
Subject Name	: Internet of Things
Department	: Computer Applications
Class	: III B.C.A.
Semester	: VI

UNIT I: 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.

1.1 INTRODUCTION

The Internet of Things (IoT) describes **the network of physical objects**—“things”—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. **Kevin Ashton** is an innovator and consumer sensor expert who coined the phrase “the Internet of Things” to describe the network connecting objects in the physical world to the Internet.

The Internet of Things refers to **the rapidly growing network of connected objects that are able to collect and exchange data in real time using embedded sensors**. Thermostats, cars, lights, refrigerators, and more appliances can all be connected to the IoT. The world's first IoT device was invented in **the early 1980s** at the Carnegie Mellon University. A group of students from the university created a way to get their campus Coca-Cola vending machine to report on its contents through a network in order to save them the trek if the machine was out of Coke.

our mobile device which contains GPS Tracking, Mobile Gyroscope, Adaptive brightness, Voice detection, Face detection etc. These components have their own individual features, but what about if these all communicate with each other to provide a better environment? For example, the phone brightness is adjusted based on my GPS location or my direction.

1.2 DEFINITION:

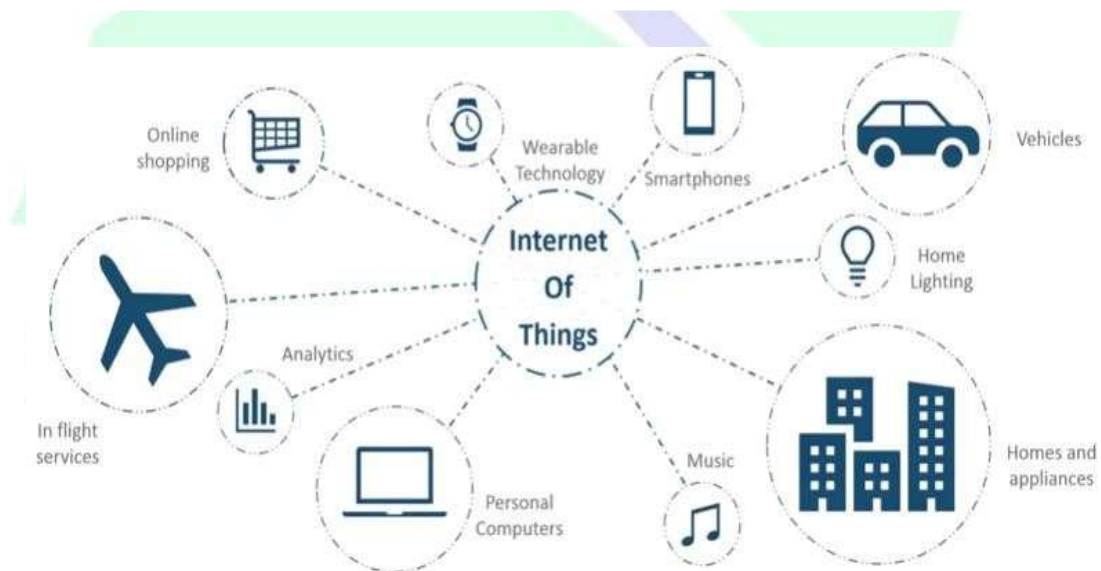
IoT is a **dynamic global network infrastructure of physical and virtual objects having unique identities**, which are embedded with software, sensors, actuators, electronic and network connectivity to facilitate intelligent applications by collecting and exchanging data.

IoT stands for Internet of Things, which means accessing and controlling daily usable equipments and devices using Internet.

1.3 CHARACTERISTICS OF IOT:

IoT is a **dynamic global network infrastructure of physical and virtual objects having unique identities**, which are embedded with software, sensors, actuators, electronic and network connectivity to facilitate intelligent applications by collecting and exchanging data. IoT has ten major features, and they are- **scalability, connectivity, artificial intelligence, security, dynamic nature, endpoint management, integration, analyzing, and compact nature of devices.**

Connecting everyday things embedded with electronics, software, and sensors to internet enabling to collect and exchange data without human interaction called as the Internet of Things (IoT).



Picture: 1.1 Internet of things in real life

IoT is an advanced automation and analytics system, which deals with artificial intelligence, sensor, networking, electronic, cloud messaging etc. to deliver complete systems for the product or services. The system created by IoT has greater transparency, control, and performance.

As we have a platform such as a cloud that contains all the data through which we connect all the things around us. For example, a house, where we can connect our home appliances such as air conditioner, light, etc. through each other and all these things are managed at the same platform. Since we have a platform, we can connect our car, track its fuel meter, speed level, and also track the location of the car.

IoT is different for different IoT echo system (architecture). However, the key concept of there working are similar. The entire working process of IoT starts with the device themselves, such as smartphones, digital watches, electronic appliances, which securely communicate with the IoT platform. The platforms collect and analyze the data from all multiple devices and platforms and transfer the most valuable data with applications to devices.

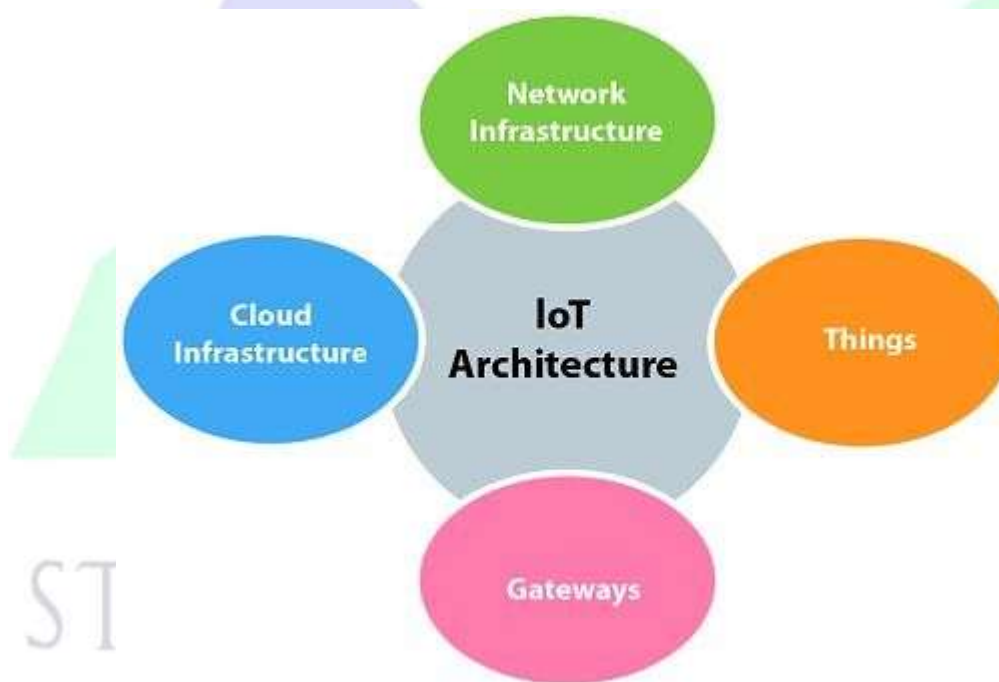


Figure: 1.2 IOT Architecture

Features of IOT

The most important features of IoT on which it works are connectivity, analyzing, integrating, active engagement, and many more. Some of them are listed below:

Connectivity: Connectivity refers to establish a proper connection between all the things of IoT to IoT platform it may be server or cloud. After connecting the IoT devices, it needs a high speed messaging between the devices and cloud to enable reliable, secure and bi-directional communication.

Analyzing: After connecting all the relevant things, it comes to real-time analyzing the data collected and use them to build effective business intelligence. If we have a good insight into data gathered from all these things, then we call our system has a smart system.

Integrating: IoT integrating the various models to improve the user experience as well.

Artificial Intelligence: IoT makes things smart and enhances life through the use of data. For example, if we have a coffee machine whose beans have going to end, then the coffee machine itself order the coffee beans of your choice from the retailer.

Sensing: The sensor devices used in IoT technologies detect and measure any change in the environment and report on their status. IoT technology brings passive networks to active networks. Without sensors, there could not hold an effective or true IoT environment.

Active Engagement: IoT makes the connected technology, product, or services to active engagement between each other.

Endpoint Management: It is important to be the endpoint management of all the IoT system otherwise; it makes the complete failure of the system. For example, if a coffee machine itself order the coffee beans when it goes to end but what happens when it orders the beans from a retailer and we are not present at home for a few days, it leads to the failure of the IoT system. So, there must be a need for endpoint management.

1.4 PHYSICAL DESIGN OF IOT

The physical design of an IoT system is referred to **the Things/Devices and protocols that used to build an IoT system**. All these things/Devices are called Node Devices and every device has a unique identity that performs remote sensing, actuating, and monitoring work.

Connectivity

Devices like USB host and ETHERNET are used for connectivity between the devices and server.

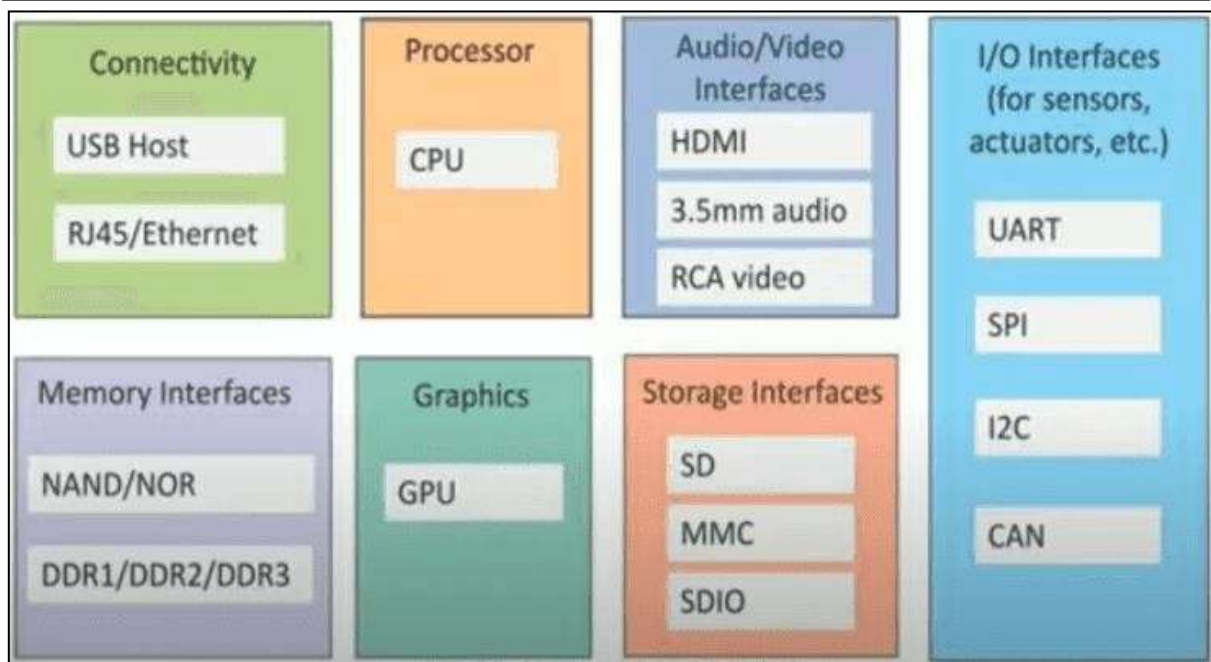


FIGURE 1.3. IoT physical design

Processor

A processor like a CPU and other units are used to process the data. These data are further used to improve the decision quality of an IoT system.

Audio/Video Interfaces

An interface like HDMI and RCA devices is used to record audio and videos in a system.

Input/Output interface

To giving input and output signals to sensors, and actuators we use things like UART, SPI, CAN, etc.

Storage Interfaces

Things like SD, MMC, SDIO are used to store the data generated from an IoT device. Other things like DDR, GPU are used to control the activity of an IoT system.

IOT Protocols:

These protocols are used to establish communication between a node device and server over the internet. it helps to send commands to an IoT device and receive data from an IoT device over the internet. we use different types of protocols that present on both the server and client-side and these protocols are managed by network layers like application, transport, network, and link layer.

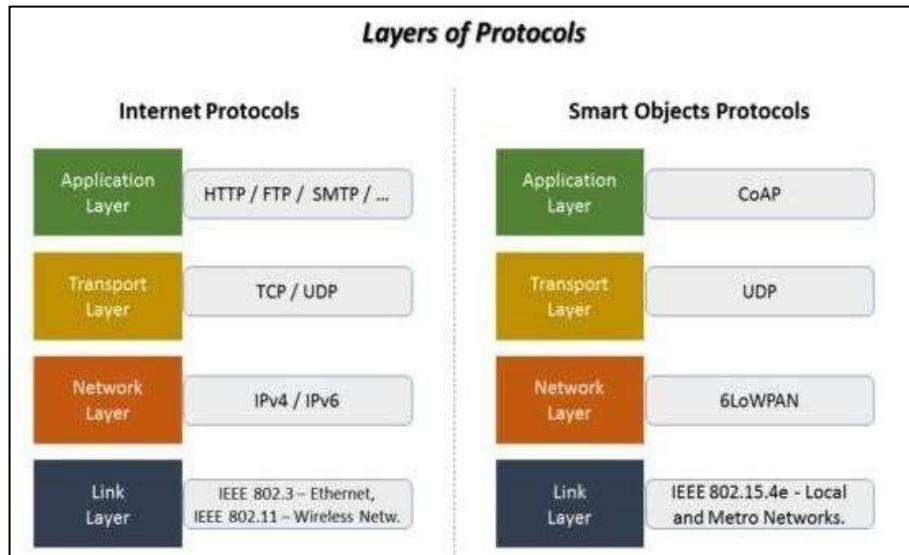


Figure 1.4. Layers of IOT Protocols

APPLICATION LAYER PROTOCOL

In this layer, protocols define how the data can be sent over the network with the lower layer protocols using the application interface. these protocols including HTTP, WebSocket, XMPP, MQTT, DDS, and AMQP protocols.

- **HTTP:** Hypertext transfer protocol is a protocol that presents in an application layer for transmitting media documents. it is used to communicate between web browsers and servers. it makes a request to a server and then waits till it receives a response and in between the request server does not keep any data between two requests.
- **WebSocket:** This protocol enables two-way communication between a client and a host that can be run on an untrusted code in a controlled environment. this protocol is commonly used by web browsers.
- **MQTT:** It is a machine-to-machine connectivity protocol that was designed as a publish/subscribe messaging transport. and it is used for remote locations where a small code footprint is required.

TRANSPORT LAYER

This layer is used to control the flow of data segments and handle the error control. also, these layer protocols provide end-to-end message transfer capability independent of the underlying network.

- **TCP:** The transmission control protocol is a protocol that defines how to establish and maintain a network that can exchange data in a proper manner using the internet protocol.
- **UDP:** a user datagram protocol is a part of internet protocol called the connectionless protocol. this protocol not required to establish the connection to transfer data.

NETWORK LAYER

This layer is used to send datagrams from the source network to the destination network. we use IPv4 and IPv6 protocols as a host identification that transfers data in packets.

- **IPv4:** This is a protocol address that is a unique and numerical label assigned to each device connected with the network. an IP address performs two main functions host and location addressing. IPv4 is an IP address that is 32 bit long.
- **IPv6:** It is a successor of IPv4 that uses 128 bits for an IP address. it is developed by the IETF task force to deal with the long-anticipated problems.

LINK LAYER

Link-layer protocols are used to send data over the network's physical layer. it also determines how the packets are coded and signaled by the devices.

- **Ethernet:** It is a set of technologies and protocols that are used primarily in LANs. it defines the physical layer and the medium access control for wired ethernet networks.
- **WiFi:** It is a set of LAN protocols and specifies the set of media access control and physical layer protocols for implementing wireless local area networks.

1.5 LOGICAL DESIGN OF IOT

Logical design of IoT system refers to an abstract representation of the entities & processes without going into the low-level specifics of the implementation. For understanding Logical Design of IoT, we describes given below terms.

- IoT Functional Blocks

- IoT Communication Models
- IoT Communication APIs

IoT Functional Blocks:

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

Device: An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.

Communication: Handles the communication for the IoT system.

Services: services for device monitoring, device control service, data publishing services and services for device discovery.

Management: This block provides various functions to govern the IoT system.

Security: this block secures the IoT system and by providing functions such as authentication, authorization, message and content integrity, and data security.

Application: This is an interface that the users can use to control and monitor various aspects of the IoT system. Application also allow users to view the system status and view or analyze the processed data.

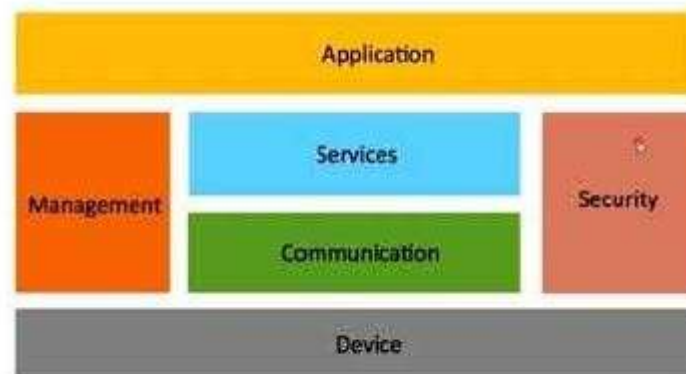


Figure: 1.5. Logical design of IoT

IoT Communication Models:

Request-Response Model

Request-response model is 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 representation, prepares the response, and then sends the response to the client. Request-response is a stateless communication model and each request-response pair is independent of others.

HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server.

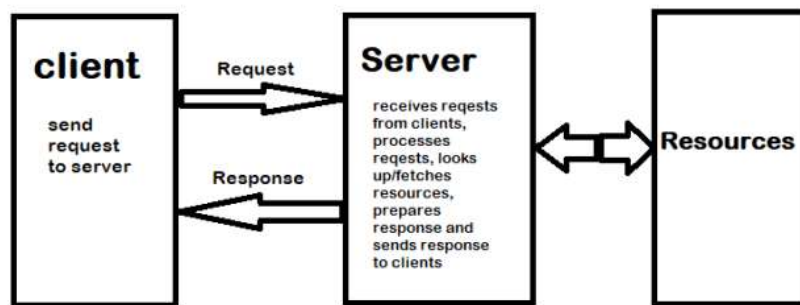


Figure: 1.6. Request – Response communication model

Publish-Subscribe 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 receive data for a topic from the publisher, it sends the data to all the subscribed consumers.

