# Internet of Things – BCS701

# AY: 2025-26

# Course Objectives

| Internet of Things | | Semester | VII |
|---|---|---|---|
| Course Code | **BCS701** | CIE Marks | 50 |
| Teaching Hours/Week (L:T:P: S) | 3:0:2:0 | SEE Marks | 50 |
| Total Hours of Pedagogy | 40 hours Theory + 8-10 Lab slots | Total Marks | 100 |
| Credits | 04 | Exam Hours | 03 |

1. Understand the fundamentals of Internet of Things and its building blocks along with their characteristics.
2. Understand the recent application domains of IoT in everyday life.
3. Understand the protocols and standards designed for IoT and the current research on it.
4. Understand the other associated technologies like cloud and fog computing in the domain of IoT.
5. Improve their knowledge about the various cutting-edge technologies in the field of IoT and machine learning applications.
6. Gain insights about the current trends of machine learning and AI techniques used in IoT to orient towards the present industrial scenario.

# Course Content

MODULE-1

Introduction to Internet of Things: Introduction, Physical design of IOT, Logical Design of IOT, IOT enabling technologies, IOT Levels & Deployment Templates.

MODULE-2

IOT and M2M: Introduction: M2M, Difference between IoT and M2M, SDN and NFV for IOT, IOT. System Management with NETCONF-YANG: Need for IOT Systems Management, Simple Network Management Protocol (SNMP), Network operator requirements, NETCONF, YANG, IoT Systems Management with NETCONF-YANG.

MODULE-3

IoT Platforms Design Methodology: Introduction, IoT Design Methodology, Case Study on IoT System for Weather Monitoring, loT Systems - Logical Design using Python: Introduction, Installing Python, Python Data Types and Data structures, Control flow, Functions, Modules, Packages, File Handling, Operations, Classes, Python Packages of Interest for IoT.

# Course Content

MODULE-4

IoT Physical Devices & End points: What is a loT Device, Raspberry Pi, About the Board, Linux on Raspberry Pi, Raspberry Pi interfaces, Programming Raspberry Pi with Python, Case Studies illustrating IoT design: Home Automation, Cities, Agriculture.

MODULE-5

Data Analytics for IoT: Introduction, Apache Hadoop, Using Hadoop MapReduce for Batch Data Analytics, Apache Oozie, Apache Spark, Apache Storm, Using Apache Storm for Real-time Data Analysis.

**Suggested Learning Resources:**

**textbook**

Arshdeep Bahga, Vijay Madisetti, " Internet of Things- A Hands On Approach", Universities press, 2014.

**Reference Books**

1. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Robert Barton, Jerome Henry,"IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things", 1 stEdition, Pearson Education (Cisco Press Indian Reprint). (ISBN: 978-9386873743)
2. Srinivasa K G, "Internet of Things", CENGAGE Leaning India, 2017.

# Course Outcomes

1. Describe the basics of the Internet of Things, including its design, technologies, and different types of deployments.
2. Explain the concepts of IoT and M2M and describe the use of network management protocols
3. Apply basic IoT design steps and programming to create simple IoT applications
4. Describe the architecture and interfaces of Raspberry Pi and implement Python-based IoT applications for different domains
5. Elaborate the need for Data Analytics in IoT.

## Module - 1

# Introduction to Internet of Things

By,
Roopa B
Asst. Professor
Dept. of CSE
ATMECE, Mysuru

# Contents

- Introduction
- Physical design of IoT
- Logical Design of IoT
- IoT enabling technologies
- IoT Levels & Deployment Templates.

# Introduction

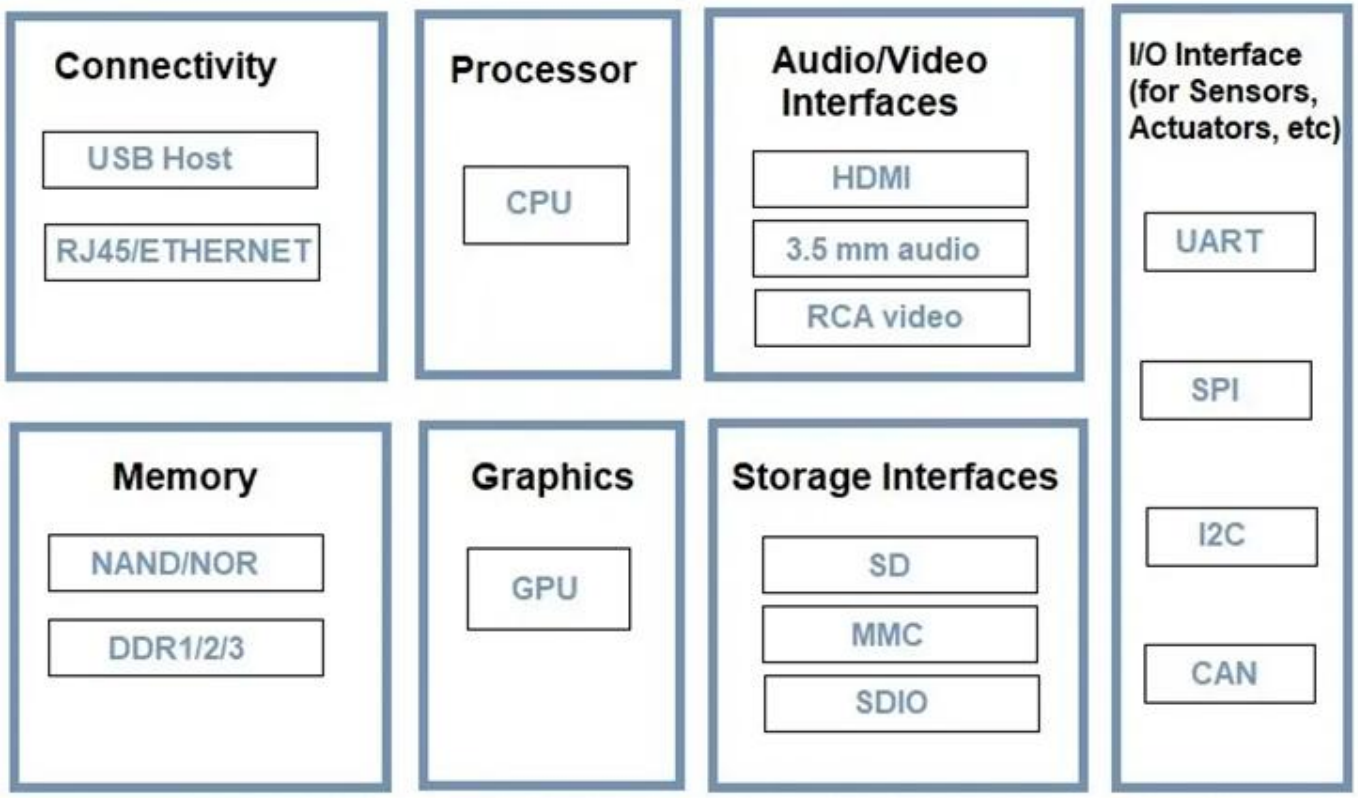
- Internet of Things (IoT) comprises things that have unique identities and are connected to the internet.
- Existing devises, such as networked computers or 4G enabled mobile phones, already have some form of unique identities and are also connected to the internet, the focus on IoT in the configuration, control and networking via the internet of devices or things, that are traditionally not associated with the Internet.
- The scope of IoT is not limited to just connected things (Devices, appliance, machines) to the Internet.
- Applications on IoT networks extract and create information from lower-level data by filtering, processing, categorizing, condensing and contextualizing the data.
- The information obtained is then organized and structured to infer knowledge about the system and or its user, its environment and its operations and progress towards its objectives, allowing a smarter performance.

# Characteristics of IoT

- Dynamic and self-Adapting
- Self – Configuring
- Interoperable communication protocols
- Unique Identity
- Integrated into information network

# Physical design of IoT

## 1. Things of IoT



# Physical design of IoT

**2. IoT Protocols**

Link Layer

- Link Layer protocols determine how the data is physically sent over the networks physical layer or medium.

Example: 802.3 Ethernet

802.1- WI-FI

802.16 wiMAX

802.15.4 LR-WPAN

2G / 3G / 4G mobile communications

# Physical design of IoT

**2. IoT Protocols**

Network / internet layer

- The network layer is responsible for sending of IP datagrams from the source network to the destination network.
- This layer Performs the host addressing and packet routing.

Example: IPV4

IPV6

6LoWPAN

# Physical design of IoT

**2. IoT Protocols**

Transport layer

- The Transport layer protocols provide end-to-end message transfer capability independent of the underlying network.
- The message transfer capability can be set up on connections, either using handshake or without handshake acknowledgements.

Example: TCP

UDP

# Physical design of IoT

**2. IoT Protocols**

Application layer

- Application layer protocol defines how the application interfaces with the lower layer protocols to send the data over the network.
- Data are typically in files, is encoded by the application layer protocol and encapsulated in the transport layer protocol.

Example: Http

CoAP

WebSocket

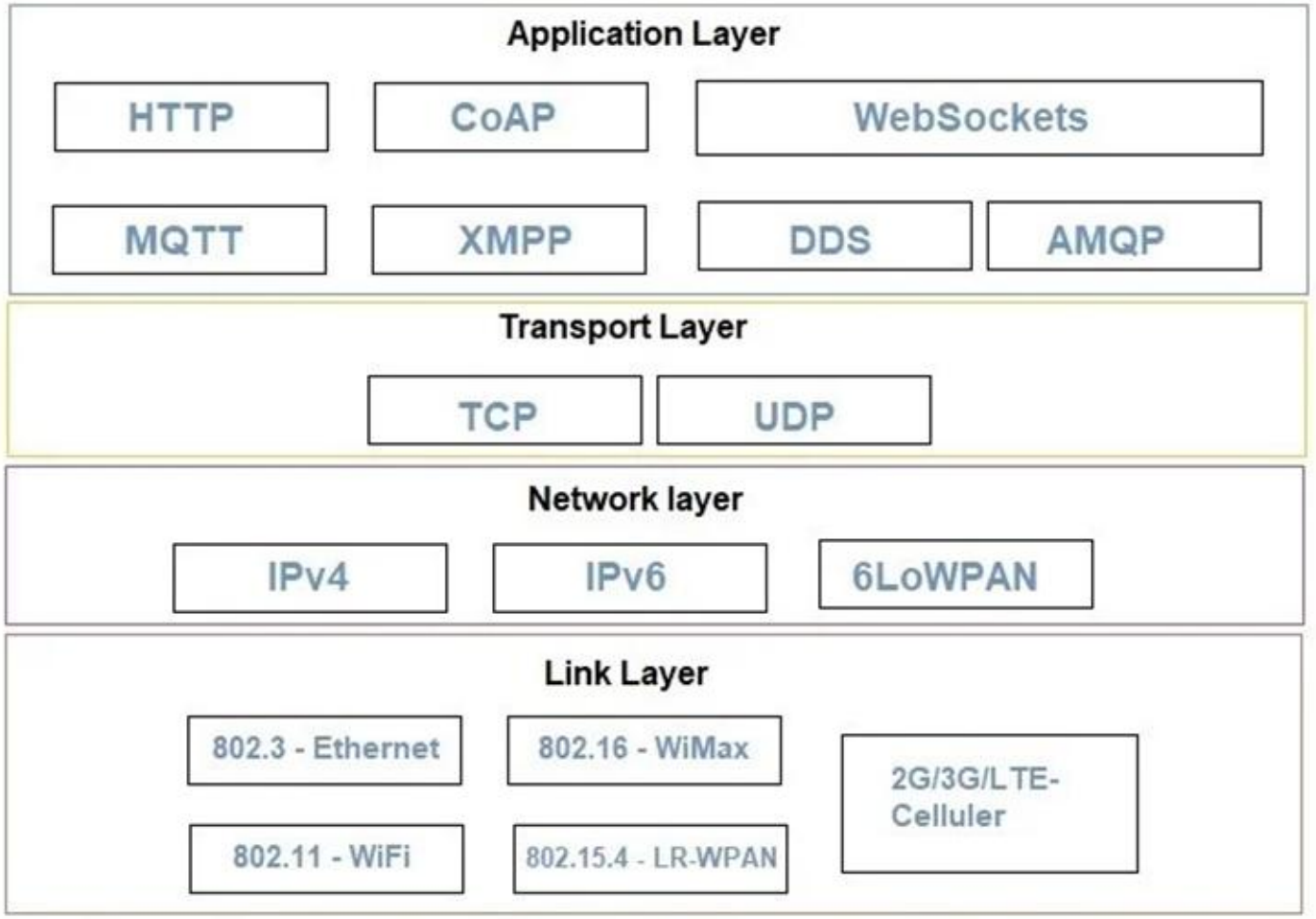
MQTT

XMPP

DDS

AMQP

# Physical design of IoT

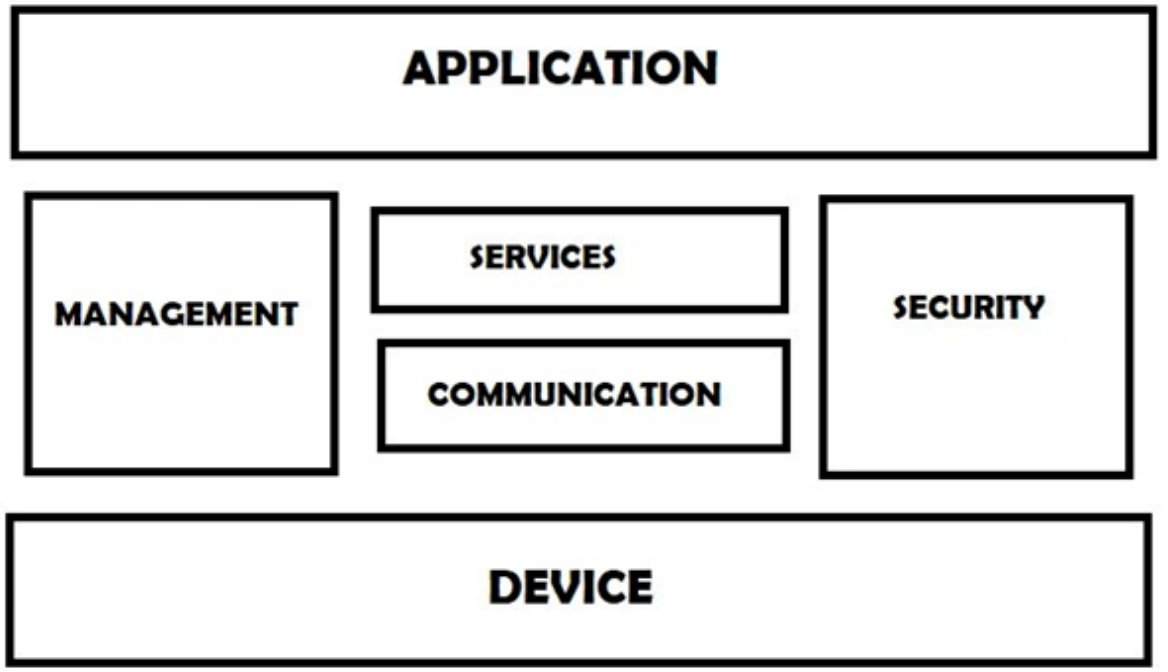


# Logical Design of IoT

- Logical design of an IoT system refers to an abstract representation of the entities and process without going into low level specification of the implementations.

