INTERNET OF THINGS
(IoT)

Unit:1 **INTRODUCTION** **15 hours**

Introduction - Definition & characteristics of IoT - physical design of IoT - logical design of IoT - IoT enabling Technologies - IoT levels & Deployment templates. Domain specific Iots : Home Automation - cities - Environment - Energy - retail - logistics - Agriculture - Industry i Health and life style.

Unit:2 **IOT and M2M** **12 hours**

IoT and M2M - Deference between Iot and M2M - SDN and NFV for lot - IoT systems management - SNMP - YANG - NETOPEER

Unit:3 **IOT SPECIFICATION** **15 hours**

IoT platforms design Methodology - purpose and specification - process specification - Domain model specification - Information model specification - Service specification - IoT level specification - functional view specification - operational view specification - Device and component Integrators - Application Development.

Unit:4 **LOGICAL DESIGN USING PHYTHON** **15 hours**

Logical design using python - Installing python - type conversions - control flow - functions - modules - File handling - classes. IoT physical devices and End points, building blocks of IoT device - Raspberry Pi - Linux on Raspberry Pi - Raspberry Pi interfaces.

Unit:5 **IOT AND CLOUD COMPUTING** **15 hours**

IoT physical servers & cloud computing - WAMP - Xively cloud for IoT - python Web application frame work - Amazon web services for IoT.

Text Book(s)

Internet of Things - A hands on Approach Authors: Arshdeep Bahga, Vijay Madisetti Publisher: Universities press.

Reference Book

Internet of Things - Srinivasa K.G., Siddesh G.M. Hanumantha Raju R. Publisher: Cengage Learning India pvt. Ltd (2018).

UNIT I

- IoT definition
- Characteristics of IoT
- Physical Design of IoT
- Logical Design of IoT
- IoT Protocols
- IoT Levels & Deployment Templates

Definition of IoT

A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users and their env**ironments.**

Characteristics of IoT

- Dynamic & Self-Adapting
- Self-Configuring
- Interoperable Communication Protocols
- Unique Identity
- Integrated into Information Networks

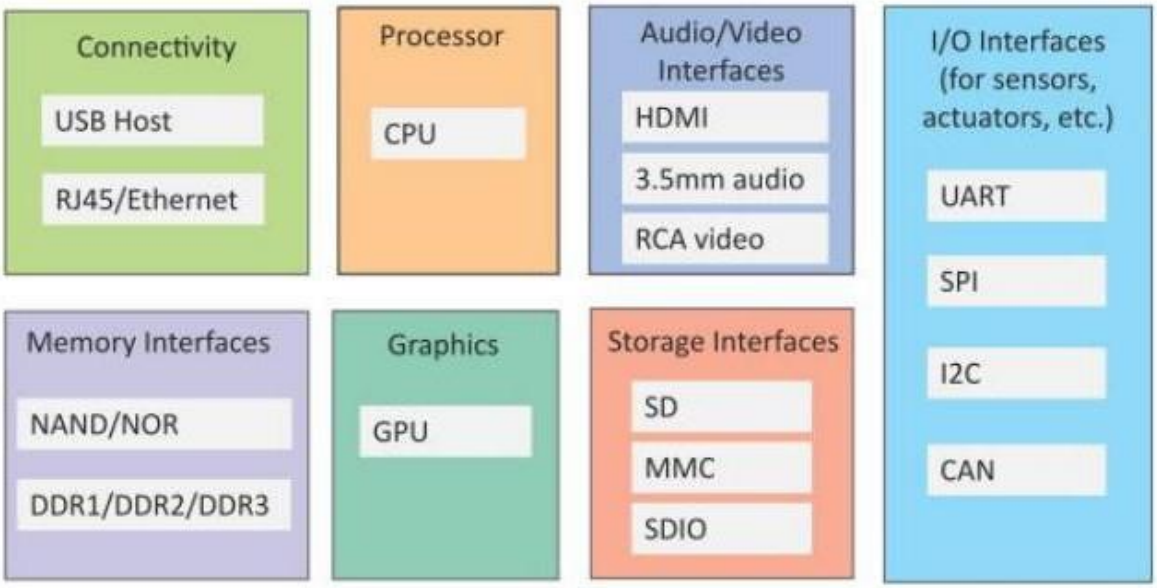
Physical Design of IoT

- The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.
- IoT devices can:
- Exchange data with other connected devices and applications (directly or indirectly)
- Collect data from other devices and process the data locally
- Send the data to centralized servers or cloud-based application back-ends for processing the data
- Perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraint**s.**

Generic block diagram of an IoT Device

An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.

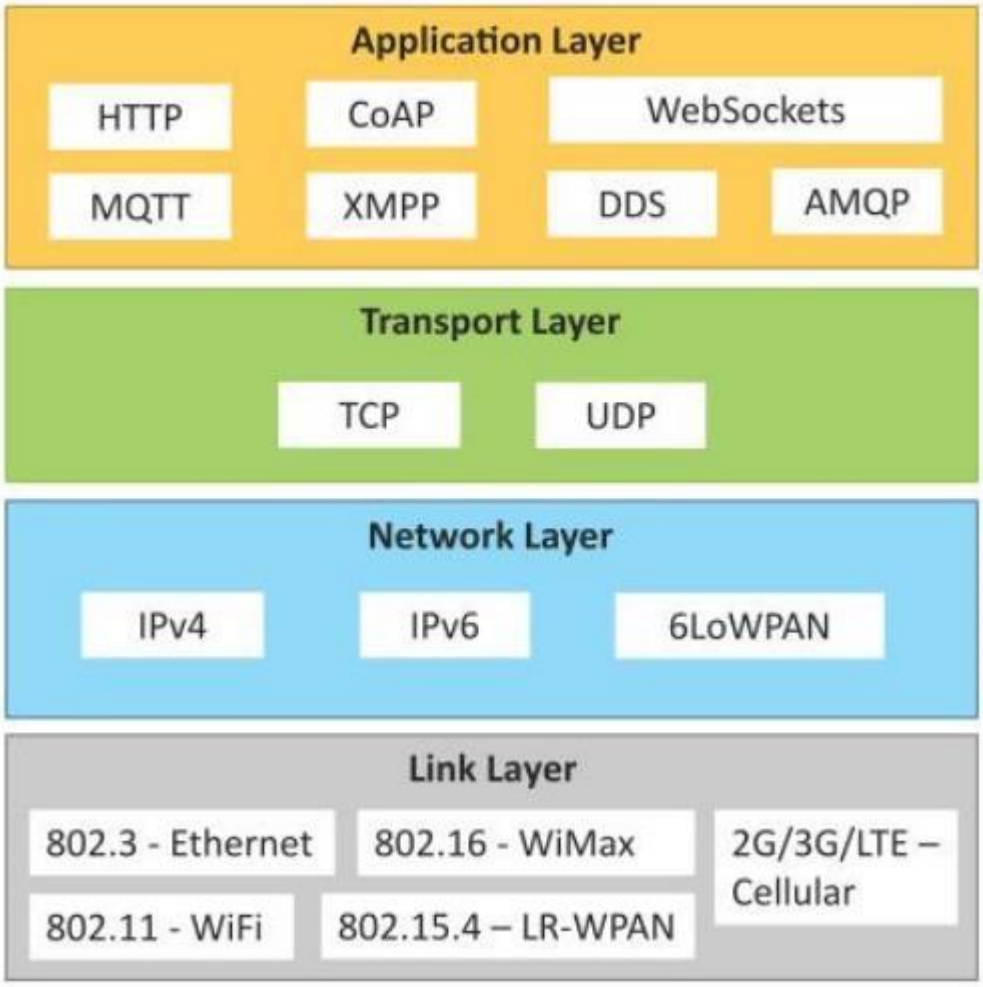
- I/O interfaces for sensors
- Interfaces for Internet connectivity
- Memory and storage interfaces
- Audio/video interface



IoT Protocols

Link Layer

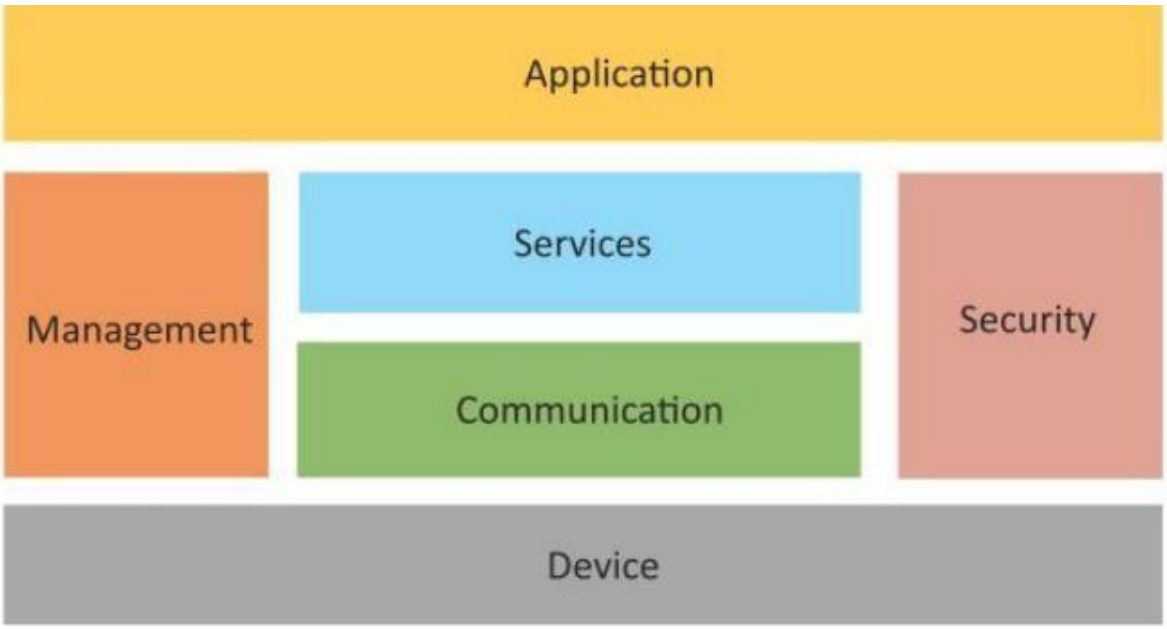
- 802.3 – Ethernet
- 802.11 – WiFi
- 802.16 – WiMax
- 802.15.4 – LR-WPAN
- 2G/3G/4G
- Network/Internet Layer
- IPv4
- IPv6
- 6LoWPAN
- Transport Layer
- TCP • UDP
- Application Layer
- HTTP
- CoAP
- WebSocket
- MQTT
- XMPP
- DDS
- AMQ



Logical Design of IoT

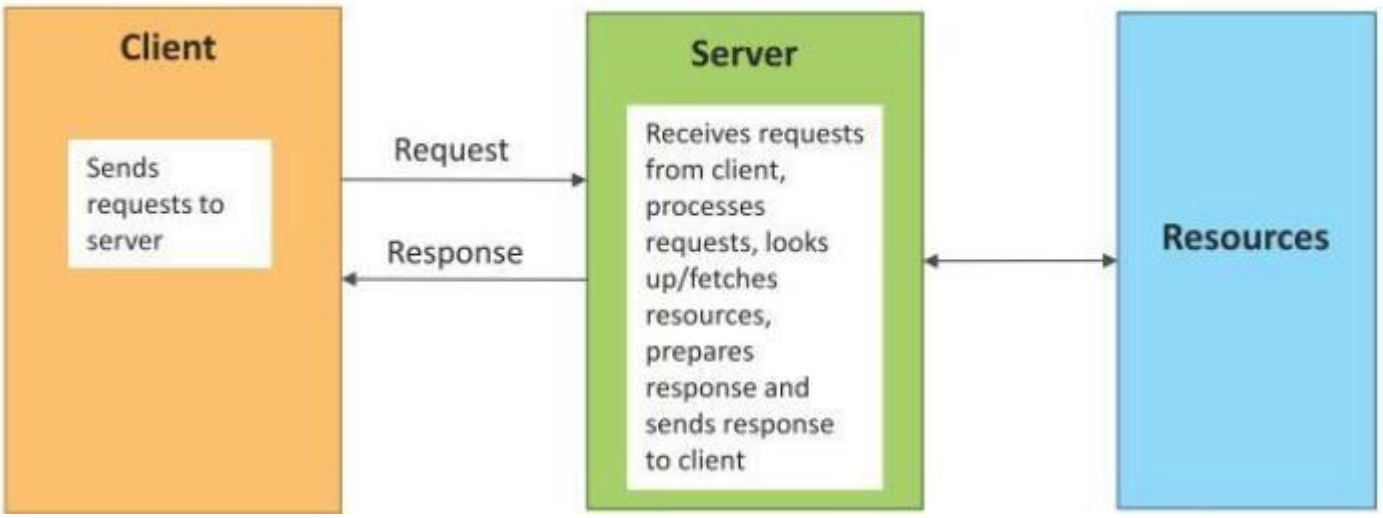
Logical design of an IoT system refers to an abstract representation of the entities and processes without going into the low-level specifics of the implementation.

An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication, and management.



Request-Response communication model

• Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests. • When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response, and then sends the response to the client.



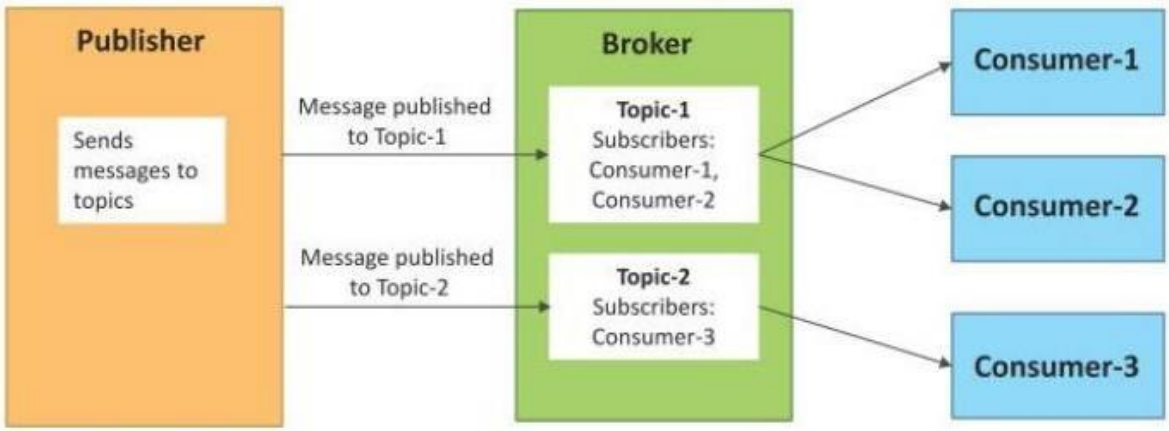
Publish-Subscribe communication Model

Publish-Subscribe is a communication model that involves publishers, brokers and consumers.

• Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers.

• Consumers subscribe to the topics which are managed by the broker.

• When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

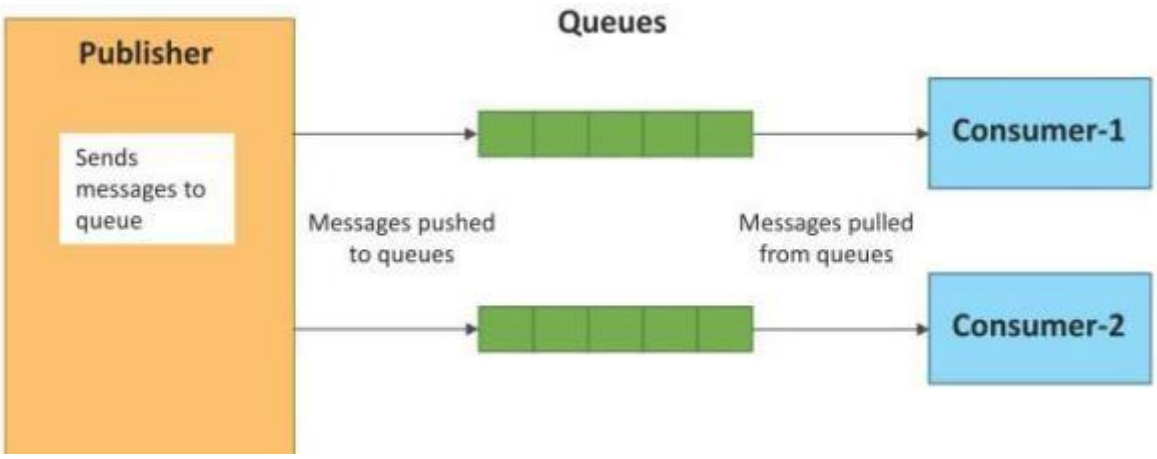


Push-Pull communication model

Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.

• Queues help in decoupling the messaging between the producers and consumers.

• Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate rate at which the consumers pull data.