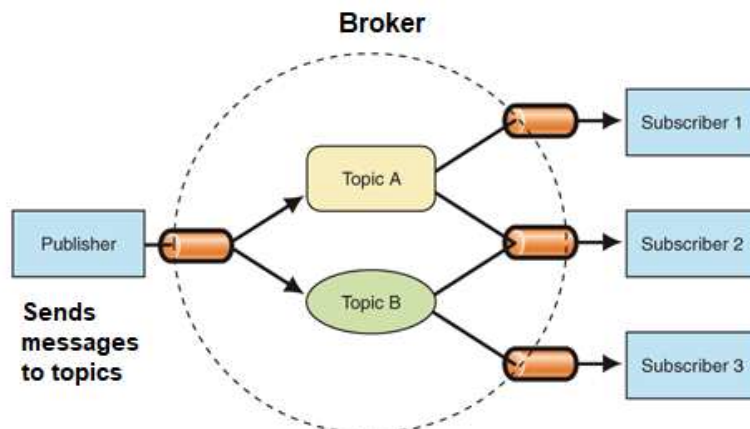
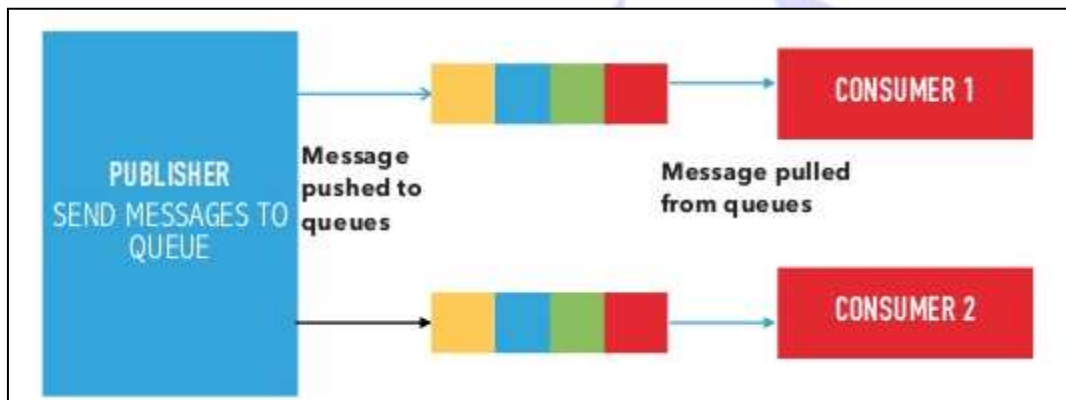


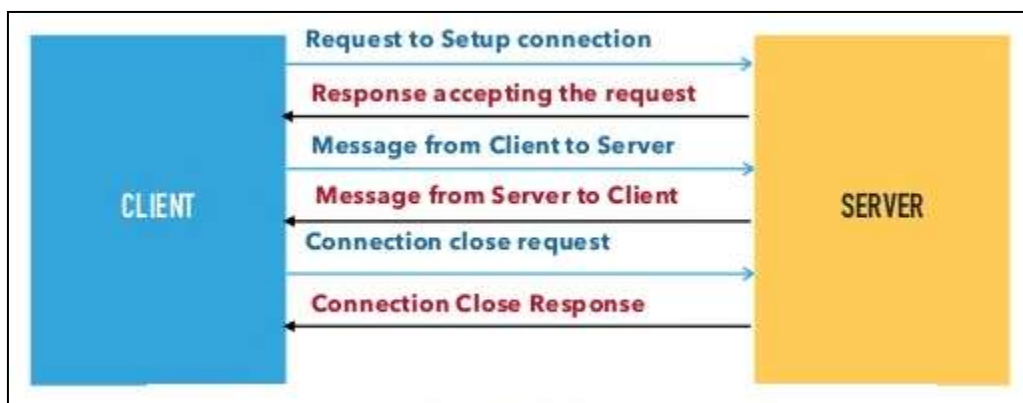
Figure: 1.7. Publish-Subscribe Model

Push-Pull 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 consumer pull data.

**Figure: 1.8. Push pull model****Exclusive Pair Model**

Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server. 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. Exclusive pair is stateful communication model and the server is aware of all the open connections.

**Figure 1.9. Exclusive Pair Model**

IoT Communication APIs

Generally, Two APIs For IoT Communication. These IoT Communication APIs are:

- REST-based Communication APIs
- WebSocket-based Communication APIs

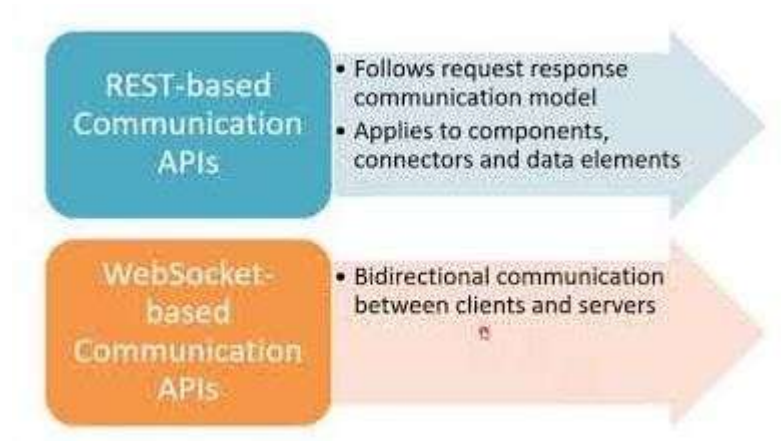


Figure: 1.10 IoT Communication APIs

REST-based Communication APIs

Representational state transfer (REST) is a set of architectural principles by which can design Web services the Web APIs that focus on systems' resources and how resource states are addressed and transferred. REST APIs that follow the request response communication model, the rest architectural constraint apply to the components, connector and data elements, within a distributed hypermedia system.

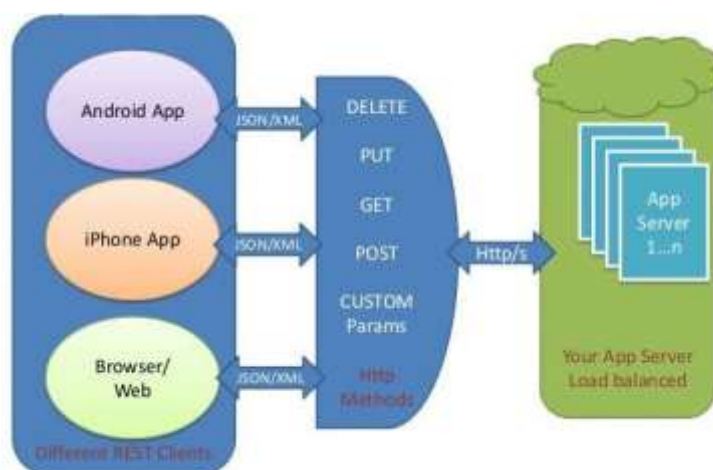


Figure: 1.11. REST API Structure

The rest architectural constraint are as follows:

Client-server – The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data, which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.

Stateless – Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.

Cache-able – Cache constraints require that the data within a response to a request be implicitly or explicitly levelled as cache-able or non cache-able. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.

Layered system – Layered system constraints constrain the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.

Uniform interface – Uniform interface constraints require that the method of communication between client and server must be uniform. Resources are identified in the requests (by URIs in web-based systems) and are themselves separate from the representations of the resources data returned to the client.

Code on demand – Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Web Socket based communication API

Web socket APIs allow bi-directional, full duplex communication between clients and servers. Web socket APIs follow the exclusive pair communication model. Unlike request-response model such as REST, the Web Socket APIs allow full duplex communication and do not require new connection to be setup for each message to be sent. Web socket communication begins with a connection setup request sent by the client to the server.

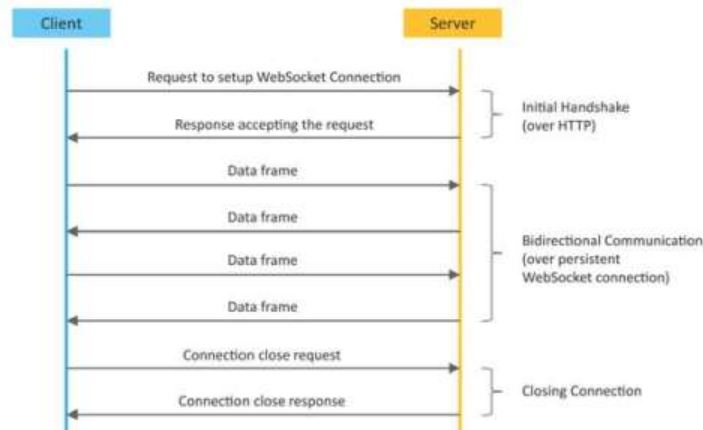


Figure: 1.12. **Web Socket API communication**

1.6 IOT ENABLING TECHNOLOGIES

Wireless Sensor Networks

A wireless sensor network comprises of distributed device with sensor, which are used to monitor the environmental and physical conditions. A WSN consists of a number of end-nodes and routers and a coordinator.

Routers are responsible for routing the data packets from end-nodes to the coordinator. The coordinator collects the data from all the nodes. Coordinator also act as a gateway that connects the WSN to the internet. Some examples of WSNs used in IoT systems are given below.

- Weather monitoring system
- Indoor air quality monitoring systems
- Soil moisture monitoring system
- Surveillance system
- Structural health monitoring system

Cloud Computing

Cloud computing is a trans-formative computing paradigm that involves delivering applications and services over the Internet Cloud computing involves provisioning of computing, networking and storage resources on demand and providing these resources as metered services to the users, in a “pay as you go” model.

The users can provision cloud-computing resources on demand, without requiring interactions with the cloud service Provider. Cloud computing resources can be accessed

over the network using standard access mechanisms that provide platform independent access with heterogeneous client platforms such as the workstations, laptops, tablets and smartphones.

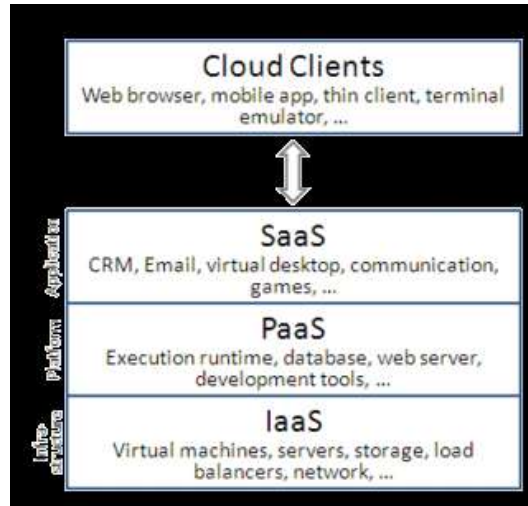


Figure: 1.13. Cloud computing

Cloud computing services are offered to users in different forms:

Infrastructure as a Service (IaaS): hardware is provided by an external provider and managed

Platform as a Service (PaaS): in addition to hardware, your operating system layer is managed

Software as a Service (SaaS): further to the above, an application layer is provided and managed.

Big Data Analytics

Big Data analytics is the process of collecting, organizing and analysing large sets of data (*called* Big Data) to discover patterns and other useful information. Analysts working with Big Data typically want the *knowledge* that comes from analyzing the data. examples of big data generated by IoT systems are given below.

- Sensor data generated by IoT system such as weather monitoring stations.
- Machine sensor data collected from sensors embedded in industrial and energy systems for monitoring their health and detecting Failures.
- Health and fitness data generated by IoT devices such as wearable fitness bands
- Data generated by IoT systems for location and tracking of vehicles

- Data generated by retail inventory monitoring systems

Communication protocols

Communication protocols form the backbone of IoT systems and enable network connectivity and coupling to applications. Communication protocols allow devices to exchange data over the network. Multiple protocols often describe different aspects of a single communication. A group of protocols designed to work together are known as a protocol suite; when implemented in software they are a protocol stack.

Internet communication protocols are published by the **Internet Engineering Task Force (IETF)**. The IEEE handles wired and wireless networking, and the **International Organization for Standardization (ISO)** handles other types. The ITU-T handles telecommunication protocols and formats for the **public switched telephone network (PSTN)**. As the PSTN and Internet converge, the standards are also being driven towards convergence.

Embedded Systems

- Embedded means something that is attached to another thing.
- An embedded system can be thought of as a computer hardware system having software embedded in it.
- An embedded system is a controller programmed and controlled by a real-time operating system (RTOS) with a dedicated function within a larger mechanical or electrical system.
- It is *embedded* as part of a complete device often including hardware and mechanical parts.
- Ninety-eight percent of all microprocessors are manufactured to serve as embedded system component.

An embedded system has three components –

- It has hardware.
- It has application software.
- It has Real Time Operating system (RTOS)

1.7 IOT LEVELS & DEPLOYMENT TEMPLATES

An IoT system comprises of the following components:

- **Device:** Allows identification, remote sensing, actuating and remote monitoring capabilities. IoT devices include wireless sensors, software, actuators, and computer devices with internet, enabling the transfer of data among objects or people automatically without human intervention.
- **Resource:** These are Software components on the IoT device for accessing, processing, and storing sensor information, or controlling actuators connected to the device.
- **Controller Service:** A native service that runs on the device and work between node device and 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.
- **Database:** A storage place for Collected or generated data. It can be local or cloud based.
- **Web Service:** Web services serve as a link between the IoT device, application, database and analysis components.
- **Analysis Component:** 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.

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

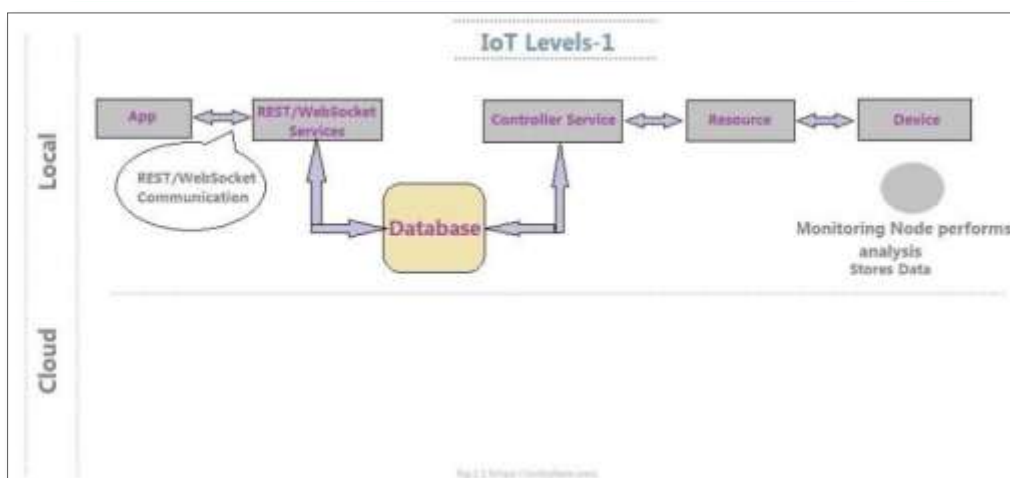
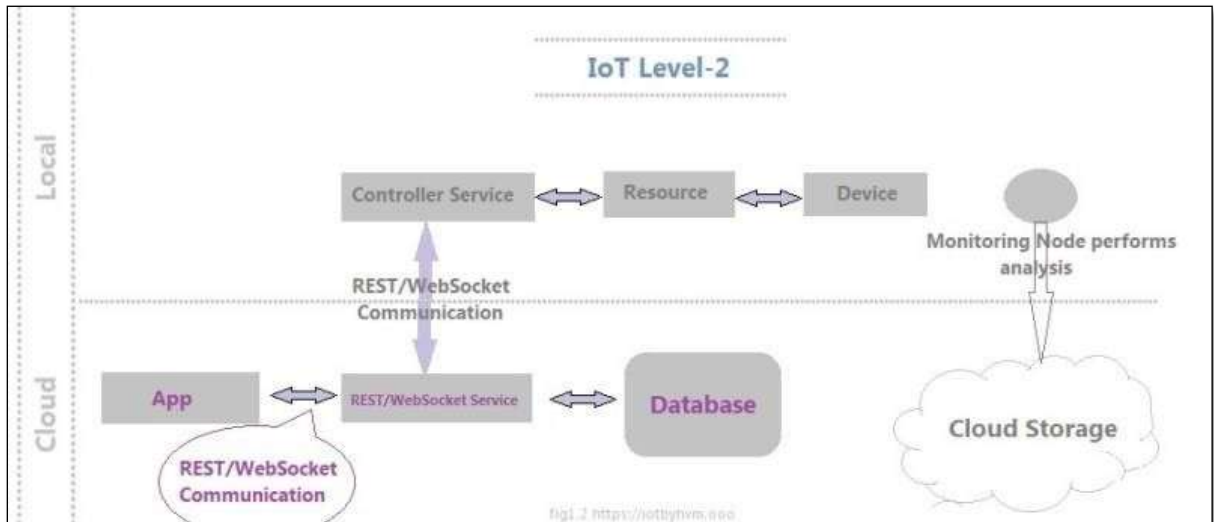


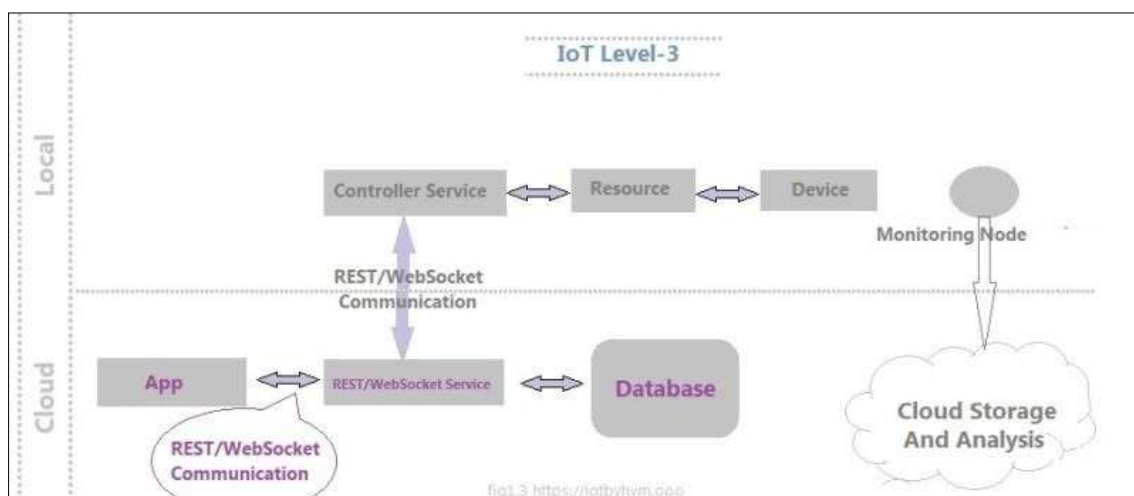
Figure:1.14. IOT Level – I

IoT Level-2

- It has a single node that performs sensing and/or actuation and local analysis (IoT Device and collected data).
- In this IoT Level Database and application establish in Cloud.

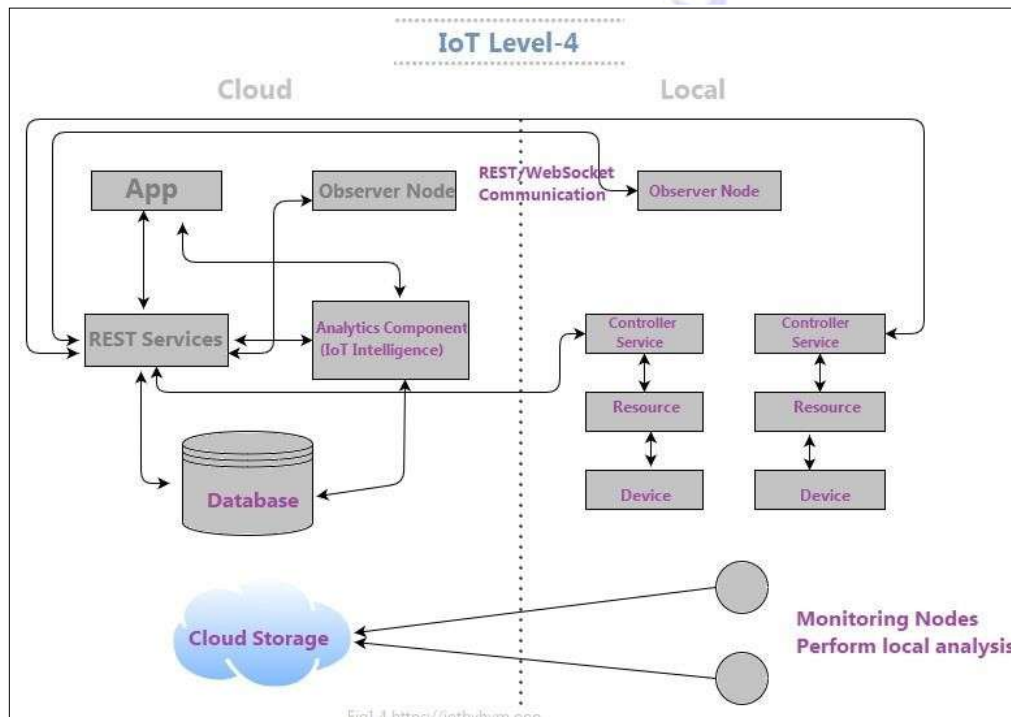
**Figure: 1.15. IOT Level – 2****IoT Level-3**

- It has has a single node. Database and application establish in the cloud.
- It is suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.

