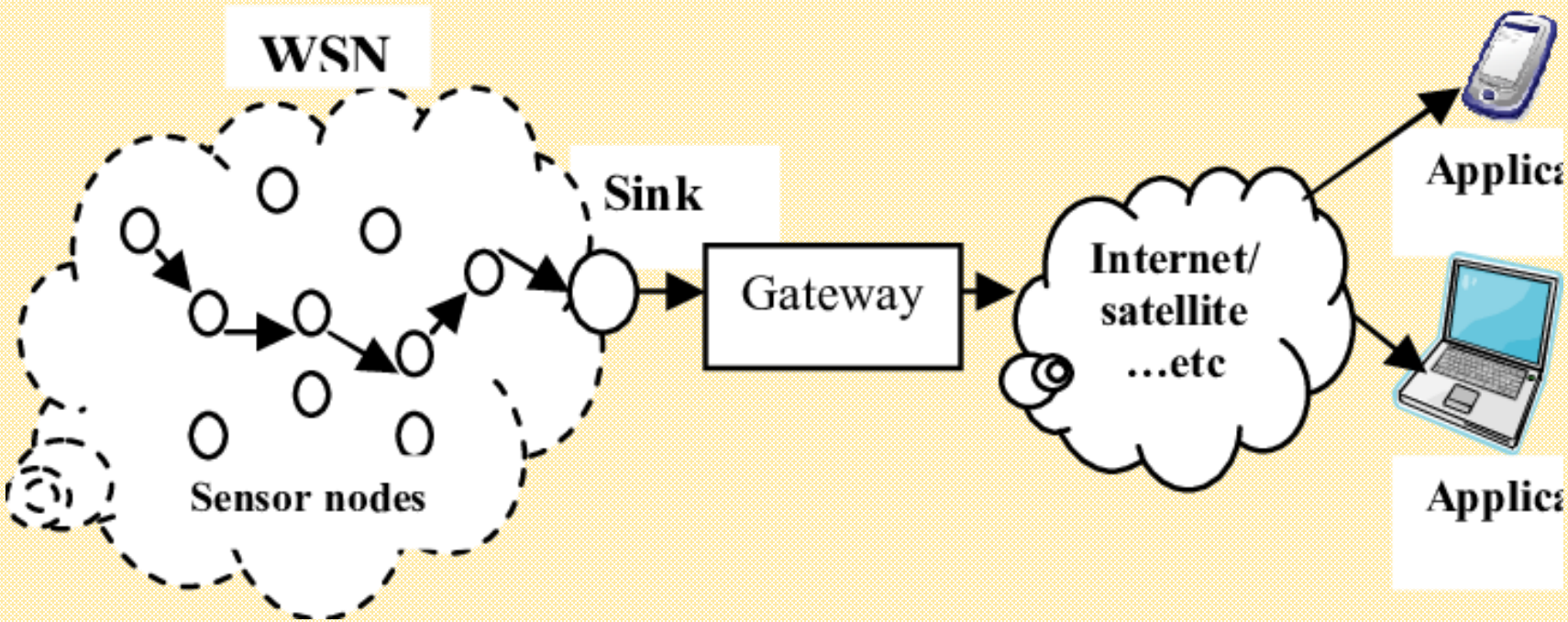
Wireless Sensor Network

- WSN: Things (sensor nodes) connected without a wire to gather some data.
- Wireless sensor network (WSN) refers to a group of dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location.
- Nodes are not directly connected to the Internet. Nodes route traffic to reach the sink node.
- WSN is sometime referred to as a subset of IoT.



The Internet of things (IoT)

IOT: WSN + Any physical object (Thing) + IP address + Internet + App + Cloud computing+ etc…

- Sensors send their data directly to the Internet because they have Internet Connection. The Internet of things (IoT) is the network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enable these objects to connect and exchange data.
- Each thing is uniquely identifiable through its embedded computing system but is able to inter-operate within the existing Internet infrastructure.
- All sensor data further processed and analyzed in the data analyzing area.

IoT Definitions

- The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

- 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.

- The Internet of things (IoT) describes physical objects (or groups of such objects) with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks.

- The Internet of Things (IoT) is a network of physical objects that are fitted with sensors, software and other technologies. Connected to the Internet, these ‘things’ are able to exchange real time data with other connected devices and systems over networks. These connected devices combine with automated systems to gather IoT data that can be analyzed to assist with tasks or learn how to improve a process.

WSN Vs IoT

Aspect	WSN (Wireless Sensor Network)	IoT (Internet of Things)
Definition	A network of spatially distributed sensor nodes to monitor physical or environmental conditions.	A broader ecosystem connecting various devices (including WSNs) to exchange data over the Internet.
Scope	Limited to sensing and short-range data transmission.	Encompasses sensing, processing, communication, and decision-making across devices globally.
Connectivity	Usually connected via local wireless protocols (e.g., Zigbee, Bluetooth, 6LoWPAN).	Uses both local (Wi-Fi, BLE) and global (Internet, cloud) connectivity.
Components	Mainly sensors, actuators, microcontrollers, and wireless transceivers.	Includes WSN components plus cloud platforms, AI, data analytics, and user interfaces.
Application Focus	Environmental monitoring, military, industrial sensing.	Smart homes, smart cities, healthcare, transportation, etc. (broader application range).
Internet Dependency	Not necessarily dependent on the Internet; often operates in isolated environments.	Strongly relies on Internet connectivity for data sharing and remote control.

IoT History

- While the Carnegie mellon University's vending machine was installed in 1982, this can't really be called the start of the IoT as a whole.

- The notion of the IoT was created in 1991 and further developed through the 1990s with Reza Raji describing the concept at the IEEE Spectrum in 1994.

- The actual term 'the Internet of Things' was created by Kevin Ashton in 1999, although the time when objects were connected directly to the Internet really began between 2008 and 2009.

- With more than 20 billion connected IoT devices today, experts are expecting this number to grow to 30 billion by 2025, 41 billion by 2027, and 50-75 billions by 2030. These figures include consumer IoT (smart home, wearables), industrial IoT (manufacturing, energy, logistics), smart cities, automotive, healthcare, and agriculture sectors.

IoT Characteristics

- **Connectivity** –

  Connectivity is an important requirement of the IoT infrastructure. Things of IoT should be connected to the IoT infrastructure. Anyone, anywhere, anytime can connect, this should be guaranteed at all times.

- **Intelligence and Identity**

  The extraction of knowledge from the generated data is very important.

- **Scalability**

  Scalability in IoT means the system can grow — adding more devices and handling more data — without losing speed or performance. It’s done by expanding hardware or using extra software layers to manage the huge data IoT generates.

- **Dynamic and Self-Adapting (Complexity)**

  IoT devices should dynamically adapt themselves to the changing contexts and scenarios.

IoT Characteristics

❖ IoT Architecture – Common Ecosystem

IoT architecture connects diverse devices from different manufacturers using various technologies and protocols. Its main role is to ensure **interoperability**, smooth **communication**, and that devices work together **without interfering** with each other — even as the number and types of devices grow.

❖ Self Configuring –

One key feature of IoT is that devices can **automatically set up**, **add themselves to a network**, and **update software** with minimal user effort. This makes IoT systems flexible and easy to expand.

