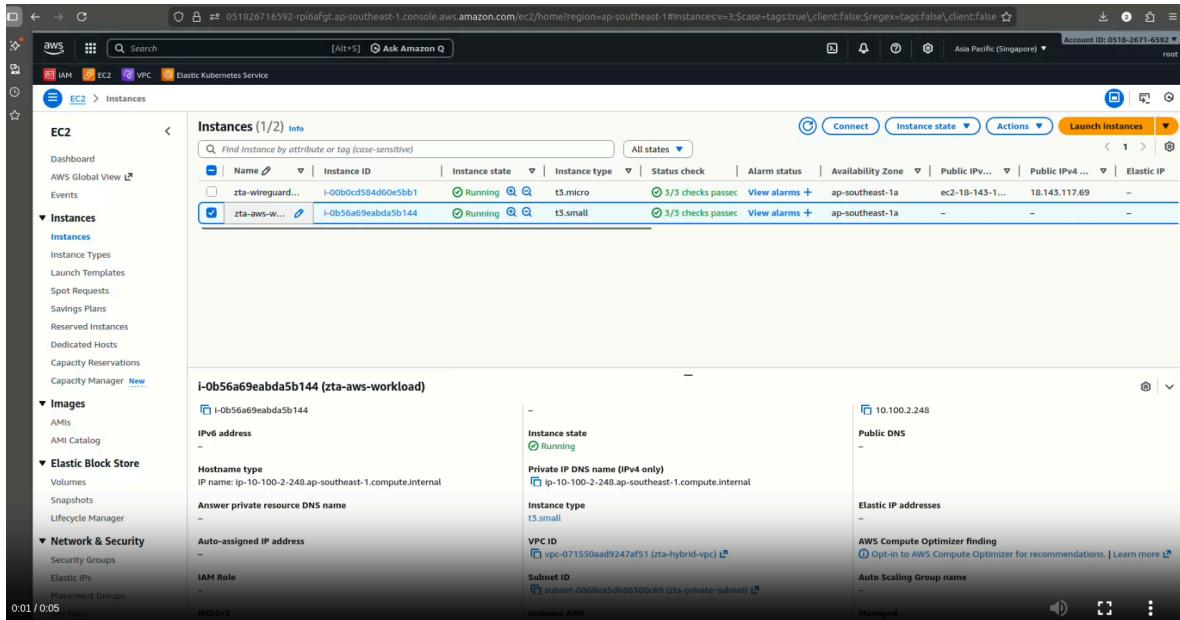
const app = express();

const schedule = {

    giangvien: { "Thứ 2": [ { time: "07:30", subject: "Mạng máy tính", class: "CNTT01" } ] },
    sinhvien: { "Thứ 2": [ { time: "07:30", subject: "Mạng máy tính", teacher: "ThS. Nguyễn Văn A" } ] }
};
```

Chương 4. DEMO VÀ ĐÁNH GIÁ

4.1 Network



The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with options like Dashboard, AWS Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, and IAM Role.

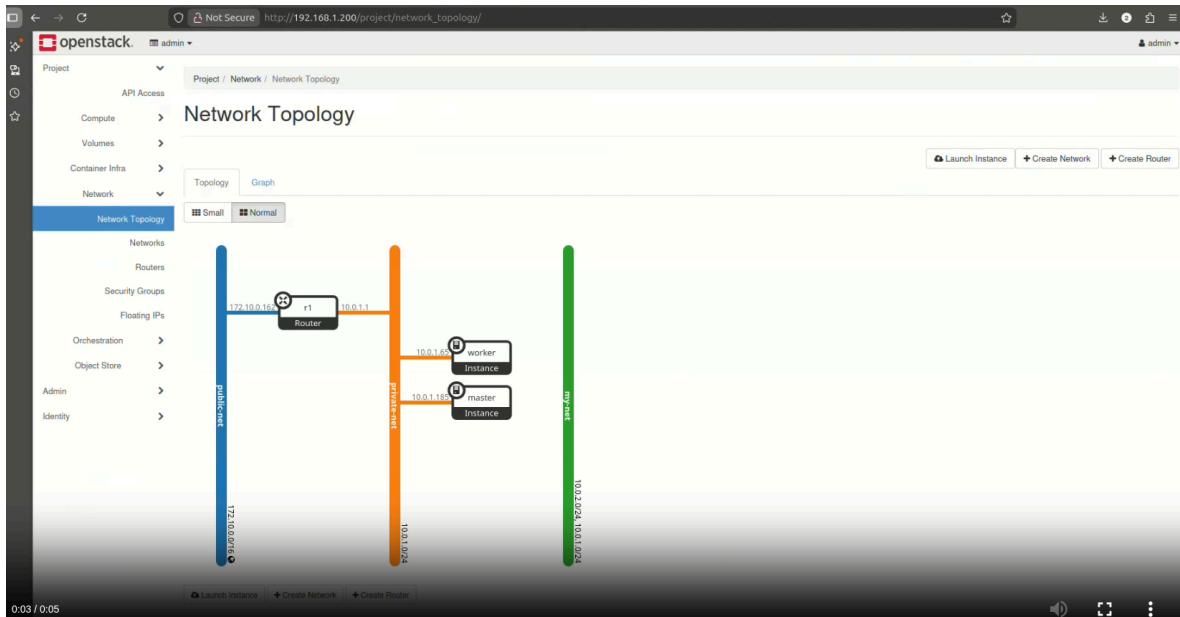
The main area displays two instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP...	Public IPv...