**Figure : 1.16. IOT Level – 3**

IoT Level-4

- It has multiple nodes that perform local analysis. It has Cloud based application and database. These IoT System contains local and cloud- based observer nodes which can subscribe to and receive information collected in the cloud from IoT node devices.
- It is suitable for solutions where we are using multiple nodes, the data involved is big and the analysis requirements are computationally intensive.

**Figure : 1.17. IOT Level – 4****IoT Level-5**

- It has multiple end nodes and one coordinator node. The end nodes use for sensing and/or actuation.
- In this model Coordinator node collects data from the end nodes and transfer to the cloud. In this model we used Cloud-based Database for store and Analyze data.
- It is suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.

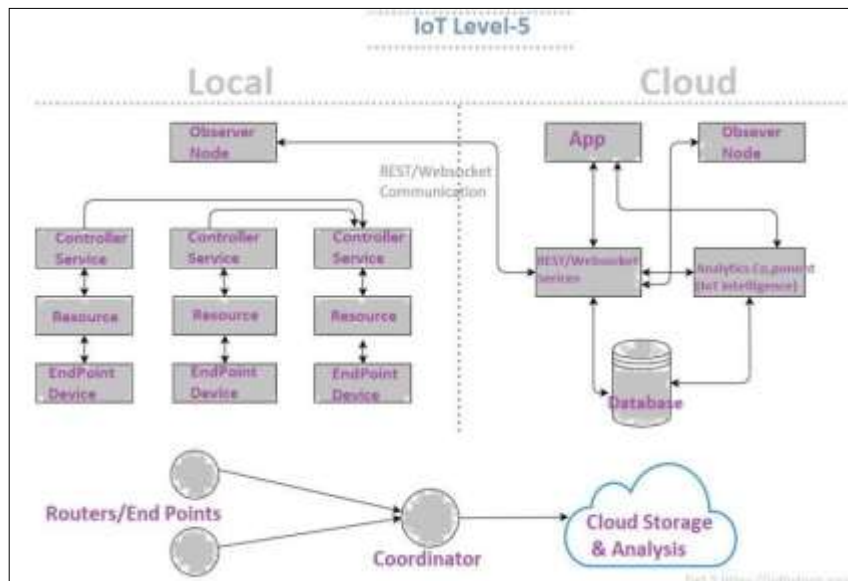


Figure: 1.18. IOT Level – 5

IoT Level-6

- It has multiple independent end nodes that used for sensing and/or actuation and transfer data to the cloud. We used Cloud-based database.
- The analytics component analyzes the data and stores the results in the cloud database and results are visualized with the cloud-based application.

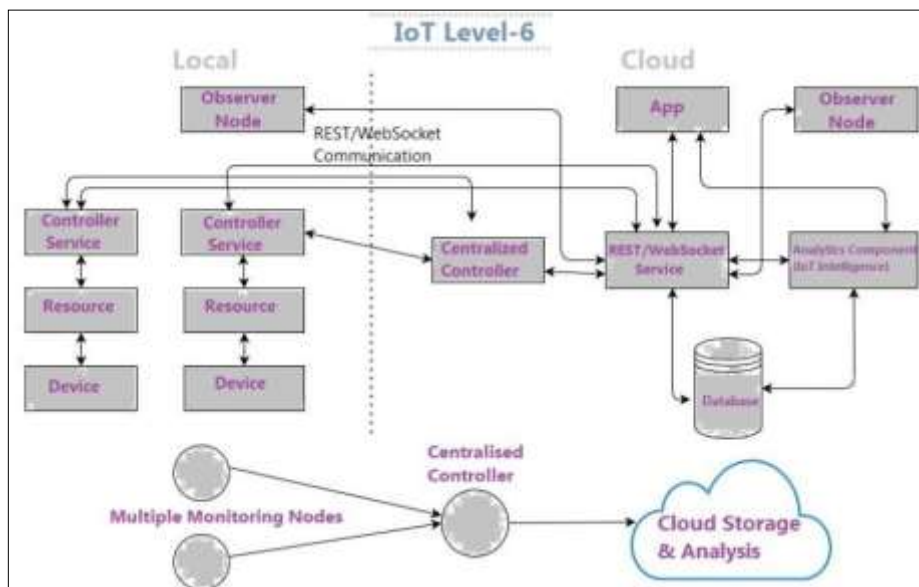


Figure: 1.19. IOT Level – 6

1.8 DOMAIN SPECIFIC IOT'S

we classify IoT use cases into the following eight application domains: **smart buildings and living**, • **smart healthcare**, • **smart environment**, • **smart city**, • **smart energy**, • **smart transport and mobility**, • **smart manufacturing and retail**, and • **smart agriculture**.

1.8.1 HOME AUTOMATION

The IoT based home automation consist of several **smart devices** for different applications of lighting, security, home entertainment etc. All these devices are integrated over a common network established by gateway and connected in a mesh network.

Home automation cases for:

1. Lighting
2. Doors
3. Windows
4. Thermostat
5. Gardens
6. Home routines

1.8.2 CITIES

- Smart Parking
- Smart Waste Management
- Traffic Management
- Air Quality Management
- Smart Infrastructure
- Smart transportation
- Smart utility meters

1.8.3 ENVIRONMENT –

- Air and Water Pollution
- Extreme Weather
- Commercial Farming
- water safety,
- endangered species protection

- noise pollution
- Forest fire detection
- River flood detection

1.8.4 ENERGY –

- Residential Energy
- Commercial Energy
- Reliability smart grid connectivity
- Renewable energy system
- Prognostics

1.8.5 RETAIL

- Smart payment
- Smart vending machine
- Smart inventory system

1.8.6 LOGISTICS

Route generation and scheduling

Fleet tracking

Shipment monitoring

Remote vehicle diagnostics

1.8.6 AGRICULTURE

- Precision Farming
- Agricultural Drones
- Livestock Monitoring
- Smart Greenhouses
- Monitor Climate Conditions
- Remote sensing
- Soil quality
- Crop Assessment
- Weather conditions
- Irrigation Monitoring

1.8.7 INDUSTRY

- Quality control
- Product safety
- Intelligent product enhancements
- Improved facility service
- Indoor air quality management

1.8.8 HEALTH AND LIFE STYLE

- Remote patient monitoring
- Glucose monitoring
- Heart-rate monitoring
- Hand hygiene monitoring
- Depression and mood monitoring
- Parkinson's disease monitoring
- Robotic surgery
- Connected contact lenses
- Health and fitness monitoring

Points to be Remember:

Basic of IOT

Physical layer in IOT

Communication protocols

IOT levels

Applications in IOT

Important Questions:

1. What are the IOT levels available?
2. What are day today IoT applications involved in the real life?
3. Explain about the physical design of the IoT.
4. Describe the characteristics of the IoT.
5. Define Communications APIs in IoT.

UNIT II: IoT and M2M - Difference between IoT and M2M - SDN and NFV for IoT - IoT systems management - SNMP - YANG - NETOPEER

2.1 IoT and M2M:

2.1.1. Machine to Machine (M2M):

Machine-to-machine, or M2M, is a describe any technology that enables networked devices to exchange information and perform actions without the manual assistance of humans.

Artificial intelligence (AI) and machine learning (ML) facilitate the communication between systems, allowing them to make their own autonomous choices.



Figure: 2.1 Machine to Machine

M2M technology was first adopted in manufacturing and industrial settings, where other technologies, such as SCADA and remote monitoring, helped remotely manage and control data from equipment.

M2M has since found applications in other sectors, such as healthcare, business and insurance. M2M is also the foundation for the internet of things (IoT). Benefits of M2M include:

- Reduced costs by minimizing equipment maintenance and downtime;
- Boosted revenue by revealing new business opportunities for servicing products in the field; and
- Improved customer service by proactively monitoring and servicing equipment before it fails or only when it is needed.



Figure: 2.2. M2M applications

2.2 Difference between IoT and M2M:

Basis of	IoT	M2M
Abbreviation	Internet of Things	Machine to Machine
Intelligence	Devices have objects that are responsible for decision making	Some degree of intelligence is observed in this
Connection type used	The connection is via Network and using various communication types.	The connection is a point to point
Communication protocol used	Internet protocols are used such as HTTP, FTP, and Telnet.	Traditional protocols and communication technology techniques are used
Data Sharing	Data is shared between other applications that are used to improve the end-user experience.	Data is shared with only the communicating parties.
Internet	Internet connection is required for communication	Devices are not dependent on the Internet.
Scope	A large number of devices yet scope is large.	Limited Scope for devices.

Business Type used	Business 2 Business(B2B) and Business 2 Consumer(B2C)	Business 2 Business (B2B)
Open API support	Supports Open API integrations.	There is no support for Open API'S
Examples	Smart wearables, Big Data and Cloud, etc.	Sensors, Data and Information, etc.

Table: 2.1. Difference between IoT and M2M

2.3. SDN AND NFV FOR LOT:

SOFTWARE DEFINED NETWORK:

SDN architecture presents a new solution that consists of separating the control plane from the data plane which is typically coupled together.

Network functions traditionally realized in specific hardware can now be abstracted and virtualized on any equipment.

A split between control and data path nodes is performed, so a centralized controller has a global view of the network while the data plane includes devices which simply forward packets following rules expressed by the controller. In order to communicate between these two layers, an open standard protocol is employed. This separation between the two layers simplifies the network management and help to simply program network control.

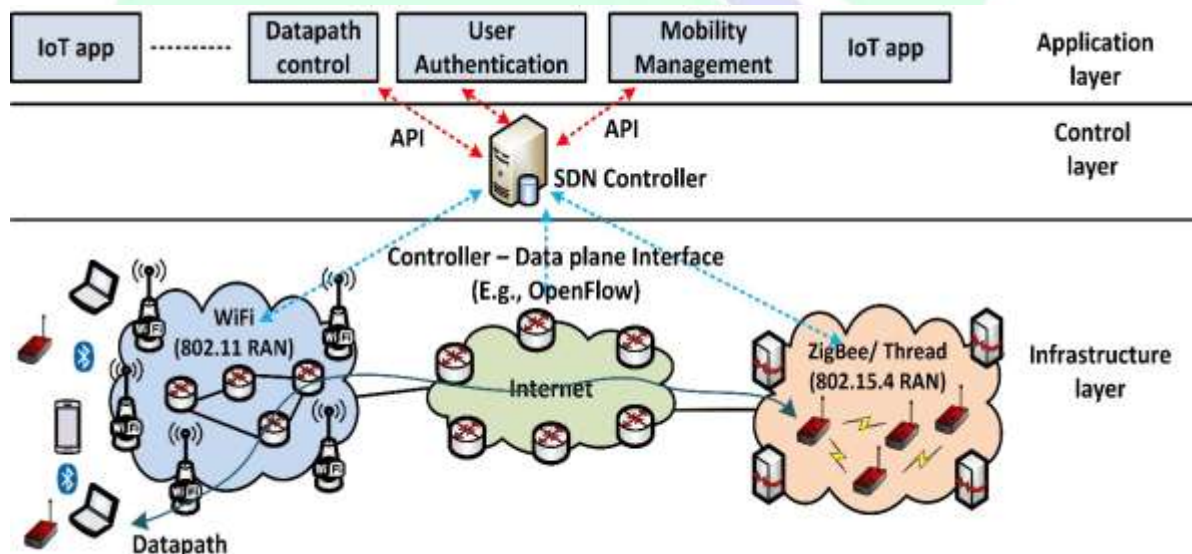


Figure 2.3. SDN Controller

The main concept of SDN architecture consists on the separation between control and forwarding functions. The Figure show demonstrate the multiple components of an SDN architecture, which is based on three layers separated by open interfaces.

The SDN application plane is a layer composed of a variety of applications that communicate via Northbound APIs. It's responsible for management, reporting functionalities such as monitoring or security.

The SDN controller plane represents the main entity in the network that facilitates the creation/destruction of network paths.

Typical SDN Controllers are Open Daylight and Floodlight. The SDN data plane includes different devices deprived from any intelligence.

They simply execute the controller's rules. The SDN architecture defines also the key interfaces between the different layers, which are East/West bound API that are implemented by the different controllers of the SDN and used to facilitate communications between them.

Hyper flow is one representative example of such APIs. Southbound API is implemented by the different forwarding devices in the SDN and enabling the communication between these devices and the controllers. For such APIs, we can enumerate Open Flow or NetConf.

The Northbound API is Implemented by the controllers of the SDN and used to facilitate the communication between controllers and the network management applications. In such SDN architecture, enhancing the QoE became easier by implementing specific algorithms within the controller entity.

The SDN manages in such case to enable new functionalities such as QoE monitoring and enforcement functions.

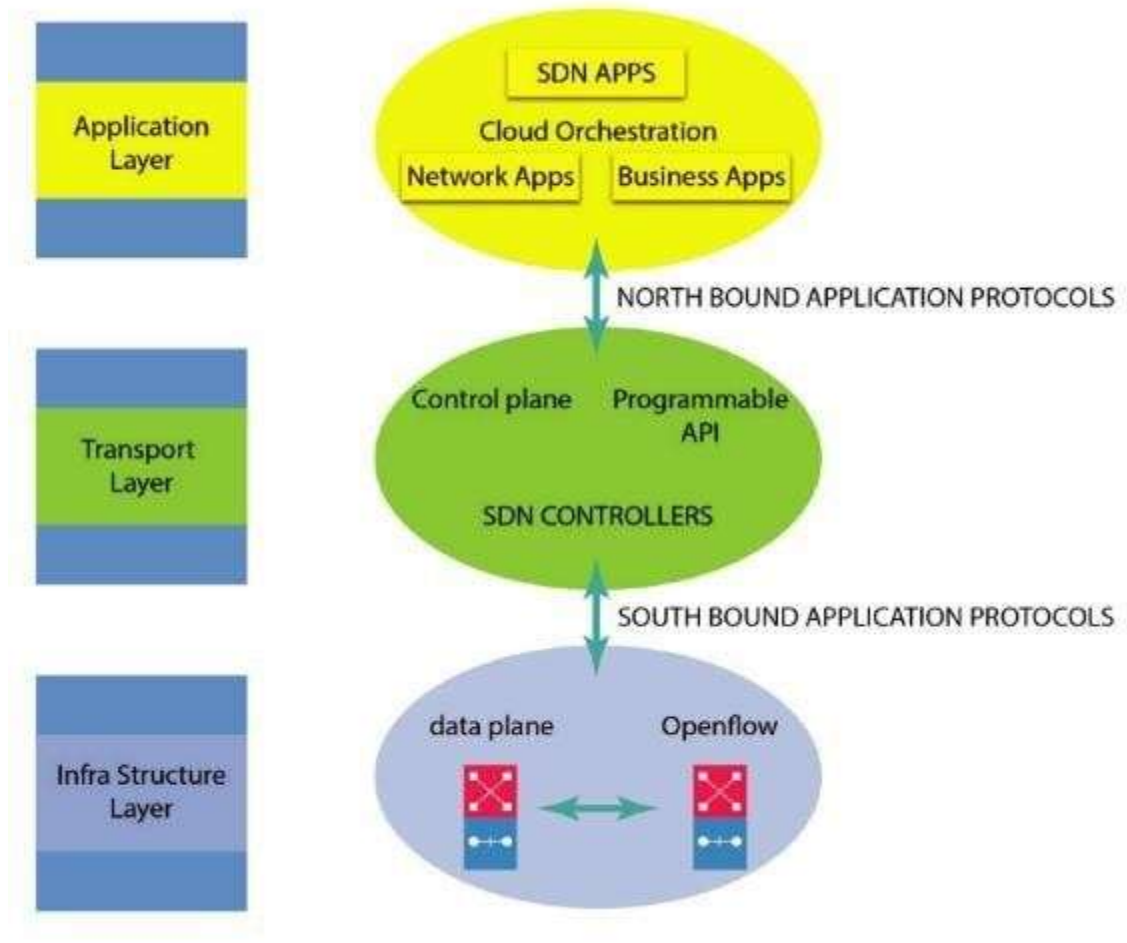


Figure: 2.4. SDN Architecture

Network Functions Virtualization

NFV allows for the separation of communication services from dedicated hardware, such as routers and firewalls. This separation means network operations can provide new services dynamically and without installing new hardware.

Deploying network components with network functions virtualization takes hours instead of months like with traditional networking. In addition, the virtualized services can run on less expensive, generic servers instead of proprietary hardware. Additional reasons to use network functions virtualization include:

Pay-as-you-go: Pay-as-you-go NFV models can reduce costs because businesses pay only for what they need.

Fewer appliances: Because NFV runs on virtual machines instead of physical machines, fewer appliances are necessary and operational costs are lower.

Scalability: Scaling the network architecture with virtual machines is faster and easier, and it does not require purchasing additional hardware.

Network functions virtualization replaces the functionality provided by individual hardware networking components. This means that virtual machines run software that accomplishes the same networking functions as the traditional hardware.

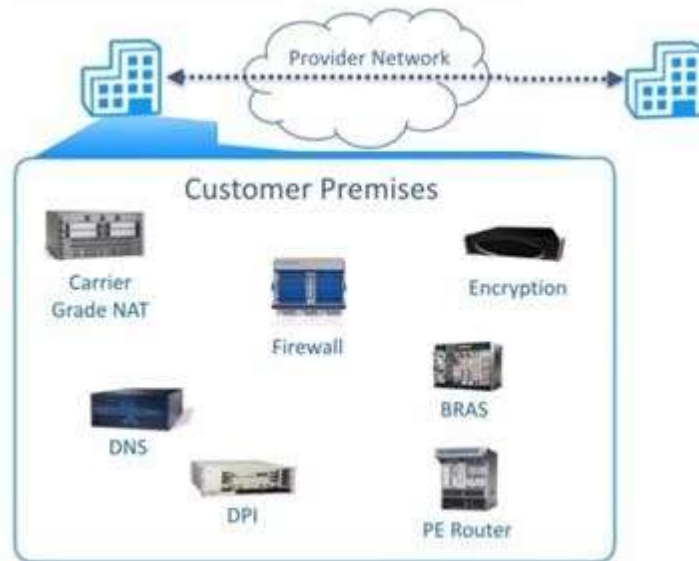


Figure: 2.5. Network Functions Virtualization

Load balancing, routing and firewall security are all performed by software instead of hardware components. A hypervisor or software-defined networking controller allows network engineers to program all of the different segments of the virtual network, and even automate the provisioning of the network. IT managers can configure various aspects of the network functionality through one pane of glass, in minutes.

Benefits of NFV

NFV allows virtual network function to run on a standard generic server, controlled by a hypervisor, which is far less expensive than purchasing proprietary hardware devices. Network configuration and management is much simpler with a virtualized network. Best of all, network functionality can be changed or added on demand because the network runs on virtual machines that are easily provisioned and managed.

Risks of network functions virtualization:

- Physical security controls are not effective
- Malware is difficult to isolate and contain
- Network traffic is less transparent
- Complex layers require multiple forms of security

NFV vs. SDN

- While NFV separates networking services from dedicated hardware appliances, software-defined networking, or SDN, separates the network control functions such as routing, policy definition and applications from network forwarding functions.
- With SDN, a virtual network control plane decides where to send traffic, enabling entire networks to be programmed through one pane of glass.
- SDN allows network control functions to be automated, which makes it possible for the network to respond quickly to dynamic workloads.
- A software-defined network can sit on top of either a virtual network or a physical network, but a virtual network does not require SDN to operate.
- Both SDN and NFV rely on virtualization technology to function.