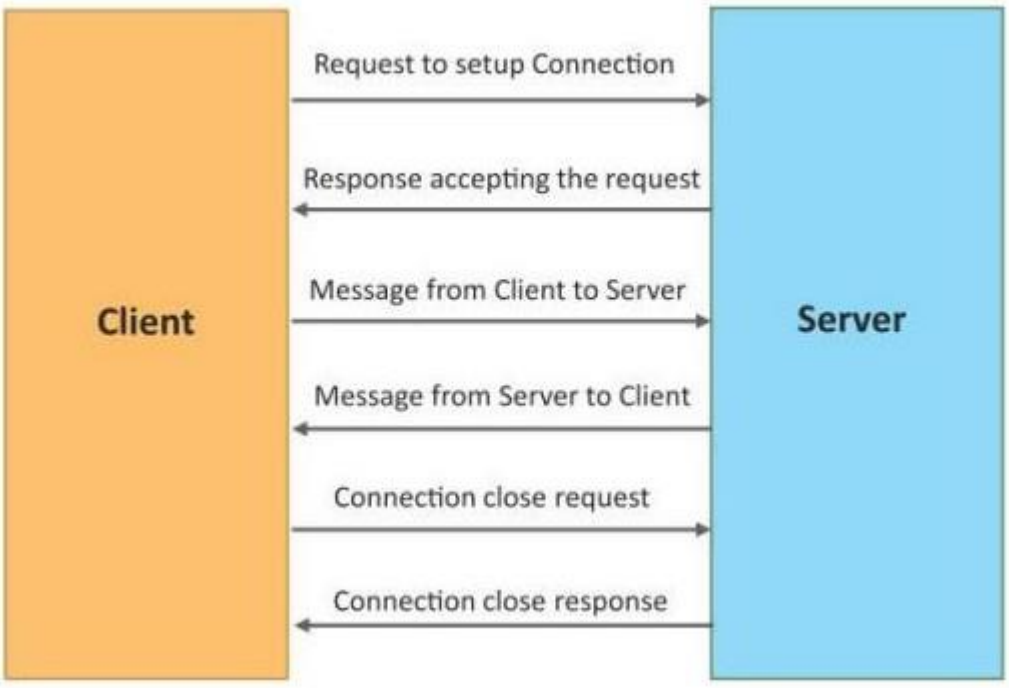


Exclusive Pair communication model

• Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.

• Once the connection is setup it remains open until the client sends a request to close the connection.

• Client and server can send messages to each other after connection setup.

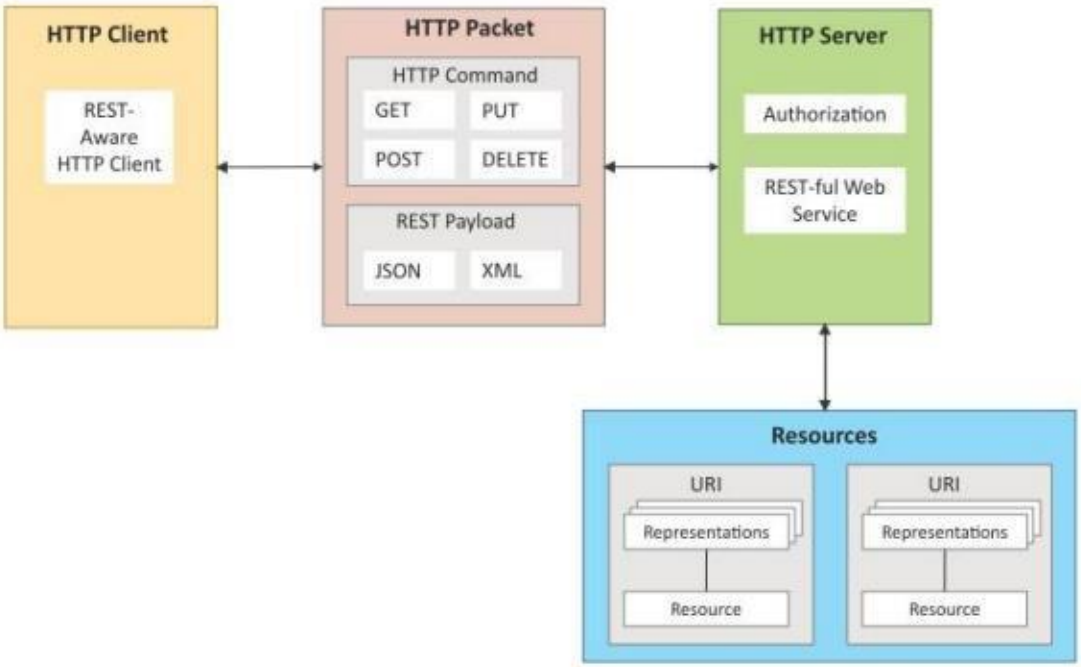


REST-based Communication APIs

Representational State Transfer (REST) is a set of architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred.

• REST APIs follow the request- response communication model.

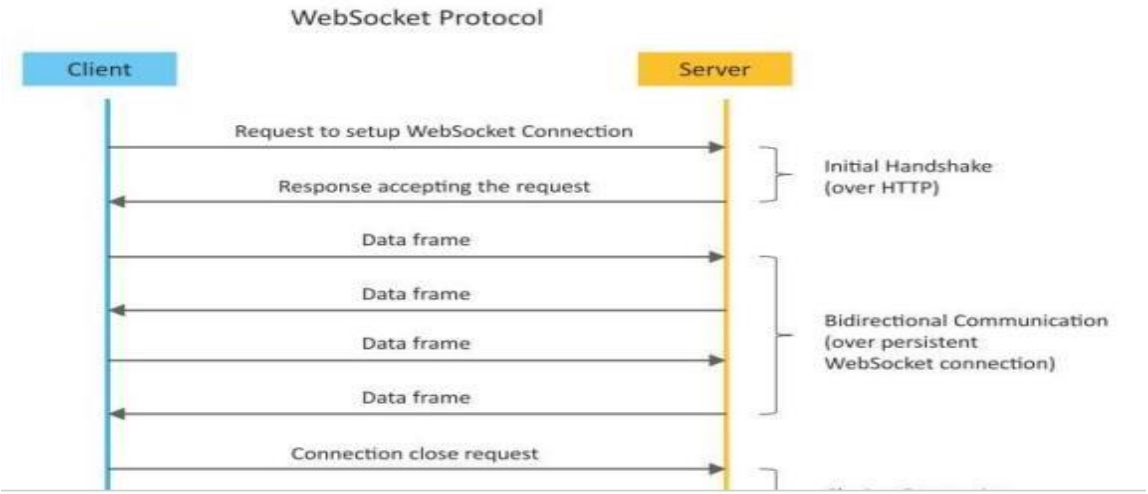
• The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia system.



WebSocket-based Communication APIs

- WebSocket APIs allow bi- directional, full duplex communication between clients and servers.

• WebSocket APIs follow the exclusive pair communication model.

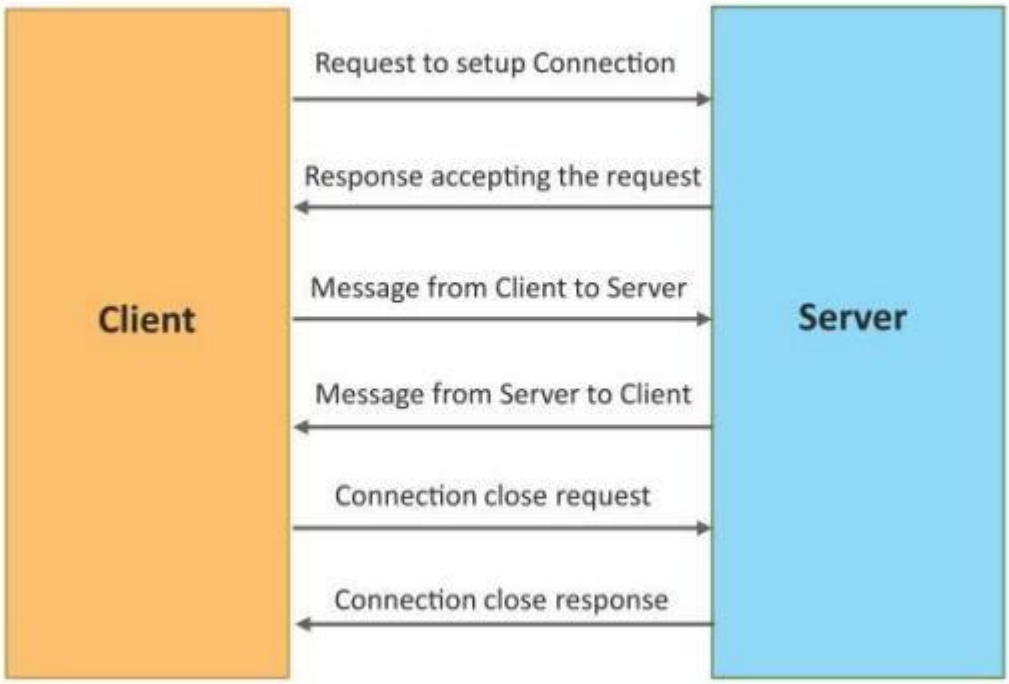


Exclusive Pair communication model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.

• Once the connection is setup it remains open until the client sends a request to close the connection.

• Client and server can send messages to each other after connection setup.



IoT Levels & Deployment Templates

An IoT system comprises of the following components:

• **Device:** An IoT device allows identification, remote sensing, actuating and remote monitoring capabilities. You learned about various examples of IoT devices in section.

• **Resource:** Resources are software components on the IoT device for accessing, processing, and storing sensor information, or controlling actuators connected to the device. Resources also include the software components that enable network access for the device.

• **Controller Service:** Controller service is a native service that runs on the device and interacts with the web services. Controller service sends data from the device to the web service and receives commands from the application (via web services) for controlling the device.

IoT Levels & Deployment Templates

• **Database:** Database can be either local or in the cloud and stores the data generated by the IoT device.

• **Web Service:** Web services serve as a link between the IoT device, application, database and analysis components. Web service can be either implemented using HTTP and REST principles (REST service) or using WebSocket protocol (WebSocket service).

• **Analysis Component:** The Analysis Component is responsible for analyzing the IoT data and generate results in a form which are easy for the user to understand.

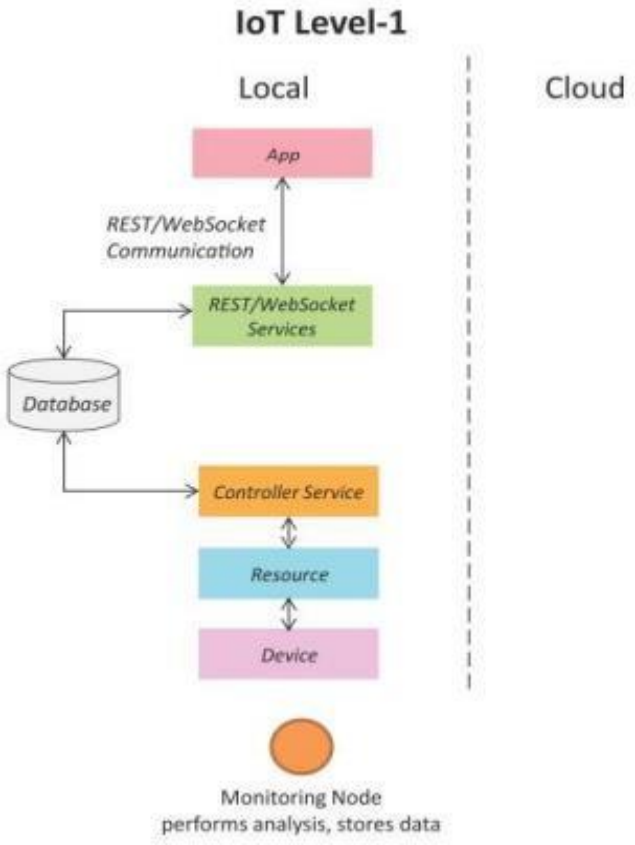
• **Application**: IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and view the processed data.

IoT LevelS

A level-1

• IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application.

• Level-1 IoT systems are suitable for modeling low- cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.

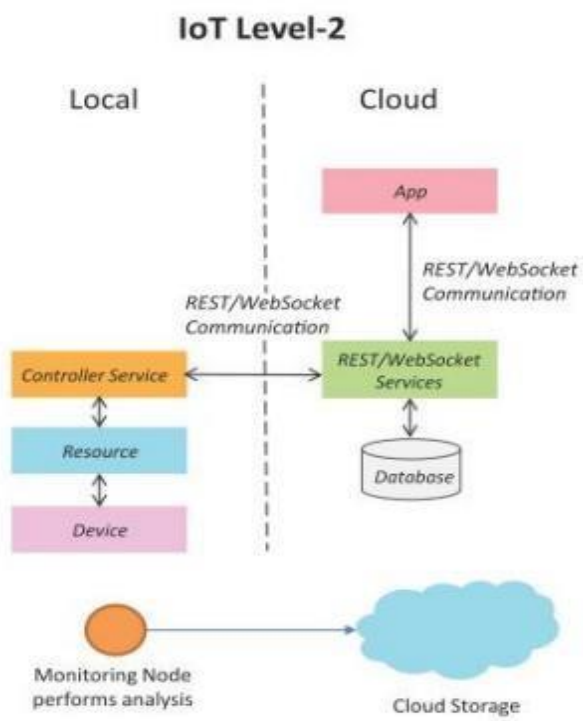


IoT Level-2

• A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis.

• Data is stored in the cloud and application is usually cloud- based.

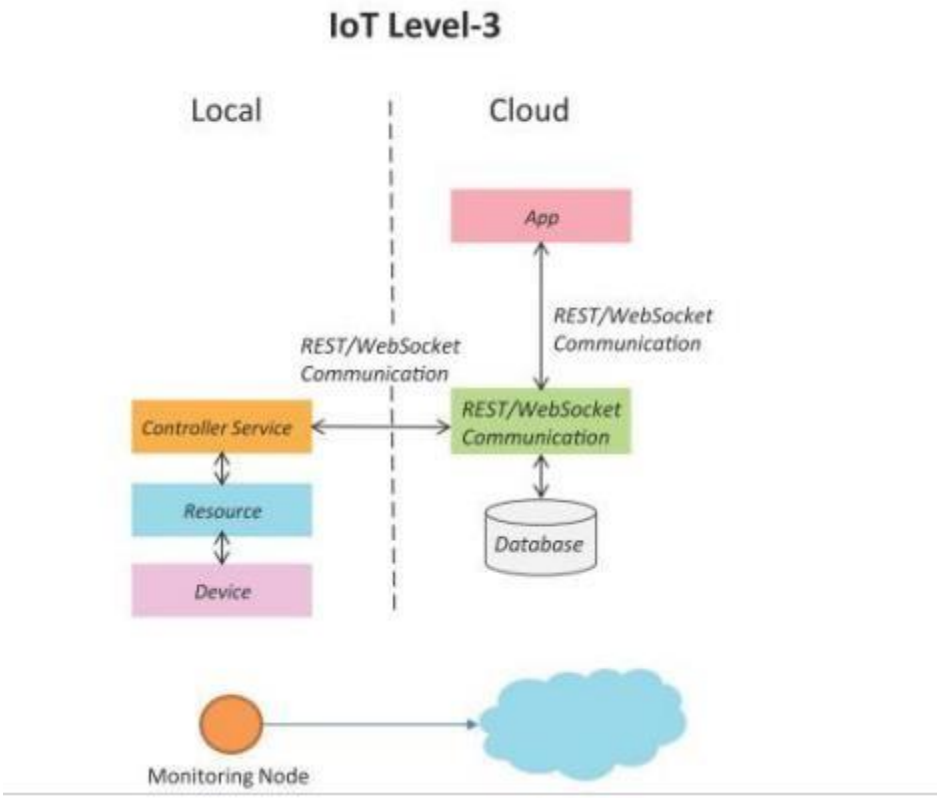
• Level-2 IoT systems are suitable for solutions where the data involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself.



IoT Level-3

• A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and application is cloud- based.

• Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.

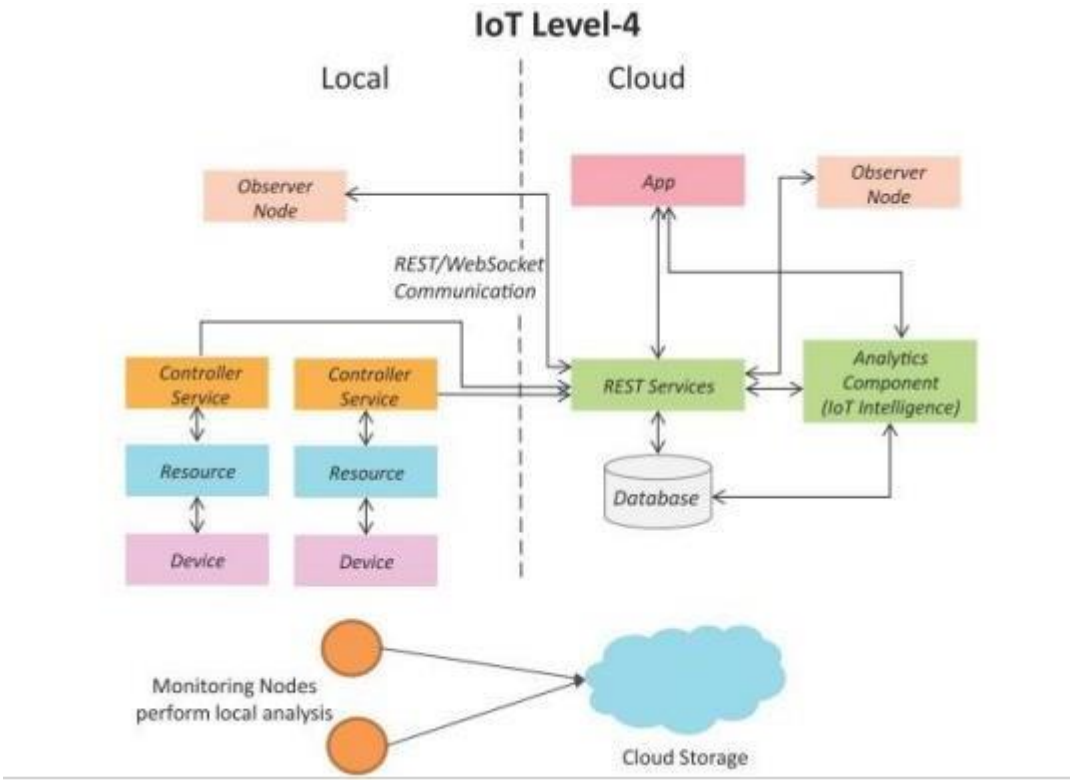


IoT Level-4

• A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based.

• Level-4 contains local and cloud- based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.

• Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.