zta-wireguard...	i-00b0cd584d60e5b61	Running	t3.micro	3/3 checks passed	View alarms	ap-southeast-1a	ec2-18-143-1...	18.143.117.69
zta-aws-w...	i-0b56a69eabda5b144	Running	t3.small	3/3 checks passed	View alarms	ap-southeast-1a	-	-

Below the instances, there's a detailed view for the second instance (i-0b56a69eabda5b144):

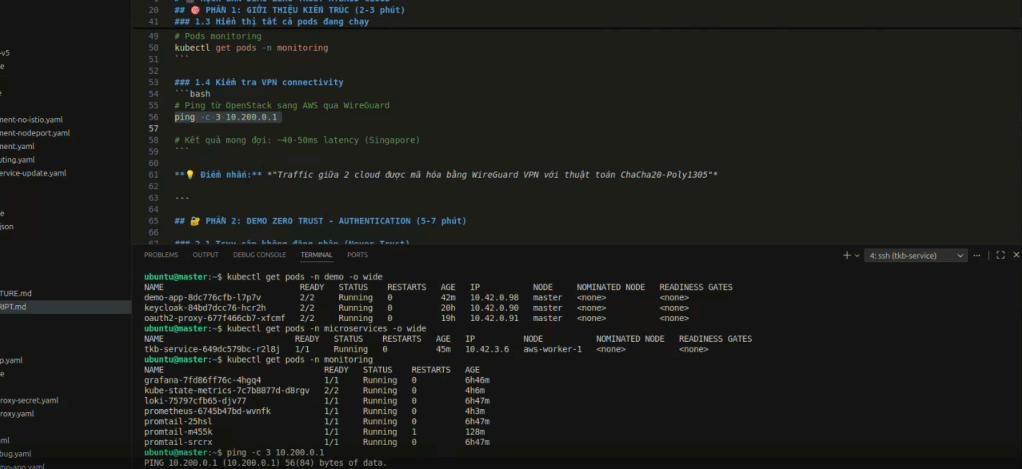
- Images:** i-0b56a69eabda5b144
- IPv4 address:** -
- Hostname type:** Private IP name (IPv4 only)
- IP name:** ip-10-100-2-249.ap-southeast-1.compute.internal
- Answer private resource DNS name:** -
- Auto-assigned IP address:** -
- VPC ID:** vpc-071550aad9247af51 (zta-hybrid-vpc)
- Subnet ID:** subnet-0860ea5d686300c69 (zta-private-subnet)
- Elastic IP addresses:** -
- AWS Compute Optimizer finding:** Opt-in to AWS Compute Optimizer for recommendations. | Learn more
- Auto Scaling Group name:** -

Hình 4.1: AWS Network



Hình 4.2: Openstack Network

4.2 Workloads và Microservices Application



The screenshot shows a terminal window with several tabs open, displaying a complex configuration script for a hybrid cloud environment. The script includes sections for setting up OpenStack, adding AWS components, and configuring pods monitoring and traffic routing between clouds. It also includes sections for authentication and security, such as 'DEMO-ZERO TRUST' and 'AUTHENTICATION'. The terminal interface includes standard Linux tools like `kubectl` and `curl`.

```
... * DEMO-SCRIPT.md * README.md .. $ setup-openstack.sh ... $ add-aws-peer.sh ... $ workload-setup.sh ... $ outbouts.tf ... variables.tf ... main.tf .. $ aws ... $ deploy-all ...  
projectinj > docx > DEMO-SCRIPT.md => KÍCH BẢN DEMO ZERO TRUST HYBRID CLOUD > ... # PHẦN 1: GIỚI THIỆU KIẾN TRÚC (2-3 phút) > ... # PHẦN 1. Kiểm tra VPN connectivity  
20 ## PHẦN 1: GIỚI THIỆU KIẾN TRÚC (2-3 phút)  
41 ## 1.3. Kiểm thử tất cả pods đang chạy  
49 # Pods monitoring  
50 kubectl get pods -n monitoring  
51 ...  
52  
53 ### 1.4. Kiểm tra VPN connectivity  
54 ````bash  
55 # Ping từ OpenStack sang AWS qua WireGuard  
56 ping -c 3 10.200.0.1  
57 ...  
58 # Kết quả mong đợi: ~40-50ms latency (Singapore)  
59 ...  
60  
61 *** ⚡ Điểm nhận: *** Traffic giữa 2 cloud được mã hóa bằng WireGuard VPN với thuật toán ChaCha20-Poly1305***  
62 ...  
63 ...  
64  
65 ## 🌐 PHẦN 2: DEMO ZERO TRUST - AUTHENTICATION (5-7 phút)  
66 ...  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ~ 4:ssh (k8s-service) ... x
```

Hình 4.3: Node và Pods đang chạy

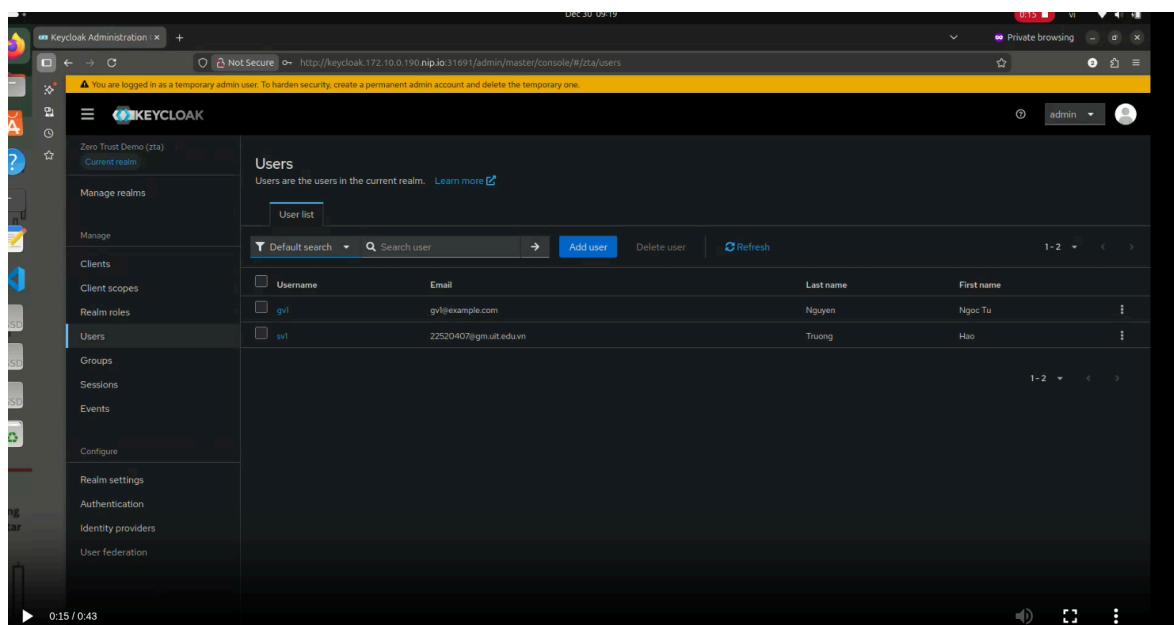
Test ping từ Openstack tới AWS thành công.

