

# Phân tích Vấn đề Traffic Khủng của Hệ thống VNeID

Hệ thống định danh điện tử quốc gia Việt Nam đang đối mặt với thách thức xử lý lượng truy cập đột biến

# Giới thiệu về VNeID

## VNeID là gì?

Ứng dụng **định danh điện tử quốc gia** của Việt Nam

## Chức năng chính

- Thay thế giấy tờ truyền thống
- Định danh công dân trên môi trường số
- Xác thực bằng số CCCD

## Nền tảng

Xây dựng trên cơ sở dữ liệu **định danh, dân cư và xác thực điện tử**



# Vấn đề hiện tại của VNeID

## ⚠ Hệ thống đang chịu traffic khủng

VNeID đang trải qua tình trạng quá tải do lượng truy cập đột biến từ người dân

## ⚠ Hậu quả

- 🚫 Hệ thống chậm hoặc không phản hồi
- ❗ Người dùng gặp lỗi 500
- 🕒 Thời gian xử lý yêu cầu tăng đáng kể

**3-6 tháng**

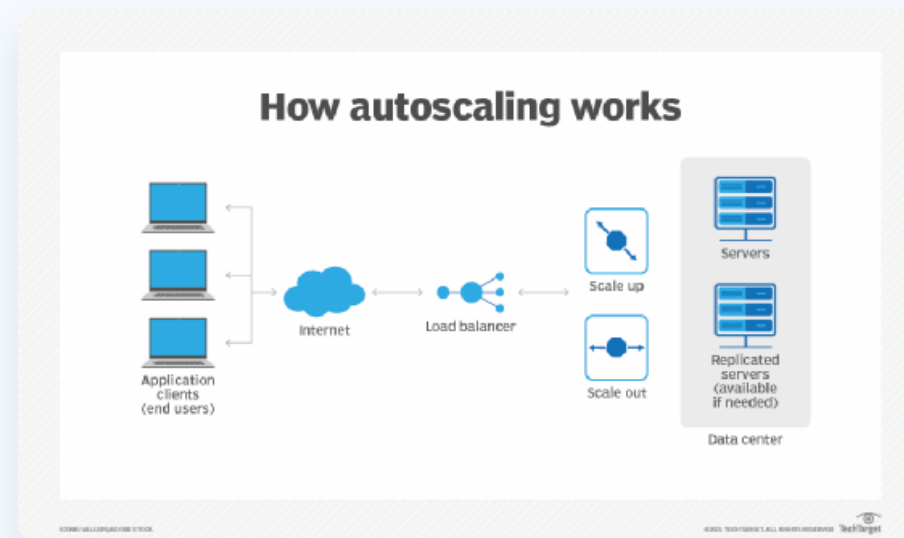
Tần suất truy cập bình thường

**Vài triệu**

Lượng truy cập cùng lúc

**100%**

Tăng trưởng đột biến



# Quan điểm 1: Lỗi của đội ngũ phát triển

## ⚠️ Lập luận chính

### Hệ thống quan trọng quốc gia

Ứng dụng quản lý dữ liệu dân cư là **national critical system** → không được phép downtime trong tình huống cao điểm

### Kiến trúc tốt phải dự đoán traffic đỉnh

- Tax filing season (Mỹ)
- Election day (Ấn Độ)
- Black Friday (Amazon)

### Công nghệ hiện nay có giải pháp

☁️ Autoscaling cloud infra

📦 Queue + async processing

↻️ Caching + CDN

### Thực tế

Nếu app sập chỉ vì vài triệu request cùng lúc → cho thấy **chưa làm kiến trúc hiệu năng nghiêm túc**

