PEAK

Proxy **E**liminating **A**rchitecture using **K**ubernetes

Leveraging Proven Technologies to Create
the Distributed System of the Future

Joar Heimonen
contact@joar.me

December 13, 2024

Abstract

The current paradigm of cloud computing heavily relies on proxies, which introduce single points of failure in systems meant to be distributed. We propose a radical simplification of the current architecture by leveraging the abundance of IPv6 addresses and utilizing modern purpose-built DNS servers to create a distributed system that is both more reliable and scales drastically better than the current cloud computing paradigm. The proposed system utilizes cluster-level DNS servers that dynamically manage service discoverability using Prometheus for monitoring. Service-to-service communication is handled through JSON Web Tokens, creating an intercommunication system that scales 1:1 with the number of users. The system achieves robust fault tolerance through native DNS client failover capabilities, leveraging the universal support for multiple record resolution and automatic retry behavior present in all modern DNS implementations. This eliminates the need for custom failover logic while providing battle-tested reliability mechanisms that operate transparently to applications. This paper describes the implementation of Peak, a proof-of-concept implementation of the proposed architecture, along with its development process and tooling.

Contents

1 Introduction

The evolution of distributed web systems can be traced back to early protocols like FastCGI, which introduced the concept of long-running service processes. This marked a significant departure from CGI's one-process-per-request approach, and established patterns of intermediary communication that would later become ubiquitous in cloud computing. FastCGI's architecture, with its process manager mediating between web servers and application processes, was a precursor to the modern cloud computing paradigm, where services are abstracted into containers and orchestrated by centralized management systems.

The modern cloud computing landscape has evolved this simple concept into a complex ecosystem of proxies, load balancers, message brokers, and service discovery mechanisms. While this architecture has served us well, it introduces significant operational complexity and creates single points of failure in systems designed to be distributed. Current deployments typically rely on multiple layers of proxies for routing, discovery, and load balancing, each representing a potential point of failure.

Our development philosophy focuses on simplification through the creative use of existing battle-tested technologies, rather than introducing more complexity and layers of abstraction into already bloated systems. We believe that simplicity is the key to creating robust and scalable distributed systems. PEAK is developed by looking at established distributed system patterns through the lens of simplicity.

With the widespread adoption of IPv6, we have entered a new era in distributed systems architecture. RIPE NCC's allocation policy provides Local Internet Registries with /29 blocks, each containing over 500,000 /48 networks. This abundance of addresses eliminates the need for network address translation and, by extension, many of the proxy-based patterns that evolved around address scarcity. Modern DNS clients support multiple record resolution and automatic retry behavior, providing a robust failover mechanism that is transparent to applications.

In this paper we present Peak and PeakDNS, a proof-of-concept implementation of the proposed PEAK (Proxy Eliminating Architecture using Kuber-

netes) architecture. Our proof-of-concept demonstrates the feasibility of such a distributed system by implementing a video sharing platform utilizing Kubernetes for container orchestration, and PeakDNS for service discoverability and metric-based record management.

1.1 Scope and Limitations

This paper focuses on the architectural design of PEAK, and the implementation of Peak and PeakDNS. Due to budget constraints, no formal performance testing has been conducted, and the system has not been deployed in a production environment. We feel that the theoretical foundation of the PEAK architecture is solid enough for the paper to stand on its own merits even without performance testing.

2 Technical background

2.1 Réseaux IP Européens (RIPE)

Réseaux IP Européens (RIPE)[10] is a regional internet registry (RIR) that allocates and registers IP addresses in Europe, the Middle East, and parts of Central Asia.

2.2 Domain Name System

The Domain Name System (DNS)[2] is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network.

2.2.1 DNS Record Types

DNS records are used to provide information about a domain or hostname.

- A (Address) - Maps a domain to an IPv4 address.
- AAAA (Address) - Maps a domain to an IPv6 address.
- CNAME (Canonical Name) - Maps a domain to another domain.
- MX (Mail Exchange) - Maps a domain to a mail server.

- NS (Name Server) - Maps a domain to a name server.
- PTR (Pointer) - Maps an IP address to a domain.
- SOA (Start of Authority) - Provides authoritative information about a DNS zone.
- SRV (Service) - Maps a domain to a service.
- TXT (Text) - Provides arbitrary text data.

2.2.2 Source of Authority

The Source of Authority (SOA) record is a type of DNS record that provides authoritative information about a DNS zone. In our case the SOA record is used to point DNS clients to the different cluster-level DNS servers that manage service discoverability.