IoT Level-5

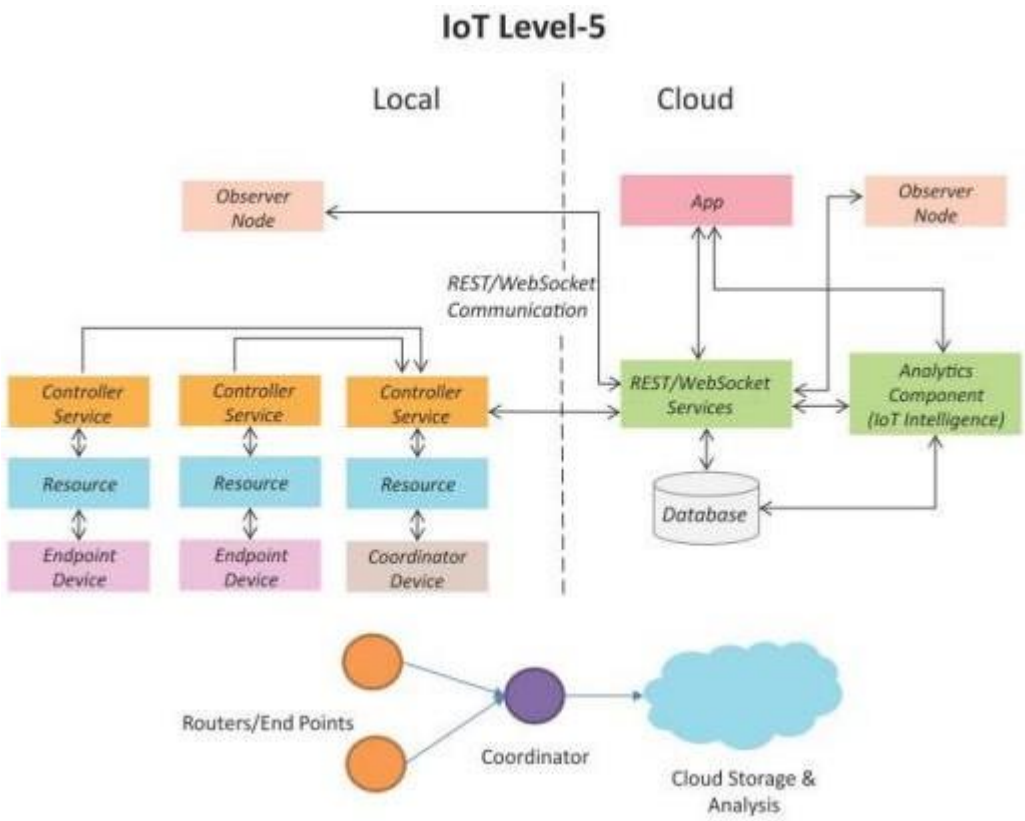
• A level-5 IoT system has multiple end nodes and one coordinator node.

• The end nodes that perform sensing and/or actuation.

• Coordinator node collects data from the end nodes and sends to the cloud.

• Data is stored and analyzed in the cloud and application is cloud-based.

• Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.



IoT Level-6

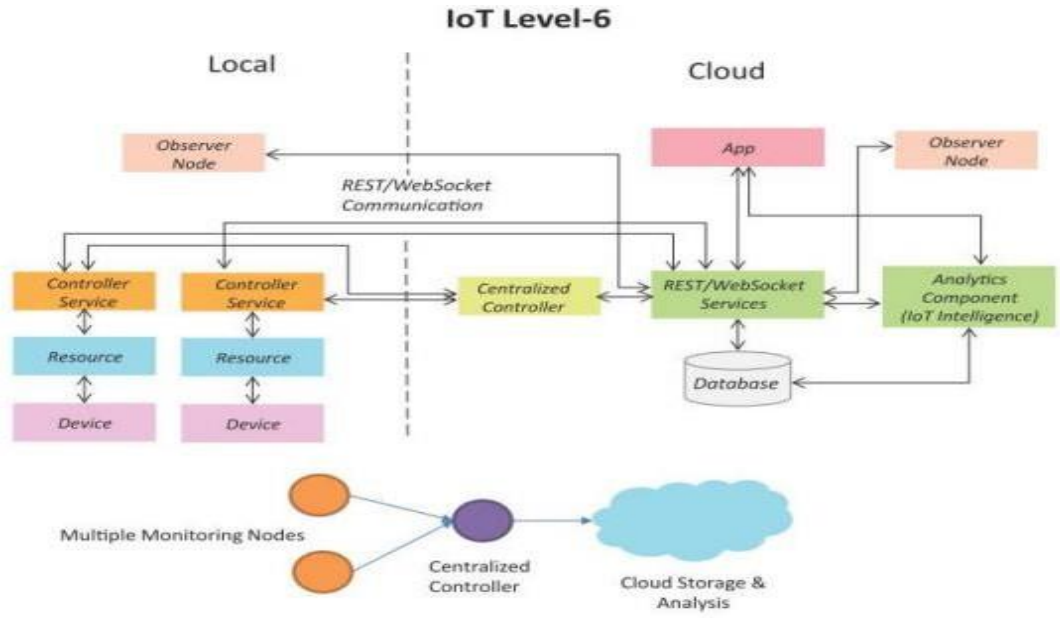
• A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.

• Data is stored in the cloud and application is cloud-based.

• The analytics component analyzes the data and stores the results in the cloud database.

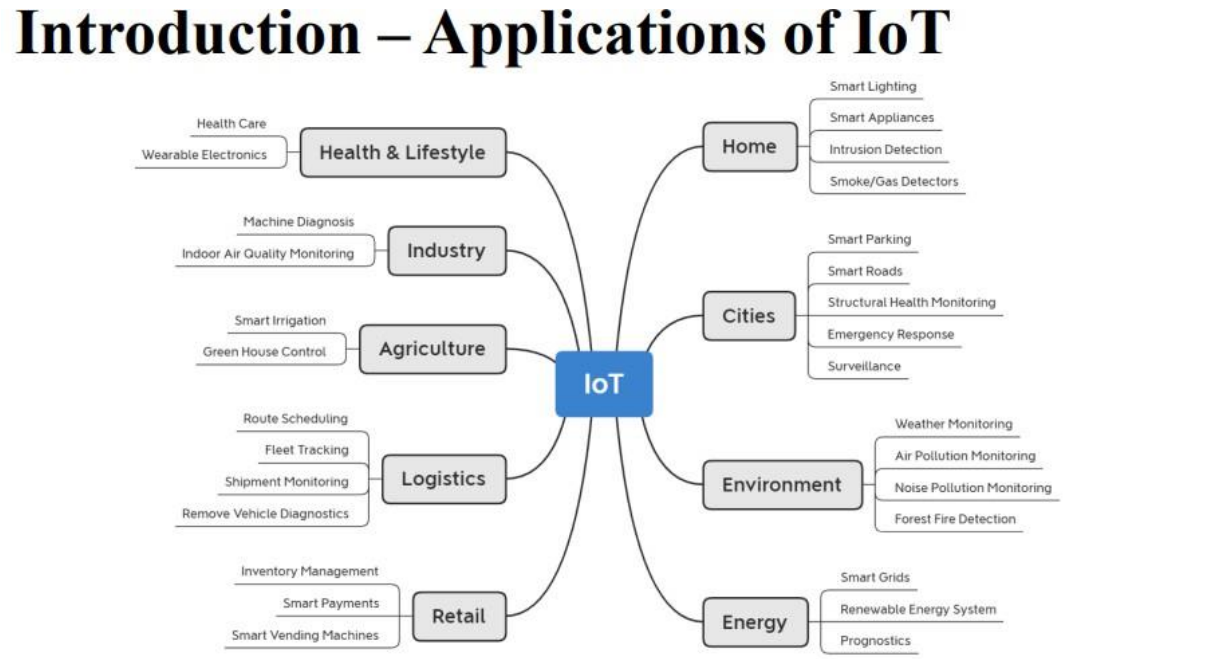
• The results are visualized with the cloud-based application.

• The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.



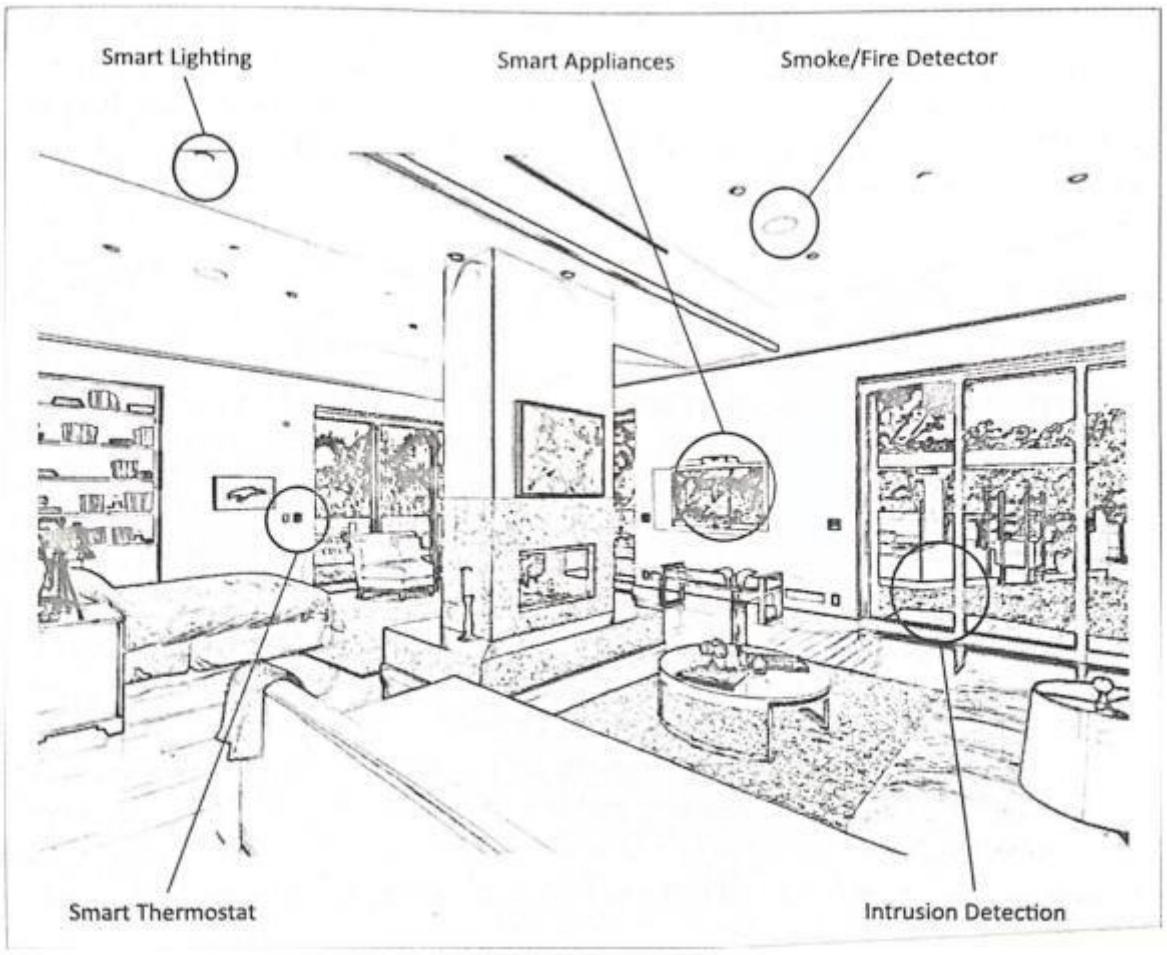
Domain Specific IoT

- Home Automation
- Cities
- Environment
- Energy
- Retail
- Logistics
- Agriculture
- Industry
- Health & Lifestyle



Home Automation

- **Smart Lighting**
- Control lighting by remotely (mobile or web applications)
- **Smart Appliances**
- Provide status information to the users remotely
- **Intrusion Detection**
- Use security cameras and sensors (PIR sensors and door sensors)
- Detect intrusions and raise alerts
- The alerts form: an SMS or an email sent to the user
- **Smoke/Gas Detectors**
- Use optical detection, ionization, or air sampling techniques to detect the smoke
- Gas detectors can detect harmful gases
- Carbon monoxide (CO)
- Liquid petroleum gas (LPG)
- Raise alerts to the user or local fire safety department



Cities

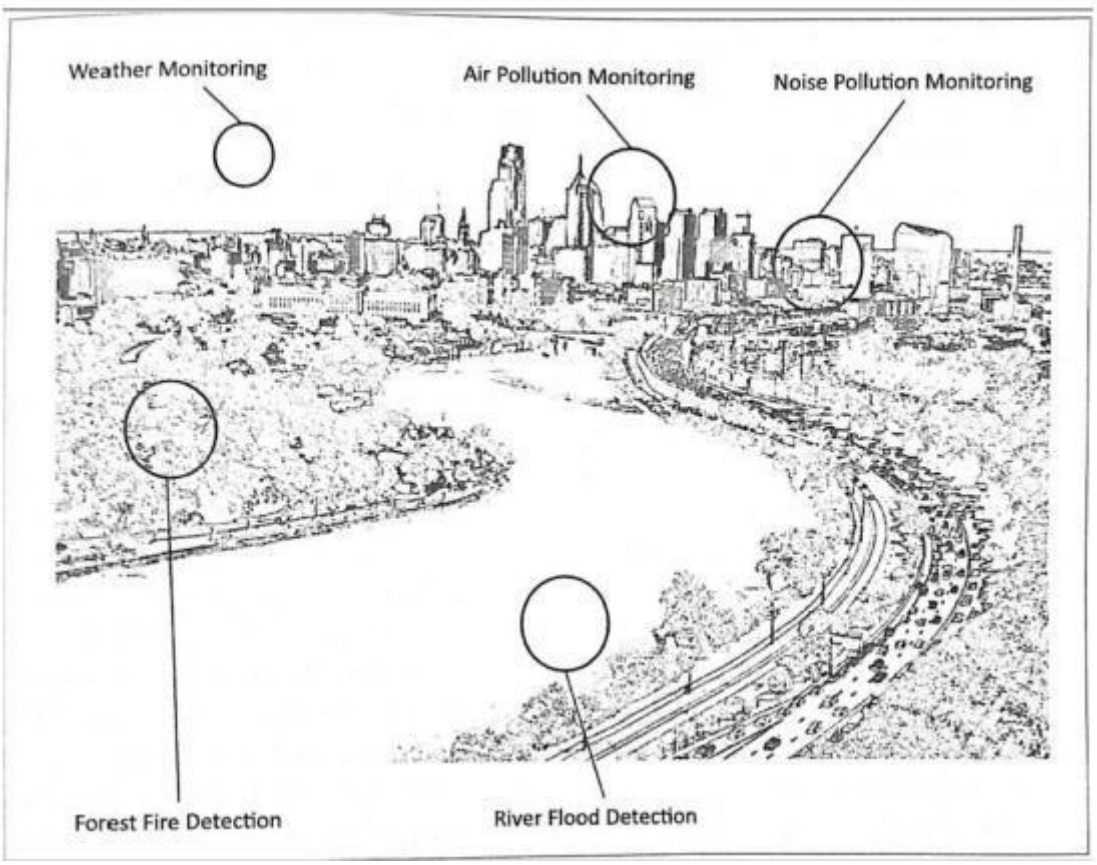
- **Smart Parking**
- Detect the number of empty parking slots
- Send the information over the internet and accessed by smartphones
- **Smart Roads**
- Provide information on driving conditions, traffic congestions, accidents
- Alert for poor driving conditions
- **Structural Health Monitoring**
- Monitor the vibration levels in the structures (bridges and buildings)
- Advance warning for imminent failure of the structure
- **Surveillance**
- Use the large number of distributed and internet connected video surveillance cameras
- Aggregate the video in cloud-based scalable storage solutions
- **Emergency Response**
- Used for critical infrastructure monitoring
- Detect adverse events

Environment

Weather Monitoring

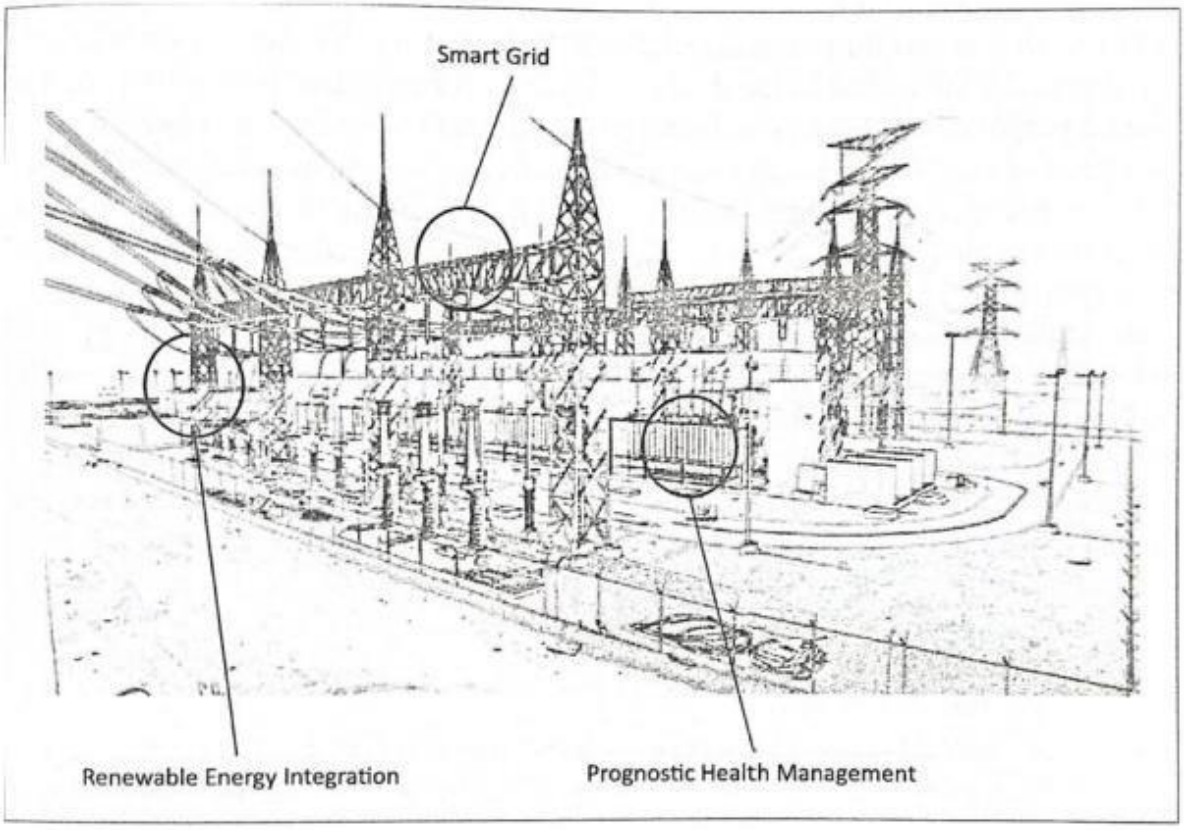
- Collect data from several sensors (temperature, humidity, pressure, etc.)
- Send the data to cloud-based applications and storage back-ends
- **Air Pollution Monitoring**
- Monitor emission of harmful gases (CO2, CO, NO, NO2 , etc.)
- Factories and automobiles use gaseous and meteorological sensors
- Integration with a single-chip microcontroller, several air pollution sensors, GPRS-modem, and a GPS module

• **Noise Pollution Monitoring**
• Use a number of noise monitoring stations
• Generate noise maps from data collected
• **Forest Fire Detection**
• Use a number of monitoring nodes deployed at different locations in a forests
• Use temperature, humidity, light levels, etc.
• Provide early warning of potential forest fire
• Estimates the scale and intensity
• **River Floods Detection**
• Monitoring the water level (using ultrasonic sensors) and flow rate (using the flow velocity sensors)
• Raise alerts when rapid increase in water level and flow rate is detected.



