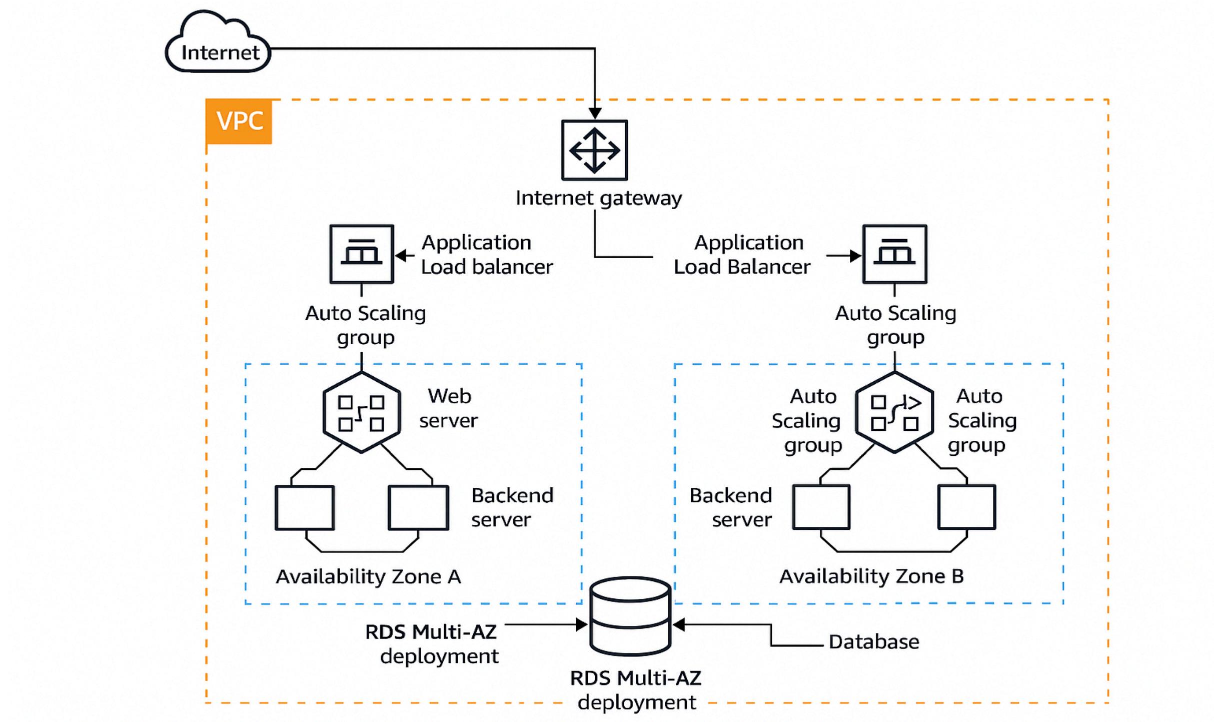
Part B: Durable System Design

Here we focused on improving the system to remain operating even if something malfunctions, like an EC2 instance failing or one of the availability zones going down. The goal was to guarantee that users receive continuous service irrespective of what takes place in the backend, without any disturbances.

What We Changed for High Availability
We improved the basic setup from Part A in several important ways to make things more robust and fault-tolerant:

Web Server
An Auto Scaling Group running at least two t3. medium EC2s replaces the one EC2 web server. Two Availability Zones are used so that requests can still be handled even if one zone fails.
if one zone fails. To divide inbound traffic across the servers, we also put in an Application Load Balancer.

Figure: Durable System Architecture (Multi-AZ + Auto Scaling)



Backend Server
Here, the same concept holds; two t3. large instances run the backend under an Auto Scaling Group. This one only manages inner traffic from the web layer; others are also positioned across two AZs and sit behind yet another Application Load Balancer.

Databases
The database in Part A was only in one availability zone. That's dangerous. Therefore, we currently use an RDS MySQL Multi-AZ deployment, which means AWS keeps a standby replica in another zone and changes to it if anything goes wrong. In this way, we avoid downtime and don't lose data.

Network + Security

Only the web layer uses an Internet Gateway to connect to the internet; the remainder of the system is still functioning inside a VPC. The backend and database are not open to view. We also used security groups to control who can communicate with what inside the network.

Why It Counts

This configuration shields the system from:
Failure events
Great traffic jumps
Load balancers, Auto Scaling, and Multi-AZ RDS allow the system to heal and scale itself no human intervention is required in case of a failure.