# PEAK

**P**roxy **E**liminating **A**rchitecture using **K**ubernetes

Leveraging Proven Technologies to Create
the Distributed System of the Future

Joar Heimonen
contact@joar.me

December 12, 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.

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 Kubernetes) 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 load-based record management.

# 2 Technical background

## 2.1 Réseaux IP Européens (RIPE)

Réseaux IP Européens (RIPE)[8] is a regional internet registry (RIR) that allocates and registers IP addresses in Europe, the Middle East, and parts of Central Asia.

## 2.2 PeakDNS

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

## 2.3 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.

### 2.3.1 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.

### 2.3.2 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.

### 2.3.3 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.

## 2.4 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.

## 2.5 IPv6

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

### 2.5.1 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 $3.4 \times 10^{38}$) 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.

### 2.5.2 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."[5]. This makes large IPv6 allocations available to any organization that can demonstrate a need for them.

## 2.6 JSON Web Tokens

JSON Web Tokens (JWT)[6] 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.

### 2.6.1 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)`

### 2.7 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.

### 2.8 Kubernetes

Kubernetes (K8s)[7] is an open-source container-orchestration system for automating computer application deployment, scaling, and management.

#### 2.8.1 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.

### 2.9 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.

### 2.9.1 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 load-based record management in PeakDNS.

## 2.10 Terminology

### 2.10.1 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.

### 2.10.2 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.

### 2.10.3 Cluster-level DNS server

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

### 2.10.4 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 System Design

## 3.1 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*.

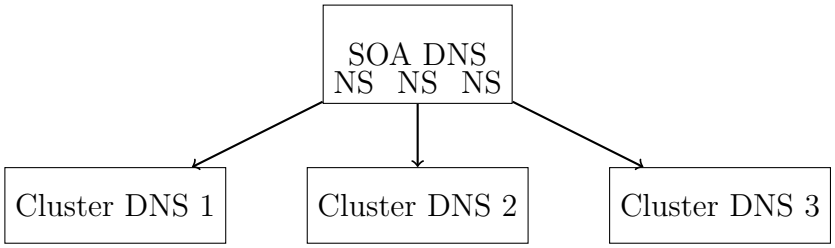


Figure 1: DNS Hierarchy with Multiple Cluster Servers

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.

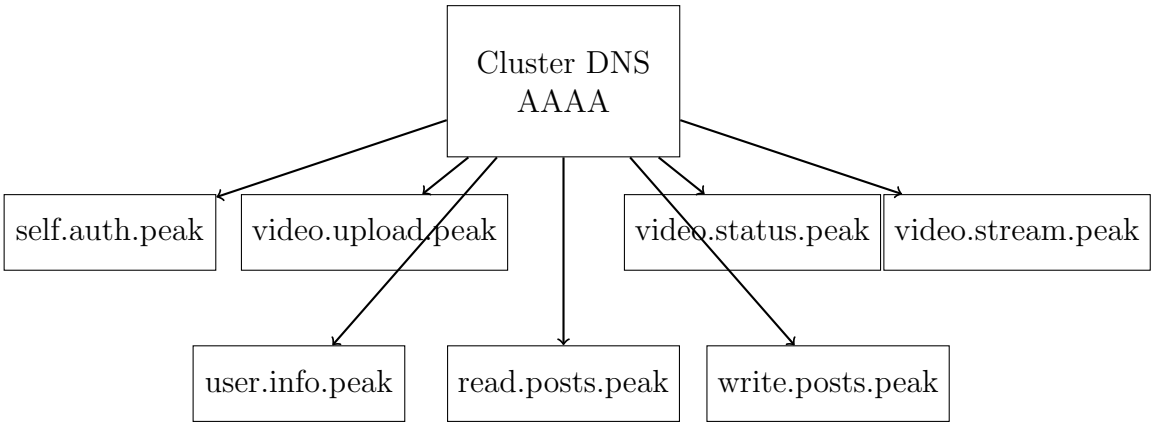


Figure 2: Direct DNS to Micro Cluster Mapping

### 3.1.1 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. An example of this can be seen in *Figure 3* where the client requests a video claim from the video service, so that it can prove ownership of the video when creating a new post.

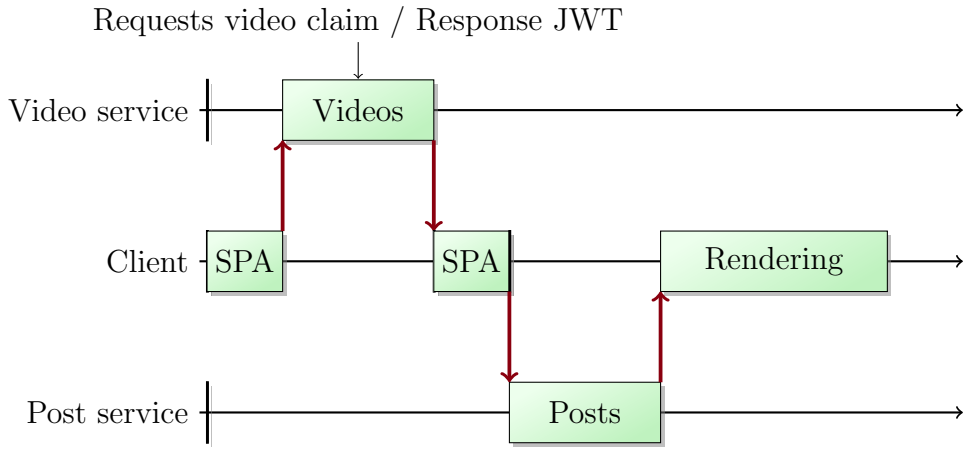


Figure 3: Timeline of two services passing data to each other using JSON Web Tokens and the client as an intermediary.

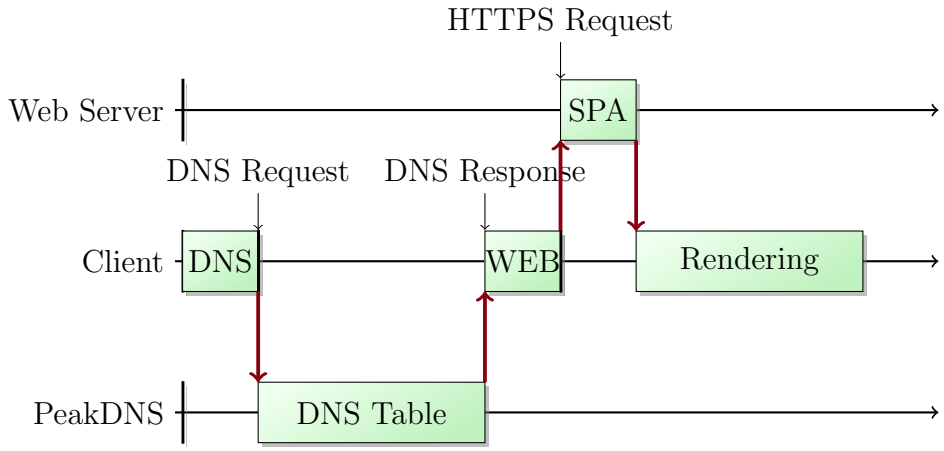


Figure 4: Timeline of a web request with DNS lookup.

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

[6] 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).

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

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