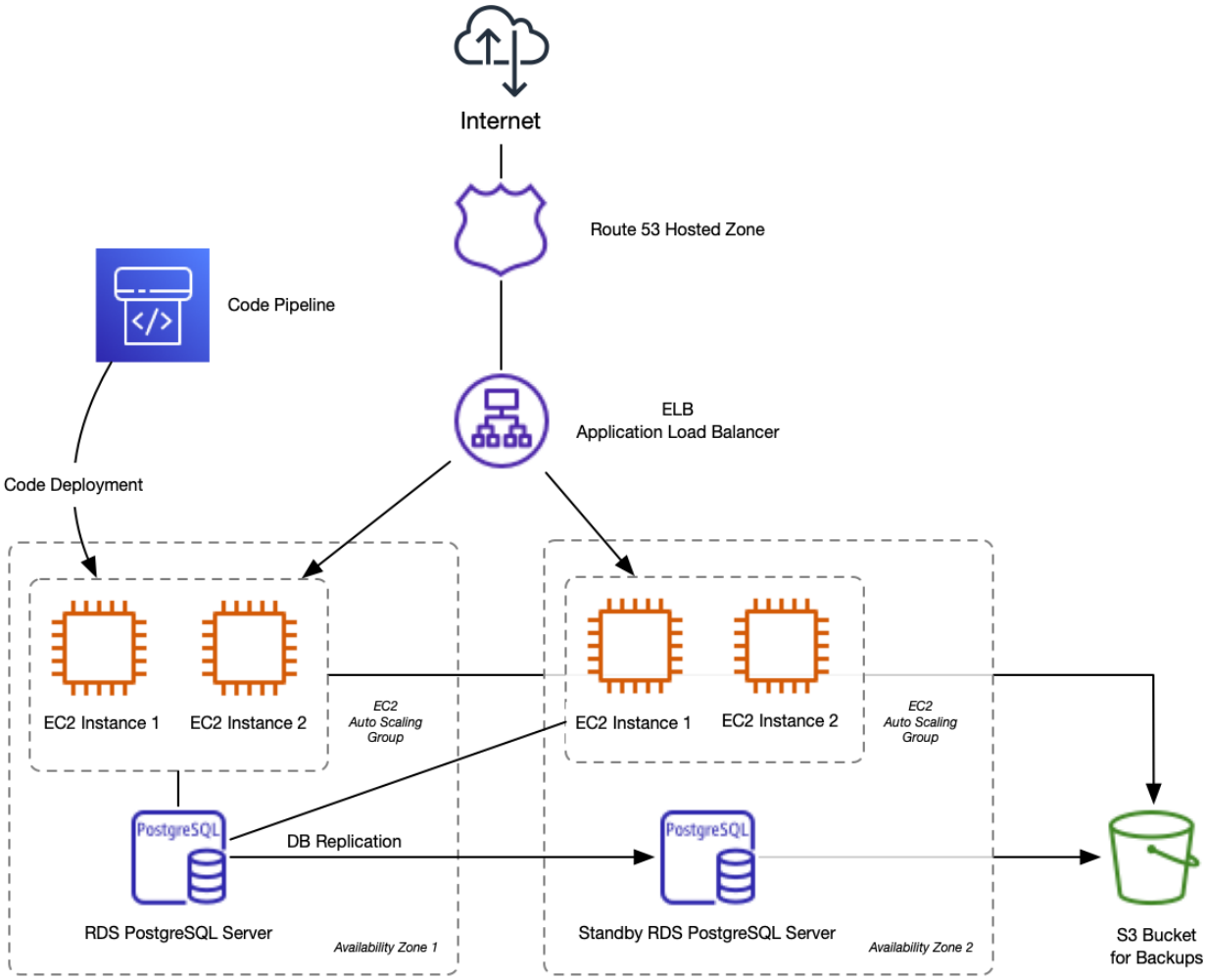
Hi Lilly,

Thank you so much for your email.

