Cloud CDN

1st Rui Santos
DCC
FCUP
Porto, Portugal
up202109728@up.pt

2nd Ricardo Sá
DCC
FCUP
Porto, Portugal
up202408644@up.pt

3rd Xavier Santos
DCC
FCUP
Porto, Portugal
up202108894@up.pt

***Abstract*—This report details the design and implementation of a decentralized CDN on Azure. The CDN employs Cloudflare Workers for intelligent routing and autoscaling VM Scale Sets (VMSS) for efficient content delivery. The project emphasizes low latency, cost optimization, and secure content management.**

Note that the analysis within this report, especially concerning cost and performance, is based on an assumed moderate user activity scenario and is an estimate; actual results will depend on the workload imposed on the CDN.

I. Introduction

This report details the design and implementation of a decentralized Content Delivery Network (CDN) on Azure. The CDN leverages Cloudflare Workers for intelligent routing and utilizes autoscaling Virtual Machine Scale Sets (VMSS) for efficient content delivery. The project focuses on achieving low latency, optimizing costs, and ensuring secure content management. It's important to note that the cost and performance analysis presented herein are based on an assumed scenario of moderate user activity and are therefore estimates; actual results may vary depending on the workload imposed on the CDN.

II. Architecture

Figure 1 illustrates the proposed architecture for our cloud-based Content Delivery Network (CDN), strategically divided into two key components: the Frontend and the Backend. The **Frontend** uses Cloudflare Workers for geographic load balancing and reverse proxy functionalities, and directs client requests to the most suitable available load balancer based on their location. The **Backend** infrastructure, deployed across different geographical regions (West Europe and East Asia in this depiction), comprises regional load balancers with public IP adresses, VMSS containing multiple virtual machines, and an origin server located in the East US region.

Our CDN implementation utilizes a caching layer within the VMSS to enhance performance. Specifically, Nginx is deployed on the VMs to serve as caching nodes, storing and delivering content to nearby clients. The origin server, on the other hand, is responsible for persistently storing all content and providing it to the caching nodes as needed.

Content uploaded to the origin server is secured with JWT token-based authentication. Users authenticate them self to obtain a token, which is then used to authorize uploads, ensuring only authenticated users can add content.

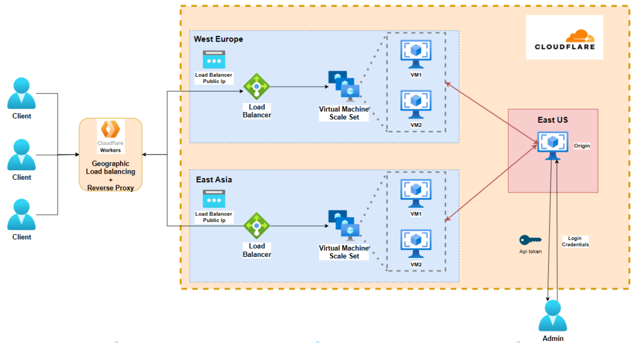

Fig. 1: Full architecture

A. *Elasticity strategy*

Our CDN architecture incorporates an elasticity strategy to dynamically adjust the number of caching nodes (VMs) in response to varying traffic demands, ensuring both performance and cost-efficiency. We achieve this by using Azure's autoscaling capabilities.

The autoscaling configuration for the VMSS is defined as follows:

- **Baseline Capacity**: The VMSS is initialized with a default capacity of 2 VMs, which also serves as the minimum number of active instances. This ensures that the CDN can handle a base level of traffic.
- **Maximum Capacity**: The autoscaling is capped at a maximum of 4 VMs. This limit prevents excessive resource consumption and uncontrolled scaling, helping to manage costs.
- **Scaling Metric**: The autoscaling rules are triggered based on the average CPU utilization of the VMs within the VMSS. This metric provides a reliable indicator of the current workload.
- **Scale-Out Rule**: When the average CPU utilization exceeds 75% over a 5-minute evaluation period, the autoscaling mechanism adds one VM to the scale set. The `time_grain` is set to 1 minute, meaning the CPU utilization is sampled every minute, and the `time_window` is 5 minutes, calculating the average over that window. A 5-minute cooldown period prevents overly aggressive scaling.
- **Scale-In Rule**: Conversely, when the average CPU utilization drops below 25% over a 5-minute evaluation period, the autoscaling removes one VM. This ensures

that resources are scaled down during periods of low traffic, optimizing cost.