## 🌐 Ví dụ tương đồng

### 🇺🇸 Mỹ

Hệ thống trợ cấp COVID-19 giai đoạn đầu cũng sập vì lượng dân đăng nhập quá lớn

**Giải pháp:** Chính phủ phải thuê AWS & Accenture để tái cấu trúc

### 🇸🇬 Singapore

App TraceTogether (COVID) ban đầu cũng nghẽn

**Giải pháp:** Sau optimize mất 4 tháng thì trơn tru

## Kết luận quan điểm 1

Đúng là có lỗi ở khâu **thiết kế hệ thống** và **quản lý hạ tầng**



# Quan điểm 2: Trường hợp đặc biệt, đầu tư dự phòng lãng phí

## ✧ Lập luận chính

### Tần suất sử dụng thấp

Bình thường người dân **3-6 tháng** mới vào 1 lần → hệ thống idle, chỉ vài nghìn concurrent user

### Đầu tư hạ tầng không hiệu quả

Đầu tư hạ tầng luôn luôn chịu tải **100% dân số** là over-provisioning, cực kỳ tốn kém

#### Chi phí hạ tầng

**Hàng trăm tỷ/năm**

Cluster khổng, băng thông khổng, DB phân tán

#### Hiệu suất sử dụng

**~5%**

Trong thời gian bình thường

### Đặc thù hệ thống hành chính

Hệ thống hành chính không phải e-commerce → trải nghiệm chậm **1-2 ngày** vẫn chấp nhận được

## 💡 Giải pháp thường thấy

### 🕒 Accept partial outage

Người dân vào chậm hơn, retry sau vài giờ

### 🔊 Truyền thông phân tán thời gian

Chia theo ngày sinh vào nộp hồ sơ để giảm tải đột biến

### ↗ Trade-off hợp lý

Sập trong vài ngày cao điểm là **trade-off hợp lý** giữa chi phí & lợi ích

## Kết luận quan điểm 2

Scale để chịu peak cực lớn mà hiếm khi xảy ra thì **không cost-effective**

# Quan điểm cá nhân của tác giả

## 🛠 Quan điểm cá nhân

Thiên về hướng thứ 2 nhưng với **trách nhiệm kỹ thuật** vẫn phải có phương án kỹ thuật

