Infrastructure Overview:

One-Server Web Stack for www.foobar.com

Scenario:

A user opens their browser and types www.foobar.com to access your website.

Components Used:

- **1 Server** (with IP 8.8.8.8)
- **Nginx** (Web Server)
- **Application Server**
- **Application Files**
- **MySQL Database**
- **Domain Name** (foobar.com, with a www subdomain)

Request Flow (Step-by-Step)

1. **User Request**:
   A user enters www.foobar.com in the browser and hits enter.
2. **DNS Resolution**:
   The browser queries DNS to resolve www.foobar.com to its IP address.
   - The **DNS A record** (Address record) maps www.foobar.com to IP 8.8.8.8.
3. **Connection to Server**:
   Browser makes an HTTP request to IP 8.8.8.8.
4. **Nginx (Web Server)** receives the request:
   It decides how to handle the request — static content is served directly, dynamic content is forwarded.
5. **Application Server**:
   Nginx forwards the dynamic request (e.g., Python, PHP, Node.js app) to the application server.
6. **Application Logic**:
   The application server executes the code (business logic, templates) and might request/modify data in the MySQL database.

7. **MySQL Database**:
   Stores the website's dynamic data (e.g., users, posts, settings). Sends requested data back to the application server.
8. **Response**:
   Application server sends the response to Nginx, which then sends it back to the user's browser.

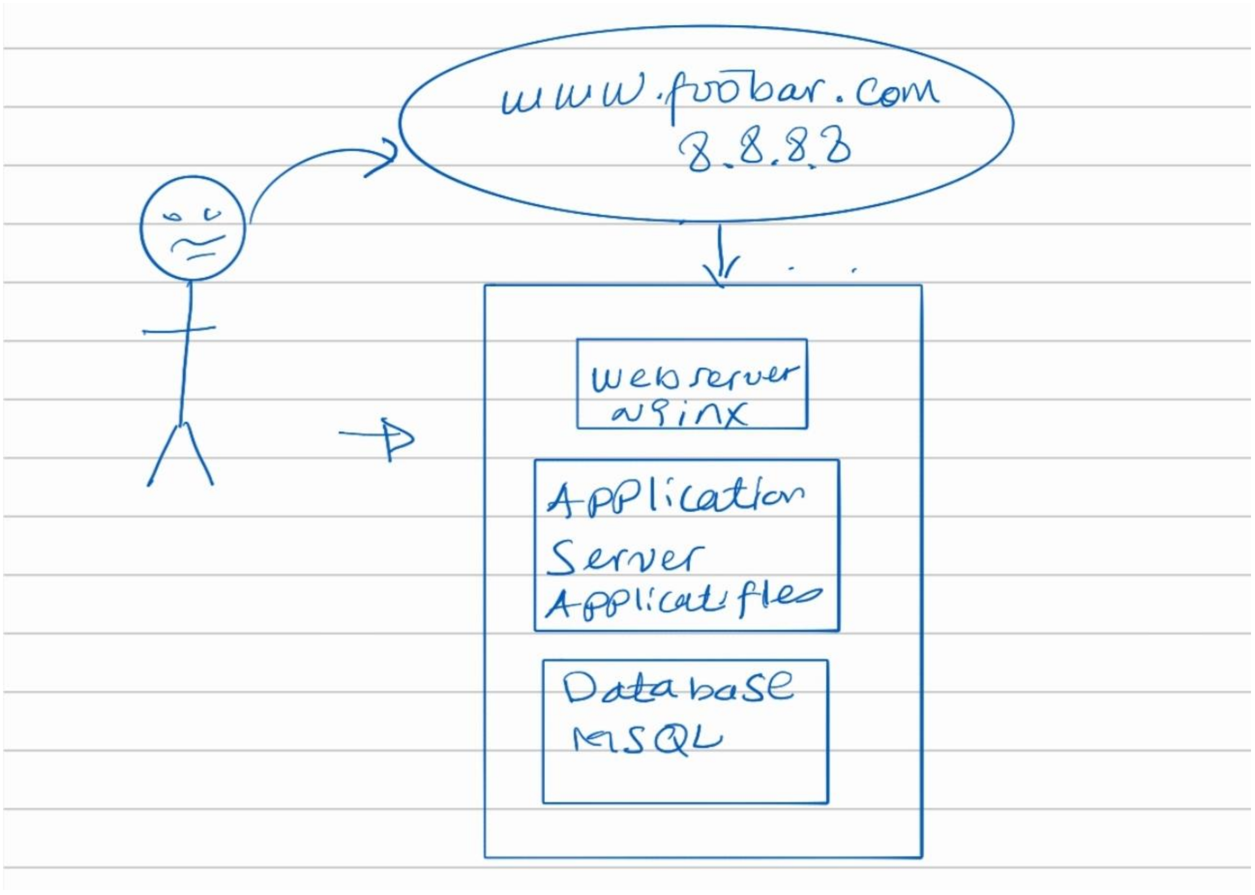
Component Roles Explained

Component	**Role**
Server	A physical or virtual machine running OS and software needed to host the site.
Domain Name	Human-readable name (foobar.com) that maps to IP address.
DNS Record (A Record)	The www subdomain is an **A record**, pointing to server IP 8.8.8.8.
Web Server (Nginx)	Handles incoming HTTP requests. Serves static files or passes dynamic ones to app server.
Application Server	Runs backend code, processes dynamic content, talks to the database.
Application Files	The source code that defines the logic of the web application.
Database (MySQL)	Stores and manages structured application data.
Communication Protocol	The server communicates with the user via **HTTP/HTTPS** using TCP/IP.

Issues with This Infrastructure

Issue	**Explanation**
Single Point of Failure (SPOF)	If the only server goes down, the entire website becomes unavailable.

Issue	Explanation
Downtime for Maintenance	Updating the app (e.g., pushing new code or restarting Nginx) causes brief or total downtime.
Scalability Limitations	All traffic goes to a single server. High traffic can overwhelm it, causing slowness or crashes.



Summary

This one-server stack is ideal for small websites or early development stages. It's simple to set up and maintain but has several limitations:

- No fault tolerance
- Downtime risks during deployments
- Cannot handle large traffic

For production-grade applications, this setup would need to evolve into a multi-server infrastructure with load balancing, redundancy, and autoscaling capabilities.