I've prepared an architecture diagram that illustrates how you can leverage more AWS services to improve your application's current performance, allow for future growth, and improve your deployment process.



It is based around Elastic Beanstalk which brings together HTTP load balancing, application servers and database servers under one umbrella. After setting up these services using Elastic Beanstalk, they can be scaled independently. For example, we can increase the resources available to the database server without affecting the app server, and vice versa.

Please find a system architecture diagram attached to this email. I will explain these parts in detail and why they were chosen.

We know that growth and scalability is important for this client.

Route 53 Hosted Zone
We will set up your DNS in AWS Route 53, AWS's DNS management system. This will allow your DNS to tie into the load balancer and make routing changes automatically.

DNS zone hosting is fairly standardized/ubiquitous so we don't need to explain it much.

Elastic Beanstalk
Your application will be deployed to Elastic Beanstalk. Simple command line tools are provided to deploy your application. You may have to make a few small changes to your code configuration for it to work on Elastic Beanstalk but we can provide guidance on that. We also consider Code Pipeline (more information further on) which could assist with deployment.

The Elastic Beanstalk environment is in three parts: Elastic Load Balancer, EC2 Compute Instances and an RDS PostgreSQL server.

Explain why Elastic Beanstalk was chosen: for simplicity.

Elastic Load Balancer
When setting up Elastic Beanstalk we can also have it set up a load balancer for us. This allows traffic to be routed to different EC2 instances in one or more availability zones. This provides redundancy and scaling.

Explain why Elastic Beanstalk was chosen: for simplicity.

The next four paragraphs just explain the benefits of using these services:

EC2 Auto Scaling Group
The main benefit of using Elastic Beanstalk is the way it ties into EC2 to automatically scale the number of instances your application runs across. We can set up rules to automatically add or remove instances as the load on your application increases. This will help if your application has peak times where the load is greater as you would only be paying for capacity when you need it.

RDS PostgreSQL Server
Elastic Beanstalk can automatically set up and manage the PostgreSQL database in RDS too. The benefits of running PostgreSQL on a separate server from your application have already been covered above.

S3
We can utilize S3 for RDS backups and backups of your files. In the future we can also consider serving static files or media from S3 if your application grows to require this.

Multiple Availability Zones
To provide more redundancy, we can set up the application in multiple availability zones (AZ). An extra RDS instance can be set up which the production instance replicates to. You may find that with the other improvements that you don't need another AZ too, as obviously there is extra cost involved.

Multiple AZs can be useful, but not all the time. We know the client's budget is not unlimited so we propose it but let them know that it's not strictly necessary.

Code Pipeline
As your application gets more users from all over the world you will find it harder to set up maintenance windows to allow downtimes during deployments. We can use Blue-Green deployments to launch a new Elastic Beanstalk instance for a new version that's deployed. The existing production environment is called the Green environment. Once the new environment (Blue) is up and running then the production URL can be switched to point to it, and it will start taking requests. This then becomes the Green environment and the previous production environment can be stopped.

This can be a manual process, or Code Pipeline can be configured to work with Elastic Beanstalk to perform this automatically. This would take the place of an existing CI/CD pipeline if you already use one.

Explain how CodePipeline works and how we can achieve zero-downtime deployment for them, another problem they initially mentioned.

Costs
After some initial performance testing to get a baseline figure we can decide what size instances to start with. This will then allow us to estimate costs going forward.

Some costs will be roughly the same every month. For example, your RDS instance price is fixed until it's moved to a smaller or larger instance. Similarly, if you were to run the same sized EC2 instance(s) for the whole month the cost of them is the same regardless of the number of requests. Other components, such as the Load Balancer accrue costs with more requests.

When we have a better picture of the number of requests you will be servicing, and the resources required for these we will work with you and use the AWS pricing calculator (https://calculator.aws/) to get concrete pricing numbers.

We can't provide concrete pricing numbers until we know the exact resource requirements, but we can give guidelines on how it can change in the future, plus work with them to get some ballpark figures.

Please let me know if you need any more information.

Kind regards,
Fred Kibutu