🌐 Cloud Project Documentation

Name : Mohammed Mahmoud Elnaggar
Project Title: Website Deployment on Huawei Cloud
Cloud Provider: Huawei Cloud

HUAWEI CLOUD

1. Introduction

This project demonstrates the design and deployment of a scalable web application architecture on **Huawei Cloud**. The goal was to build a robust, secure, and highly available infrastructure that supports website hosting while leveraging Huawei Cloud's core services.

2. Architecture Overview

The architecture was designed to ensure **security, scalability, availability, and controlled internet access** while deploying the web application.

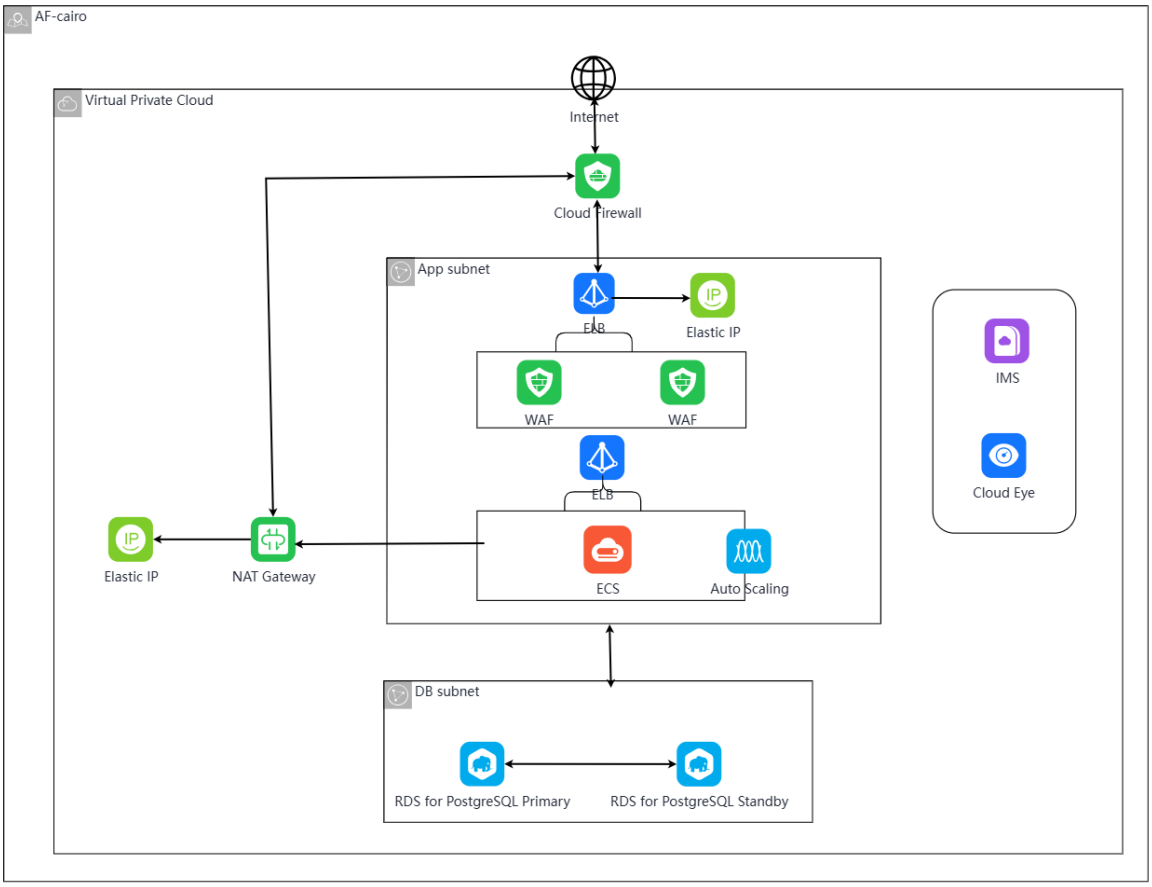
- **Elastic Cloud Server (ECS):** Hosted the Dockerized **frontend** and **backend** application containers.
- **Elastic Load Balance (ELB):** Used at two levels in the architecture to distribute traffic and ensure high availability.
- **Web Application Firewall (WAF):** Secured the application by filtering malicious traffic before requests reached ECS.
- **Firewall:** Controlled public access to the **NAT Gateway** and the **ELB** public IPs.
- **Auto Scaling (AS):** Managed ECS scaling dynamically based on demand and CPU utilization.
- **NAT Gateway:** Provided secure outbound internet access for ECS instances, mapped to a fixed public IP.
- **Subnet with PostgreSQL Database:** A private subnet hosted a **primary/standby PostgreSQL database cluster**, ensuring high availability and fault tolerance. ECS instances in the Auto Scaling Group communicated with this subnet for application data.

