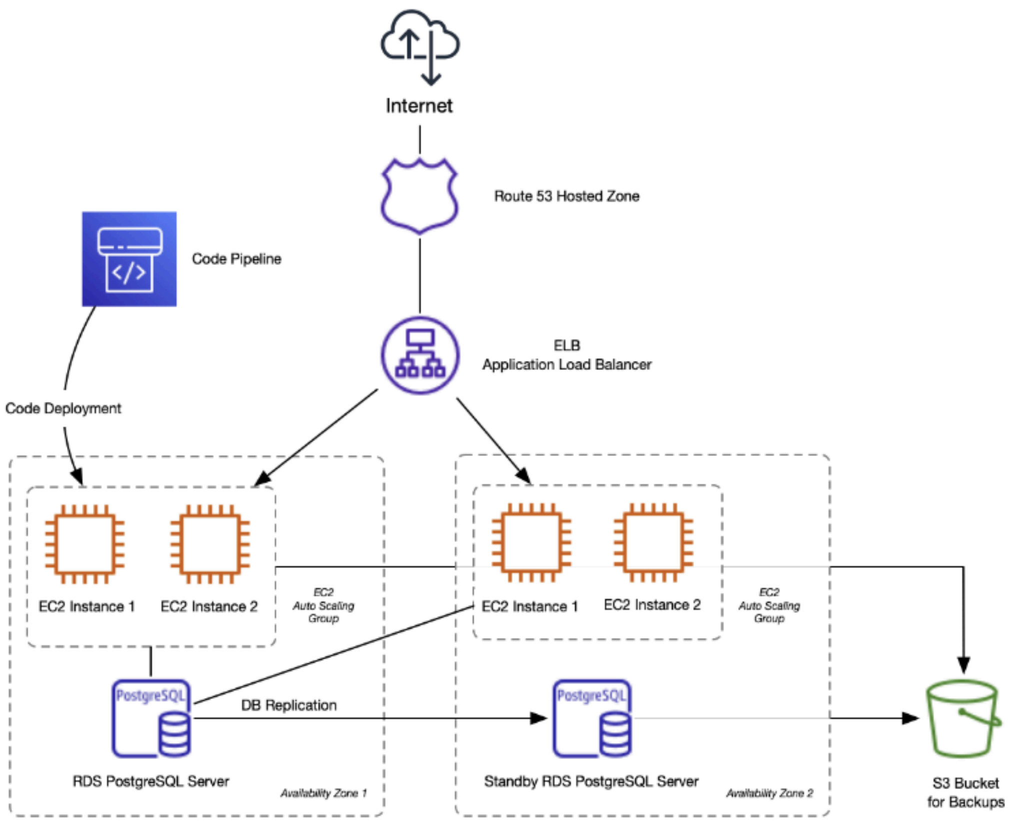
Architecture overview

We'll break this into layers: frontend, backend, database, deployment strategy, and monitoring.

Solution diagram



🔹 Route 53 (DNS Service)

- Manages your domain name and routes traffic to the correct AWS resources.
- Chosen for its deep integration with other AWS services and ability to handle health checks and routing policies.

🔹 **Elastic Load Balancer (ELB)**

- Automatically distributes incoming application traffic across multiple EC2 instances.
- Helps ensure high availability and smooth scaling, and protects against a single point of failure.

🔹 **AWS Elastic Beanstalk (with Auto Scaling)**

- Manages deployment, scaling, and health monitoring for your Flask application.
- Supports **blue/green deployments**, which reduce downtime during releases — a key concern for your team.
- Automatically scales EC2 instances based on traffic, so you can handle peak loads without overpaying during quieter periods.

🔹 **Amazon RDS (PostgreSQL)**

- Fully managed PostgreSQL database, replacing your local instance.
- Supports multi-AZ deployment for high availability and built-in backups for disaster recovery.
- Reduces operational overhead and improves performance consistency.

🔹 **Amazon S3**

- Hosts your static content (e.g., images, JavaScript, CSS, or even your entire React SPA).
- Inexpensive, durable, and integrates easily with CloudFront (if needed for global content delivery).

🔹 **AWS CodePipeline**

- Automates your deployment pipeline from Git to production.
- Ensures you can roll out changes frequently, safely, and with zero downtime when combined with Elastic Beanstalk’s deployment options.

💰 Cost Considerations

We understand cost control is important, especially as a startup. While we won't dive into exact figures, here's a breakdown of how costs may behave month-to-month:

- **Elastic Beanstalk (EC2 Auto Scaling Group):**
  - Costs are driven by the number and type of EC2 instances.
  - You can scale down to a single instance during off-hours and scale up during peak loads, which keeps costs efficient.
  - As your traffic grows, the autoscaling group ensures you're only paying for the compute you need.
- **RDS:**
  - You'll pay based on the instance type, storage, and backup retention.
  - Like EC2, RDS can be sized up/down as needed and supports Reserved Instances for significant discounts once usage is stable.
- **S3:**
  - Very cost-effective for static asset hosting.
  - You're billed by storage used and the number of requests — both tend to remain low-cost unless you serve high-volume media.
- **Route 53 & ELB:**
  - Route 53 charges a small monthly fee per hosted zone plus traffic-based costs for DNS queries.
  - Load balancer costs are based on hours active and amount of data processed.
- **CodePipeline:**
  - Has a modest monthly cost per pipeline, which remains predictable and low even with frequent deployments.

If you continue growing at a linear pace, your AWS costs will scale accordingly. Importantly, this setup gives you the flexibility to optimize at any stage — for example, shifting to Reserved Instances or Spot Instances for EC2 when ready.

✅ In Summary

This architecture will address the performance and reliability issues you’re currently facing, support your projected growth, and eliminate downtime during deployments. It also establishes a strong foundation for disaster recovery and long-term scalability — all while keeping monthly costs tied closely to actual usage.

AWS APAC Solutions Architecture virtual experience program on Forage - June 2025

- Designed and simple and scalable hosting architecture based on Elastic Beanstalk for a client experiencing significant growth and slow response times.
- Described my proposed architecture in plain language ensuring my client understood how it works and how costs will be calculated for it.

AWS APAC Solutions Architecture virtual experience program on Forage - June 2025

- Designed and simple and scalable hosting architecture based on Elastic Beanstalk for a client experiencing significant growth and slow response times.
- Described my proposed architecture in plain language ensuring my client understood how it works and how costs will be calculated for it.