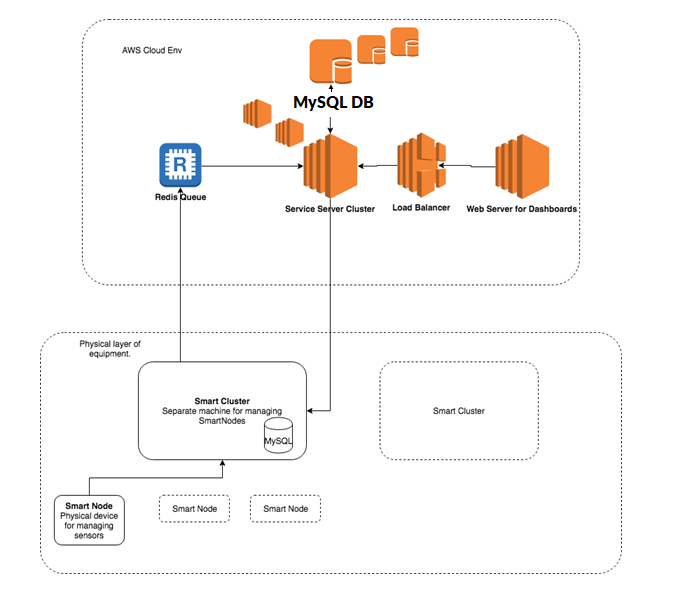
# **3. System infrastructure and architectures**

**3.1 Cloud-based IOT sensor system infrastructure design**

The real Smart Agriculture Network will contain from two main parts:

* Physical layer of equipment
* Cloud based software to collect and manage data and equipment

The diagram below shows those two elements:

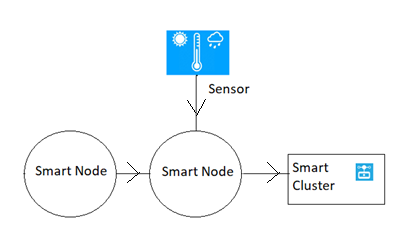


**3.1.1 Physical layer of equipment**

The Physical layer of equipment contain Smart Nodes and Smart Clusters.

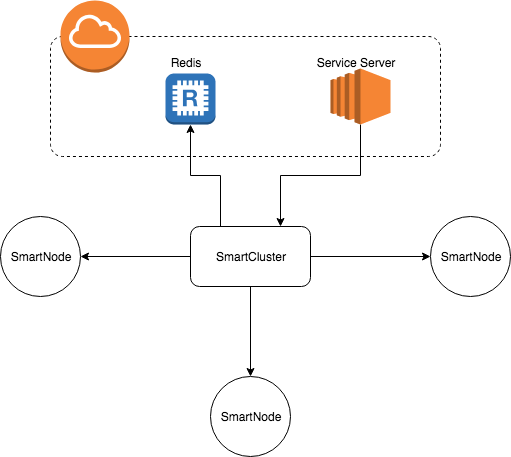
*SmartNode responsibility is*:

* Collect data from sensors
* Collect health data of sensors
* Allow disable/enable, connect/disconnect different (supported) types of sensors
* Pass the information to other nodes



*SmartCluster responsibility is:*

* Register SmartNodes
* Collect data from SmartNodes with the intervals determined by configuration
* Collect health data from SmartNodes
* Push data into Queue (Redis Queue)



***For education purpose the Physical layer of equipments will be forge with cloud based applications which will mock the real devices by implementing appropriate API interfaces.***

***All below architecture and analysis will be based on this statement!***

**3.1.2 Cloud based software**

The Cloud based software will contain from the next main components:

* Service Server (Node JS)
* Queue (Redis)
* DataBase (MySQL)
* Web Server (Node JS)

*Service Server responsibility:*

* Manage and configure SmartClusters
* Pulling and transform Sensors Data and Health data from Queue and store it in DB
* Expose Rest API to access to Sensors Data

In terms of support horizontal scalability all Service Servers will be behind Load Balancer.

*Queue (Redis) responsibility:*

* Accumulate data in the queue
* Give back data from the queue

The main purpose to use Queue is to prevent data loss in case of Service Server maintenance or crashing.

