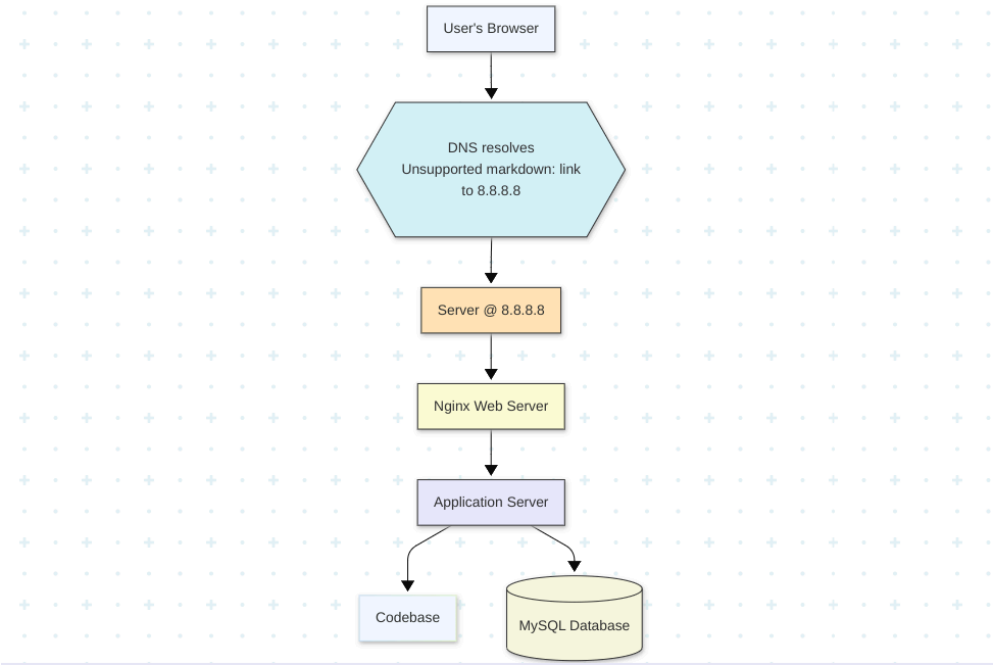
Task 0 - Simple web stack

A user types `www.foobar.com` in their browser. This domain points to the server at IP `8.8.8.8`, which hosts a basic web infrastructure. The single server runs a web server (Nginx), an application server, your codebase, and a MySQL database — all working together to deliver the website.



User Request:
A user types `www.foobar.com` in the browser → DNS resolves `www` to IP `8.8.8.8`.

Components and Their Roles

- **Server:**
  A physical or virtual machine hosting the entire web stack.

- **Domain Name:**
  `foobar.com` makes the site human-readable. `www` is a **DNS A record** pointing to `8.8.8.8`.

- **Web Server (Nginx):**
  Listens on port 80/443, handles HTTP(S) requests, serves static files, forwards dynamic requests to the app server.

- **Application Server:**
  Executes the backend logic (e.g., PHP, Python), processes user requests, connects to the database.

- **Application Files (Codebase):**
  The website's backend code (e.g., **`.php`, `.py`, `.js`)** that powers dynamic behavior.

- **Database (MySQL):**
  Stores and retrieves data like users, posts, and site content.

- **Communication:**
  Server communicates with the user's browser via **TCP/IP** over HTTP or HTTPS.

Infrastructure Issues

- **SPOF (Single Point of Failure):**
  One server — if it fails, the whole website goes down.

- **Downtime for Maintenance:**
  Restarting the web server (e.g., after code updates) causes service interruption.

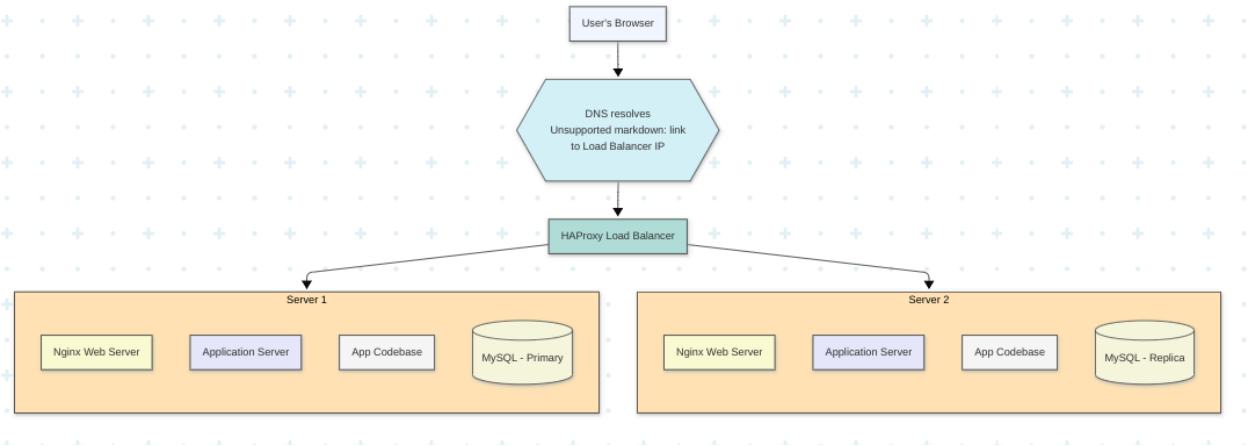
- **No Scalability:**
  One server can't handle too much traffic — no load balancing or redundancy.

