# Building a Pull-Based Content Delivery Network

This project explores the implementation of a pull-based content delivery network leveraging Go programming and Google Cloud Platform services, demonstrating practical distributed systems.

**Arvin Samuel A - 23BCE1283**

# Learning Objectives for GCP Project

- **Hands-on Familiarity with GCP**

  Engage with the Google Cloud Platform to build practical skills.

- **Implementing Pull-based CDN**

  Create a CDN architecture similar to industry leaders like Cloudflare and Akamai.

- **Understanding Containerization**

  Explore containerization concepts integral to cloud services and deployment.

- **Load Balancing Techniques**

  Learn about load balancing strategies to optimize resource distribution.

- **Distributed Caching Concepts**

  Delve into distributed caching for improved data access and performance.

- **Event-driven Communication**

  Understand how event-driven architectures enhance system responsiveness.

# Challenges and Solutions in CDN

Exploring issues and innovative solutions in content delivery

01 **Global User Access Needs**

Users worldwide require fast and reliable access to content, necessitating improved delivery methods.

02 **Central Server Bottlenecks**

Without a CDN, content is fetched from a central server, leading to high latency and potential bottlenecks.

03 **Traditional CDN Limitations**

Traditional CDN systems are often expensive and lack transparency for developers and students.

04 **Custom Pull-Based CDN Design**

Our solution involves designing a custom, pull-based CDN using cloud-native GCP services.

05 **Enhanced Edge Caching**

The solution also enables edge caching and on-demand content delivery from a single origin server.

# Comprehensive System Architecture Overview

Understanding the components of GCP architecture

- **Main Server Location**

  Utilizes a GKE Cluster situated in Asia-South-1 (Mumbai) for centralized processing.

- **Distributed Edge Nodes**

  Incorporates 7 global Cloud Run edge nodes to enhance content delivery speed.

- **In-Memory Cache Layer**

  Employs Redis for fast read/write operations, improving data retrieval efficiency.

- **Event-Based Messaging**

  Uses Redis Pub/Sub for efficient update propagation across the system.

- **Automated Content Invalidation**

  Implements a TTL mechanism for automatic content expiration and refresh.

- **Global Load Balancing**

  Features a global load balancer providing a single, unified global IP address.

# CDN Pull Workflow Overview

Understanding the CDN Pull Mechanism in GCP

01

### Client Request Routing

Client requests are routed through a global load balancer to the nearest Cloud Run node for optimal performance.

02

### Node Cache Check

Nodes check their cache: serve content instantly on hits, or fetch from main server on misses.

03

### Content Caching Mechanism

Content is stored in Redis with key-hash and hash-value pairs to avoid duplicate storage of the same content.

04

### Cache Update Notification Process

Nodes publish cache updates via Pub/Sub, allowing other nodes to subscribe and refresh their cache.

05

### Content Expiration Management

A Time-to-Live (TTL) ensures that stale content is automatically removed from the cache.

# Comprehensive Overview of Technology Stack

Exploring the Essential Technologies Used

| Component | Technology Used |
| --- | --- |
| Language | Go (for concurrency & speed) |
| Containerization | Docker |
| Main Server | GKE (Google Kubernetes Engine) |
| Edge Nodes | Cloud Run (Serverless, scalable) |
| Caching | Redis (used as key-value store) |
| Communication | Redis Pub/Sub |
| Load Balancing | Global HTTP(S) Load Balancer (GCP) |
| Registry | Artifact Registry |
| Testing | Insomnia |

# Comprehensive Testing Methodology

Exploring Tools and Metrics for API Performance

- **Use of Insomnia for API Testing**

  Insomnia is utilized for simulating APIs and conducting load tests effectively.

- **Simulated Real-World Scenarios**

  Scenarios include cache hits/misses and TTL expiration to mimic actual usage.

- **Key Metrics for Performance**

  Metrics like latency, cache hit ratio, and response time are captured to assess performance.

- **Latency Measurement**

  Latency is measured before and after cache warm-up to evaluate improvements.

- **Cache Hit Ratio**

  The cache hit ratio is analyzed to determine the efficiency of caching mechanisms.

- **Response Time Distribution**

  Response time distribution helps in understanding the variability in API responses.

- **Traffic Routing Efficiency**

  Traffic routing efficiency is assessed based on GCP location to ensure optimal performance.

# Performance Results of CDN Implementation

Overview of cache performance improvements

| Scenario | Observation |
| --- | --- |
| First-time request | Cache miss → Pull from GKE → Cache store |
| Repeated requests | Cache hit → Near-instant response |
| Pub/Sub test | Updates propagated across subscribed nodes |
| TTL test | Expired content removed, re-fetched correctly |

# Key Challenges in Implementation

- **Networking Configuration**

  Establishing secure private communication between GKE and Cloud Run.

- **Inter-Service Communication**

  Complex integration of Pub/Sub communication between GKE Cluster and Nodes required IAM and firewall adjustments.

- **Redis Caching Logic**

  Implemented a 2-layered key→hash and hash→value model for efficiency.

# Essential Insights on GCP Infrastructure

Understanding GCP Services and Best Practices

- **GCP Infrastructure Overview**
  Gained a solid understanding of GCP services like GKE and Cloud Run.
- **Container Orchestration Mastery**
  Learned about effective container orchestration and its applications.
- **Event-Driven Architecture**
  Explored the principles of event-driven architecture in cloud systems.
- **Distributed Caching Techniques**
  Implemented distributed caching strategies including TTL invalidation.
- **Multi-Region Complexity**
  Realized the intricacies of managing multi-region distributed systems.
- **Cloud Security Best Practices**
  Understood best practices for cloud security and service isolation.

# Expanding Future Scope of CDN Features

Innovative features for next-gen content delivery

- **Support for Advanced Data Types**

  Plans to extend support to videos, documents, and encrypted binaries.

- **Intelligent Edge Selection**

  Utilizing ML for smart edge selection based on popularity and predictions.

- **Dynamic Node Management**

  Implementing auto-scaling and dynamic node registration for efficiency.

- **Monitoring and Prefetching Logic**

  Adding features for prefetching logic and dashboard monitoring.

# Dynamic Data Handling in GCP

Exploring efficient data caching and retrieval in GCP

### Client Data Requests

Client requests uncached data, demonstrating the system's ability to access real-time information.

### Data Caching Mechanism

Initial data request is pulled from Google Kubernetes Engine (GKE) and then cached for efficiency.

### Cache Hit Efficiency

Subsequent requests for the same data result in cache hits, speeding up response times.

### TTL Expiration Process

Cached data has a Time to Live (TTL) expiration, triggering an automatic refetch when needed.

### Update Propagation Method

Data updates are communicated effectively using Pub/Sub for real-time adjustments.

### Latency Management

Cloud Run handles latency across various regions, ensuring consistent performance for users worldwide.

Enter your source text

# Conclusion: Pull-Based CDN Implementation

- **Successful Pull-Based CDN Deployment**

  The project achieved a fully functional pull-based CDN using GCP-native solutions.

- **Real-World Simulation**

  GCP-native tools were utilized to create a realistic distributed network environment.

- **Learnings Reinforced**

  This project reinforced key concepts from the Cloud Computing course, enhancing understanding.

- **Scalability and Production Readiness**

  The implementation is ready to scale or move to production with minimal adjustments needed.

# Thank You!