## 🔍 Mindset của SA cho VNeID

### ☁ Không duy trì hạ tầng khủng thường xuyên

Sử dụng cloud burst / elastic scaling khi có sự kiện

### 📦 Có queue/virtual waiting room

Tránh cảnh toàn dân bấm vào cùng lúc rồi 500 error

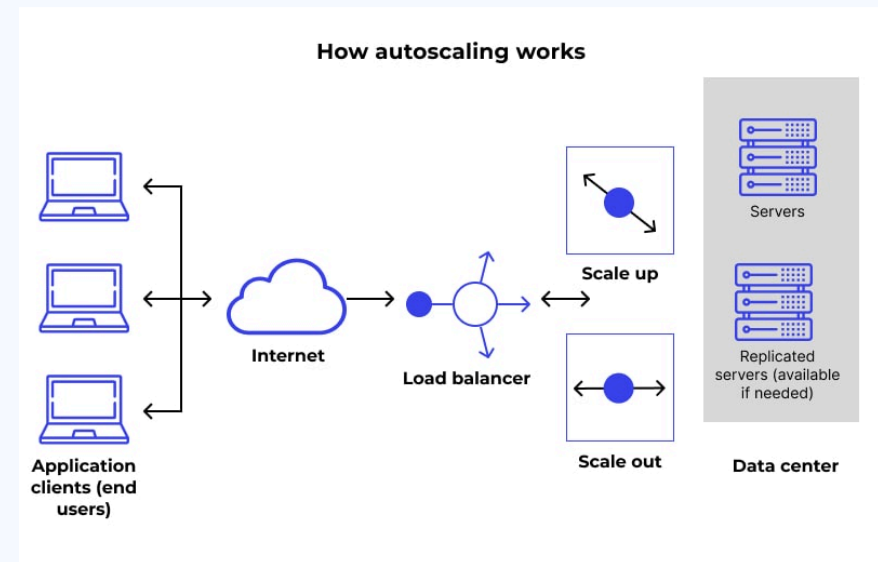
### 💎 Phân tầng truy cập

Thông tin tĩnh cache/CDN, chỉ tương tác nhạy cảm mới vào core DB

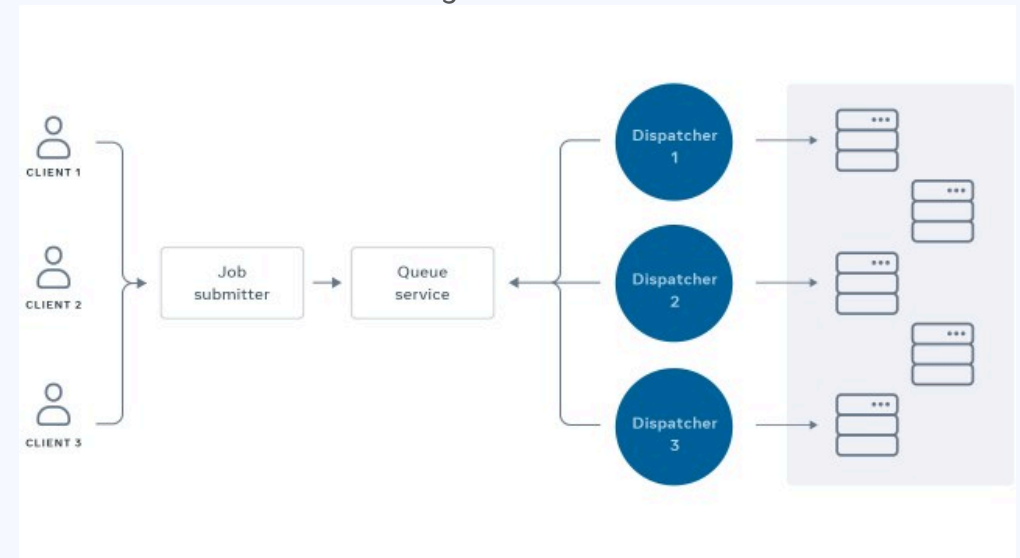
## ⚖ Trade-off hợp lý

Chậm hoặc nghẽn vài ngày là **acceptable trade-off**, vì việc scale để chịu peak cực lớn mà hiếm khi xảy ra thì không cost-effective

## 👤 Giải pháp kỹ thuật đề xuất



Autoscaling cloud infrastructure



Queue và xử lý bất đồng bộ

## 🔍 Phân tích cuối cùng

Nếu app sập hoàn toàn, không có queue, không có plan B → đó là **lỗi kiến trúc và triển khai** (dev/ops)

# Kết luận

## 📄 Tóm tắt các quan điểm

### Quan điểm 1

Lỗi của đội ngũ phát triển do hệ thống không tối ưu

### Quan điểm 2

Trường hợp đặc biệt, đầu tư dự phòng lãng phí

### Quan điểm cá nhân

Thiên về hướng thứ 2 nhưng vẫn có phương án kỹ thuật

## 🌟 Kết luận cuối cùng

- ❗ **Lỗi kiến trúc** nếu hệ thống sập hoàn toàn, không có queue, không có plan B
- ✅ **Trade-off hợp lý** nếu chỉ chậm hoặc nghẽn vài ngày

## 💡 Bài học kinh nghiệm

### ⚙️ Thiết kế hệ thống

- ☁️ **Cloud burst / elastic scaling** khi có sự kiện
- 📦 **Queue/virtual waiting room** để tránh 500 error
- 💎 **Phân tầng truy cập** với cache/CDN cho thông tin tĩnh

### ⚖️ Cân bằng chi phí & hiệu năng

- 📈 **Over-provisioning** là lãng phí với use case hiếm khi xảy ra
- 🕒 **Partial outage** có thể chấp nhận được với hệ thống hành chính
- 🔊 **Truyền thông** phân tán thời gian truy cập