2.2.3 Time to Live

Each DNS record has a Time to Live (TTL) value that specifies how long the record should be cached. By keeping strict control over the records path of authority and low TTLs, we can ensure that changes to the DNS records propagate quickly.

2.2.4 Multi record resolution

DNS clients support multiple record resolution, which allows multiple records to be returned for a single query. This feature is used to provide fault tolerance and load balancing by returning multiple IP addresses for a single domain. It is up to the client to decide which record to use, and most modern clients implement automatic retry behavior.

2.3 PeakDNS

PeakDNS[3] is a purpose-built DNS server that manages service discoverability and metric-based record management for the Peak distributed system. It is designed to integrate with Prometheus for monitoring and alerting. and Kubernetes for container discoverability. Note that PeakDNS originally was named GoodDNS.

2.4 IPv6

Internet Protocol version 6 (IPv6)[4] is the most recent version of the Internet Protocol (IP)

2.4.1 IPv6 Address Structure

IPv6 addresses are 128 bits long and are represented as eight groups of four hexadecimal digits separated by colons. For example, a typical IPv6 address might look like this:

```
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

To make these addresses more manageable, leading zeros within a group can be omitted, and one consecutive sequence of zero groups can be replaced with a double colon (::), giving us:

```
2001:db8:85a3::8a2e:370:7334
```

This introduces an IP space that contains 2^{128} (approximately 340 undecillion, or 3.4×10^{38}) addresses, eliminating the need for network address translation. To put this in perspective, the entire IPv4 address space (2^{32} addresses) could fit into the IPv6 address space 2^{96} times (approximately 79 octillion times). In other words, we could replicate the entire Internet's IPv4 address space 79,228,162,514,264,337,593,543,950,336 times within IPv6's address space.
For the purpose of network allocation, IPv6 addresses are commonly divided into prefixes using CIDR notation. For example, in a /48 allocation (as mentioned in the RIPE allocation policy):