- **Scaling Action**: Both scaling actions (scale-out and scale-in) adjust the number of VMs by one at a time (`value` = ”1”). This gradual adjustment helps to avoid sudden spikes or drops in capacity.

This elasticity strategy allows our CDN to automatically scale its caching capacity in response to fluctuations in user demand. By adding VMs during peak traffic, we maintain performance and responsiveness. By removing VMs during low traffic, we minimize operational costs.

III. Security Framework

The security of our CDN is addressed through two primary components: Cloudflare and the upload API. Each of these will be discussed in detail in the following subsections.

A. Cloudflare

Initially, we acquired a free ”.pt” domain (for a one-year period) and subsequently transferred its management to Cloudflare by updating its name servers to Cloudflare’s designated ones. Following this initial setup, we implemented a crucial security measure by configuring firewall rules within both the Virtual Machine Scale Sets (VMSS) and the load balancers. These rules were specifically designed to permit inbound traffic exclusively originating from Cloudflare’s known IP address ranges (refer to [1] for a comprehensive list) while explicitly blocking all traffic from any other source.

This configuration leverages the free security features provided by Cloudflare, which include: unmetered application layer DDoS protection, IP-based rate limiting, Protect against high severity and widespread vulnerabilities with WAF, Detect and challenge common bots only, Cloudflare IPs , etc... .

By restricting access at the VMSS and load balancer levels to only Cloudflare IPs, we enforce that all user traffic to the CDN must pass through the Cloudflare infrastructure. This effectively prevents direct access to our backend components, thereby significantly reinforcing the overall security posture.

Furthermore, we configured A records within Cloudflare’s DNS management for each component possessing a public IP address, namely the origin server and both regional load balancers (in West Europe and East Asia). This ensures that our domain name correctly resolves to the appropriate Cloudflare entry points for each service.

B. Upload API

As previously mentioned, we implemented a simple API authentication mechanism using JWT (JSON Web Token) based authentication. This mechanism governs the upload of new content to our CDN, ensuring that only users with the appropriate privileges can introduce new assets into the system.

IV. Scenario

The Origin Server that stores all content. A scalable group of Caching Nodes is managed by a VMSS to distribute content closer to users. Users access the CDN via a domain name, which is managed by Cloudflare and resolves to the public IP address of an Azure Load Balancer (see Fig 1).

For our cost and performance analysis, we assumed a scenario with a substantial user base where requested content has a high likelihood of being present in a caching node’s cache.

A. Origin VM & Caching nodes specs

For the origin server, we choose the **Standard_B2ms** size, which offers a cost-effective entry point. It provides 2 vCPUs and 8 GiB of RAM. Furthermore, we will utilize a **Standard SSD LRS** managed disk for the operating system and data. While the origin server’s primary role involves storage and infrequent direct access (as caching will be handled by other VMs), using an SSD provides significantly faster read and write speeds compared to traditional HDDs. Given the trade-offs of cost and performance, we decided that the burstable performance of the B-series coupled with the speed of an SSD is sufficient for the origin scenario.

For the caching nodes, created in the VMSS, for both the **East US** and **West Europe** regions, we used the same size as the origin VM with also a smaller SSD. Based on our research, we defined this as the best option in terms of performance/cost for the defined caching scenario.

V. Cost

It is important to note that the cost of this CDN solution is not definitive. Due to the dynamic nature of CDN usage, costs fluctuate significantly based on factors such as traffic volume, data transfer, and the frequency of cache hits. Therefore, the cost analysis presented in this report is based on an assumed scenario mentioned previously .

The cost analysis in this report is based on information from [4] and the price calculator [5] provided by Azure .

The overall cost can be seen in table I

A. Caching nodes cost

The size of the VMs within the VMSS is **Standard_B2ms**. This incurs an estimated monthly cost of approximately 63.14$ per VM in West Europe and 70.34 $ per VM in East Asia (based on current Pay-As-You-Go Linux pricing and the cost of a 32GB Standard SSD LRS disk). It’s important to consider that, with autoscaling, the total cost for the VMSS can increase up to fourfold, as the system scales to a maximum of 4 instances.