Task 1- Distributed web infrastructure

A **distributed web infrastructure** is a system where the components of a website (web server, application server, and database) are spread across **multiple servers**. Instead of using a single machine to handle all user requests, the workload is **distributed** to improve **performance, availability, and fault tolerance**.

In this setup:

- A **load balancer** (HAProxy) receives all incoming traffic and distributes it between two backend servers.
- Each backend server has its own **web server**, **application logic**, and **database**.
- The databases are configured as **Primary** (write) and **Replica** (read), enabling redundancy.



Specifics About This Infrastructure

🔹 **For every additional element, why you are adding it**

- **Load Balancer (HAProxy):**
  Distributes traffic between backend servers to avoid overloading a single machine and to provide redundancy.
- **Two Backend Servers:**
  Allow traffic to be balanced. If one fails, the other can continue serving the application.
- **Databases (Primary and Replica):**
  A Primary DB handles writes; the Replica improves performance by handling read requests and serves as backup.

🔹 **What distribution algorithm your load balancer is configured with and how it works**

- **Algorithm: Round Robin**
  The load balancer sends each new request to the next available server in order.
  For example:
    - Request 1 → Server A
    - Request 2 → Server B
    - Request 3 → Server A again, and so on.
      This keeps the load evenly distributed when the servers are stateless.

🔹 **Is your load-balancer enabling an Active-Active or Active-Passive setup? Explain the difference.**

- **Setup: Active-Active**
  Both backend servers are online and serving requests at the same time.
  If one fails, the other continues, ensuring high availability.
- **Difference:**
    - **Active-Active:** Both servers handle traffic continuously.
    - **Active-Passive:** One server handles all traffic; the other only activates if the first one fails.

🔹 **How a database Primary-Replica (Master-Slave) cluster works**

- The **Primary database** handles all **write operations** (e.g., creating or updating data).
- The **Replica database** receives **real-time copies** of the Primary's data (using replication) and handles **read-only queries**.
- This improves performance and provides a backup in case the Primary fails.

🔹 **What is the difference between the Primary node and the Replica node in regard to the application?**

- **Primary Node:**
    - Accepts **read and write** operations.
    - Is the **source of truth** for the data.
- **Replica Node:**
    - Accepts **read-only** operations.
    - Syncs data from the Primary.

    - Helps reduce the load on the Primary and improves response times for read-heavy applications.

Issues with This Infrastructure

- **Where are SPOF (Single Points of Failure)?**

  - **Load Balancer:**
    If there's only one HAProxy load balancer and it goes down, no traffic can reach the backend servers. This is a **critical single point of failure**.

  - **Primary Database:**
    If the Primary database fails and there's no automatic failover to promote the Replica, the application can no longer perform write operations.

- **Security Issues (no firewall, no HTTPS)**

  - **No Firewall:**
    Without a firewall, all ports and services are exposed to the public. This increases the risk of:

    - Unauthorized access
    - Port scanning and exploitation
    - Denial-of-service (DoS) attacks

  - **No HTTPS (SSL/TLS):**
    Without HTTPS:

    - Data is transmitted in **plain text**.
    - Vulnerable to **Man-in-the-Middle (MITM)** attacks.
    - User credentials, cookies, and session data can be intercepted.