1. IoT functional block
2. IoT communication model
3. IoT communication APIs

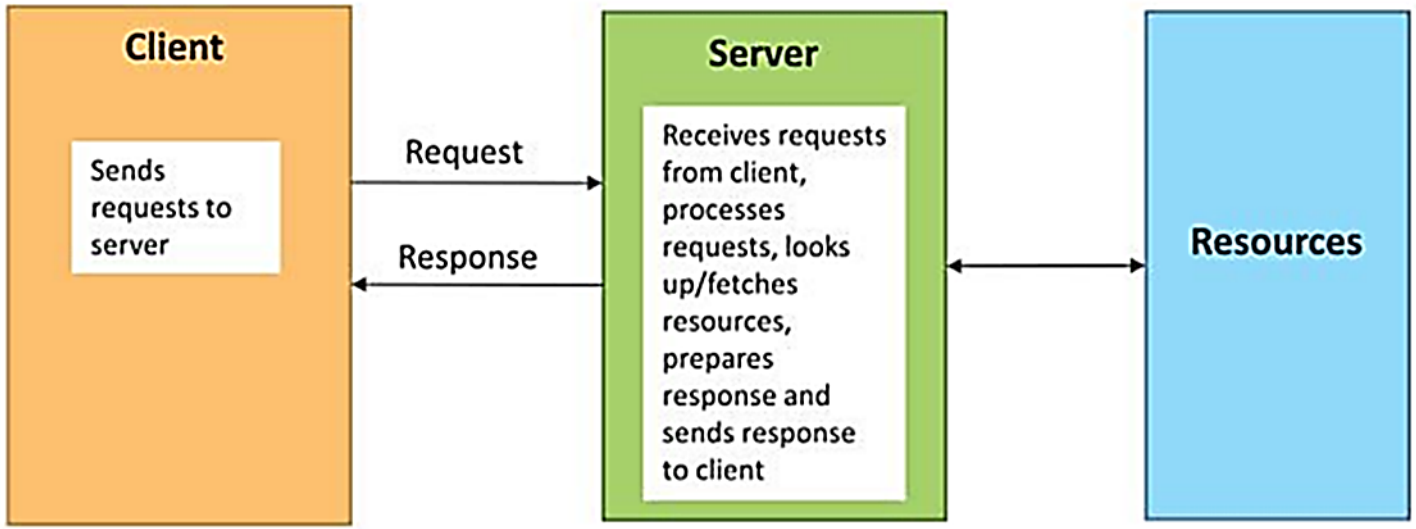
# Logical Design of IoT



# Logical Design of IoT

## 2. IoT communication model

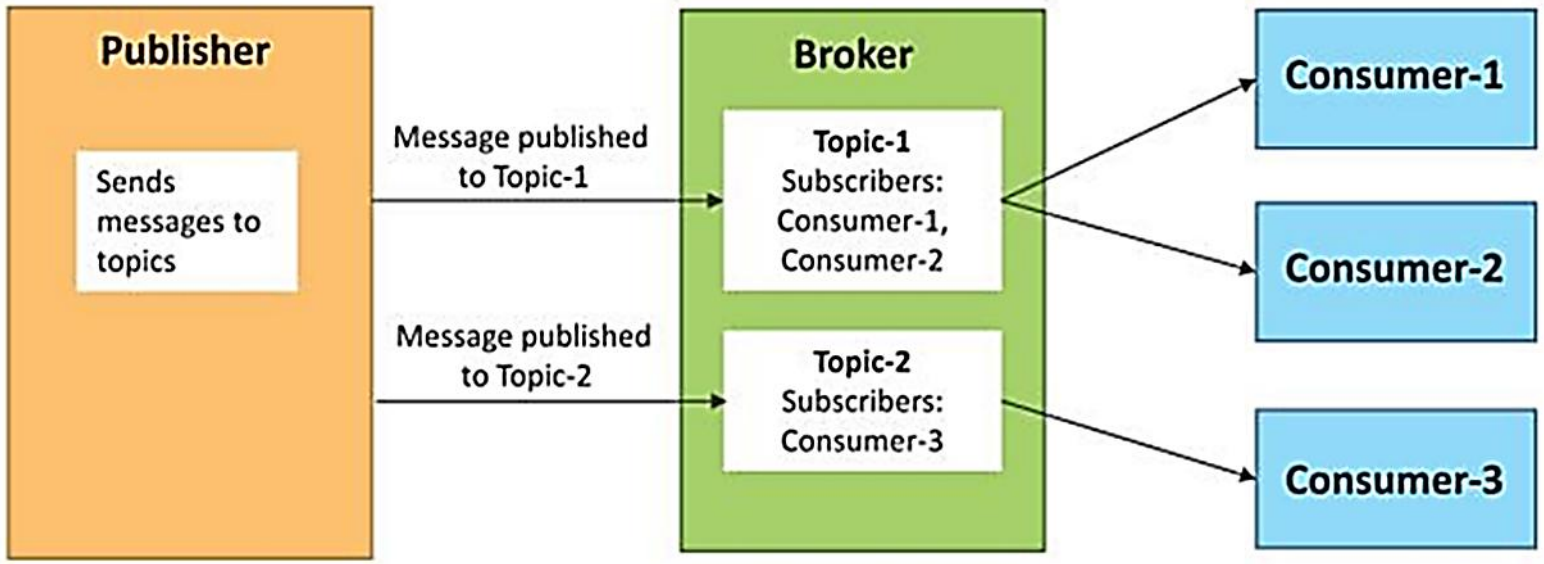
- Request response: Request-response is a Communications model in which the client sends request to the server and the server responds to the requests. when the server receives a request, it decides how to respond, if it shows the data retrieved resources definitions for the response and then send the response to the client.



# Logical Design of IoT

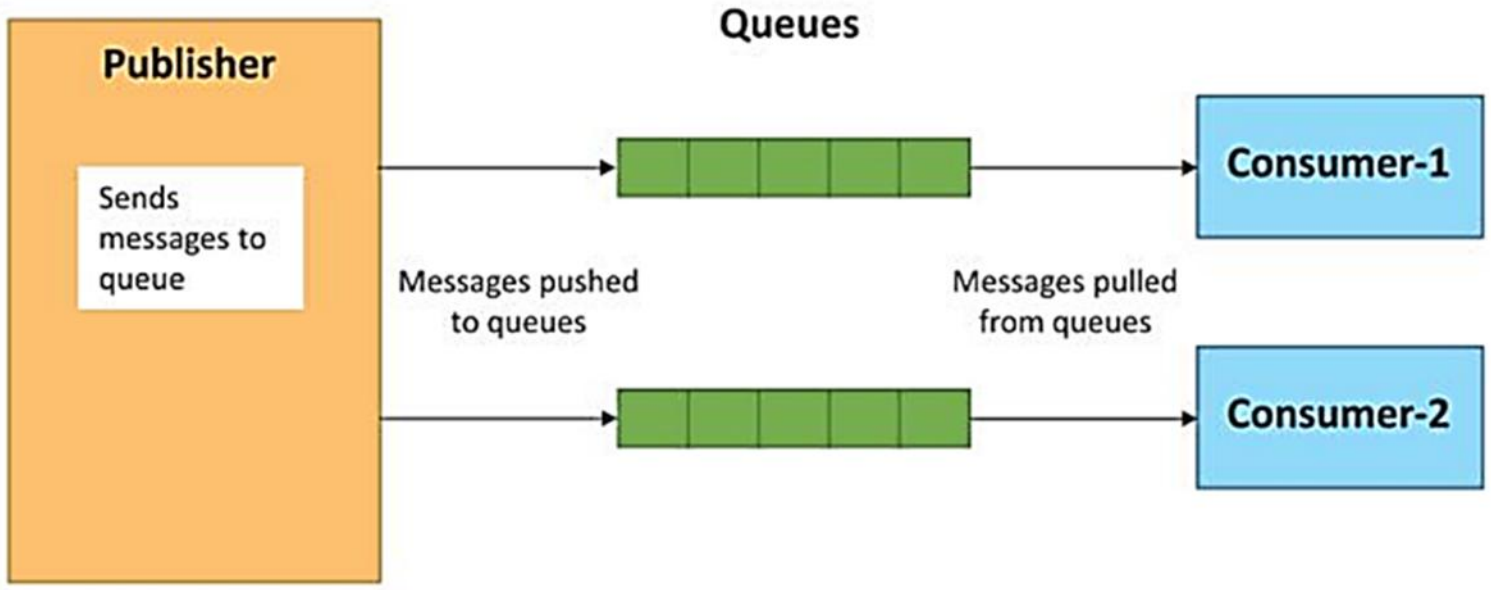
**2. IoT communication model**

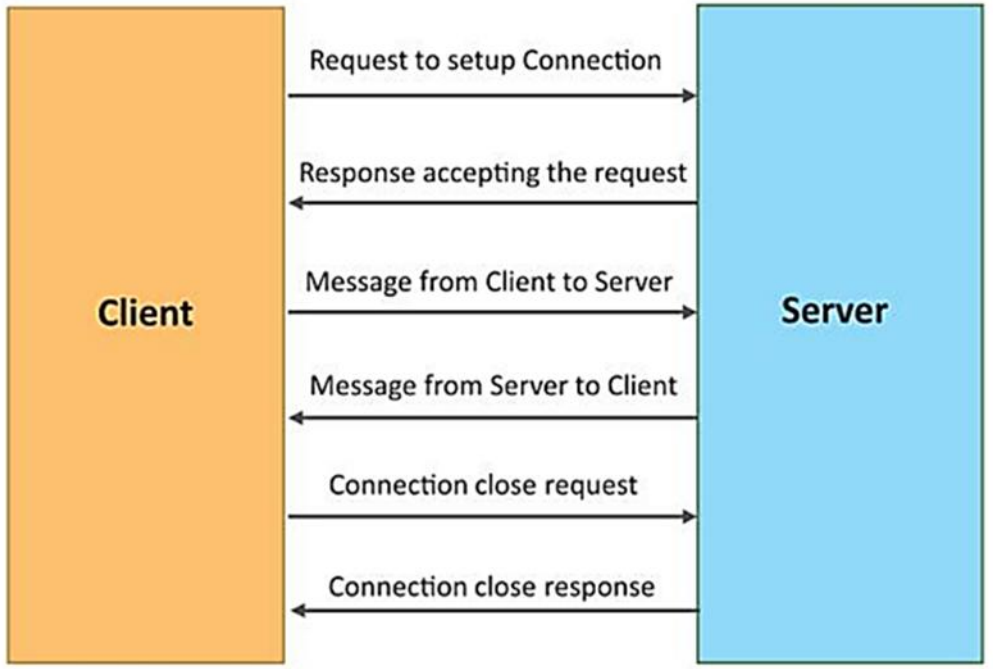
- Publish - Subscribe: Publishers are the source of data. Publishers send the data to the topics which is managed by the broker and are not aware of the consumer. Consumers Subscribe to the topic which are managed by the broker. When the broker receives the data for a topic from the publisher, it sends the data to all the subscribed consumers.



# Logical Design of IoT

- Push pull: Push pull is 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 consumer.
- Exclusive pair: Exclusive pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and the server. once the condition is setup it remains open until the client sends a request to close the connection.



