Internet of Things

INFORMATION TECHNOLOGY

V Semester (R16)

Prepared by
Dr. C Santhaiah, Professor, CSE
Ms. N M Deepika, Assistant Professor, CSE
Ms. G Nishwitha, Assistant Professor, CSE

UNIT- I
Introduction of Internet of Things

Introduction

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 environments.

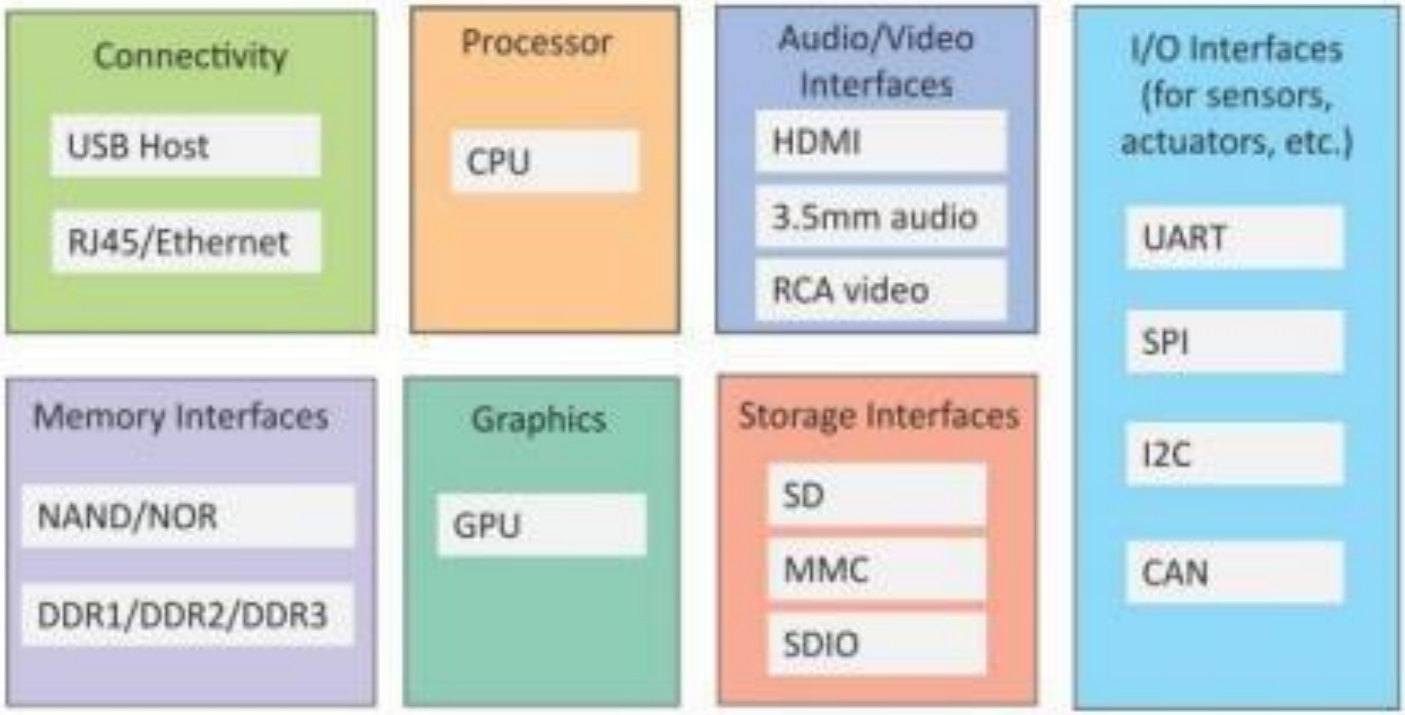
CHARACTERISTICS OF IOT

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

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 constraints.

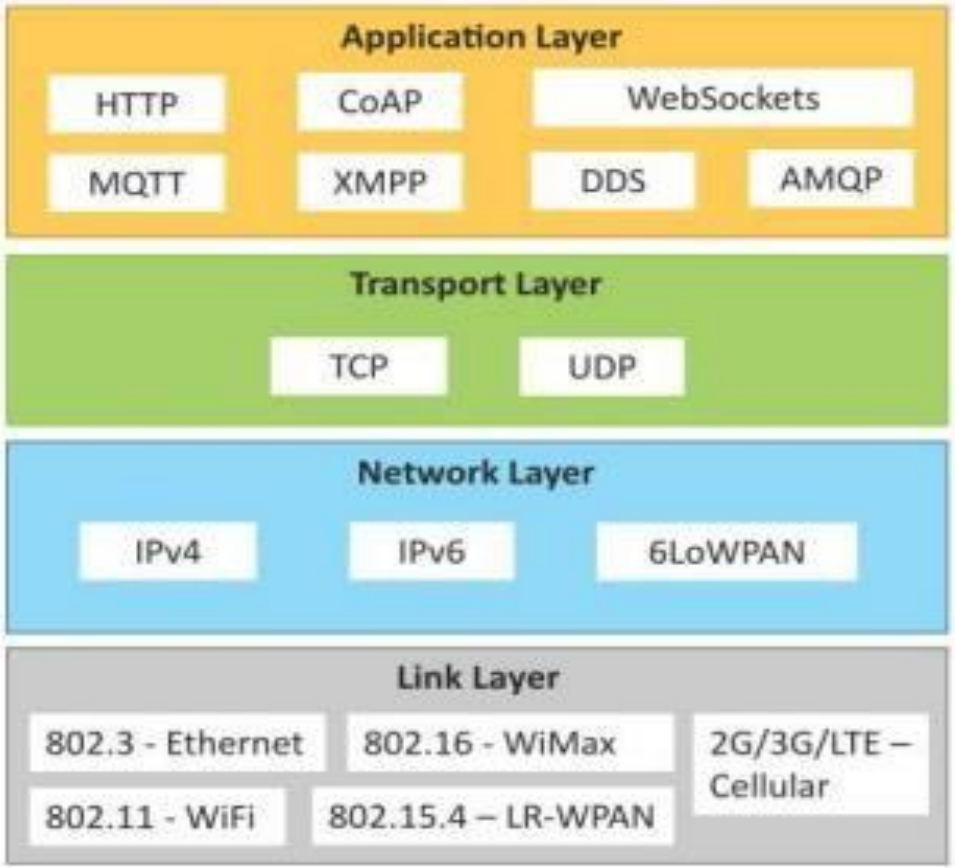
GENERIC BLOCK DIAGRAM OF AN IOT DEVICE



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 interfaces.

IoT Protocols



IoT Protocols

1. **Link Layer**
   - 802.3 – Ethernet
   - 802.11 – WiFi
   - 802.16 – WiMax
   - 802.15.4 – LR-WPAN
   - 2G/3G/4G

IoT Protocols

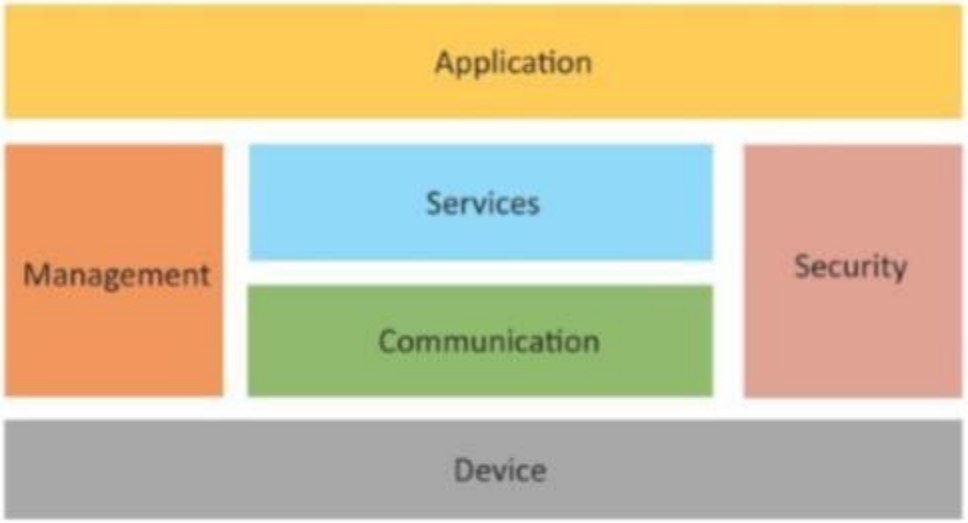
2. **Network/Internet Layer**
   - IPv4
   - IPv6
   - 6LoWPAN
3. **Transport Layer**
   - TCP
   - UDP

IoT Protocols

4. Application Layer

- HTTP
- CoAP
- WebSocket
- MQTT
- XMPP
- DDS
- AMQP

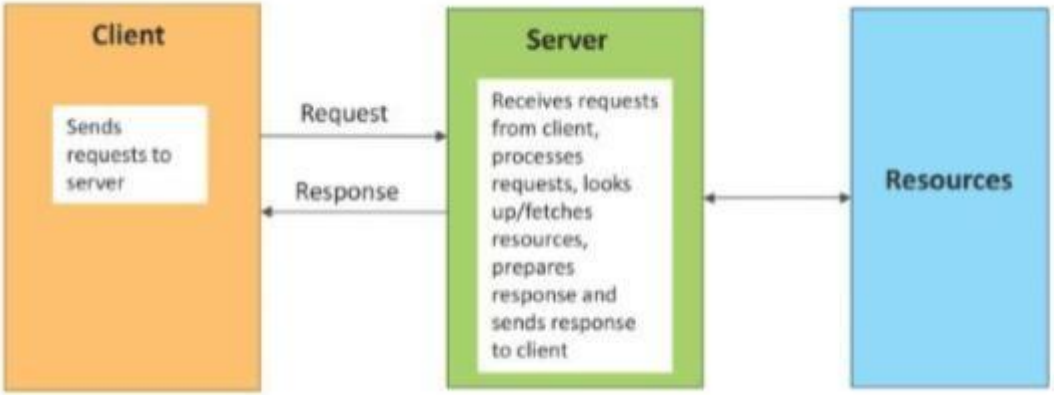
LOGICAL DESIGN OF IoT



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.

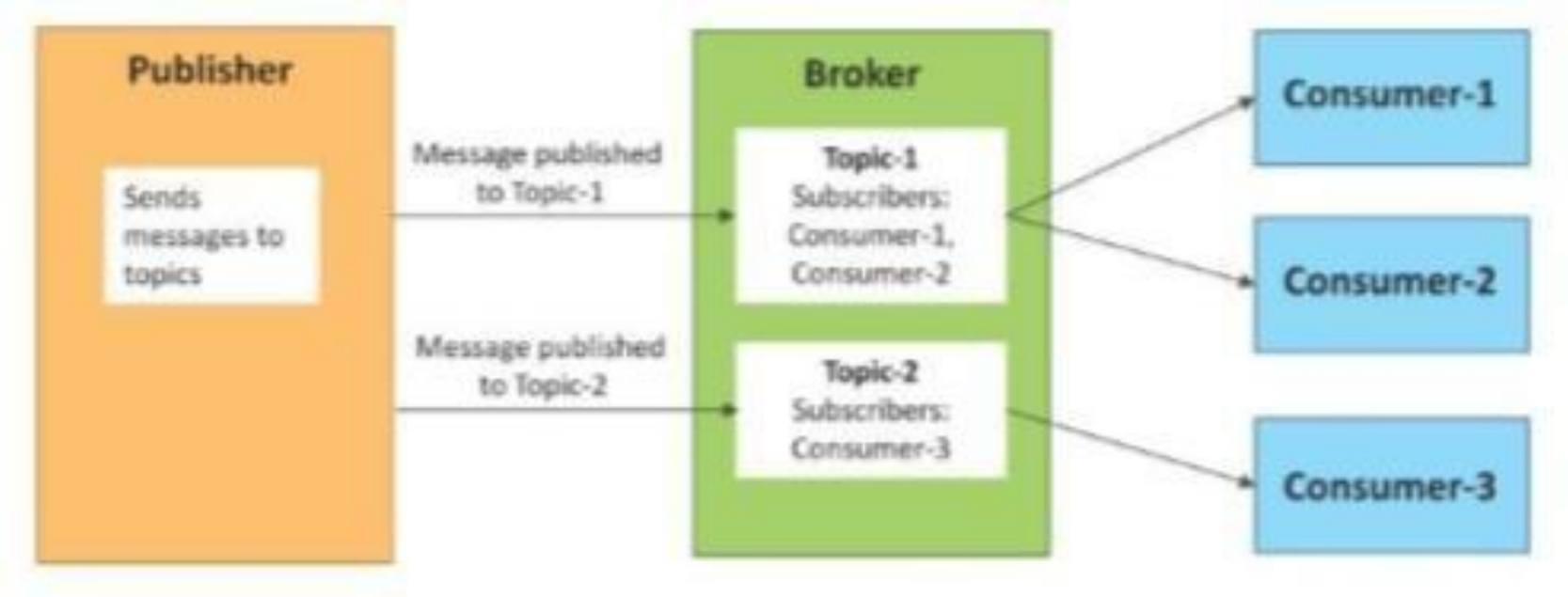
REQUEST-RESPONSE COMMUNICATION MODEL



REQUEST-RESPONSE 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.