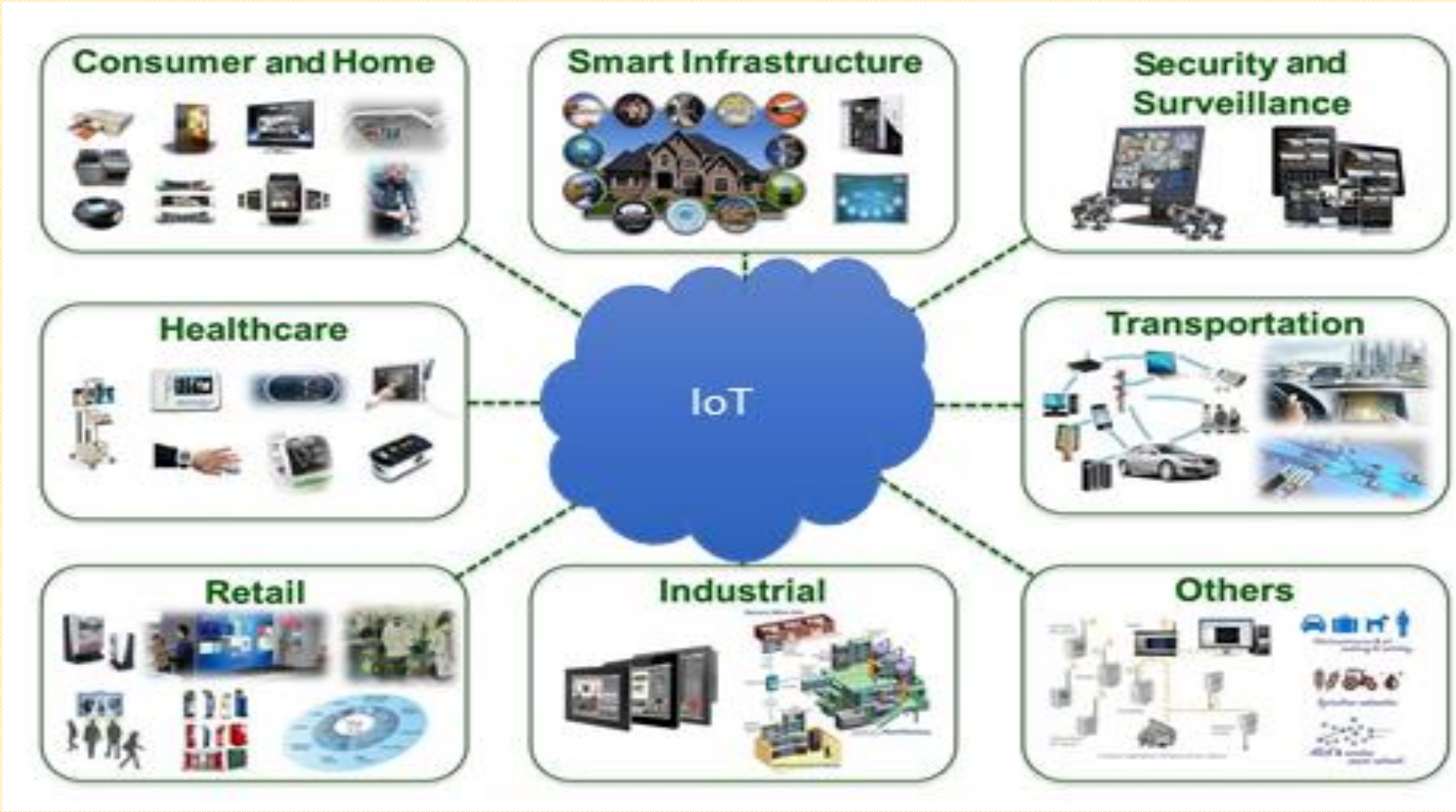
❖ Unique Identity of Things

Every IoT device has a **unique identity** (like a name or ID number). This unique ID helps distinguish each device, allows it to be managed, addressed, and controlled individually in a large network.

IoT Applications

Smart Home	Smart Lightening
	Smart Appliances
	Intrusion Detection
	Smoke/Gas Detection
Smart Cities	Smart Parking
	Smart Road & Traffic
	Structural Health Monitoring
	Emergency Response
Smart Environment	Weather Monitoring
	Air Pollution Monitoring
	Noise Pollution Monitoring
	Forest Fire Detection
Smart Energy	Smart Grids
	Renewable Energy Systems
	Prognostics

Smart Retail	Inventory Management
	Smart Payments
	Smart Vending Machine
Smart Logistics	Route Generation and Scheduling
	Fleet Tracking
	Ship Monitoring
	Remote Vehicle Diagnostics
Smart Agriculture	Smart Irrigation
	Green House Control
	Pest Spraying Control
	Plants Disease Detection
Smart Industry	Machine Diagnosis & Prognosis
	Indoor Air Quality Management
Smart Health & Lifestyle	Health & Fitness Monitoring
	Wearable Sensors


Consumer and Home
Smart Infrastructure
Security and Surveillance
Healthcare
IoT
Transportation
Retail
Industrial
Others

IoT Advantages

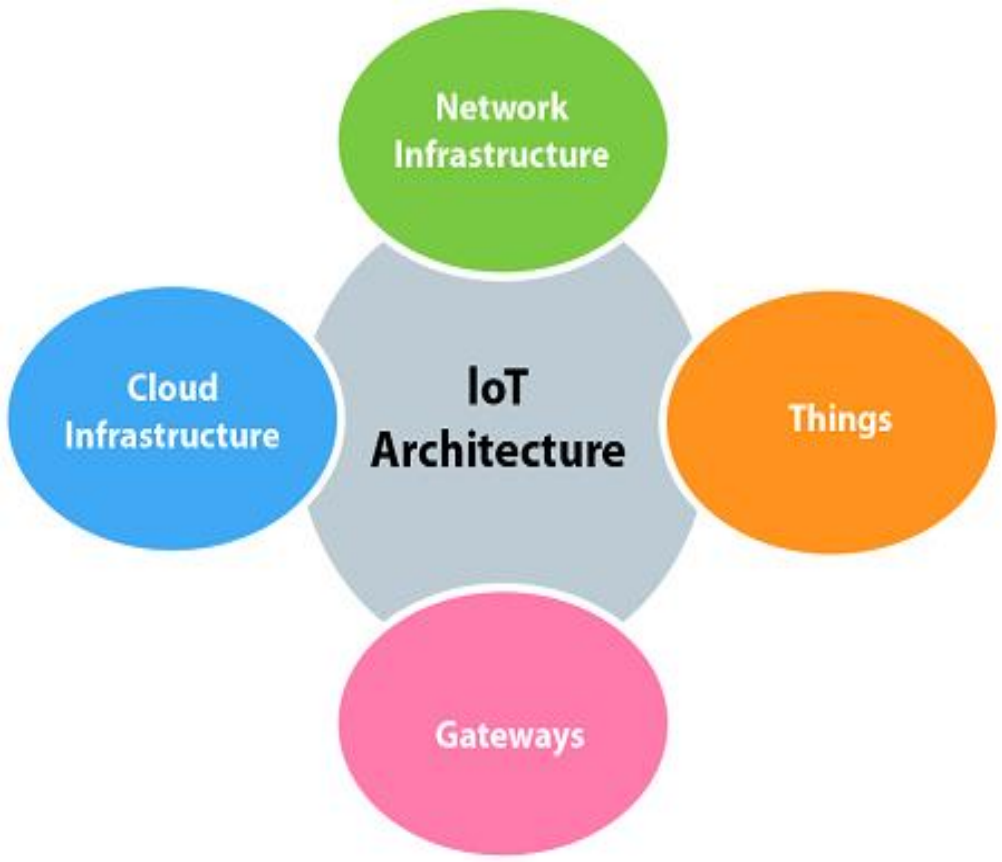
- **Automation and Control**
  - Enables automatic control of devices and processes.
  - Reduces human intervention and errors (e.g., smart thermostats, industrial robots).
- **Efficiency and Productivity**
  - Optimizes energy usage, time, and resources.
  - Boosts performance in sectors like agriculture (smart irrigation), manufacturing (predictive maintenance), and logistics (smart tracking).
- **Data Collection and Insights**
  - Continuously collects real-time data.
  - Helps in making data-driven decisions (e.g., health monitoring, traffic control).
- **Remote Monitoring and Management**
  - Monitor and control devices remotely (e.g., home automation, smart meters).
  - Increases convenience and safety.
- **Improved Quality of Life**
  - Smart homes, wearable health devices, and connected vehicles enhance comfort, safety, and health.
- **Cost Reduction**
  - Preventive maintenance and optimized operations can reduce operational costs in the long run.

