# Decentralized IoT Technical Design Document

## 1. Tech Stack

Requirement for : Building a API framework for communication between two IoT devices.

1. Language to build the APIs that enable communication between the IoT devices: Python
2. Web Framework: Flask (v1.0.2)
3. Source Code Control: Git (Github)
4. IDE: PyCharm (v2018.3.1)
5. Hardware requirements: Minimum 512 GB RAM, 1GHz single-core CPU (Based on Raspberry Pi Zero)

## 2. Accounts and Infrastructure

### 2.1 Development

**Dev Environment**
Operating System: Raspbian (Debian based operating system)
Test IoT device : Raspberry Pi Zero Ws for simulation of the scenarios along with tactile push buttons and LEDs.

### 2.2 Production

**Production Environment**
Operating System: Raspbian (Debian based operating system)
IoT device: Raspberry Pi Zero W

## Data Sources, Models, Timing

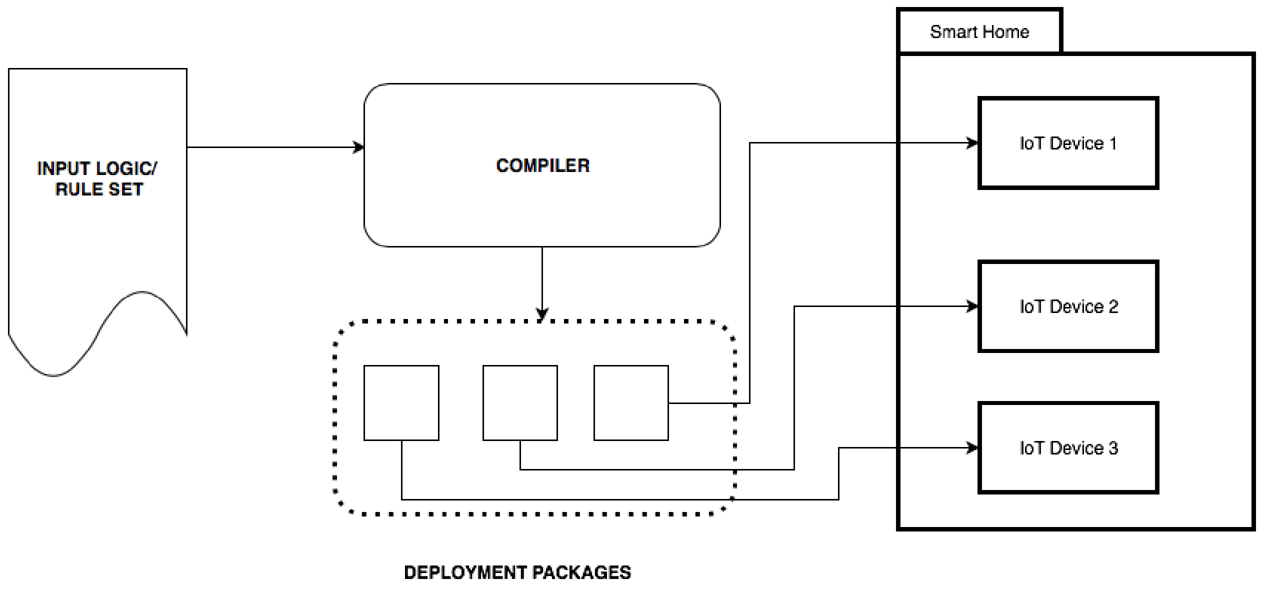
### 1.1 Data Sources

- Data sources will the IoT devices which keep sending the status updates to manager every time state change occurs.
- On every keepalive from the device, the device piggybacks the metadata to the manager

## 1.2 Timing

- For every predefined time interval (defined in the initial YAML configuration file), the device sends a keep alive message to the manager. With the keep alive, the metadata of the device is also piggy backed.
- For the current implementation, the manager does not report the state of all the devices to the cloud.

## System Architecture Diagram



## Deployment Methodology

1. Compiler which is a python script parses the IoT scenario YAML configuration and creates as many python files as many devices in the cluster. The compiler also creates a python file should be running on the manager. The name of the compiler script is ***Compiler.py*** which parses the config file named ***Config.yaml***
2. All the above files are bundled in a folder for each device and contains the parameters for each file. For example for parking lot, it would have ***Entry.txt,Exit.txt,Lot.txt***
3. While deploying the framework, the folder will have to be uploaded to the devices and running the properties in the corresponding *.py file will start the Flask servers that will help communicate with other devices.
4. In case of the simulation of one of the scenarios, along with the Flask applications, a script that monitors GPIO will also run. The GPIO process is necessary to record input and send output for the simulation.

Note: The project does not focus on the methodology of how to deliver the folders to the devices. That is a solved problem in the domain of continuous integration/continuous deployment.