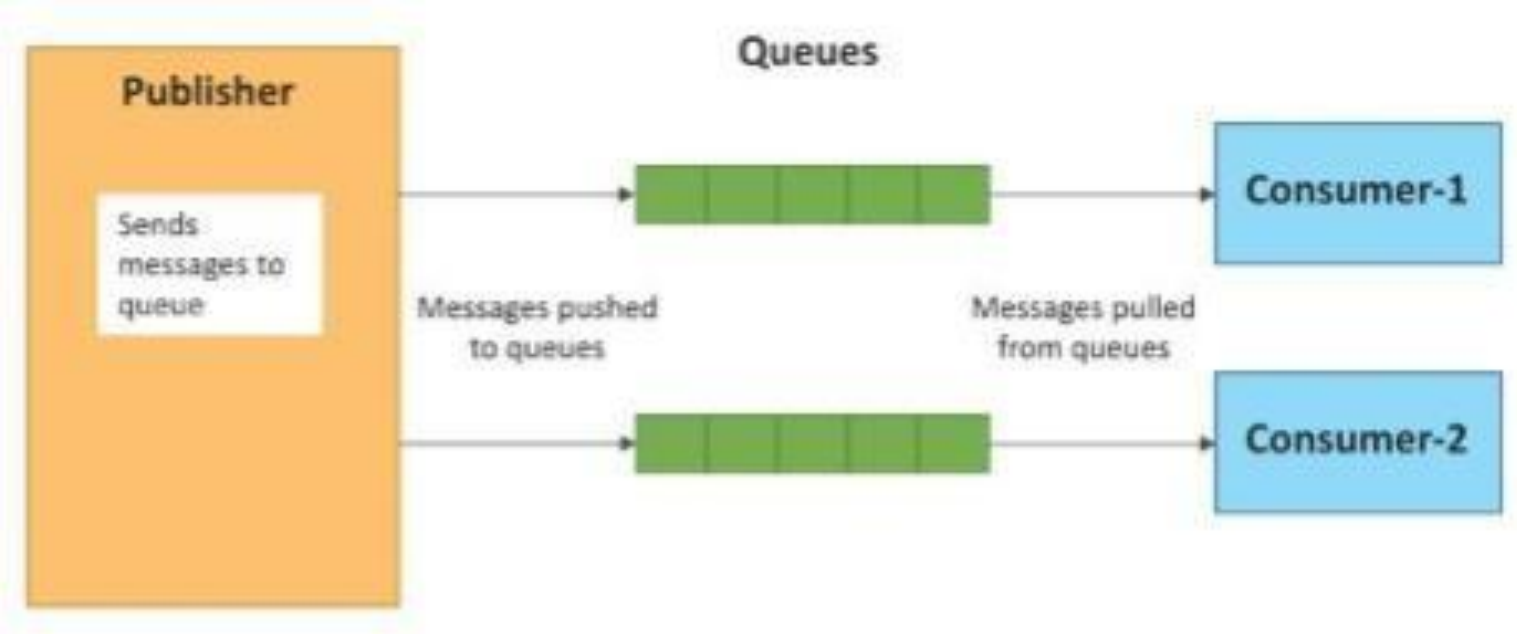
PUBLISH-SUBSCRIBE COMMUNICATION MODEL



PUBLISH-SUBSCRIBE 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.

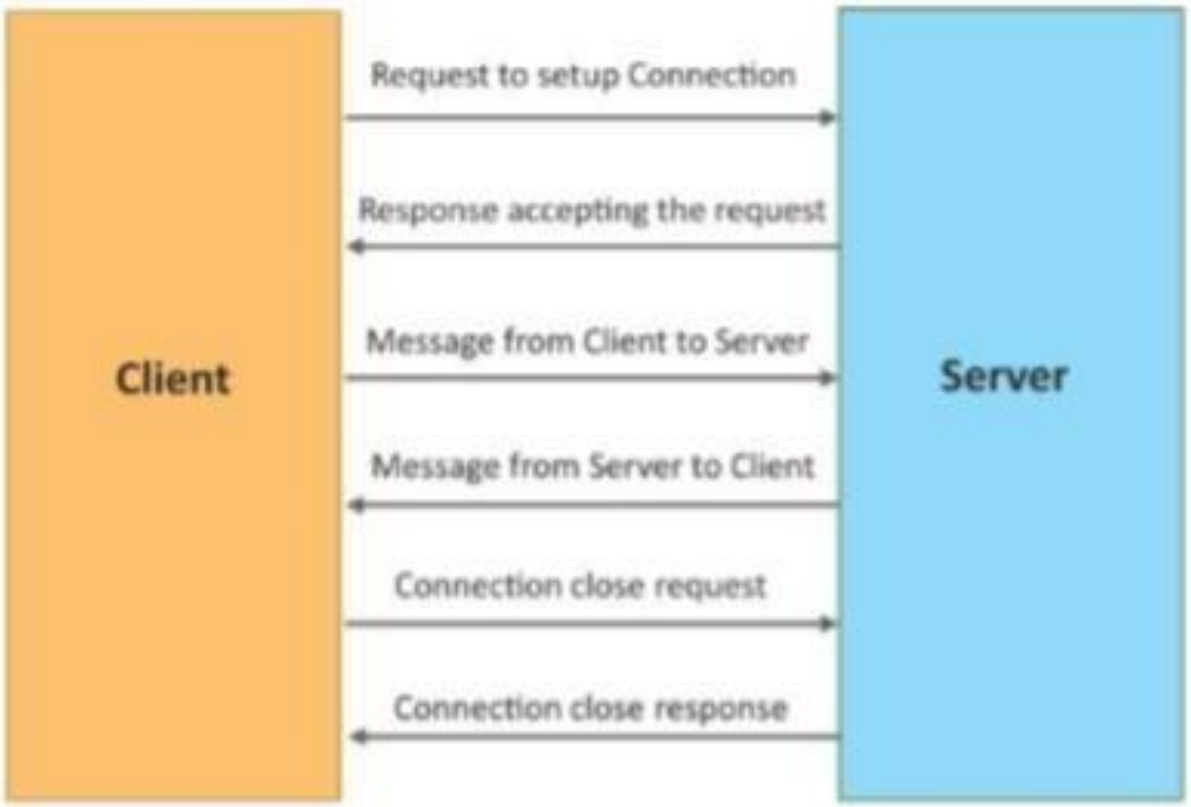
PUSH-PULL COMMUNICATION MODEL



PUSH-PULL 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 at which the consumers pull data.

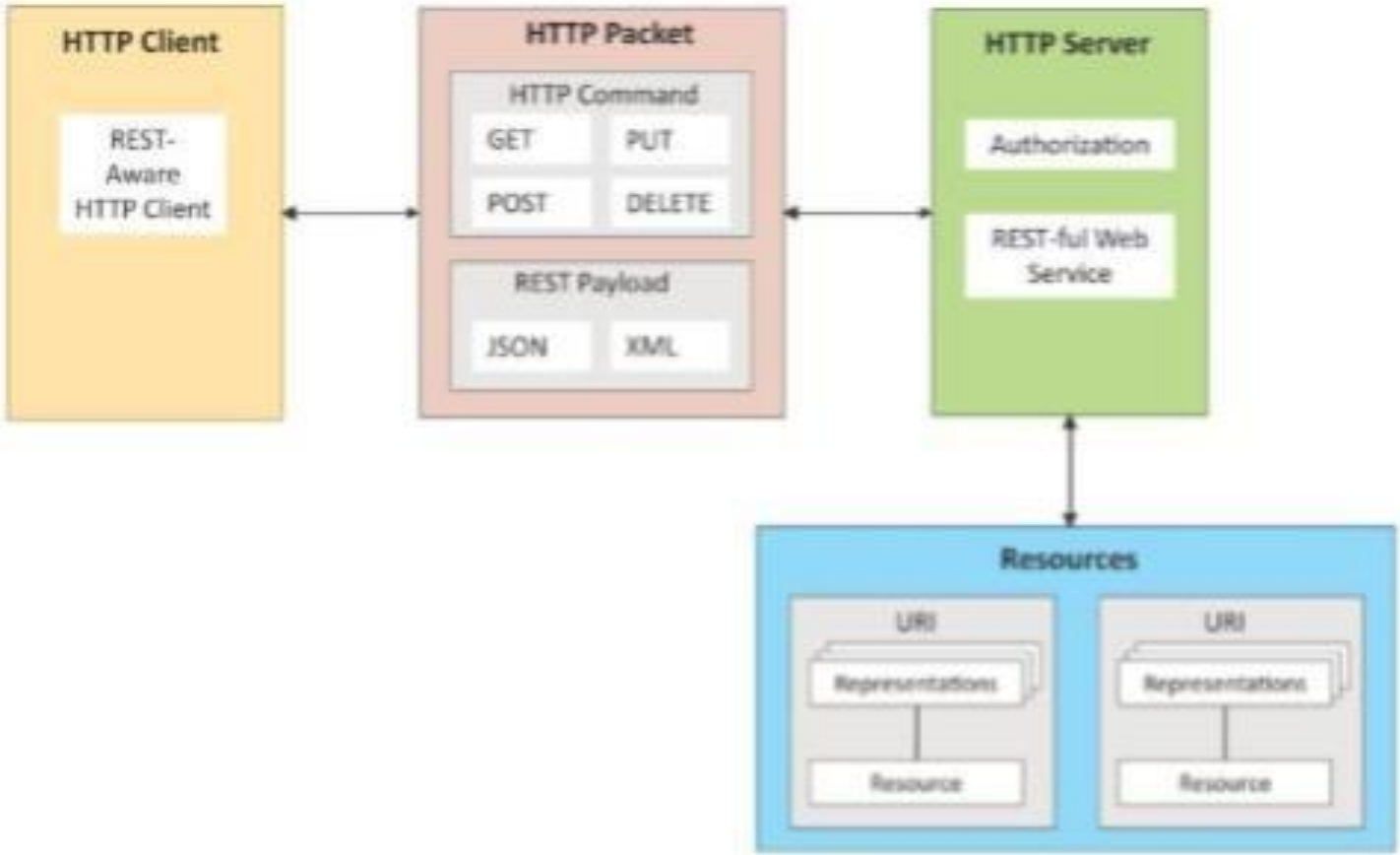
EXCLUSIVE PAIR COMMUNICATION MODEL



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.

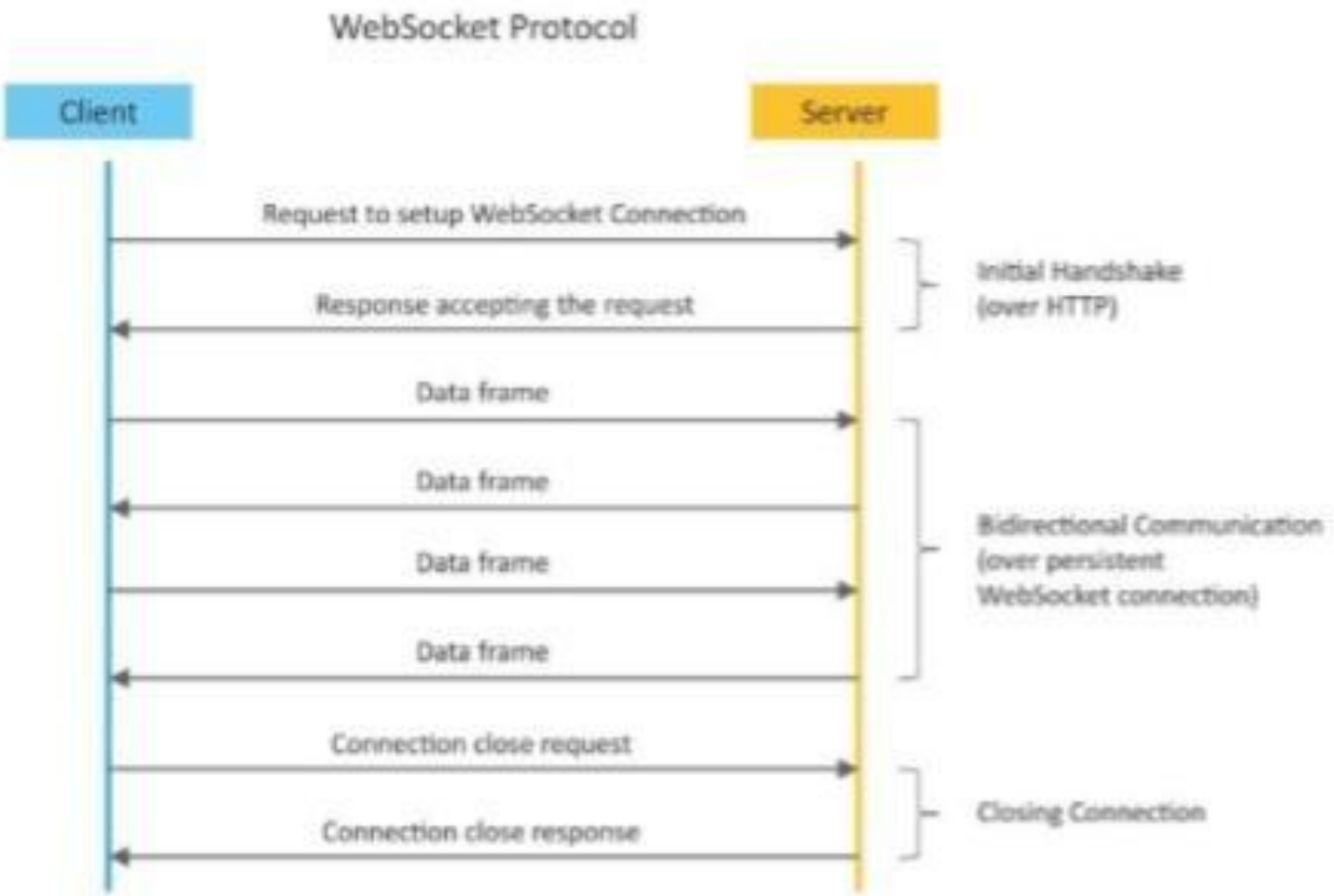
REST-based Communication APIs



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.

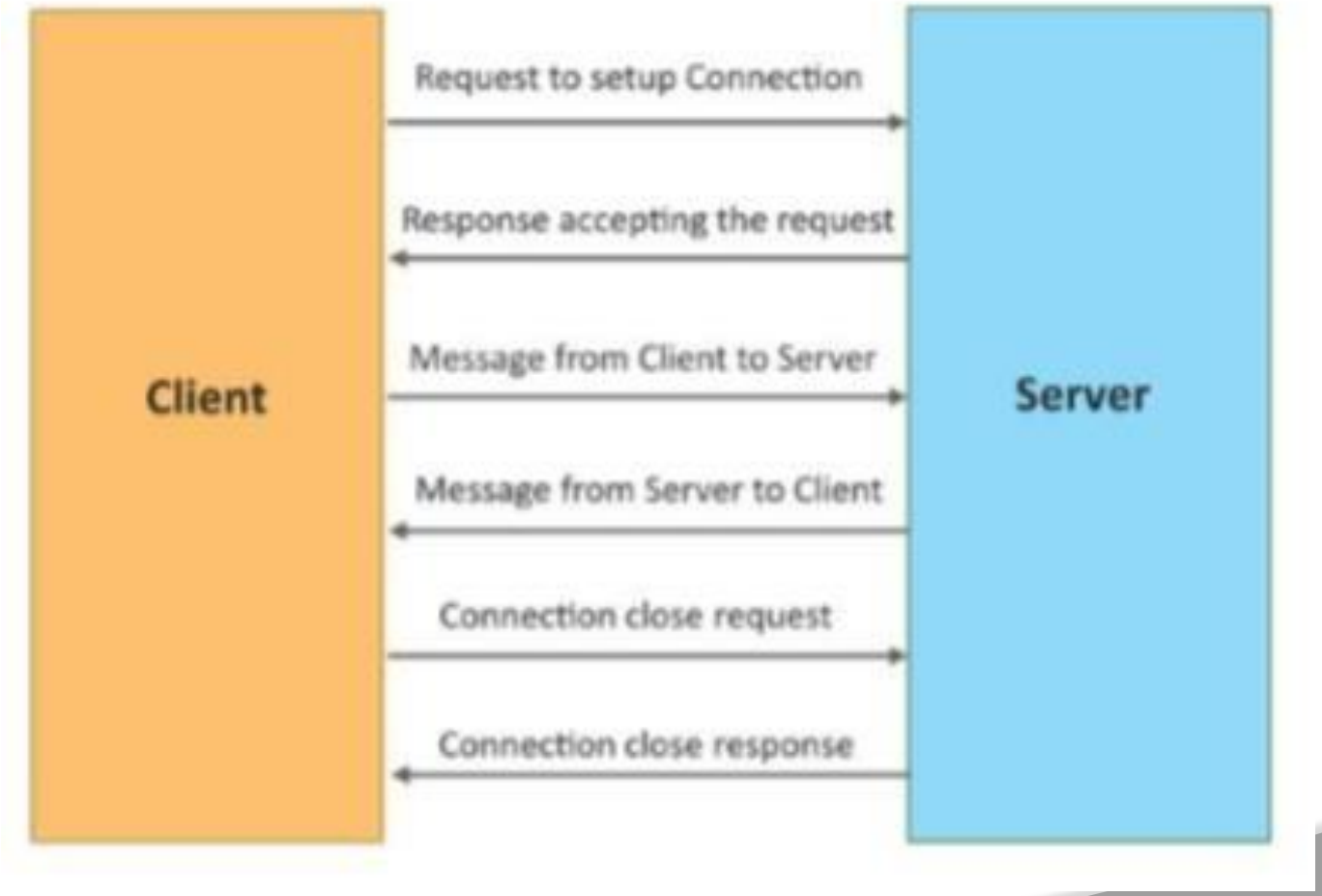
WebSocket-based Communication APIs



WebSocket-based Communication APIs

- WebSocket APIs allow bidirectional, full duplex communication between clients and servers.
- WebSocket APIs follow the exclusive pair communication model.