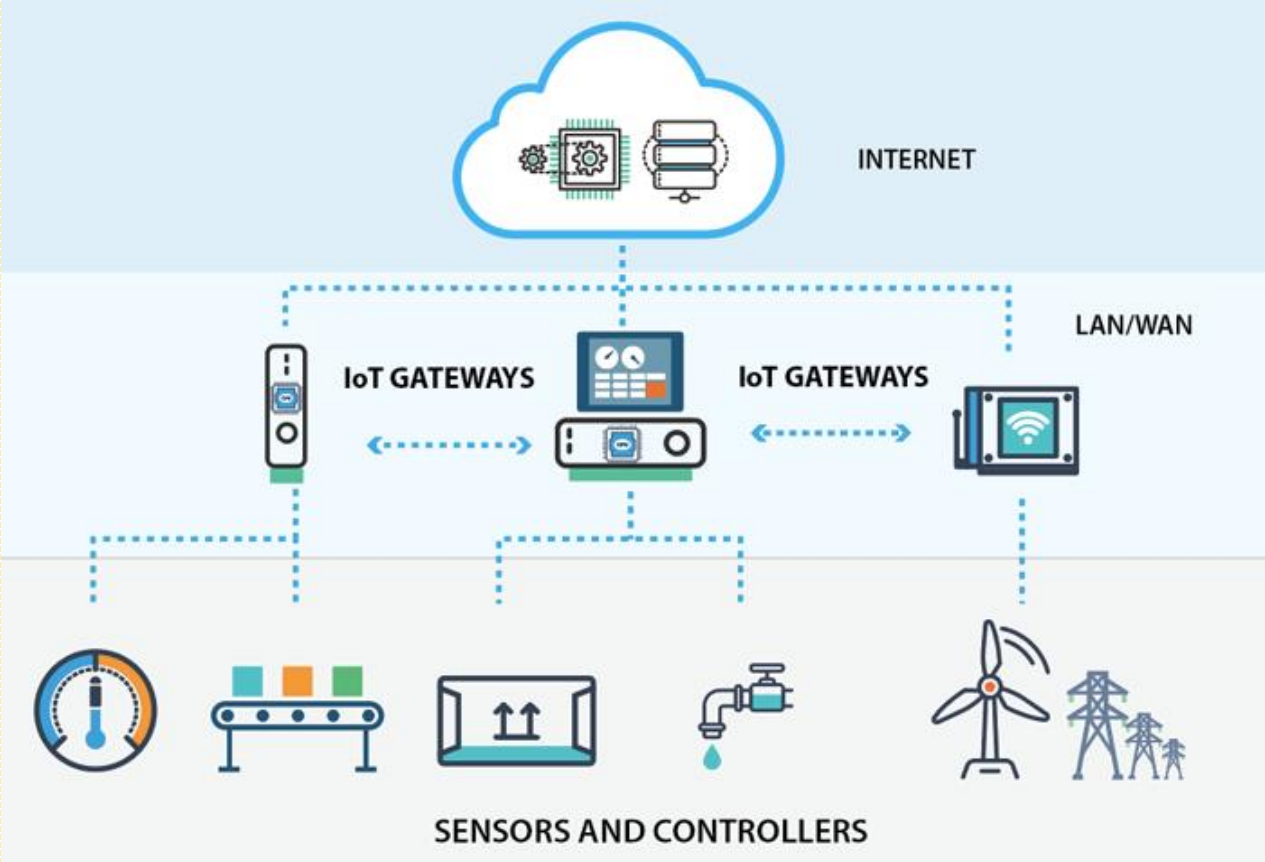
IoT Disadvantages

- **Security and Privacy Issues**
  - Vulnerable to cyberattacks, data breaches, and unauthorized access.
  - Lack of strong encryption in low-cost devices is a concern.
- **Complexity and Interoperability**
  - Integrating different IoT devices and platforms is complex.
  - No universal standard for communication protocols.
- **Dependence on Internet Connectivity**
  - Functionality is affected if the network is slow or down.
  - Offline operations are limited.
- **Data Overload and Management**
  - Massive volume of data requires robust processing, storage, and analytics systems.
- **High Initial Cost**
  - Setup of infrastructure, sensors, gateways, and analytics tools can be expensive.
- **Ethical and Social Concerns**
  - Constant surveillance may lead to ethical issues and reduced privacy.
  - Job displacement due to automation.

