**AWS High Level Architecture**

Our aim is to build a micro service architecture for the backend system for Mintology Backend. The entire architecture would be divided into different functionalities (such as Auth service, NFT service etc). This would allow each service to be independent of others and be able to scale by itself and serve errors by itself. I believe it will help us in troubleshooting errors too.

***AWS Cloud Watch monitoring***

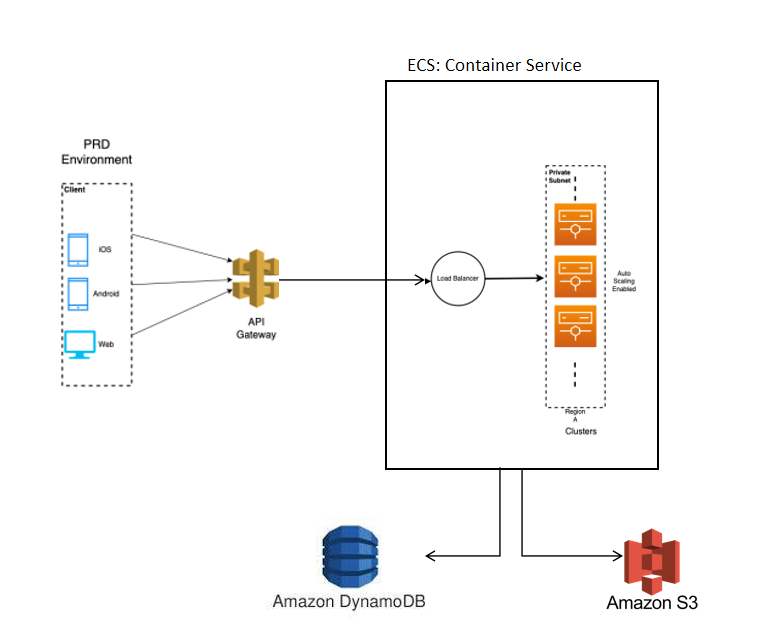
Its a monitoring service for Amazon Web Services (AWS) cloud resources and the applications you run on AWS. We use CloudWatch to collect and track metrics, collect and monitor log files, set alarms, and automatically react to changes in your AWS resources 1. It can monitor AWS resources such as Amazon EC2, ECR, ECS instances, Lambda, and Amazon Dyanmo DB instances. We also use it to generate alarms for errors in services and nft functions.

***System Testing:***

Divided the entire testing system into two:

1. *Load testing:*
   1. It is a type of performance testing that determines the performance of a system, software product, or software application under real-life based load conditions. It simulates multiple users accessing the program concurrently to test its performance under different load conditions 1. The goal of load testing is to identify bottlenecks and determine the maximum number of users or nft functions the system can handle
2. *Unit testing:*
   1. The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements. Unit tests are automated and are run each time the code is changed to ensure that new code does not break existing functionality

***The high level architecture:***



|  |  |  |
| --- | --- | --- |
| **S.No.** | **Cloud Resource** | **Description** |
| 1 | Load balancer | Managing the incoming load and ensuring all servers are getting a uniform number of requests.  Also helps in auto scaling by coding triggers into it |
| 2 | EC2 vs ECS(Container service) vs Lambda | Server instances to process user flows. We’ll decide on which to choose based on the type of scaling and application we use |
| 3 | VPC | VPC enables network isolation, allowing us to control inbound and outbound traffic. VPC also offers flexibility in configuring network settings, such as IP address ranges and subnets.  Think of it as a safe room inaccessible from any person outside but us inside the cloud system |
| 4 | API Gateway | API Gateway is essential for several reasons. Firstly, it provides a centralized and secure way to manage APIs, allowing us to control access, monitor usage, and enforce authentication and authorization policies.    Secondly, it simplifies API development and deployment by handling tasks such as request/response transformation, protocol translation, and caching. |
| 5 | MySQL vs RDS vs DynamoDB | Ensuring the right type of database to use based on the type of data we are storing and the read/write speed required |
| 6 | S3 buckets | To store larger size data such as Images, Face Models of users etc. |
| 7 | DEV and PRD environment | Best practice is to first deploy a new change in the backend on the DEV side. Test it on DEV and then release to the customer base via the PRD environment.  The DEV environment can have low availability and scalability to save on cost. |