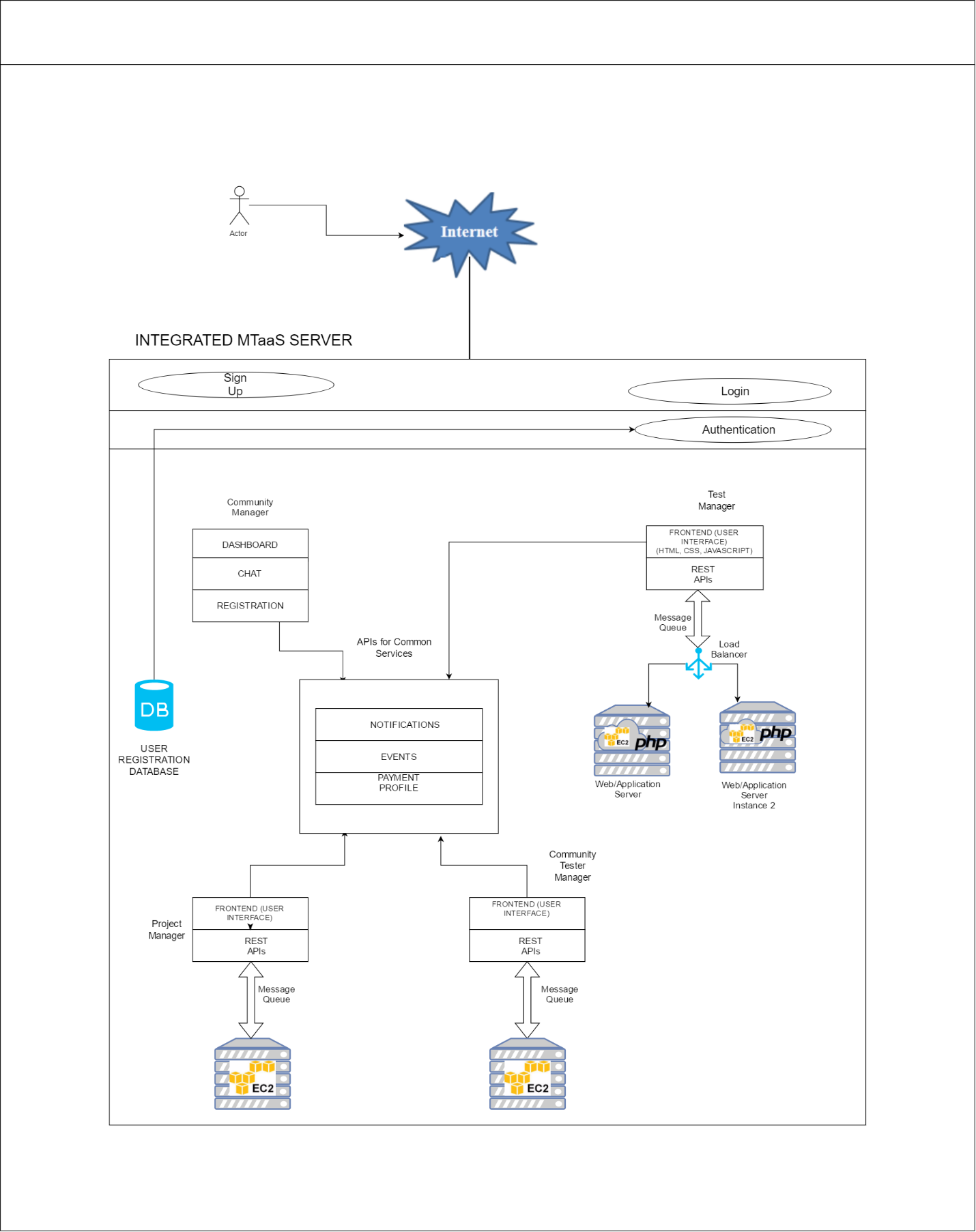
Below is the system architecture and infrastructure diagram. In this section, we’re going to explain how all the components of the project work together to give a complete Mobile Testing as a Service (MTaaS) platform that helps in testing mobile devices in a crowd sourced environment.



1. **MTaaS Architecture :**

The platform is divided into 4 main modules:

1) Community Manager

2) Community Tester Manager

3) Project Manager

4) Test Manager

All these modules have their specific functions and do not rely on other modules for most of the part. However, there are certain functionalities which these modules share, like the notifications and events. The APIs for these functionalities are exposed and they can be called from any of the 4 modules given above. The 4 modules may or may not be hosted on different cloud services. This facilitates the high availability which is desired in enterprise level applications. That way, even if one module fails, other modules might work. In other words, not all the services fail at the same time.

We have chosen the **Component 4** for our project. In the development of this module, we’re using Amazon EC2 cloud services to host our application as well as web server. We’re using PHP in the back end and HTML, CSS, JavaScript in the front-end development of this project.