B. Origin server

A significant cost component is the origin server, for which we are using a **Standard_B2ms** VM with a 128 GB SSD. The estimated monthly expense for this VM in the East US is approximately 99.14$.

C. Cloudfare & DNS domain

Cloudflare's free plan was used, so there were no Cloudflare-related costs. While the DNS domain was initially free for one year, it's crucial to consider the potential future cost. The annual renewal fee for the domain with our provider is 33.49$.

Resource	**Cost (monthly)**
Cache nodes (West EU)	$n \times 63.14$$
Cache nodes (East Asia)	$m \times 70.34$$
Origin	99.14$
DNS (after 1 year free trial)	33.49$
Cloudflare (free plan)	0$
Basic load balancer (for West EU and East Asia)	0$

TABLE I: Overall Cost of each component (per month)

VI. Performance

To evaluate the performance of our Cloud CDN, we conducted tests focused on measuring the time it takes for a client to receive a specific file. As mentioned before, we used the **Standard B2ms** size for all vms present in the CDN (caching and origin).

The testing procedure involves the following steps:

1) A client initiates a request for a designated file.
2) This request is routed through Cloudflare to the appropriate regional load balancer.
3) The load balancer distributes the request to a VM within the corresponding VMSS.
4) If the file is present in the VM's cache, it is served directly to the client.
5) If the file is not cached, the VM retrieves it from the origin server, serves it to the client, and caches it for subsequent requests.
6) The time taken for the client to receive the complete file is recorded.

By measuring this end-to-end delivery time, we can assess the CDN's latency and overall performance

A. Results

Utilizing my personal computer and home network (2.4GHz WiFi with average download speeds of 40-50 MBps), I conducted performance testing for the EU client.The results can be seen in table II.

	MISS	**HIT**
200 Mb	$\approx$ 25.87s	$\approx$ 22.29s
20 Mb	$\approx$ 6.35s	$\approx$ 3.33s
4 Kb	$\approx$ 0.99s	$\approx$ 0.43s

TABLE II: Performance Test Results (WEST EU)

Unlike the EU client testing on my local network, the Asia client evaluation utilized a simple Azure VM situated in the same region as the VMSS and Load Balancer. This resulted in dramatically better performance, primarily because the caching was geographically closer to the client and the Azure VM offered a much more reliable and faster network connection.The results can be seen in table III.

	MISS	**HIT**
200 Mb	$\approx$ 16.40s	$\approx$ 2.48s
20 Mb	$\approx$ 3.83s	$\approx$ 0.46s
4 Kb	$\approx$ 0.99s	$\approx$ 0.34s

TABLE III: Performance Test Results (EAST ASIA)

VII. Deployment

We utilized Terraform for the deployment of our cloud CDN. Terraform is an Infrastructure as Code (IaC) tool that allows us to manage and provision our cloud infrastructure through declarative configuration files. This approach enabled us to automate the creation and configuration of resources, such as the VM setup scripts for the VMs in the VMSS and the origin server VM, ensuring consistency and reproducibility. The complete Terraform configuration files are available in [2].

VIII. Conclusion

In conclusion, the implemented CDN architecture successfully demonstrates a decentralized approach on Azure, integrating Cloudflare for intelligent routing and security, and employing autoscaling VMSS for dynamic content delivery. The performance tests, while dependent on network conditions and geographical proximity, highlight the benefits of caching and the impact of proximity to the caching nodes. The cost analysis provides a transparent overview of the expenses associated with the various components, emphasizing the cost-effectiveness of Cloudflare's free plan and the dynamic nature of VMSS costs based on autoscaling. Ultimately, the selection of appropriate VM sizes and specifications for both the origin server and the caching nodes within the VMSS should be preceded by a thorough cost/performance analysis tailored to anticipated usage patterns and budgetary considerations.

References

[1] https://www.cloudflare.com/ips/
[2] https://github.com/RS181/Cloud/tree/main
[3] https://developer.hashicorp.com/terraform
[4] https://instances.vantage.sh/azure/
[5] https://azure.microsoft.com/en-us/pricing/calculator/