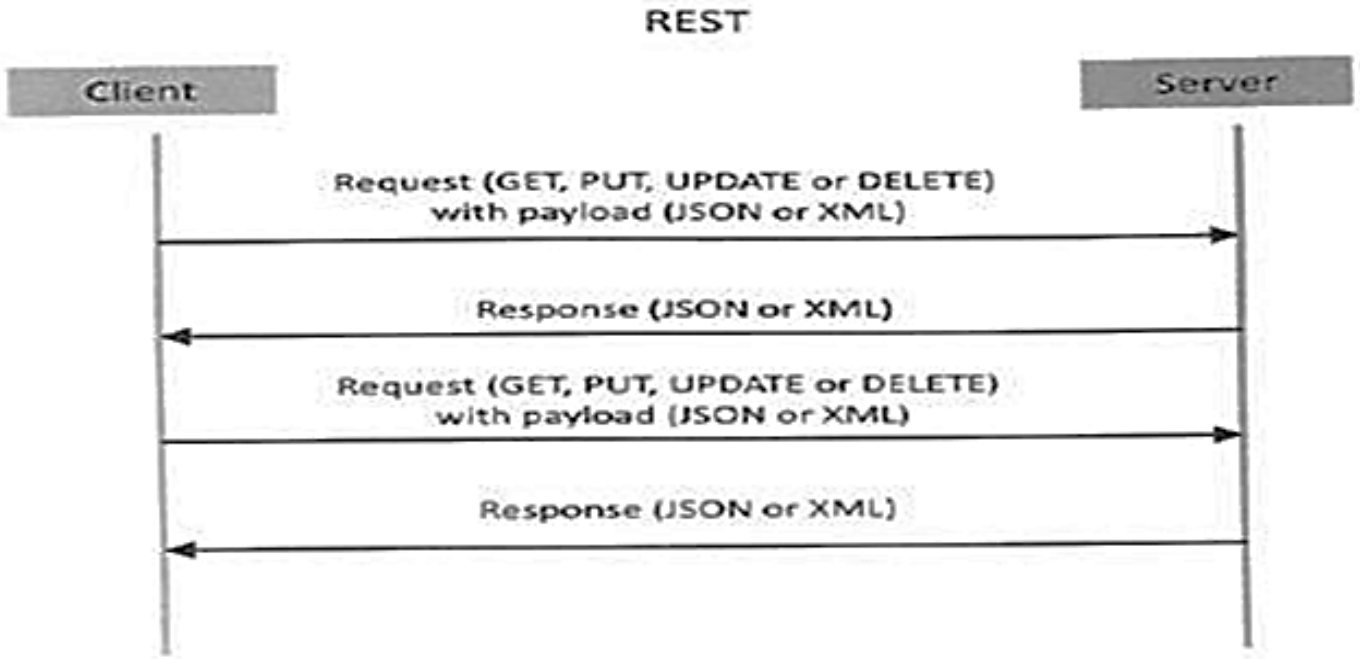



# Logical Design of IoT

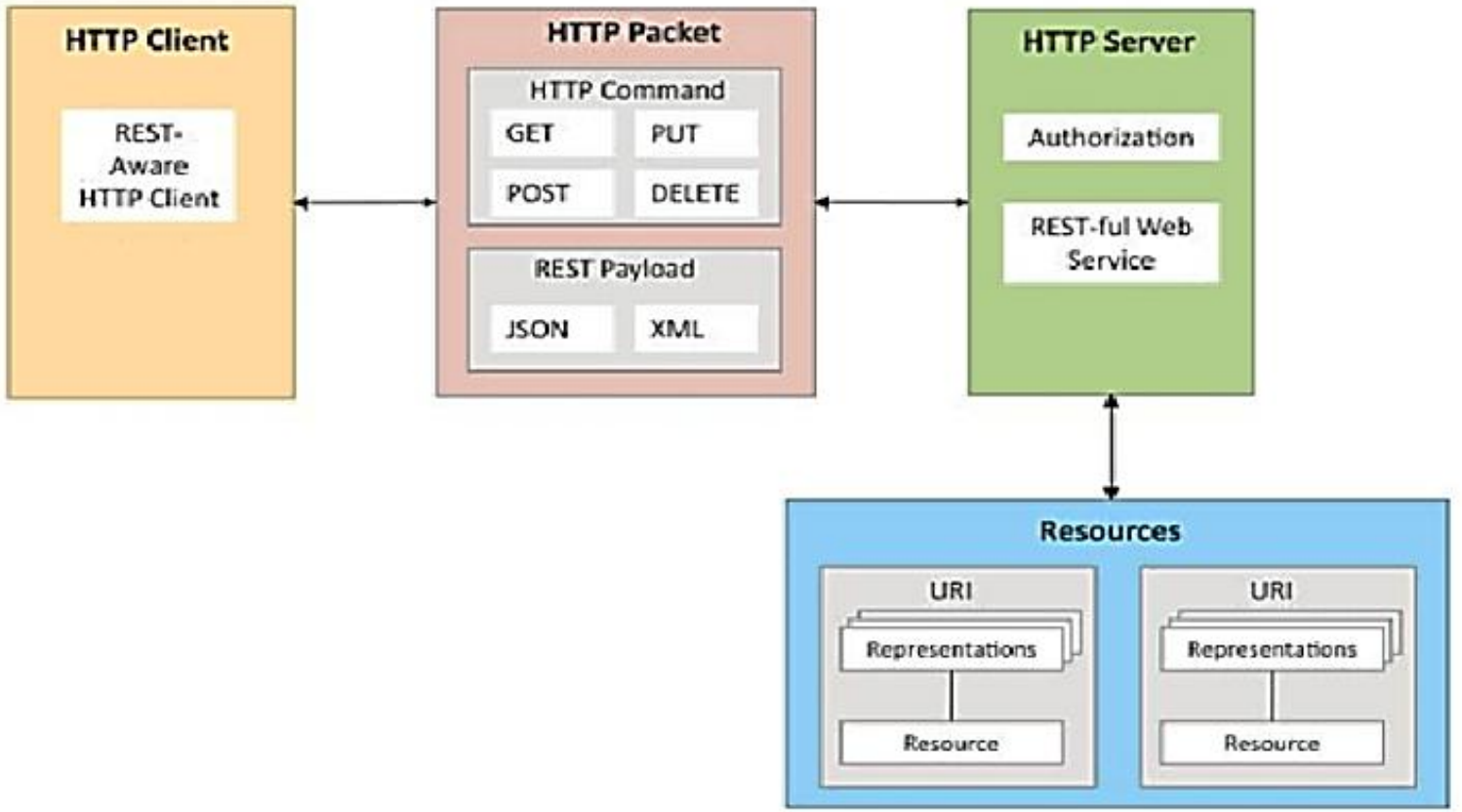
### 3. IoT communication APIs

REST- based communication API:

- Representational state transfer is a set of architectural principles by which you can design web service and Web API that focus on a system resource and how resources states and addressed the transferred.
- REST API follow the request- response communication model.



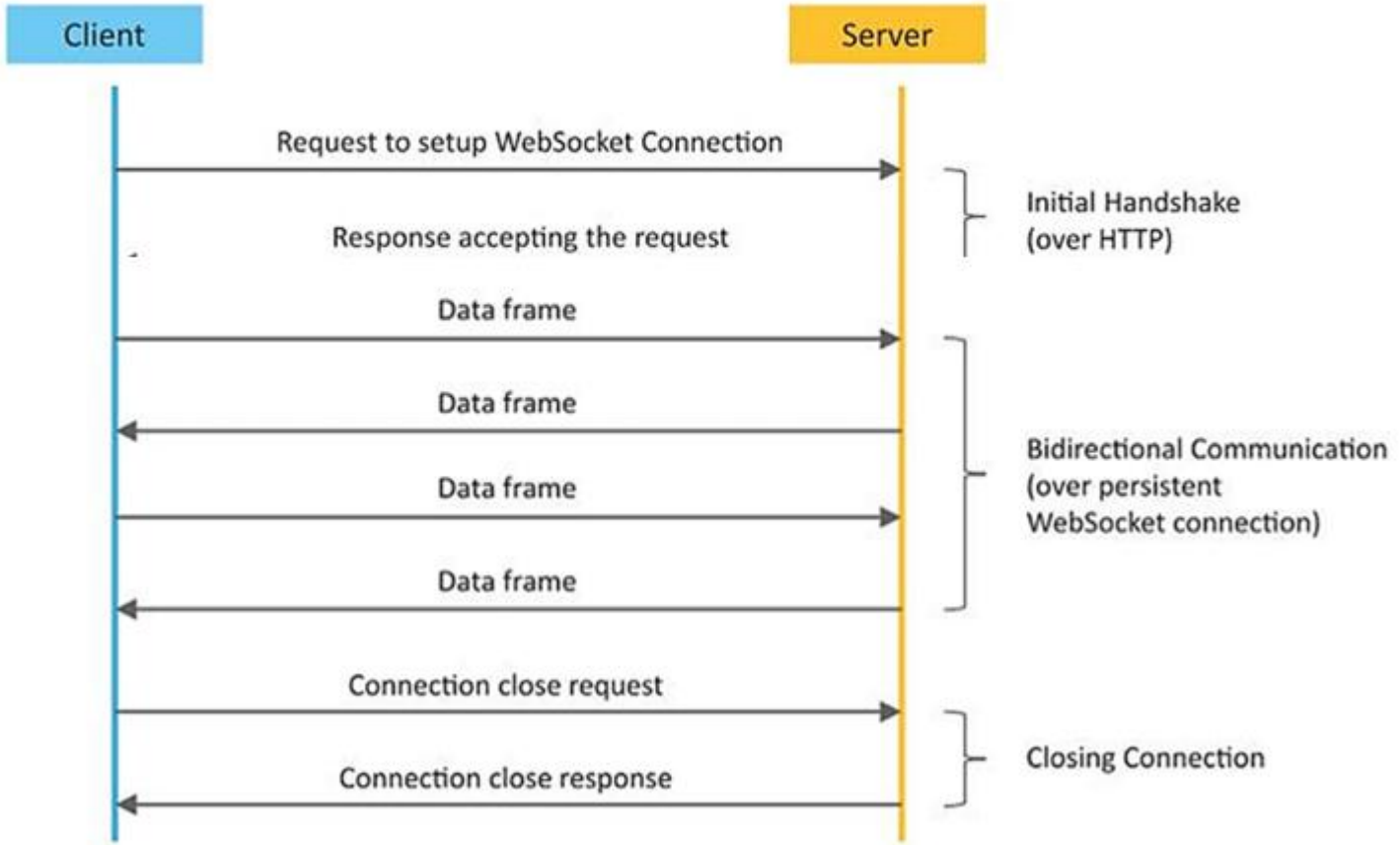
# Logical Design of IoT



# Logical Design of IoT

WebSocket based communication API:

WebSocket API allow bidirectional, full duplex communication between client and server.



# IoT enabling technologies

Iot is enabled by several technologies including

- Wireless sensor networks
- Cloud computing
- Big data analytics
- Embedded system
- Security protocols and architectures
- Communication protocols
- Web service
- Mobile internet and
- Semantic search engine.

# IoT enabling technologies

1. Wireless Sensor Network

- WSN comprises of distributed devices with the sensor which are used to monitor the environmental and physical conditions.
- A WSN consists of several end nodes and routers and a coordinator.
- End nodes have several sensors attached to them. End node can also act as a router.
- Routers are responsible for routing the data packet from end nodes to the coordinator.
- The coordinator node collects the data from all the notes coordinators also act as a Gateway that connects the WSN to the internet.

# IoT enabling technologies

2. Cloud computing

- Cloud Computing is a transformative 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.

Cloud Computing services are offered to user in different forms:

- Infrastructure as a service (IAAS)
- Platform as a service (PaaS)
- Software as a service (SaaS)

# IoT enabling technologies

**3. Big Data Analytics**

- Big data is defined as collections of data set whose volume, velocity in terms of its temporal variations or variety, is so large that it is difficult to store, manage, process and analyse the data using traditional database and data processing tools.
- Big Data Analytics involving several steps starting from Data cleaning data munging data processing and visualization.

Characteristics of data include:

- Volume
- Velocity
- Variety

# IoT enabling technologies

**4. Communications protocol**

- Communications protocols form the backbone of IoT system and enable network connectivity and coupling to applications. Communications protocols allow device to exchange data over the network. These protocols define the data exchange formats and data encoding schemes for devices and routing of packets from source to destination.

**5. Embedded systems**

- An Embedded system is computer system that has computer hardware and software embedded perform specific task. In contrast to general purpose computers or personal computers which can perform various types of tasks, embedded systems are designed to perform a specific set of tasks.

# IoT Levels & Deployment Templates

IoT system comprises of the following components:

1. Device: An IoT device allow identification, remote sensing, actuating and remote monitoring capabilities.

2. Resources: Resources are software components on the device for accessing and storing information for controlling actuator connected to the device also include software components that enable network access for the device.

3. Controller service: Controller Service is a native service that runs on the device and interact with the web services. Controller service sends data from the device to the web service receive command from the application from controlling the device.

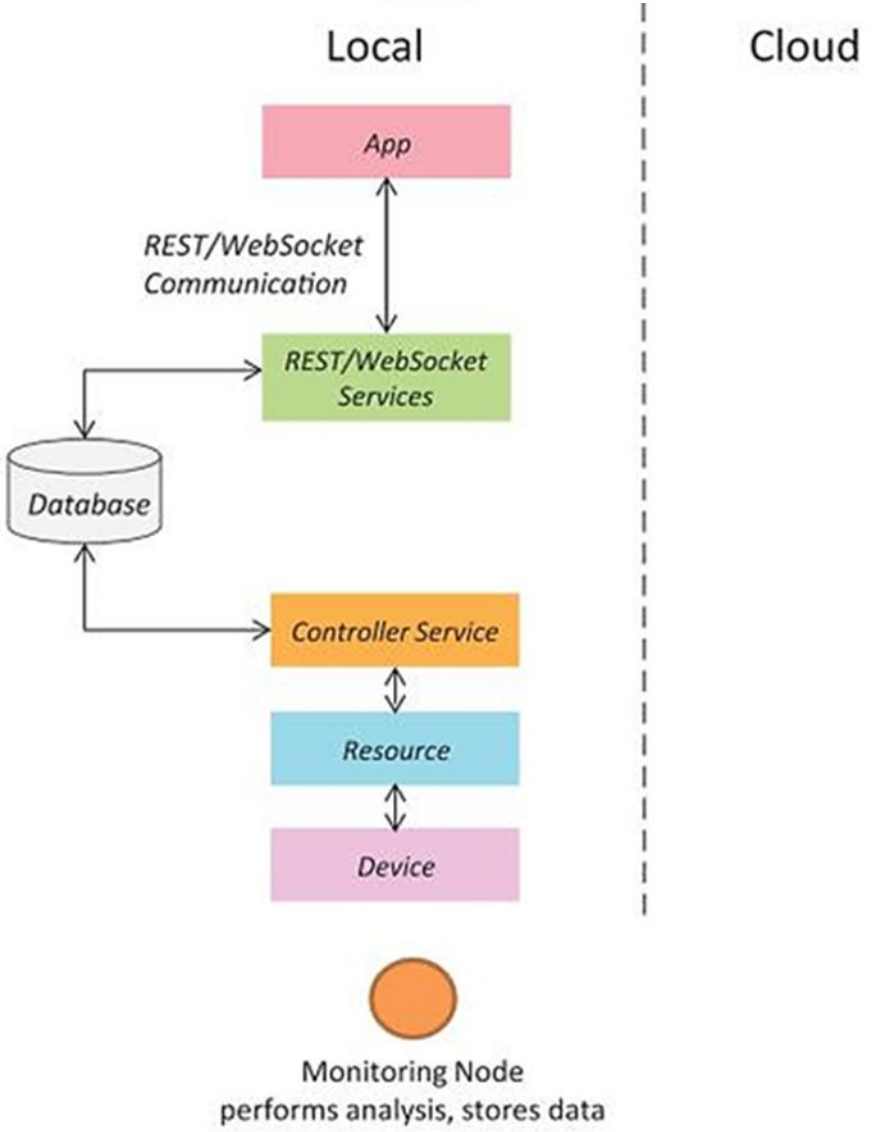
4. Database: Database can be either local or in the cloud and stores the data generated by the IoT device.