Hình 4.4: Policy của OPA

Policy tạo RBAC cho hệ thống từ OPA rego.

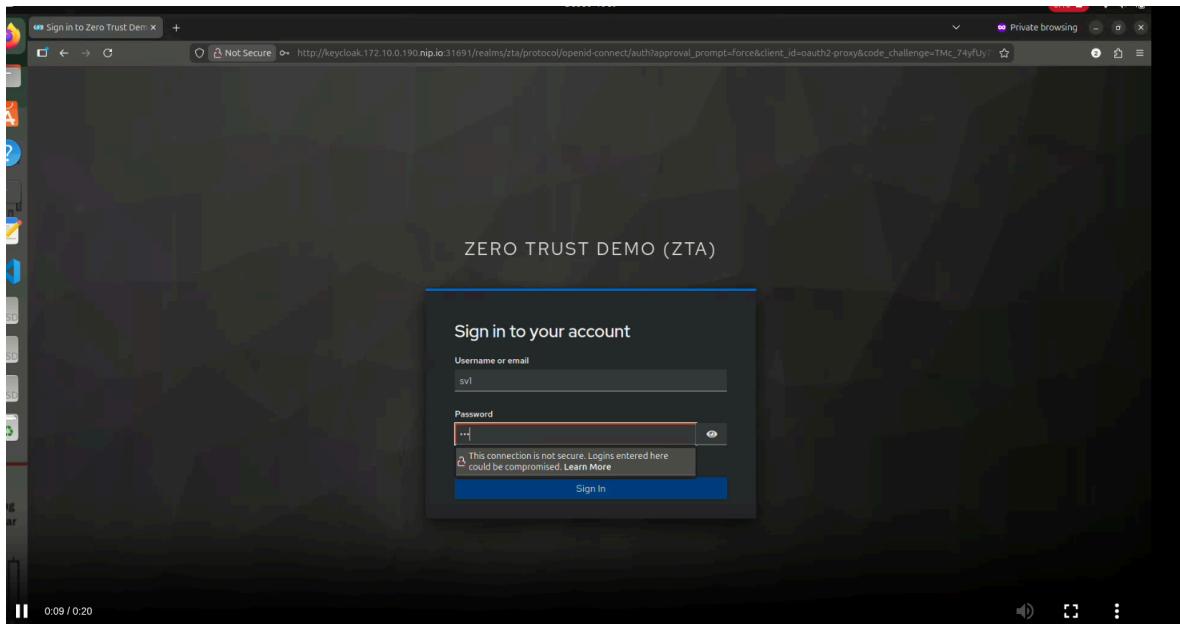
Hình 4.5: SPIRE Workload Identity

Module SPIRE Agent để cung cấp chứng chỉ để xác minh giữa các workloads.

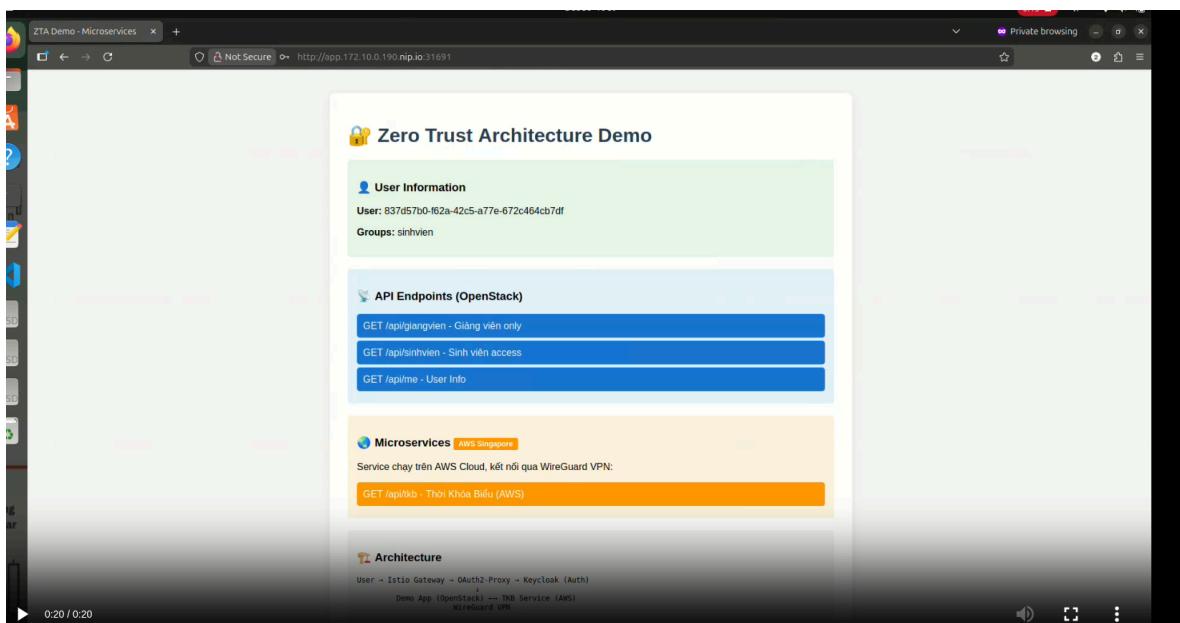


Hình 4.6: Keycloak Dashboard

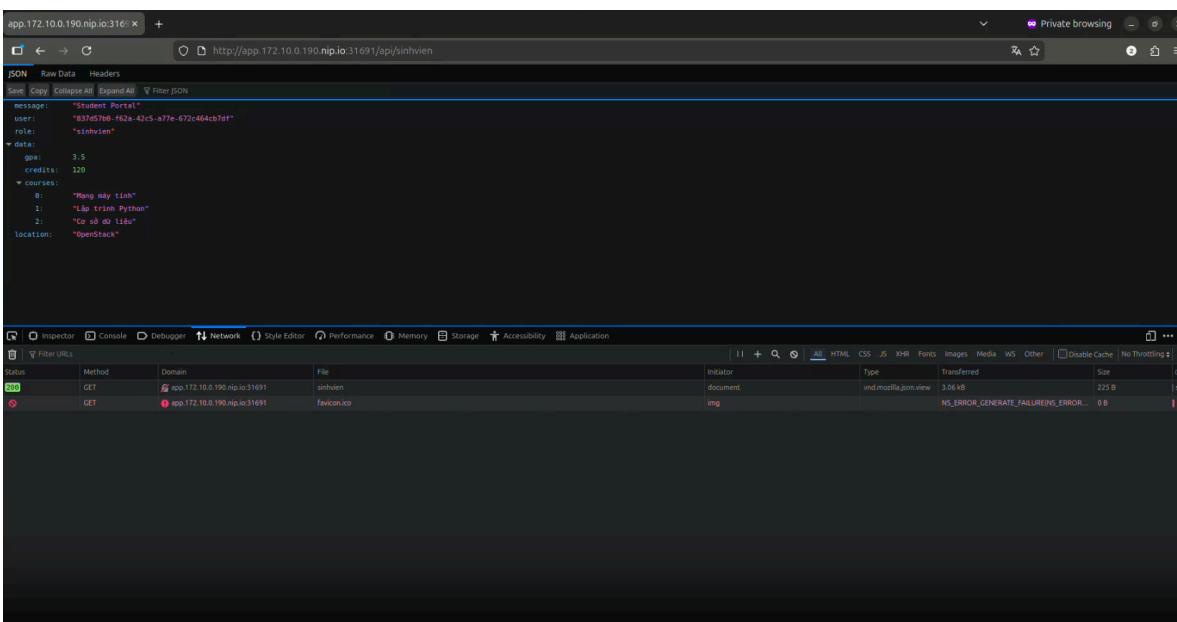