*DataBase (MySQL) responsibility:*

* Store Sensors Data and HealthData
* Provisioning different types of searching requests
* Provisioning good horizontal scalability

In terms of horizontal scalability, which is for my opinion is very important for such type of data (sensors time series data) the most suitable DB is MySQL.

*Web Server (Node JS) responsibility:*

* Provide 3 types of dashboard for users (Farmers, IOT Support, Infrastructure manager)
* Users Authentication and Authorization
* User management functionality for 4th type of user - admin.

**3.2 Deployment-Oriented System Infrastructure**

Our Cloud Infrastructure Provider is AWS. For SmartNodes, SmartClusters, Service Server and Web Server will be used ec2 instances.

As a software deployment mechanism the docker image containing will be used.

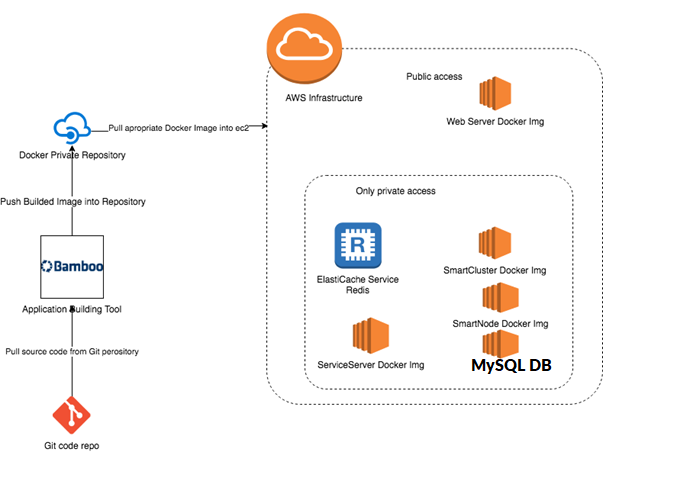
The deployment process:

* Instantiate ec2 instance in AWS - Amazon Linux **AMI 2018.03.0** with Docker
* ssh into instance
* Run Docker image with appropriate settings

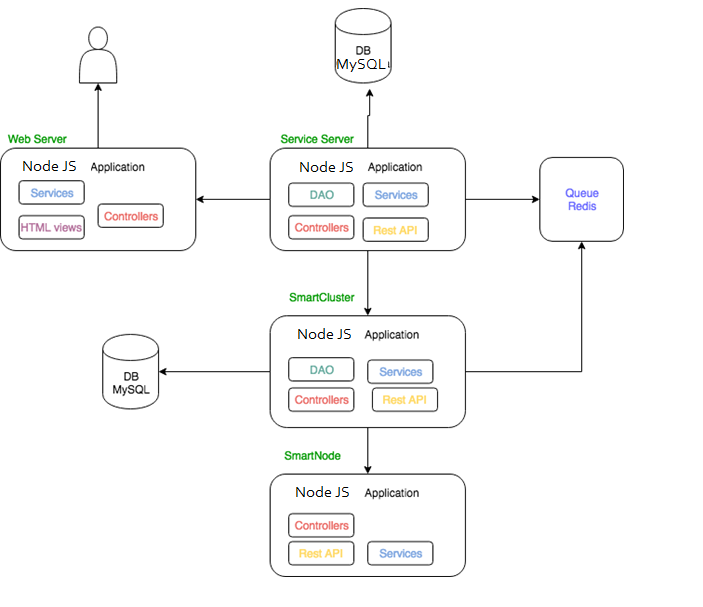
Security groups:

* Private access security group (SmartNode, SmartCluster, Queue, Service Server)
* Public access security group (Web Server)

In real life the Queue should have public access and SmartCluster should use authentication credentials to be able to push data into. In our education project we don’t need it since SmartNodes and SmartClusters will be run in Private Security Group.



**3.3 System-Oriented Component Functional View**



**3.4 Database design**

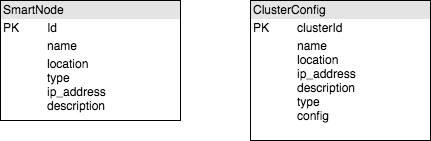
In our project will be used two types of DB: MySQL for Smart Cluster and MySQL for ServiceServer.