5. Web service: Serve as a link between the device, application database and analysis components. Web Services can be implemented using HTTP and REST principles or using website protocol.

# IoT Levels & Deployment Templates

## IoT level 1

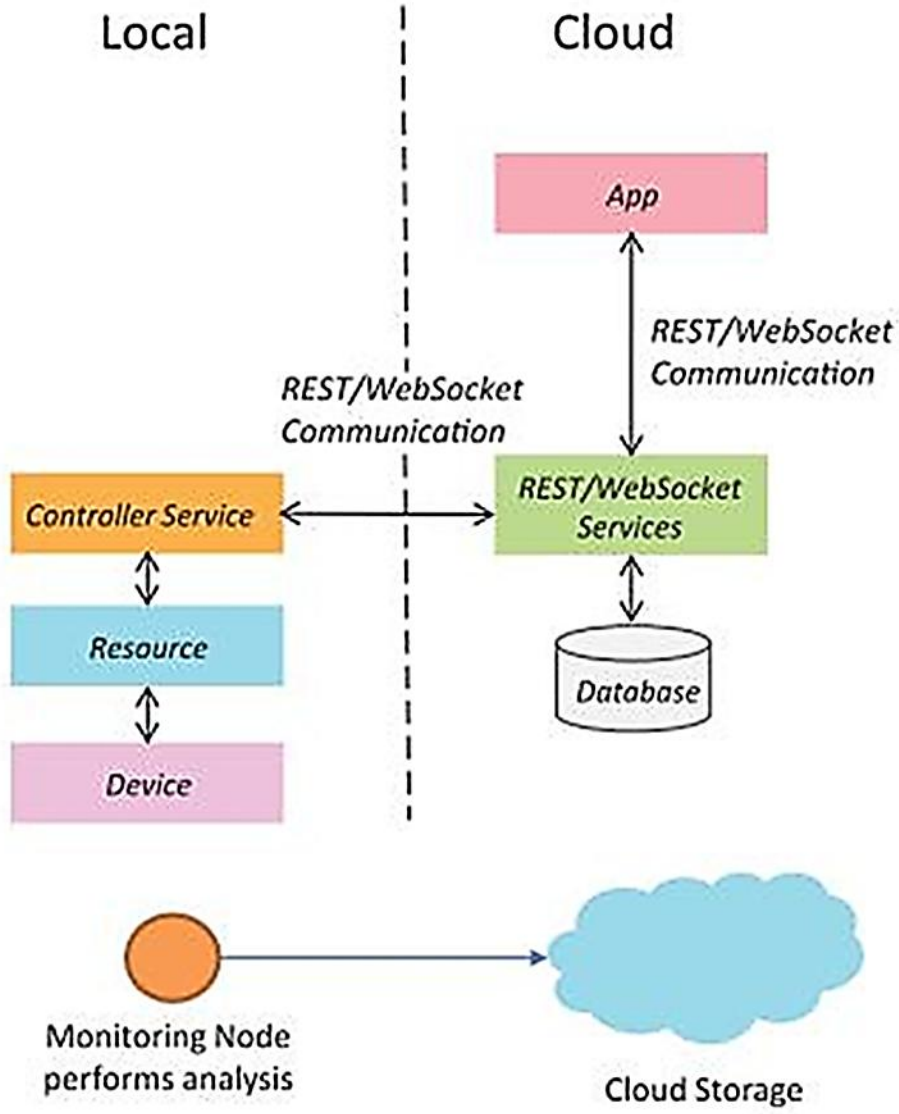
- Level 1 IoT system has a single node/ device that performs sensing and/or actuation, stores data, reforms analysis and the host to the application.
- Level 1 IoT systems are suitable for modelling low cost and low complexity solutions where the data involving is not big, and the analysis requirements are not computationally intensive.



# IoT Levels & Deployment Templates

## IoT level 2

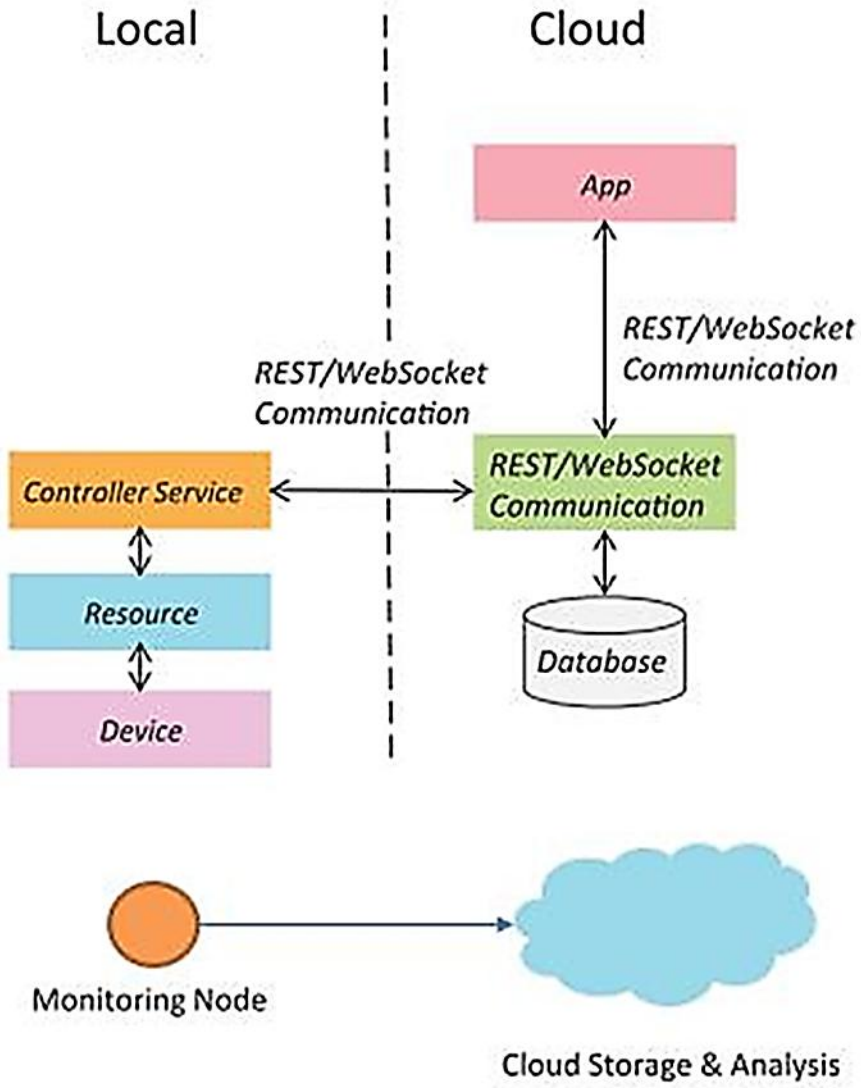
- Level 2 IoT system has a single node that performs sensing and/or actuation and local analysis.
- Data is stored in the cloud and application is usually cloud based systems are suitable for solutions where the data in world is big, however the primary analysis requirement is not computationally intensive and can be done local itself.



# IoT Levels & Deployment Templates

## IoT Level 3

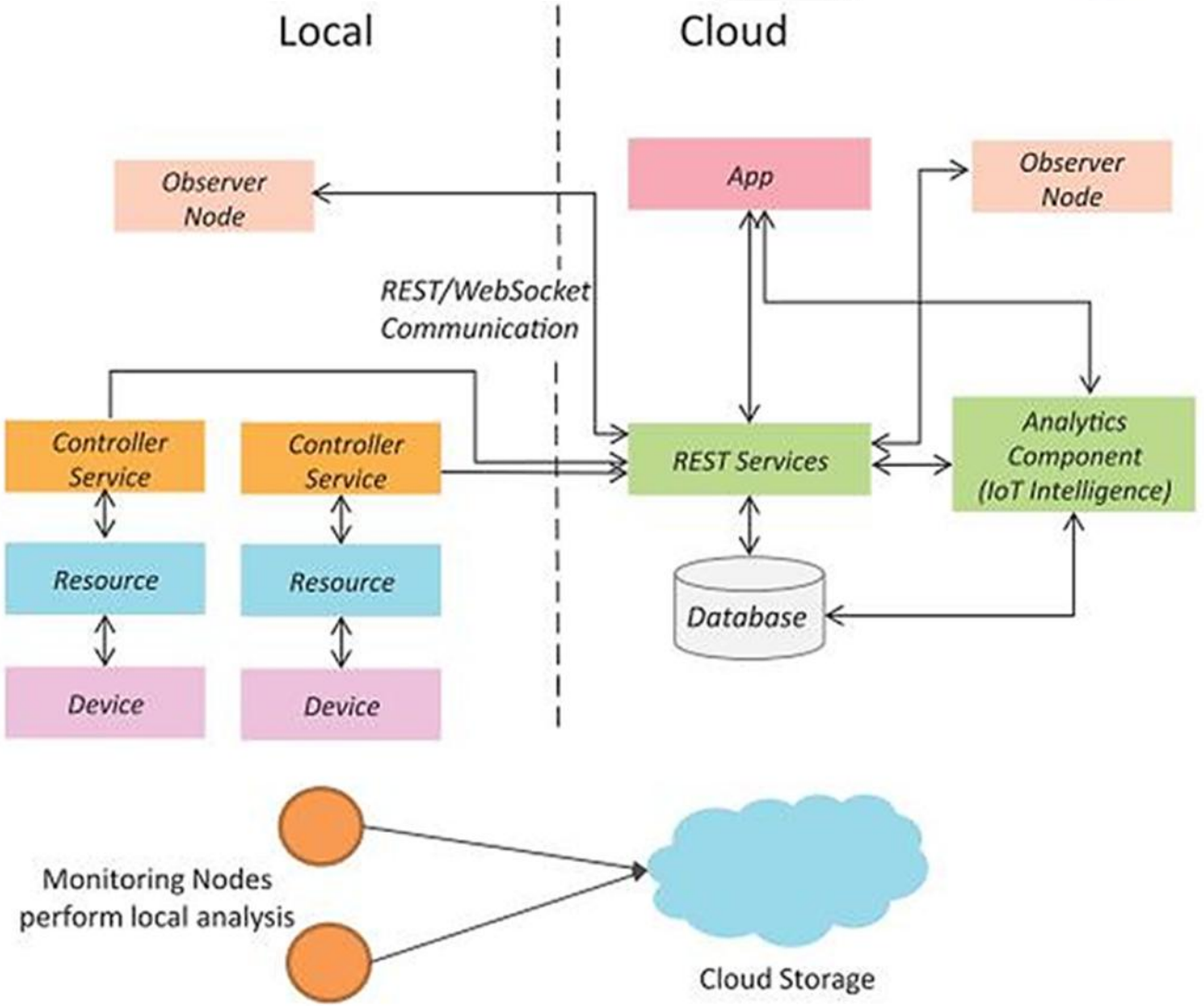
- Level 3 system has a single node. Data is stored and analysed in the cloud application is cloud-based.
- Level 3 IoT system suitable for solutions where the data involved is big and analysis requirements computationally intensive.



# IoT Levels & Deployment Templates

**IoT level 4**

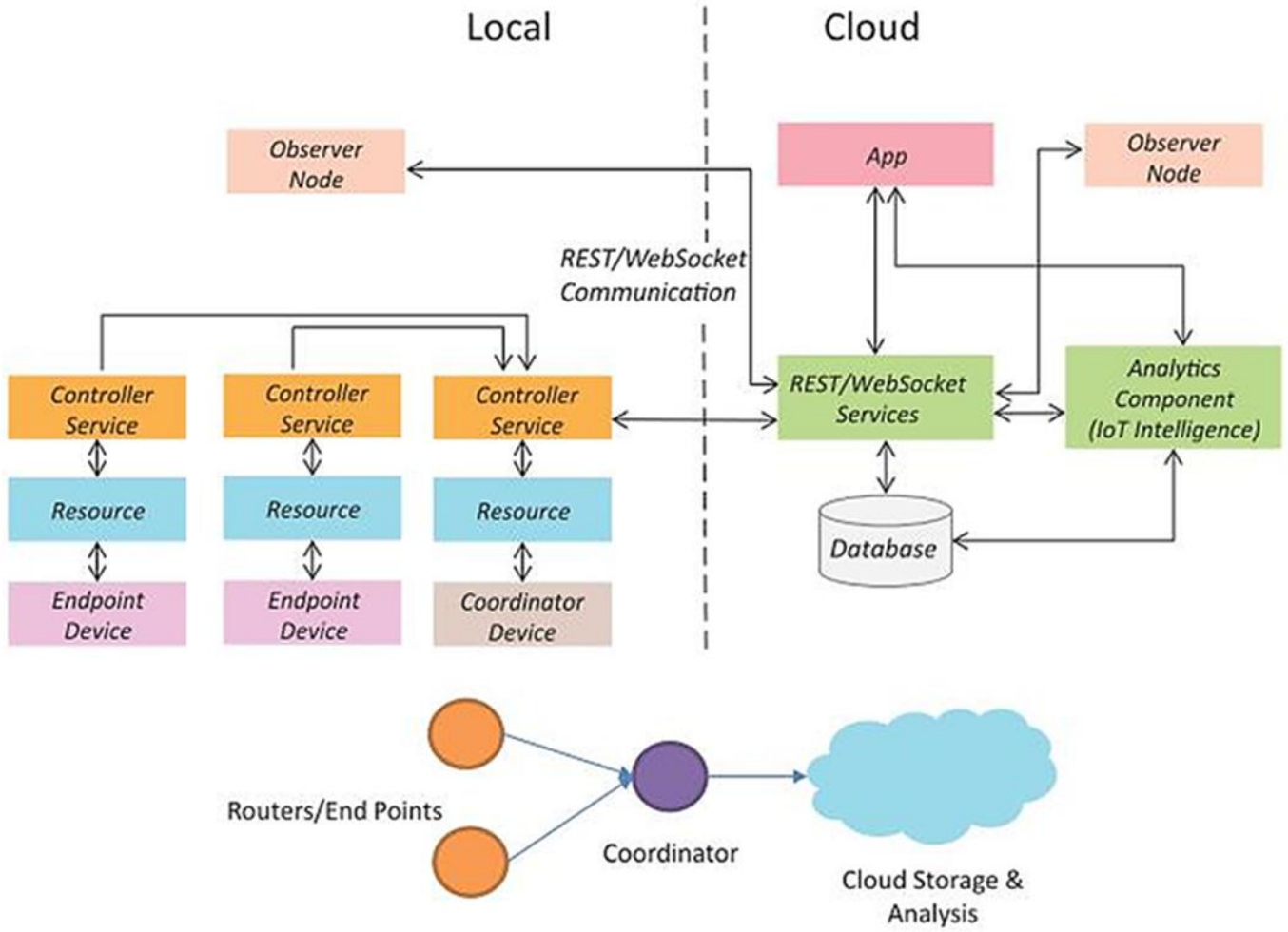
- A level 4 IoT system has multiple notes that perform local analysis.
- Data is stored in the cloud and application is cloud based, level 4 contains local and cloud-based observer notes which can subscribe to and receive information collected in the cloud from IoT devices.