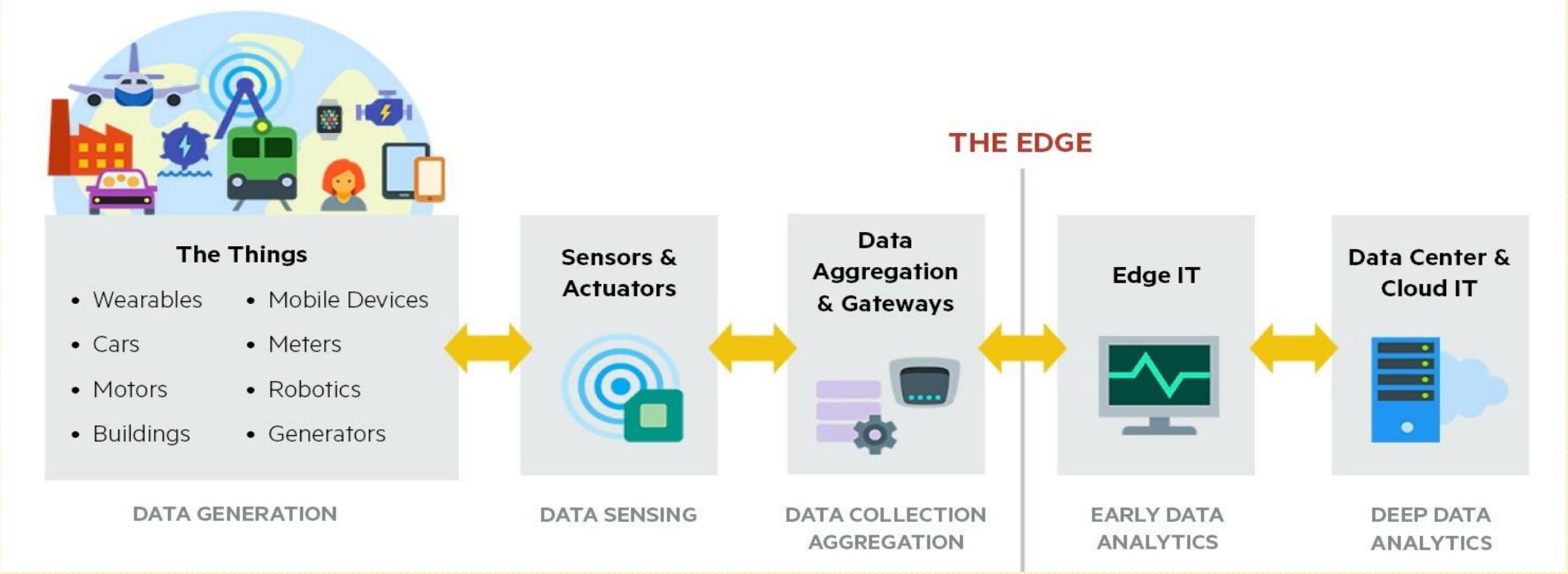
IoT Architecture Components

An IoT architecture is a mix of hardware and software components that interact together to make up a smart cyber-digital system. Interoperating with one another, these components make up a base for an IoT solution to be built upon. Before we dive into the details, let's get things straight: there's no one-size-fits-all approach to designing an IoT architecture. Still, the basic layout stays largely the same no matter the solution.





IoT Architecture



- **Devices:** This stage is about the actual devices in the IoT solutions. These devices could be sensors or actuators in the Perception layer. Those devices will generate data (in the case of sensors) or act on their environment (in the case of actuators). The data produced is converted in a digital form and transmitted to the internet gateway stage. Unless a critical decision must be made, the data is typically sent in a raw state to the next stage due to the limited resources of the devices themselves.

IoT Architecture

- **Internet gateways:** The **internet gateway** collects **raw data** from IoT devices and **pre-processes** it (like filtering or basic formatting) before sending it to the cloud. It can be built into the device or be a separate device that connects sensors to the Internet.

- **Edge or fog computing:** This layer processes **time-critical data** closer to where it's generated — at the **edge of the network** — for **faster response**. It handles **recent data** to detect urgent issues quickly, doing some **pre-processing** to reduce what needs to go to the cloud.

- **Cloud or data center:** In the final stage, all data is **stored, deeply analyzed, and managed** in the cloud or data center. Here, advanced tasks like **machine learning, big data analytics, and dashboards** run to extract valuable insights and support business decisions.

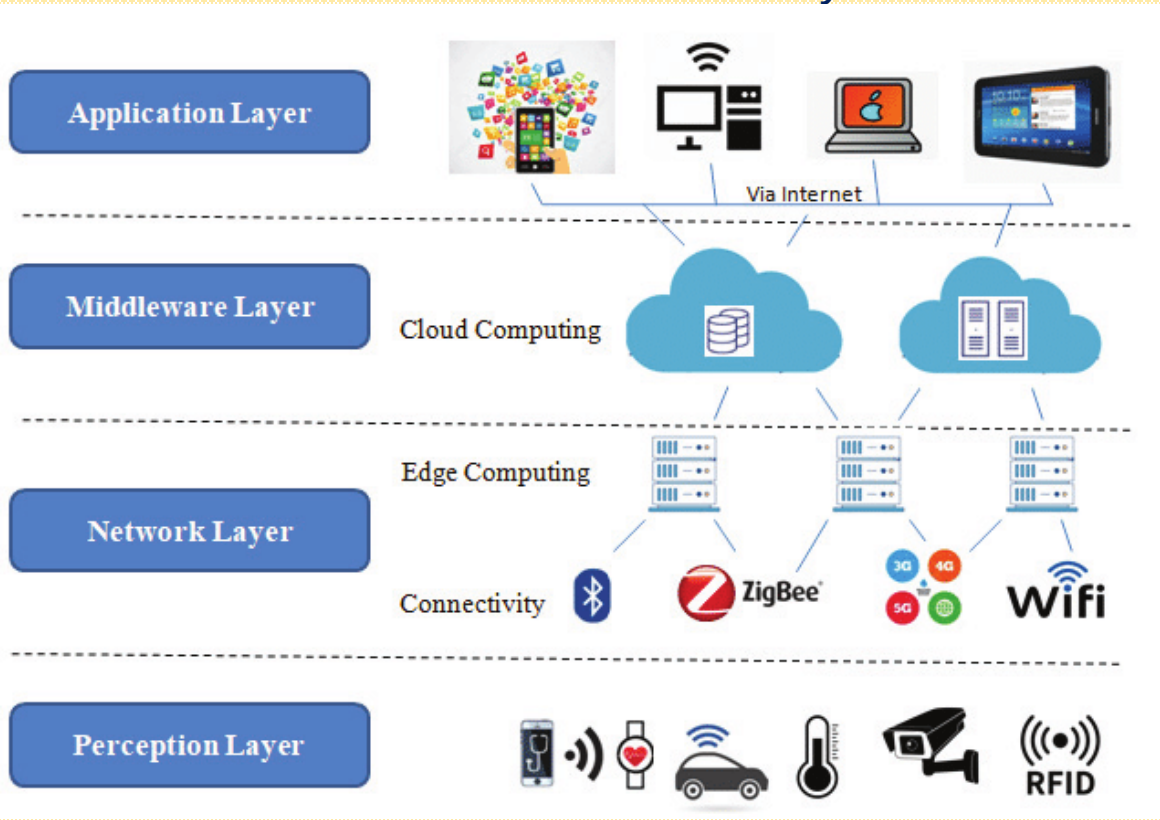
IoT Layered Architecture

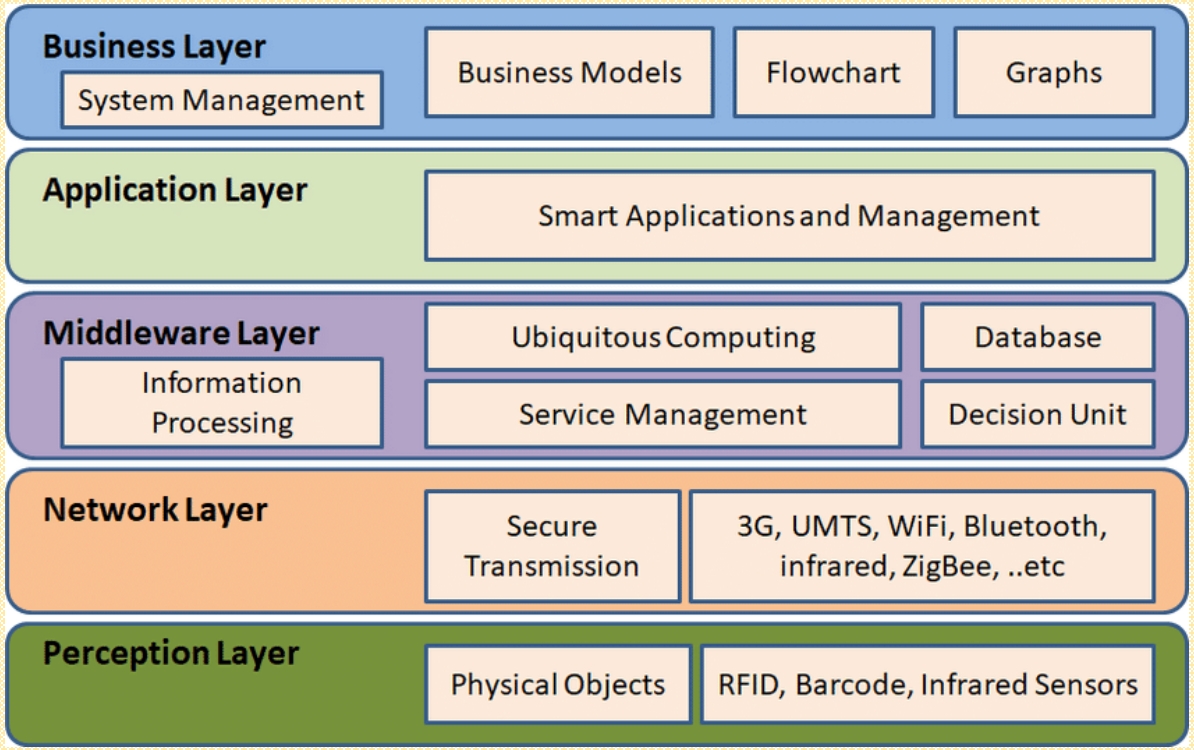
What is an IoT Architecture?