Giao diện Keycloak, module để tạo user và gán role cho user. Và khi đăng nhập vào app, trình duyệt sẽ redirect qua keycloak để xác minh user.



Hình 4.7: User Login

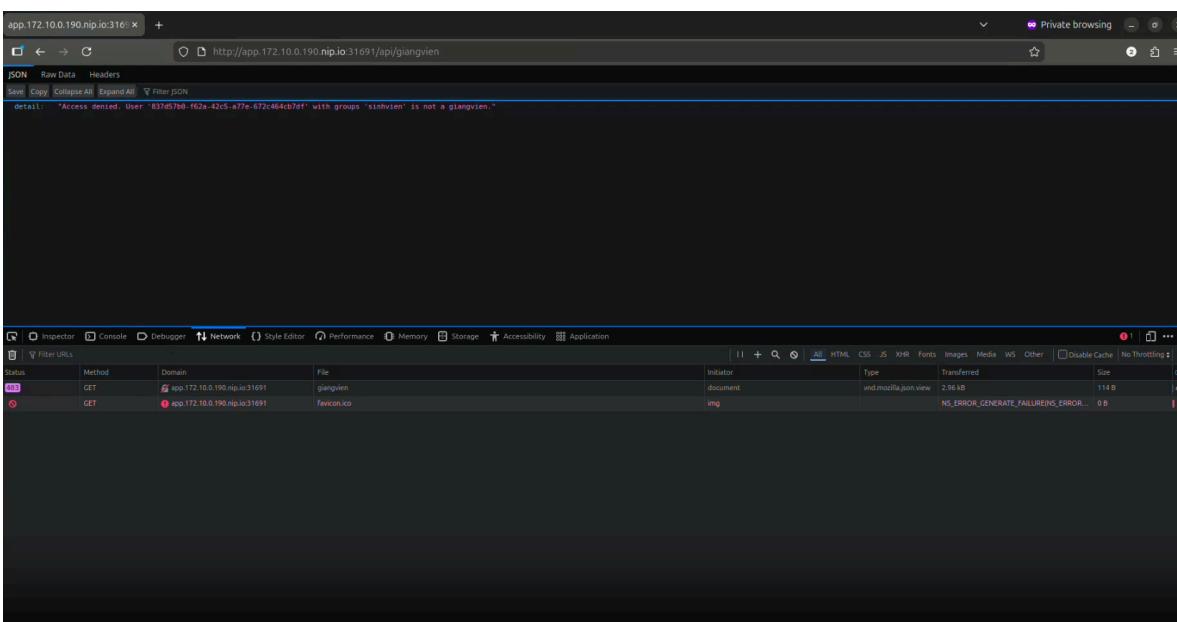


Hình 4.8: Website

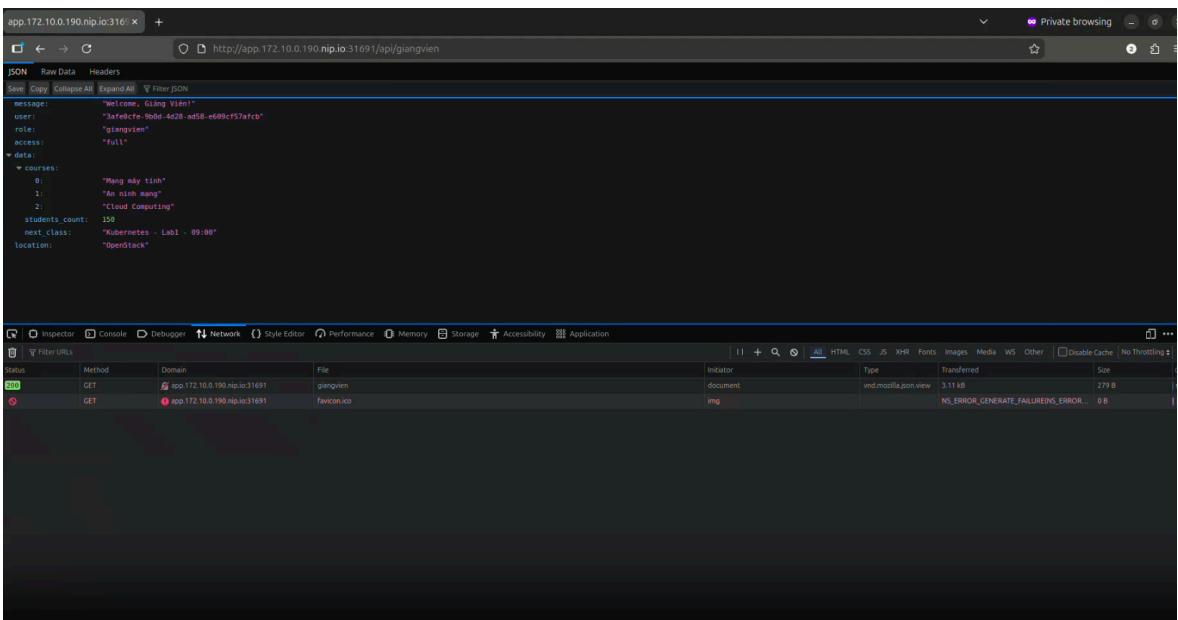


Hình 4.9: Sinhvien Login

Sinh viên thì có thể vào endpoint /sinhvien nhưng không thể vào xem nội dung /giangvien do RBAC.