# IoT Levels & Deployment Templates

**IoT Level 5**

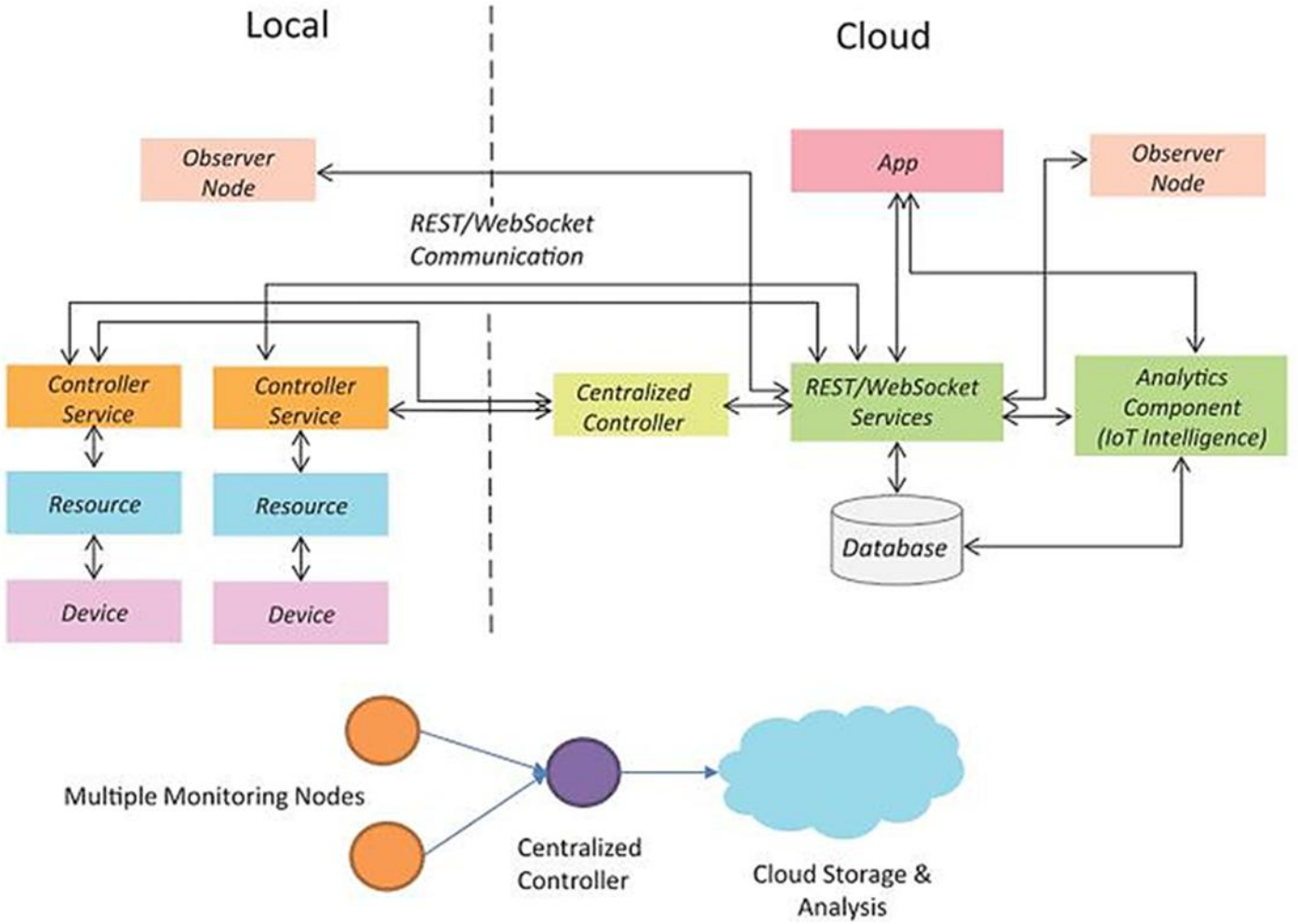
- IoT system has multiple end nodes and one coordinator notes and notes that perform sensing and / or actuation.
- Coordinator node collects data from the entry and send to the cloud.
- Data is stored and analysed in the cloud and applications is cloud based.



# IoT Levels & Deployment Templates

**IoT Level 6**

- IoT Level 6 system has multiple Independent and nodes that perform sensing and / or actuations and send data to the cloud.
- Data is stored in the cloud and applications is cloud based.



## Module - 2

# Chapter 1: IoT and M2M

By,
Roopa B
Asst. Professor
Dept. of CSE
ATMECE, Mysuru

# Contents

- Introduction to M2M
- Difference between IoT and M2M
- SDN and NFV for IOT

# Introduction to M2M

- Machine to machine (M2M) refers to networking of Machines for the purpose of remote monitoring and control and data exchange.
- An M2M area network comprises of machines which have embedded hardware module for sensing actuation and communication.
- Various Communication protocols can be used for M2M local area network such as Zigbee, Bluetooth, Modbus M–bus, wireless, power LINE Communication, 6LoWPAN.

# Difference between IoT and M2M

| M2M | IoT |
|---|---|
| M2M is about direct communication between machines. | The IoT is about sensors automation and Internet platform. |
| It supports point-to-point communication. | It supports cloud communication. |
| Devices do not necessarily rely on an Internet connection. | Devices rely on an Internet connection. |
| M2M is mostly hardware-based technology. | The IoT is both hardware- and software-based technology. |
| Machines normally communicate with a single machine at a time. | Many users can access at one time over the Internet. |
| A device can be connected through mobile or other network. | Data delivery depends on the Internet protocol (IP) network. |

# SDN and NFV for IoT

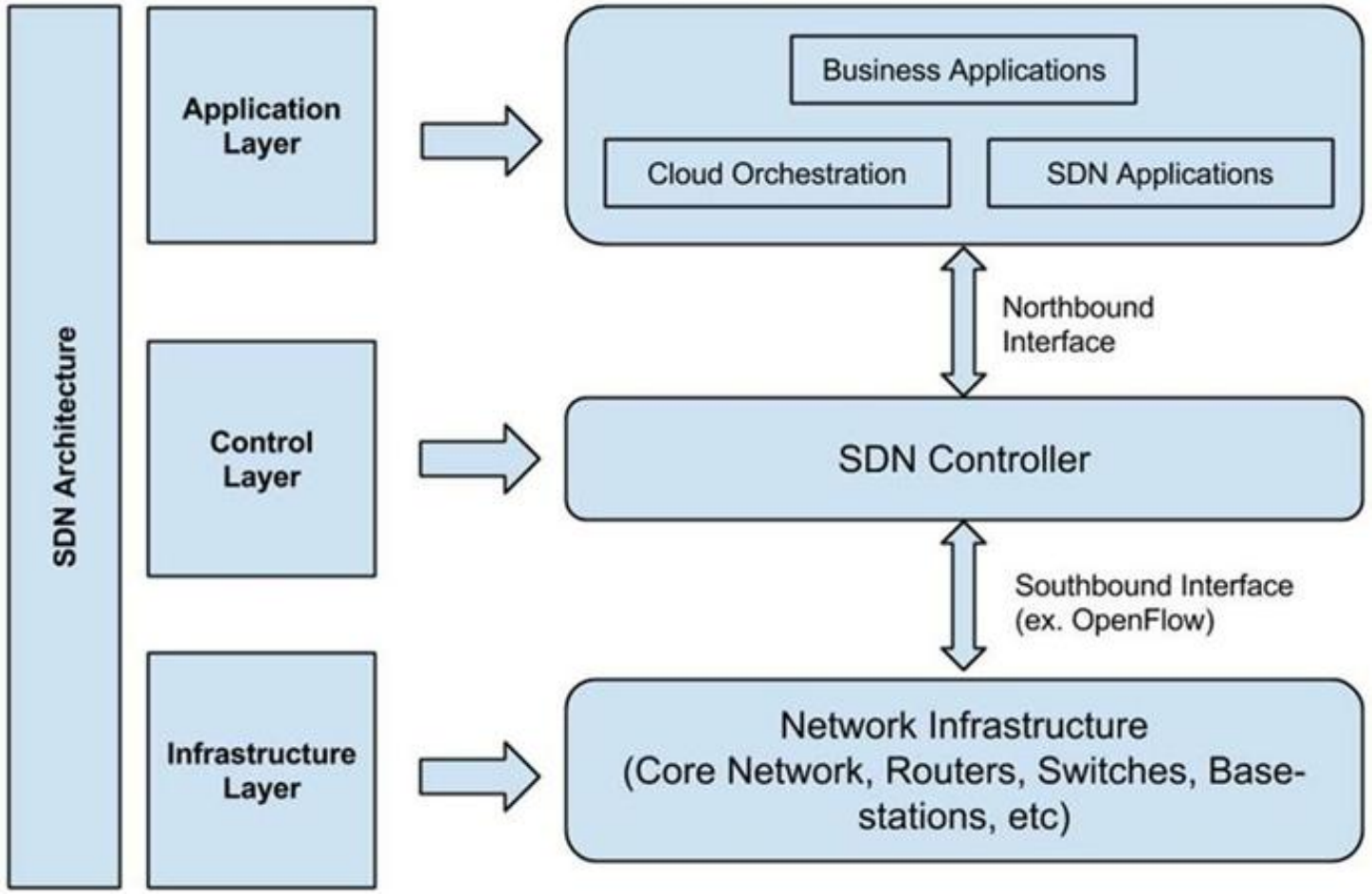
Software Defined Networking (SDN)

- SDN is the networking architecture that separates the control plan from the data plan and centralizes race the network controller. Conventional network architecture builds with specialized hardware (switches, router etc).

The limitations of the conventional network architecture as follows:

- Complex network devices
- Management overhead
- Limited scalability

# SDN and NFV for IoT

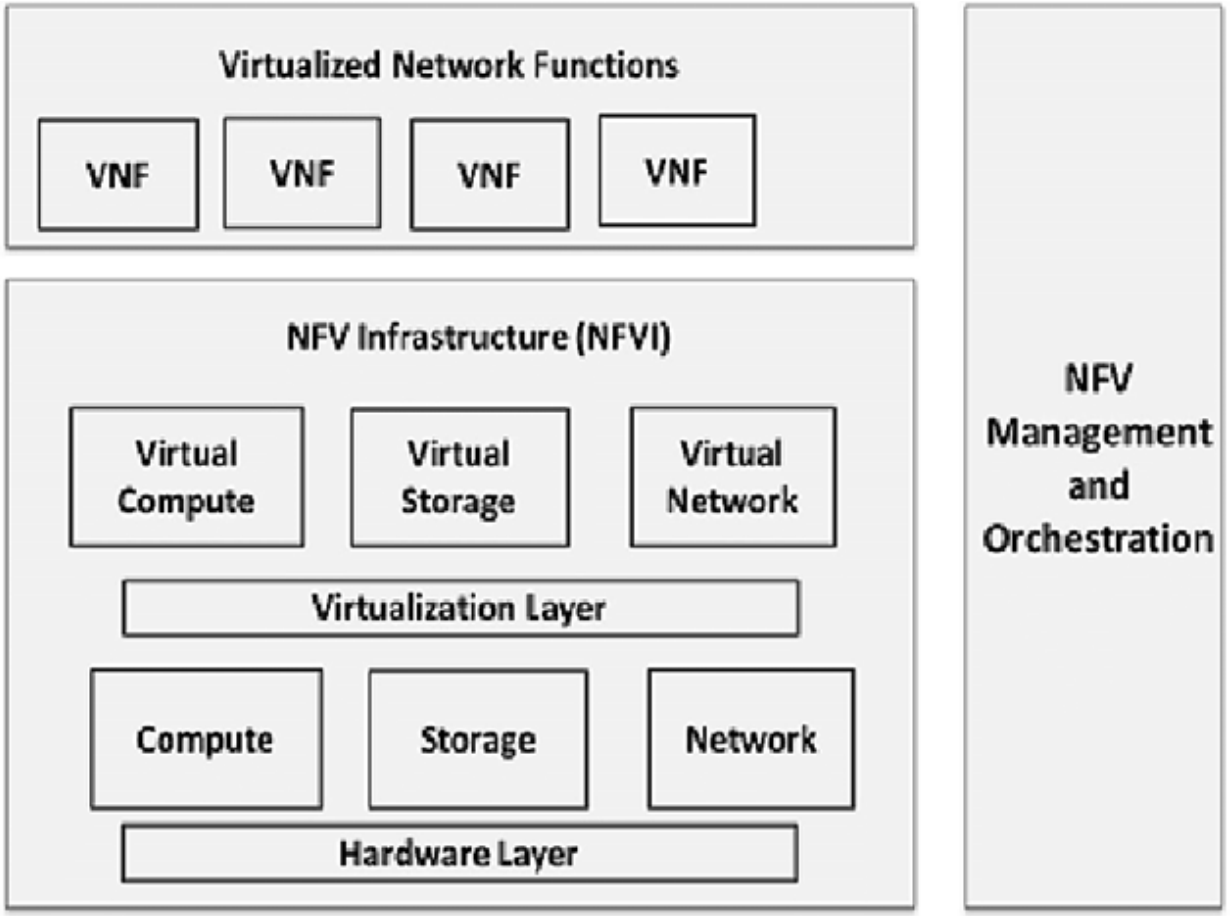


# SDN and NFV for IoT

Network function virtualization (NFV)

- Network function virtualization is a technology that leverages virtualization to consolidate the heterogeneous network devices on to industry standard high-volume service switches and storage.
- NFV is complementary to SDN as NFV can provide at the infrastructure on which SDN can run. NFV and SDN mutually beneficial to each other but not dependent. Network functions can be virtualized without SDN, similarly SDN can run without NFV.