2.4 IoT SYSTEMS MANAGEMENT

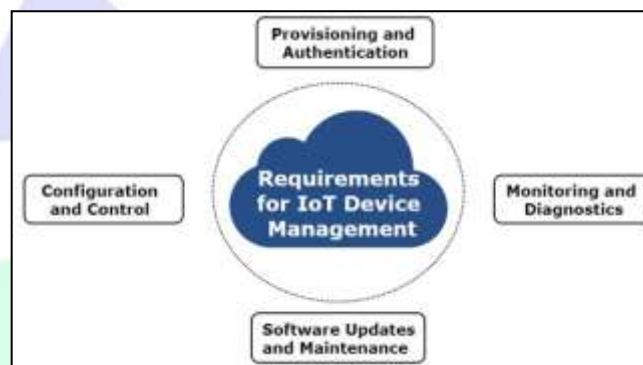


Figure 2.6. IoT System Management

1. Provisioning and Authentication

When adding new IoT devices, you want to make sure that only trusted, secure devices can be added. You wouldn't want bad actors to be able to connect devices to your IoT solution that aren't genuine, aren't running trusted software, or aren't working on the behalf of a trusted user.

- Provisioning is the process of enrolling a device into the system.
- Authentication is a step in that process of provisioning whereby you verify that only devices with the proper credentials get enrolled.

2. Configuration and Control

Devices are imperfect when they're deployed out in the field, whether that's a tracker on a mobile asset like a car or a sensor for remote monitoring like of a

refrigeration system. After deployment, there may be configurable device settings that you want to adjust over time, such as decreasing the frequency with which your trackers report position messages to increase battery life.

The ability to configure and control devices even after deployment is therefore critical to ensuring functionality, improving performance, and protecting from security threats. You may also want the ability to reset devices to their factory default configuration when you decommission them.

3. Monitoring and Diagnostics

In addition to configuring certain device settings, there may also be unforeseen operational issues and/or software bugs that you'll need to address. But to address them, you need to identify them in the first place.

Therefore, the ability to monitor and detect when something is amiss, such as higher-than-normal CPU utilization, is essential to proactively identifying and diagnosing potential bugs/issues. Device management software can provide program logs needed to make diagnoses.

4. Software Maintenance and Updates

If we do identify a bug with your devices and/or a security flaw, we will need to be able to make updates to device software (or even firmware) from afar. With thousands or millions of devices, getting physical access to each device to update them manually just isn't practically possible. If physical access is required, your IoT solution may be doomed, or at least very brittle and precarious in the end.

The ability to update and maintain remote device software securely is thus one of the most important components of good device management.

2.5 SNMP

- SNMP stands for **Simple Network Management Protocol**.
- SNMP is a framework used for managing devices on the internet.
- It provides a set of operations for monitoring and managing the internet.
- SNMP has two components Manager and agent.
- The manager is a host that controls and monitors a set of agents such as routers.
- It is an application layer protocol in which a few manager stations can handle a set of agents.

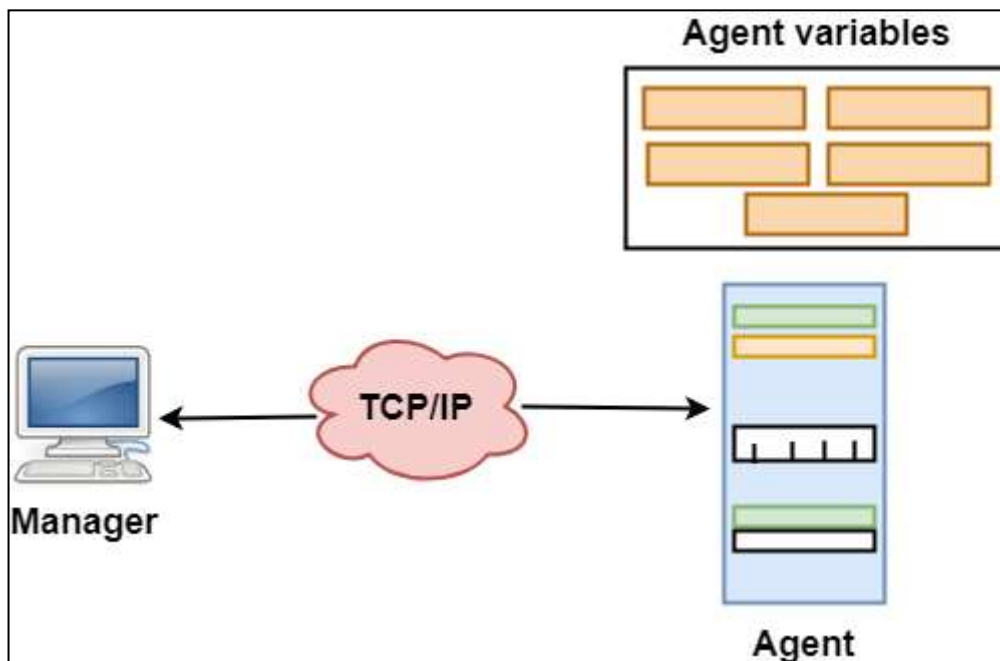


Figure: 2.7. Simple Network Management Protocol

- The protocol designed at the application level can monitor the devices made by different manufacturers and installed on different physical networks.
- It is used in a heterogeneous network made of different LANs and WANs connected by routers or gateways.

Management with SNMP has three basic ideas:

- A manager checks the agent by requesting the information that reflects the behaviour of the agent.
- A manager also forces the agent to perform a certain function by resetting values in the agent database.
- An agent also contributes to the management process by warning the manager regarding an unusual condition.

Management Components

- Management is not achieved only through the SNMP protocol but also the use of other protocols that can cooperate with the SNMP protocol. Management is achieved through the use of the other two protocols: SMI (Structure of management information) and MIB(management information base).

- Management is a combination of SMI, MIB, and SNMP. All these three protocols such as abstract syntax notation 1 (ASN.1) and basic encoding rules (BER).

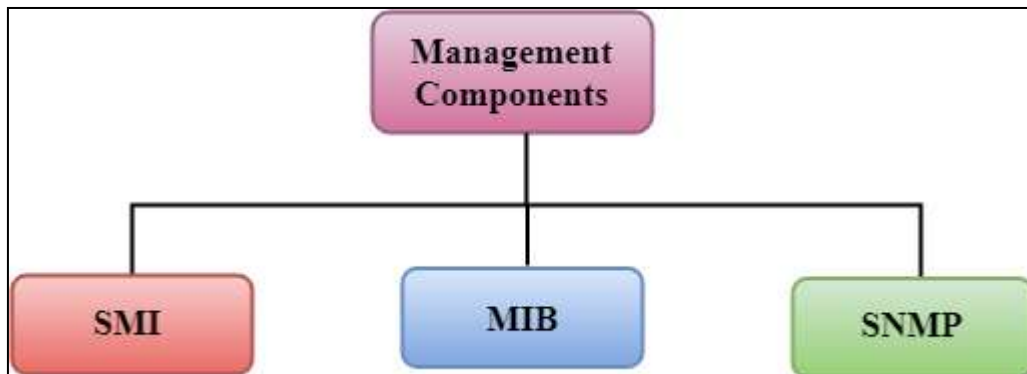


Figure: 2.8. Management Components

SMI

The SMI (Structure of Management Information) is a component used in network management. Its main function is to define the type of data that can be stored in an object and to show how to encode the data for the transmission over a network.

MIB

- The MIB (Management information base) is a second component for the network management.
- Each agent has its own MIB, which is a collection of all the objects that the manager can manage. MIB is categorized into eight groups: system, interface, address translation, IP, ICMP, TCP, UDP, and EGP. These groups are under the MIB object.

SNMP

SNMP defines five types of messages: GetRequest, GetResponse, GetNextRequest, SetRequest, and Trap.

- **GetRequest:** The GetRequest message is sent from a manager (client) to the agent (server) to retrieve the value of a variable.
- **GetResponse:** The GetResponse message is sent from an agent to the manager in response to the GetRequest and GetNextRequest message. This message contains the value of a variable requested by the manager.

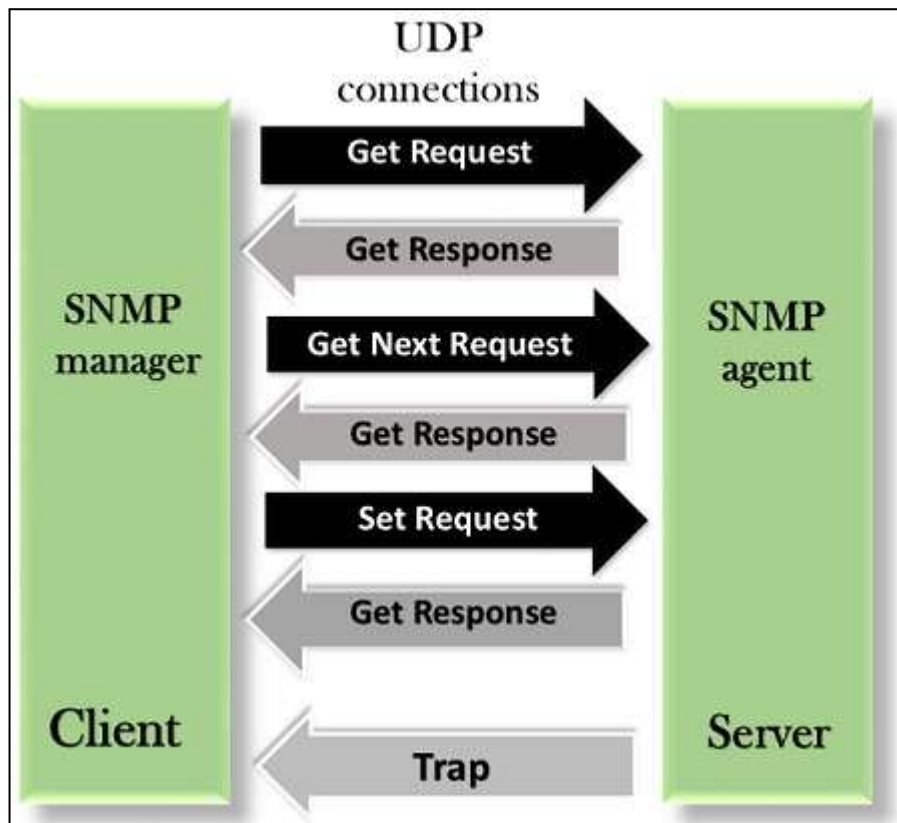


Figure: 2.9. SNMP

- **GetNextRequest:** The GetNextRequest message is sent from the manager to agent to retrieve the value of a variable. This type of message is used to retrieve the values of the entries in a table. If the manager does not know the indexes of the entries, then it will not be able to retrieve the values. In such situations, GetNextRequest message is used to define an object.
- **SetRequest:** The SetRequest message is sent from a manager to the agent to set a value in a variable.
- **Trap:** The Trap message is sent from an agent to the manager to report an event. For example, if the agent is rebooted, then it informs the manager as well as sends the time of rebooting.

2.6 YANG

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

Key YANG Capabilities

- Human readable, easy to learn representation
- Hierarchical configuration data models
- Reusable types and groupings (structured types)
- Extensibility through augmentation mechanisms
- Supports the definition of operations (RPCs)
- Formal constraints for configuration validation
- Data modularity through modules and submodules
- Versioning rules and development support

YANG models data using a hierarchical, tree-based structure with nodes. YANG defines four nodes types. Each node has a name, and depending on the node type, the node might either define a value or contain a set of child nodes. The nodes types are:

- **Leaf node**—Contains a single value of a specific type
- **Leaf-list node**—Contains a sequence of leaf nodes
- **Container node**—Contains a grouping of related nodes containing only child nodes, which can be any of the four node types
- **List node**—Contains a sequence of list entries, each of which is uniquely identified by one or more key leafs

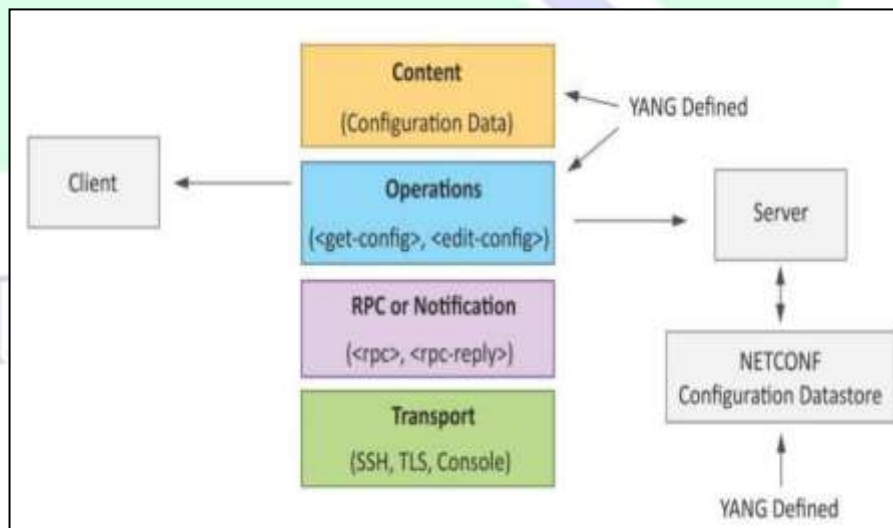


FIGURE: 2.9. YANG with NETCONF protocol

Example code for Yang code for User define data type:

```
typedef dotted-quad
{
    type string
    {
        Pattern '([([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'+ '([0-9]|[1-
9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])';
    }
    description "Four octets written as decimal numbers and separated with the '.'
(full stop) character.";
}
```

2.7 NETOPEER

NETCONF **provides mechanisms to install, manipulate, and delete the configuration of network devices** The NETCONF protocol uses an Extensible Markup Language (XML) based data encoding for the configuration data as well as the protocol messages. The protocol messages are exchanged on top of a secure transport protocol

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.

The NETCONF Client Model supports both the **SSH and TLS transport** protocols, using the SSH client and TLS client groupings defined in [I-D. ietf-netconf-ssh-client-server] and [I-D. ietf-netconf-tls-client-server] respectively.

NETCONF on the other hand is designed **for device configuration** and provides network-wide transactions, making it possible to change configuration for all devices in the network at once. A change will either succeed on all devices or it will not be activated at all.

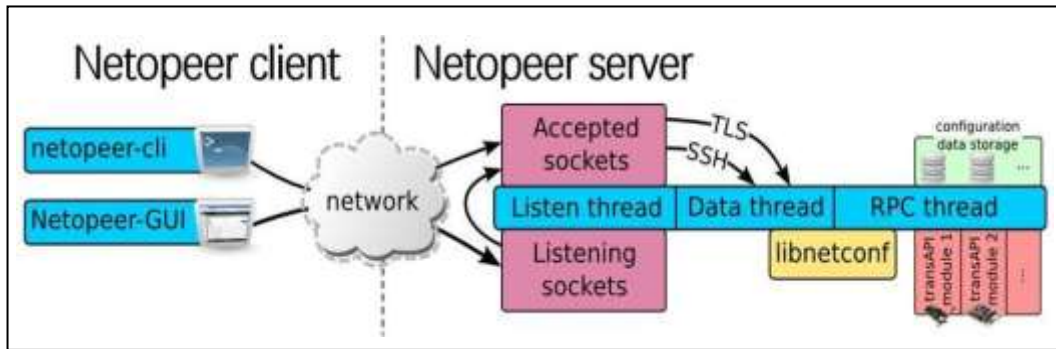


FIGURE: 2.10. Netopeer client and Server

Points to Remember:

- Machine to machine data transmission
- Network connectivity in IOT with M2M
- Software Defined network protocol
- Network function virtualization protocol
- Basic needs of IOT System management
- SNMP
- YANG data modelling language
- NETOPEER tools for NETCONF code execution

Important Questions:

1. Define machine to machine in IOT.
2. How to establish the connection between device to device.
3. What is meant by Software defined network protocol
4. Explain about Network function virtualization protocol with example diagram
5. Explain the simple network management protocol.
6. What is YANG? Give some example data modelling code in YANG.

UNIT III: 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.

3.1 IOT PLATFORMS DESIGN METHODOLOGY

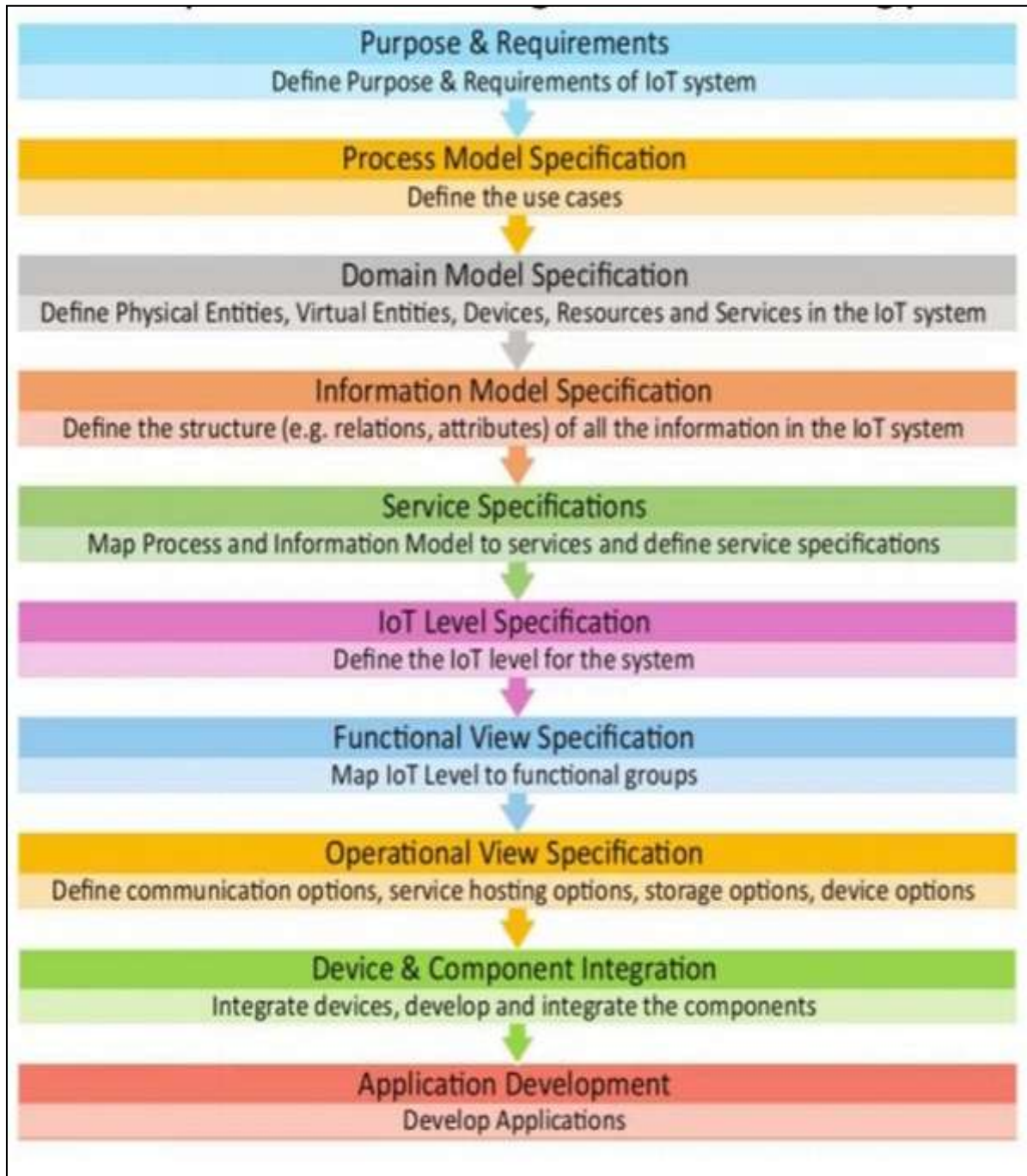


Figure: 3.1. Steps involving in Design Methodology

Step 1 : Purpose and Requirement Specification

Defines:

- System purpose
- behavior and
- Requirements (such as data collection requirements, data analysis requirement, system management requirements, data privacy and security requirements, User interfaces requirements)
 - **Purpose** : An automated irrigation mechanism which turns the pumping motor ON and OFF on detecting the moisture content of the earth without the intervention of human
 - **Behavior**: System should monitor the amount of soil moisture content in soil. In case the soil moisture of the soil deviates from the specified range, the watering system is turned ON/OFF. In case of dry soil, it will activate the irrigation system, pumping water for watering the plants.
 - **System Management Requirements** : system should remotely provide monitoring and control functions
 - **Data Analysis Requirements** : system should perform local analysis of data
 - **Application Deployment Requirement**: Deployed locally on device, but acts remotely without manual intervention.
 - **Security** : Authentication to Use the system must be available.