```
2001:db8:85a3::/48
```

This prefix reserves the first 48 bits (the first three groups) for network identification, leaving the remaining 80 bits for subnetting and host addressing within the organization's network.

2.4.2 IPv6 Address Allocation policy

RIPE NCC's allocation policy provides Local Internet Registries with /32 up to /29 blocks, each containing over 500,000 /48 networks. To qualify for these

allocations the Local Internet registry "must have a plan for making sub-allocations to other organizations and/or End Site assignments within two years."[6]. This makes large IPv6 allocations available to any organization that can demonstrate a need for them.

2.5 JSON Web Tokens

JSON Web Tokens (JWT)[7] are an open, industry-standard RFC 7519 method for representing claims securely between two parties. A claim is a piece of information asserted about a subject, usually consisting of a key-value pair. JWTs are signed using a secret or a public/private key pair, which allows the receiving party to verify the integrity of the token.

2.5.1 JWT Structure

A JWT consists of three parts separated by dots:

- Header - Contains the type of the token and the signing algorithm.
- Payload - Contains the claims.
- Signature - Used to verify the integrity of the token.

The three sections are usually base64 encoded and concatenated with dots to form the JWT. For illustration purposes, a JWT might look like this:

- Header: {`"alg": "HS256", "typ": "JWT"`}
- Body: {`"userId": "1234567890", "name": "John Doe", "admin": true, iat: 1516239022, exp: 1516239022`}
- Signature: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)`

2.6 Docker

Docker[1] is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.

2.7 Kubernetes

Kubernetes (K8s)[9] is an open-source container-orchestration system for automating computer application deployment, scaling, and management.

2.7.1 Minikube

A minikube is a small Kubernetes cluster that runs on a local machine. This is useful for development and testing purposes. As one can test the deployment of services in a Kubernetes cluster without the need for a full-scale deployment.

2.8 Prometheus

Prometheus is an open-source monitoring and alerting toolkit originally built at SoundCloud. It uses pull-based metrics collection approach this allows for quicker detection of downed services.

2.8.1 PromQL

Prometheus Query Language (PromQL) is a query language for Prometheus that allows you to select and aggregate time series data. In our case it is used to dynamically manage the metric-based record management in PeakDNS.

3 Terminology

3.1 Cluster

A cluster is a group of services that are deployed together and share the same DNS server. These clusters are usually located in the same data center or region.

3.2 Micro-cluster

A micro-cluster is a group of services that are deployed under the same cluster and shares the same DNS record. These are usually referred to as deployments in Kubernetes.

3.3 Cluster-level DNS server

A cluster-level DNS server is a DNS server that is responsible for a whole set of micro-clusters.

3.4 Pod

A pod is a unit that usually contains one container, but can contain multiple containers that are deployed together. In our architecture we follow a strict separation of concerns resulting in as few containers per pod as possible.

3.5 Metrics-Based Record Management

Metrics-based record management is a system where DNS records are managed based on the metrics of the services they point to. This allows for dynamic load balancing and fault tolerance based on the performance of the services. Metrics-Based record management allows for arbitrary metrics to be used for managing the DNS records, in our case we use CPU, memory and network metrics.