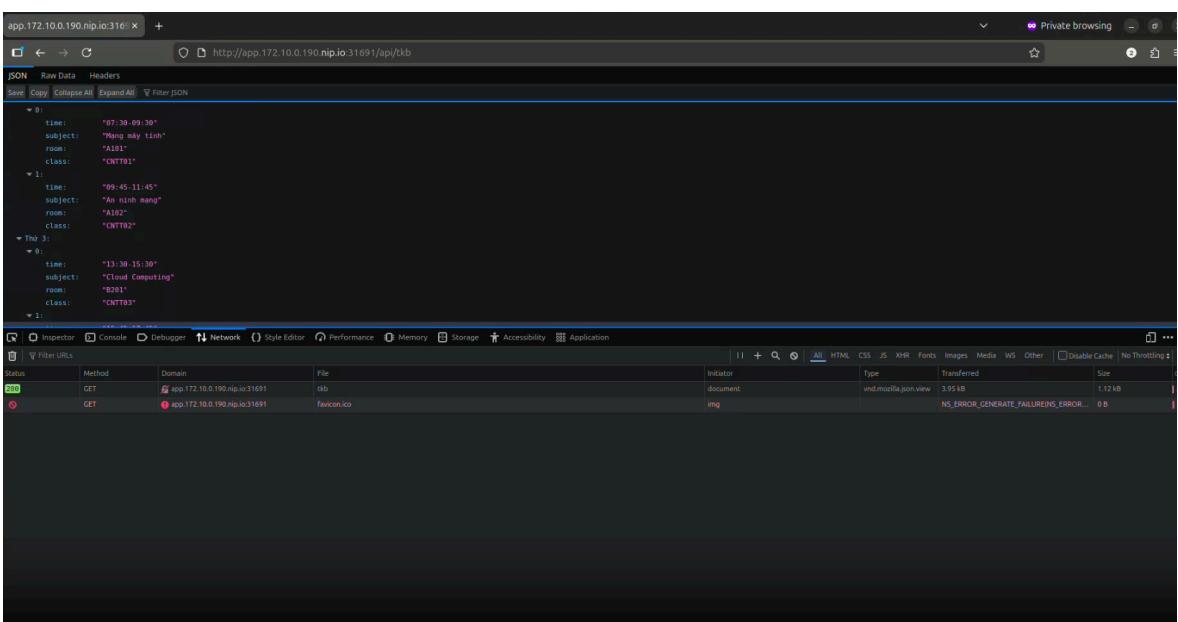


Hình 4.10: Sinhvien Restricted



Hình 4.11: Giangvien access

Nhưng Giangvien thì có thể truy cập endpoint /giangvien để xem nội dung.



Hình 4.12: Test Microservices AWS

Có thể truy cập để xem nội dung thời khóa biểu, chức năng được deploy bên AWS.

The screenshot shows a terminal window with several tabs and multiple command-line sessions. The tabs include:

- README.md
- setup-openstack.sh
- add-laws-peer.sh
- workload-setup.sh
- outputs.tf
- variables.tf
- main.tf
- /aws
- deploy-all.sh

The main session displays the following commands and output:

```
# DEMO-SCRIPT.md x ④ README.md /aws $ setup-openstack.sh ... KICK BẢN DEMO ZERO TRUST HYBRID CLOUD
1 # KICK BẢN DEMO ZERO TRUST HYBRID CLOUD
159 ## PHẦN 4: DEMO HYBRID CLOUD MICROSERVICES (5-7 phút)
284 ### 4.3 Verify TKB chạy trên AWS [Terminal]
287 kubectl get pods -n microservices -o wide
288
289 # Kết quả: tkb-service-xxx chạy trên NODE: aws-worker-1
290
291 # Gọi truy cập tiếp TKB qua WireGuard
292 curl -s http://10.200.0.1:30880/health | jq
293
294
295 **Kết quả:**
296
297 {
298   "status": "healthy",
299   "service": "tkb-service",
300   "version": "1.0.0",
301 }
302
303
ubuntu@master:~$ kubectl get pods -n microservices -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP          NODE      NOMINATED NODE   READINESS   GATES
tkb-service-649dc579c-218   1/1     Running   0          74m   10.42.3.6   aws-worker-1   <none>      <none>
ubuntu@master:~$ curl -s http://10.200.0.1:30880/health | jq
{
  "status": "healthy",
  "service": "tkb-service",
  "location": "AWS Singapore",
  "node": "tkb-service-649dc579c-218",
  "timestamp": "2023-09-07T10:15:10.317Z"
}
ubuntu@master:~$
```

Hình 4.13: AWS healthcheck

Kiểm tra dịch vụ đang chạy trên AWS.