3.2 PURPOSE AND SPECIFICATION

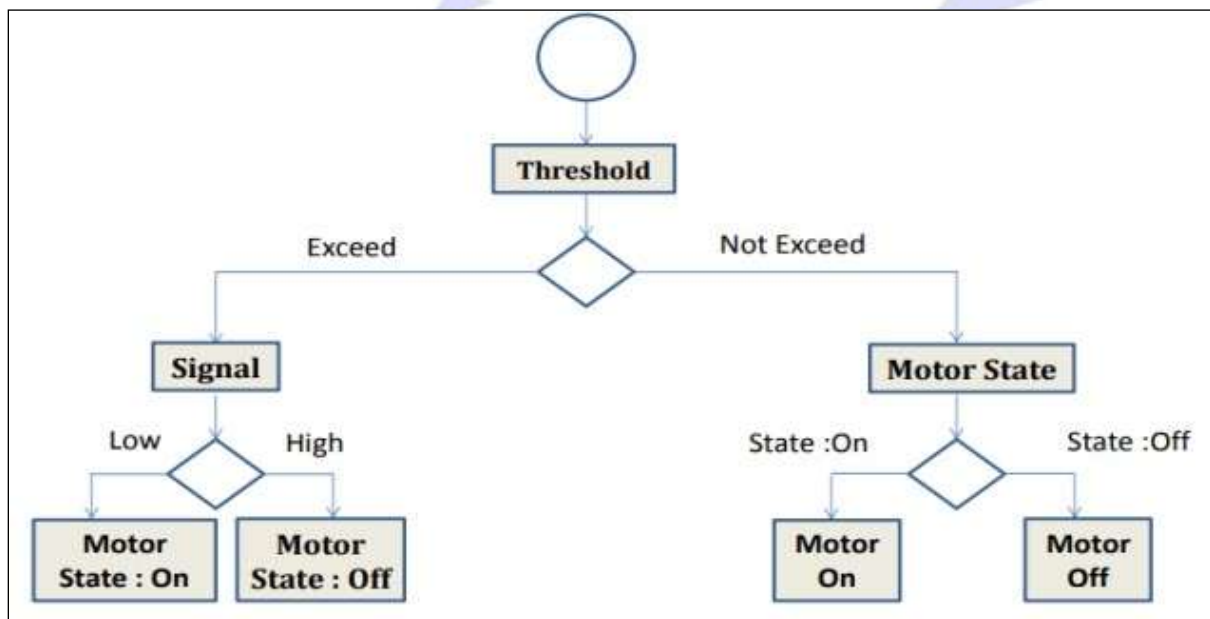
The first step in IoT system design methodology is to define the purpose and requirements of the system.

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.

3.3 PROCESS SPECIFICATION

Step 2 :

- Define the process with the help of use cases
- The use cases are formally described based on Purpose & requirement specification In this
- Use case: – Circle denotes a state or an attribute
- 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.

**Figure: 3.2. Process Specification****3.4 DOMAIN MODEL SPECIFICATION****Step 3 : Domain Model Specification**

- Describes the main concepts, entities and objects in the domain of IoT system to be designed
- It defines the attributes of the objects and relationships between them
- Entities, Objects and Concepts include the following: Physical entity, Virtual entity, Device, Resource, Service

• **Physical Entity:**

- Discreet identifiable entity in physical environment
- For Example Pump, motor, LCD
- The IoT System provides the information about the physical entity (using sensors) or performs actuation upon the Physical entity (like switching a motor on etc.)
- In smart irrigation example, there are three Physical entities involved :
 - Soil (whose moisture content is to be monitored)
 - Motor (to be controlled)
 - Pump (To be controlled)

• **Virtual Entity:**

- Representation of physical entity in digital world
- For each physical entity there is a virtual entity

• **Device:**

- Medium for interactions between Physical and Virtual Entities.
- Devices (Sensors) are used to gather information from the physical entities
- Devices are used to identify Physical entities (Using Tags)
- In Smart Irrigation System, device is soil moisture sensor and buzzer as well as the actuator (relay switch) attached to it.

• **In smart irrigation system there are three services :**

- A service that sets the signal to low/ high depending upon the threshold value
- A service that sets the motor state on/off
- A controller service that runs and monitors the threshold value of the moisture and switches the state of motor on/off depending upon it. When threshold value is not crossed the controller retrieves the motor status from database and switches the motor on/off.

3.5 INFORMATION MODEL SPECIFICATION

Step 3:

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

- Defines the structure of all the information in the IoT system (such as attributes, relations etc.)
- It does not describe the specifics of how the information is represented or stored.
- This adds more information to the Virtual entities by defining their attributes and relations
- I: e, Draw Class diagram

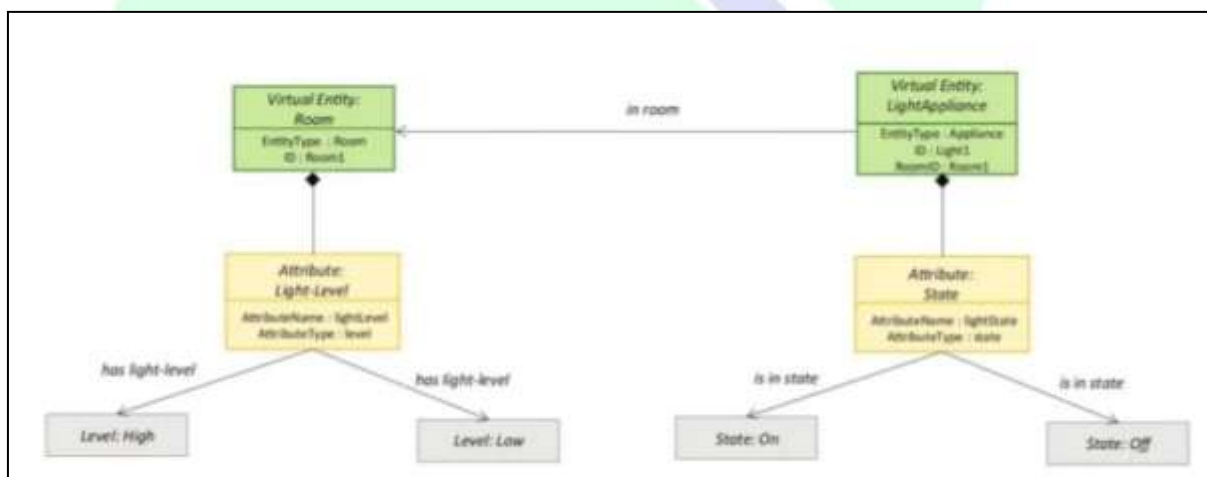


Figure: 3.3. Information Model Specification

- 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.

3.6 SERVICE SPECIFICATION

Define the services in IoT System, service types, service inputs/outputs, service endpoints, service schedules, service preconditions and service effects

- Services can be controller service, Threshold service, state service for smart irrigation system
- These services either change the state/attribute values or retrieve the current values.

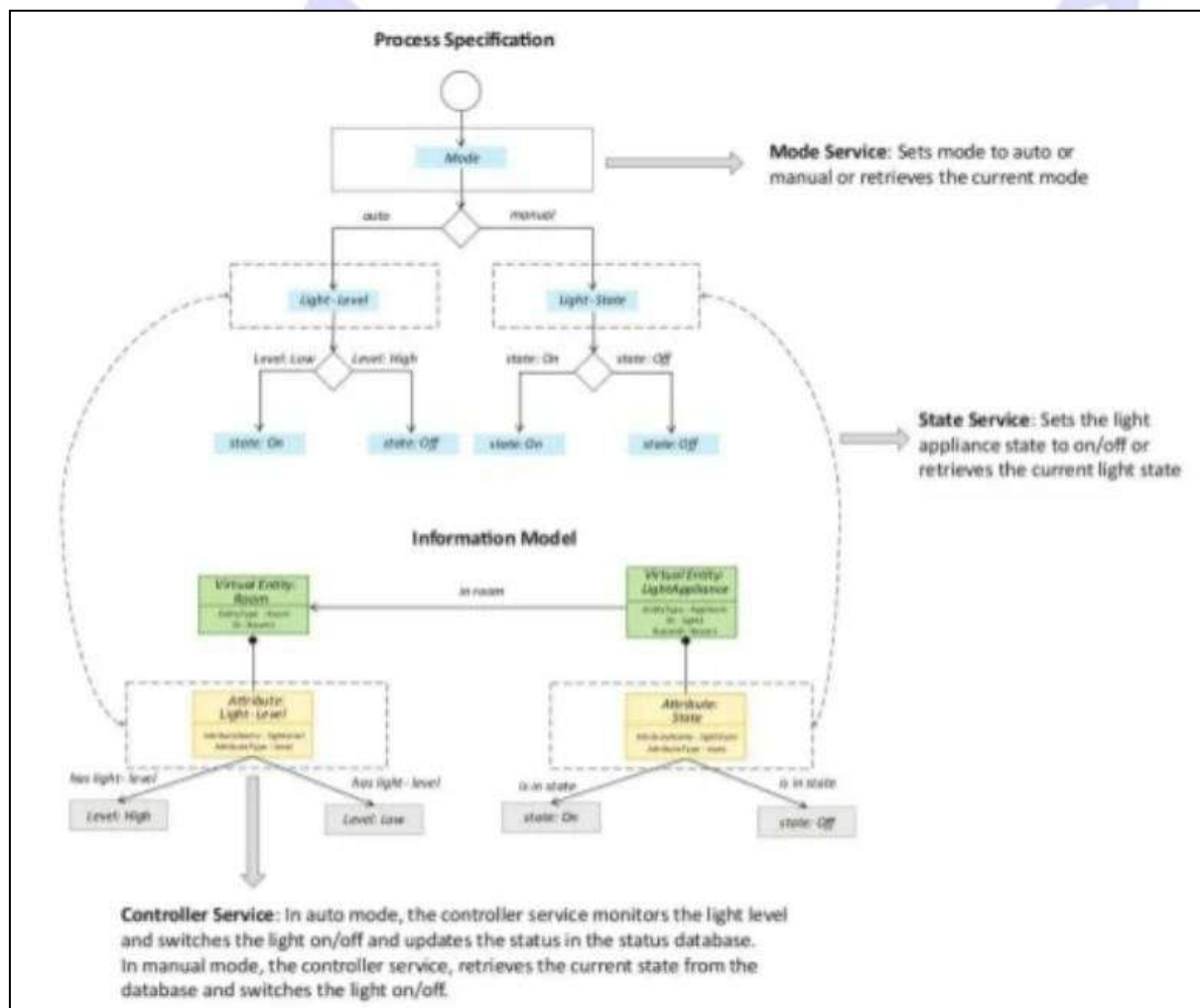
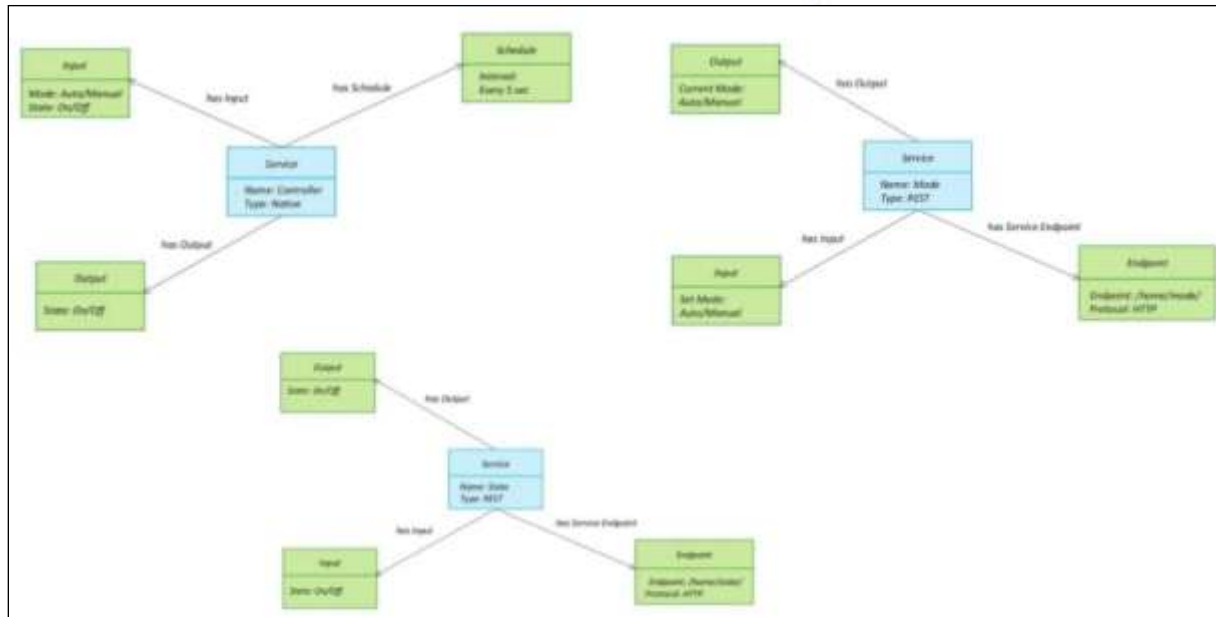


Figure: 3.4. Service Specification

• For example,

- Threshold service sets signal to high or low depending upon the soil moisture value.
- State service sets the motor state: on or off
- Controller service monitors the threshold value as well as the motor state and switches the motor on/off and updates the status in the database.

**Figure: 3.5. Service Specification****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.

3.7 IOT LEVEL SPECIFICATION**Step 6 : IoT Level Specification**

- Decide the deployment level of IoT System. Here I am using Deployment Level 1.

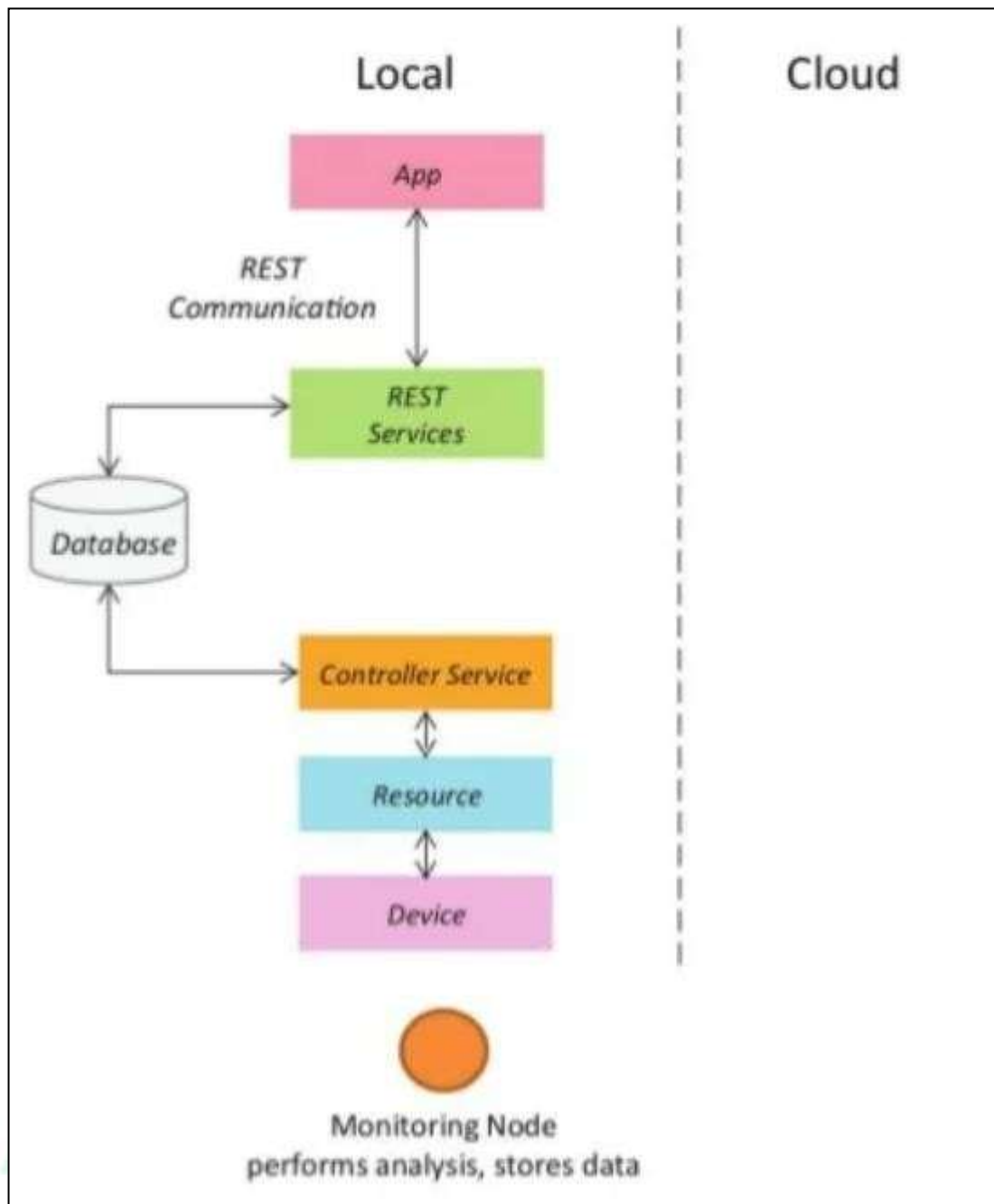


Figure: 3.6. IoT Level Specification

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.

3.8 FUNCTIONAL VIEW SPECIFICATION

Step 7 : Functional View Specification

- Define the functions of IoT System grouped into various functional groups.

- These functional groups provide functionalities for interacting with the concepts defined in Domain model specification.

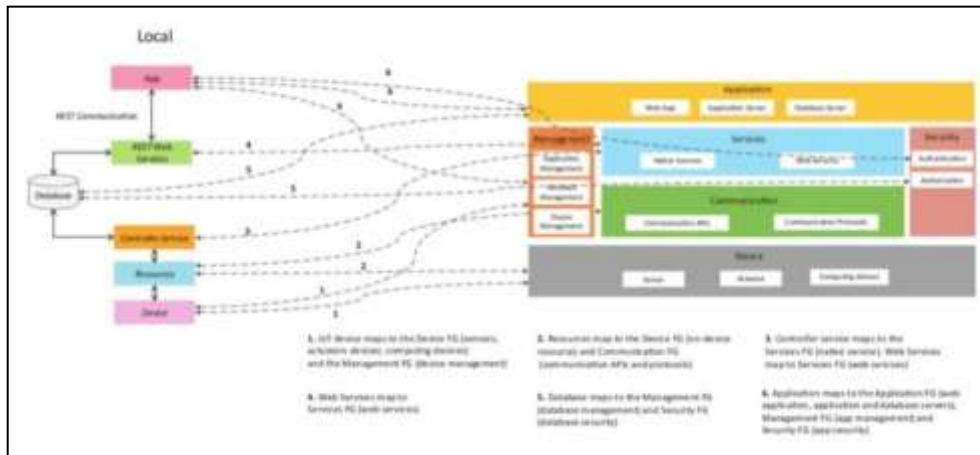


Figure: 3.7. Functional View Specification

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.