4 Related Work

While PEAK might seem entirely unique at a glance, it is not. The idea of using DNS for load balancing is not a new one. The following papers describes systems for DNS based load balancing.

"Load balancing of DNS-based distributed Web server systems with page caching"[11] by Zhong Xu, Rong Huang and Laxmi N. Bhuyan published in 2004 Examines "various caching issues, in-cluding the load balancing algorithms of the DNS, the lo- cations of caches, and the page caching policies, for the DNS-based web server system."

"Dynamic Load Balancing Method Based on DNS for Distributed Web Systems"[8] by Jong-Bae Moon and Myhung-Ho Kim published in 2005 proposes a similar method as this paper "The proposed method performs effective load balancing without modification of the DNS. In this method, a server is dynamically added to or removed from the DNS list according to the server's load."

"DNS-Based Load Balancing in Distributed Web-server Systems"[5] by Y. S. Hong, J. H. No, and SY. Y. Kim published in 2006 describes "A cluster Web-server system can be deployed to support high request rates to Web application server (WAS) in Internet-banking. The domain name system

(DNS) servers dispatch the client requests among the Web-servers through the URL-name to IP-address mapping mechanism."

While all the above papers has a focus on optimizing DNS-based load balancing for distributed systems. They do not take into account the address scarcity that has been prevalent since the mid 2000s. And choose to focus on the technical details of DNS load balancing instead of the larger architectural opportunities this approach presents.

5 System Design

5.1 PEAK Architectural Overview

The system consists of a main SOA DNS server that delegates to multiple cluster-level DNS servers. Each cluster-level DNS server is responsible for a whole set of services, and the SOA DNS server delegates to them using NS records. This creates a hierarchical system where the SOA DNS server is the top-level authority, and the cluster-level DNS servers are responsible for service discoverability, this can be seen in *Figure 1*.

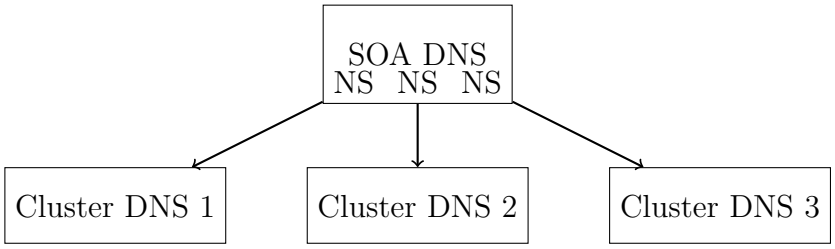

Figure 1: DNS Hierarchy with Multiple Cluster Servers

Each cluster-level DNS server is responsible for a set of micro clusters, and points to them using AAAA records. This creates a direct mapping from DNS to services, as seen in *Figure 2*. In our case each of the cluster-level DNS servers are responsible for micro-clusters of all the services required for the application to function. This reduces the need for inter-cluster communication between data centers.

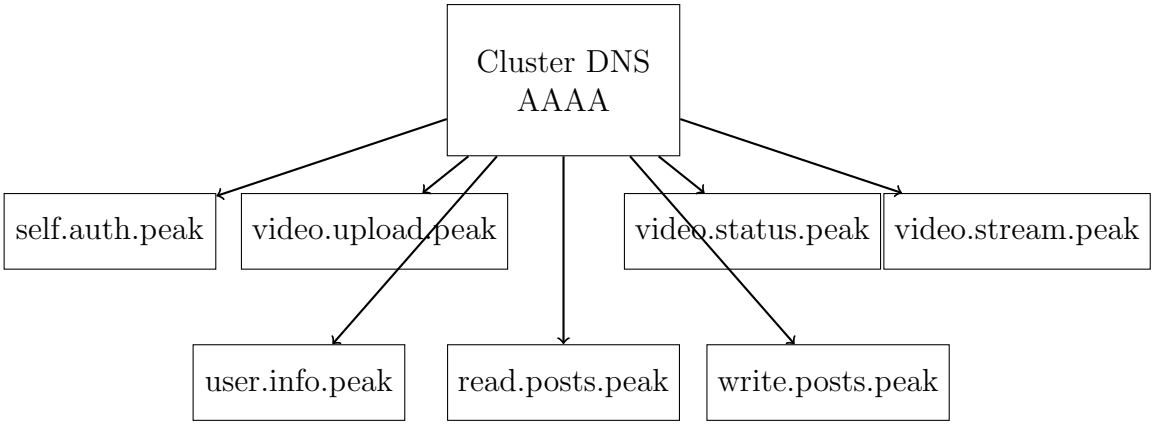

Figure 2: Direct DNS to Micro Cluster Mapping

5.1.1 Authentication

The authentication service is responsible for registering and authenticating users. It generates the JSON Web Tokens that are used for service-to-service communication. All JWTs have an expiration time and the authentication service is the only service that can issue a new expiration time for a JWT. Other services can append claims to the JWT and re-sign them, but they cannot change the expiration time.

5.1.2 Metric-Based record management

The cluster-level DNS server uses Prometheus to monitor the metrics of the micro-clusters it is responsible for. It prompts the Prometheus server with dynamic PromQL queries, the following is an example of a PromQL query that could be used:

```
sum(
    nginx_connections_active{container="nginx-exporter", namespace="%namespace%", pod="%pod-name%"}
) +
sum(
    rate(container_cpu_usage_seconds_total{namespace="%namespace%", pod="%pod-name%"}[5m])
) +
sum(
    container_memory_working_set_bytes{namespace="%namespace%", pod="%pod-name%"} / 1024 / 1024
)
```

Note that the namespace and pod-names are placeholders that are replaced with actual values during runtime. This enables PeakDNS to dynamically prompt Prometheus for the metrics of single pods in the micro-cluster.

PeakDNS implements two different modes for metric-based record management:

- **Round-robin** - The DNS server returns all the IP addresses of the micro-cluster and removes the IP address with a metric that exceeds a threshold calculated from the average metrics of the micro-cluster.
- **Single Best** - The DNS server returns the IP address of the pod with the lowest metric.

An illustration of how PeakDNS queries Prometheus for service metrics can be seen in *Figure 3*. These metrics are then used to manage the DNS records of the services in the micro-cluster.

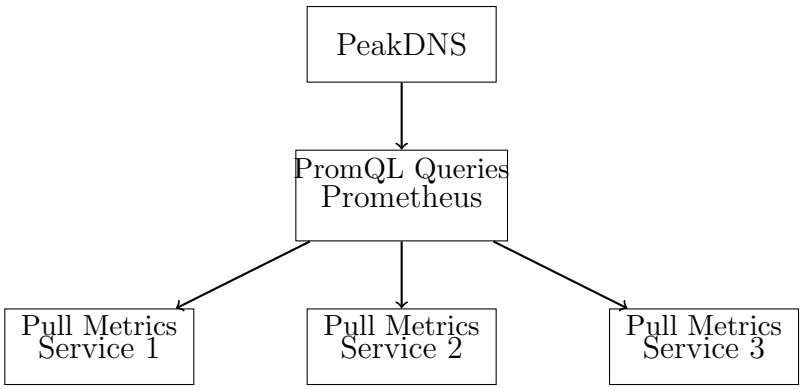

Figure 3: PeakDNS querying Prometheus for service metrics

Figure 4 illustrates the complete flow of a web request with DNS lookup.

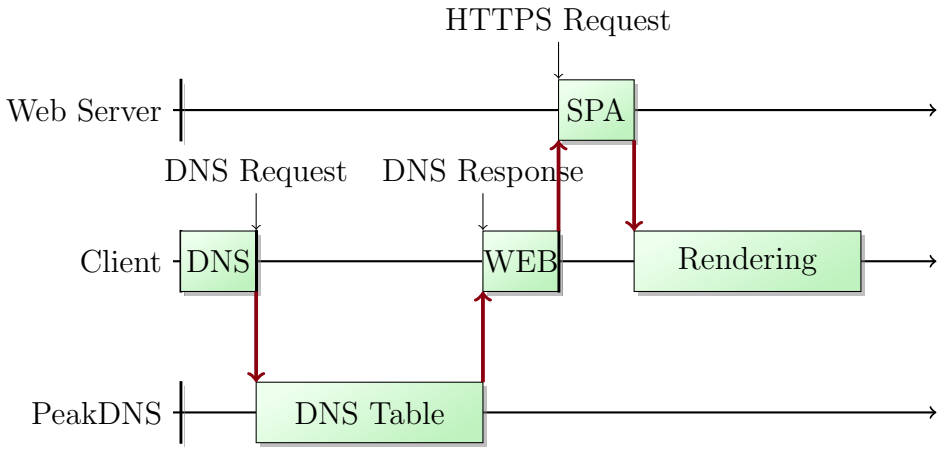

Figure 4: Timeline of a web request with DNS lookup.

5.1.3 Client-Mediated communication

To reduce the complexities of inter-service communication, we have devised a system where the client acts as an intermediary. This allows for service-to-service communication that always scales 1:1 with the number of users. This approach greatly simplifies the inter-service communication and is a radical departure from the current cloud computing paradigm. *Figure 5* illustrates the timeline of two services passing data to each other using JSON Web Tokens and the client as an intermediary.

Client-mediated communication is perfect for systems built using stateless microservices. This means that the services do not store any state related to the user or otherwise. All information needed by a service is fetched at request time from the other services in a conventional architecture. By bundling this information with user requests, the client can act as an intermediary between services without adding to the complexity of the system.

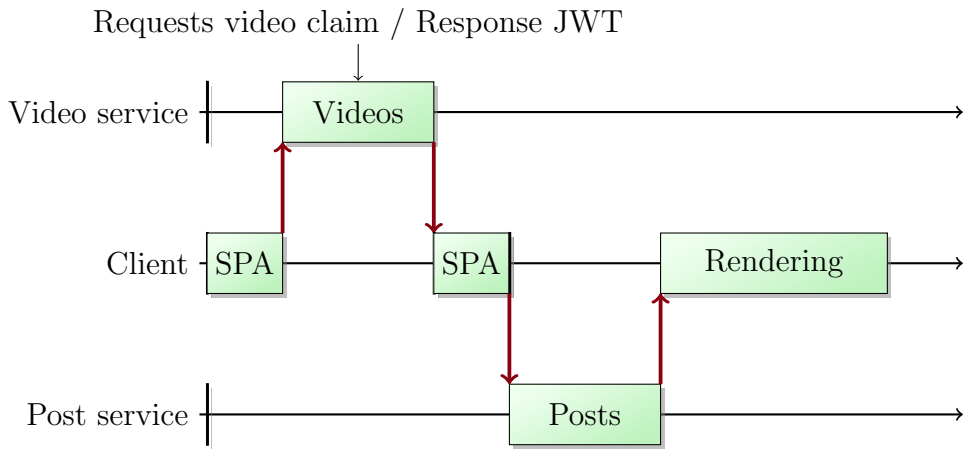

Figure 5: Timeline of two services passing data to each other using JSON Web Tokens and the client as an intermediary.

5.1.4 Cluster-Level DDoS mitigation "The rug-pull"