IoT Architecture refers to the structured framework that defines how IoT devices, networks, platforms, and applications interact with each other to collect, process, and use data effectively. It outlines the layers and components required for seamless communication, data management, and intelligent decision-making in an IoT ecosystem.

Typical Layers of IoT Architecture

A standard IoT architecture is commonly divided into **4 or 5 layers**. Here's a **4-layer model** for clarity:





IoT Layered Architecture

1. Connected Objects (Sensing Layer)

- This is the foundation of any IoT system. It includes physical devices such as:
  - **Sensors** (e.g., temperature, motion, humidity)
  - **Actuators** (devices that can act, like turning off a valve or light)
- **Function**:

  Sensors collect real-world data.

  Actuators respond to commands and interact with the physical environment.
- **Example**: A soil moisture sensor detects dryness and an actuator turns on the irrigation pump.

IoT Layered Architecture

2. Internet Gateway (Network Layer)

- Acts as a **bridge** between connected objects and processing systems.
- **Components**:
  - **Data Acquisition System (DAS)**: Collects and digitizes sensor data.
  - **Internet Gateway**: Routes data to the network for further processing.
- **Function**:
  - Aggregates data from multiple devices.
  - Converts analog signals into digital data.
  - Transmits data over local or wide-area networks.
- **Example**: A gateway collects data from sensors and transmits it over Wi-Fi or LTE to processing units.

IoT Layered Architecture

3. Edge IT Systems (Edge Computing Layer)

- **Description**: Performs **local processing** and **pre-analysis** of the data before it reaches the cloud.
- **Technologies**:
  - **Edge servers or local processors**
  - **Machine learning** for real-time decisions
  - **Visualization tools** for user-friendly output
- **Function**:
  - Reduces network load by filtering, summarizing, or acting on data locally.
  - Provides faster response times for critical applications.
- **Example**: A machine learning model on an edge device detects equipment failure and alerts the operator instantly.

IoT Layered Architecture

4. Data Centers and Cloud Storage (Application/Business Layer)

- **Description**: The **final stage** where data is stored, deeply analyzed, and integrated with other systems.
- **Technologies**:
  - **Cloud computing platforms** (e.g., AWS, Azure, Google Cloud)
  - **Big data analytics tools**
  - **Data warehousing**
- **Function**:
  - Long-term storage of sensor data.
  - In-depth analysis and integration with business intelligence systems.
  - Supports dashboards, reporting tools, and enterprise decision-making.
- **Example**: Cloud-based analytics combine sensor data with weather data to predict crop yield in smart agriculture.

IoT Designing

An IoT application structure have basically four design paradigms.

1. **Physical Design**
   - IoT Device
   - IoT Layer Architectures (Protocols)

2. **Logical Design**
   - IoT Functional Blocks
   - IoT Communication Models
   - IoT Communication APIs

3. **IoT Enabling Technologies**
   - Wireless Sensor Networks
   - Cloud Computing
   - Big Data Analytics
   - Communication Protocols
   - Embedded System
   - Artificial Intelligence & Machine Learning

IoT Designing

4. IoT Levels

- IoT Level 1
- IoT Level 2
- IoT Level 3
- IoT Level 4
- IoT Level 5
- IoT Level 6

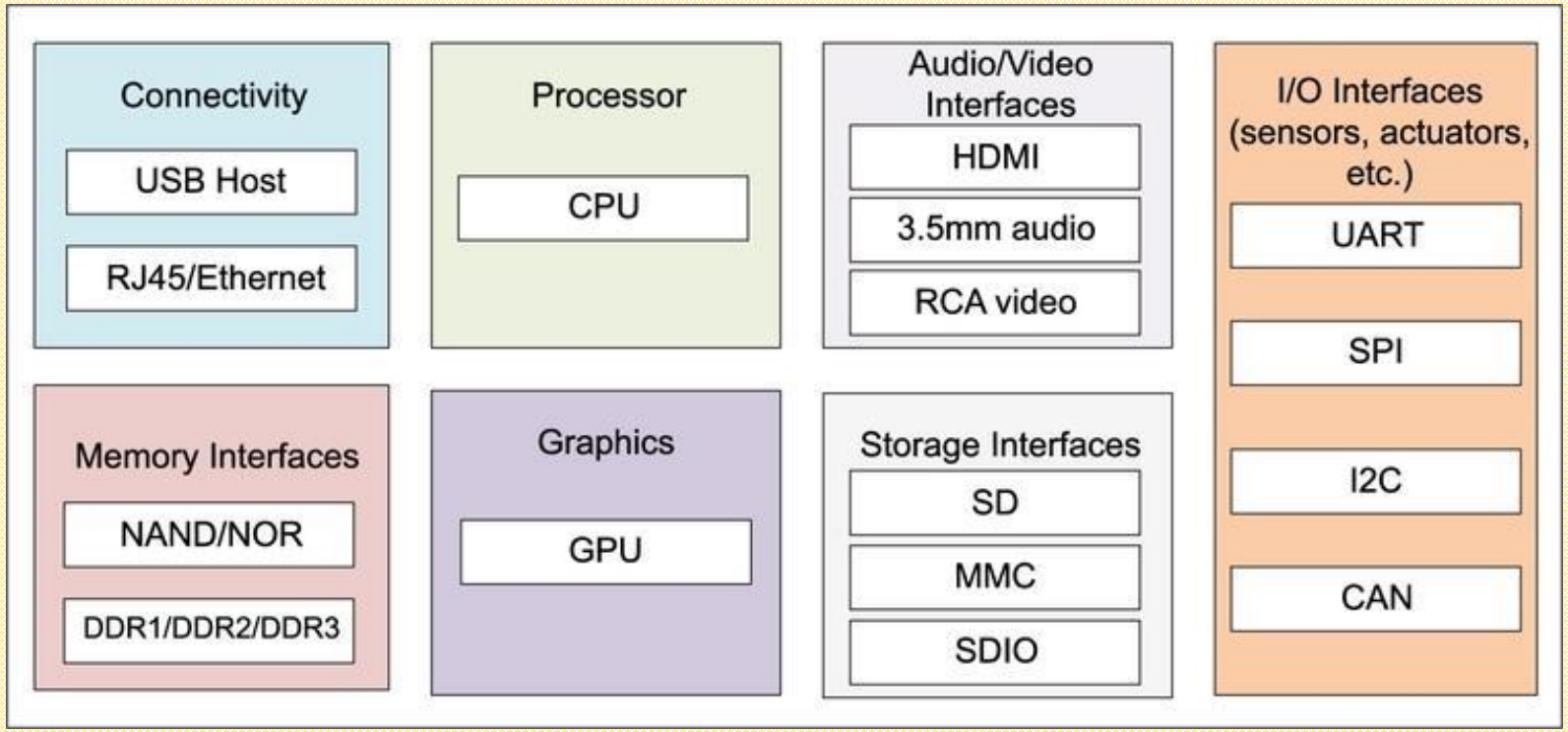
IoT Physical Design

IoT Things:

The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.