4.3 Attack Simulation Testing

```
EXPLORER PROJECTS ... DEMO-SCRIPT.md README.md _flow_ $ setup-openstack.sh _flow_ $ add-aws-peer.sh $ workload-setup.sh $ outputs.tf $ variables.tf $ main.tf _flow_ $ deploy-all.sh

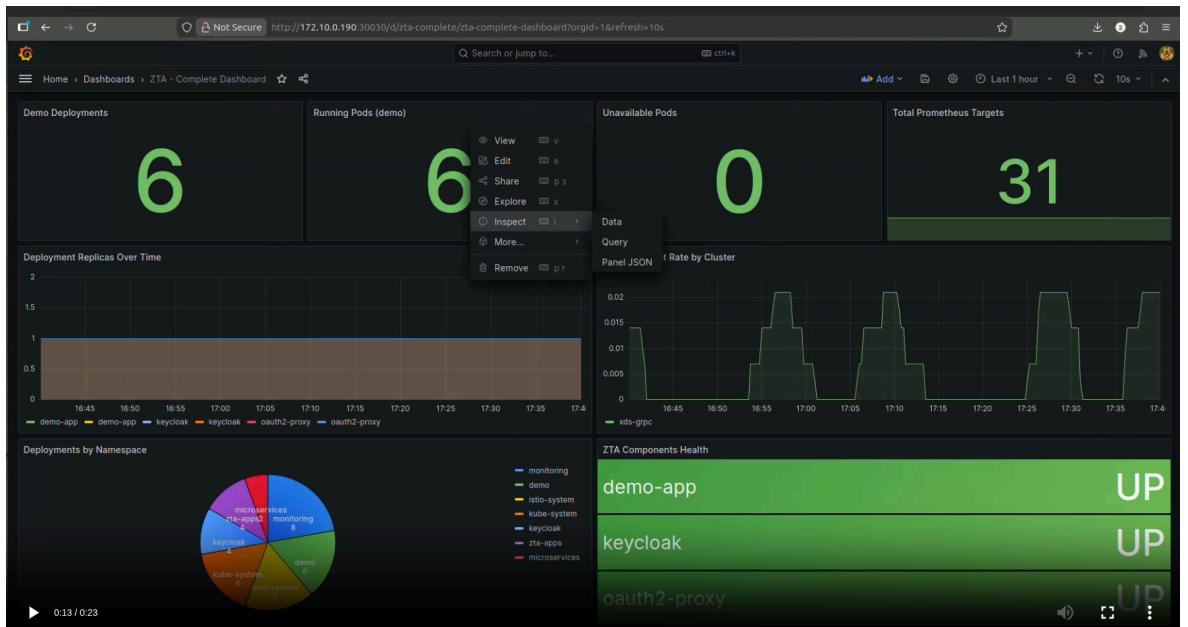
projectfinal> dcos > DEMO-SCRIPT.md > ==> KICK BẢN ĐỀ BẢN DEMO ZERO TRUST HYBRID CLOUD ==> ==> PHẦN S: DEMO ATTACK SIMULATION (5-7 phút) ==> ==> ## 5.2 Fake JWT Token Attack
1 # KICK BẢN ĐỀ BẢN DEMO ZERO TRUST HYBRID CLOUD
228 ## PHẦN 5: DEMO ATTACK SIMULATION (5-7 phút)
230 ### 5.1 Header Injection Attack
242 **! Giảm nhât:** "OAuth2-Proxy" t' ca header x-forwarded-* đến từ bên ngoài. Chỉ sau khi xác thực thành công với Keycloak, các header mới được set bởi OAuth2-Proxy - Không phải từ user"
243
244 #### 5.2 Fake JWT Token Attack
245 ````bash
246 # Tạo fake JWT token
247 curl -v -H "Authorization: Bearer $FAKE_TOKEN" \
248     "http://app.172.10.0.190.nip.io:31691/api/giangvien"
249 ...
250 ...
251 ...
252 ...
253 **Kết quả:** ❌ 302 Redirect to Keycloak
254
255 **! Nhấn nháy:** "Token này không được xác nhận bởi Keycloak nên bị reject. Đầu là nguyên tắc Verify Explicitly"
256
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v 4: sh(bk-service) ... x
ubuntu@master: ~ $ FAKE_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cI6IkXCV39.yJzWlI01JhZGlpbiIsImYdb3VwcyI6MyJhZGlpbi3dfQ.fake"
curl -v -H "Authorization: Bearer $FAKE_TOKEN" \
    "http://app.172.10.0.190.nip.io:31691/api/giangvien"
```

Hình 4.14: Fake JWT

Thử truy cập trang web bằng JWT kí sai thì sẽ bị chặn.