Note: This system has not been implemented and is purely theoretical.
In traditional DDoS attacks the proxy servers are almost always targeted, this is due to purely practical reasons. The proxy servers are the only exposed servers in the system and are therefore the easiest to target. Due to the PEAK architecture's distributed nature we can take a different approach to DDoS mitigation. An illustration of the following steps can be seen in *Figure 6*.

1. **Report the attack** - Upon attack detection, malevolent IP addresses are reported to the cluster-level DNS server using the Prometheus API.
2. **Address removal** - The affected pods disable their network interfaces.
3. **DNS Resolution blocking** - PeakDNS blocks the IP addresses belonging to the attackers.
4. **Address reacquisition** - Each pod re-enables their network interfaces while also requesting new IP addresses from the DHCP server (or similar).
5. **Metrics-based record management** - PeakDNS updates the DNS records with the new IP addresses.

6. **The attack is mitigated** - The attack targets dead IP addresses and the system is back to normal.

As most of the ideas presented in this paper, this system is a radical departure from the current DDoS mitigation strategies. The primary goal of the "rug-pull" is service survival. During a DDoS attack the attacker's goal is to overwhelm the services. Therefor, it becomes acceptable to abruptly disconnect the affected services in the hope of a quick recovery. This involves dropping all legitimate traffic to the affected services, while prioritizing speed of change over service availability.

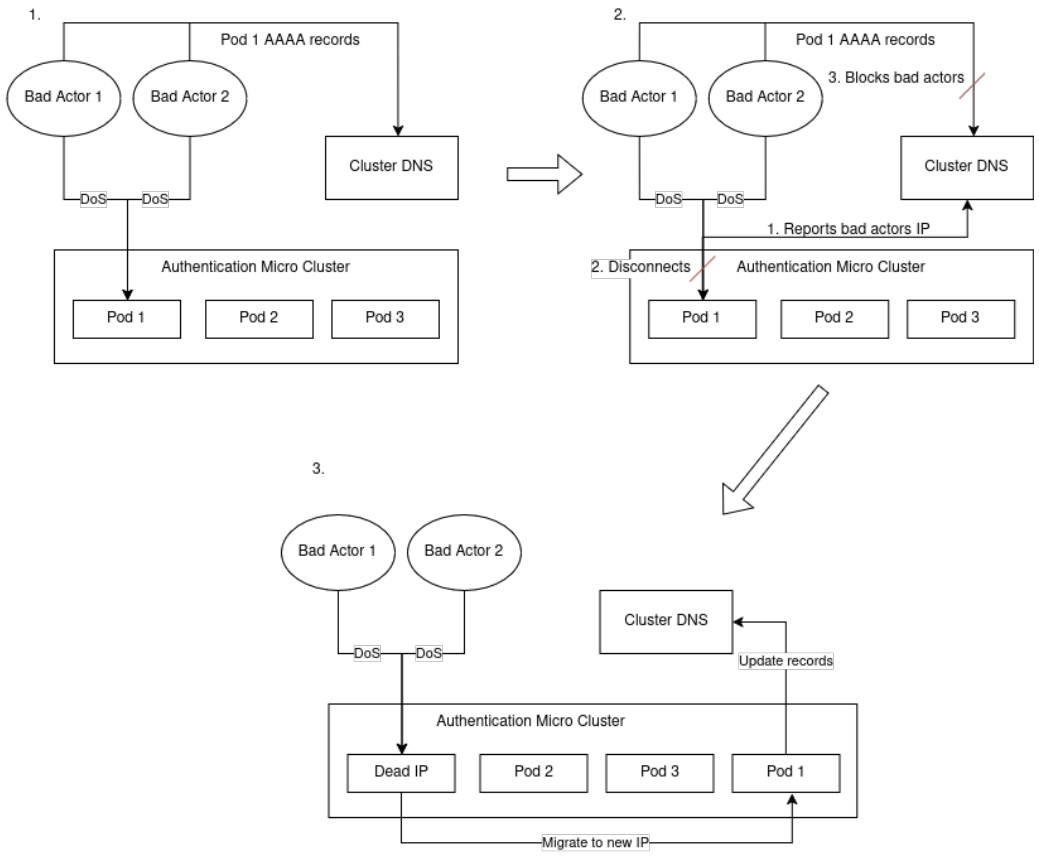

Figure 6: An illustration of the "rug-pull" DDoS mitigation strategy.

5.2 Reference Implementation Overview

Our reference implementation is named Peak and consists of seven different services as seen in *Figure 2*. Peak is a video sharing platform that has been developed in parallel with the PEAK architecture. This section will cover architectural details that are specific to the reference implementation. Our hope is that this section will demonstrate how conventional architectural concepts still apply in the PEAK architecture. The reference implementation only implements PEAK at the cluster level, and does not implement the SOA DNS server.

5.2.1 Service Overview

A full diagram of the micro-clusters in the Peak reference implementation can be seen in *Figure 7*.

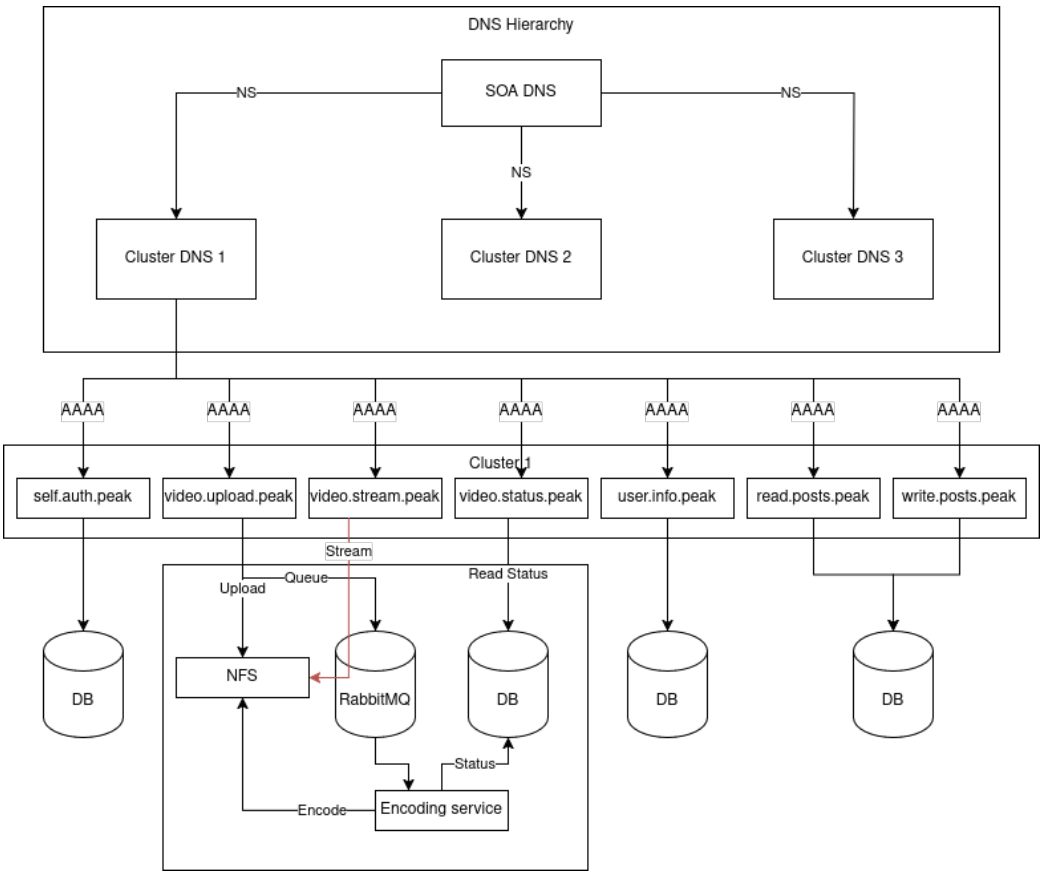