# SDN and NFV for IoT



## Module - 2

# Chapter 2: IoT System Management with NETCONF-YANG

By,
Roopa B
Asst. Professor
Dept. of CSE
ATMECE, Mysuru

# Contents

- Need for IoT Systems Management
- Simple Network Management Protocol (SNMP)
- Network operator requirements
- NETCONF
- YANG
- IoT Systems Management with NETCONF-YANG.

# Need for IoT Systems Management

Managing multiple devices within a single system requires advanced management capabilities. The need for managing IoT systems is described as follows:

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

# Simple Network Management Protocol (SNMP)

- SNMP allows monitoring and configuring network devices such as routers, switches, servers, printers etc. SNMP is an application layer protocol that uses UDP as the transport protocol.
- Entities involved in managing a device with SNMP are Network Management Station (NMS), Managed devices, Management information base (MIB) and the SNMP Agent.

**Limitations of SNMP**

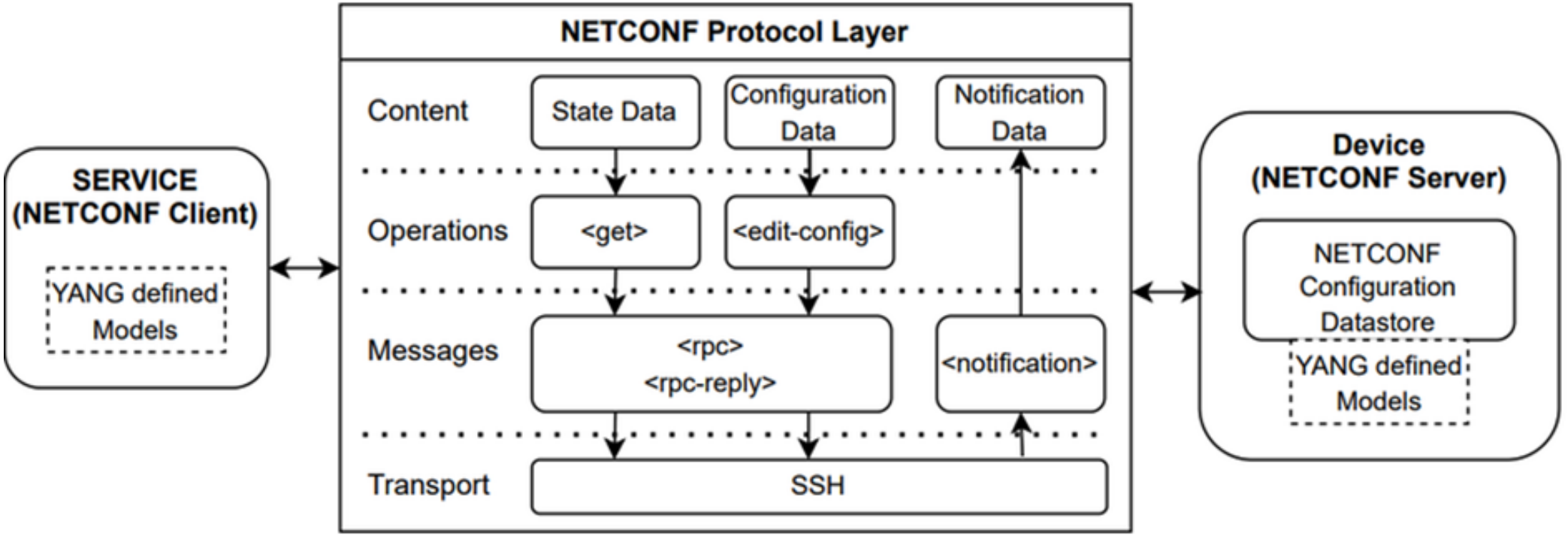
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features making the management information vulnerable to network intruders.

# Network operator requirements

- Ease of use
- Distinction between configuration and state data
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration validation
- Configuration database schemas
- Comparing configuration
- Role based access control
- Consistency of access control lists
- Multiple configuration sets
- Support for both data-oriented and task-oriented access control

# NETCONF

- Network Configuration Protocol is a session-based network management protocol. It allows retrieving state or configuration data and manipulating configuration data on network devices.

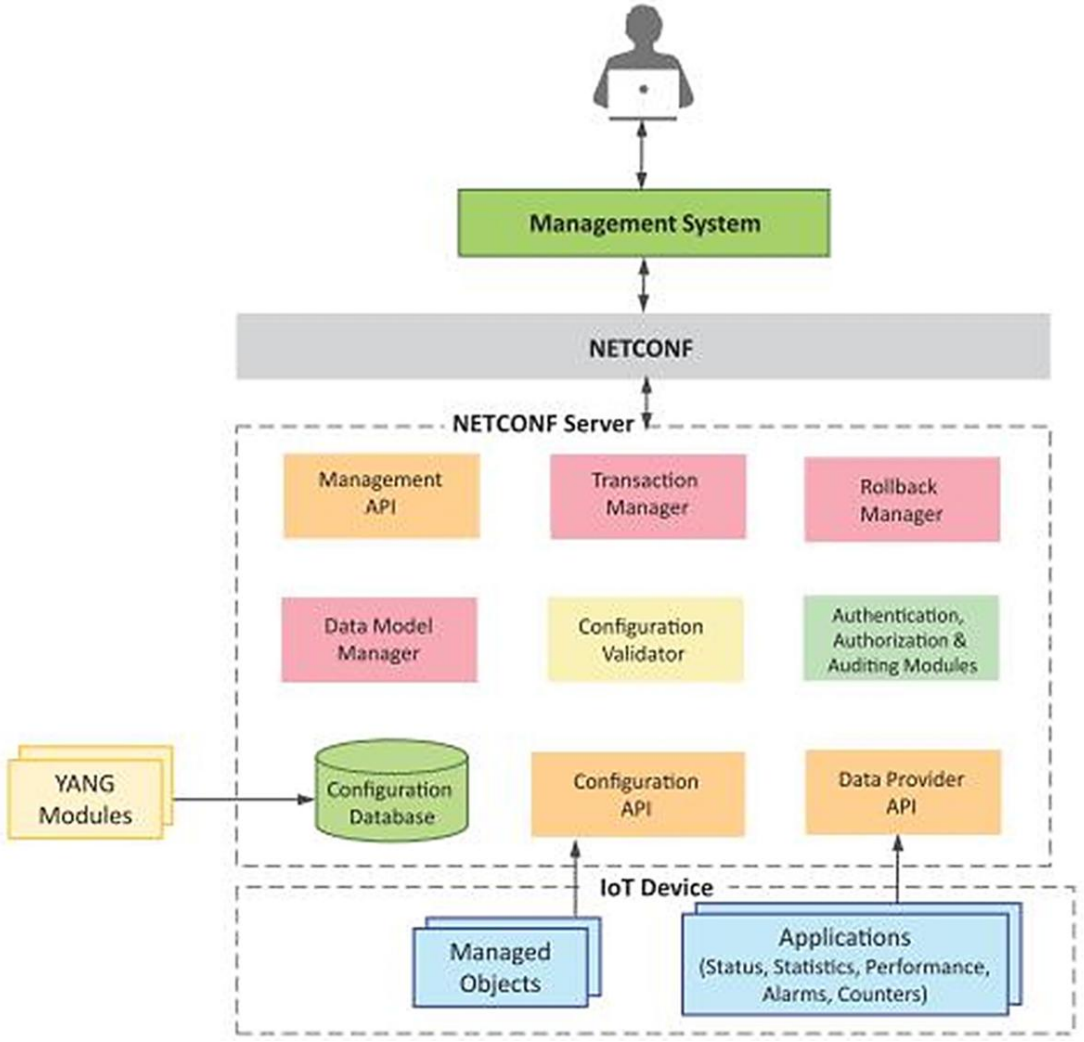


# YANG

- YANG is a data modelling language used to model configuration and state data manipulated by the NETCONF protocol. YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.

| Node Type | Description |
|---|---|
| **Leaf Nodes** | Contains simple data structures such as an integer or a string. Leaf has exactly one value of a particular type and no child nodes. |
| **Leaf-list Nodes** | Is a sequence of leaf nodes with exactly one value of a particular type per leaf. |
| **Container Nodes** | Used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers and leaf-lists). |
| **List Nodes** | Defines a sequence of list entries. Each entry is like a structure or a record instance and is uniquely identified by the value of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type. |

# IoT Systems Management with NETCONF-YANG



# NETOPEER

- It is set of open source NETCONF tools built on the Libnetconf library.

The Netopeer tools include:

- Netopeer-server: It is a NETCONF protocol server that runs on the managed device. Netopeer-server provides an environment for configuring the device using NETCONF RPC operations and also retrieving the state data from the device.
- Netopeer-agent: It is the NETCONF protocol agent running as a SSH/TLS subsystem. It accepts incoming NETCONF connection and passes the NETCONF RPC operations received from the NETCONF client to the Netopeer server.
- Netopeer-cli: It is a NETCONF client that provides a command line interface for interacting with the Netopeer-server. The operator can use the Netopeer-cli from the management system to send NETCONF RPC operations for configuring the device and retrieving the state information.
- Netopeer-manager: Netopeer-manager allows managing the YANG and Libnetconf Transaction API (TransAPI) modules on the Netopeer-server. With Netopeer-manager modules can be loaded or removed from the server.
- Netopeer-configurator: It is a tool that can be used to configure the Netopeer-server.

# NETOPEER

**Steps for IoT device Management with NETCONF-YANG**

1) Create a YANG model of the system that defines the configuration and state data of the system.

2) Complete the YANG model with the 'Inctool' which comes with Libnetconf.

3) Fill in the IoT device management code in the Trans API module.

4) Build the callbacks C file to generate the library file.

5) Load the YANG module and the TransAPI module into the Netopeer server using Netopeer manager tool.

6) The operator can now connect from the management system to the Netopeer server using the Netopeer CLI.

7) Operator can issue NETCONF commands from the Netopeer CLI. Command can be issued to change the configuration data, get operational data or execute an RPC on the IoT device.

## Module - 3

# Chapter-1: IoT Design Methodology

By,
Roopa B
Asst. Professor
Dept. of CSE
ATMECE, Mysuru

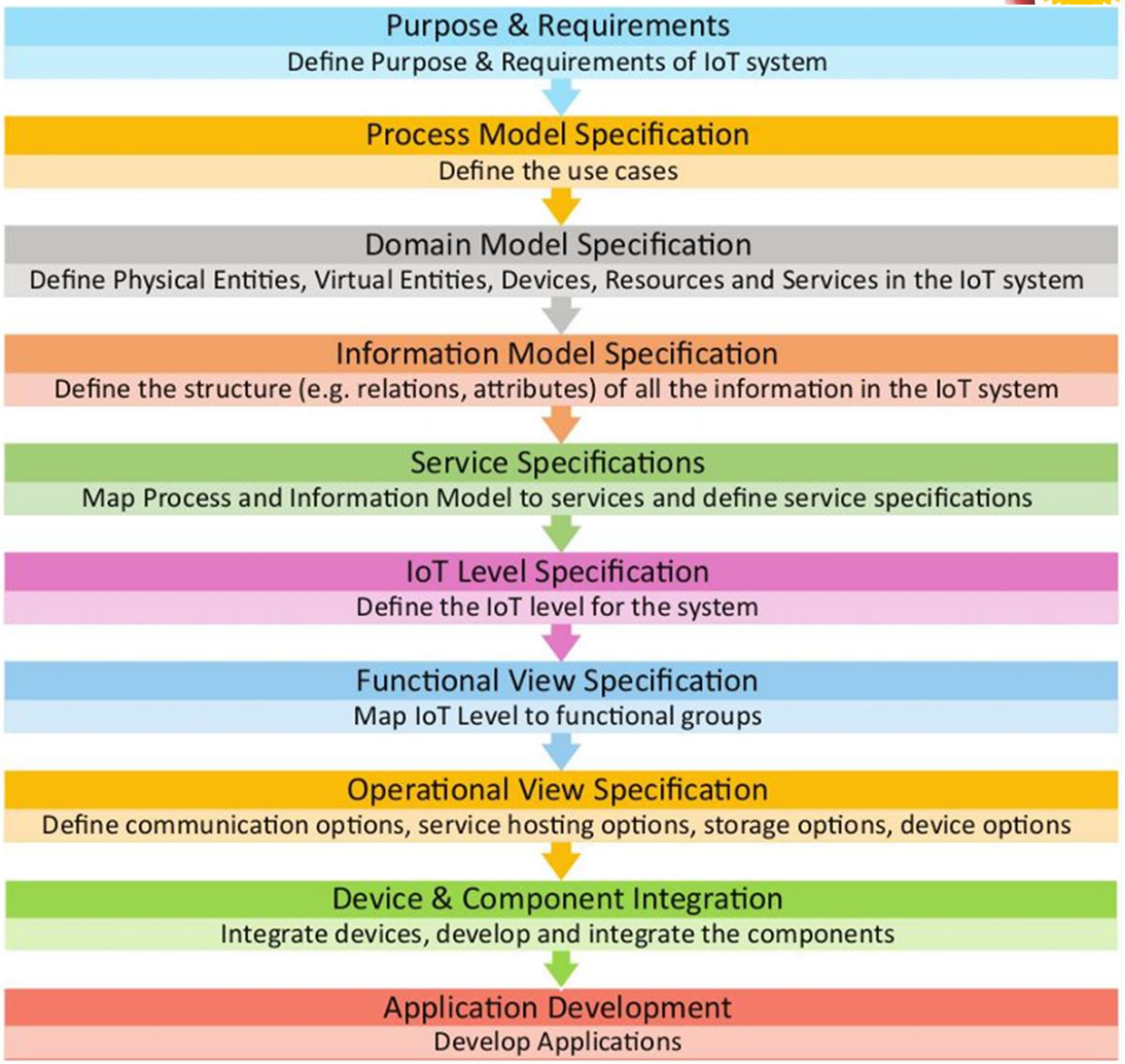
# Contents

- Introduction
- IoT Design Methodology
- Case Study on IoT System for Weather Monitoring