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.

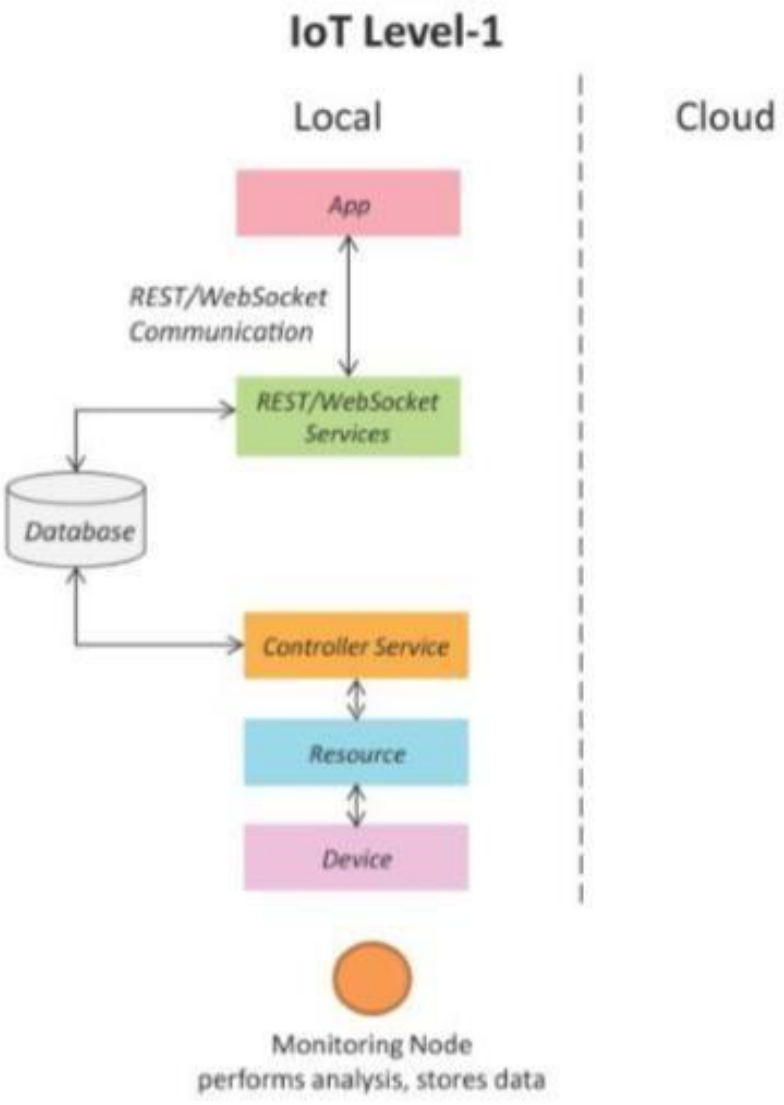
IoT Levels & Deployment Templates

- 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.

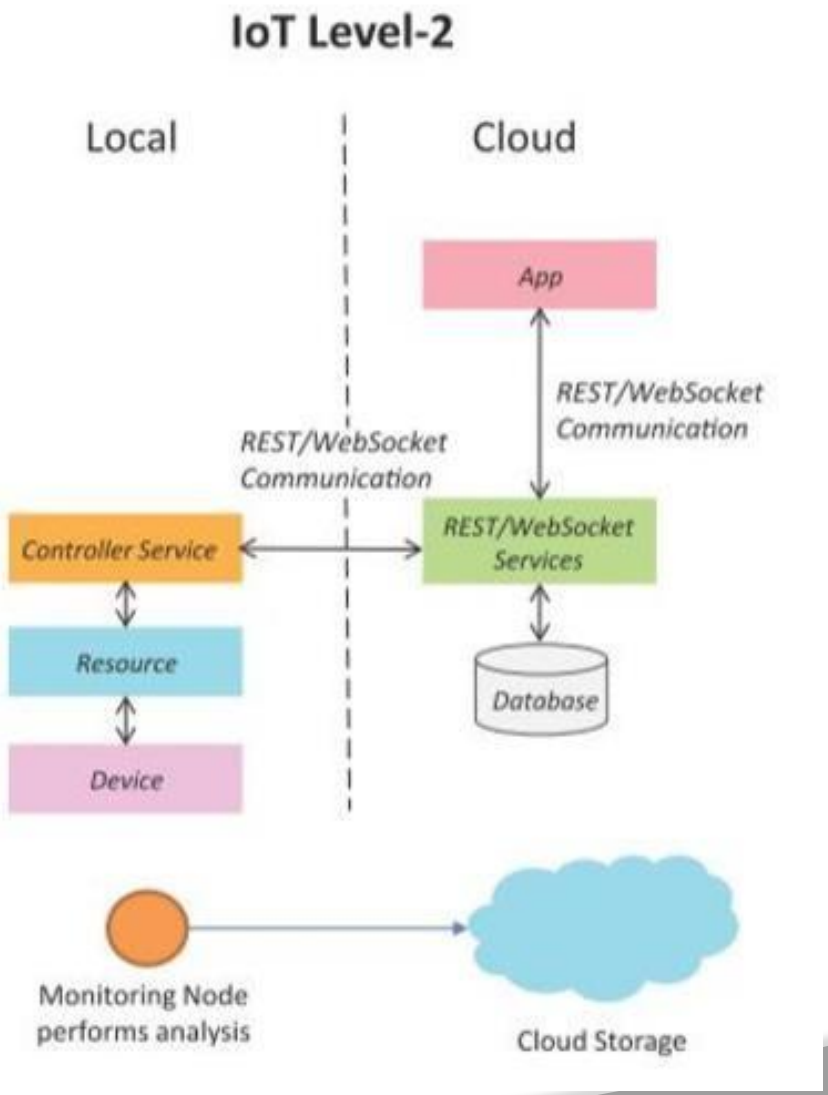
IoT LEVEL-1


IoT Level-1
Local
Cloud
App
REST/WebSocket
Communication
REST/WebSocket
Services
Database
Controller Service
Resource
Device
Monitoring Node
performs analysis, stores data

IoT LEVEL-1

- 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.

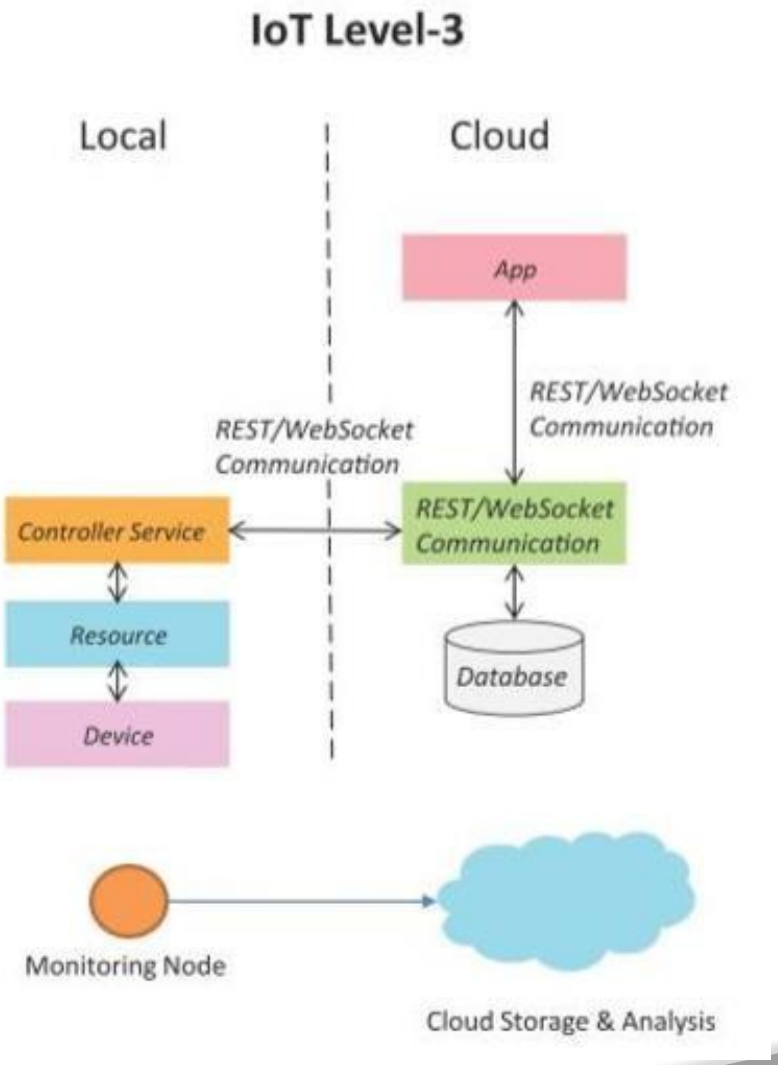
IoT LEVEL-2


IoT Level-2
Local
Cloud
App
REST/WebSocket
Communication
REST/WebSocket
Communication
Controller Service
REST/WebSocket
Services
Resource
Database
Device
Monitoring Node
performs analysis
Cloud Storage

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.

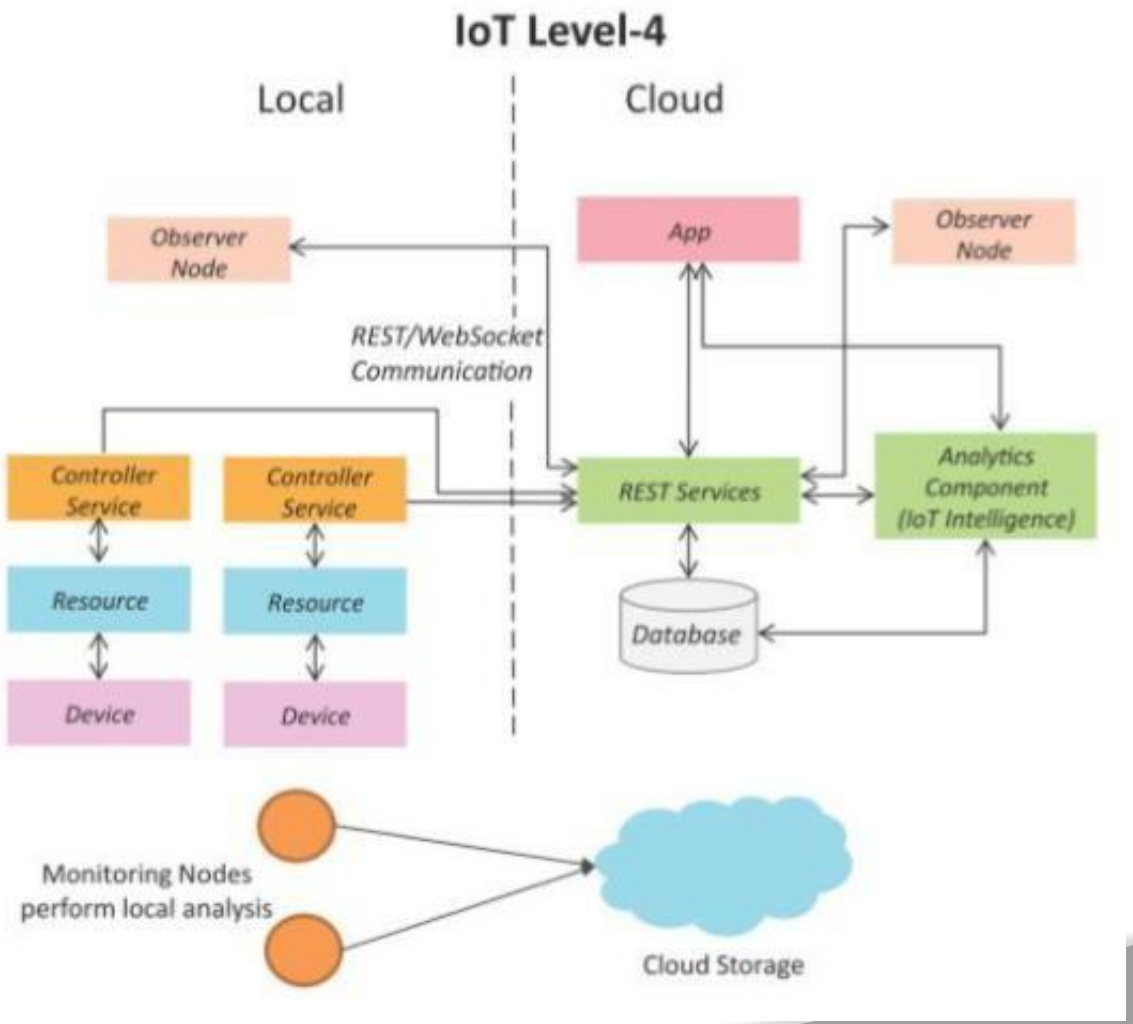
IoT LEVEL-3


IoT Level-3
Local
Cloud
App
REST/WebSocket
Communication
REST/WebSocket
Communication
Controller Service
REST/WebSocket
Communication
Resource
Database
Device
Monitoring Node
Cloud Storage & Analysis

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.

