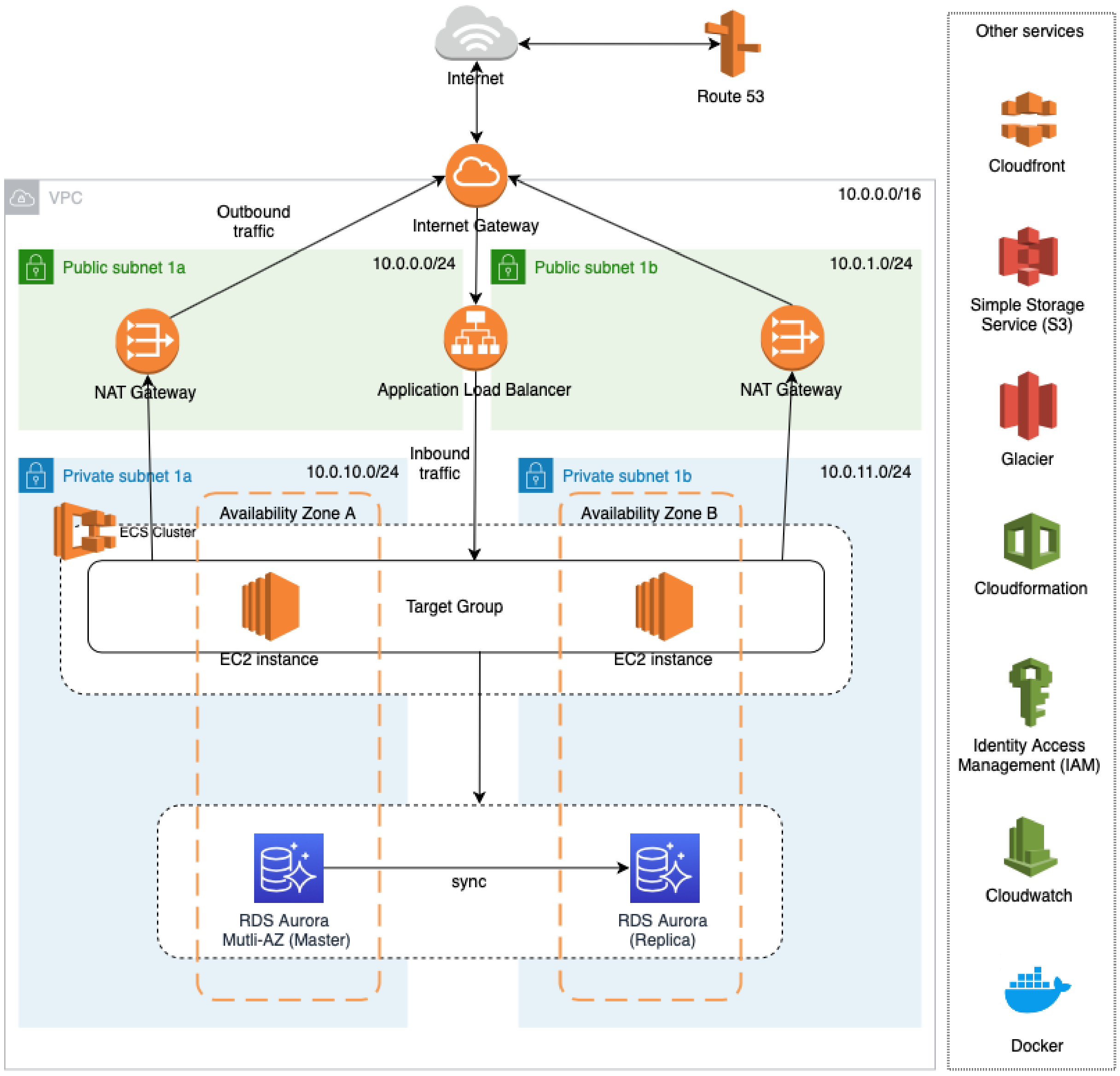
Author: Lim, Chooi Guan

## AWS PROPOSED ARCHITECTURE



## OVERVIEW

To handle demand and infrastructure needs, I propose using the Elastic Container Service (ECS) running on Elastic Compute Cloud (EC2) instances. Note that this will require prior containerization of the web application, which I have provided a fully working sample of in a git repo. EC2 Auto Scaling will handle scaling up of EC2 instances depending on workload increases. This can be achieved by creating Cloudwatch alarms which can trigger specific actions to take when certain metrics are in a state of alarm. For example, if the ECS cluster is running high on CPU usage, you can create an action to add another EC2 instance. When the load has subsided, then another action can be executed to remove the additional EC2 instance. Load distribution is handled by the Application Load Balancer (ALB) which uses the round robin algorithm by default for routing. Self healing of services will be handled by ECS and self healing of EC2 instances will be handled by EC2 Auto Scaling as well.