Figure 7: A diagram of the services in the Peak reference implementation. Each of the boxes in the cluster represents a micro-cluster containing multiple instances of the service.

5.2.2 Authentication: self.auth.peak

This is our authentication service, it is written to be as simple as possible. It reads and writes to a table containing only the most basic user information. As seen in *Table 1* the table contains the user's email, username, password, and the time the user was created. The authentication service is the only service that can issue a JWT with a new expiration time, this happens at login, registering and when the JWT is about to expire. In an architecture like this it is paramount that all services checks the JWT signature and expiration time before accepting it as valid. If these checks are missing the

system is vulnerable to a range of attacks.

uid	email	username	password	created_at
1001	user1@example.com	johndoe	$2a$12$LQv3c1yqBg...	2024-12-13 10:30:00
1002	user2@example.com	janedoe	$2a$12$9iR2xk5yqoF...	2024-12-13 10:45:00
1003	user3@example.com	bobsmith	$2a$12$kJH3ns7yjkZ...	2024-12-13 11:15:00

Table 1: Example User Data in Authentication Service. Note that some data has been shortened.

5.2.3 Video Services

The video system consists of four services:

- **video.upload.peak** - This service is responsible for uploading videos
- **video.status.peak** - This service is responsible for checking the status of video encoding
- **video.stream.peak** - This service is responsible for streaming videos
- **encoding service** - This service is responsible for encoding videos

As one can see in *Figure 8* the video services looks quite conventional. The only difference from a conventional architecture is that the client communicates directly with each service, We have removed the proxy layer that usually would be present in a system like this, and we have also reduced inter-service communication to a minimum using only databases and queues for direct inter-service communication. All the services share the same network file system (NFS) for storing videos.

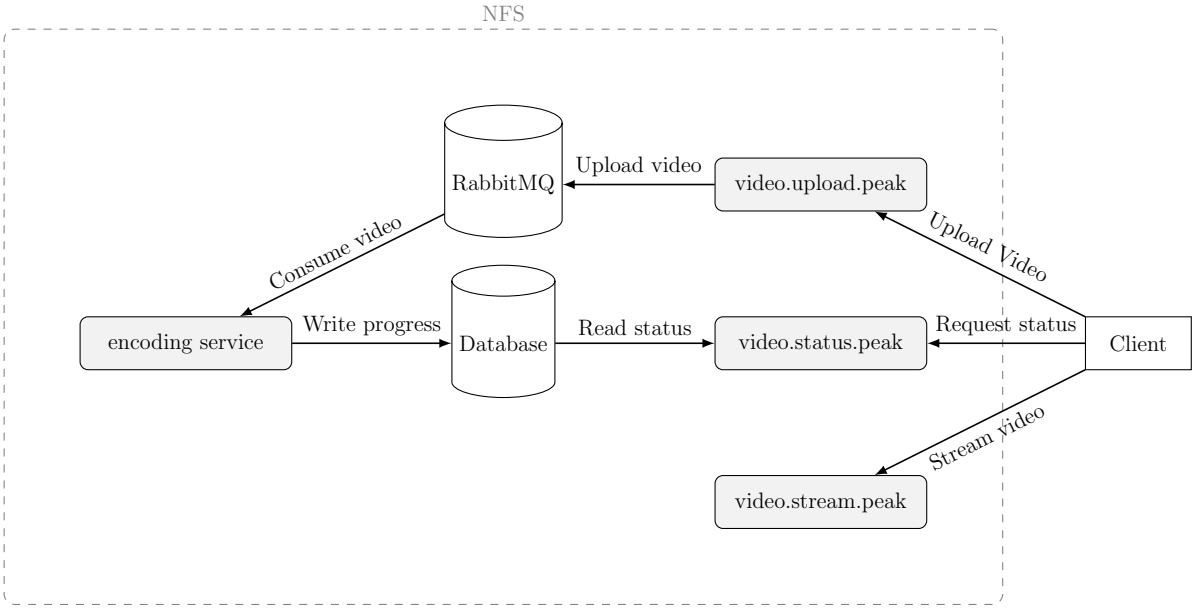

Figure 8: Video Services Overview

6 Post Services

There are two post services in the Peak reference implementation:

- **read.posts.peak** - This service is responsible for reading posts
- **write.posts.peak** - This service is responsible for writing posts