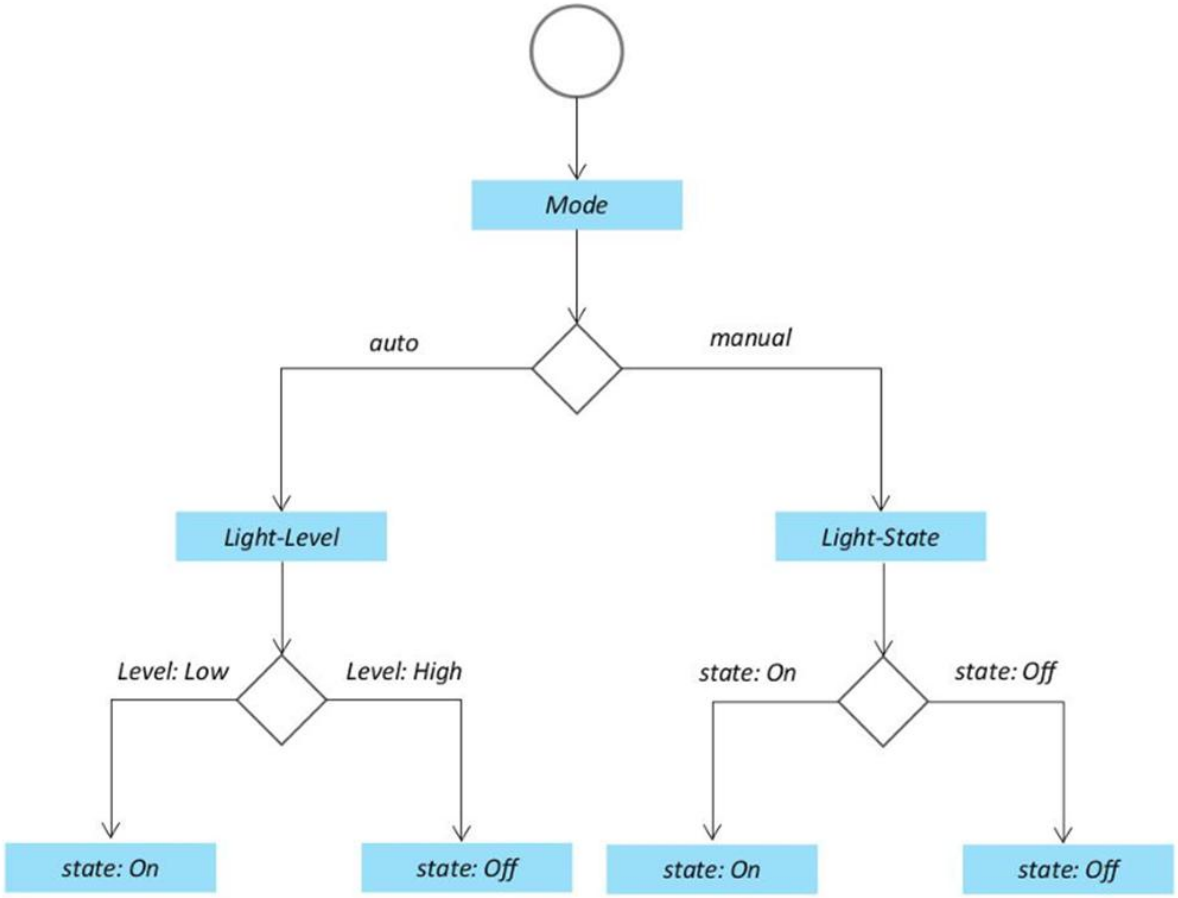
# IoT Design Methodology

- Designing IoT systems can be a complex task as these systems involve interactions between various components such as IoT device and network resources, web services, analytics components, application and database servers.
- IoT system designers often tend to design IoT systems keeping specific product or services in mind. For such systems, updating the system design to add new feature becomes very complex, and in many cases may require complete re-design of the system.

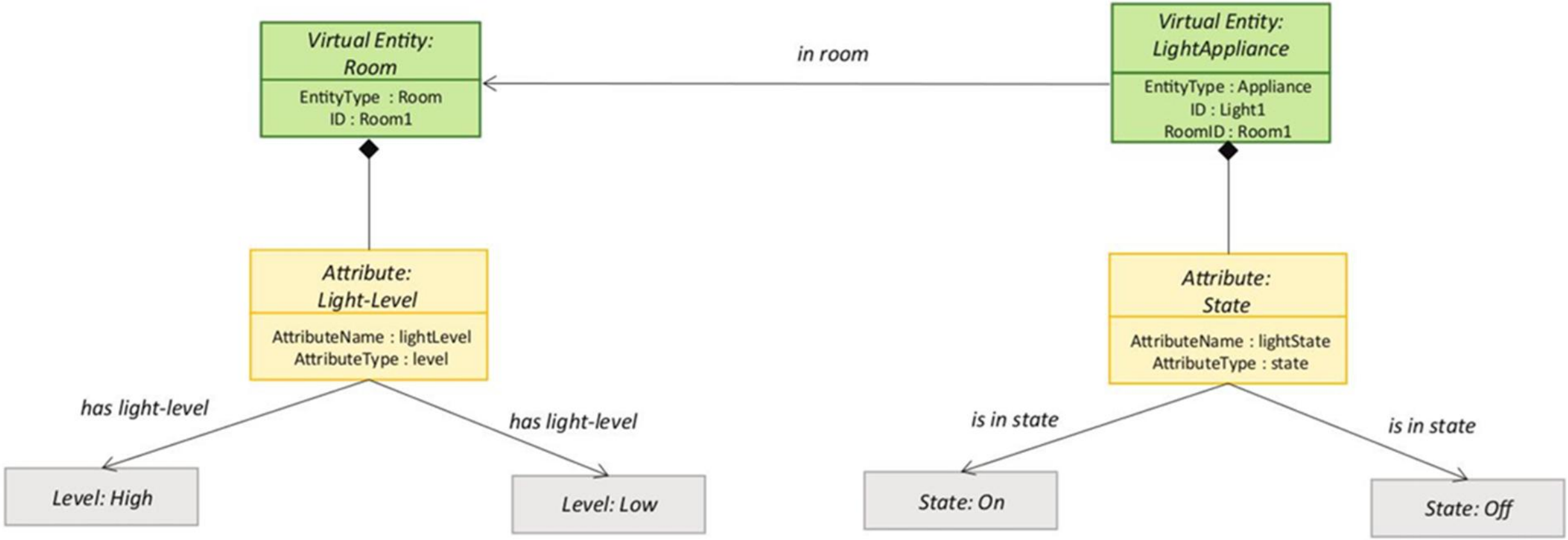
# IoT Design Methodology



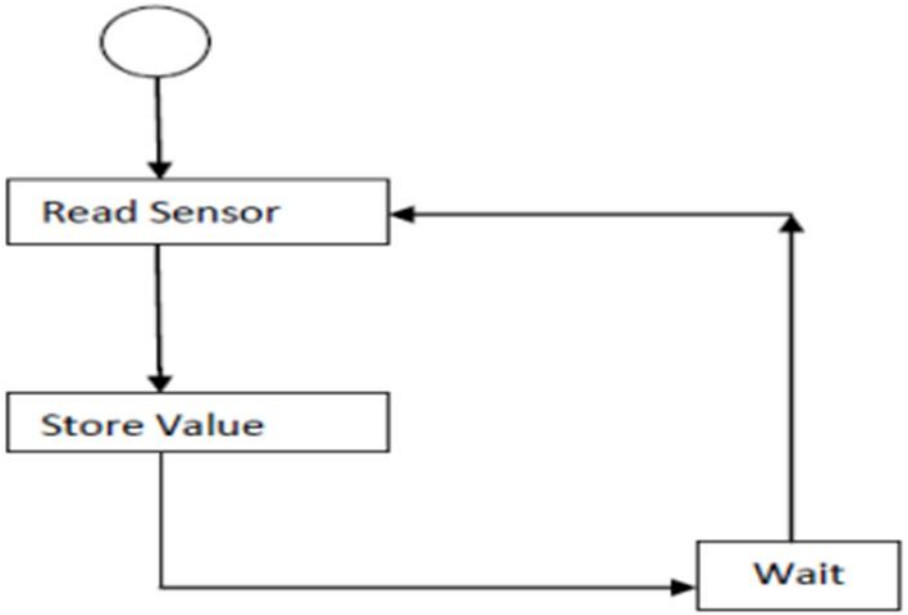
# Smart Home Automation System

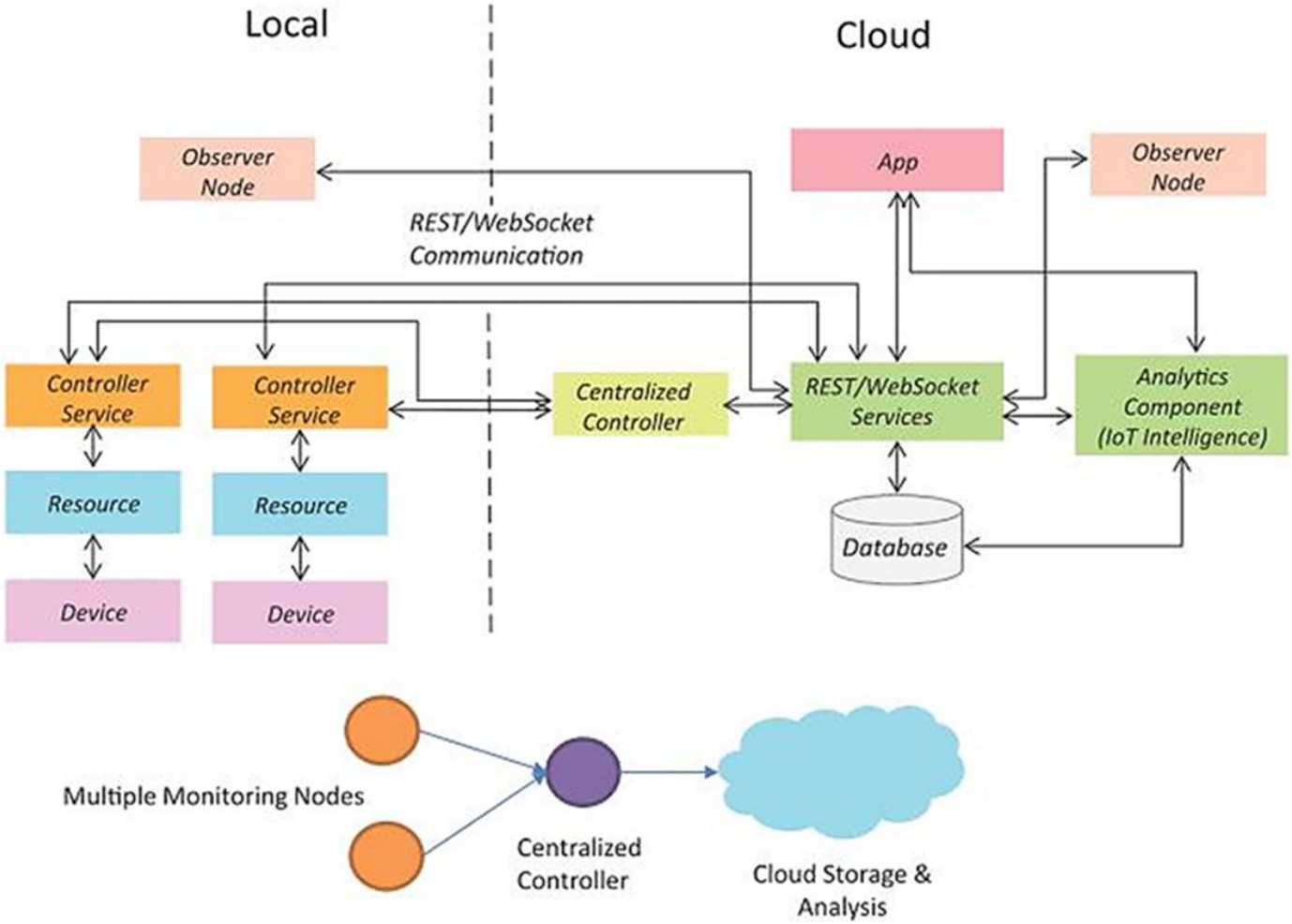



# Smart Home Automation System



# Case Study on IoT System for Weather Monitoring



## Module - 3

# Chapter-2: IoT Systems – Logical Design using Python

By,
Roopa B
Asst. Professor
Dept. of CSE
ATMECE, Mysuru

# Contents

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

# Python

- Python is a general-purpose high level programming language

The main characteristics of Python are:

- Multi-paradigm programming language: Python supports more than one programming paradigms including object-oriented programming and structured programming
- Interpreted Language: Python is an interpreted language and does not require an explicit compilation step. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.
- Interactive Language: Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

# Python data types & Data Structures

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

```
#Integer
>>>a=5
>>>type(a)
<type 'int'>

#Floating Point
>>>b=2.5
>>>type(b)
<type 'float'>

#Long
>>>x=9898878787676L
>>>type(x)
<type 'long'>

#Complex
>>>y=2+5j
>>>y
(2+5j)
>>>type(y)
<type 'complex'>
>>>y.real
2
>>>y.imag
5
```