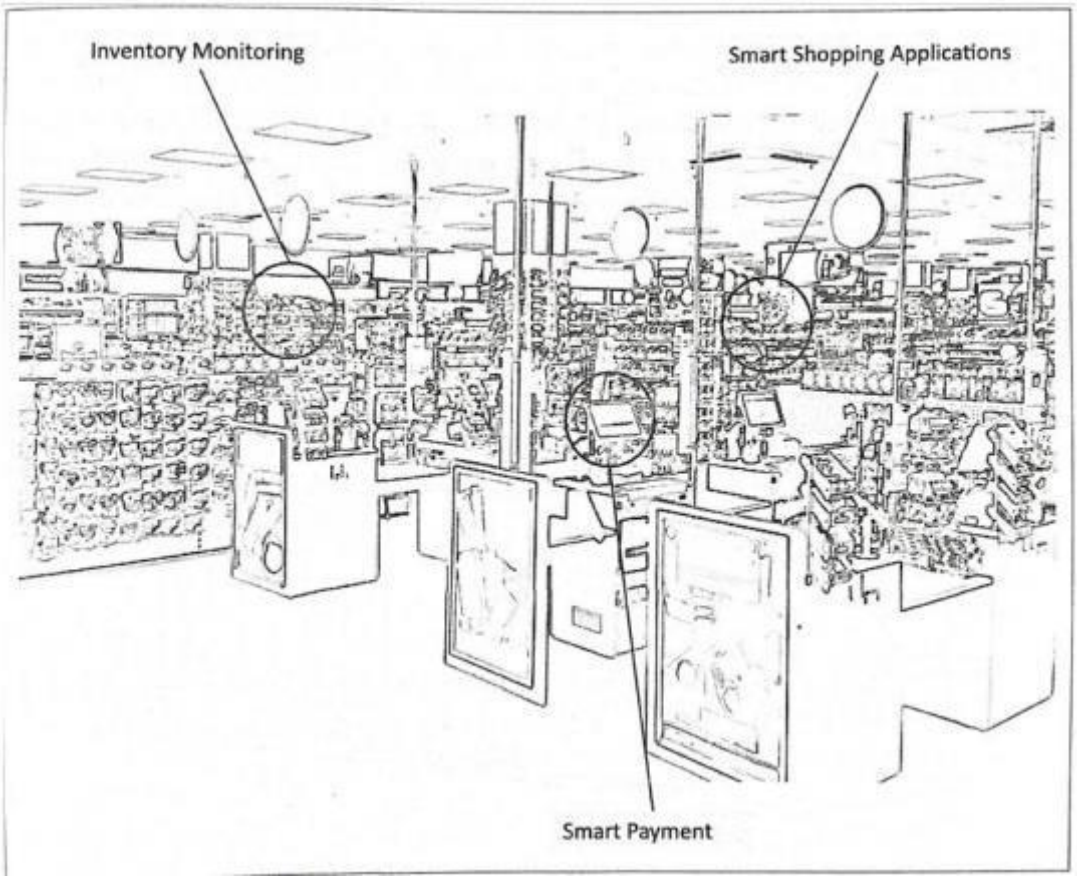
Energy

• **Smart Grids**
• Collect data regarding electricity generation, consumption, storage (conversion of energy into other forms), distribution, equipment health data
• Control the consumption of electricity
• Remotely switch off supply
• **Renewable Energy Systems**
• Measure the electrical variables
• Measure how much the power is fed into the grid
• **Prognostics**
• Predict performance of machines or energy systems
• By collect and analyze the data from sensors.



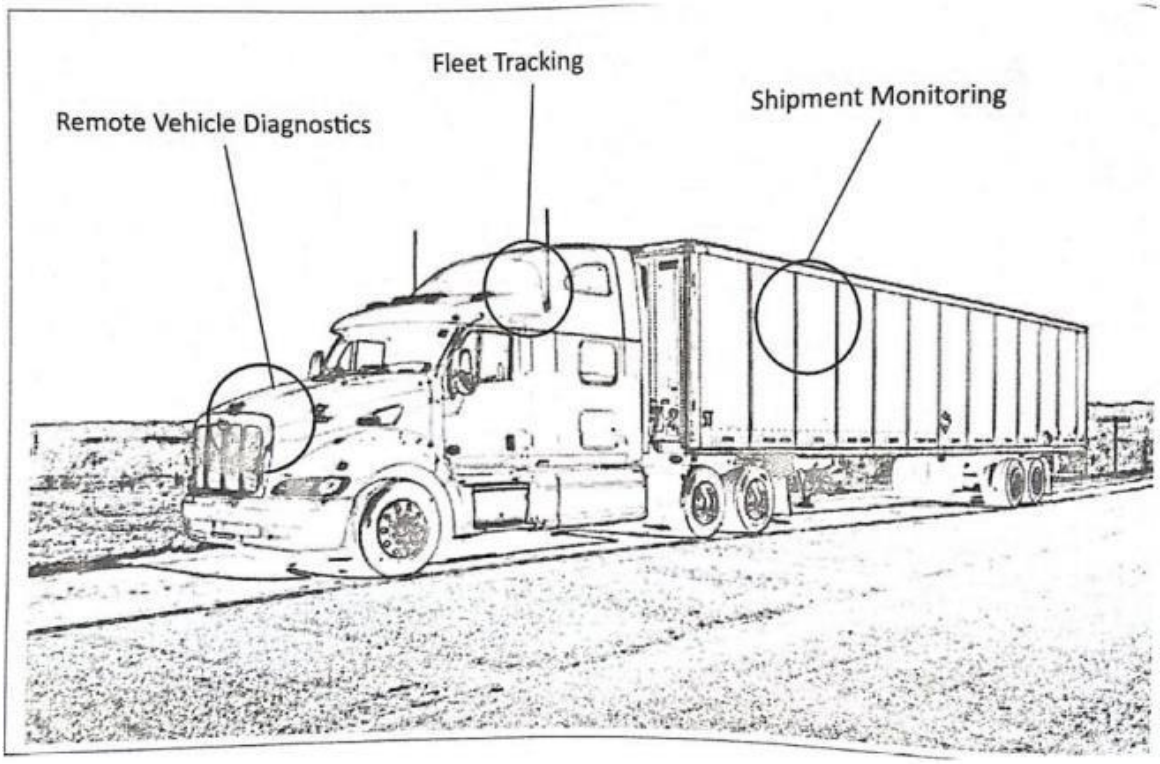
Retail

- **Inventory Management**
  - Monitoring the inventory by the RFID readers
  - Tracking the products
- **Smart Payments**
  - Use the NFC
  - Customers store the credit card information in their NFC-enabled
- **Smart Vending Machines**
- Allow remote monitoring of inventory levels
- Elastic pricing of products
- Contact-less payment using NFC
- Send the data to the cloud for predictive maintenance
  - The information of inventory levels
- The information of the nearest machine in case a product goes out of stock in a machine



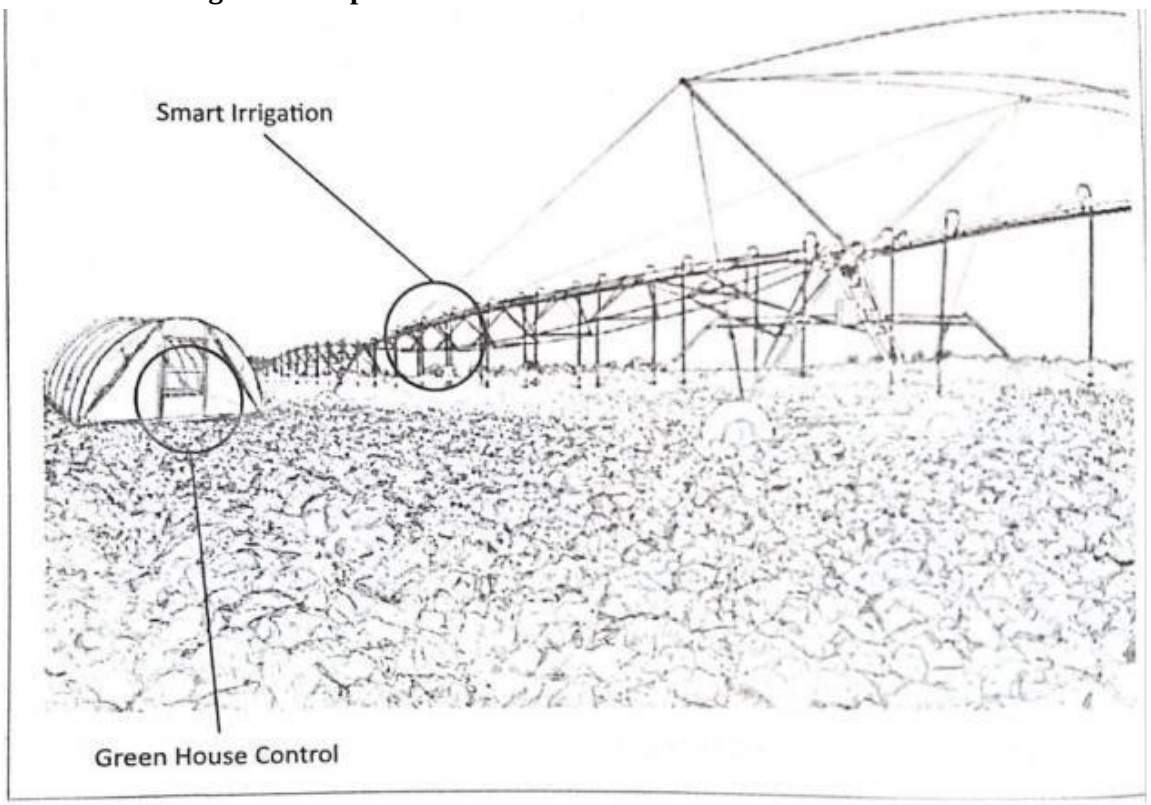
Logistics

- **Route Generation & Scheduling**
- Generate end-to-end routes using combination of route patterns
- Provide route generation queries
- Can be scale up to serve a large transportation network
- **Fleet Tracking**
- Track the locations of the vehicles in real-time
- Generate alerts for deviations in planned routes
- **Shipment monitoring**
- Monitoring the conditions inside containers
- Using sensors (temperature, pressure, humidity)
- Detecting food spoilage
- **Remote Vehicle Diagnostics**
- Detect faults in the vehicle
- Warn of impending faults
- IoT collects the data on vehicle (speed, engine RPM, coolant temperature)
- Generate alerts and suggest remedial actions



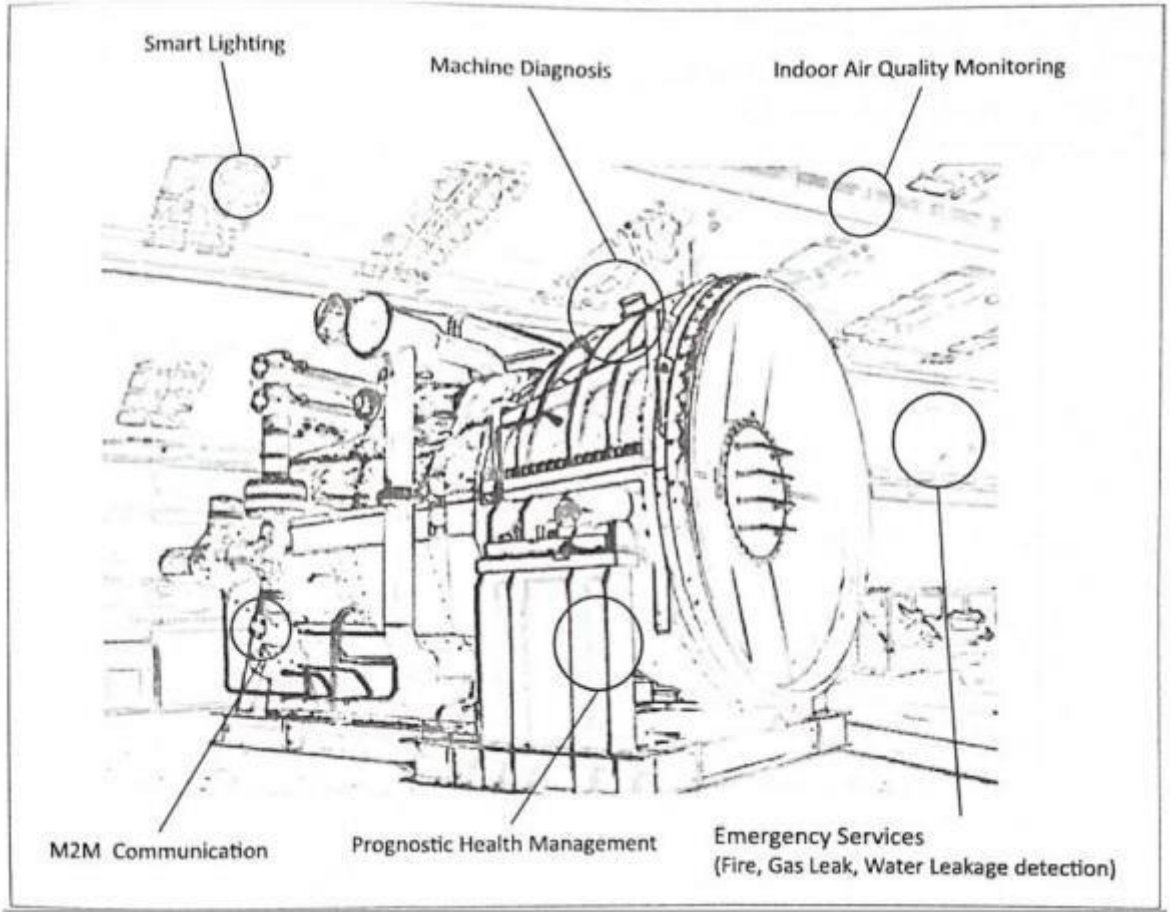
Agriculture

- **Smart Irrigation**
- Use sensors to determine the amount of moisture in the soil
- Release the flow of water
- Using predefined moisture levels
- Water Scheduling
- **Green House Control**
- **Automatically control the climate logical conditions inside a green house**
- Using several sensors to monitor
- Using actuation devices to control
- Valves for releasing water and switches for controlling fans
- **Maintenance of agricultural production**



Industry

- **Machine Diagnosis**
  - Sensors in machine monitor the operating conditions
- For example: temperature & vibration levels
- Collecting and analyzing massive scale machine sensor data
- For reliability analysis and fault prediction in machines
- **Indoor Air Quality Monitoring**
  - Use various gas sensors
  - To monitor the harmful and toxic gases (CO, NO, NO2 , etc.)
  - **Measure the environmental parameters to determine the indoor air quality**
- Temperature, humidity, gaseous pollutants, aerosol



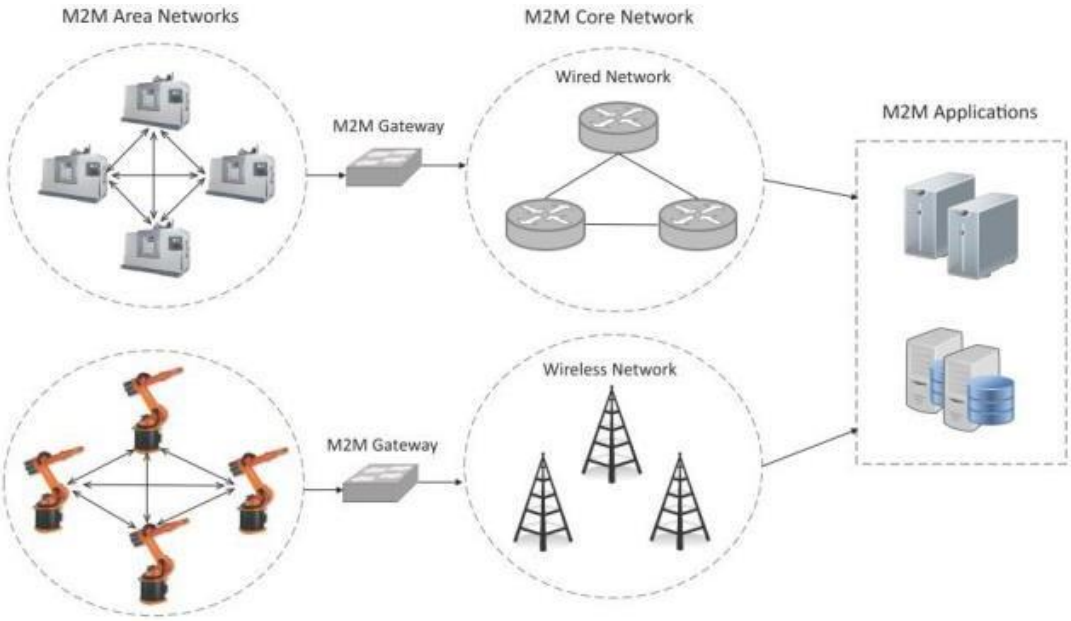
Health & Lifestyle

- **Health & Fitness Monitoring**
  - Collect the health-care data
- Using some sensors: body temperature, heart rate, movement (with accelerometers), etc.
- Various forms : belts and wrist-bands
- **Wearable electronic**
- Assists the daily activities
- Smart watch
  - Smart shoes
- Smart wristbands

UNIT II

Machine-to-Machine (M2M)

Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange.