The post services implement client-mediated communication, as seen in *Figure 5*. After a video has been encoded the client requests video.status.peak to append a new claim to the JWT containing a list of all the users uploaded videos. This JWT is then appended to the request to the post services. The post service checks what video is in the post request and compares its identification to the list of uploaded videos in the JWT. This ensures that only the user that uploaded the video can post it.

7 Future work

- **Performance testing** - The system has not been performance tested, and it is unclear how it will perform under load.

- **Production deployment** - The system has not been deployed in a production environment, and it is unclear how it will perform in a real-world scenario.
- **Cluster-Level DDoS mitigation** - The "Rug-Pull" warrant its own paper.

References

[1] *Docker: Accelerated Container Application Development*. https://www.docker.com/. May 2022. (Visited on 12/11/2024).

[2] *Domain Names - Implementation and Specification*. Request for Comments RFC 1035. Internet Engineering Task Force, Nov. 1987. DOI: 10.17487/RFC1035. (Visited on 12/11/2024).

[3] Joar Heimonen. *PeakDNS*. Dec. 2024. (Visited on 12/11/2024).

[4] Bob Hinden and Steve E. Deering. *Internet Protocol, Version 6 (IPv6) Specification*. Request for Comments RFC 2460. Internet Engineering Task Force, Dec. 1998. DOI: 10.17487/RFC2460. (Visited on 12/11/2024).

[5] Y.S. Hong, J.H. No, and S.Y. Kim. "DNS-based Load Balancing in Distributed Web-server Systems". In: *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*. Apr. 2006, 4 pp.-. DOI: 10.1109/SEUS-WCCIA.2006.23. (Visited on 12/13/2024).

[6] *IPv6 Address Allocation and Assignment Policy*. https://www.ripe.net/publications/docs/ripe-738/. (Visited on 12/11/2024).

[7] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. Request for Comments RFC 7519. Internet Engineering Task Force, May 2015. DOI: 10.17487/RFC7519. (Visited on 10/14/2024).

[8] Jong-Bae Moon and Myung-Ho Kim. "Dynamic Load Balancing Method Based on DNS for Distributed Web Systems". In: *E-Commerce and Web Technologies*. Springer, Berlin, Heidelberg, 2005, pp. 238–247. ISBN: 978-3-540-31736-4. DOI: 10.1007/11545163_24. (Visited on 12/13/2024).

[9] *Production-Grade Container Orchestration*. https://kubernetes.io/. (Visited on 12/11/2024).

[10] *Welcome to RIPE and the RIPE NCC*. https://www.ripe.net/. Nov. 2024. (Visited on 12/11/2024).

[11] Zhong Xu, Rong Huang, and L.N. Bhuyan. “Load Balancing of DNS-based Distributed Web Server Systems with Page Caching”. In: *Proceedings. Tenth International Conference on Parallel and Distributed Systems, 2004. ICPADS 2004.* July 2004, pp. 587–594. DOI: 10.1109/ICPADS.2004.1316141. (Visited on 12/13/2024).