📌 **Traffic Flow:**

1. Incoming requests from the internet first reach the **firewall**, which filters access to the **ELB** and **NAT Gateway**.
2. The **first ELB** distributes traffic across **two WAFs** for advanced security filtering.
3. The **WAFs** pass only valid traffic to the **second ELB**, which balances load across ECS instances in the **Auto Scaling Group**.
4. ECS instances interact with the **PostgreSQL primary/standby cluster** in the private subnet for database operations.
5. When ECS instances require outbound internet access (e.g., fetching updates), traffic exits through the **NAT Gateway**, mapped to a dedicated public IP.

📌 *Architecture Diagram should illustrate:*

- Internet → Firewall → ELB → WAFs → ELB → ECS AS Group → PostgreSQL Subnet
- Outbound ECS traffic → NAT Gateway → Public IP



3. Cloud Infrastructure Setup

3.1 Compute (ECS + Auto Scaling)

- Provisioned **Elastic Cloud Servers (ECS)** running Ubuntu 22.04.
- Installed **Docker** and deployed both **frontend** and **backend** application containers using `docker-compose`.
- Configured **Auto Scaling Group (ASG)** with:
    - Minimum instances: **1**
    - Maximum instances: **2**
    - Scaling up trigger: CPU > 70%
    - Scaling down trigger : CPU <= 20%

3.2 Networking & Security

- **Virtual Private Cloud (VPC):** Created an isolated network environment.
- **Subnets:**
    - Public subnet for ECS and ELBs.
    - Private subnet hosting the **primary/standby PostgreSQL database** .
- **Firewall:** Configured to allow only essential inbound traffic (HTTP/HTTPS to ELB, SSH restricted by IP).
- **Security Groups:** Applied rules for granular access control:
    - ECS to PostgreSQL subnet (port 5432).
    - ECS to NAT Gateway for outbound traffic.

3.3 Load Balancing & Web Security

- **First ELB (Public-facing):** Exposed to the internet behind the firewall, distributing traffic across **two WAFs**.
- **Web Application Firewalls (WAFs):** Inspected traffic for threats (SQL injection, XSS, DDoS).
- **Second ELB (Internal-facing):** Received clean traffic from WAFs and distributed it across ECS instances in the Auto Scaling Group.

3.4 Database (PostgreSQL)

- Deployed a **primary/standby PostgreSQL cluster** inside a private subnet for high availability.
- Configured replication and automatic failover.
- ECS instances in the Auto Scaling Group accessed the database securely through private networking (no public exposure).
-

3.5 Outbound Internet Access (NAT Gateway)

- **NAT Gateway:** Configured for outbound traffic from ECS instances.
- Mapped to a **dedicated public IP**.
- Used for pulling container images, software updates, and external API calls while keeping ECS private IPs hidden.

4. Deployment Process

4.1 Containerization

- Built **Docker images** for the **frontend** (React + Nginx) and **backend** (Java Spring Boot).
- Defined multi-container setup in `docker-compose.yml` for easy deployment.
- Uploaded images to a private image repository (Huawei Cloud SWR or Docker Hub).

4.2 ECS Deployment

1. Connected to ECS instances over SSH.
2. Installed Docker & Docker Compose:
3. `sudo apt update && sudo apt install -y docker.io docker-compose`
4. Pulled container images from repository:
5. `docker pull <repo>/frontend:latest`
6. `docker pull <repo>/backend:latest`
7. Launched services:
8. `docker-compose up -d`
9. Configured **systemd service** so that containers auto-start after ECS reboot.