Machine-to-Machine (M2M)

- An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.

- Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.

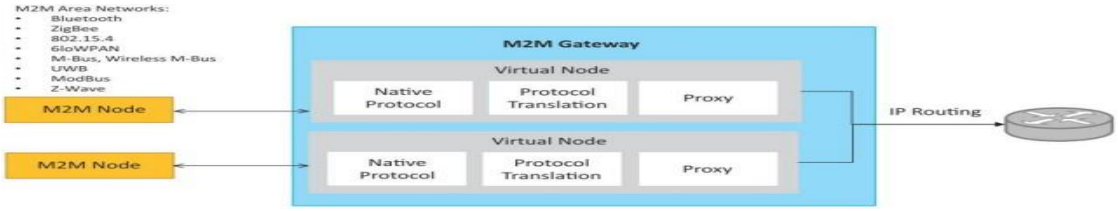
- The communication network provides connectivity to remote M2M area networks.

- The communication network can use either wired or wireless networks (IP- based).

- While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks.

M2M gateway

- Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network.
- To enable the communication between remote M2M area networks, M2M gateways are used



Difference between IoT and M2M

Communication Protocols

• M2M and IoT can differ in how the communication between the machines or devices happens.

• M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks.

• Machines in M2M vs Things in IoT

• The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states.

• M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.

Difference between IoT and M2M

Hardware vs Software Emphasis

• While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.

• **Data Collection & Analysis**

• M2M data is collected in point solutions and often in on-premises storage infrastructure.

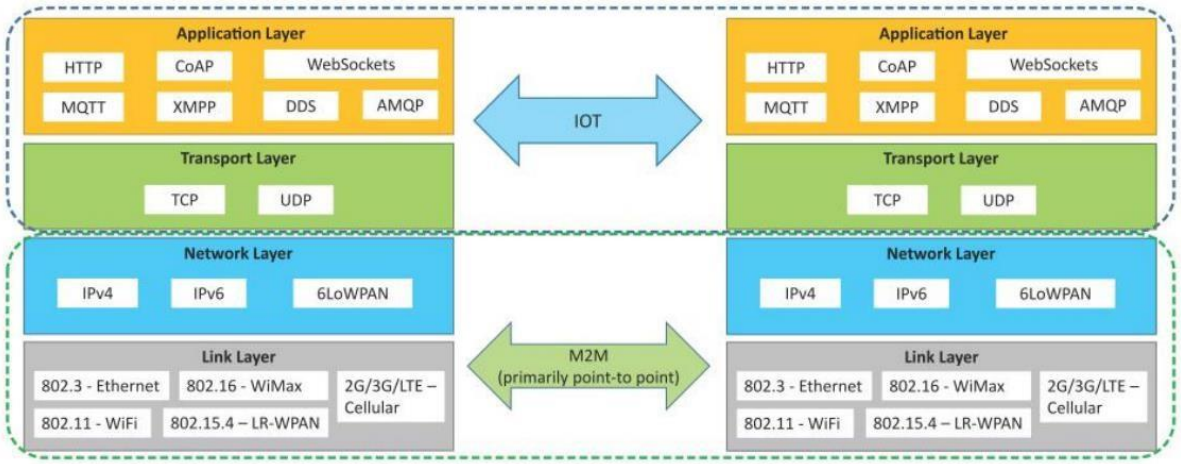
• In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud).

• **Applications**

• M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premisis enterprise applications.

• IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc..

Communication in IoT vs M2M

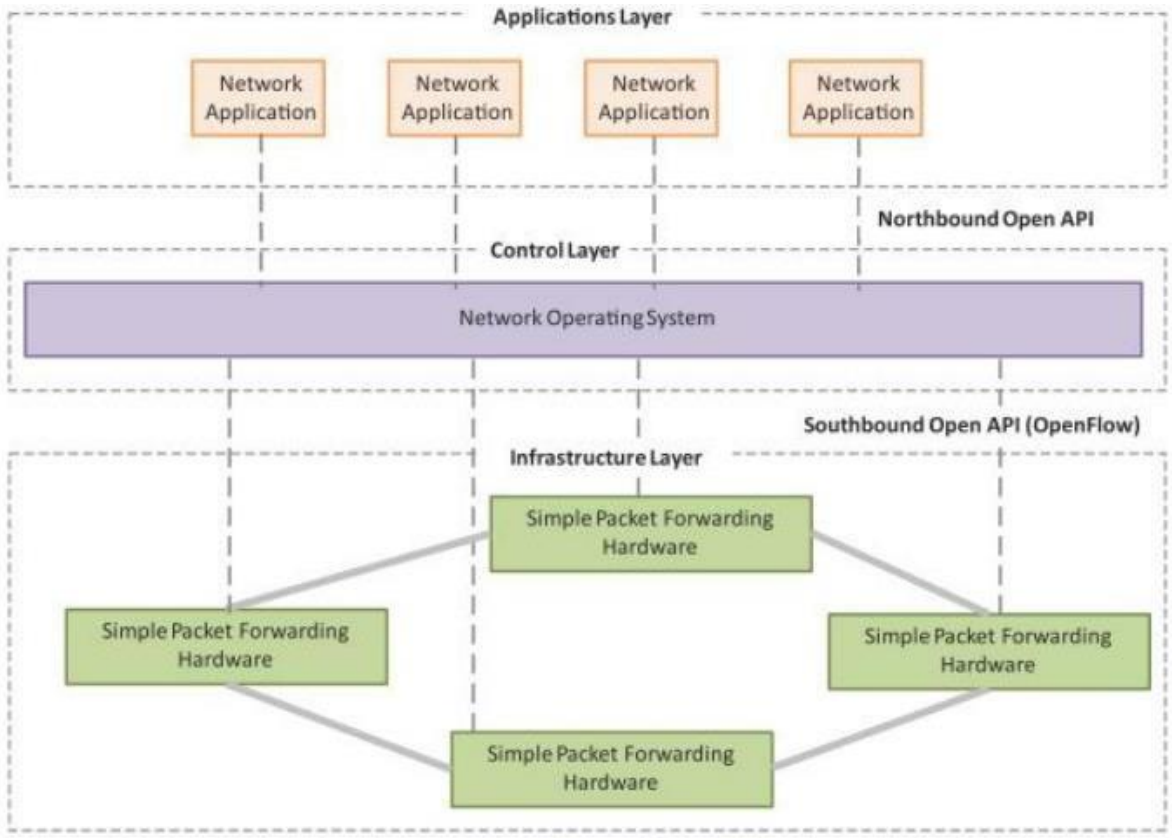


SDN

• Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.

• Software-based SDN controllers maintain a unified view of the network and make confi guration, management and provisioning simpler.

• The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.



Key elements of SDN

- Centralized Network Controller

• With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.

• Programmable Open APIs

• SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).

• Standard Communication Interface (OpenFlow)

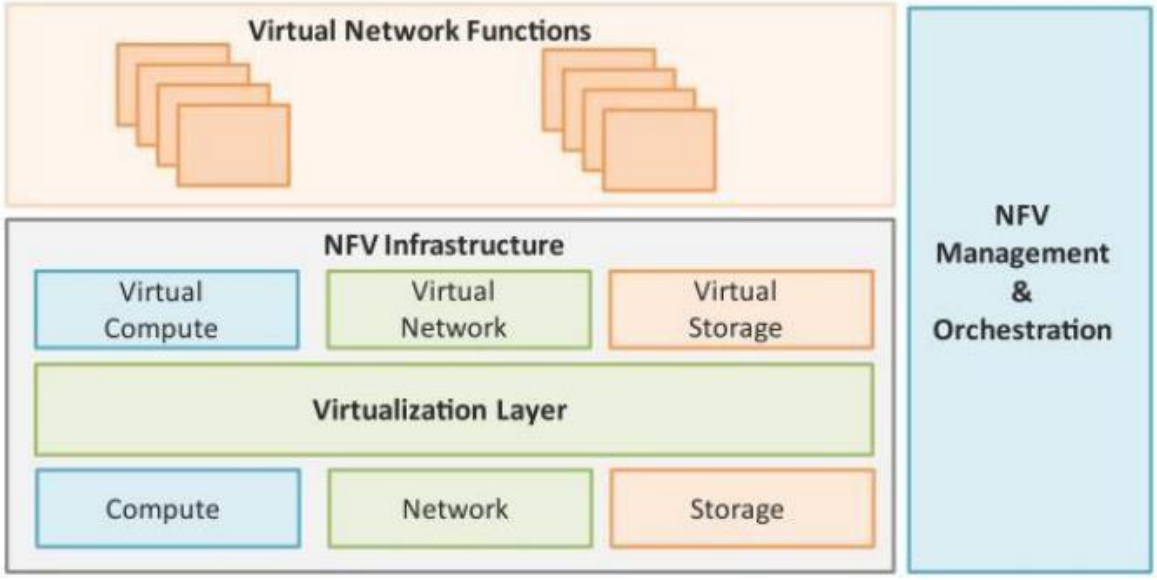
• SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface).

• OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.

NFV

• Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.

• NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.



Key elements of NFV

Virtualized Network Function (VNF):

• VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).

• **NFV Infrastructure (NFVI):**

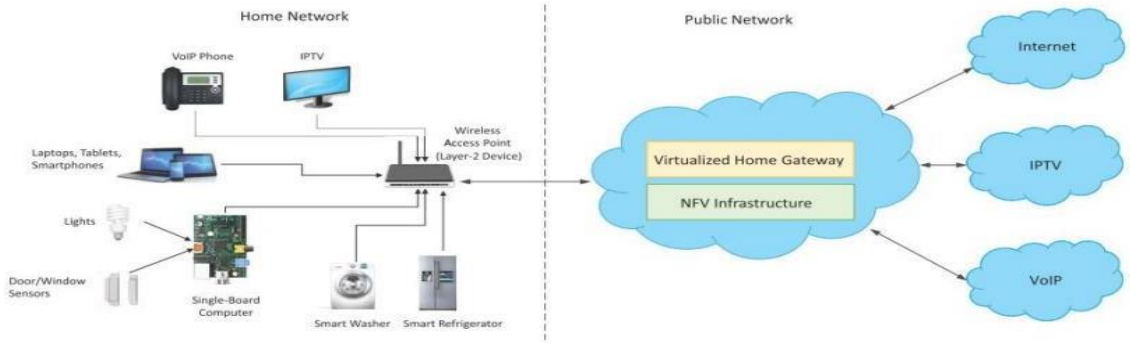
• NFVI includes compute, network and storage resources that are virtualized.

• **NFV Management and Orchestration:**

• NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFS.

NFV Use Case

NFV can be used to virtualize the Home Gateway. The NFV infrastructure in the cloud hosts a virtualized Home Gateway.
The virtualized gateway provides private IP addresses to the devices in the home. The virtualized gateway also connects to network services such as VoIP and IPTV.



IoT System Management

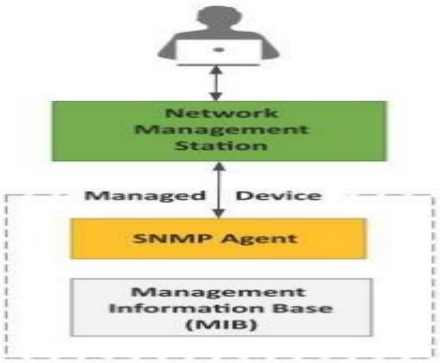
- Need for IoT Systems Management
- SNMP
- YANG
- NETOPEER

Need for IoT Systems Management

- Automating Configuration
- Monitoring Operational & Statistical Data
- Improved Reliability
- System Wide Configurations
- Multiple System Configurations
- Retrieving & Reusing Configurations

Simple Network Management Protocol (SNMP)

- SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc.



- SNMP component include
- Network Management Station (NMS)
- Managed Device
- Management Information Base (MIB)
- SNMP Agent that runs on the device

Limitations of SNMP

- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features.

YANG

YET ANOTHER NEXT GENERATION data send over the network management protocols such as NETCONF and RESTCONF.

• YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol.

• YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.

• YANG modules defines the data exchanged between the NETCONF client and server.

• A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure.

• The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs.

• Leaf nodes are organized using 'container' or 'list' constructs.

• A YANG module can import definitions from other modules.

• Constraints can be defined on the data nodes, e.g. allowed values.

• YANG can model both configuration data and state data using the 'config' statement.

YANG Module Example

This YANG module is a YANG version of the toaster MIB

• The toaster YANG module begins with the header information followed by identity declarations which define various bread types.

• The leaf nodes (‘toasterManufacturer’ , ‘toasterModelNumber’ and oasterStatus’) are defined in the ‘toaster’ container.

• Each leaf node definition has a type and optionally a description and default value.

• The module has two RPC definitions (‘make-toast’ and ‘cancel-toast’).

NETCONF

• NETCONF works on SSH transport protocol.

• Transport layer provides end-to-end connectivity and ensure reliable deliveryof messages.

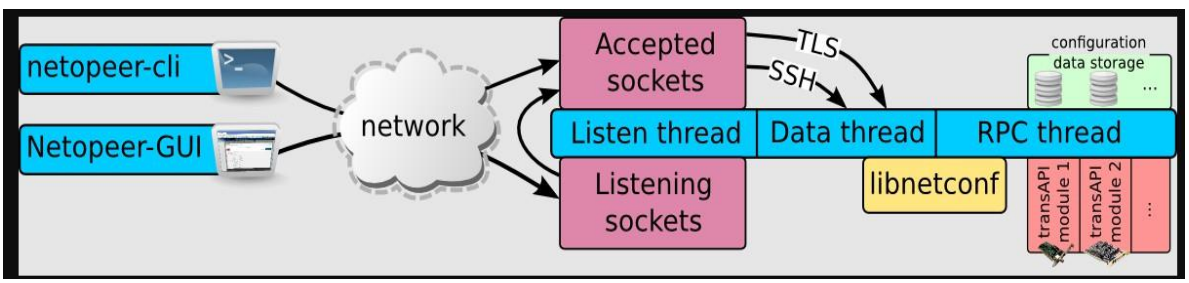
• NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request andresponse messages.

• The RPC layer provides mechanism for encoding of RPC calls and notifications.

• NETCONF provides various operations to retrieve and edit configuration datafromnetwork devices.

NETOPEER

- **Netopeer** is a set of NETCONF tools built on the libnetconf library.
- It allows operators to connect to their NETCONF-enabled devices as well as developers to allow control their devices via NETCONF.
- With the experiences from **Netopeer**, we have moved our activities to work on next generation of this NETCONF toolset based on libyang library.
- Netopeer is mature enough to be used as a replacement of the original **Netopeer** tools. Therefore, the **Netopeer** is no more developed neither maintained.

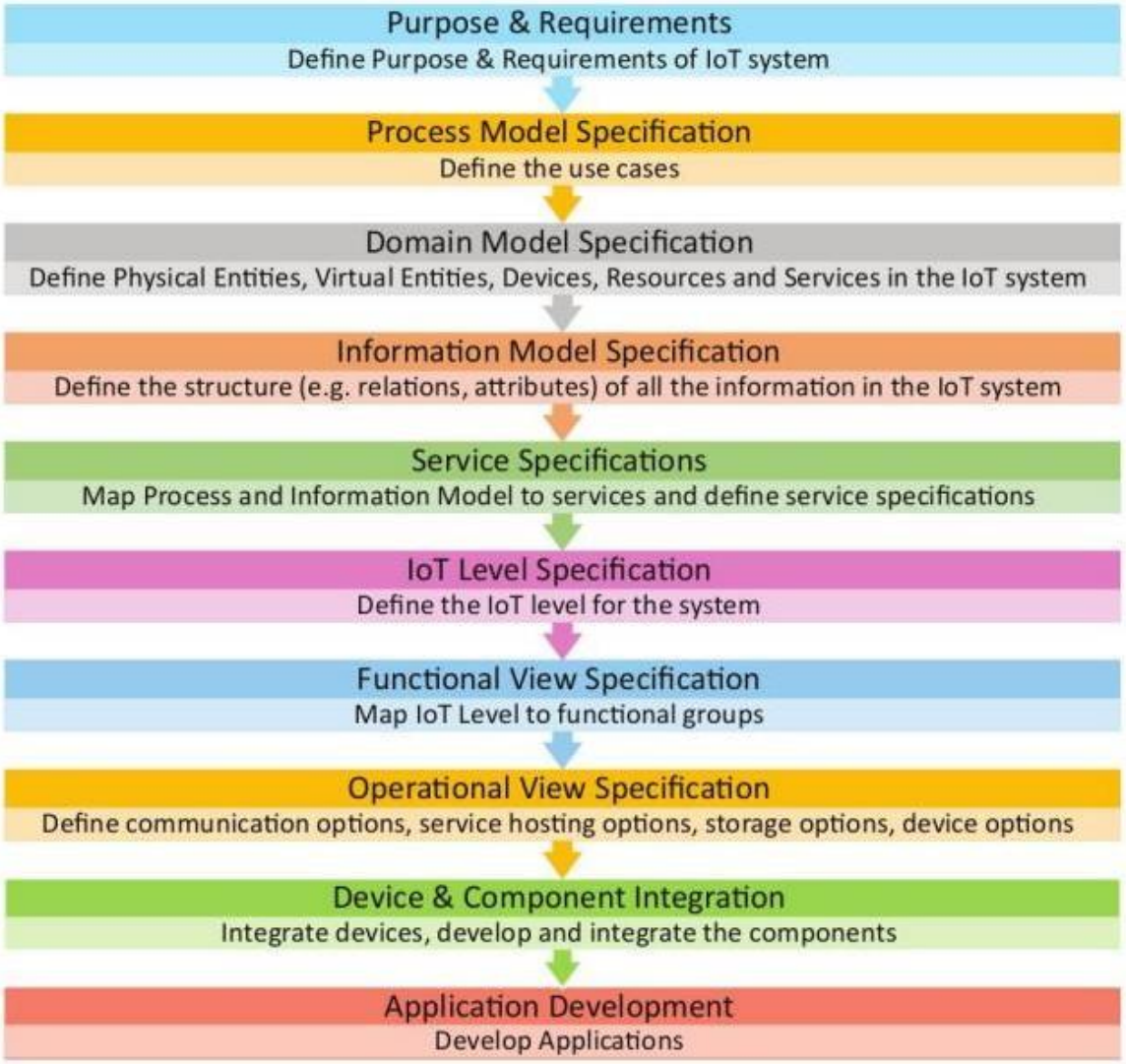


UNIT III

IoT Design Methodology that includes:

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification
- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development

IoT Design Methodology - Steps



Step 1: Purpose & Requirements Specification

• The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured.

Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:

• **Purpose :** A home automation system that allows controlling of the lights in a home remotely using a web application.

• **Behavior :** The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.

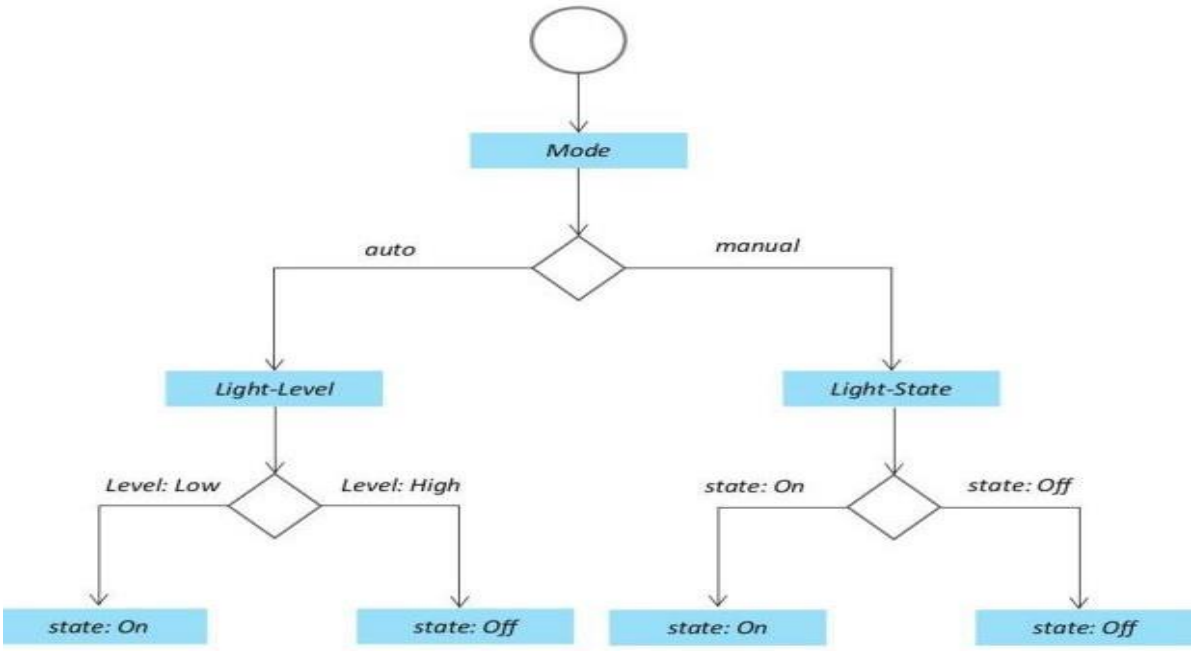
• **System Management Requirement :** The system should provide remote monitoring and control functions.

• **Data Analysis Requirement :** The system should perform local analysis of the data.• Application Deployment Requirement : The application should be deployed locally on the device, but should be accessible remotely.

• **Security Requirement :** The system should have basic user authentication capability.

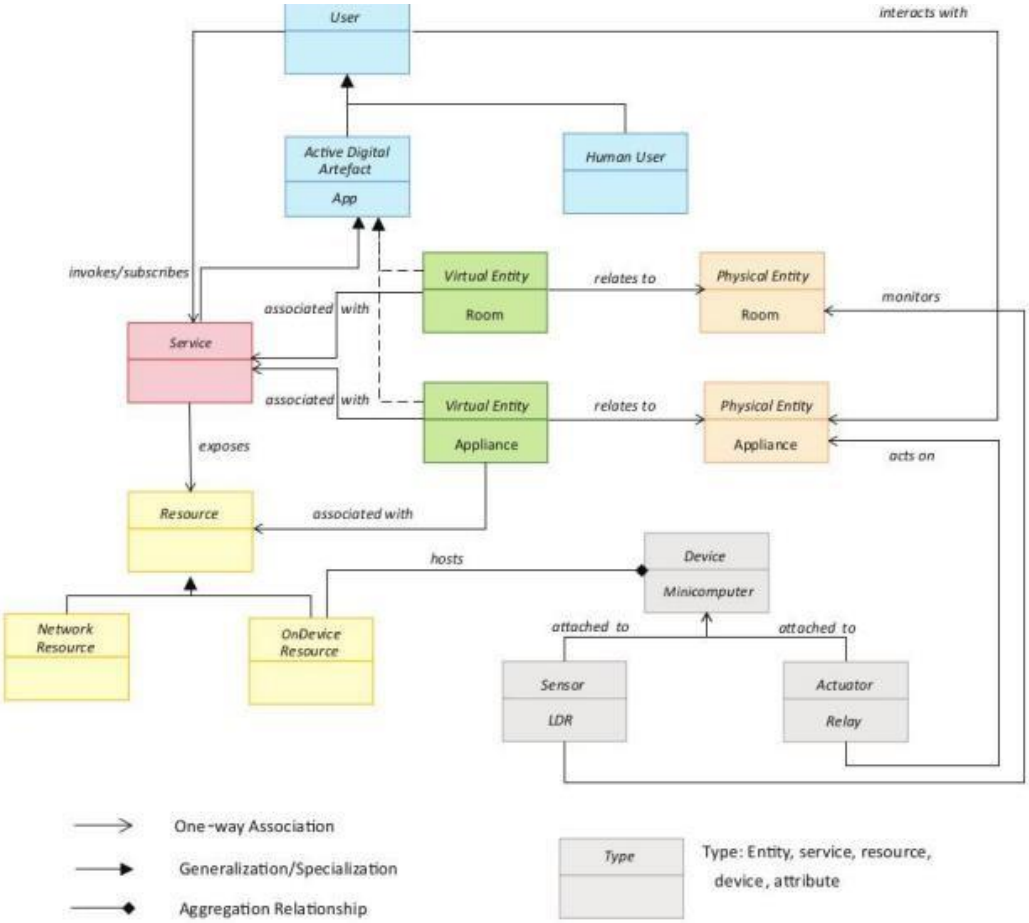
Step2: Process Specification

• The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.



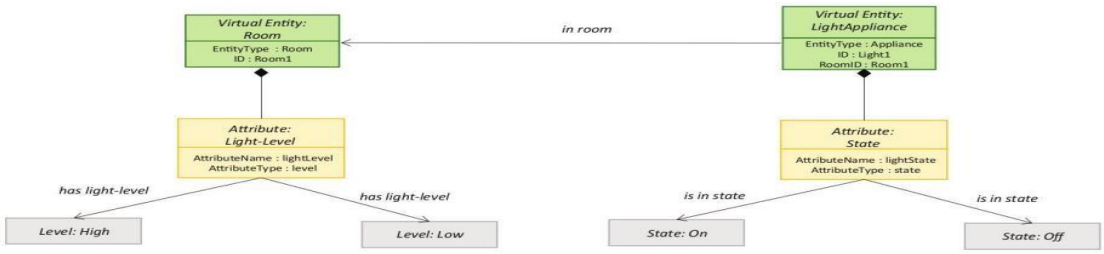
Step 3: Domain Model Specification

• The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.



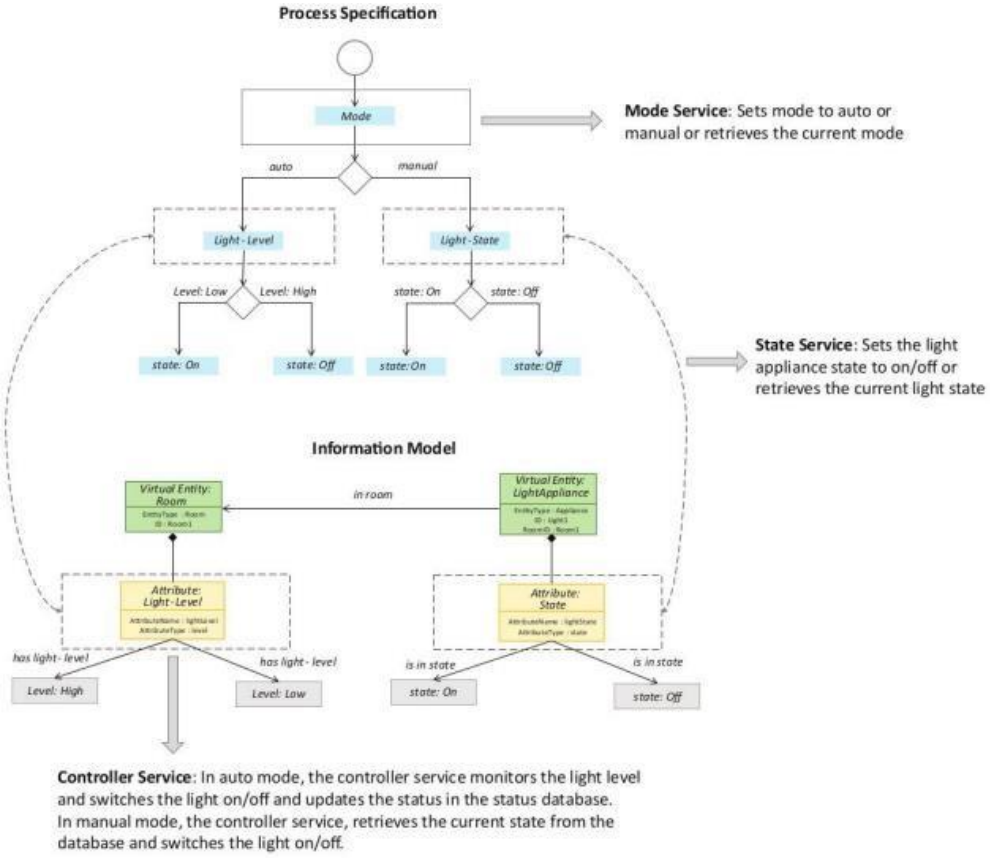
Step 4: Information Model Specification

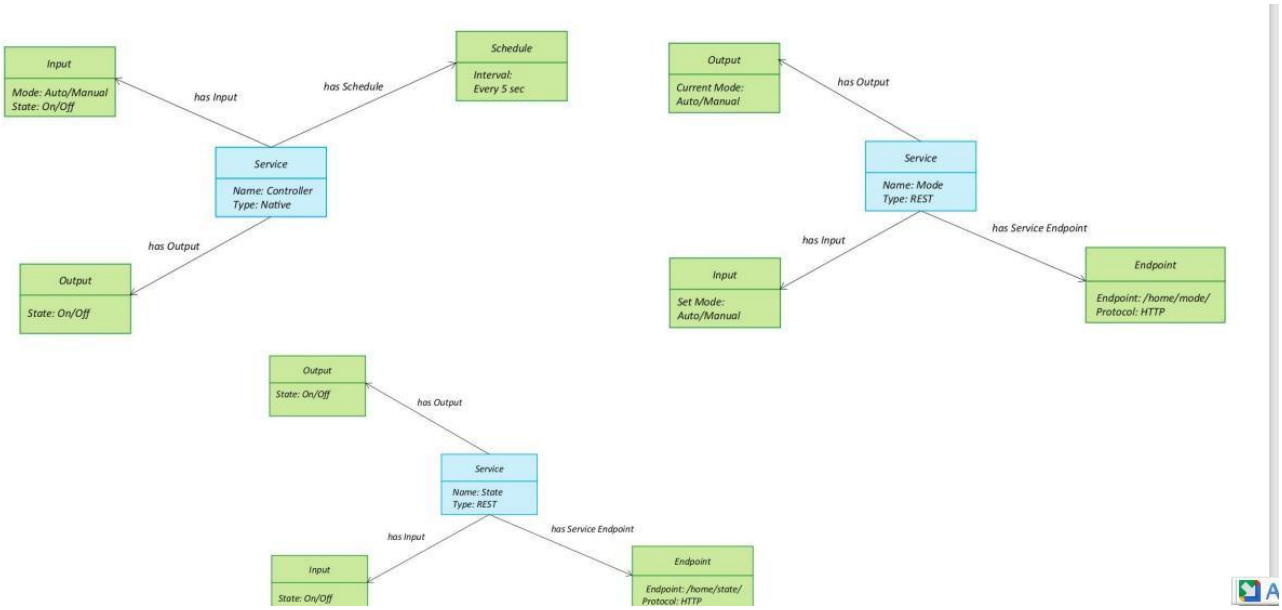
• The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.



Step 5: Service Specifications

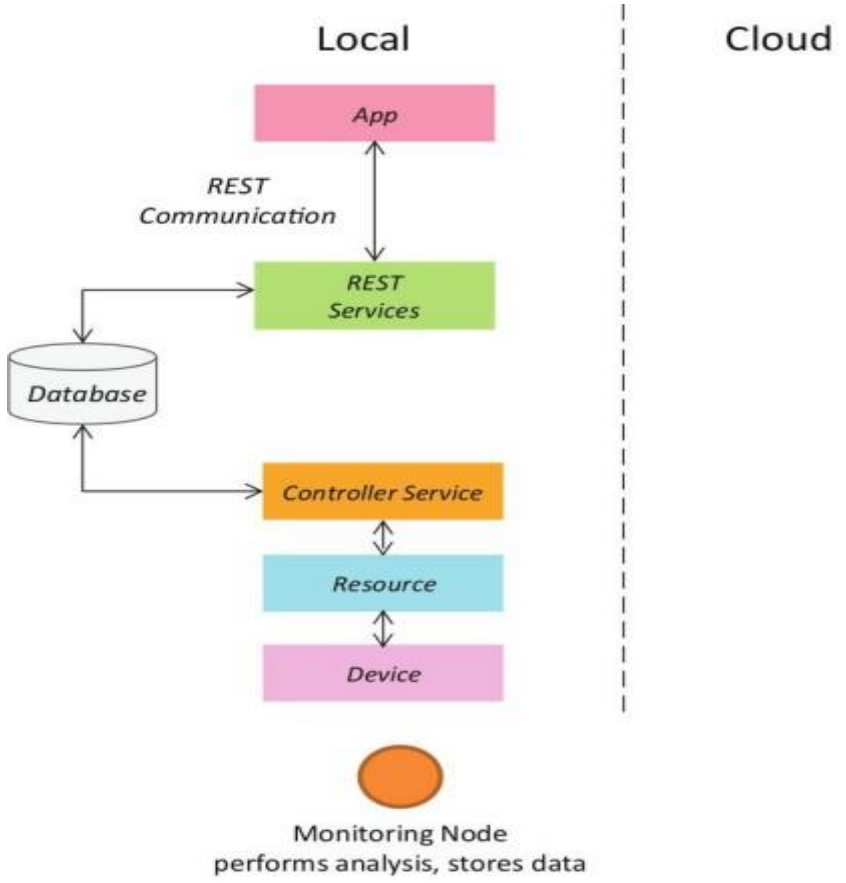
• The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.





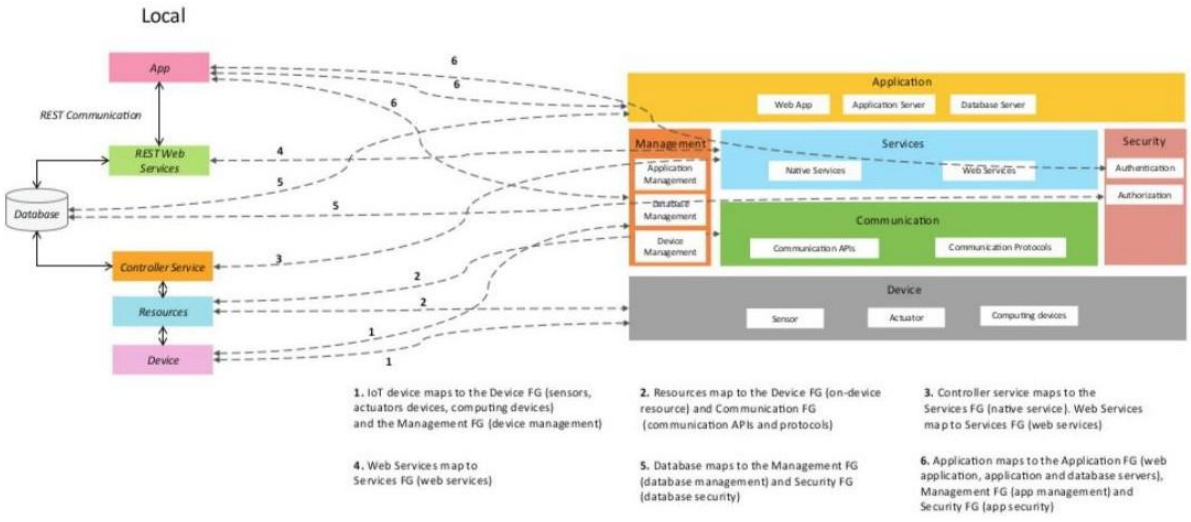
Step 6: IoT Level Specification

•The sixth step in the IoT design methodology is to define the IoT level for the system. In Chapter-1, we defined five IoT deployment levels.