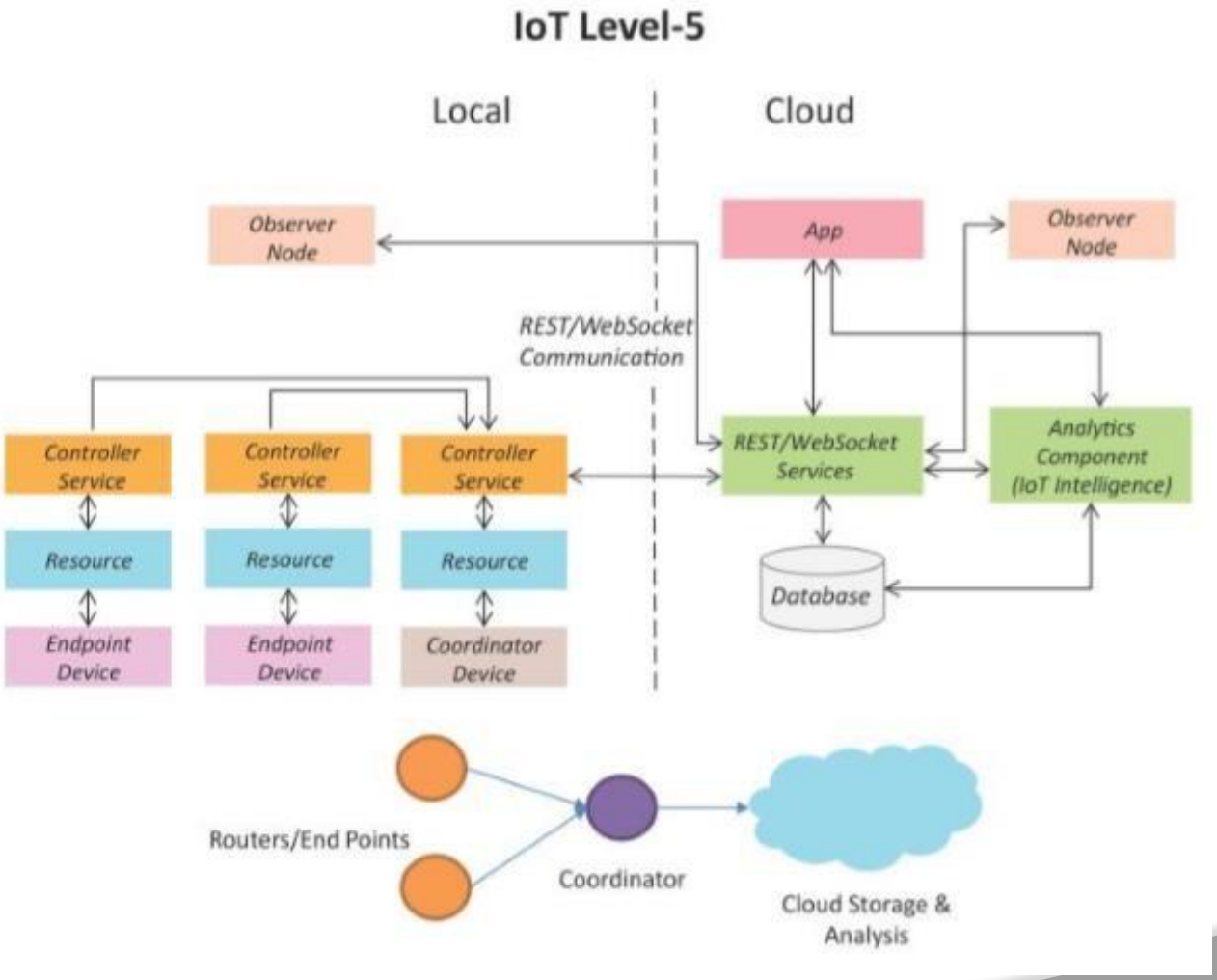
IoT LEVEL-4


IoT Level-4
Local
Cloud
Observer Node
App
Observer Node
REST/WebSocket Communication
Controller Service
Controller Service
REST Services
Analytics Component (IoT Intelligence)
Resource
Resource
Database
Device
Device
Monitoring Nodes perform local analysis
Cloud Storage

IoT LEVEL-3

- 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.

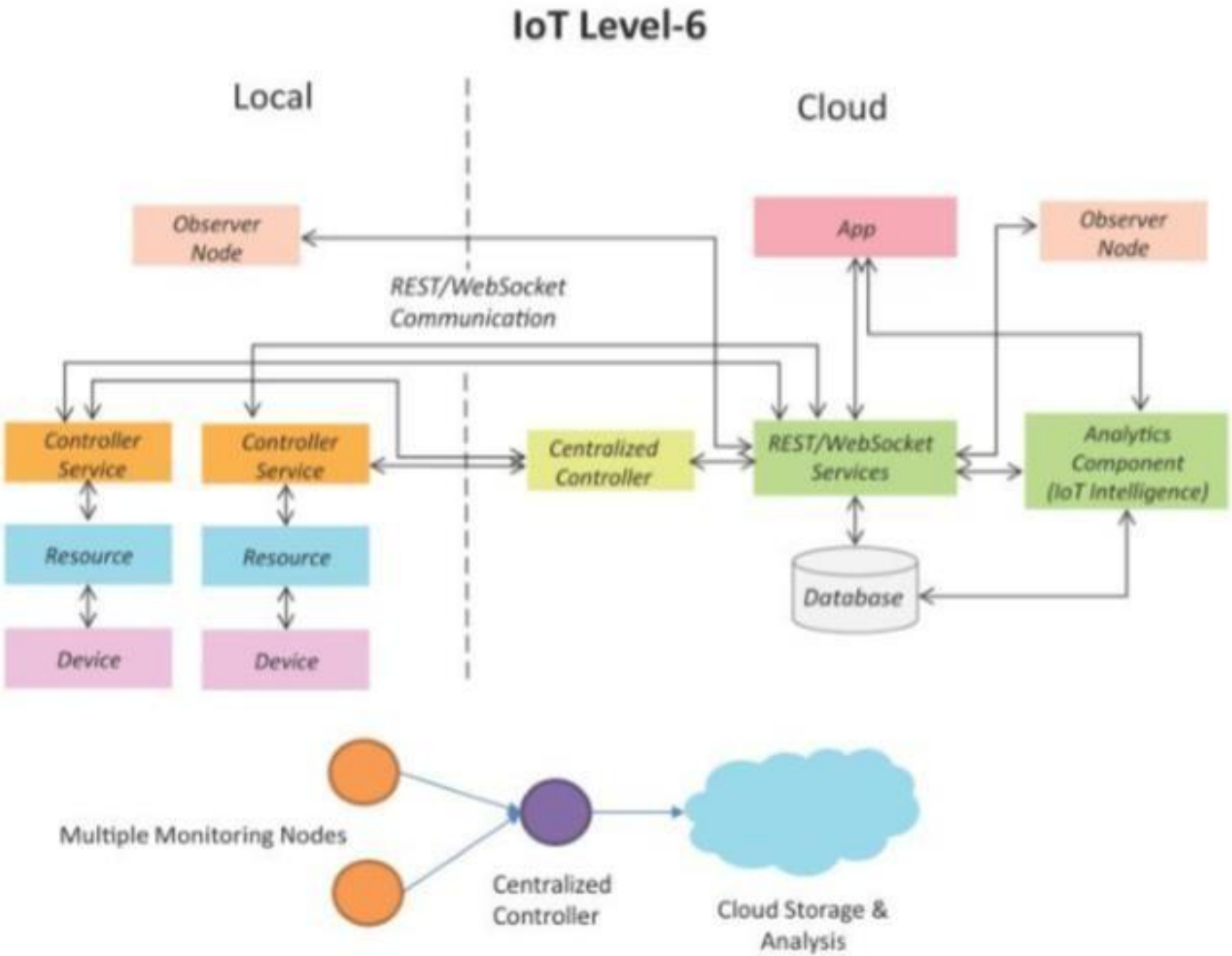
IoT LEVEL-5


IoT Level-5
Local
Cloud
Observer Node
App
Observer Node
REST/WebSocket Communication
Controller Service
Controller Service
Controller Service
REST/WebSocket Services
Analytics Component (IoT Intelligence)
Resource
Resource
Resource
Database
Endpoint Device
Endpoint Device
Coordinator Device
Routers/End Points
Coordinator
Cloud Storage & Analysis

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-5

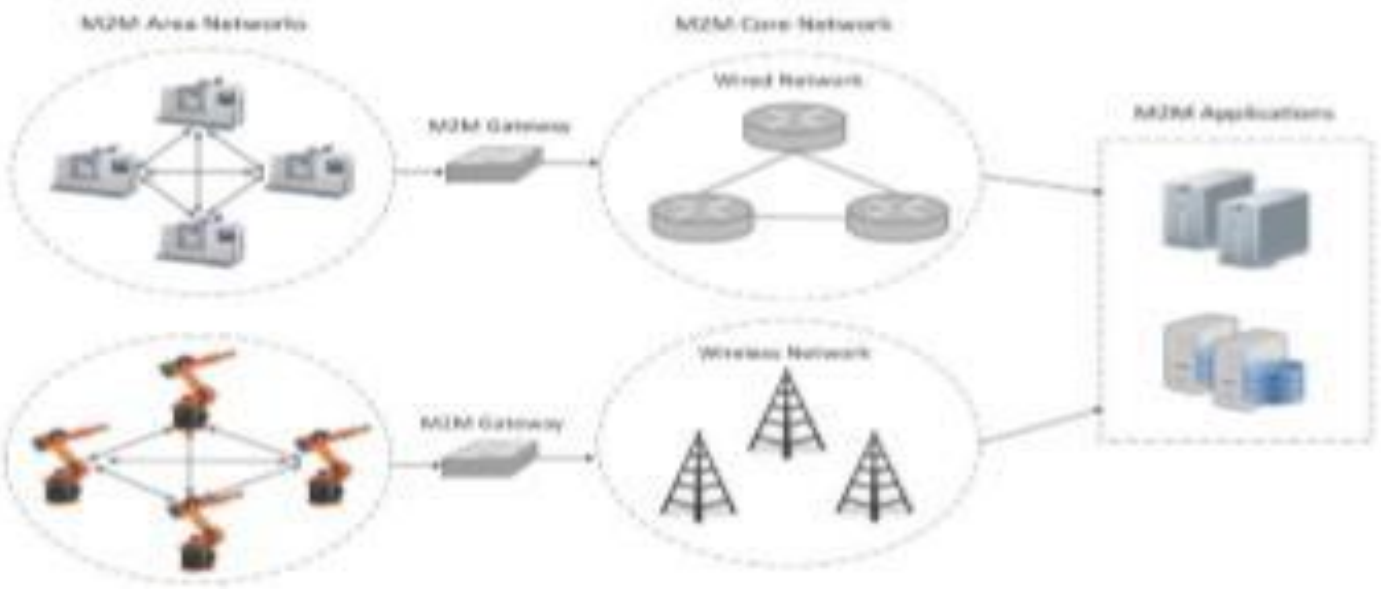

IoT Level-6
Local
Cloud
Observer Node
REST/WebSocket Communication
App
Observer Node
Controller Service
Controller Service
Centralized Controller
REST/WebSocket Services
Analytics Component (IoT Intelligence)
Resource
Resource
Database
Device
Device
Multiple Monitoring Nodes
Centralized Controller
Cloud Storage & Analysis

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.

UNIT- II
IoT AND M2M

Machine-to-Machine (M2M)



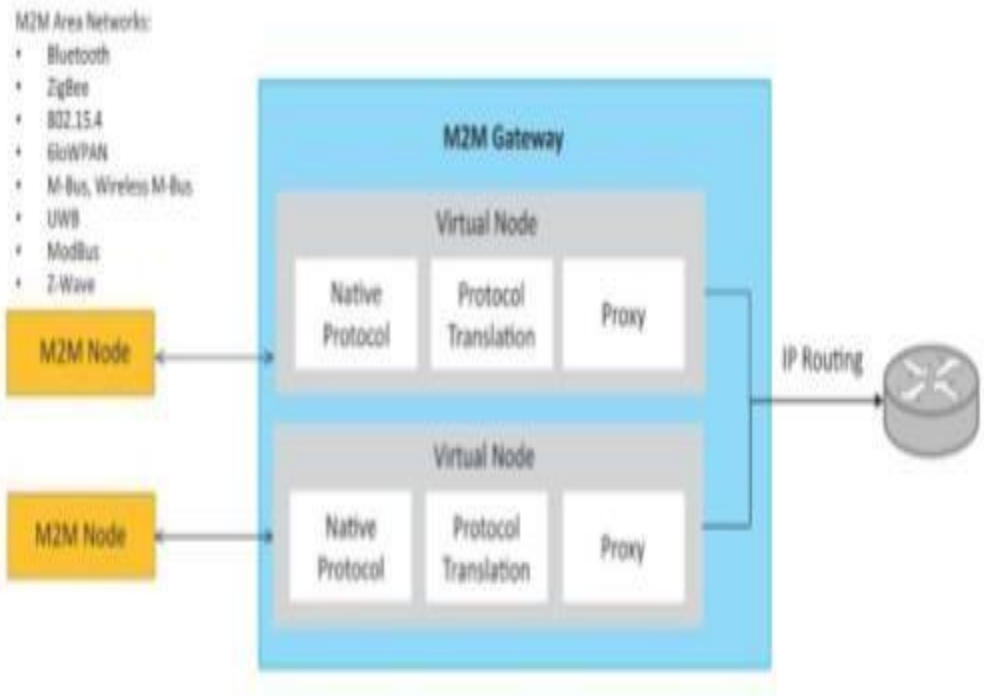
M2M

- 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 (IPbased).
- 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

1. 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.
2. 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

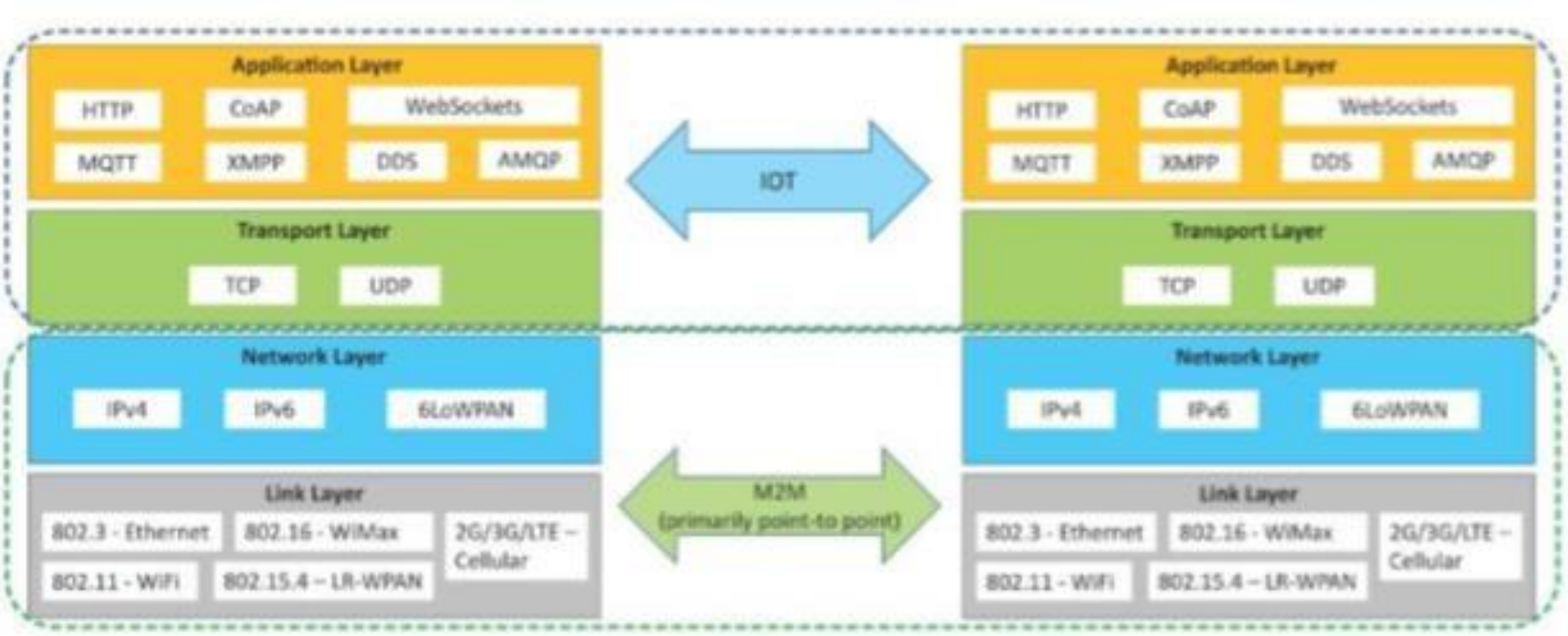
3. Hardware vs Software Emphasis
   - While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.