The functionalities supported by Component 4 are as follows-

1. Create Test Cases/Problem statements for bugs
2. Read Test Cases/Problem statements for bugs
3. Update Test Cases/Problem statements for bugs
4. Delete Test Cases/Problem statements for bugs
5. View Test Cases/Problem statements for bugs

We’re using message queues for queuing the messages to web/application servers. The front end is being developed in the HTML and CSS. REST API calls go through the front end to the message queues and then to the servers.

For other components, any technology for front end and back end could be used, as long as APIs are consistent and configurable for rest of the architecture. Also, it is recommended that each of the service component be hosted on different cloud vendors or different cloud instances, so that service could be highly available.

1. **Cloud Infrastructure:**

MTaaS can be categorized as a SaaS, i.e. Software as a service. The only difference the MTaaS environment has from other SaaS environment is that here, we are providing a platform for Testers. Most of the SaaS services are meant for developers. Here, we are exploring a new realm in which Testing – which complements software development and is crucial to it, is being provided as a software service.

This discussion brings us to our next obvious question – What should be the infrastructure for this innovative platform?

Clearly, since we are building a SaaS environment here, we have to look for IaaS to support it. That is, Infrastructure as a Service. The only leading vendor and the reliable one at that is Amazon Web Services (AWS). AWS has many IaaS services that cater to the need of cloud developers. They have lots of options for all kinds of requirements. From lightweight services to heavy, compute intensive enterprise level services, Amazon has all the answers.

Let’s take a look at all the services provided by AWS and decide which one is most suitable for our platform.

Amazon EC2: Elastic Compute Cloud has following cloud instances

1) General Purpose

* T2
* M4
* M3

2) Compute Optimized

* C4
* C3

3) Memory Optimized

* X1
* R4
* R3

4) Accelerated Computing Instances

* P2
* G2
* F1

1. Storage Optimized I3 – High I/O Instances

* D2

All of these instances are specifically targeted to various computational needs. For the sake of this project, we will be focusing on general purpose instances.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | vCPU | CPU Credits / hour | Mem (GiB) | Storage |
| t2.nano | 1 | 3 | 0.5 | EBS-Only |
| t2.micro | 1 | 6 | 1 | EBS-Only |
| t2.small | 1 | 12 | 2 | EBS-Only |
| t2.medium | 2 | 24 | 4 | EBS-Only |
| t2.large | 2 | 36 | 8 | EBS-Only |
| t2.xlarge | 4 | 54 | 16 | EBS-Only |
| t2.2xlarge | 8 | 81 | 32 | EBS-Only |

The above table is from Amazon EC2 website. It shows different instances with their specifications and cost.

We’ll be using T2.nano instances for this term project as it is only in experimental phase right now. As the load on servers increases, we can scale up the instances or choose instances with higher computational power.

**Major Features of EC2**

1. Fully virtualized instances. We don’t have to worry about virtualization, it’s taken care of internally.
2. Elastic IPv4 addresses. Though cloud instances are dynamic in nature, we get a fixed IP address(static) which can be attached with our cloud instance.
3. Wide options for selecting the physical location of instances. This ensures the resiliency against natural disasters for distributed enterprise services. Amazon has many data centers across US and all over the world. You can choose from any of those data centers depending on latency and resiliency needs.
4. AMIs: Amazon Machine Images are operating System images that are ready to be launched on instances. We don’t have to worry about compatibility of OS and underlying cloud hardware.
5. Storage capabilities: EBS (Elastic Bean Storage) service by Amazon for storing EC2 instance data is very efficient in performance. It is persistent storage, even if instances might not be so. Also, they support memory snapshot capabilities. These snapshots can be used as the starting point for new Amazon EBS volumes, and can protect your data for long term durability.
6. Scaling up/down on the fly: You can reconfigure your instances on the fly if workload is varying. The instances can be reconfigured based on the memory, processing power needed for current processes being executed by the EC2 instances.

**Queueing:**

Message queues are necessary for streamlined flow of data to and from the server to and from client. Message queues are asynchronous. They store the messages until the messages are not fetched from them by either client/server. Typically, there are 2 message queues acting together.

The most common tool used for message queueing is the **RabbitMQ.** It is very flexible to manage and gives acknowledgements for the messages. It is also asynchronous and hence suitable for our needs.

**Database:**

**MySQL** is an open source database which can cater our needs. It is relational DBMS. We can replace it with non-relational databases like MongoDB as they are good for document storage.

For now, we’re storing test cases and bug reports as Text fields, not documents. Therefore, **MySQL** is suitable for our needs.

1. **Scalability and Load balancing:**

Scalability and load balancing are crucial for any enterprise level SaaS service. If the service can’t be scaled as and when required, it’s useless as it can crumble under the load.

To address this scalability issue, we run multiple instances of the server. These instances can be scaled up or down depending on the number of incoming requests. To further increase the scalability, we can choose different kind of instances other than general purpose T2 instances. This process can be carried out without interrupting the service. So that overhead is minimal.

For load balancing, we have introduced a load balancer which has its input at the server-end of the message queue. Whenever requests pass through the queue, they’re redirected towards that instance of the server which can handle it. This ensures **high-availability** of the services.