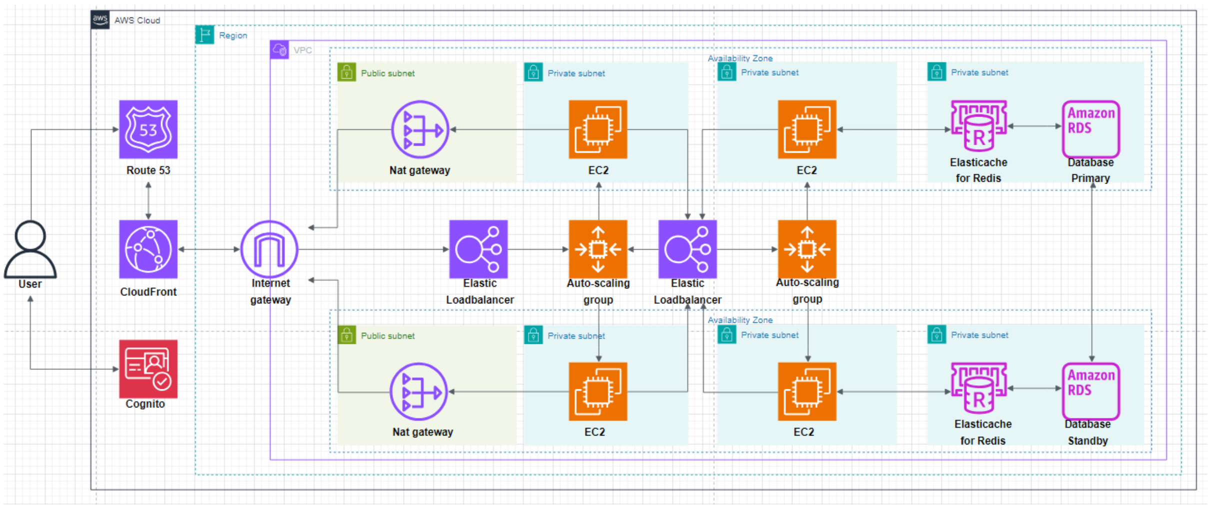
Architecture



Computing

Our goal was to make a highly available, fault tolerant and scalable computing strategy for our application. Our architecture makes use of many of the AWS well-architected frameworks design principles such as not having to guess capacity, horizontal scaling, and automatic failure recovery.

At the heart of our computing strategy are EC2 instances. We chose EC2 as our processing method because of its flexibility. By utilizing regular EC2 instances instead of for example elastic beanstalk, we do not limit our options in the future. This choice comes at the expense of added difficulty, as we need to define the instance environments ourselves.

Our architecture consists of 2 computing tiers. Business logic and webserver are separated into different EC2 instances. Together with the database tier, this makes our architecture a 3-ter system. Separating architecture into multiple tiers offers benefits such as improved scalability as we only must scale the parts we need. Added benefits come from improved security as accessing our data and logical tiers becomes ever more difficult.

We chose the method of scaling computing infrastructure to be auto scaling groups. Auto scaling groups will be set up to automatically scale our EC2 instances horizontally. This works particularly well in our environment that utilizes multiple availability zones. All availability zones will be scaled in or out concurrently as needed.

Elastic load balancers are the final piece of our computing strategy. ELB's are a part of an AWS region. They can route requests to EC2 instances that have available processing power. ELB's help us avoid cases where problems in one availability zone hinders performance. Traffic can be routed to a working EC2 instance safely. ELB's also work well in web applications that need to save session data.

Database

When designing the data storage system we wanted to focus on speed, availability, and reliability. Designing databases is its own skill altogether and when we were discussing our prior experience, knowledge of SQL databases was thorough. Despite some advantages a no- SQL database could provide in speed, the well-structured SQL database along with expertise in the field lead us to choose RDS as our AWS service of choice.

The structured nature of an SQL database works well for our application. Our application stores user, organization, and event data that is uniform. All events, organizations and users need to be interconnected which can lead into somewhat complex queries. SQL databases work well for complex queries when a no-SQL counterpart could have some loss of performance.

Another key reasoning for choosing RDS is that it is a fully managed service. This means that AWS can take care of as many administrative tasks as we please. Tasks such as software patching and provisioning can be managed by AWS.

Our database infrastructure consists of two AWS RDS instances. One of the instances is the primary instance, and the other one is a second instance, which replicates the data from the primary. The second instance is the backup in case the primary becomes unavailable. It can also be used as a read only instance, lightening the load for the primary. This system makes our data very durable and difficult to lose.

In addition to RDS instances our database structure makes use of Elasticache for Redis. Elasticache for Redis is a caching tool that can be used to make cached database queries much faster. The caching strategy we plan to implement is the Lazy loading method. In this method, our cache only caches data that is needed. Lazy loading is the preferred method of caching for social media applications.

Network

Our networking components work to allow easy and efficient access to our website. While accessing our website needs to be highly available, it also needs to be able to keep intruders out.

To make it possible for users to find our service in the first place, we use the Route 53 service. One benefit Route 53 offers is the ability to register domain names. Additionally, Route 53 allows us to set up DNS routing. This can be helpful in case we decide to go global with our application. People from different parts of the world can be routed to different versions of the website, allowing us to offer a more relevant experience for our users.

The second service we use that does not reside within our VPC, is the AWS CloudFront. CloudFront is a mixed bag of benefits in all areas of the application. The main benefits in the first phase of development are reduced costs and increased performance, thanks to its caching capability. Several user generated queries can be quickly responded to without the need to query our actual databases. Further benefits come from CloudFront’s ability to use edge locations. Users can be served closed to their location, reducing latency. Another benefit that has potential to be a life saver in the current geopolitical environment is the DDoS protection that can be enabled with CloudFront.

Communication to and from our VPC goes through an internet gateway. Our VPC is split into multiple components for increased availability and security. Within the VPC, traffic is routed to each availability zone by using Elastic Load balancers and Auto-scaling groups.

Our application makes use of redundancy by residing in multiple availability zones. This is to ensure functionality of our application in a scenario where an entire availability zone goes down.

In our application, EC2 instances and database instances reside in private subnets. This allows us to further limit access to the instances from outside of our VPC. SSH can still be used to complete any maintenance tasks EC2 instances may require.

Instances inside of our private subnets can only send messages to the internet through public subnets. Our public subnets contain only a Nat gateway, that is used to send messages back to the internet and our users.

In every step of the way, security groups are in place. This is to ensure only the type of traffic we want, can traverse our environment.