4. 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).

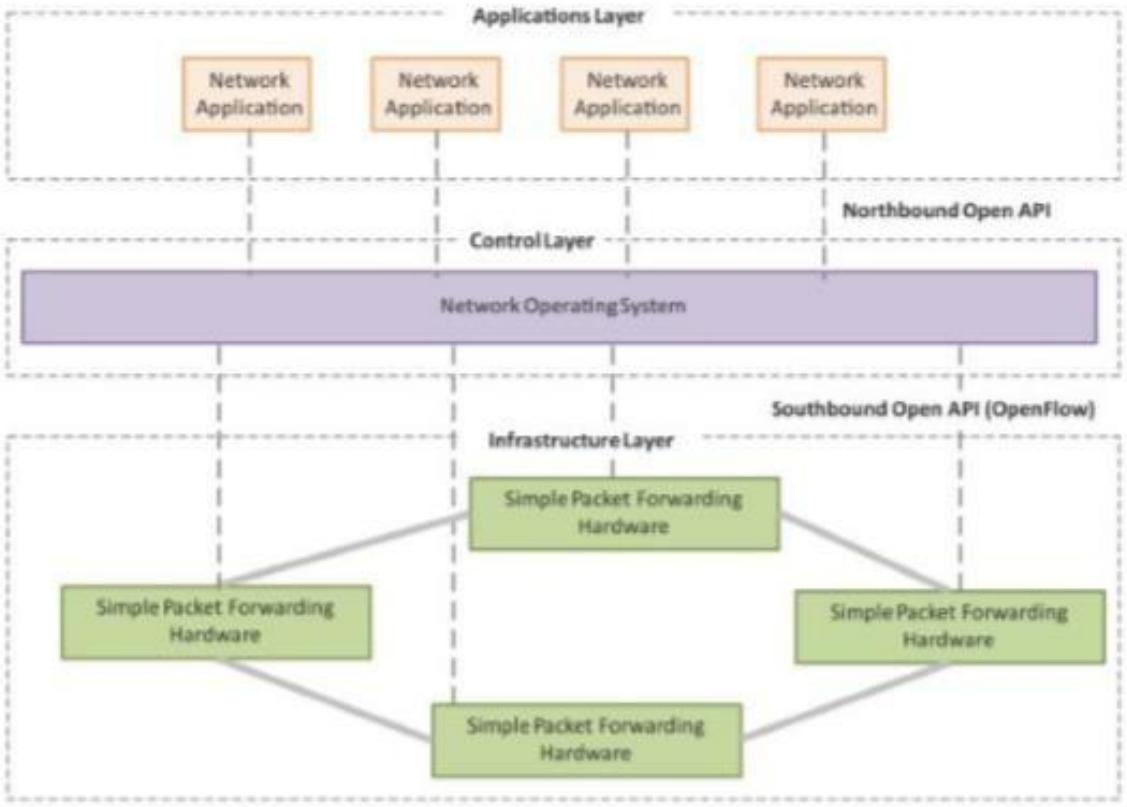
Difference between IoT and M2M

3. 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



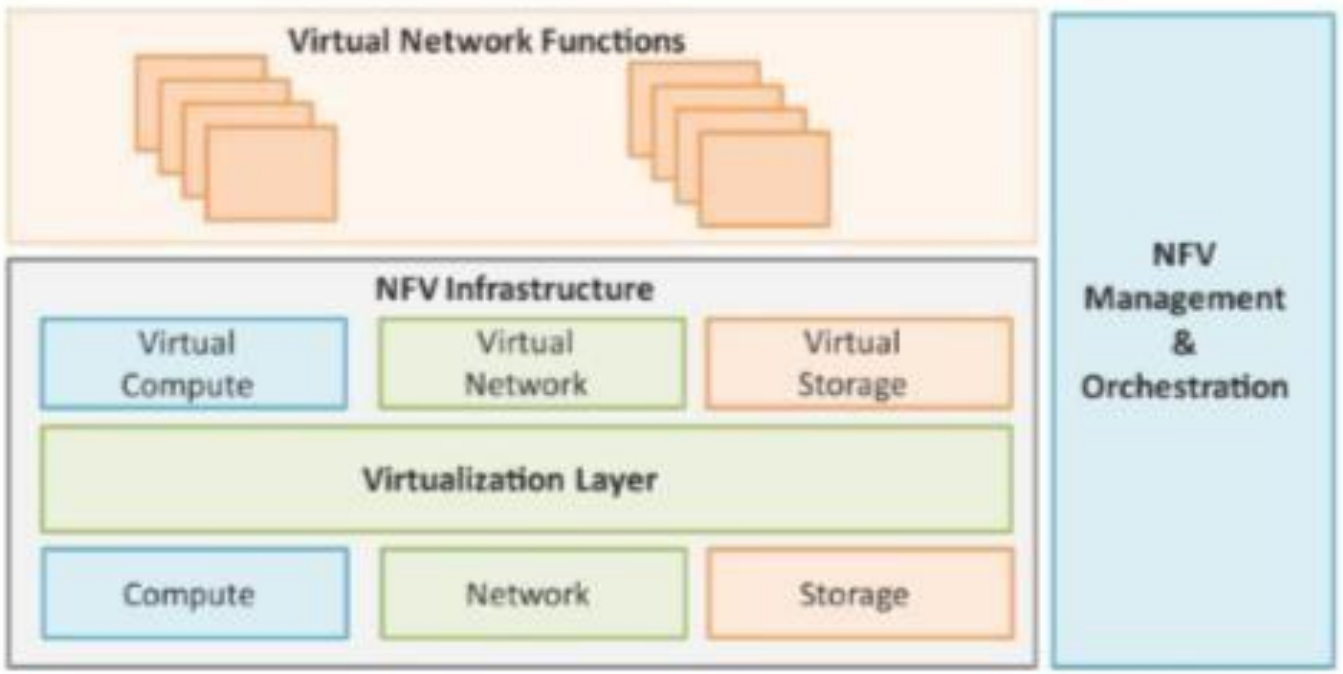
SDN


Applications Layer
Network Application
Network Application
Network Application
Network Application
Northbound Open API
Control Layer
Network Operating System
Southbound Open API (OpenFlow)
Infrastructure Layer
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware

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 configuration, management and provisioning simpler.
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

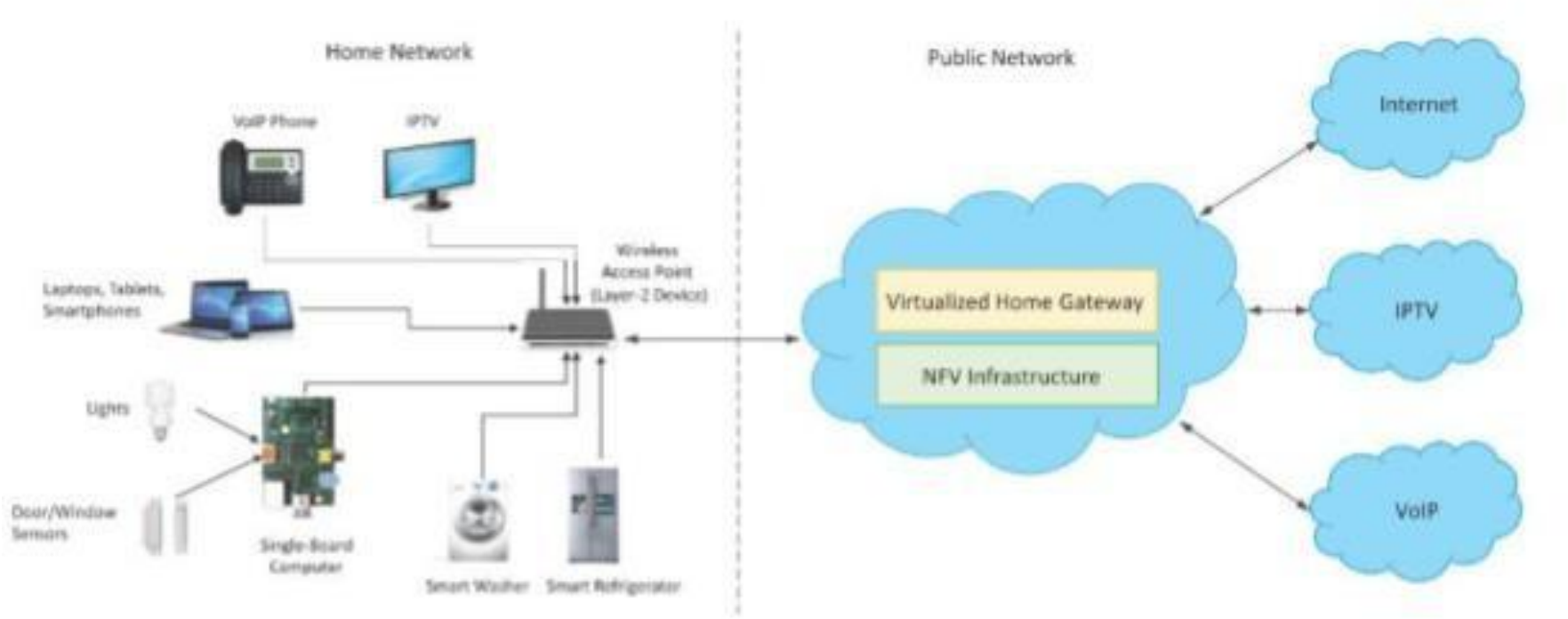
Network Function Virtualization (NFV)



Network Function Virtualization (NFV)

1. Virtualized Network Function (VNF):
   - VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
2. NFV Infrastructure (NFVI):
   - NFVI includes compute, network and storage resources that are virtualized. Standard Communication Interface (OpenFlow)
3. NFV Management and Orchestrat
   - 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


Home Network
Public Network
VoIP Phone
IPTV
Wireless Access Point (Layer-2 Device)
Laptops, Tablets, Smartphones
Lights
Door/Window Sensors
Single-Board Computer
Smart Washer
Smart Refrigerator
Internet
Virtualized Home Gateway
NFV Infrastructure
IPTV
VoIP

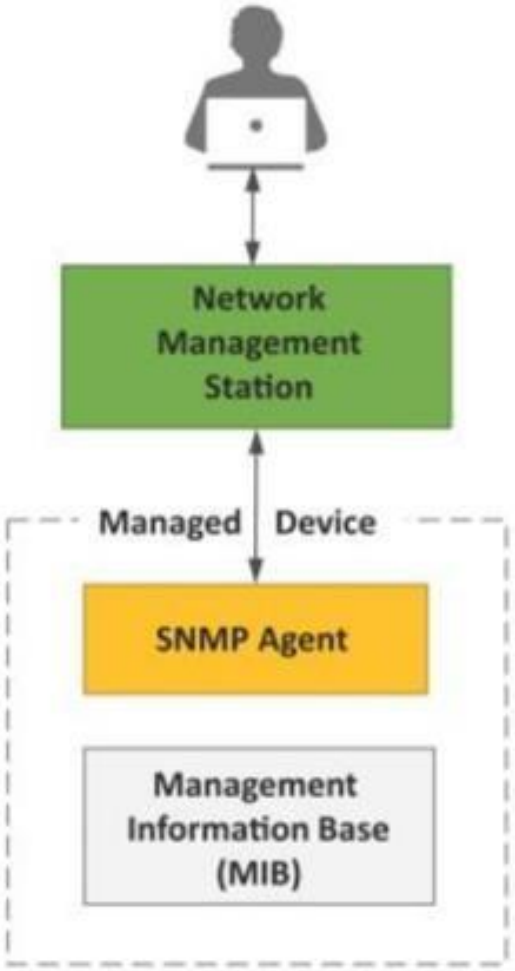
NFV

- 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.

Need for IoT Systems Management

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

NFV Use Case


Network
Management
Station
Managed Device
SNMP Agent
Management
Information Base
(MIB)

Need for IoT Systems Management

- 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.

Network Operator Requirements

- Ease of use
- Distinction between configuration and state data
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration database schemas
- Comparing configurations

YANG

- 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.