- **No Monitoring**

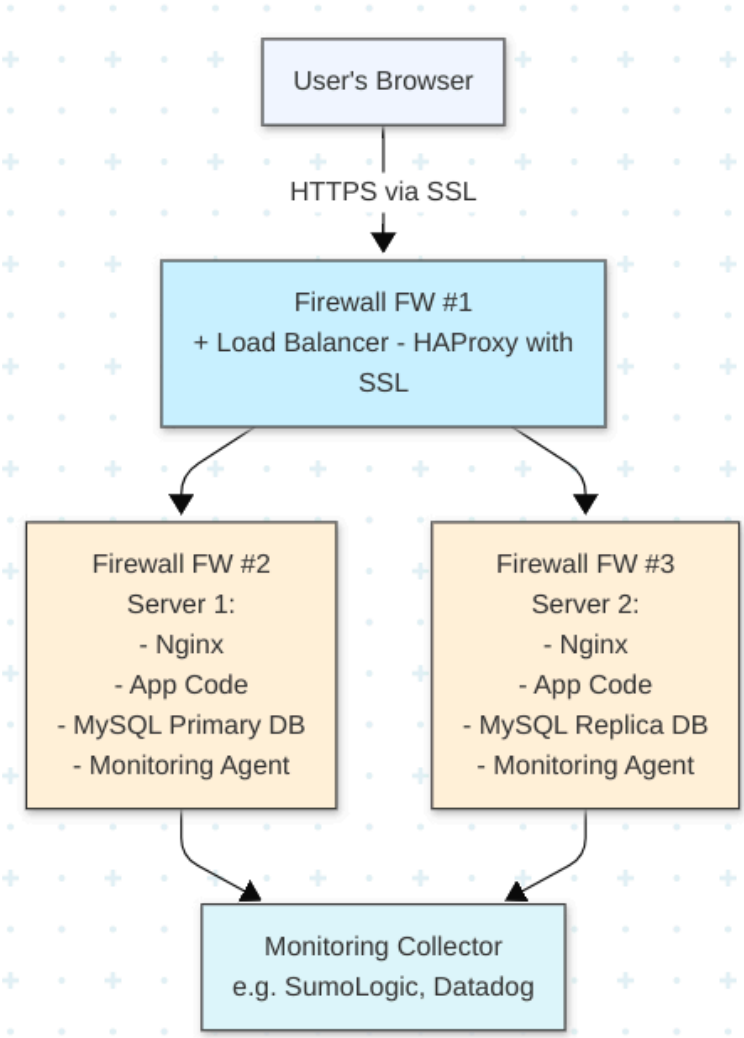
  - Without a monitoring system (like Prometheus, Datadog, or SumoLogic):

    - You cannot detect when a server or service goes down.
    - No visibility into performance, errors, CPU, memory, or disk usage.
    - No logs or alerts — making debugging and response slower.
    - No SLA (Service Level Awareness) or proactive incident response.

Task 2- Secured and monitored web infrastructure

This infrastructure secures and monitors a distributed web system.

- HTTPS ensures encrypted traffic.
- A firewall protects the load balancer (HAProxy), which distributes requests to two backend servers.
- Each server runs Nginx, app code, MySQL (Primary or Replica), and a monitoring agent.
- A centralized monitoring tool collects logs and metrics.



Specifics About This Infrastructure

- **For every additional element, why you are adding it**

  - **Firewall:**
    Protects each server by filtering incoming and outgoing network traffic based on security rules.

  - **HTTPS (SSL):**
    Encrypts data between the user's browser and the server to ensure privacy and prevent interception.

  - **Monitoring Agents:**
    Installed on each server to collect system metrics, logs, and performance data.

  - **Monitoring Collector:**
    Central server (e.g., SumoLogic or Datadog) that receives and displays data from the agents for analysis.

- **What are firewalls for**

  - To block unauthorized access.

  - To allow only specific traffic (e.g., port 80/443 for web, 22 for SSH).

  - To reduce the attack surface of the system.

- **Why is the traffic served over HTTPS**

  - To **encrypt** communication between client and server.

  - To **protect sensitive data** (like login credentials or personal info).

  - To **prevent MITM (Man-in-the-Middle) attacks** and data leaks.

  - To improve **SEO and browser compatibility**, as most modern browsers require HTTPS.

- **What monitoring is used for**

  - To detect issues in real time (downtime, CPU overload, memory spikes).

- To track performance, availability, and usage metrics.
- To trigger alerts when thresholds are breached.
- To assist in debugging and post-incident analysis.