The functional group (FG) included in a functional view include:

Device: The device FG contains devices for monitoring and control. In the home automation example, the device FG includes a single board mini-computer, a light.

Communication: The communication FG handles the communication for the IoT system. The communication FG includes the communication protocols that form the backbone of IoT systems and enable network connectivity.

Communication protocols include. IPv4, IPv6, TCP, HTTP, 802.11

Services: the services FG includes various services involved in the IoT system such as services for device monitoring, device control services, data publishing services and services for device discovery.

Management: The security FG includes applications that provide an interface to the users to control and monitor various aspects of the IoT system. Applications also allow user to view the system status and the processed data.

3.9 OPERATIONAL VIEW SPECIFICATION

Step 8 : Operational View Specification

- Define the Operations/options related to IoT System development
- Such as Device options, Storage options, Application hosting option

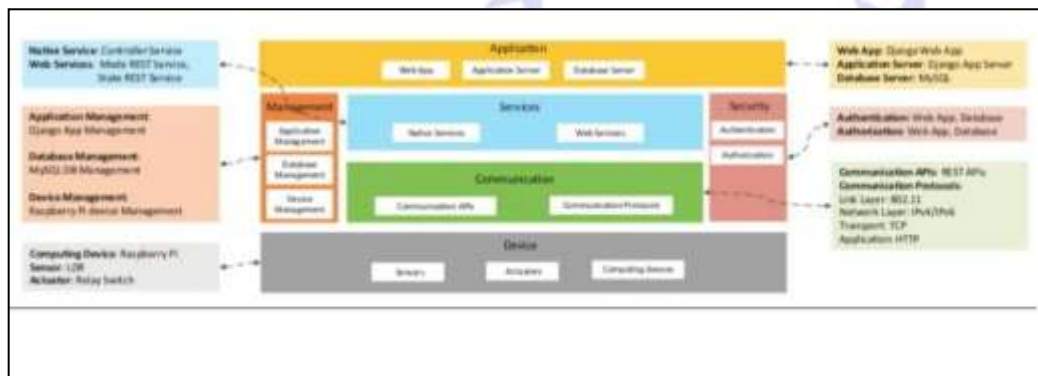


Figure: 3.8. Operational View Specification

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.

Applications:

Web applications: Django web applications

Application Server: Django App Server

Database server: MySQL

3.10 DEVICE AND COMPONENT INTEGRATORS

Step 9: Device and Component Integration

- Integrates the devices and components and draw a schematic diagram showing the same.

Step 9: Device & Component Integration

- The ninth step in the IoT design methodology is the integration of the devices and components.

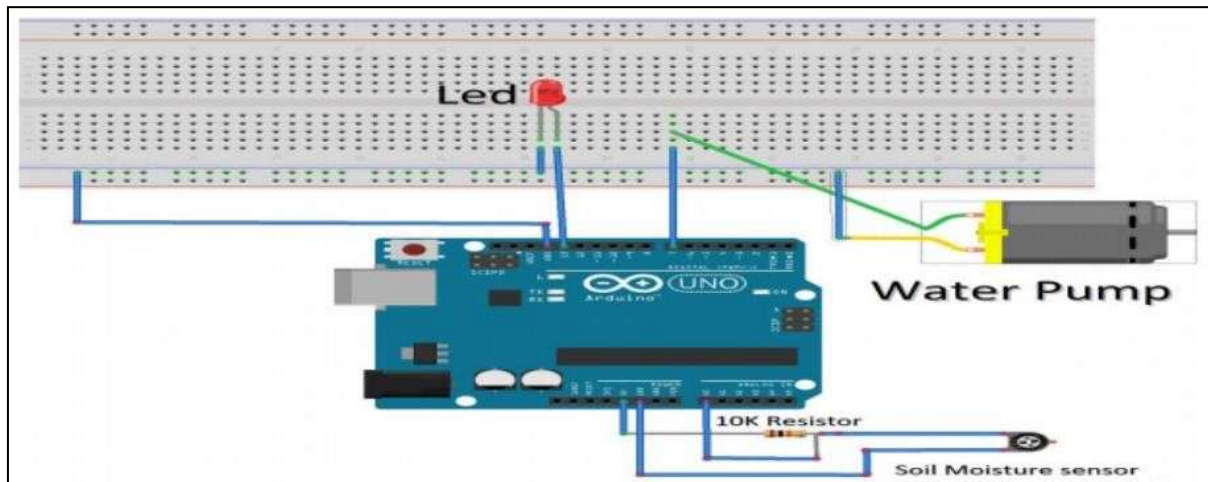


Figure: 3.9. Device and Component

3.11 APPLICATION DEVELOPMENT

Step 10: Application development

- GUI / Screenshot of IoT Application

Step 10: Application Development

- The final step in the IoT design methodology is to develop the IoT application.

Important questions:

1. Explain the about IoT platforms design Methodology.
2. Define the purpose and specification of IOT platform design.
3. Elaborate the process specification of IOT platform design
4. Describe the Domain model specification of IOT platform design
5. Functional view specification of IoT platform design
6. Device and component Integrators of IoT platform design.

UNIT IV: 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.

4.1 Logical design using python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of **Python Programming** –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.

- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Applications of Python

Python is one of the most widely used language over the web.

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

4.1.1 Installing python

Step 1 – Select Version of Python to Install

Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available.

Step 2 – Download Python Executable Installer

On the web browser, in the official site of python (www.python.org), move to the Download for Windows section.

All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.9.1 version.



Looking for a specific release?
Python releases by version number:

Release version	Release date		Click for more
Python 3.8.7	Dec 21, 2020	 Download	Release Notes
Python 3.8.5	Dec 7, 2020	 Download	Release Notes
Python 3.8.0	Oct 5, 2020	 Download	Release Notes
Python 3.8.0	Sept 24, 2020	 Download	Release Notes
Python 3.8.10	Sept 5, 2020	 Download	Release Notes
Python 3.7.9	Aug 17, 2020	 Download	Release Notes
Python 3.6.12	Aug 17, 2020	 Download	Release Notes
Python 3.6.8	Jul 28, 2020	 Download	Release Notes

[View older releases](#)

On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer, which suits your system operating system, and download the installer. Let suppose, we select the Windows installer (64 bits).

STARS RISE HERE...

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		429ae95d24278fa1568684fad8fca7	25372998	51G
XZ compressed source tarball	Source release		61991498e75ac8f00adc09082615adb6	18897104	51G
macOS 64-bit Intel installer	Mac OS X	for macOS 10.9 and later	74f5cc5b5783ce8fb2ca55f11f3f0699	29795899	51G
macOS 64-bit universal2 installer	Mac OS X	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental)	8b19748473609241e60aa3618bba0ed	37451735	51G
Windows embeddable package (32-bit)	Windows		96cfa811e8b650e68c3dd4c1258ae317	7571141	51G
Windows embeddable package (64-bit)	Windows		e70e5c22432d857a497cde5ec2a5ce2	8482331	51G
Windows help file	Windows		c49dfb6e88c031ed0e2d39bc42b316	8787443	51G
Windows installer (32-bit)	Windows		dde210ea04a31c27486695a9e7cd297a	27126136	51G
Windows installer (64-bit)	Windows	Recommended	b3f6c2ed88bc315ad2bc49eae48a94487	28204528	51G

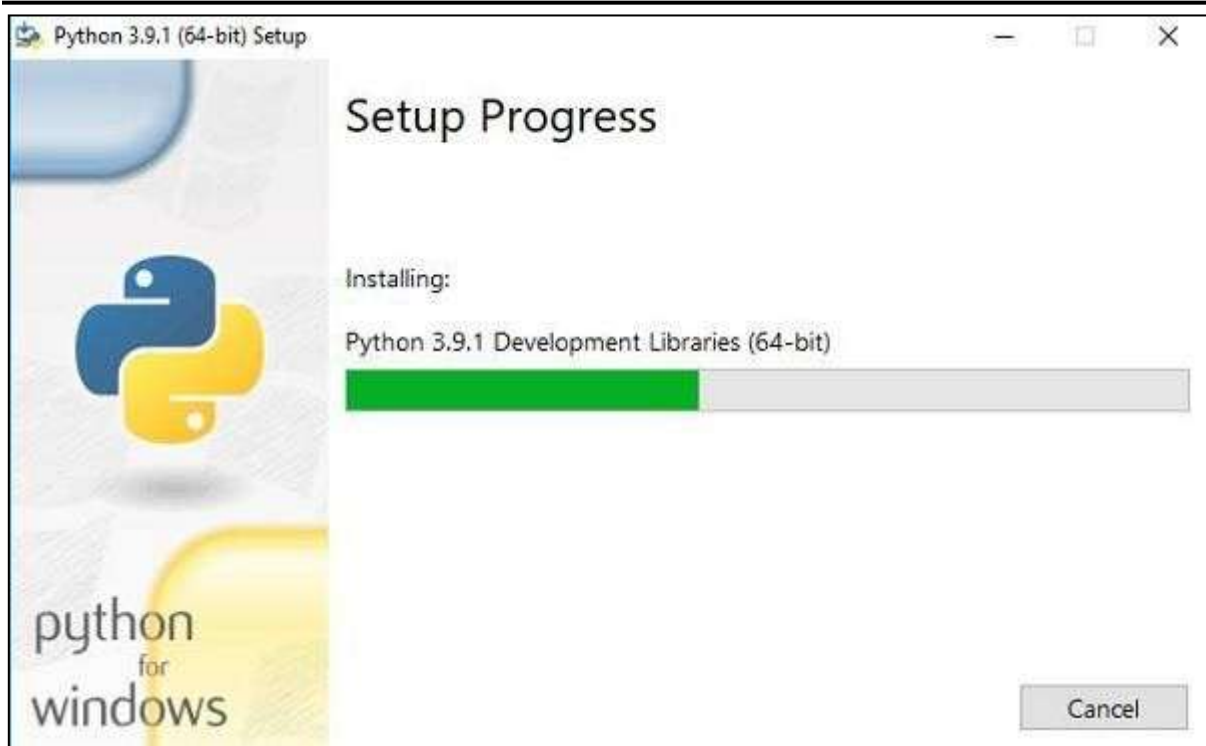
Step 3 – Run Executable Installer

We downloaded the Python 3.9.1 Windows 64 bit installer.

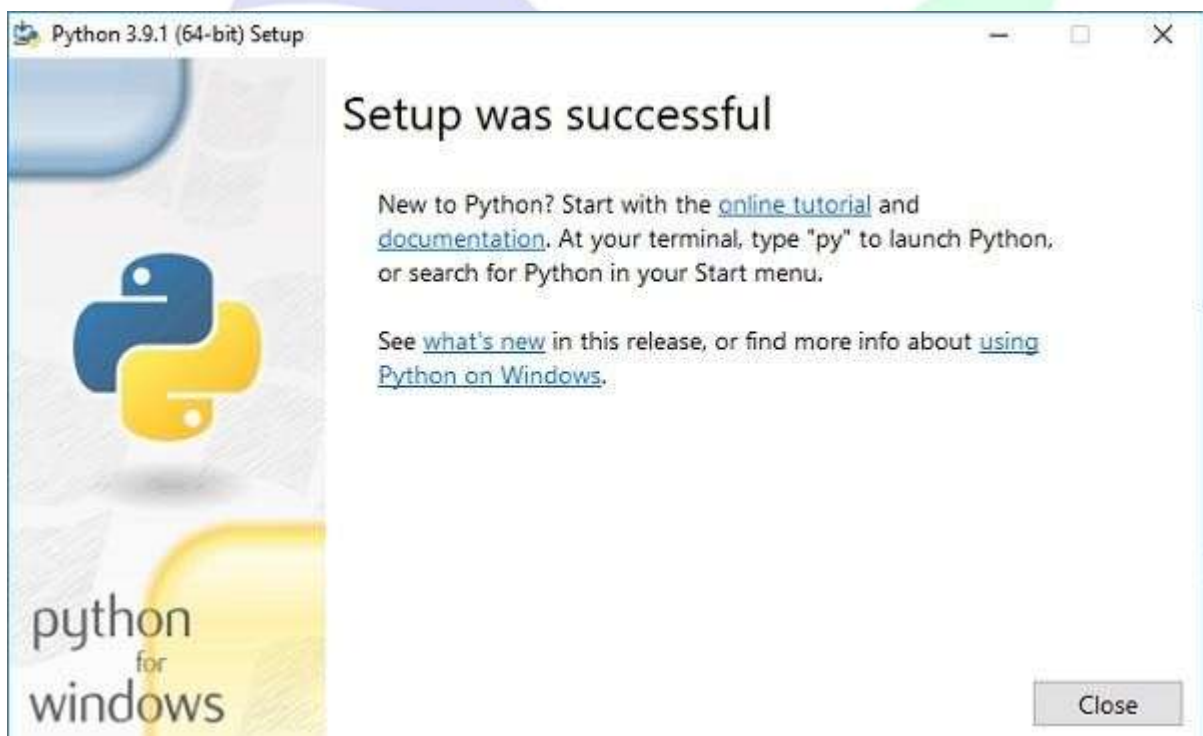
Run the installer. Make sure to select both the checkboxes at the bottom and then click Install New.



On clicking the Install Now, The installation process starts.



The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.



Step 4 – Verify Python is installed on Windows

To ensure if Python is successfully installed on your system. Follow the given steps –

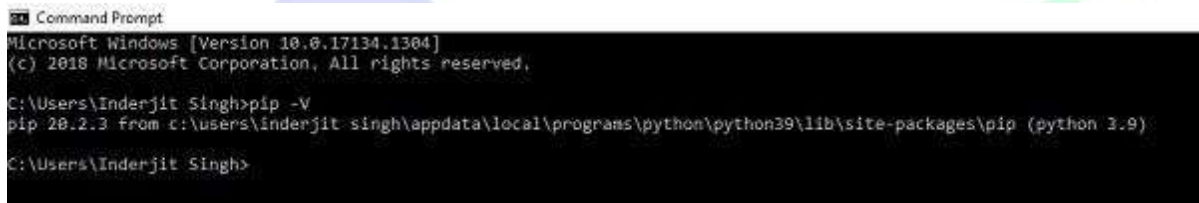
- Open the command prompt.
- Type 'python' and press enter.
- The version of the python which you have installed will be displayed if the python is successfully installed on your windows.

Step 5 – Verify Pip was installed

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps –

- Open the command prompt.
- Enter pip -V to check if pip was installed.
- The following output appears if pip is installed successfully.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Inderjit Singh>pip -V
pip 20.2.3 from c:\users\inderjit singh\appdata\local\programs\python\python39\lib\site-packages\pip (python 3.9)

C:\Users\Inderjit Singh>
```

We have successfully installed python and pip on our Windows system.

3.1. Type conversions

The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages.

```
$ python
Python 3.7.1 (#1, Nov 11 2021, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!");**. However in Python version 2.4.3, this produces the following result –

Hello, Python!

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension **.py**. Type the following source code in a test.py file –

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows –

```
$ python test.py
```

This produces the following result –

Hello, Python!

Assigning values to variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example –

```
#!/usr/bin/python
```

```
counter = 100      # An integer assignment
```

```
miles  = 1000.0    # A floating point
```

```
name   = "John"    # A string
```

```
print counter
```

```
print miles
```

```
print name
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result –

```
100
```

```
1000.0
```

```
John
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example –

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example –

```
a,b,c = 1,2,"john"
```

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

- Numbers
- String
- List
- Tuple

- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example –

```
var1 = 1
```

```
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is –

```
del var1[,var2[,var3[...varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example –

```
del var
```

```
del var_a, var_b
```

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers –

INT	LONG	FLOAT	COMPLEX
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example –

```
#!/usr/bin/python

str = 'Hello World!'

print str      # Prints complete string
print str[0]   # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
print str[2:]  # Prints string starting from 3rd character
print str * 2  # Prints string two times
print str + "TEST" # Prints concatenated string
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list      # Prints complete list
print list[0]   # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:]  # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
```



```
print list + tinylist # Prints concatenated lists
```

This produce the following result –

```
['abcd', 786, 2.23, 'john', 70.2]
```

```
abcd
```

```
[786, 2.23]
```

```
[2.23, 'john', 70.2]
```

```
[123, 'john', 123, 'john']
```

```
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as **read-only** lists. For example –

```
#!/usr/bin/python
```

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
```

```
tinytuple = (123, 'john')
```

```
print tuple          # Prints the complete tuple
```

```
print tuple[0]       # Prints first element of the tuple
```

```
print tuple[1:3]     # Prints elements of the tuple starting from 2nd till 3rd
```

```
print tuple[2:]      # Prints elements of the tuple starting from 3rd element
```

```
print tinytuple * 2  # Prints the contents of the tuple twice
```

```
print tuple + tinytuple # Prints concatenated tuples
```

This produce the following result –

```
['abcd', 786, 2.23, 'john', 70.2]
```

```
abcd
```

```
(786, 2.23)
```

```
(2.23, 'john', 70.2)
```

```
(123, 'john', 123, 'john')
```

```
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example –

```
#!/usr/bin/python

dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"

tinydict = {'name': 'john', 'code': 6734, 'dept': 'sales'}

print dict['one']    # Prints value for 'one' key
print dict[2]        # Prints value for 2 key
print tinydict       # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

This produce the following result –

```
This is one
```

```
This is two
```

```
{'dept': 'sales', 'code': 6734, 'name': 'john'}
```

```
['dept', 'code', 'name']
```

```
['sales', 6734, 'john']
```

Data Type Conversion

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Sr.No.	Function & Description
1	int(x [,base]) Converts x to an integer. base specifies the base if x is a string.
2	long(x [,base]) Converts x to a long integer. base specifies the base if x is a string.
3	float(x) Converts x to a floating-point number.
4	complex(real [,imag]) Creates a complex number.
5	str(x) Converts object x to a string representation.
6	repr(x) Converts object x to an expression string.
7	eval(str) Evaluates a string and returns an object.
8	tuple(s) Converts s to a tuple.
9	list(s) Converts s to a list.
10	set(s) Converts s to a set.
11	dict(d) Creates a dictionary. d must be a sequence of (key,value) tuples.
12	frozenset(s) Converts s to a frozen set.
13	chr(x) Converts an integer to a character.
14	unichr(x) Converts an integer to a Unicode character.
15	ord(x) Converts a single character to its integer value.
16	hex(x) Converts an integer to a hexadecimal string.
17	oct(x) Converts an integer to an octal string.

4.2 Control flow

Decision-making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions, which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

Sr.No.	Statement & Description
1	<u>if statements</u> An if statement consists of a Boolean expression followed by one or more statements.
2	<u>if...else statements</u> An if statement can be followed by an optional else statement , which executes when the Boolean expression is FALSE.
3	<u>nested if statements</u> You can use one if or else if statement inside another if or else if statement(s).

Simple if:

Here is an example of a **one-line if** clause –

```
#!/usr/bin/python  
var = 100
```

```
if ( var == 100 ) : print "Value of expression is 100"  
print "Good bye!"
```

When the above code is executed, it produces the following result –

Value of expression is 100

Good bye!