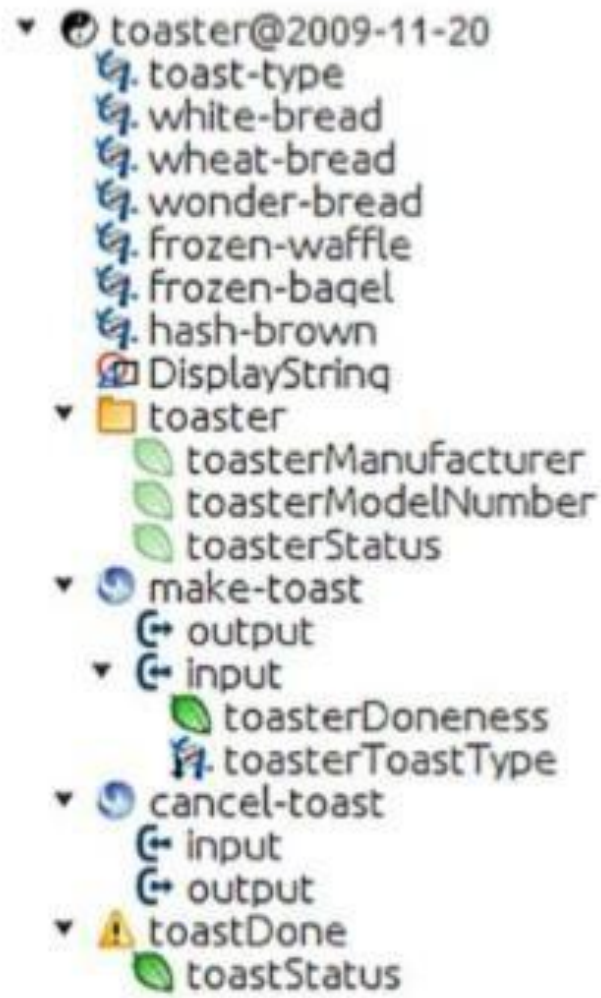
YANG

- 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.

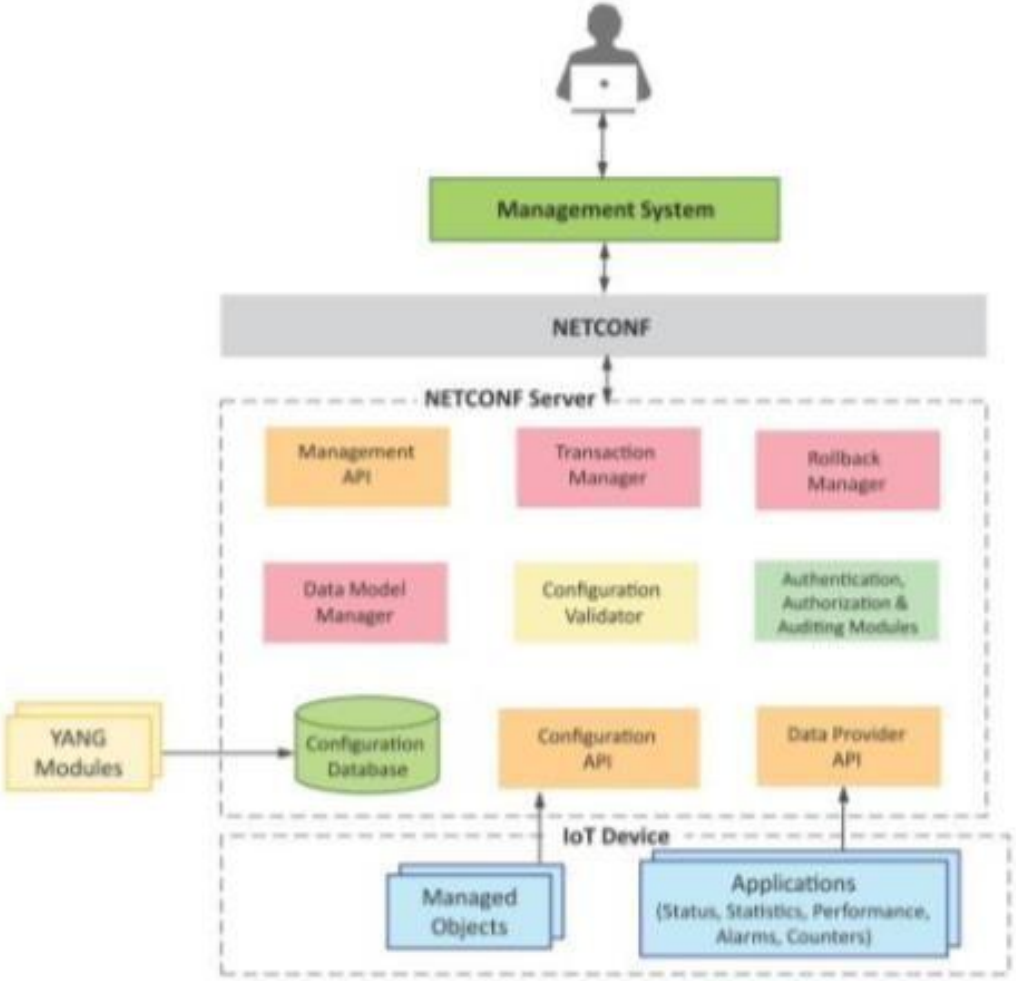
YANG Module Example

- 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’).

YANG Module Example



NETCONF-YANG



IoT Systems Management with NETCONF-YANG

- Management System
- Management API
- Transaction Manager
- Rollback Manager
- Data Model Manager
- Configuration Validator
- Configuration Database
- Configuration API
- Data Provider API

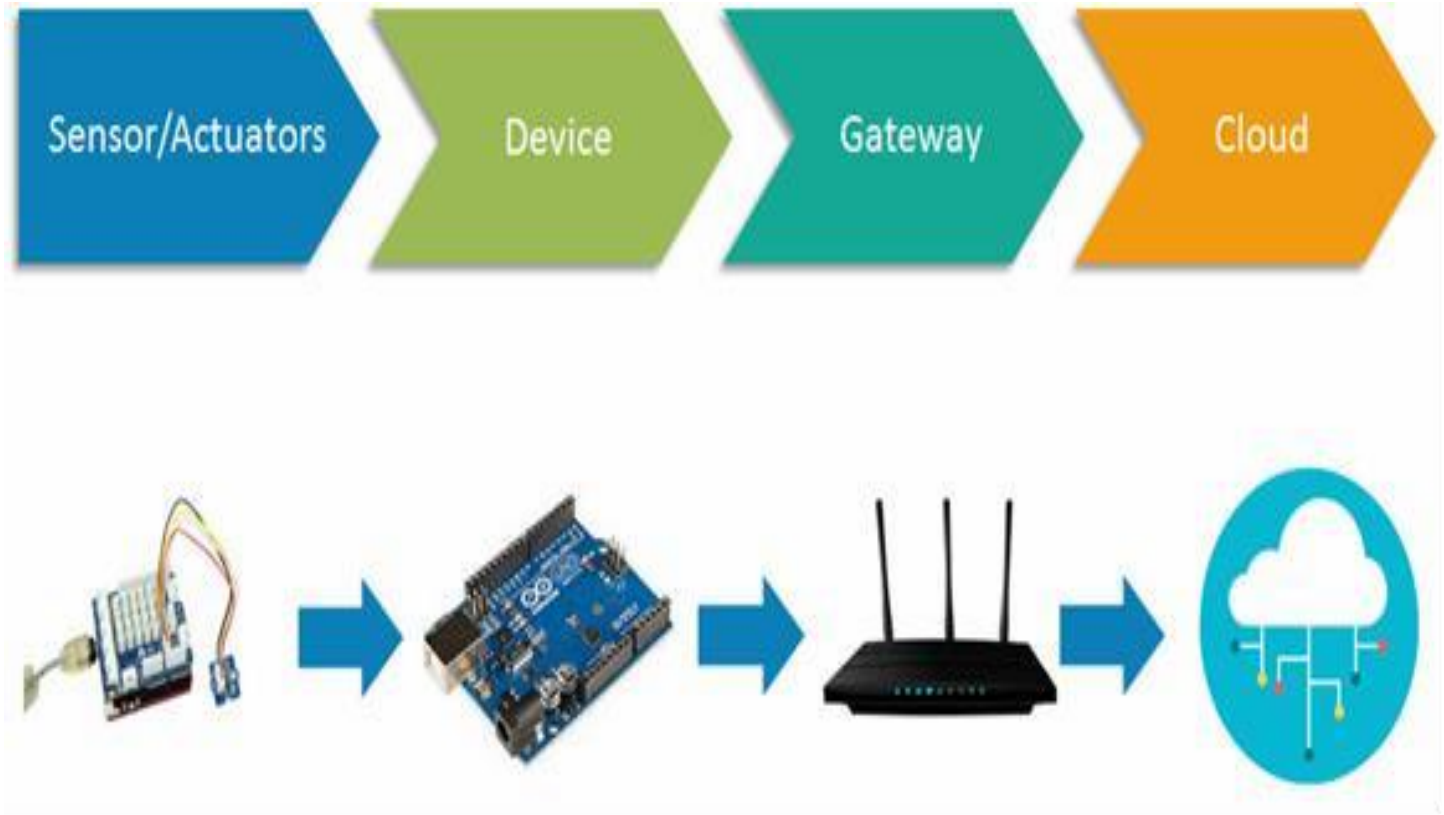
UNIT-III
IoT Platforms Design Methodology

State of the art

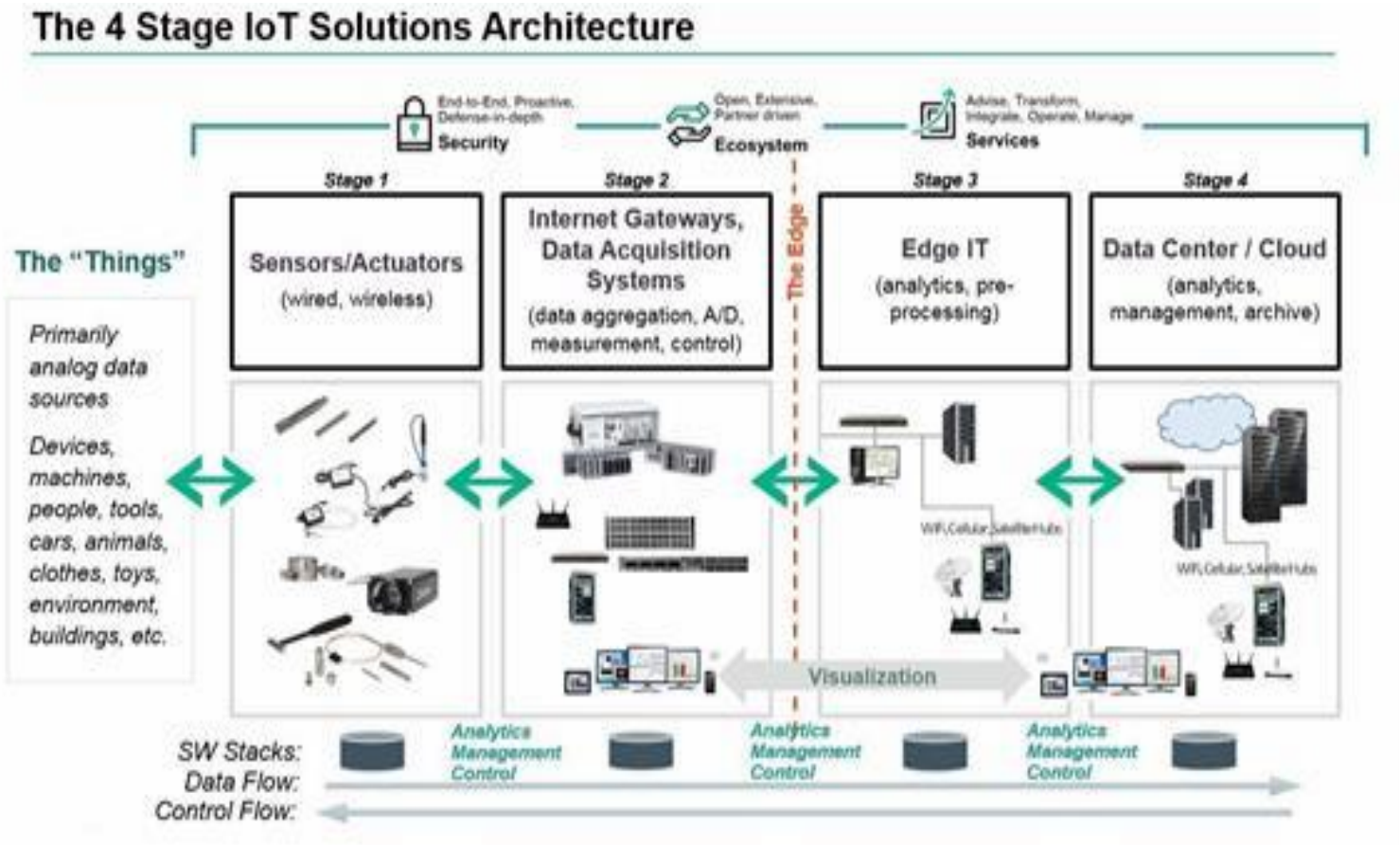
IoT architecture varies from solution to solution, based on the type of solution which we intend to build. IoT as a technology majorly consists of four main components, over which an architecture is framed.

1. Sensors
2. Devices
3. Gateway
4. Cloud

State of the art



Stages of IoT Architecture



Stage 1:- Sensors/actuators

- Sensors collect data from the environment or object under measurement and turn it into useful data. Think of the specialized structures in your cell phone that detect the directional pull of gravity and the phone's relative position to the “thing” we call the earth and convert it into data that your phone can use to orient the device.
- Actuators can also intervene to change the physical conditions that generate the data. An actuator might, for example, shut off a power supply, adjust an air flow valve, or move a robotic gripper in an assembly process.
- The sensing/actuating stage covers everything from legacy industrial devices to robotic camera systems, water level detectors, air quality sensors, accelerometers, and heart rate monitors.

Stage 1:- Sensors/actuators

- And the scope of the IoT is expanding rapidly, thanks in part to low-power wireless sensor network technologies and Power over Ethernet, which enable devices on a wired LAN to operate without the need for an A/C power source.

Stage 2:- The Internet gateway

- The data from the sensors starts in analog form. That data needs to be aggregated and converted into digital streams for further processing downstream.
- Data acquisition systems (DAS) perform these data aggregation and conversion functions.
- The DAS connects to the sensor network, aggregates outputs, and performs the analog-to-digital conversion. The Internet gateway receives the aggregated and digitized data and routes it over Wi-Fi, wired LANs, or the Internet, to Stage 3 systems for further processing.
- Stage 2 systems often sit in close proximity to the sensors and actuators.

Stage 2:- The Internet gateway

- For example, a pump might contain a half-dozen sensors and actuators that feed data into adata
- aggregation device that also digitizes the data. This device might be physically attached to the pump. An adjacent gateway device or server would then process the data and forward it to the Stage 3 or Stage 4 systems.
- Intelligent gateways can build on additional, basic gateway functionality by adding such capabilities as analytics, malware protection, and data management services. These systems enable the analysis of data streams in real time.