An IoT device may consist of several interfaces for communication to other devices both wired and wireless. These includes

(i) I/O interfaces for sensors,
(ii) Interfaces for internet connectivity
(iii) Memory and storage interfaces
(iv) Audio/Video interfaces.



IoT Physical Design

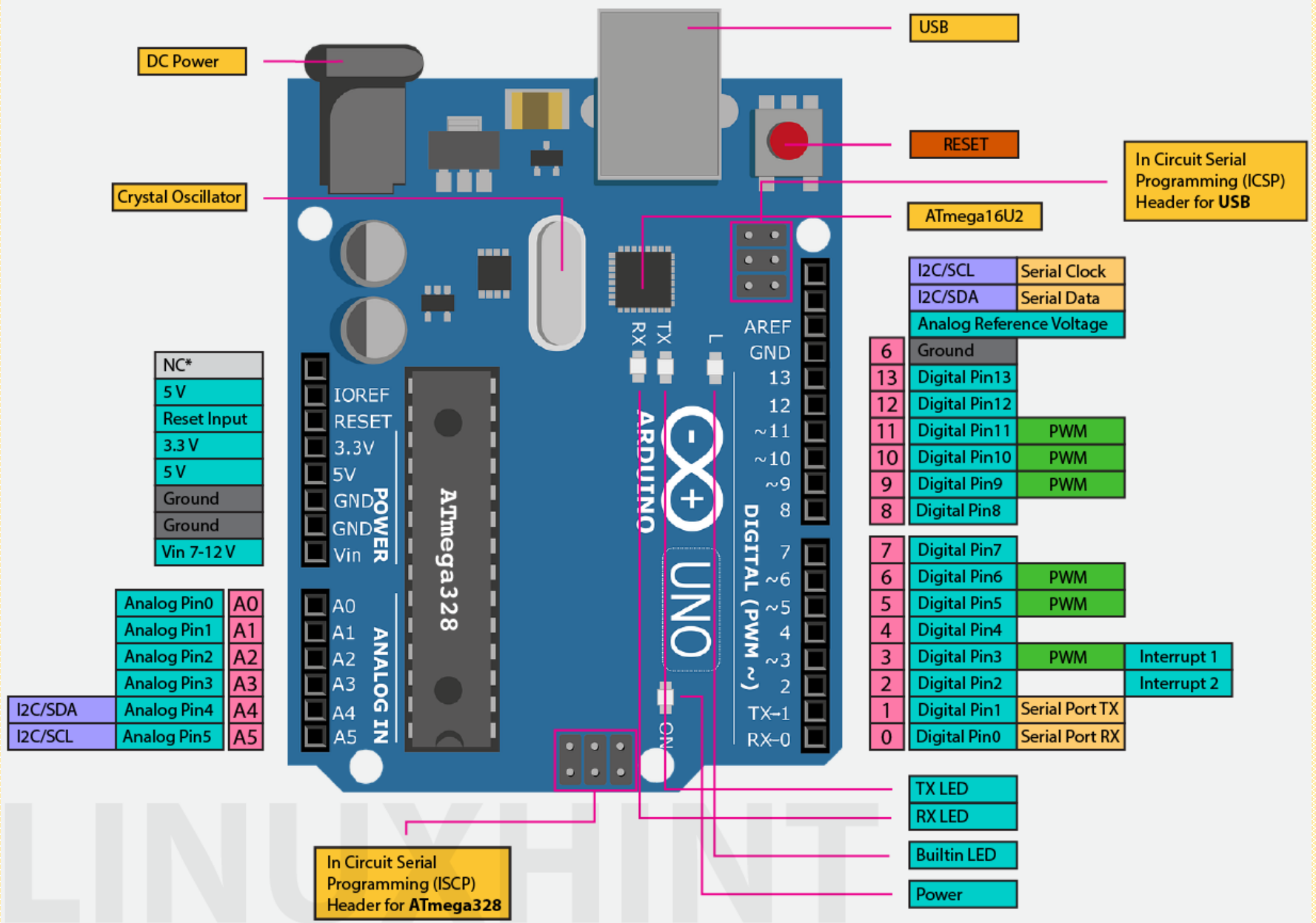
UART/RS232 - UART is usually an individual (or part of an) integrated circuit (IC) used for **serial communications over a computer or peripheral device serial port**. One or more UART peripherals are commonly integrated in microcontroller chips. Specialized UARTs are used for automobiles, smart cards and SIMs.

SPI - Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others.

I²C—Shorthand for inter-integrated circuit or I squared C, this is a low-speed serial bus that is used for sending data from one chip to another on the same PC board or over short cables between two pieces of equipment. I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

CAN Controller Area Network Interface Bus (CAN)—This is controller area network serial bus in an interface used to link micros together in small networks. It is used in automotive and industrial applications. The usual cable is twisted pair and a ground. Speeds range from 10 kbps to 1 Mbps with 20 kbps being typical. The data is sent in specifically formatted frames as determined by a standard protocol. It uses start and stop bits like the RS-232 and some similar control codes defined in ASCII. It also includes error detection and correction capability, so it is very reliable.

IoT Physical Design



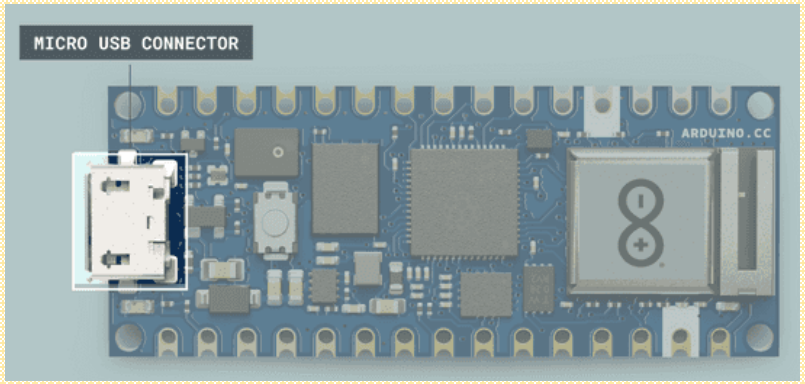
IoT Physical Design

IoT Physical Design

Powering Alternatives

Arduino boards have **five** options in which they can be powered:

1. Powering via USB connector
2. Powering via the onboard barrel jack connector (if available on the board)
3. Powering via the onboard battery connector (if available on the board)
4. Powering via the VIN (Voltage In) pin
5. Powering via the 3V3/5V pin*



*Powering your board via the 3V3/5V pins is not recommended, as it can damage your board's voltage regulator.