4.3 Load Balancing & WAF Integration

- Configured the **public-facing ELB** to accept inbound HTTP/HTTPS requests.
- Linked ELB to **two Web Application Firewalls (WAFs)** for security inspection.
- Configured WAF policies to block SQL injection, XSS, and DDoS attacks.
- Deployed a **second ELB** behind WAFs to distribute valid traffic across ECS Auto Scaling Group instances.

4.4 Database Integration

- Deployed **primary/standby PostgreSQL cluster** in a private subnet.
- Configured ECS application containers to connect securely via private VPC route:
- `DB_HOST=postgresql-primary-subnet.local`
- `DB_USER=app_user`
- `DB_PASSWORD=********`
- `DB_NAME=myappdb`
- Verified replication and failover between primary and standby PostgreSQL instances.

4.6 Outbound Internet Configuration

- Configured **NAT Gateway** for outbound ECS traffic.
- Ensured ECS instances could securely pull updates, packages, and images while masking private IPs behind the NAT public IP.

5. Challenges & Solutions

Challenge 1: ECS Containers Not Starting After Reboot

- **Problem:** After rebooting ECS instances, the frontend and backend containers did not start automatically.
- **Solution:** Configured a **systemd service** to ensure `docker-compose up -d` runs at boot, keeping services always online.

Challenge 2: Secure Internet Access for ECS Instances

- **Problem:** ECS instances required internet access (for updates, image pulls), but exposing them directly with public IPs introduced security risks.
- **Solution:** Implemented a **NAT Gateway** with a dedicated public IP, ensuring outbound traffic was secure without exposing ECS private IPs.

Challenge 3: Database High Availability

- **Problem:** Application required a resilient database setup to avoid downtime in case of failures.
- **Solution:** Deployed a **primary/standby PostgreSQL cluster** in a private subnet with replication and automatic failover.

Challenge 4: Protecting Against Malicious Traffic

- **Problem:** Public-facing ELB was vulnerable to attacks like SQL injection, XSS, and DDoS.
- **Solution:** Integrated **two WAFs** in front of ECS Auto Scaling Group instances, enforcing strong security policies before traffic reached the application.

7. Future Improvements

While the current deployment achieved scalability, availability, and security, several enhancements could be applied to strengthen and optimize the architecture further:

1. **Kubernetes Orchestration (CCE):**

    - Move containerized applications from ECS with Docker Compose to **Huawei Cloud Container Engine (CCE)**.
    - Provides better orchestration, rolling updates, and scaling for microservices.
2. **Global Content Delivery (CDN):**
    - Integrate **Huawei Cloud CDN** to cache static content closer to users worldwide.
    - Improves latency, load times, and user experience.
3. **CI/CD Automation:**
    - Implement a pipeline with **Huawei Cloud DevCloud** or GitHub Actions.
    - Automates testing, building, and deploying updated images to ECS or CCE.
4. **Serverless Functions (FunctionGraph):**
    - Offload certain lightweight backend tasks (e.g., notifications, scheduled jobs) to **FunctionGraph** for cost-efficiency.
5. **Advanced Monitoring & Logging:**
    - Use **Cloud Eye Service (CES)** with log aggregation and visualization tools (like ELK/Prometheus).
    - Provides better visibility into application performance and errors.
6. **Multi-Region Disaster Recovery:**
    - Deploy the PostgreSQL database cluster and ECS instances in multiple regions.
    - Ensures continuity if one region experiences downtime.

8. Conclusion

This project successfully designed and deployed a **secure, scalable, and highly available website architecture** on **Huawei Cloud**. Key achievements include:

- Deployment of **frontend and backend containers** on ECS instances with **Auto Scaling** for dynamic load management.
- Implementation of **multi-layer load balancing and WAF protection** to secure public-facing traffic.
- Integration of a **primary/standby PostgreSQL database cluster** in a private subnet for high availability.
- Secure outbound internet access for ECS via a **NAT Gateway**, avoiding public exposure of private IPs.
- Automation of container startup using **systemd services**, ensuring resilience on reboot.

This project demonstrates practical **cloud engineering skills** including infrastructure design, containerization, load balancing, database management, and secure network architecture. It reflects the ability to build production-ready cloud solutions that are both **robust and scalable**, making it a strong addition to a professional portfolio.