# Koala: A simplified framework for implementing Fog, Edge and Cloud computing applications/research

Shreshth Tuli, Shikhar Tuli, Redowan Mahmud, Rajkumar Buyya

## Introduction

## Model

Koala is a lightweight and platform independent framework which allows an end-to-end implementation of a fog, edge and cloud computing environment. It is designed to run on almost all IoT/embedded devices and support a wide range of machines in terms of their computation capabilities. A driving factor for making this framework independent of operating system and machine architecture is to allow the diversity of embedded devices in the IoT world. The framework integrates all edge devices (using a lightweight balancing framework for small-scale computation) and also cloud/other heavy devices (using the Aneka framework for heavy computation) and provides an end to end solution for integrating computation at different levels to provide better latency (by allocating simple and small-scale work to edge nodes) and high computation power (by allocating heavy computation work to cloud), which in effect to the end user feels like high computation with low latency. This is the fundamental ideology on which fog computing is based.

All devices interact at different levels with different devices. The overall model is based on Master-Slave design where the Master node receives data from sensor for analysis/computation. The Master node then sends this data as a task to different workers: Local worker, Private Cloud or Public Cloud. Therefore, the core components of network include Master Node, Worker Nodes, Sensors and Cloud. All other components including gateways, switches and other interface devices ‘glue’ the core components together.

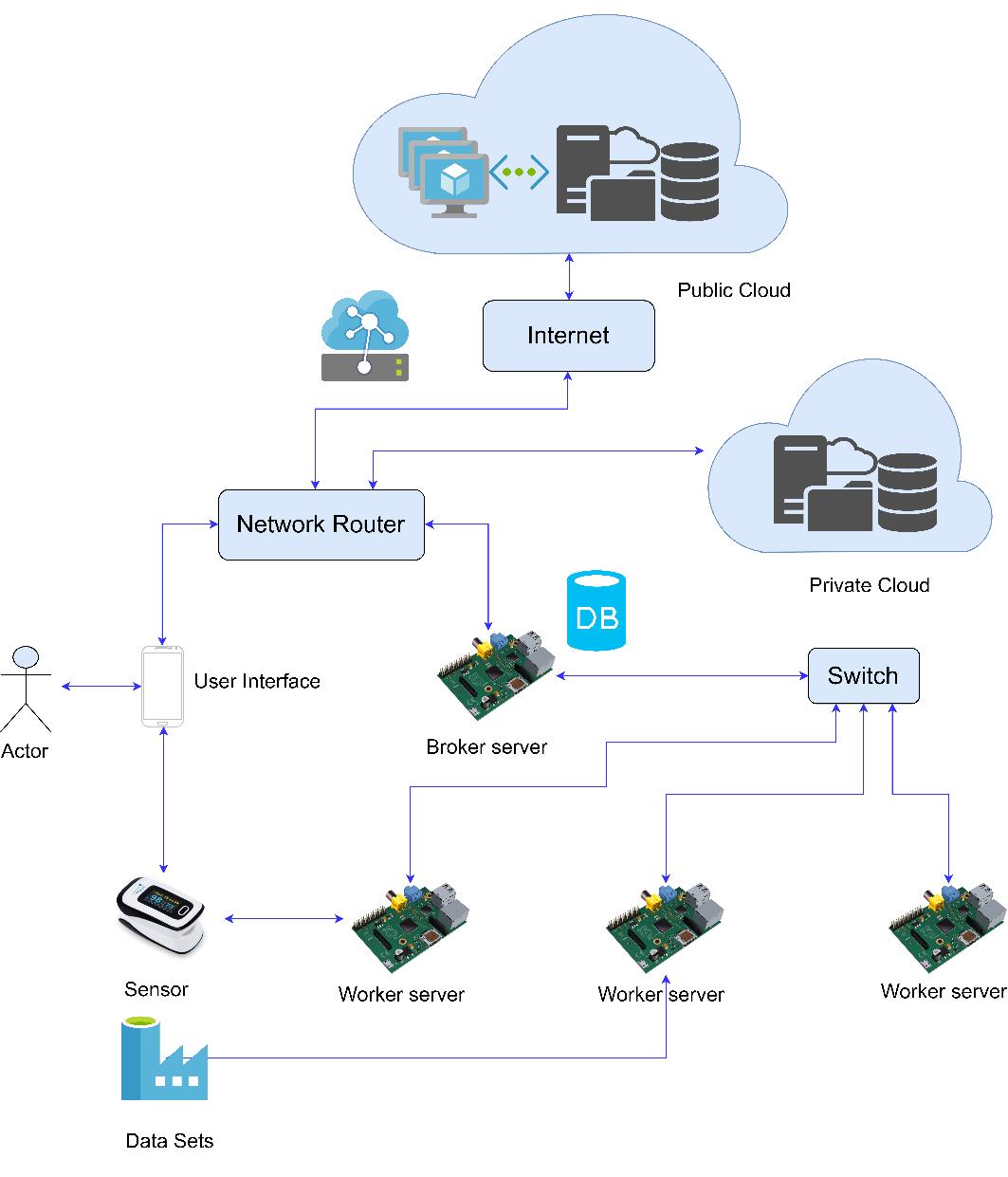
The model is developed to provide a framework whose aims include:

1. Allow input of data from sensors and output of information/actions to end user/actuators
2. Provide an end-to-end and seamless communication between different fog and cloud resources
3. Provide a platform to easily form an integrated system that can use this interaction to provide services to the end user
4. Implement resource management policies and allow easy extension/implementation of such policies for efficient load distribution
5. Set up a friendly interface between the framework and end user

### Network Level Model

As described before, the core components of the network model are shown in Figure 1. The User connects the local network and interacts with the Master Node using an interface which can be an Android or other smart device. This interface allows communication between sensor/actuator with the Master Node.