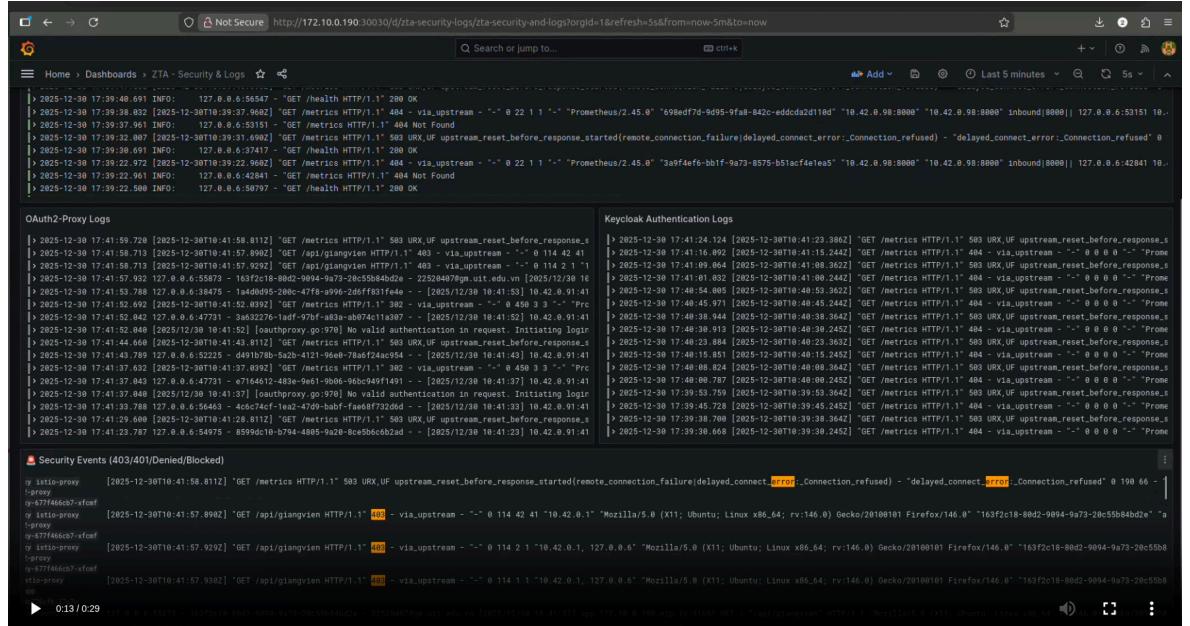
Hình 4.15: Prevent Fake Token

4.4 Logging và Monitoring

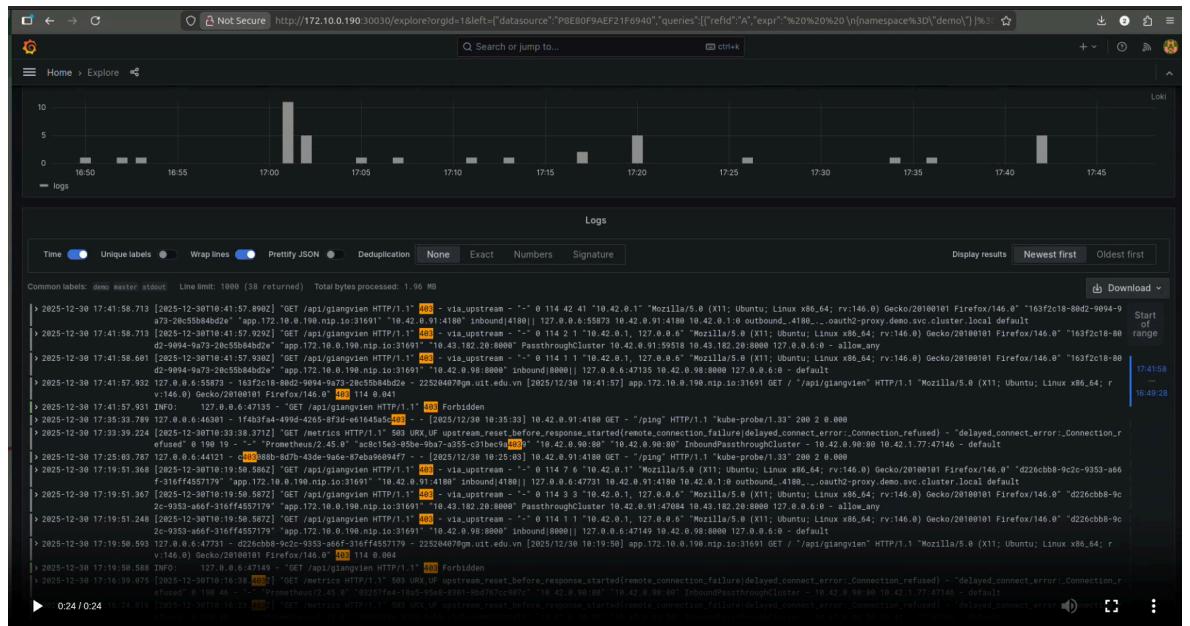


Hình 4.16: Grafana

Sử dụng Grafana để trực quan thông tin về các workloads, pods, services.



Hình 4.17: Security Monitoring



Hình 4.18: Loki

Log được lấy từ module Loki.

Sử dụng Grafana để giám sát các log chặn truy cập từ đó phát hiện bất thường.

```

up(app="demo-app", instance="10.42.0.98.15020", job="kubernetes-pods", kubernetes_namespace="demo", kubernetes_pod_name="demo-app-8d776fb-17p7v", pod_template_hash="8d776fb", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="demo-app", service_istio_io_canonical_revision="latest")
up(app="demo-app", instance="10.42.0.98.15090", job="demo-pods", namespace="demo", pod_name="demo-app-8d776fb-17p7v", pod_template_hash="8d776fb", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="demo-app", service_istio_io_canonical_revision="latest")
up(app="demo-app", instance="10.42.0.98.15090", job="envoy-stats", namespace="demo", pod_name="demo-app-8d776fb-17p7v", pod_template_hash="8d776fb", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="demo-app", service_istio_io_canonical_revision="latest")
up(app="demo-app", instance="10.42.0.98.80", job="demo-pods", namespace="demo", pod_name="demo-app-8d776fb-17p7v", pod_template_hash="8d776fb", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="demo-app", service_istio_io_canonical_revision="latest")
up(app="demo-app", instance="10.42.0.98.9000", job="demo-pods", namespace="demo", pod_name="demo-app-8d776fb-17p7v", pod_template_hash="8d776fb", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="demo-app", service_istio_io_canonical_revision="latest")
up(app="istio-egressgateway", instance="istio", app_kubernetes_io_instance="istio", app_kubernetes_io_name="istio-egressgateway", app_kubernetes_io_managed_by="Helm", app_kubernetes_io_name="istio-egressgateway", app_kubernetes_io_part_of="istio", app_kubernetes_io_version="1.28.1", helm_sh_chart="istio-egress-1.28.1", helm_sh_release="0.0.13", helm_sh_revision="latest", istio_egressgateway_8d6887f9-srb*, operator_istio_io_owners="unknown", instance="10.42.0.53.15020", istio_egressgateway_8d6887f9-srb*, istio_dataplane_mode="none", istio_rev="default", job=kubernetes-pods, kubernetes_namespace="istio-system", kubernetes_pod_name="istio-egressgateway-8d6887f9-srb*", operator_istio_io_component="EgressGateways", pod_template_hash="8d6887f9", release="0.0.13", service_istio_io_canonical_name="istio-egressgateway", service_istio_io_canonical_revision="latest", sidecar_istio_inject="false")
up(app="istio-egressgateway", instance="istio", app_kubernetes_io_instance="istio", app_kubernetes_io_managed_by="Helm", app_kubernetes_io_name="istio-egressgateway", app_kubernetes_io_part_of="istio", app_kubernetes_io_version="1.28.1", helm_sh_chart="istio-egress-1.28.1", helm_sh_release="0.0.13", helm_sh_revision="latest", istio_egressgateway_8d6887f9-srb*, operator_istio_io_owners="unknown", instance="10.42.0.53.15090", istio_egressgateway_8d6887f9-srb*, istio_dataplane_mode="none", istio_rev="default", job=kubernetes-pods, kubernetes_namespace="istio-system", operator_istio_io_component="EgressGateways", pod_template_hash="8d6887f9", release="0.0.13", service_istio_io_canonical_name="istio-egressgateway", service_istio_io_canonical_revision="latest", sidecar_istio_inject="false")
up(app="istio-ingressgateway", instance="istio", app_kubernetes_io_instance="istio", app_kubernetes_io_managed_by="Helm", app_kubernetes_io_name="istio-ingressgateway", app_kubernetes_io_part_of="istio", app_kubernetes_io_version="1.28.1", helm_sh_chart="istio-ingress-1.28.1", helm_sh_release="0.0.13", helm_sh_revision="latest", istio_ingressgateway_64cf74b5b-g7q*, operator_istio_io_owners="unknown", instance="10.42.0.74.15020", istio_ingressgateway_64cf74b5b-g7q*, istio_dataplane_mode="none", istio_rev="default", job=kubernetes-pods, kubernetes_namespace="istio-system", kubernetes_pod_name="istio-ingressgateway-64cf74b5b-g7q*", operator_istio_io_component="IngressGateways", pod_template_hash="64cf74b5b", release="0.0.13", service_istio_io_canonical_name="istio-ingressgateway", service_istio_io_canonical_revision="latest", sidecar_istio_inject="false")