# Python data types & Data Structures

A string is simply a list of characters in order.

```
#Create string
>>>s="Hello World!"
>>>type(s)
<type 'str'>

#String concatenation
>>>t="This is sample program."
>>>r = s+t
>>>r
'Hello World!This is sample program.'

#Get length of string
>>>len(s)
12

#Convert string to integer
>>>x="100"
>>>type(s)
<type 'str'>
>>>y=int(x)
>>>y
100
```

```
#Print string
>>>print s
Hello World!

#Formatting output
>>>print "The string (The string (Hello World!)
has 12 characters

#Convert to upper/lower case
>>>s.upper()
'HELLO WORLD!'
>>>s.lower()
'hello world!'

#Accessing sub-strings
>>>s[0]
'H'
>>>s[6:]
'World!'
>>>s[6:-1]
'World'
```

# Python data types & Data Structures

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

```
#Create List
>>>fruits=['apple','orange','banana','mango']
>>>type(fruits)
<type 'list'>

#Get Length of List
>>>len(fruits)
4

#Access List Elements
>>>fruits[1]
'orange'
>>>fruits[1:3]
['orange', 'banana']
>>>fruits[1:]
['orange', 'banana', 'mango']

#Appending an item to a list
>>>fruits.append('pear')
>>>fruits
['apple', 'orange', 'banana', 'mango', 'pear']
```

# Python data types & Data Structures

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

```
#Create a Tuple
>>>fruits=("apple","mango","banana","pineapple")
>>>fruits
('apple', 'mango', 'banana', 'pineapple')

>>>type(fruits)
<type 'tuple'>

#Get length of tuple
>>>len(fruits)
4
```

```
#Get an element from a tuple
>>>fruits[0]
'apple'
>>>fruits[:2]
('apple', 'mango')

#Combining tuples
>>>vegetables=('potato','carrot','onion','radish')
>>>eatables=fruits+vegetables
>>>eatables
('apple', 'mango', 'banana', 'pineapple', 'potato', 'carrot', 'onion', 'radish')
```

# Python data types & Data Structures

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

```
#Create a dictionary
>>>student={'name':'Mary','id':'8776','major':'CS'}
>>>student
{'major': 'CS', 'name': 'Mary', 'id': '8776'}
>>>type(student)
<type 'dict'>

#Get length of a dictionary
>>>len(student)
3

#Get the value of a key in dictionary
>>>student['name']
'Mary'

#Get all items in a dictionary
>>>student.items()
[('gender', 'female'), ('major', 'CS'), ('name', 'Mary'),
('id', '8776')]
```

# Control Flow – if statement

```
>>>a = 25**5
>>>if a>10000:
                print "More"
        else:
                print "Less"

More
```

```
>>>if a>10000:
        if a<1000000:
                print "Between 10k and 100k"
        else:
                print "More than 100k"
        elif a==10000:
                print "Equal to 10k"
        else:
                print "Less than 10k"

More than 100k
```

```
>>>s="Hello World"
>>>if "World" in s:
        s=s+"!"
print s

Hello World!
```

```
>>>student={'name':'Mary','id':'8776'}
>>>if not student.has_key('major'):
        student['major']='CS'

>>>student
{'major': 'CS', 'name': 'Mary', 'id': '8776'}
```

# Control Flow – for and while statement

```
#Looping over characters in a string

helloString = "Hello World"

for c in helloString:
        print c
```

```
#Looping over items in a list

fruits=['apple','orange','banana','mango']

i=0
for item in fruits:
        print "Fruit-%d: %s" % (i,item)
        i=i+1
```

```
#Prints even numbers upto 100

>>> i = 0

>>> while i<=100:
if i%2 == 0:
    print i
i = i+1
```

# Control Flow – break, continue statements

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

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

apple
orange
mango
```

# Functions

- A function is a block of code that takes information in (in the form of parameters), does some computation, and returns a new piece of information based on the parameter information.

```
students = { '1': {'name': 'Bob', 'grade': 2.5},
             '2': {'name': 'Mary', 'grade': 3.5},
             '3': {'name': 'David', 'grade': 4.2},
             '4': {'name': 'John', 'grade': 4.1},
             '5': {'name': 'Alex', 'grade': 3.8}}


def averageGrade(students):
        "This function computes the average grade"
        sum = 0.0
        for key in students:
                sum = sum + students[key]['grade']
                average = sum/len(students)
        return average


avg = averageGrade(students)
print "The average garde is: %0.2f" % (avg)
```

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

#Using default arguments
>>>displayFruits()
apple
orange


>>>fruits = ['banana', 'pear', 'mango']
>>>displayFruits(fruits)
banana
pear
mango
```

# Functions

Functions can also be called using keyword arguments that identifies the arguments by the parameter name when the function is called. Python functions can have variable length arguments. The variable length arguments are passed to as a tuple to the function with an argument prefixed with asterix (*)

```
>>>def displayFruits(fruits):
        print "There are %d fruits in the list" % (len(fruits))
        for item in fruits:
                print item
        print "Adding one more fruit"
        fruits.append('mango')

>>>fruits = ['banana', 'pear', 'apple']
>>>displayFruits(fruits)
There are 3 fruits in the list
banana
pear
apple

#Adding one more fruit
>>>print "There are %d fruits in the list" % (len(fruits))
There are 4 fruits in the list
```

```
#name is a formal argument.
#**kwargs is a keyword argument that receives all arguments except the formal argument as a dictionary.

>>>def student(name, **kwargs):
        print "Student Name: " + name
        for key in kwargs:
              print key + ': ' + kwargs[key]

>>>student(name='Bob', age='20', major = 'CS')
Student Name: Bob
age: 20
major: CS
```

```
>>>def student(name, *varargs):
        print "Student Name: " + name
                for item in varargs:
        print item

>>>student('Nav')
Student Name: Nav

>>>student('Amy', 'Age: 24')
Student Name: Amy
Age: 24

>>>student('Bob', 'Age: 20', 'Major: CS')
Student Name: Bob
Age: 20
Major: CS
```

# File Handling

Python allows reading and writing to files using the file object.

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

**# Example of reading an entire file**

```
>>>fp = open('file.txt','r')
>>>content = fp.read()
>>>print content
This is a test file.
>>>fp.close()
```

**# Example of reading line by line**

```
>>>fp = open('file1.txt','r')
>>>print "Line-1: " + fp.readline()
Line-1: Python supports more than one programming paradigms.
>>>print "Line-2: " + fp.readline()
Line-2: Python is an interpreted language.
>>>fp.close()
```

**# Example of reading lines in a loop**

```
>>>fp = open('file1.txt','r')
>>>lines = fp.readlines()
>>>for line in lines:
        print line

Python supports more than one programming paradigms.
Python is an interpreted language.
```

# File Handling

**# Example of reading a certain number of bytes**

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>fp.close()
```

**# Example of seeking to a certain position**

```
>>>fp = open('file.txt','r')
>>>fp.seek(10,0)
>>>content = fp.read(10)
>>>print content
ports more
>>>fp.close()
```

**# Example of getting the current position of read**

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>currentpos = fp.tell
>>>print currentpos
<built-in method tell of file object at 0x0000000002391390>
>>>fp.close()
```

**# Example of writing to a file**

```
>>>fo = open('file1.txt','w')
>>>content='This is an example of writing to a file in
Python.'
>>>fo.write(content)
>>>fo.close()
```

# Modules

- A module is a Python file that defines some functionality in the form of functions or classes.
- Modules can be imported using the import keyword can be imported must be present in the search path.

```
#student module - saved as student.py
def averageGrade(students):
        sum = 0.0
        for key in students:
           sum = sum + students[key]['grade']
           average = sum/len(students)
        return average

def printRecords(students):
        print "There are %d students" %(len(students))
        i=1
        for key in students:
           print "Student-%d: " % (i)
           print "Name: " + students[key]['name']
           print "Grade: " + str(students[key]['grade'])
           i = i+1
```

```
# Importing a specific function from a module
>>>from student import averageGrade

# Listing all names defines in a module
>>>dir(student)
```

```
#Using student module
>>>import student
>>>students =  '1': 'name': 'Bob', 'grade': 2.5,
'2': 'name': 'Mary', 'grade': 3.5,
'3': 'name': 'David', 'grade': 4.2,
'4': 'name': 'John', 'grade': 4.1,
'5': 'name': 'Alex', 'grade': 3.8

>>>student.printRecords(students)
There are 5 students
Student-1:
Name: Bob
Grade: 2.5
Student-2:
Name: David
Grade: 4.2
Student-3:
Name: Mary
Grade: 3.5
Student-4:
Name: Alex
Grade: 3.8
Student-5:
Name: John
Grade: 4.1
```

# Packages

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

```
# skimage package listing

skimage/                    Top level package
        __init__.py         Treat directory as a package

        color/ color        color subpackage
                __init__.py
                colorconv.py
                colorlabel.py
                rgb_colors.py

        draw/ draw          draw subpackage
                __init__.py
                draw.py
                setup.py

        exposure/           exposure subpackage
                __init__.py
                _adapthist.py
                exposure.py

        feature/            feature subpackage
                __init__.py
                _brief.py
                _daisy.py
...
```

# Date/Time Operations

- Python provides several functions for date and time access and conversions.
- The datetime module allows manipulating date and time in several ways.
- The time module in Python provides various time-related functions.

```
# Examples of manipulating with date

>>>from datetime import date

>>>now = date.today()
>>>print "Date: " + now.strftime("%m-%d-%y")
Date: 07-24-13

>>>print "Day of Week: " + now.strftime("%A")
Day of Week: Wednesday

>>>print "Month: " + now.strftime("%B")
Month: July

>>>then = date(2013, 6, 7)
>>>timediff = now - then
>>>timediff.days
47
```

```
# Examples of manipulating with time

>>>import time
>>>nowtime = time.time()
>>>time.localtime(nowtime)
time.struct_time(tm_year=2013, tm_mon=7, tm_mday=24, tm_ec=51, tm_wday=2, tm_yday=205,
tm_isdst=0)

>>>time.asctime(time.localtime(nowtime))
'Wed Jul 24 16:14:51 2013'

>>>time.strftime("The date is %d-%m-%y. Today is a %A. It is %H hours, %M minutes and %S seconds now.")
'The date is 24-07-13. Today is a Wednesday. It is 16 hours, 15 minutes and 14 seconds now.'
```

# Classes

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

Instance/Object

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

Inheritance

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

```
# Examples of a class
class Student:
        studentCount = 0

        def __init__(self, name, id):
                print "Constructor called"
                self.name = name
                self.id = id
                Student.studentCount = Student.studentCount + 1
                self.grades={}

        def __del__(self):
                print "Destructor called"

        def getStudentCount(self):
                return Student.studentCount

        def addGrade(self,key,value):
                self.grades[key]=value
                def getGrade(self,key):
                return self.grades[key]

        def printGrades(self):
                for key in self.grades:
                        print key + ": " + self.grades[key]
```

```
>>>s = Student('Steve','98928')
Constructor called

>>>s.addGrade('Math','90')
>>>s.addGrade('Physics','85')
>>>s.printGrades()
Physics: 85
Math: 90

>>>mathgrade = s.getGrade('Math')
>>>print mathgrade
90

>>>count = s.getStudentCount()
>>>print count
1

>>>del s
Destructor called
```

# Classes

Class Inheritance

• In this example Shape is the base class and Circle is the derived class. The class Circle inherits the attributes of the Shape class.

• The child class Circle overrides the methods and attributes of the base class (eg. draw() function defined in the base class Shape is overridden in child class Circle).

```
# Examples of class inheritance
class Shape:
        def __init__(self):
                print "Base class constructor"
                self.color = 'Green'
                self.lineWeight = 10.0

        def draw(self):
                print "Draw - to be implemented"
                def setColor(self, c):
                self.color = c
                def getColor(self):
                return self.color

        def setLineWeight(self,lwt):
                self.lineWeight = lwt

        def getLineWeight(self):
                return self.lineWeight
```

```
class Circle(Shape):
        def __init__(self, c,r):
                print "Child class constructor"
                self.center = c
                self.radius = r
                self.color = 'Green'
                self.lineWeight = 10.0
                self.__label = 'Hidden circle label'

        def setCenter(self,c):
                self.center = c
                def getCenter(self):
                return self.center

        def setRadius(self,r):
                self.radius = r

        def getRadius(self):
                return self.radius

        def draw(self):
                print "Draw Circle (overridden function)"
```

```
class Point:
        def __init__(self, x, y):
                self.xCoordinate = x
                self.yCoordinate = y

        def setXCoordinate(self,x):
                self.xCoordinate = x

        def getXCoordinate(self):
                return self.xCoordinate

        def setYCoordinate(self,y):
                self.yCoordinate = y

        def getYCoordinate(self):
                return self.yCoordinate
```

```
>>>p = Point(2,4)
>>>circ = Circle(p,7)
Child class constructor
>>>circ.getColor()
'Green'
>>>circ.setColor('Red')
>>>circ.getColor()
'Red'
>>>circ.getLineWeight()
10.0
>>>circ.getCenter().getXCoordinate()
2
>>>circ.getCenter().getYCoordinate()
4
>>>circ.draw()
Draw Circle (overridden function)
>>>circ.radius
7
```

# Python Packages of Interest for IoT

**1. JSON**

- JavaScript object Notation (JSON) is an easy to read and write data-interchange format.
- JSON is used as an alternative to XML and is easy for machines to parse and generate.
- JSON is built on two structure- a collection of name-value pairs (e.g. a python dictionary) and ordered lists of values (e.g. a python list).

**2. XML**

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

**3. HTTPLib & URLLib**

- HTTPLib2 and URLLib2 are python libraries used in network/internet programming.
- HTTPLib2 is an HTTP client library and URLLib2 is a library for fetching URLS.

**4. SMTPLib**

- Simple Mail Transfer Protocol (SMTP) is a protocol which handle sending email and routing e-mail between mail servers.
- The python smtplib module provides an STMP client session object that can be used to send email.
- Example: sending email from a Gmail account. To send email from a Gmail account the Gmail STMP server is specified in the server string.