The Master Node also referred to as the Broker Node collects data from the sensor uses it itself for analysis or sends it to one of the followings:

1. Worker Node in the local network
2. Private Cloud
3. Public Cloud or datacenter

The Master Node also collects results/actions from the devices mentioned above and send it to user interface/actuator. The master or worker nodes can also receive data directly from data sets and perform actions accordingly.

It is in the hands of the user, which service to run and which master to connect to. There can be Master nodes dedicated for specific tasks and in IoT environment multiple services can even run simultaneously requiring connections with multiple masters and/or multiple service running on same master node.

Figure 1: Network Level Model

### Interaction among devices

### Functional Components

1. **Resource Management:**

As devices at different levels of the network, as shown in the ‘Network Level Model’, offer range of computational capabilities and latencies, it is important to distribute work accordingly to ensure optimum QoS. The devices at the ‘edge’ of the network provide very low latency due to close proximity with the user, sensors and actuators, but have significantly low computation power and thus higher execution times. The devices far away from users like those on the cloud provide significantly higher latency but due to their much higher computation capabilities provide much lesser execution time. There is thus a trade-off between network propagation latency and computation latency. Depending on different scenarios, different resource management techniques are used.

There is scope of considerable reduction of latency caused due to propagation of data if low-computation workload is allocated to edge devices. Even though their execution time would be higher but the critical factor here would be the time caused by data transfer over the network. On the other hand, significant reduction in time can be achieved if high-computation workload is sent to more powerful machines like cloud because in such tasks edge devices might lead to deadlocks and cause the whole system to collapse.

The Resource manager of the model should decide the following:

1. Allocate task to a local worker node or not: As the latency offered by local machines is much lower in terms of the network propagation of data this is a crucial decision to be taken by the master node. Sending to cloud would require considerable overhead and might not be suitable for real-time and mission critical tasks like healthcare, robotics, etc.
2. Allocate task to itself or not: Some small tasks can be performed by the master itself. As the master deals with the load balancing and other tasks like maintaining integrity of data, it might be that the master node is itself busy. Even then some small tasks may arise on rare occasions which require immediate results. At such stages, the master may decide to perform it itself to eliminate data propagation over the network and other authentication and handshaking protocols.

It is a design decision in the implementation strategy how these decisions are taken. Some parameters that are important for taking such decisions include:

1. **Quality of results**: Each application has its own quality thresholds. As the required quality of results increases, more complex algorithms may be required. Sometimes the algorithm remains same but more number of iterations are required to reach the required confidence threshold. This has a direct impact on the computation difficulty and how well and fast a machine is able to perform repetitive tasks.
2. **Deadline**: The time in which the results are required plays a crucial role in determining the load balancing scheme. As hinted earlier, devices at different levels have different execution times and data propagation times. Based on data flow rate and application runtime it is important to balance tasks among different levels of devices.
3. **Frequency of data sharing**: Closely related to point 2, some tasks might require small amount of data to be analyzed but with high frequency for example -----. In such cases execution time needs to be very low and network bandwidth very high. The device themselves should allow high I/O data flow.
4. **Computation Complexity of Application**: The order of complexity of application determines the number of task request to be sent to cloud. For very high complexity tasks it might me suitable to only use cloud resources to prevent bottlenecking of the software because of low computational capabilities of edge nodes.
5. **Cost of Computation**: Different levels have different currency for computation. Devices over the cloud are metered and are charged based on time of usage. Embedded devices in IoT applications are charged based on the initial physical component cost and then the cost of energy expenditure for functioning of these devices. By normalizing these different currencies using conversion factors, one can determine the cumulative cost for running the framework per unit time/task/user. Inherent to the model requirements, but depending on the scenario, this cost needs to be minimized.

Different load balancing algorithms can be implemented that specialize in improving some of these parameters based on user needs. A weighted sum of quantifiable factor of each of these parameters and the weights themselves would determine which balancing technique should be used for a specific situation.

## Further Scope of developments

This software forms a base for setting up Fog Computing Environment which is not OS or architecture specific. Though the framework is complete by itself and fully functions to perform real-time, sensor-based Sleep Apnea analysis, but has a large scope of improvement and further developments:

Load Balancing Scheme

As discussed earlier, the current load balancing scheme is naive and can be greatly improved. Currently, the load balancing scheme focuses on task distribution based on CPU load, whichever device has the least load, the task is given to it. There can be a cumulative ranking parameter which takes into account many other factors as per requirement. These factors can be and are not limited to: Network Bandwidth, Memory Load, etc. Different weights can be given to these parameters are per the scenario.

### Data Integrity

Health Analysis data is important for patients as their treatment and lives depend on it, thus it is important to save this data from fraudulent manipulation or sabotage from hackers. Thus, to maintain data integrity many techniques like Blockchain can be implemented to ensure that data is secure.

### Data Privacy

Some applications require data to be secure as well as kept private. The current system is prone to attacks and unwanted display of data to others. Privacy policies like encrypting the data can be used to ensure that the data cannot be seen by others.

### Data Authenticity

The current system allows any device to connect to master and share or view data. This can be used by hackers to forge DDoS or similar attacks. As Fog platforms also contains low range devices with limited threshold management, such attacks even at a low scale can destroy such devices. Thus, a signature-based validation technique can be used to ensure user and data authenticity.