up(app="istio-ingressgateway", instance="istio", app_kubernetes_io_instance="istio", app_kubernetes_io_managed_by="Helm", app_kubernetes_io_name="istio-ingressgateway", app_kubernetes_io_part_of="istio", app_kubernetes_io_version="1.28.1", helm_sh_chart="istio-ingress-1.28.1", helm_sh_release="0.0.13", helm_sh_revision="latest", istio_ingressgateway_64cf74b5b-g7q*, pod_template_hash="64cf74b5b", release="0.0.13", service_istio_io_canonical_name="istio-ingressgateway", service_istio_io_canonical_revision="latest", sidecar_istio_inject="false")
up(app="istio-ingressgateway", instance="istio", app_kubernetes_io_instance="istio", app_kubernetes_io_managed_by="Helm", app_kubernetes_io_name="istio", app_kubernetes_io_part_of="istio", app_kubernetes_io_version="1.28.1", helm_sh_chart="istio-ingress-1.28.1", helm_sh_release="0.0.13", helm_sh_revision="latest", istio_ingressgateway_64cf74b5b-g7q*, pod_template_hash="64cf74b5b", release="0.0.13", service_istio_io_canonical_name="istio-ingressgateway", service_istio_io_canonical_revision="latest", sidecar_istio_inject="false")
up(app="keycloak", component="identity", instance="10.42.0.90.15020", job="kubernetes-pods", kubernetes_namespace="demo", kubernetes_pod_name="keycloak-84bd7dcf76-hc2h", pod_template_hash="84bd7dcf76", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="keycloak", service_istio_io_canonical_revision="latest")
up(app="keycloak", component="identity", instance="10.42.0.90.15090", job="demo-pods", namespace="demo", pod_name="keycloak-84bd7dcf76-hc2h", pod_template_hash="84bd7dcf76", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="keycloak", service_istio_io_canonical_revision="latest")
up(app="keycloak", component="identity", instance="10.42.0.90.15090", job="envoy-stats", namespace="demo", pod_name="keycloak-84bd7dcf76-hc2h", pod_template_hash="84bd7dcf76", security_istio_io_tlsMode="istio", service_istio_io_canonical_name="keycloak", service_istio_io_canonical_revision="latest")

```

Hình 4.19: Prometheus

Sử dụng Prometheus để lấy thông tin từ các namespaces, pods (up, down).

4.5 Kết luận

4.5.1. Đóng góp chính của đồ án

Đã xây dựng thành công một mô hình Zero Trust thực tế, có khả năng áp dụng cho doanh nghiệp vừa và nhỏ sử dụng Hybrid Cloud.

Kết hợp nhuần nhuyễn các công nghệ Cloud Native (K8s, Istio, OPA) để giải quyết bài toán bảo mật phức tạp.

Chứng minh được khả năng ngăn chặn các cuộc tấn công mạng phổ biến thông qua thực nghiệm.

4.5.2. Hạn chế tồn tại

Chưa triển khai Device Posture Check (Kiểm tra sức khỏe thiết bị truy cập - ví dụ: máy có cài antivirus không mới cho vào).

Hệ thống Keycloak hiện tại là Single Instance, chưa có cơ chế High Availability (HA), nếu Keycloak sập thì toàn bộ hệ thống xác thực ngừng hoạt động.

4.5.3. Hướng phát triển trong tương lai

Context-aware Access: Bổ sung luật truy cập dựa trên ngữ cảnh (ví dụ: chỉ cho phép truy cập trong giờ hành chính, hoặc chỉ từ IP Việt Nam).

SIEM Integration: Đẩy logs từ Loki về hệ thống giám sát an ninh tập trung (như Elastic SIEM) để cảnh báo thời gian thực.

TÀI LIỆU THAM KHẢO

1. **NIST SP 800-207 (2020)**, *Zero Trust Architecture*, National Institute of Standards and Technology.
2. **Istio Authors**, *Istio Security Documentation*, <https://istio.io/latest/docs/concepts/security/>
3. **Keycloak Documentation**, *Server Administration Guide*,
<https://www.keycloak.org/documentation>
4. **Styra**, *Open Policy Agent Ecosystem*, <https://www.openpolicyagent.org/docs/>
5. **Jason A. Donenfeld**, *WireGuard: Next Generation Kernel Network Tunnel*,
<https://www.wireguard.com/>