Disaster recovery can be planned in advance and automated using Cloudformation. Utilising Infrastructure as Code (IaC), a full blueprint of the service(s) and infrastructure can be saved as a Cloudformation template and deployed in another region in a matter of minutes. For example, if the Asia Pacific region for Singapore (ap-southeast-1) is down, identical infrastructure and service(s) be deployed using the Cloudformation template into a US region (e.g. us-east-1) that is available. Subsequently, all that remains to be done is to modify the hosted zone's record set record in Route 53 (responsible for handling DNS) to point to the newly created Application Load Balancer. A more costly alternative is to deploy the application to multiple regions and enable latency based routing which can also be enabled in Route 53. This is an active-active approach.

The database can be migrated to Amazon Aurora which is fully managed RDS. Aurora can be deployed in multiple availability zones and in a private subnet. Amazon Database Migration Service can be used to migrate a MySQL database to Amazon Aurora. Two Aurora database instances will be deployed, one for redundancy purposes. A cloudformation template for Aurora is available in my git repo if required.

Content will be distributed using Cloudfront so that users will have the lowest latency possible when retrieving media content. Cloudfront is a content distribution network which speeds up distribution of static, dynamic content and media files by delivering content through a worldwide network of data centers, i.e. edge locations. This means that even users which are far away will be served from an edge location which is geographically close to them, thereby providing the lowest latency for them.

All data in transit will be protected by enforcing encryption in transit. This means using secure protocols such as SSL. The ALB supports SSL offloading and can thus save compute cycles. Only the HTTPS listener should be enabled for the ALB, and the security group attached to the ALB should only allow the HTTPS protocol if no other ports are required. All services (in this case, the only service is the website) are deployed as containers on EC2 within the private subnet and fronted by the ALB. Therefore, each EC2 will have its security group's ingress restricted only to the ALB. For SSL certificates, this can be managed by AWS Certificate Manager (ACM). Communication with Cloudfront as well as Aurora will also be via HTTPS. To further increase data security in transit within the private subnet, a service mesh such as Linkerd on ECS can be deployed to enable all HTTP traffic based communication to use mTLS (mutual TLS) between services in the future. All data that is at rest (i.e media content) will be stored in Amazon S3 and not be accessible to the public by default. Instead the data will be served via Cloudfront and only accessible through Cloudfront's origin access identity. You can also further increase security by additionally encrypting all data in S3 by default. Application secrets such as passwords can be securely stored and encrypted using the AWS Systems Manager Parameter Store or AWS Key Management Service.

Access to the AWS environment will be managed by Identity Access Management (IAM). Only IAM users will have access to the AWS environment and will be assigned different permissions and roles depending on each user's requirements.

S3 buckets can be configured with lifecycle policies to archive objects older than 6 months into Amazon Glacier.

Multiple environments can be created and managed using Cloudformation. For example, the same blueprint can be used to deploy to a test environment, dev environment and production environment. To save costs, you can use fewer instances for testing and dev environments, but each environment should have the same identical infrastructure blueprint, this is a best practice known as "host parity".

## ASSUMPTIONS

The application is a website running on port 80 (HTTP) or 443 (HTTPS). The client's customer base will be in Singapore or a region supported by AWS. Policies about data protection laws and where data is stored have been excluded in the design consideration. The version of MySQL is supported by Aurora RDS, i.e. 5.5, 5.6 or

5.7. Migrating the database has been excluded, but instructions on how to migrate to Aurora can be found here.

## CONCLUSION

The architecture above can be built upon and easily evolve to being serverless in the future. For example EC2 instances can be migrated to Fargate which allows you to run containers without managing EC2 instances. Aurora can be migrated to Aurora serverless. By adopting containerisation, time will be saved in configuring environments not only for developers, but also when performing deployments and running tests. The organization will be able to easily grow organically in team size and increase software development velocity since there will be fewer "it works on my machine" issues. Additionally containers can be deployed in any container supported platform and can be started and shut down quickly.

A fully working Cloudformation blueprint of the architecture (along with a demo) can be found in my git repo.

If you have any questions about the proposal or require help in making the necessary application level changes, feel free to contact me.