Tech Stack **Game Service**
Muhammad Soliman m.soliman@hotmail.com

I can depict how the system should be architected if required

1) I recommend the use of **AWS** cloud in which I have extensive experience especially in architecting highly available and scalable solutions:

2) Amazon **DynamoDB** can be used or may be cluster of **MongoDB** to be deployed in private subnet in a customized VPC network with having **NAT** servers in the public subnet for maintaining patch updates.

When using DynamoDB, we should only care about LSI/GSI/Read Capacity/Write Capacity without any infrastructure maintainance however in MongoDB we should make the deployments for the cluster (primary/secondary replicas)

3) Elastic Load Balancer (**ELB**) should be used in front of service layer to load balance traffic to only health services or endpoints (TCP Ping) . On premises we can use **Nginx** instead.

It should distribute the traffic to different instances and make sure that it distribute to healthy ones.

4) Service layer should have multiple Availability Zones (**AZs**) to support high availability and in each zone we should have **Autoscaling** configured to make the service elastic in case of high workloads.

e.g. Keeping the service deployed in multiple AZs keeps it highly available as if one AZ failed the other one is still working.

5) **Containers** can be also used to host micro-services architectures using **Amazon ECS** by defining tasks to be run in AWS.

e.g. Build docker images based on nodejs and deploy it to registeries then pull anywhere and run directly. It is Keeping different environments (test,staging, production) consistent.

*6) Instead of using Servers or containers, I can also recommend using **Amazon Lambda** as **Serverless** architecture. It is based on computation power that we actually use which is less expensive than the whole machine we spin up and we might even have it **Idle** most of the time.*

e.g. Service can be completely run in Lambda however I did not use before heavily to know if there might be some limitations, it is based on requirements and experience.

7) **ChaosMonkey** can be used to test how much service is tolerant in case of failures, I did not used before however - https://github.com/netflix/chaosmonkey

8) Decoupling components can be done using services like **SQS** (Simple Queue Service) to send messages to queues and there should be workers behind which are polling messages from queues and updating the other system or the intended part of the system (usually this is used in case of critical integrations between systems and also in time-consuming functionalities like generating thumbnails from an image or transcoding video to different quality)

9) **Fanout** scenarios in case we need to make multiple actions based on specific action like logging, updating other system, updating database, and handling notifications can be achieved using **SNS** (Simple Notification Service) and **SQS** (Simple Queue Service)

e.g. Win 5 games for example and update db and logging and purchase automatically him a prize from Amazon as Award

10) Logging (plays history or explored games, stats like win/loses) can be streamed quickly to Amazon **Kensis** or Apache **Kafka** on premises.

11) Analysis (to analyze the recommended game based on his playing) can be done using Dataware house solutions like Amazon **RedShift**.

12) All security token generated after logins and that we should use to make subsequent HTTP requests/calls should be persistent in external storage like cluster of **Redis** or **DynamoDB**.

13) **CloudFormation** should be used to quickly and easily provision complete stacks (having resources/parameters/configurations) and tears them down. Also it simplifies cloning infrastructure to different regions in AWS and also source controlling our infrastructure as code.

14) **CodeCommit**, **CodeBuild, CodePipeline** and **CodedDeploy** can be used to automate the process of build, test and deployment.

15) **Route53** can be used either as internal or public DNS service.

16) **Last but not least,** Identity and Management Service (**IAM**) maintains security to provide access to different resources using u**sers/groups/roles** with even fine-grained control to make Identity Federation from company's infrastructure Identities (Active directory) or Web Identities (FaceBook/Google..etc) to AWS.