Elif

The elif keyword is python's way of saying "if the previous conditions were not true, then try this condition".

Example

```
a = 33  
b = 33  
if b > a:  
    print("b is greater than a")  
elif a == b:  
    print("a and b are equal")
```

In this example **a** is equal to **b**, so the first condition is not true, but the **elif** condition is true, so we print to screen that "a and b are equal".

Else

The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200  
b = 33  
if b > a:  
    print("b is greater than a")  
elif a == b:  
    print("a and b are equal")
```

else:

```
print("a is greater than b")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
```

```
b = 330
```

```
print("A") if a > b else print("B")
```

Nested If

You can have if statements inside if statements, this is called *nested if* statements.

Example:

```
x = 41
```

```
if x > 10:
```

```
    print("Above ten,")
```

```
    if x > 20:
```

```
        print("and also above 20!")
```

```
    else:
```

```
        print("but not above 20.")
```

Python - Loops

Python programming language provides following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	<u>while loop</u> Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	<u>for loop</u> Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	<u>nested loops</u> You can use one or more loop inside any another while, for or do..while loop.

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

```
for x in "banana":  
    print(x)
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the `range()` function:

```
for x in range(6):
```

```
    print(x)
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
```

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in adj:
```

```
    for y in fruits:
```

```
        print(x, y)
```

4.3 Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the **def** keyword:

Example

```
def my_function():
```

```
    print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():  
    print("Hello from a function")  
my_function()
```

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
def my_function(fname):  
    print(fname + " Refsnes")  
  
my_function("Emil")  
my_function("Tobias")  
my_function("Linus")
```

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a *** before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly:

Example

If the number of arguments is unknown, add a `*` before the parameter name:

```
def my_function(*kids):  
    print("The youngest child is " + kids[2])
```

```
my_function("Emil", "Tobias", "Linus")
```

Keyword Arguments

You can also send arguments with the *key = value* syntax.

This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):  
    print("The youngest child is " + child3)
```

```
my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

Example

```
def my_function(country = "Norway"):  
    print("I am from " + country)
```

```
my_function("Sweden")
```

```
my_function("India")
```

```
my_function()
```

```
my_function("Brazil")
```


Python Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda arguments : expression

The expression is executed and the result is returned:

Example

Add 10 to argument **a**, and return the result:

```
x = lambda a : a + 10
```

```
print(x(5))
```

Lambda functions can take any number of arguments:

Example

Multiply argument **a** with argument **b** and return the result:

```
x = lambda a, b : a * b
```

```
print(x(5, 6))
```

4.4 Modules

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension **.py**:

Example

Save this code in a file named **mymodule.py**

```
def greeting(name):
```

```
    print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the **import** statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule  
mymodule.greeting("Jonathan")
```

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file **mymodule.py**

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule  
  
a = mymodule.person1["age"]  
print(a)
```

Naming a Module

We can name the module file whatever you like, but it must have the file extension **.py**

Re-naming a Module

We can create an alias when you import a module, by using the **as** keyword:

Example

Create an alias for `mymodule` called `mx`:

```
import mymodule as mx
```

```
a = mx.person1["age"]
```

```
print(a)
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the `platform` module:

```
import platform
```

```
x = platform.system()
```

```
print(x)
```

Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The `dir()` function:

Example

List all the defined names belonging to the `platform` module:

```
import platform
```

```
x = dir(platform)
```

```
print(x)
```

Import From Module

We can choose to import only parts from a module, by using the `from` keyword.

Example

The module named `mymodule` has one function and one dictionary:

```
def greeting(name):  
    print("Hello, " + name)
```

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

4.5 File handling

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The open Function

Before you can read or write a file, we have to open it using Python's built-in `open()` function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details –

- **file_name** – The `file_name` argument is a string value that contains the name of the file that you want to access.
- **access_mode** – The `access_mode` determines the mode in which the file has to be opened, i.e., read, write, append, etc.
- **buffering** – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file.

Here is a list of the different modes of opening a file –

Sr.No.	Mode & Description
1	r : Opens a file for reading only.
2	Rb : Opens a file for reading only in binary format.

AJK COLLEGE OF ARTS AND SCIENCE

Navakkarai, Coimbatore - 641 105

3	r+ : Opens a file for both reading and writing.
4	rb+ :Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	W : Opens a file for writing only.
6	Wb : Opens a file for writing only in binary format.
7	w+ :Opens a file for both writing and reading.
8	wb+ : Opens a file for both writing and reading in binary format.
9	A : Opens a file for appending. The file pointer is at the end of the file if the file exists.
10	Ab : Opens a file for appending in binary format.
11	a+ : Opens a file for both appending and reading.
12	ab+ :Opens a file for both appending and reading in binary format.

```
#!/usr/bin/python3  
  
# Open a file  
fo = open("foo.txt", "wb")  
print ("Name of the file: ", fo.name)  
print ("Closed or not : ", fo.closed)  
print ("Opening mode : ", fo.mode)  
fo.close()
```

This produces the following result –

```
Name of the file: foo.txt  
Closed or not : False  
Opening mode : wb
```

Once a file is opened and you have one *file* object, you can get various information related to that file.

Here is a list of all the attributes related to a file object –

Sr.No.	Attribute & Description
1	file.closed :Returns true if file is closed, false otherwise.
2	file.mode : Returns access mode with which file was opened.

3	file.name : Returns name of the file.
---	--

```
#!/usr/bin/python3
```

```
# Open a file
```

```
fo = open("foo.txt", "wb")  
print ("Name of the file: ", fo.name)  
print ("Closed or not : ", fo.closed)  
print ("Opening mode : ", fo.mode)  
fo.close()
```

This produces the following result –

```
Name of the file: foo.txt  
Closed or not : False  
Opening mode : wb
```

CLOSE() METHOD

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax

```
fileObject.close();
```

4.6. Classes

Classes

Classes provide a means of bundling data and functionality together. Creating a new class creates a new *type* of object, allowing new *instances* of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

Class Definition Syntax

The simplest form of class definition looks like this:


```
class ClassName:
```

```
    <statement-1>
```

```
    .
```

```
    .
```

```
    .
```

```
    <statement-N>
```

Class definitions, like function definitions (`def` statements) must be executed before they have any effect. (You could conceivably place a class definition in a branch of an `if` statement, or inside a function.)

In practice, the statements inside a class definition will usually be function definitions, but other statements are allowed, and sometimes useful.

The function definitions inside a class normally have a peculiar form of argument list, dictated by the calling conventions for methods.

—When a class definition is entered, a new namespace is created, and used as the local scope — thus, all assignments to local variables go into this new namespace. In particular, function definitions bind the name of the new function here.

Class Objects

Class objects support two kinds of operations: attribute references and instantiation.

Attribute references use the standard syntax used for all attribute references in Python: `obj.name`. Valid attribute names are all the names that were in the class's namespace when the class object was created. So, if the class definition looked like this:

```
class MyClass:
```

```
    """A simple example class"""
```

```
    i = 12345
```

```
    def f(self):
```

```
        return 'hello world'
```

then `MyClass.i` and `MyClass.f` are valid attribute references, returning an integer and a function object, respectively. Class attributes can also be assigned to, so you can change the value of `MyClass.i` by assignment. `_doc_` is also a valid attribute, returning the docstring belonging to the class: `"A simple example class"`.

Class *instantiation* uses function notation. Just pretend that the class object is a parameterless function that returns a new instance of the class. For example (assuming the above class):

```
x = MyClass()
```

creates a new *instance* of the class and assigns this object to the local variable `x`.

The instantiation operation ("calling" a class object) creates an empty object. Many classes like to create objects with instances customized to a specific initial state. Therefore a class may define a special method named `__init__()`, like this:

```
def __init__(self):  
    self.data = []
```

When a class defines an `__init__()` method, class instantiation automatically invokes `__init__()` for the newly-created class instance. So in this example, a new, initialized instance can be obtained by:

```
x = MyClass()  
  
>>> class Complex:  
...     def __init__(self, realpart, imagpart):  
...         self.r = realpart  
...         self.i = imagpart  
...  
>>> x = Complex(3.0, -4.5)  
>>> x.r, x.i  
(3.0, -4.5)
```

Instance Objects

The only operations understood by instance objects are attribute references. `x.counter = 1`

```
while x.counter < 10:  
    x.counter = x.counter * 2  
print(x.counter)  
del x.counter
```

Method Objects

Usually, a method is called right after it is bound:

```
x.f()
```

In the MyClass example, this will return the string 'hello world'. However, it is not necessary to call a method right away: `x.f` is a method object, and can be stored away and called at a later time. For example:

```
xf = x.f  
while True:  
    print(xf())
```

will continue to print hello world until the end of time.

class Dog:

```
    kind = 'canine'    # class variable shared by all instances  
  
    def __init__(self, name):  
        self.name = name # instance variable unique to each instance
```

```
>>> d = Dog('Fido')  
>>> e = Dog('Buddy')  
>>> d.kind    # shared by all dogs
```

'canine'

>>> e.kind *# shared by all dogs*

'canine'

>>> d.name *# unique to d*

'Fido'

>>> e.name *# unique to e*

'Buddy'

4.7 IoT physical devices and end points

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, 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

4.8 Building blocks of IoT device

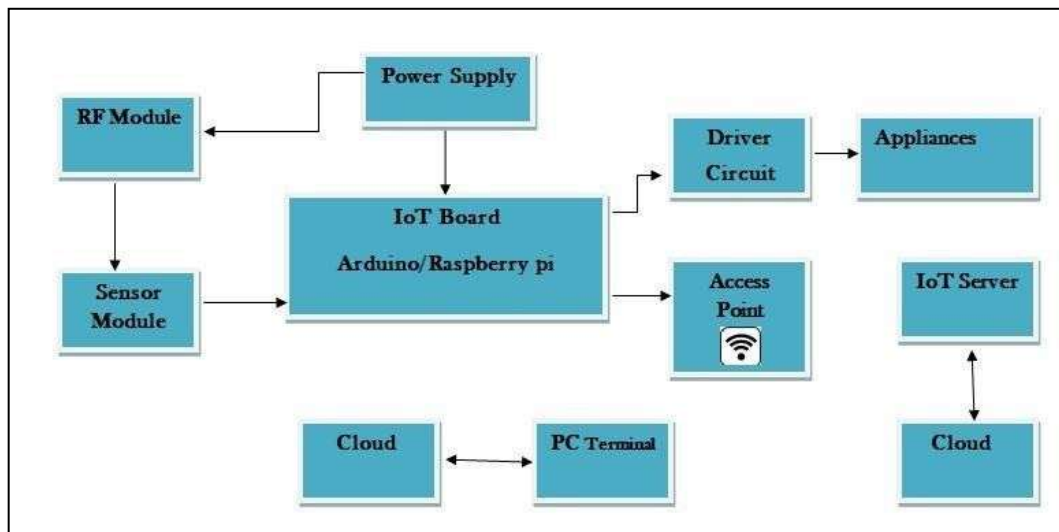
Basic building blocks of an IoT Device

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 near 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 Internet of Things (IoT)

4.9 Raspberry Pi

Raspberry Pi (RPi) defines as a **series of single-board computers** that are now increasingly being used to connect IoT devices. RPi can be plugged into a computer monitor. It is a capable little device that enables people to explore computing and learn how to program in languages like Scratch and Python.

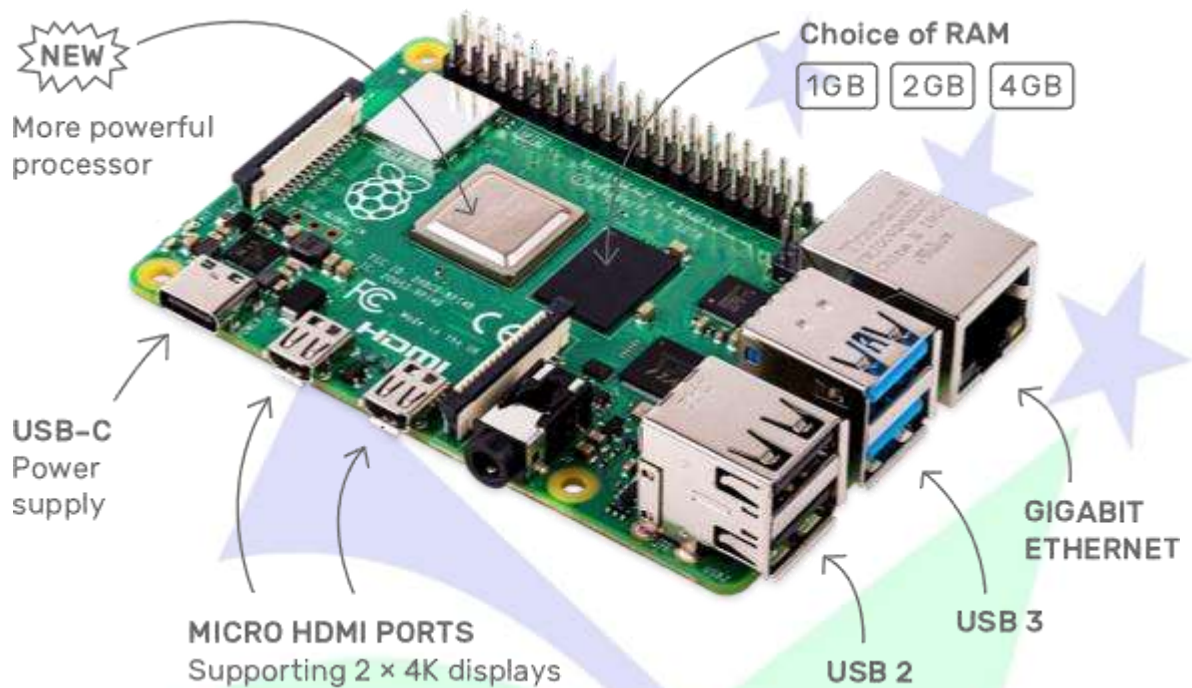
The powerful **CPU coupled with Wireless LAN and Bluetooth 4.1 radio** makes it an ideal candidate for IoT projects, because multiple sensors can be connected to it simultaneously. In addition, the Raspberry Pi has a 40-pin GPIO (General Purpose I/O) connector for interfacing with external sensors.

The Raspberry Pi is a series of **low-cost, programmable computers** that include a set of GPIO, or 'General Purpose Input Output', pins that can be used to connect and control external electronic devices, and to create Internet of Things (IoT) solutions.

The work for IoT based home automation is completed successfully using internet source and Raspberry pi. It is **reliable and scalable home automation system with low**

cost and easy to implement. It makes human life easy and comfortable. It is possible to operate home appliances from any part of the globe.

The IoT devices include **wireless sensors, software, actuators, computer devices and more.** They are attached to a particular object that operates through the internet, enabling the transfer of data among objects or people automatically without human intervention.

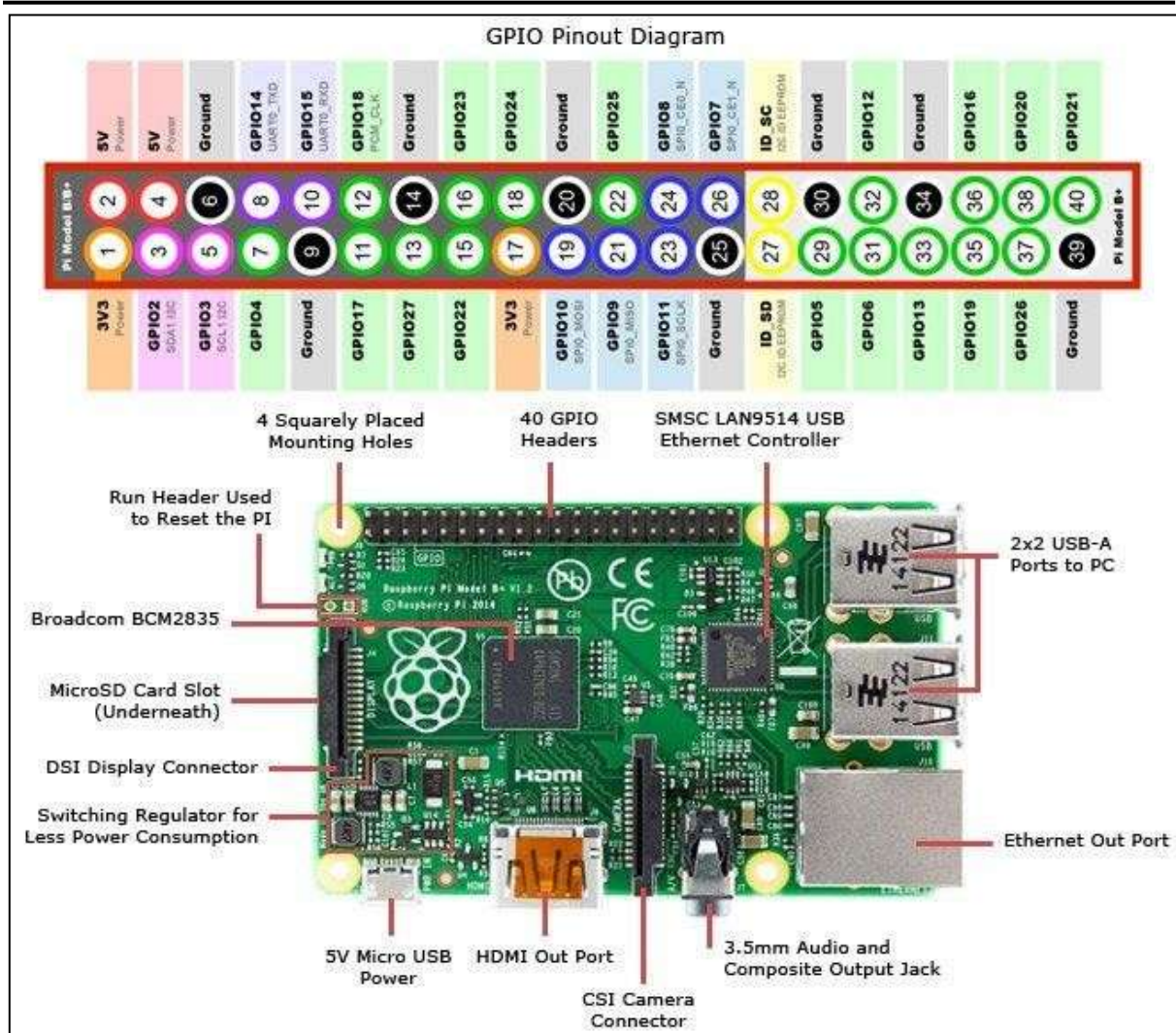


Device Raspberry Pi

4.10. Linux on Raspberry Pi

1. **Raspbian:** Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.
2. **Arch:** Arch is an Arch Linux port for AMD devices.
3. **Pidora:** Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
4. **RaspBMC:** RaspBMC is an XBMC media-center distribution for Raspberry Pi.
5. **OpenELEC:** OpenELEC is a fast and user-friendly XBMC media-center distribution.
6. **RISC OS:** RISC OS is a very fast and compact operating system.