Stage 3:- Edge IT

- Once IoT data has been digitized and aggregated, it's ready to cross into the realm of IT. However, the data may require further processing before it enters the data center.
- This is where edge IT systems, which perform more analysis, come into play. Edge IT processing systems may be located in remote offices or other edge locations, but generally these sit in the facility or location where the sensors reside closer to the sensors, such as in a wiring closet.
- Because IoT data can easily eat up network bandwidth and swamp your data center resources, it's best to have systems at the edge capable of performing analytics as a way to lessen the burden on core IT infrastructure.

Stage 3:- Edge IT

- You'd also face security concerns, storage issues, and delays processing the data. With a staged approach, you can preprocess the data, generate meaningful results, and pass only those on. For example, rather than passing on raw vibration data for the pumps, you could aggregate and convert the data, analyze it, and send only projections as to when each device will fail or need service.

Stage 4:- The data center and cloud

- Data that needs more in-depth processing, and where feedback doesn't have to be immediate, gets forwarded to physical data center or cloud-based systems, where more powerful IT systems can analyze, manage, and securely store the data.
- It takes longer to get results when you wait until data reaches Stage 4, but you can execute a more in-depth analysis, as well as combine your sensor data with data from other sources for deeper insights.
- Stage 4 processing may take place on-premises, in the cloud, or in a hybrid cloud system, but the type of processing executed in this stage remains the same, regardless of the platform.

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:

- Multi-paradigm programming language

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

- 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.

- Interactive Language

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

Class

- **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.

- **Strings**

  A string is simply a list of characters in order. There are no limits to the number of characters you can have in astring.

- **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.

Class

- **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.

- **Strings**

  A string is simply a list of characters in order. There are no limits to the number of characters you can have in astring.

- **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.

Control Flow statements

- **for**

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.

- **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.

Control Flow statements

- **While**

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

- **Range statement**

  The range statement in Python generates a list of numbers in arithmetic progression.

- **Break**

  Break statement breaks out of the for/while loop

- **Continue**

  Continue statement continues with the next iteration

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 function.

- **Keyword Arguments**

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

- **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 (*)

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.

Date/Time Operations

- 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

UNIT-IV
IoT Physical Devices and Endpoints

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.

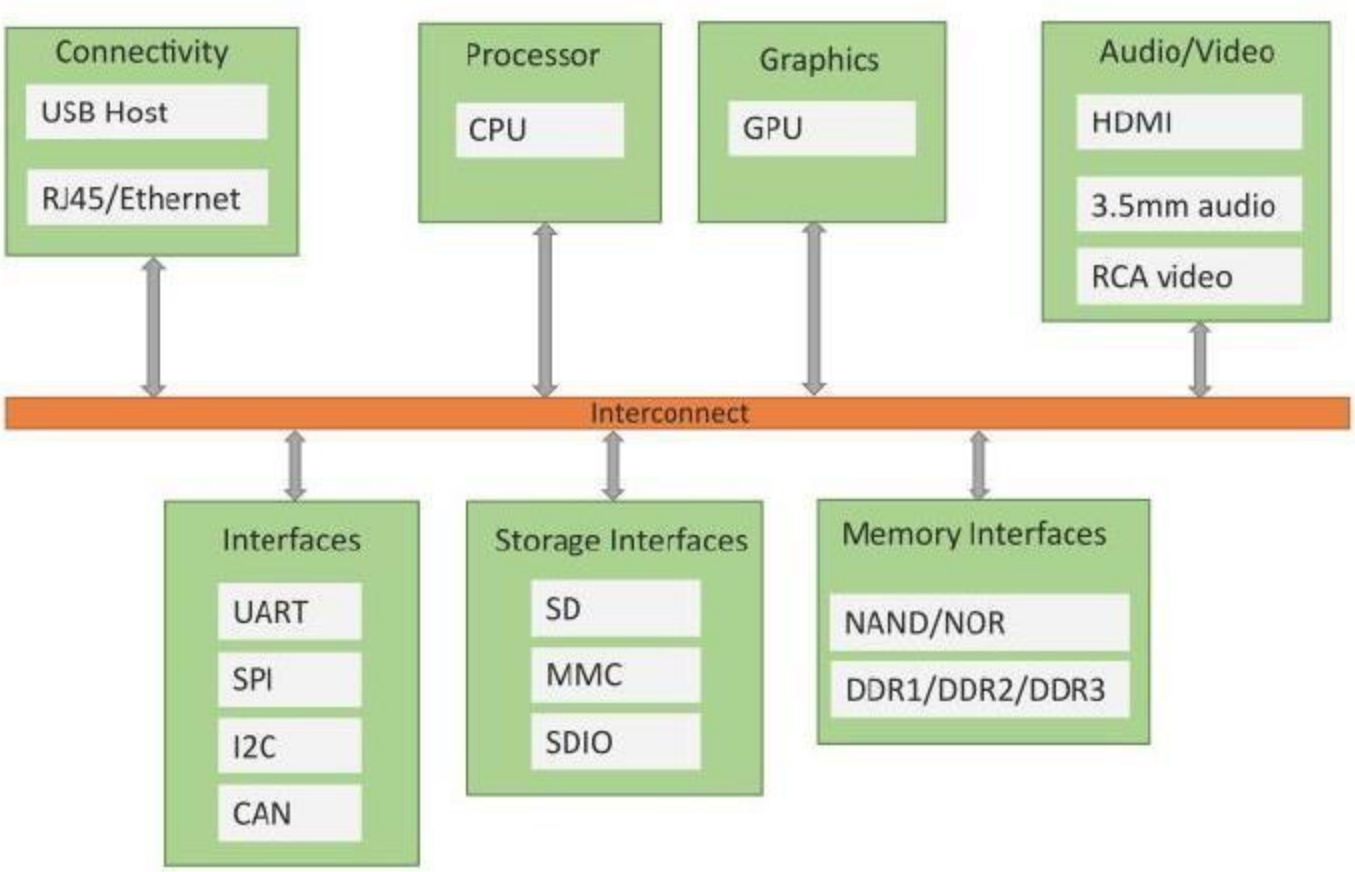
IoT Device Examples

- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information about 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 service.

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.

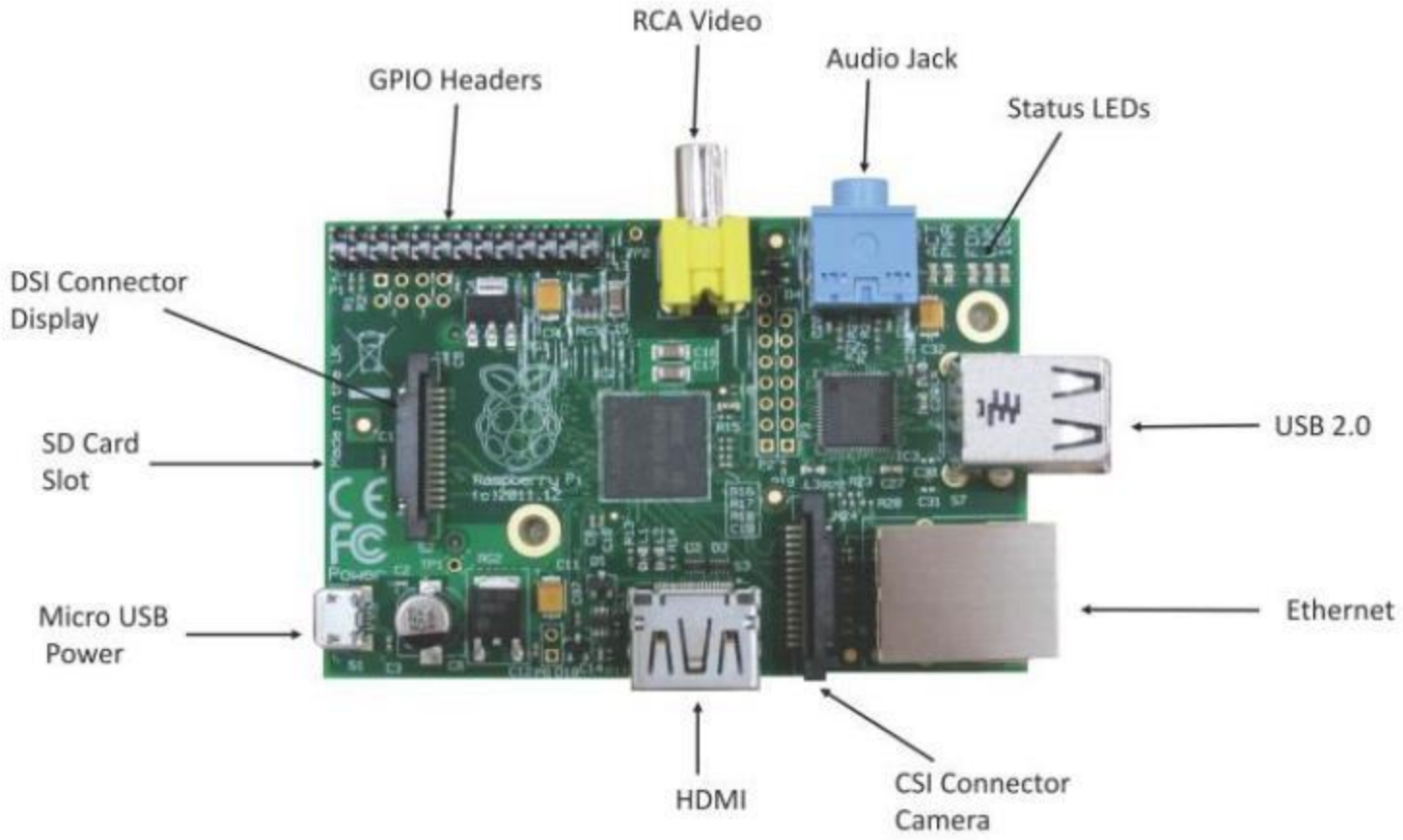
Block diagram of an IoT Device



Exemplary Device: 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 the box".

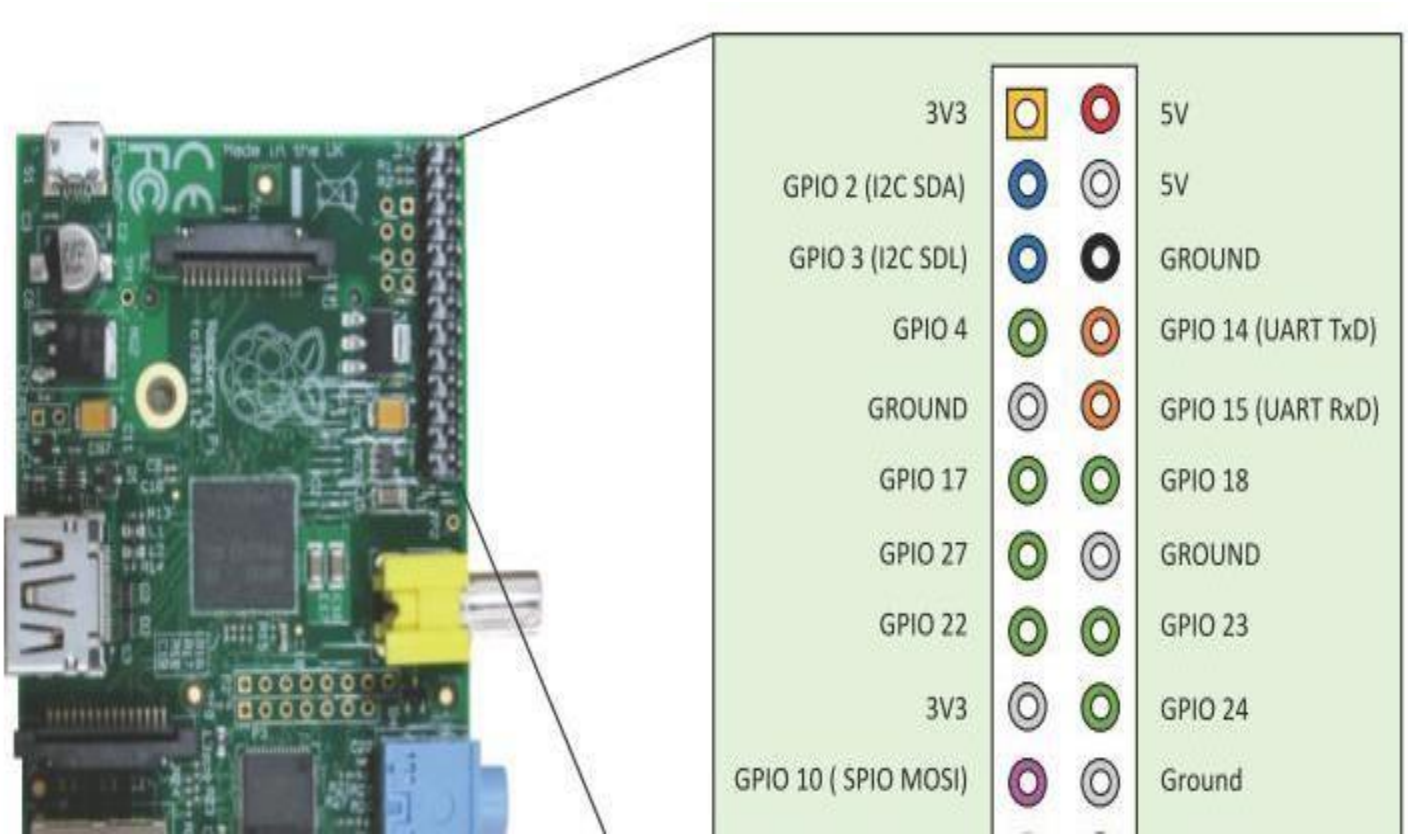
Raspberry Pi



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 system.