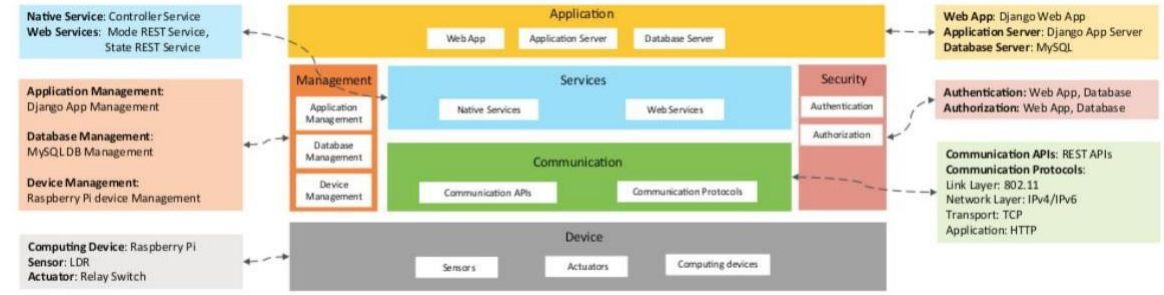
Step 7: Functional View Specification

• The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.



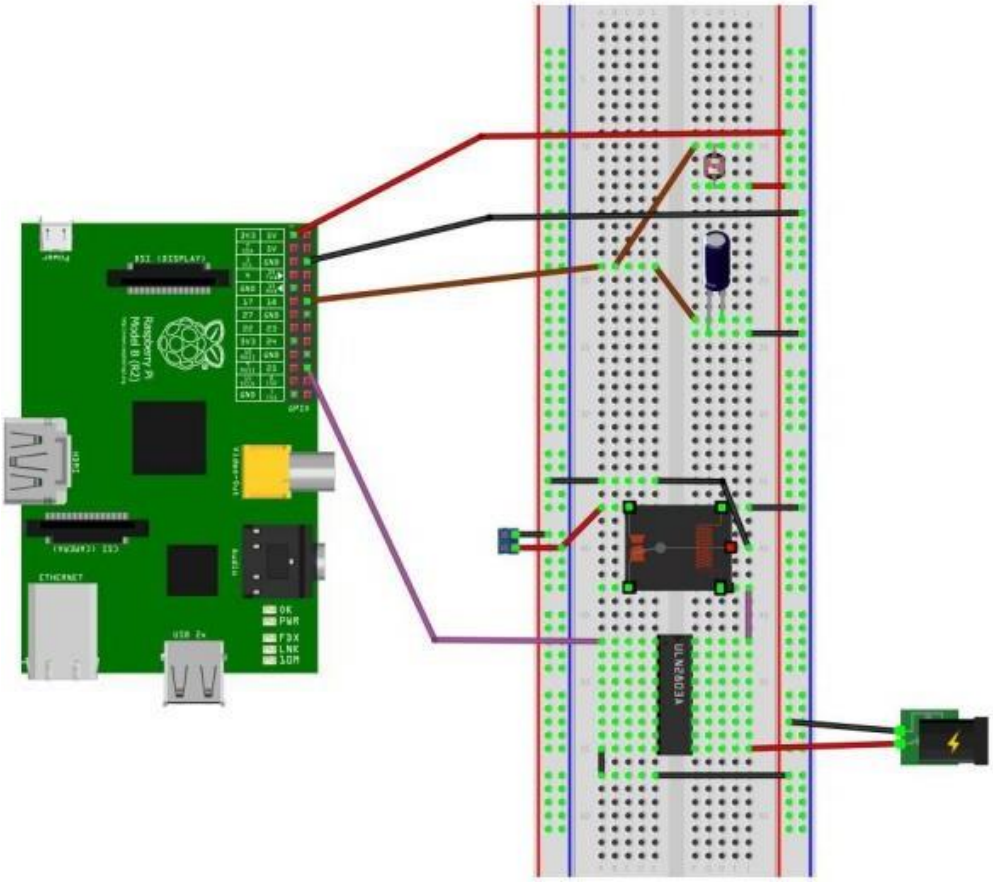
Step 8: Operational View Specification

• The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc.



Step 9: Device & Component Integration

• The ninth step in the IoT design methodology is the integration of the devices and component.



Step 10: Application Development

- The final step in the IoT design methodology is to develop the IoT applications.
- **Auto**

  Controls the light appliance automatically based on the lighting conditions in the room
- **Light**

  When Auto mode is off, it is used for manually controlling the light appliance.

  When Auto mode is on, it reflects the current state of the light appliance.

Unit-IV

- Introduction to Python
- Installing Python
- Python Data Types & Data Structures
- Control Flow
- Functions
- Modules
- Packages
- File Input/Output
- Date/Time Operations
- Classess

Python

Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.

The main characteristics of Python are:

1. **Multi-paradigm programming language**

Python supports more than one programming paradigms including object-oriented programming and structured programming

2. **Interpreted Language**

Python is an interpreted language and does not require an explicit compilation step. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.

3. **Interactive Language**

Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

Python – Benefits

1. **Easy-to-learn, read and maintain**

Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages.

Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.

2. **Object and Procedure Oriented**

Python supports both procedure-oriented programming and object-oriented programming.

Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.

3. **Extendable**

Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.

4 **Scalable**

• Due to the minimalistic nature of Python, it provides a manageable structure for large programs.

5. Portable

Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source.

6. Broad Library Support

Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.

Python – Setup

Windows

• Python binaries for Windows can be downloaded from http://www.python.org/getit .

• For the examples and exercise in this book, you would require Python 2.7 which can be directly downloaded from: http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi

• Once the python binary is installed you can run the python shell at the command prompt using > python.

Linux

#Install Dependencies
sudo apt-get install build-essential
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev
#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5
#Install Python
./configure make sudo make install

Numbers

Number data type is used to store numeric values. Numbers are immutable data types, therefore changing the value of a number data type results in a newly allocated object.

Example

#Integer
>>>a=5
>>>type(a)
#Floating Point
>>>b=2.5
>>>type(b)

#Long

```
>>>x=9898878787676L
>>>type(x)
#Complex >>>y=2+5j
 >>>y
(2+5j)
>>>type(y)
<type ‘complex’>
 >>>y.real
 2 >>>y.imag 5
```

Example 2

#Addition

```
 >>>c=a+b
>>>c
7.5 >>>type(c)
```

#Subtraction

```
>>>d=a-b
>>>d
2.5 >>>type(d)
type<’float’>
```

#Multiplication

```
 >>>e=a*b
>>>e 12.5
>>>type(e)
type<’float’>
```

Strings

A string is simply a list of characters in order. There are no limits to the num number.

Example

```
#Create string
 >>>s="Hello World!"
 >>>type(s)
#String concatenation
>>>t="This is sample program."
 >>>r = s+t
>>>r
’Hello World!This is sample program.’
#Get length of string
 >>>len(s)
 12
#Convert string to integer
 >>>x="100"
 >>>type(s)
<type ‘str’>
 >>>y=int(x)
 >>>y
100
```

Lists

List a compound data type used to group together other values. List items need not all have the same type. A list contains items separated by commas and enclosed within square brackets.

Tuples

A tuple is a sequence data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed within parentheses. Unlike lists, the elements of tuples cannot be changed.

```
#Create a Tuple
>>>fruits=("apple","mango","banana","pineapple")
 >>>fruits
('apple', 'mango', 'banana', 'pineapple')
>>>type(fruits)
#Get length of tuple
>>>len(fruits)
 4
```

Dictionaries

Dictionary is a mapping data type or a kind of hash table that maps keys to values. Keys in a dictionary can be of any data type, though numbers and strings are commonly used for keys. Values in a dictionary can be any data type or object.

Example

```
#Create a dictionary
 >>>student={'name':'Mary','id':'8776','major':'CS'}
>>>student
 {'major': 'CS', 'name': 'Mary', 'id': '8776'}
>>>type(student)
<type 'dict'>
#Get length of a dictionary
 >>>len(student)
 3
#Get the value of a key in dictionary
 >>>student['name']
 'Mary'
 #Get all items in a dictionary
>>>student.items()
[('gender', 'female'), ('major', 'CS'), ('name', 'Mary'),
('id', '8776')
```

Type Conversions

Examples

```
#Convert to string
>>>a=10000
>>>str(a)
 '10000'
#Convert to int
>>>b="2013"
 >>>int(b)
2013
```

#Convert to float

```
>>>float(b)
2013.0
```

Control Flow – if statement

• The if statement in Python is similar to the if statement in other languages.

```
>>>a = 25**5
>>>if a>10000:
print "More"
else:
print "Less"
if a>10000:
if a<1000000:
print”Between 10k and 100k”
else:
print”More than 100k”
```

Control Flow – for statement

The for statement in Python iterates over items of any sequence (list, string, etc.) in the order in which they appear in the sequence.

This behavior is different from the for statement in other languages such as C in which an initialization, incrementing and stopping criteria are provided.

Example

```
#Looping over characters in a string
helloString = "Hello World"
for c in helloString:
 print c.
```

Control Flow – while statements

The while statement in Python executes the statements within the while loop as long as the while condition is true.

Example

```
#Prints even numbers upto 100
>>> i = 0
>>> while i<=100:
if i%2 == 0:
print i
 i = i+1
```

Control Flow – range statement

Example

```
#Generate a list of numbers from 0 – 9
>>>range (10)
[0, 1, 2, 3, 4, 5,6,7,8,9]
```

Control Flow – break/continue Statement

The break and continue statements in Python are similar to the statements in C.

Break

Break statement breaks out of the for/while loop.

Example

```
#Break statement example
>>>y=1
>>>for x in range(4,256,4):
y = y * x
if y > 512:
break
print y
4
32
384
```

Continue

• Continue statement continues with the next iteration.

```
#Continue statement example
>>>fruits=['apple','orange','banana','mango']
>>>for item in fruits:
if item == "banana":
continue
else:
print item
apple
orange
mango
```

Control Flow – pass statement

The pass statement in Python is a null operation.

The pass statement is used when a statement is required syntactically but you do not want any command or code to execute.

Example

```
>fruits=['apple','orange','banana','mango']
>for item in fruits: if item == "banana":
Pass
else:
print item
apple
orange
mango
```

Functions - Default Argument

Functions can have default values of the parameters.

If a function with default values is called with fewer parameters or without any parameter, the default values of the parameters are used.

```
>>>def displayFruits(fruits=['apple','orange']):
print "There are %d fruits in the list" % (len(fruits))
for item in fruits:
print item
```

```
#Using default arguments
>>>displayFruits()
apple
orange
>>>fruits = [’banana’, ’pear’, ’mango’]
>>>displayFruits(fruits)
banana
pear
mango
```

Functions - Passing by Reference

All parameters in the Python functions are passed by reference.

If a parameter is changed within a function the change also reflected back in the calling fuction.

```
>>>def displayFruits(fruits):
print "There are %d fruits in the list" % (len(fruits))
for item in fruits:
print item
print "Adding one more fruit"
fruits.append('mango')
>>>fruits = ['banana', 'pear', 'apple']
>>>displayFruits(fruits)
There are 3 fruits in the list
Banana
Pear
Apple
#Adding one more fruit
>>>print "There are %d fruits in the list" % (len(fruits))
There are 4 fruits in the list
```

Functions - Keyword Arguments

Functions can also be called using keyword arguments that identifies the arguments by the parameter name when the function is called.

```
#Correct use
>>>printStudentRecords(name=’Alex’)
Name: Alex
Age: 20
Major: CS
>>>printStudentRecords(name=’Bob’,age=22,major=’EC E’)
Name: Bob
Age: 22
Major: ECE
>>>printStudentRecords(name=’Alan’,major=’ECE’)
Name: Alan
Age: 20
Major: EC
```

Functions - Variable Length Arguments

Python functions can have variable length arguments. The variable length arguments are passed to as a tuple to the function with an argument prefixed with asterix (*)
Example
>>>def student(name, *varargs):
print "Student Name: " + name
for item in varargs:
print item
>>>student('Nav')
Student Name: Nav
>>>student('Amy', 'Age: 24')
Student Name: Amy
Age: 24
>>>student('Bob', 'Age: 20', 'Major: CS')
Student Name: Bob
Age: 20
Major: C

Modules

Python allows organizing the program code into different modules which improves the code readability and management.
• A module is a Python file that defines some functionality in the form of functions or classes. • Modules can be imported using the import keyword.
• Modules to be imported must be present in the search path.

Example
#student module - saved as student.py
def averageGrade(students):
sum = 0.0
for key in students:
sum = sum + students[key]['grade']
average = sum/len(students)
return average
def printRecords(students):
print "There are %d students" %(len(students))
i=1
for key in students:
print "Student-%d: " % (i)
print "Name: " + students[key]['name']
print "Grade: " + str(students[key]['grade'])
i = i+1

Packages

Python package is hierarchical file structure that consists of modules and subpackages.
Packages allow better organization of modules related to a single application environment.

File Handling

Python allows reading and writing to files using the file object.
• The open(filename, mode) function is used to get a file object.
• The mode can be read (r), write (w), append (a), read and write (r+ or w+), read-binary (rb), write-binary (wb), etc.
• After the file contents have been read the close function is called which closes the file object.

Example

```
# Example of reading a certain number of bytes
>>>fp = open('file.txt','r')
>>>fp.read(10)
 'Python sup' >>>fp.close()
```

Date/Time Operation

Python provides several functions for date and time access and conversions.

• The datetime module allows manipulating date and time in several ways.

• The time module in Python provides various time-related functions.

Example

```
# Examples of manipulating with date
>>>from datetime import date
 >>>now = date.today()
>>>print "Date: " + now.strftime("%m-%d-%y")
Date: 07-24-13
>>>print "Day of Week: " + now.strftime("%A")
Day of Week: Wednesday
>>>print "Month: " + now.strftime("%B")
 Month: July
>>>then = date(2013, 6, 7)
 >>>timediff = now – then
 >>>timediff.days
47
```

Classes

Python is an Object-Oriented Programming (OOP) language. Python provides all the standard features of Object Oriented Programming such as classes, class variables, class methods, inheritance, function overloading, and operator overloading.

• **Class**

A class is simply a representation of a type of object and user-defined prototype for an object that is composed of three things: a name, attributes, and operations/methods.

• **Instance/Object**

Object is an instance of the data structure defined by a class.

• **Inheritance**

Inheritance is the process of forming a new class from an existing class or base class.

• **Function overloading**

Function overloading is a form of polymorphism that allows a function to have different meanings, depending on its context.

• **Operator overloading**

Operator overloading is a form of polymorphism that allows assignment of more than one function to a particular operator.

IoT physical devices and End points

• Basic building blocks of an IoT Device

• Exemplary Device: Raspberry Pi

• Raspberry Pi interfaces

• Programming Raspberry Pi with Python

• Other IoT devices

What is an IoT Device

A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smart TV, computer, refrigerator, car, etc.).

IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely.

IoT Device Examples

A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.

An industrial machine which sends information abouts its operation and health monitoring data to a server.

A car which sends information about its location to a cloud-based service.

A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based services.

1. Basic building blocks of an IoT Device

- **Sensing**

Sensors can be either on-board the IoT device or attached to the device.

- **Actuation**

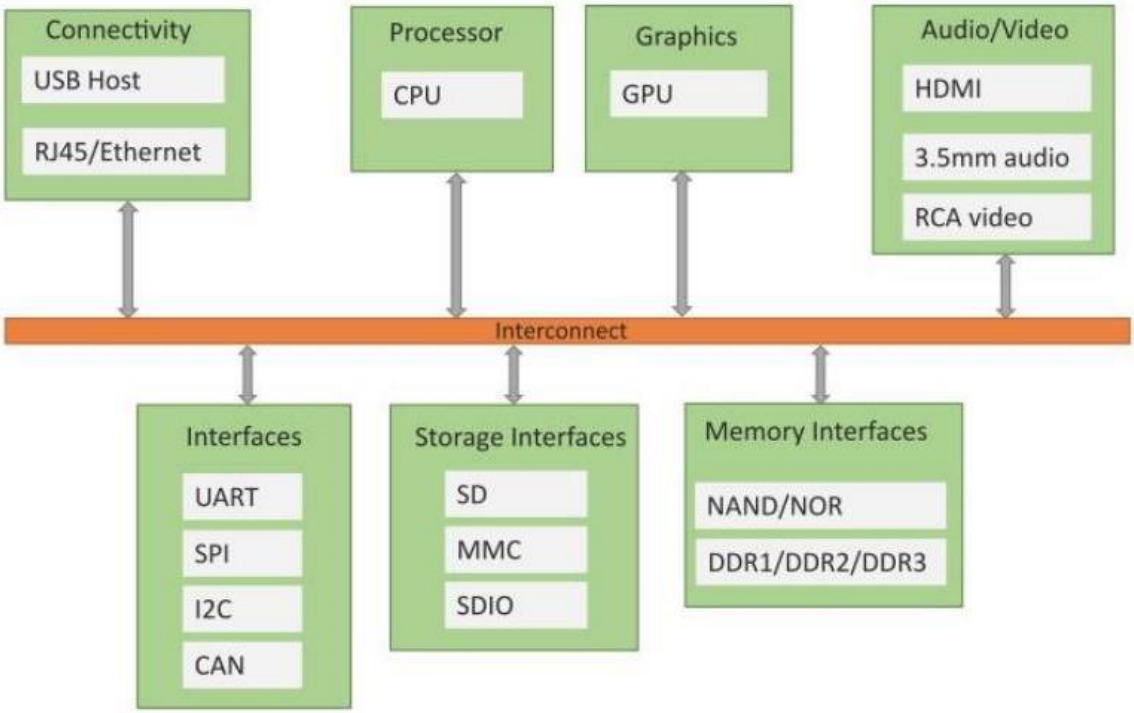
IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device.