🔹 **How the monitoring tool is collecting data**

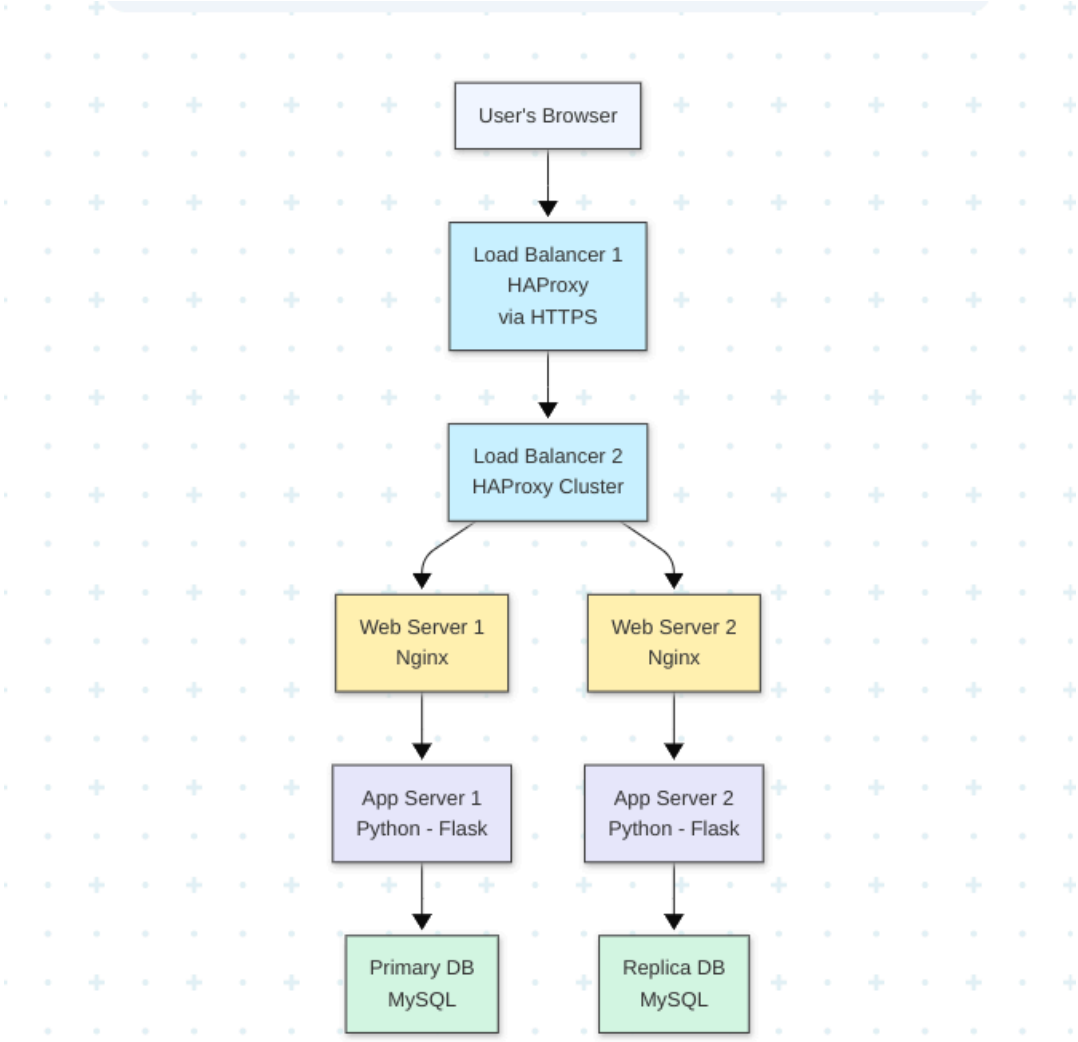
- **Monitoring agents** run on each server and:
    - Collect logs (e.g., Nginx access logs, syslogs).
    - Gather metrics (CPU, memory, disk, uptime, etc.).
    - Send this data to the central **Monitoring Collector** via secure channels.

🔹 **Explain what to do if you want to monitor your web server QPS (Queries Per Second)**

- Configure your monitoring agent to read from:
    - **Nginx access logs**, or
    - **HAProxy statistics endpoint**
- Parse log lines or request counters to calculate:
    - Number of HTTP requests per second
- Visualize this data in the monitoring dashboard (e.g., Datadog, Prometheus + Grafana).

Task 3 - Scale up (Scalable Web Infrastructure Design for www.foobar.com)

This diagram shows a scalable and secure web infrastructure with two HAProxy load balancers, two web servers running Nginx, two application servers using Python and Flask, and a MySQL database with primary-replica setup. Traffic from the user is encrypted via HTTPS and distributed across the system for high availability and performance.



This infrastructure separates the web server, application server, and database into **dedicated servers**, and introduces a **clustered load balancer** setup for high availability.

Specifics About This Infrastructure

1. **Second Load Balancer (HAProxy Cluster)**

To eliminate the **Single Point of Failure (SPOF)** at the load-balancer level.
If Load Balancer 1 fails, Load Balancer 2 can still route traffic, ensuring high availability.

2. Extra Server (to split components

To **isolate roles** and prevent resource conflicts:

- Web server doesn't fight with app logic or DB processes.
- Each layer can be scaled or maintained independently.

3. Dedicated Web Server (Nginx)

To handle **HTTP/HTTPS requests**, serve static content (images, CSS, JS), and forward dynamic requests to the app server. This improves performance and response time.

4. Dedicated Application Server (Flask, Django, etc.)

To execute business logic, process user input, handle sessions, APIs, and connect to the database. This separation ensures that backend logic doesn't interfere with web traffic handling.

5. Dedicated Database Server (MySQL/PostgreSQL)

To manage persistent data (users, content, sessions) securely and efficiently.
This setup improves scalability, simplifies backups, and allows for database tuning separate from app/web layers.