Raspberry Pi GPIO



Raspberry Pi Interfaces

1. **Serial**

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

2. **SPI**

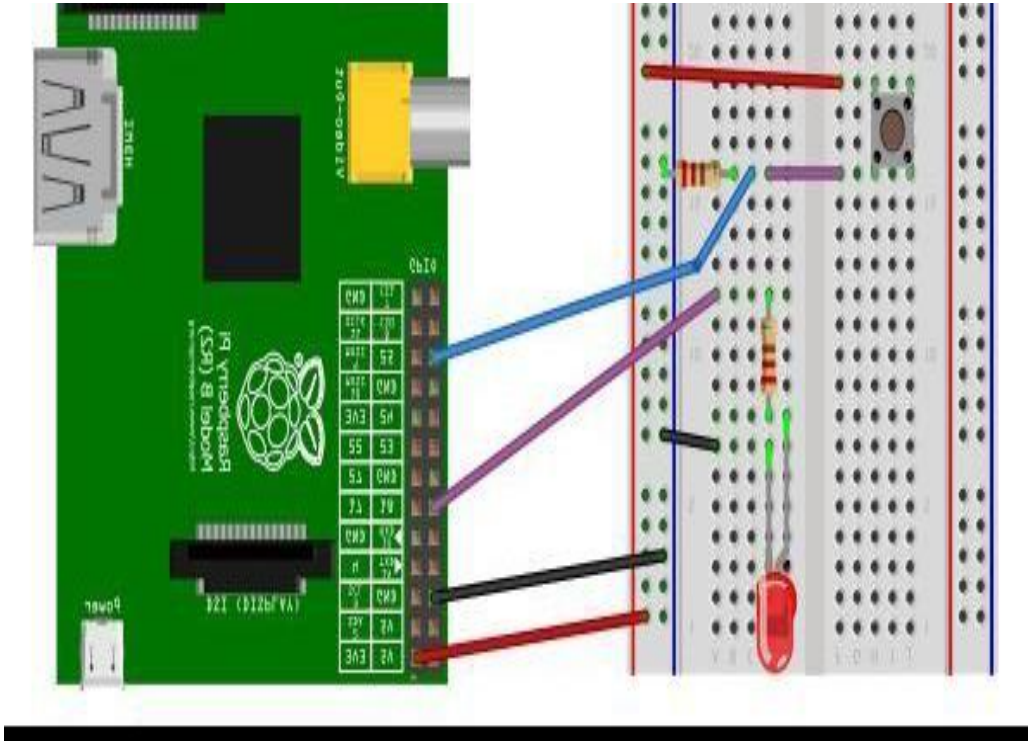
   Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.

3. **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).

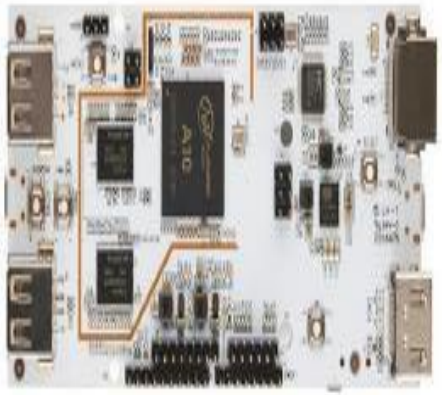
Interfacing LED and switch

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)

#Switch Pin
GPIO.setup(25, GPIO.IN)
#LED Pin
GPIO.setup(18, GPIO.OUT)
state=false

def toggleLED(pin):
        state = not state
        GPIO.output(pin, state)

while True:
        try:
                if (GPIO.input(25) == True):
                        toggleLED(pin)
                sleep(.01)
                except KeyboardInterrupt:
                        exit()
```



Other Devices

- pcDuino
- BeagleBone Black
- Cubieboard



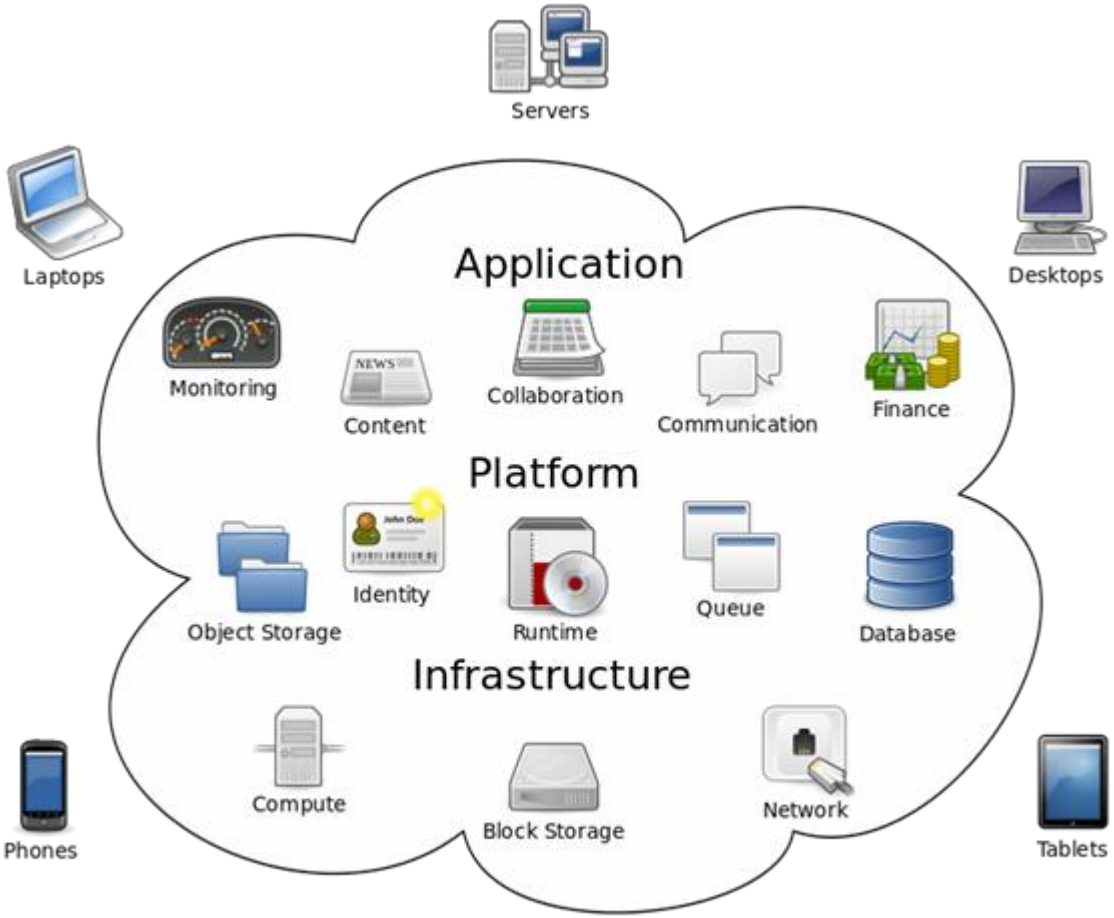
UNIT-V
IoT Physcial Servers and Cloud Offerings

Introduction to Cloud Computing

The worker can use a cloud computing service to finish their work because the data is managed remotely by a server. Another example: you have a problem with your mobile device and you need to reformat it or reinstall the operating system. You can use Google Photos to upload your photos to internet-based storage.

After the reformat or reinstall, you can then either move the photos back to you device or you can view the photos on your device from the internet when you want.

Cloud Computing



Cloud computing

Characteristics

Third, cloud computing allows for resource pooling, meaning information can be shared with those who know where and how (have permission) to access the resource, anytime and anywhere. This lends to broader collaboration or closer connections with other users. From an IoT perspective, just as we can easily assign an IP address to every "thing" on the planet, we can share the "address" of the cloud-based protected and stored information with others and pool resources.

Fourth, cloud computing features rapid elasticity, meaning users can readily scale the service to their needs. You can easily and quickly edit your software setup, add or remove users, increase storage space, etc. This characteristic will further empower IoT by providing elastic computing power, storage and networking.

Service and Deployment

Service models

Service delivery in cloud computing comprises three different service models: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS).

Software as a service (SaaS) provides applications to the cloud's end user that are mainly accessed via a web portal or service-oriented architecture-based web service technology. These services can be seen as ASP (application service provider) on the application layer. Usually, a specific company that uses the service would run, maintain and give support so that it can be reliably used over a long period of time.

Service and Deployment

Platform as a service (PaaS) consists of the actual environment for developing and provisioning cloud applications. The main users of this layer are developers that want to develop and run a cloud application for a particular purpose. A proprietary language was supported and provided by the platform (a set of important basic services) to ease communication, monitoring, billing and other aspects such as startup as well as to ensure an application's scalability and flexibility. Limitations regarding the programming languages supported, the programming model, the ability to access resources, and the long-term persistence are possible disadvantages.

Service and Deployment

Platform as a service (PaaS) consists of the actual environment for developing and provisioning cloud applications. The main users of this layer are developers that want to develop and run a cloud application for a particular purpose. A proprietary language was supported and provided by the platform (a set of important basic services) to ease communication, monitoring, billing and other aspects such as startup as well as to ensure an application's scalability and flexibility. Limitations regarding the programming languages supported, the programming model, the ability to access resources, and the long-term persistence are possible disadvantages.

Deployment models

Platform as a service (PaaS) consists of the actual environment for developing and provisioning cloud applications. The main users of this layer are developers that want to develop and run a cloud application for a particular purpose. A proprietary language was supported and provided by the platform (a set of important basic services) to ease communication, monitoring, billing and other aspects such as startup as well as to ensure an application's scalability and flexibility. Limitations regarding the programming languages supported, the programming model, the ability to access resources, and the long-term persistence are possible disadvantages.

Deployment models

Finally, a hybrid cloud combines two or more distinct private, community or public cloud infrastructures such that they remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability. Normally, information that's not critical is outsourced to the public cloud, while business-critical services and data are kept within the control of the organization.

DeploCLOUD STORAGE API

To the application, the cloud storage system is just another target device, like tape or disk-based storage. An application program interface (API) is code that allows two software programs to communicate with each other. The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application. APIs are implemented by function calls composed of verbs and nouns. The required syntax is described in the documentation of the application beingcalled.

CLOUD STORAGE API

Three basic types of APIs

APIs take three basic forms: local, web-like and program-like.

1. **Local APIs** are the original form, from which the name came. They offer OS middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local API form.
2. **Web APIs** are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API. Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.

CLOUD STORAGE API

Three basic types of APIs

APIs take three basic forms: local, web-like and program-like.

1. **Local APIs** are the original form, from which the name came. They offer OS middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local API form.
2. **Web APIs** are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API. Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.

CLOUD STORAGE API

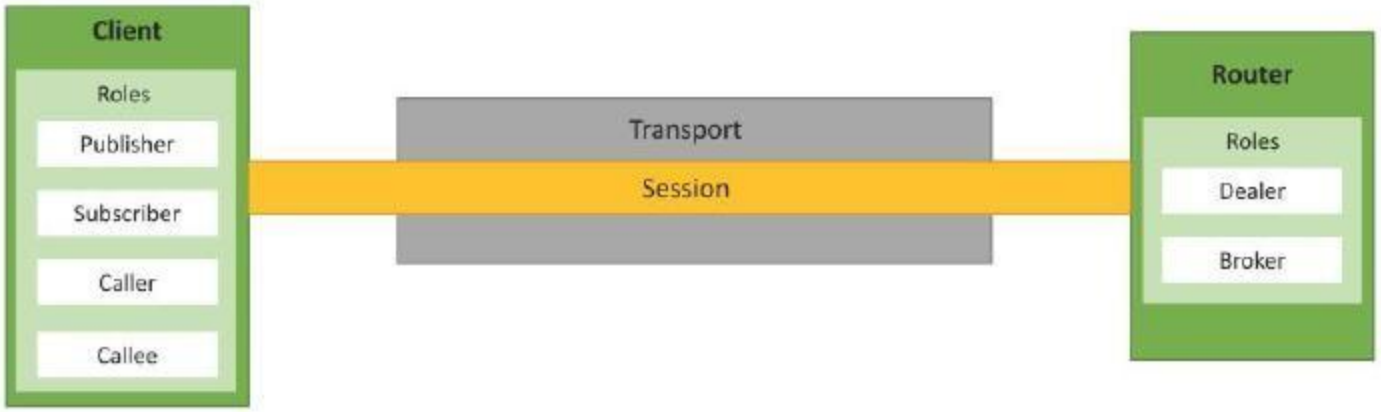
Program APIs are based on remote procedure call (RPC) technology that makes a remote program component appear to be local to the rest of the software. Service Oriented Architecture (SOA) APIs, such as Microsoft's WS-series of APIs, are program APIs.

IoT / Cloud Convergence

- Internet-of-Things can benefit from the scalability, performance and pay-as-you-go nature of cloud computing infrastructures. Indeed, as IoT applications produce large volumes of data and comprise multiple computational components (e.g., data processing and analytics algorithms), their integration with cloud computing infrastructures could provide them with opportunities for cost-effective on-demand scaling. As prominent examples consider the following settings:
- A Small Medium Enterprise (SME) developing an energy management IoT product, targeting smart homes and smart buildings. By streaming the data of the product (e.g., sensors and WSN data) into the cloud it can accommodate its growth needs in a scalable and cost effective fashion. As the SMEs acquires more customers and performs more deployments of its product, it is able

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 (Publisher, Subscriber, Callee or Caller).

Amazon EC2 – Python Example

- Boto is a Python package that provides interfaces to Amazon Web Services (AWS)

- In this example, a connection to EC2 service is fi rst established by calling boto.ec2.connect_to_region.
- The EC2 region, AWS access key and AWS secret key are passed to this function. After connecting to EC2 , a new instance is launched using the conn.run_instances function.
- The AMI-ID, instance type, EC2 key handle and security group are passed to this function.

```
#Python program for launching an EC2 instance
import boto.ec2
from time import sleep
ACCESS_KEY="<enter access key>"
SECRET_KEY="<enter secret key>"

REGION="us-east-1"
AMI_ID = "ami-d0f89fb9"
EC2_KEY_HANDLE = "<enter key handle>"
INSTANCE_TYPE="t1.micro"
SECGROUP_HANDLE="default"

conn = boto.ec2.connect_to_region(REGION, aws_access_key_id=ACCESS_KEY,
                                  aws_secret_access_key=SECRET_KEY)

reservation = conn.run_instances(image_id=AMI_ID, key_name=EC2_KEY_HANDLE,
                                 instance_type=INSTANCE_TYPE,
                                 security_groups = [ SECGROUP_HANDLE, ] )
```

Amazon AutoScaling – 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, etc.

```
#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)
```

- 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.

```
#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)
```

- 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 Example

- 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='<enter>',
   aws_secret_access_key='<enter>')

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 Example

- 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, etc.

```
#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, ] )
```

Amazon DynamoDB – Python Example

- In this example, a connection to DynamoDB service is first established by calling boto.dynamodb.connect_to_region.
- After connecting to DynamoDB service, a schema for the new table is created by calling conn.create_schema.
- The schema includes the hash key and range key names and types.
- A DynamoDB table is then created by calling conn.create_table function with the table schema, read units and write units as input parameters.

```
# Python program for creating a DynamoDB table (excerpt)
import boto.dynamodb

ACCESS_KEY="<enter>"
SECRET_KEY="<enter>"
REGION="us-east-1"

#Connecting to DynamoDB
conn = boto.dynamodb.connect_to_region(REGION,
  aws_access_key_id=ACCESS_KEY,
  aws_secret_access_key=SECRET_KEY)

table_schema = conn.create_schema(
    hash_key_name='msgid',
    hash_key_proto_value=str,
    range_key_name='date',
    range_key_proto_value=str
  )

#Creating table with schema
table = conn.create_table(
    name='my-test-table',
    schema=table_schema,
    read_units=1,
    write_units=1
  )
```

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.

#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

- 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.