- **Communication**

Communication modules are responsible for sending collected data to other devices or cloud- based servers/storage and receiving data from other devices and commands from remote applications.

- **Analysis & Processing**

Analysis and processing modules are responsible for making sense of the collected data.



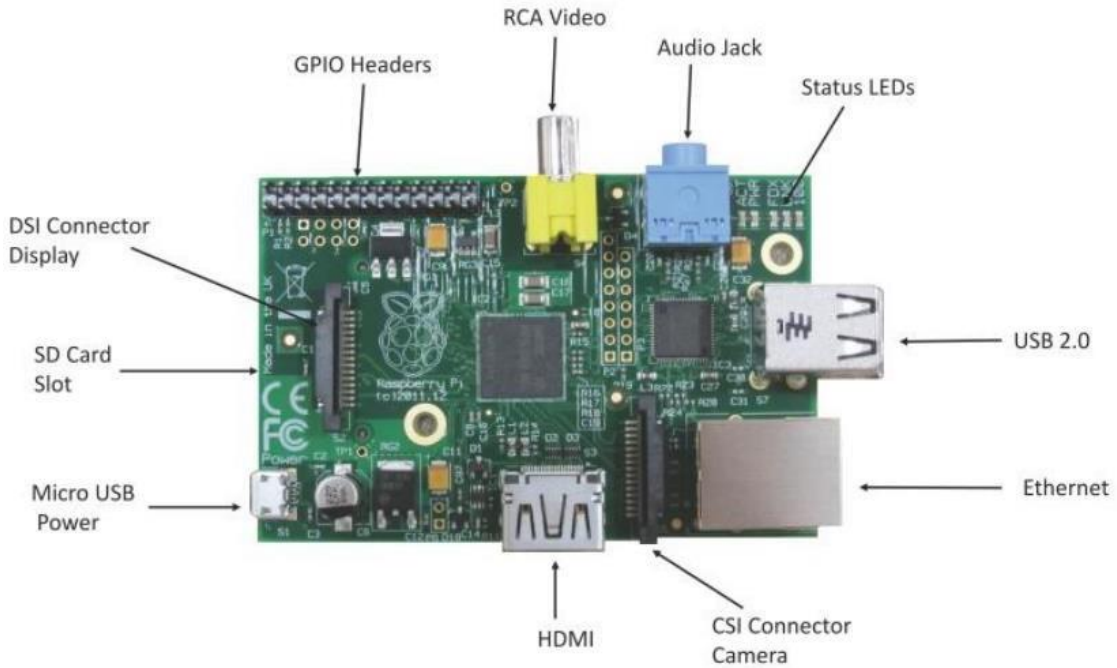
Raspberry Pi

• Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.

• Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.

• Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins.

• Since Raspberry Pi runs Linux operating system, it supports Python "out of box"



Linux on Raspberry Pi

• Raspbian

Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.

• Arch

Arch is an Arch Linux port for AMD devices.

• Pidora

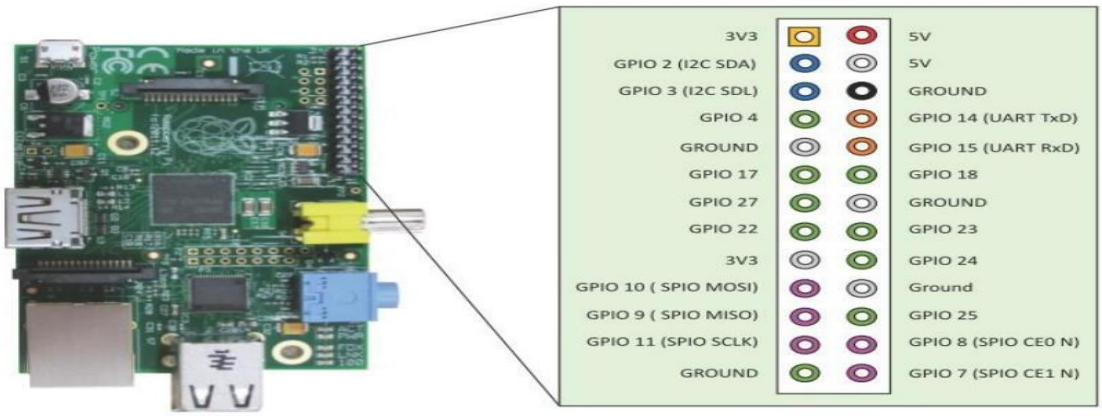
Pidora Linux is a Fedora Linux optimized for Raspberry Pi.

• RaspBMC

RaspBMC is an XBMC media-center distribution for Raspberry Pi.

• OpenELEC

OpenELEC is a fast and user-friendly XBMC media-center distribution. • RISC OS • RISC OS is a very fast and compact operating systems.



Raspberry Pi Interfaces

• Serial

The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.

• SPI

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.

• I2C

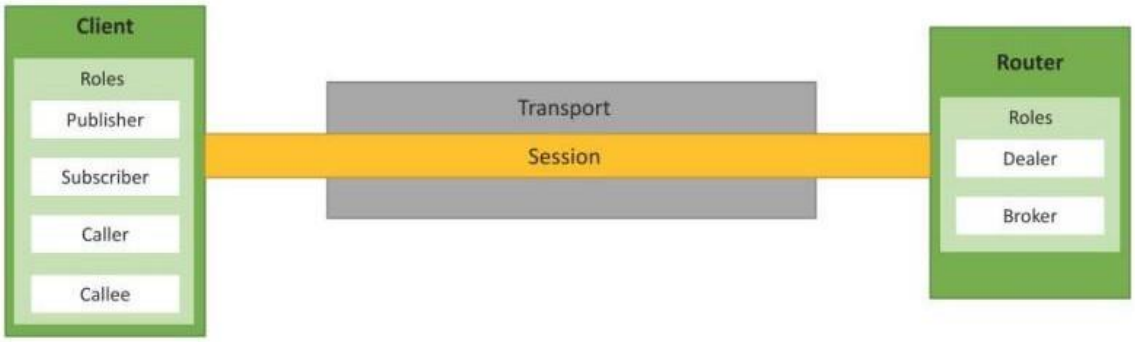
The I2C interface pins on Raspberry Pi allow you to connect hardware modules. I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

UNIT V

• WAMP – Auto Bahn for IoT
• Python for Amazon Web Services
• Python for MapReduce
• Python Packages of Interest
• Python Web Application Framework – Django
• Development with Django.

WAMP for IoT

Web Application Messaging Protocol (WAMP) is a sub-protocol of Websocket which provides publish-subscribe and remote procedure call (RPC) messaging patterns.



WAMP – Concepts

• **Transport:**

Transport is channel that connects two peers.

• **Session:**

Session is a conversation between two peers that runs over a transport.

• **Client:**

Clients are peers that can have one or more roles. In publish-subscribe model client can have following roles:

Publisher:

Publisher publishes events (including payload) to the topic maintained by the Broker.

Subscriber:

Subscriber subscribes to the topics and receives the events including the payload.

In RPC model client can have following roles:

Caller: Caller issues calls to the remote procedures along with call arguments.

Callee: Callee executes the procedures to which the calls are issued by the caller and returns the results back to the caller.

• **Router:** Routers are peers that perform generic call and event routing. In publish-subscribe model Router has the role of a Broker:

Broker: Broker acts as a router and routes messages published to a topic to all subscribers subscribed to the topic.

In RPC model Router has the role of a Broker: –

Dealer: Dealer acts a router and routes RPC calls from the Caller to the Callee and routes results from Callee to Caller.

• **Application Code:** Application code runs on the Clients.

Amazon EC2 – Python Example

• **AutoScaling Service**

• A connection to AutoScaling service is first established by calling boto.ec2.autoscale.connect_to_region function.

• **Launch Configuration**

• After connecting to AutoScaling service, a new launch configuration is created by calling conn.create_launch_con f iguration.

Launch configuration contains instructions on how to launch new instances including the AMI-ID, instance type, security groups, etc.

• **AutoScaling Group**

After creating a launch configuration, it is then associated with a new AutoScaling group. AutoScaling group is created by calling conn.create_auto_scaling_group.

The settings for AutoScaling group such as the maximum and minimum number of instances in the group, the launch configuration, availability zones, optional load balancer to use with the group.

Example:

```
#Python program for creating an AutoScaling group (code excerpt)
import boto.ec2.autoscale
:
print "Connecting to Autoscaling Service"
 conn = boto.ec2.autoscale.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
print "Creating launch configuration"
 lc = LaunchConfiguration(name='My-Launch-Config-2',
image_id=AMI_ID,
key_name=EC2_KEY_HANDLE,
instance_type=INSTANCE_TYPE,
security_groups = [ SECGROUP_HANDLE, ])
```

```
conn.create_launch_configuration(lc)
 print "Creating auto-scaling group"
 ag = AutoScalingGroup(group_name='My-Group',
availability_zones=['us-east-1b'],
launch_config=lc, min_size=1, max_size=2,
connection=conn) conn.create_auto_scaling_group(ag)
```

Amazon AutoScaling – Python Ex

• AutoScaling Policies
• After creating an AutoScaling group, the policies for scaling up and scaling down are defined.
• In this example, a scale up policy with adjustment type ChangeInCapacity and scaling_ad justment = 1 is defined.
• Similarly a scale down policy with adjustment type ChangeInCapacity and scaling_ad justment = -1 is defined.

Example:

```
#Creating auto-scaling policies
scale_up_policy = ScalingPolicy(name='scale_up',
adjustment_type='ChangeInCapacity',
as_name='My-Group',
scaling_adjustment=1,
cooldown=180)
 scale_down_policy = ScalingPolicy(name='scale_down',
 adjustment_type='ChangeInCapacity',
as_name='My-Group',
scaling_adjustment=-1,
 cooldown=180) conn.create_scaling_policy(scale_up_policy)
conn.create_scaling_policy(scale_down_policy)
```

Amazon AutoScaling – Python Exa

• CloudWatch Alarms

• With the scaling policies defined, the next step is to create Amazon CloudWatch alarms that trigger these policies.

• The scale up alarm is defined using the CPUUtilization metric with the Average statistic and threshold greater 70% for a period of 60 sec.

The scale up policy created previously is associated with this alarm. This alarm is triggered when the average CPU utilization of the instances in the group becomes greater than 70% for more than 60 seconds.

• The scale down alarm is defined in a similar manner with a threshold less than 50%.

```
#Connecting to CloudWatch
cloudwatch = boto.ec2.cloudwatch.connect_to_region(REGION,
 aws_access_key_id=ACCESS_KEY,
 aws_secret_access_key=SECRET_KEY)
 alarm_dimensions = {"AutoScalingGroupName": 'My-Group'}
 #Creating scale-up alarm
scale_up_alarm = MetricAlarm( name='scale_up_on_cpu', namespace='AWS/EC2',
metric='CPUUtilization', statistic='Average',
comparison='>', threshold='70',
```

```
period='60', evaluation_periods=2,
alarm_actions=[scale_up_policy.policy_arn],
dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_up_alarm)
#Creating scale-down alarm
 scale_down_alarm = MetricAlarm( name='scale_down_on_cpu', namespace='AWS/EC2',
metric='CPUUtilization', statistic='Average',
comparison='<', threshold='40',
period='60', evaluation_periods=2,
alarm_actions=[scale_down_policy.policy_arn],
dimensions=alarm_dimensions)
cloudwatch.create_alarm(scale_down_alarm)
```

Amazon S3 – Python

• In this example, a connection to S3 service is first established by calling boto.connect_s3 function. • The upload_to_s3_bucket_path function uploads the file to the S3 bucket specified at the specified path.

```
# Python program for uploading a file to an S3 bucket
 import boto.s3
 conn = boto.connect_s3(aws_access_key_id='',
aws_secret_access_key='')
def percent_cb(complete, total):
 print ('.')
def upload_to_s3_bucket_path(bucketname, path, filename):
mybucket = conn.get_bucket(bucketname)
fullkeyname=os.path.join(path,filename)
key = mybucket.new_key(fullkeyname)
 key.set_contents_from_filename(filename,  cb=percent_cb,  num_cb=10)
```

Amazon RDS – Python Exa

• In this example, a connection to RDS service is first established by calling boto.rds.connect_to_region function.

• The RDS region, AWS access key and AWS secret key are passed to this function.

• After connecting to RDS service, the conn.create_dbinstance function is called to launch a new RDS instance.

• The input parameters to this function include the instance ID, database size, instance type, database username, database password, database port, database engine (e.g. MySQL5.1), database name, security groups.

Example

```
#Python program for launching an RDS instance (excerpt)
import boto.rds
ACCESS_KEY="<enter>"
SECRET_KEY="<enter>"
REGION="us-east-1"
INSTANCE_TYPE="db.t1.micro"
ID = "MySQL-db-instance-3"
USERNAME = 'root'
```

```
PASSWORD = 'password'
DB_PORT = 3306
DB_SIZE = 5
DB_ENGINE = 'MySQL5.1'
DB_NAME = 'mytestdb'
SECGROUP_HANDLE="default"
#Connecting to RDS
conn = boto.rds.connect_to_region(REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
#Creating an RDS instance
db = conn.create_dbinstance(ID, DB_SIZE, INSTANCE_TYPE, USERNAME, PASSWORD,
port=DB_PORT, engine=DB_ENGINE, db_name=DB_NAME, security_groups = [
SECGROUP_HANDLE, ] )
```

Python for Map

• The example shows inverted index mapper program.

• The map function reads the data from the standard input (stdin) and splits the tab-limited data into document-ID and contents of the document.

• The map function emits key-value pairs where key is each word in the document and value is the document-ID.

Example

```
#Inverted Index Mapper in Python
 #!/usr/bin/env python
 import sys
for line in sys.stdin:
doc_id, content = line.split(’’)
words = content.split()
for word in words:
 print ’%s%s’ % (word, doc_id)
```

Python for MapReduce

• The example shows inverted index reducer program.

• The key-value pairs emitted by the map phase are shuffled to the reducers and grouped by the key.

• The reducer reads the key-value pairs grouped by the same key from the standard input (stdin) and creates a list of document-IDs in which the word occurs.

• The output of reducer contains key value pairs where key is a unique word and value is the list of document-IDs in which the word occurs.

Example

```
#Inverted Index Reducer in Python
 #!/usr/bin/env python
 import sys
current_word = None
current_docids = []
word = None
for line in sys.stdin:
```

```
# remove leading and trailing whitespace
line = line.strip()

# parse the input we got from mapper.py
word, doc_id = line.split('')
if current_word == word:
current_docids.append(doc_id)
else:
if current_word:
print '%s%s' % (current_word, current_docids)
current_docids = []
current_docids.append(doc_id)
current_word = word
```

Python Packages of Interest

• **JSON**

JavaScript Object Notation (JSON) is an easy to read and write data-interchange format. JSON is used as an alternative to XML and is is easy for machines to parse and generate.

JSON is built on two structures - a collection of name-value pairs (e.g. a Python dictionary) and ordered lists of values (e.g.. a Python list).

• **XML**

XML (Extensible Markup Language) is a data format for structured document interchange.

The Python minidom library provides a minimal implementation of the Document Object Model interface and has an API similar to that in other languages.

• **HTTPLib & URLLib**

HTTPLib2 and URLLib2 are Python libraries used in network/internet programming

• **SMTPLib**

Simple Mail Transfer Protocol (SMTP) is a protocol which handles sending email and routing e-mail between mail servers. The Python smtplib module provides an SMTP client session object that can be used to send email.

• **NumPy**

NumPy is a package for scientific computing in Python. NumPy provides support for large multi-dimensional arrays and matrices

• **Scikit-learn**

Scikit-learn is an open source machine learning library for Python that provides implementations of various machine learning algorithms for classification, clustering, regression and dimension reduction problems.

Python Web Application Framework

Django is an open source web application framework for developing web applications in Python.

• A web application framework in general is a collection of solutions, packages and best practices that allows development of web applications and dynamic websites.

• Django is based on the Model-Template-View architecture and provides a separation of the data model from the business rules and the user interface.

• Django provides a unified API to a database backend.
• Thus web applications built with Django can work with different databases without requiring any code changes.

• With this fiexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem, Django is best suited for cloud applications.

• Django consists of an object-relational mapper, a web templating system and a regular-expression-based URL dispatcher.

Django Architecture

Django is Model-Template-View (MTV) framework.

• **Model**

The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc.

A Django model is a Python class that outlines the variables and methods for a particular type of data.

• **Template**

In a typical Django web application, the template is simply an HTML page with a few extra placeholders.

Django's template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV, etc.)

• **View**

The view ties the model to the template. The view is where you write the code that actually generates the web pages. View determines what data is to be displayed, retrieves the data from the database and passes the data to the template.

Text Book(s)	Reference Book
Internet of Things - A hands on Approach Authors: Arshdeep Bahga, Vijay Madisetti Publisher: Universities press.	Internet of Things - Srinivasa K.G., Siddesh G.M. Hanumantha Raju R. Publisher: Cengage Learning India pvt. Ltd (2018).

Prepared by

R.Nagadevi,
Assistant Professor,
Department of BCA,
Vidyasagar College of arts and science,
Udumalpet.

Reference Website :

www.studoc.com