4.11. Raspberry Pi interfaces.



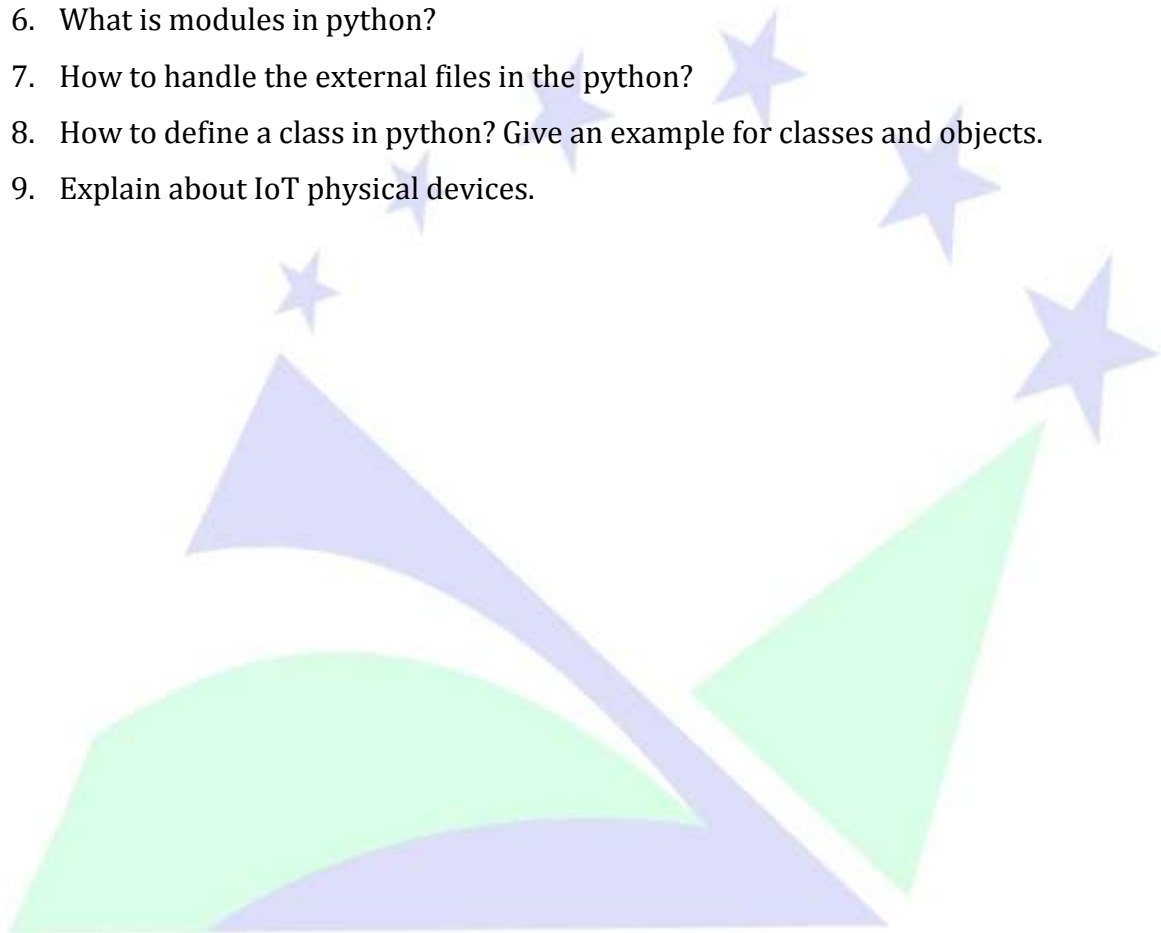
Raspberry Pi interfaces

Interfaces

- Camera — enable the Raspberry Pi Camera Module.
- SSH — allow remote access to your Raspberry Pi from another computer using SSH.
- VNC — allow remote access to the Raspberry Pi Desktop from another computer using VNC.
- SPI — enable the SPI GPIO pins.
- I2C — enable the I2C GPIO pins.

Important Questions:

1. How to installing python?
2. Define type conversions in python.
3. Write a short note on If – else in python.
4. Write a short note looping in python
5. Define user functions in python
6. What is modules in python?
7. How to handle the external files in the python?
8. How to define a class in python? Give an example for classes and objects.
9. Explain about IoT physical devices.



STARS RISE HERE...

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

5.1. IoT physical servers & cloud computing

Internet of things widely known as IoT is a **network of physical objects** like devices, vehicles, buildings, etc.

IoT devices share the **sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be analyzed or analyzed locally**. Sometimes, these devices communicate with other related devices and act on the information they get from one another.

Just like cloud computing is built on the tenets of speed and scale, **IoT applications are built on the principle of mobility and widespread networking**. Hence, it is essential that both cloud and IoT form cloud-based IoT applications in a bid to make the most out of their combination.

Cloud computing, as well as IoT, work towards increasing the efficiency of everyday tasks and both have a complementary relationship. On one hand, IoT generates lots of data while on the other hand, cloud computing paves way for this data to travel.

Many cloud providers take advantage of this to provide a pay-as-you-use model where customers pay for the specific resources used. Also, cloud hosting as a service adds value to IoT start-ups by providing economies of scale to reduce their overall cost structure.

Cloud computing has more or less penetrated mainstream IT and its infrastructure. Many tech biggies such as Amazon, Alibaba, Google and Oracle are building machine-learning tools with the help of cloud technology to offer a wide range of solutions to businesses worldwide.

Communication API

API stands **for Application Programming Interface**. It's a set of lines of codes and specifications that allow two devices to communicate with one another. They serve as the interface between different programs. And, to put it bluntly, the whole concept of IoT falls down without APIs.

Web Socket APIs are suitable for IoT Applications with low latency or high throughput requirements. Difference between Rest API and Web Socket API : S.NO. 1

Communication APIs, like the term sounds, are APIs built for the communications space. They establish a standardized syntax and methods of communication. In other words, Communication APIs define rules of what interactions are possible between servers and communication applications.

An Application Program Interface (API) is **a set of routines, protocols, and tools for building software applications**; it specifies how software components should interact. REST allows data to flow over internet protocols and to delegate and manage authorization

“APIs are the market enabler, and ‘internet of things’ devices would be useless without them. By exposing data that enables multiple devices to be connected, APIs provide an interface between the internet and the things to reveal previously unseen possibilities,” said Chris O’Connor, IBM’s GM for IoT, in a blog entry. “In the year to come, the power and importance of APIs will be at the forefront of the conversation around enabling—and more important—monetizing the ‘internet of things.’”

Enter cloud computing- an on-demand delivery of computing power, database storage, applications and IT resources. It enables organizations to consume a compute resource, like a virtual machine (VM) instead of building a computing infrastructure on premise.

5.2. WAMP (Windows, Apache, MySQL, PHP)

Web Socket is a network communication protocol, which is required for many advanced features.

WAMP is sometimes used as an abbreviated name for the software stack Windows, Apache, MySQL, PHP. It is derived from LAMP which stands for Linux, Apache, MySQL, and PHP. As the name implies, while LAMP is used on Linux servers, WAMP is used on Windows servers.

WampServer refers to a solution stack for the Microsoft Windows operating system, created by Romain Bourdon and consisting of the Apache web server, OpenSSL for SSL support, MySQL database and PHP programming language.

WAMP Server is a server which is used to host PHP pages. PHP is a server-side scripting language developed by Rasmus Lerdorf. WAMP stands for – Window Apache MySQL and PHP.

Features of WAMP

- preconfigured WAMP-System with Apache, PHP, MySQL, phpMyAdmin, Mercury/32 Mail-Server and SQLite
- Equally useful for developers as for productive usages thanks to switchable settings between security- and developer-functionality
- Centralized GUI for all important features of the above applications
- Assistant to operate the applications as a restricted user
- Add-on system for reloading and integration of other applications (such as Tomcat, Python, SVN, etc.) from the network
- 1-Click Installer for popular web applications such as phpBB, WordPress u.v.a.
- Assistant for performance tuning with automatic Evaluation of the System configuration (number of CPUs, RAM, etc.)
- Editor for the Windows HOSTS file to create "virtual domains"
- dynamic and automatic updates of individual components (Apache, MySQL, etc.) directly from the software
- Focus on a lightweight and fast system - the complete setup consists of only a 33MB file.

5.3. Xively cloud for IoT

Xively (formerly known as Cosm and Pachube) is an Internet of Things (IoT) platform owned by Google.

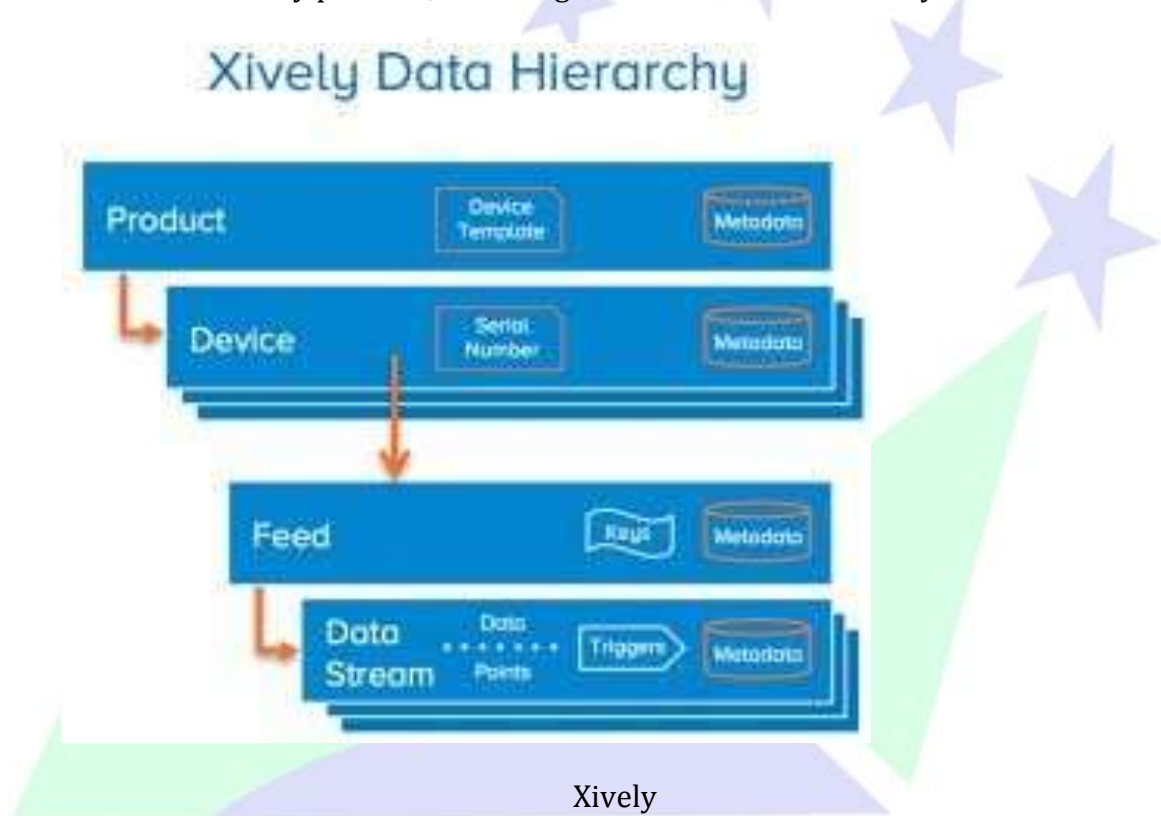
Xively offers product companies a way to connect products, manage connected devices and the data they produce, and integrate that data into other systems. It is pronounced "zively" (rhymes with lively).

Xively is an IoT cloud platform that is “an enterprise platform for building, managing, and deriving business value from connected products”. Moreover, it provides a cloud-based API with an SDK that simplifies the development process. It supports several platforms and technologies: Android.

Xively is a data collection, management, and distribution infrastructure. It also provides APIs to connect and develop IoT applications. Xively comes under the category of Connected Product Management (CPM) platform. This helps in convenient management of apps with Xively cloud and other APIs.

Xively (formerly Cosm and Pachube) is an Internet of Things (IoT) platform from Google.

Xively offers product companies a way to connect products, manage the connected devices and the data they produce, and integrate this data with other systems.



- Programmers or Developers have to register with Xively to use cloud services.
- After registration and account creation, developers can create different devices for which he has to create an IoT app. It can be easily done using the templates provided in the Web Interface of Xively.
- Each connected devices is allocated a unique FEED_ID. It specifies the data stream and metadata of the connected device.
- Once this is done permissions on the IoT devices are assigned using the available APIs. The available permissions are Create, Update, Delete and Read.

- One or more bidirectional channels are created after we connect a device with Xively. Each channel is unique to the device connected.
- Xively cloud is connected with the help of these channels.
- Xively APIs are used by IoT devices to create communication enabled products.

5.4. Python Web application frame work

Python Web framework is a collection of packages or modules that allow developers to write Web applications or services. With it, developers do not need to handle low-level details like protocols, sockets or process/thread management.

Python is one of the most acceptable languages among web and application developers because of its strong emphasis on efficiency and readability. There are numerous outstanding Python web frameworks, each with their own specialities and features.

DIANGO

Category – Django belongs to the full-stack Python framework.

Release – Latest release – 2.1 version, commonly used release – 1.8, 1.6 version.

About – Django is a high level Python web framework which allows rapid, clean and pragmatic design development.

Django handles much of the complexities of web development, so you can focus on writing your app without a need to reinvent the wheel. It's free and open source.

To map objects to database table, Django uses ORM and the same is used to transfer from one database to other.

It works with mostly all important databases like Oracle, MySQL, PostgreSQL, SQLite, etc.

There are numerous websites in the industry which uses Django as their primary framework for backend development.

Features of Django

Some of the exemplary features of this Python web framework are –

- URL routing
- Authentication
- Database schema migrations
- ORM (Object-relational mapper)
- Template engine

FLASK

Category – Flask belongs to Non Full-stack frameworks.

Release – 1.0.2 released on 2018-05-02

About – It is classified as a micro-framework as we don't require any particular libraries or tools. It has no form validation or database abstraction layer or any other components where pre-existing third party libraries provide common functions.

However, flask support multiple extensions which extended the application features as if they were implemented in Flask itself.

Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common frameworks related tools.

Features of Flask

- Integrated support for unit testing
- Restful request dispatching
- Contains development server and debugger
- Support for secure cookies
- Unicode-based
- 100% WSGI 1.0 compliant
- Extensive documentation
- Google App Engine compatibility
- Extensions available to enhance features desired

WEB2PY

Category – Web2py belongs to Full-stack framework family.

Release – 2.17.1, released on 2018-08-06

About – Python 2.6, 2.7 to Python 3.x version. Development, database administration, debugging, deployment, testing, and maintenance of applications all can be done through web interface but generally not required.

A scalable open source framework comes with its own web-based IDE alongside a code editor, one-click deployment and debugger.

Features of Web2py

- This framework comes with many developing tools and built-in features that eliminate the hassle of complexity to the developers.
- With no installation and configuration, it is easy to run.
- Supports almost all major operating system, like Windows, Unix/Linux, Mac, Google App Engine and almost all web hosting platform through Python 2.7/3.5/3.6/ version.
- Easy to communicate with MySQL, MSSQL, IBM DB2, Informix, Ingres, MongoDB, SQLite, PostgreSQL, Sybase, Oracle and Google App Engine.
- It prevents the most common types of vulnerabilities including Cross Site Scripting, Injection Flaws, and Malicious File Execution.
- Supports error tracking and internationalization.
- Multiple protocols readability.
- Employs successful software engineering practices that makes code easy to read and maintain.
- Ensure user-oriented advancements through backward compatibility.

PYRAMID

Category – Pyramid is a non-Full Stack Frameworks

Release – 1.9.2, released on 2018-04-23

About – Pyramid is a small, fast, down-to-earth Python web framework. It is developed as part of the Pylons Project.

It is licensed under a BSD-like license. It makes real-world web application development and deployment more fun, more predictable and more productive.

Features of Pyramid

Python Pyramid is an open sourced framework with the following features –

Simplicity – Anyone can start to work with it without any prior knowledge about it.

Minimalism – Quite out of the box, Pyramid comes with only some important tools, which are needed for almost every web application, may it be security or serving static assets like JavaScript and CSS or attaching URLs to code.

Documentation – Includes exclusive and up to date documentation.

Speed – Very fast and accurate.

Reliability – It is developed, keeping in mind that it is conservative and tested exhaustively. If not tested properly, it will be considered as broke.

Openness – It's sold with a permissive and open license.

DASH

Category – The Dash framework belongs to “other” Python web frameworks.

Release – 0.24.1, core dash backend.

About – Dash as an open source library for creating interactive web-based visualizations.

Features of Dash

Provides access to configurable properties and Flask instance

Through Flask plugins, we can extend the capabilities of the Dash application

Mobile-ready

5.5 Amazon web services for IoT

AWS IoT (Amazon internet of things) is an Amazon Web Services platform that collects and analyzes data from internet-connected devices and sensors and connects that data to AWS cloud applications.

A developer can configure rules in a syntax that's similar to SQL to transform and organize data.

AWS IoT 1-Click

AWS IoT 1-Click is a service that enables simple devices to trigger AWS Lambda functions that can execute an action.

AWS IoT 1-Click supported devices enable we to easily perform actions such as notifying technical support, tracking assets, and replenishing goods or services. You can also track device health and activity with the pre-built reports.

AWS IoT Analytics

AWS IoT Analytics is a fully-managed service that makes it easy to run and operationalize sophisticated analytics on massive volumes of IoT data without having to worry about the cost and complexity typically required to build an IoT analytics platform.

It is the easiest way to run analytics on IoT data and get insights to make better and more accurate decisions for IoT applications and machine learning use cases.

IoT data is highly unstructured which makes it difficult to analyze with traditional analytics and business intelligence tools that are designed to process structured data.

AWS IoT Button

The AWS IoT Button is a programmable button based on the Amazon Dash Button hardware.

This simple Wi-Fi device is easy to configure, and it's designed for developers to get started with AWS IoT Core, AWS Lambda, Amazon DynamoDB, Amazon SNS, and many other Amazon Web Services without writing device-specific code.

We can code the button's logic in the cloud to configure button clicks to count or track items, call or alert someone, start or stop something, order services, or even provide feedback.

AWS IoT Core

AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices.

AWS IoT Core can support billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely.

With AWS IoT Core, your applications can keep track of and communicate with all your devices, all the time, even when they aren't connected.

AWS IoT Device Defender

AWS IoT Device Defender is a fully managed service that helps you secure your fleet of IoT devices.

AWS IoT Device Defender continuously audits your IoT configurations to make sure that they aren't deviating from security best practices.

A configuration is a set of technical controls you set to help keep information secure when devices are communicating with each other and the cloud.

AWS IoT Device Defender makes it easy to maintain and enforce IoT configurations, such as ensuring device identity, authenticating and authorizing devices, and encrypting device data.

AWS IoT Device Defender continuously audits the IoT configurations on your devices against a set of predefined security best practices.

AWS IoT Device Management

As many IoT deployments consist of hundreds of thousands to millions of devices, it is essential to track, monitor, and manage connected device fleets. You need to ensure your IoT devices work properly and securely after they have been deployed.

We also need to secure access to your devices, monitor health, detect and remotely troubleshoot problems, and manage software and firmware updates.

AWS IoT Events

AWS IoT Events is a fully managed IoT service that makes it easy to detect and respond to events from IoT sensors and applications.

Events are patterns of data identifying more complicated circumstances than expected, such as changes in equipment when a belt is stuck or connected motion detectors using movement signals to activate lights and security cameras.

AWS IoT SiteWise

AWS IoT SiteWise is a managed service that makes it easy to collect, store, organize and monitor data from industrial equipment at scale to help you make better, data-driven decisions.

We can use AWS IoT SiteWise to monitor operations across facilities, quickly compute common industrial performance metrics, and create applications that analyze industrial equipment data to prevent costly equipment issues and reduce gaps in production.

AWS Partner Device Catalog

The AWS Partner Device Catalog helps you find devices and hardware to help you explore, build, and go to market with your IoT solutions.

Search for and find hardware that works with AWS, including development kits and embedded systems to build new devices, as well as off-the-shelf-devices such as gateways, edge servers, sensors, and cameras for immediate IoT project integration.

Important questions:

1. Define IoT physical servers & cloud computing
2. Explain WAMP.
3. What is the usage of Xively cloud for IoT?
4. How to work with python Web application framework?
5. Explain about Amazon web services for IoT