**3.4.1 MySQL for Smart Cluster**

This DB dedicates to store the SmartCluster related data like:

* Smart Nodes data (id, name, location, description, type, ip\_address)
* Smart Cluster data (clusterId, name, location, description, config, ip\_address, type)

Config data will contain: Smart Nodes status ping intervals, sensors data collecting intervals.

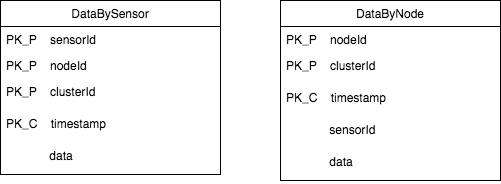


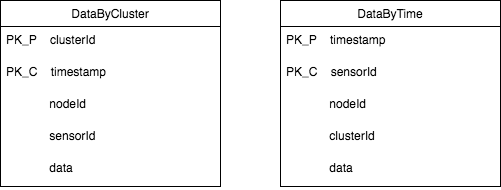
**3.4.2 MySQL for ServiceServer**

This DB dedicates to store:

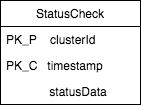
* sensors time series data
* Smart Nodes statuses data

Sensors time series data.





Statuses Data



PK\_P - Primary Key Partitioned

PK\_C - Primary Key Clustered

**3.4.3 Redis Queue**

In purpose of buffering the data we are going to use Redis Queue. There are 2 types of messages:

* sensors data
* Smart Nodes statuses data

As data structure the JSON will be used:

*{“messageType”: MessageType, “content”: {contentJsonObject}}*

*enum MessageType:{*

*STATUS, SENSOR\_DATA*

*}*

**6. Cloud Technology and Validation**

**6.1 Cloud Technology**

The cloud technology that we would use for this project is Amazon Web Services for hosting the application. AWS is an Infrastructure As A Service and we will use it too instantiate multiple EC2 instances for servers and clients to support a Distributed Architecture.

The application is a full stack software designed using the Javascript framework. The backend will be in node js and the client side is designed in react js. Some components of the frontend will support react-redux with Apache Kafka as the messaging queue. The major database in the application is MySql but some of the data will be stored in MongoDB to increase performance and to show the scalablitiy of the application.

Amazon MSK ie Managed Streaming for Kafka may also be used depending on the schedule of the application for supporting Kafka Messaging. The reasons for selecting AWS as our application hosting technology are as follows:

* Pay-as-you-go: The best feature of using AWS that the billing is completely based on the usage of the resources. AWS provides several kinds and variety of resources to choose from depending on the application that we intend to deploy on it. Therefore, it removes the dependency of an application on the local machine as there might be some resources that are overutilized while some might be underutilized. AWS provides us with the exact amount of resources that we need for the application.
* Scalability: AWS helps in making our application scalable by providing services like load balancing and database sharding. It also supports auto-scaling which reduces the developer’s work by a considerable amount and because of it’s elasticity in such domains, AWS becomes the best choice.
* Security and Network: AWS has recognized certifications for security like HIPPA and SAS70 which is a huge assurance as code plagiarism is very common. The applications have a variety of options to configure the application like bandwidth and protocols.
* Flexibility: The very option of picking our own configuration for deployment and designing any application makes AWS one of the most flexible platforms available. Right from storage, memory to network, developers are free to configure their own platforms for their applications. It is under our control.
* Reliability: AWS is highly reliable as the instances are rapidly commissioned. It makes the application safe as there is very less or no latency or loss in efficiency.

**6.2 Validation Environment**

Validation is one of the most important and under-rated phases of the software development cycle. It’s however very important to make sure that the final application matches the requirement stated for the application. It’s also important to measure the performance of the application after deployment on AWS.

Below are the mentioned validation environments we will be following:

* **JMeter:** Apache JMeter is a load testing tool especially for analyzing and measuring the performance of different services provided by any web application. It is an open source Java application and covers several categories of tests like load testing, functional requirements, regression testing and performance testing. We will be validating the software with and without deployment on the cloud and with and without redis implementation with the SQL tables.
* **Manual Testing:** Manual testing will be performed by individual team members with their respective components as they implement new functionalities. This would include different approaches of white box testing.