Powering via USB connector

- The USB connector provides a regulated 5V line to power the board's electronics. However, **5V from the USB connector can also power external components through the 5V pin** that can be found in Arduino boards.
- Arduino boards that run at 5V use the USB-regulated 5V line directly, boards that run at 3V3 regulate the 5V line from the USB connector to 3V3 using their onboard voltage regulator. Output current rating from the 5V pin will vary, depending on the 5V power source.
- Current from USB ports of computers is usually limited to 500mA.

IoT Physical Design

Powering via the onboard barrel jack connector (if available on the board)

The voltage line from the barrel jack connector is regulated in Arduino boards using their onboard voltage regulator; usually, it is first regulated to 5V and then regulated again to 3V3 in most Arduino boards. The **recommended voltage and current ratings for external regulated DC power supplies** connected to the barrel jack connector are summarized in the table below:

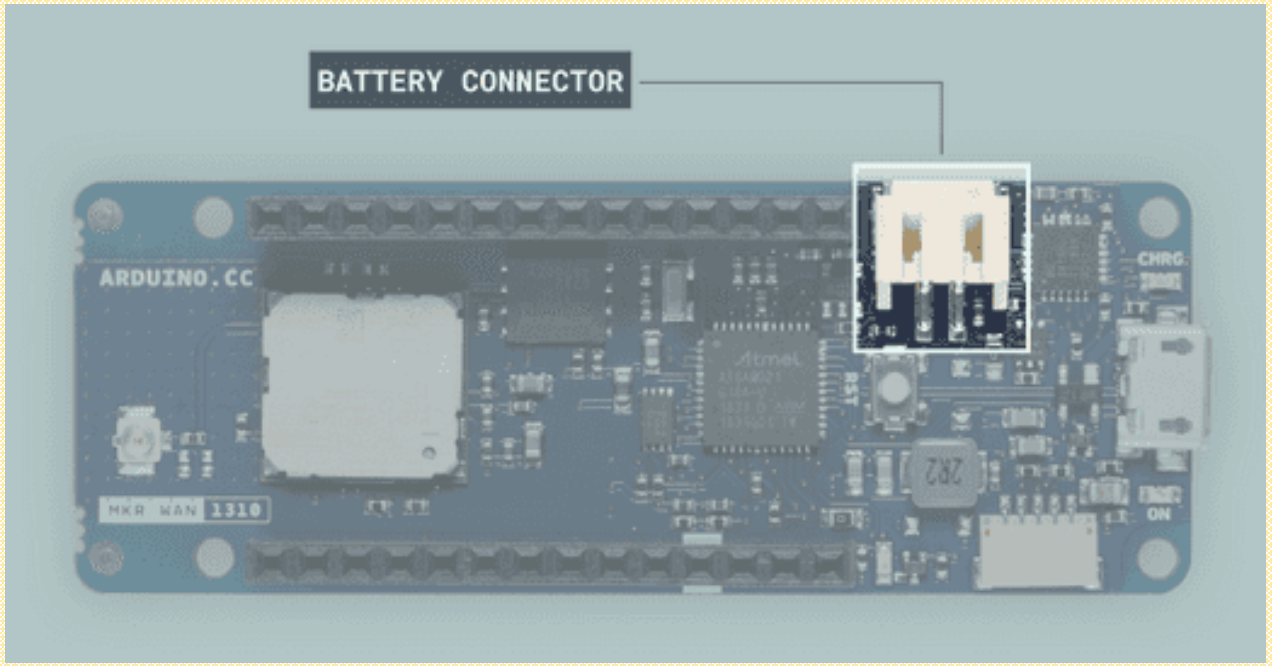
Board	External Power Supply Voltage (V)	External Power Supply Current (A)
Arduino UNO Rev3	7-12	1
Arduino UNO WiFi Rev2	7-12	1.5
Arduino Leonardo	7-12	1
Arduino Mega 2560 Rev3	7-12	1
Arduino Due	7-12	1.5
Arduino Zero	5-18	1

IoT Physical Design

Powering via the onboard battery connector (if available on the board)

Some Arduino boards have an **onboard battery connector** to connect a battery to the board and use it as its primary or secondary power supply. The Arduino boards with an onboard battery connector are the following:

- Arduino Portenta H7
- Arduino Nicla Sense ME
- Arduino Nicla Vision
- Arduino MKR NB 1500
- Arduino MKR Vidor 4000
- Arduino MKR WiFi 1010
- Arduino MKR ZERO
- Arduino MKR WAN 1310
- Arduino MKR GSM 1400



Battery connector of the Arduino MKR WAN 1310 board

Pro family boards use a 3-pin, 1.2mm SMD ACH battery connector; MKR family boards use a 2-pin, 2mm SMD PH battery connector.

IoT Physical Design

Powering via the VIN (Voltage In) pin

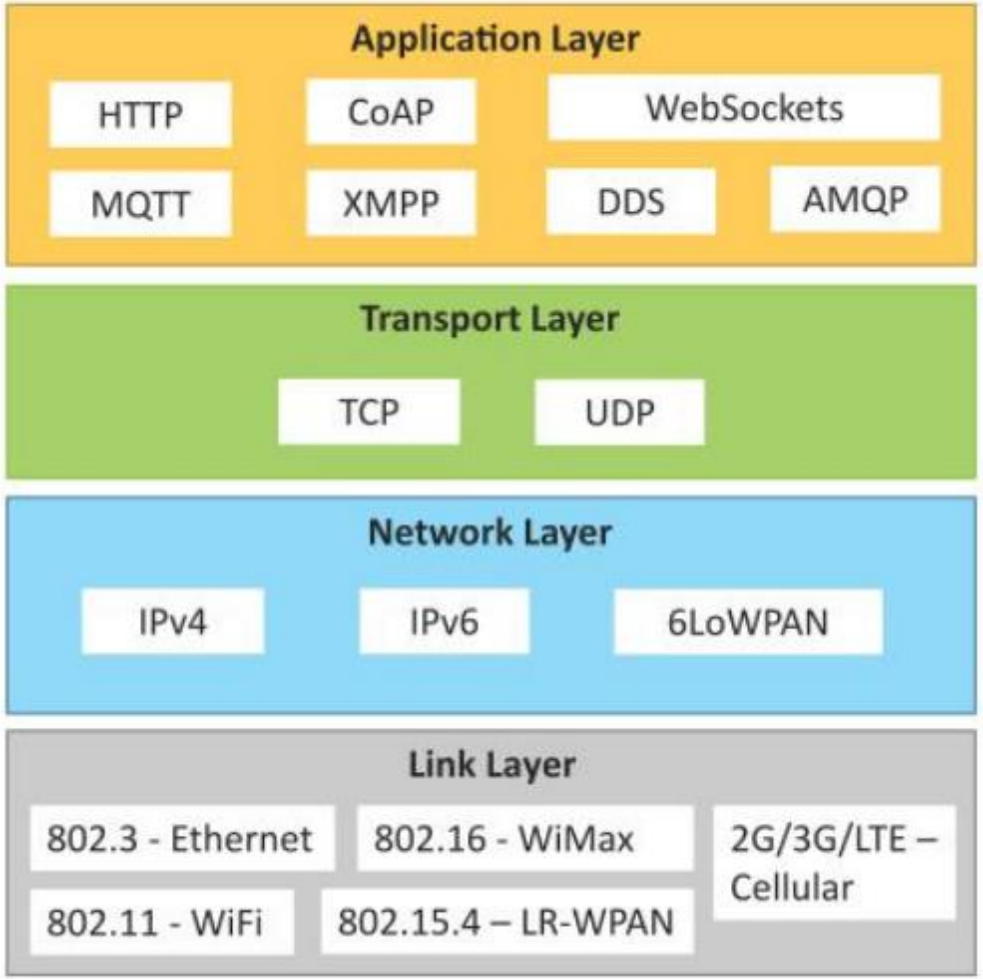
- The VIN pin in Arduino boards is a power pin with a dual function.
- This pin can work as a **voltage input for regulated external power supplies** that do not use a barrel jack connector.
- This pin can also work as a **voltage output when an external power supply is connected to the barrel jack connector** present in some Arduino boards.
- An important consideration is that the VIN pin is connected directly to the input pin of the onboard voltage regulator on Arduino boards.
- Since the VIN pin is directly connected to the voltage regulator, the **VIN pin does not have reverse polarity protection**.
- Use the VIN pin carefully to avoid damaging your Arduino board since it does not have reverse polarity protection.
- The **minimum and maximum voltages** that can be applied to the VIN pin are determined by the onboard voltage regulator on Arduino boards, varying from board to board. Those voltages are summarized in the table below:

Board	VIN Voltage (V)
UNO Mini	5-18
UNO Rev3	7-12
UNO WiFi Rev2	7-12
UNO Rev3 SMD	7-12
Leonardo	7-12
Mega 2560 Rev3	7-12
Due	7-12
Micro	7-12
Zero	5-18
Portenta H7	5
Nicla Sense ME	5
Nano RP2040 Connect	5-18
MKR NB 1500	5-7
MKR GSM 1400	5-7
MKR Vidor 4000	5-7
MKR WiFi 1010	5-7
MKR Zero	5-5.5
MKR1000 WIFI	5-5.5
MKR WAN 1300	5-5.5
MKR WAN 1310	5-7
Nano	7-12
Nano Every	7-18
Nano 33 IoT	5-18
Nano 33 BLE	5-18
Nano 33 BLE Sense	5-18

IoT Physical Design

The **IoT Protocol Layer Model** organizes various communication protocols used in IoT into **layered architecture**, similar to the OSI model. Each layer serves a specific role in **enabling communication, processing, and service delivery** across the IoT ecosystem.

Provides services to end-users and applications	MQTT, CoAP, HTTP, XMPP, AMQP, DDS
Reliable or lightweight data delivery	TCP, UDP, DTLS (for secure UDP)
Routing, addressing, and internetworking	IPv6, RPL, 6LoWPAN, ICMP
Node-to-node data transfer and error handling	IEEE 802.15.4, IEEE 802.11 (Wi-Fi), Ethernet, MAC, LPWAN
Data sensing, physical signal transmission	RFID, Zigbee, Bluetooth, NFC, Sensors, Actuators



IoT Physical Design

Protocols:

- **802.3-Ethernet:** IEEE802.3 is collection of wired Ethernet standards for the link layer. E.g.:
  - 802.3 uses co-axial cable;
  - 802.3i uses copper twisted pair connection;
  - 802.3j uses fiber optic connection;
  - 802.3ae uses Ethernet over fiber.

- **802.11-WiFi:** IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. E. g.
  - 802.11a operates in 5GHz band,
  - 802.11b and 802.11g operates in 2.4GHz band,
  - 802.11n operates in 2.4/5GHz band,
  - 802.11ac operates in 5GHz band,
  - 802.11ad operates in 60Ghz band.

- **802.16 - WiMAX:** IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. Fixed WiMAX provide data rates from 1.5 Mb/s to 1Gb/s. WiMax operates on frequency band 2-11 GHz and covers up to 50 km area. While mobile WiMax operates on 2.3 GHz, 2.5 GHz, 3.5 GHz and covers between 5-15 Kms area.

IoT Physical Design

- **802.15.4-LR-WPAN:**
  - **LR-WPAN** stands for **Low-Rate Wireless Personal Area Network**.
  - It is a wireless network designed for **short-range, low-data-rate, low-power, and low-cost communication** between devices, especially **sensors and embedded systems** in IoT applications.
  - LR-WPAN is standardized under **EEE 802.15.4.**
  - This standard defines the **physical (PHY)** and **MAC (Medium Access Control)** layers for low-rate wireless networks.
  - Provides data rate from 40kb/s to 250kb/s.
  - Typically ranges between 10-100 mtrs.

- **2G/3G/4G-Mobile Communication:** These are the different generations of mobile communications standards:
  - 2G including GSM and CDMA,
  - 3G including UMTS and CDMA3000,
  - 4G including LTE,
  - Data rates from 9.6kb/s(2G) up to100Mb/s(4G).

Aspects	ZigBee	WiFi	Bluetooth
standard	802.15.4	802.11a,b,g	802.15.1
application	automation, control	Web, e-mail, video	replacement of cableing
data rate	50 - 60 kbyte	> 1 Mbyte	> 250 kbyte
battery lifetime	> 1000	1 -5	1 -7
network size	65535	32	7
bandwith (kb/s)	20 - 250	11000	720
transmission distance	100+ m	100 m	10 m
advantage	reliability, performance, cost	data rate, flexibility	cost, comfortable

IoT Physical Design

LoRa

- LoRa, which is an abbreviation of “long range”, from RF technology for an LPWAN.
- LoRa is one of the most prominent wireless technologies in the low-power wide-area network (LPWAN) family.
- LoRa is a patented energy-efficient wireless communication protocol that achieves very low-power and very long-range transmissions, of over 10 km in line-of-sight, trading-off data rate, and time-on-air.
- LoRa’s duty cycle is limited by regional regulations because it operates using unlicensed sub-GHz radio bands, mostly on the 433, 868, and 915 MHz frequency bands.
- The data transmission rate supported by LoRa varies from 300 bps to 50 kbps, depending on spreading factor (SF) and channel bandwidth settings.
- Taking into account this and the low-transmission bandwidth, LoRa is naturally most suitable for applications where transmissions are sparse in time and payloads are relatively small.

IoT Physical Design

Network/Internet Layer:

Responsible for sending IP datagrams from source network to destination network. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- **IPv4:** Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32-bit address. Allows total of 2^{32}addresses. These addresses got exhausted in the year of 2011. IPv4 has been succeeded by IPv6. IP protocols establish connection over packet networks, but do not guarantee delivery of packets.

- **IPv6:**It is newest version of IPv4. Internet Protocol version6 uses 128-bit address scheme and allows 2^{128} addresses.

- **6LOWPAN**:(IPv6 over Low Power Wireless Personal Area Network) operates in 2.4 GHz frequency range and data transfer 250 kb/s. It brings IP protocol for low power devices having limited processing capability.

Transport Layer:

Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.

Protocols:

- **TCP:** Transmission Control Protocol used by web browsers (along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. TCP also provides the error detection capability so that duplicate packets can be discarded and loss packets are retransmitted. Avoids n/w congestion and congestion collapse. It also ensures flow control to adjust the mismatch of transmission and receiving speed of sender and receiver.

- **UDP:** User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery, ordering of messages and duplicate elimination.

Application Layer:

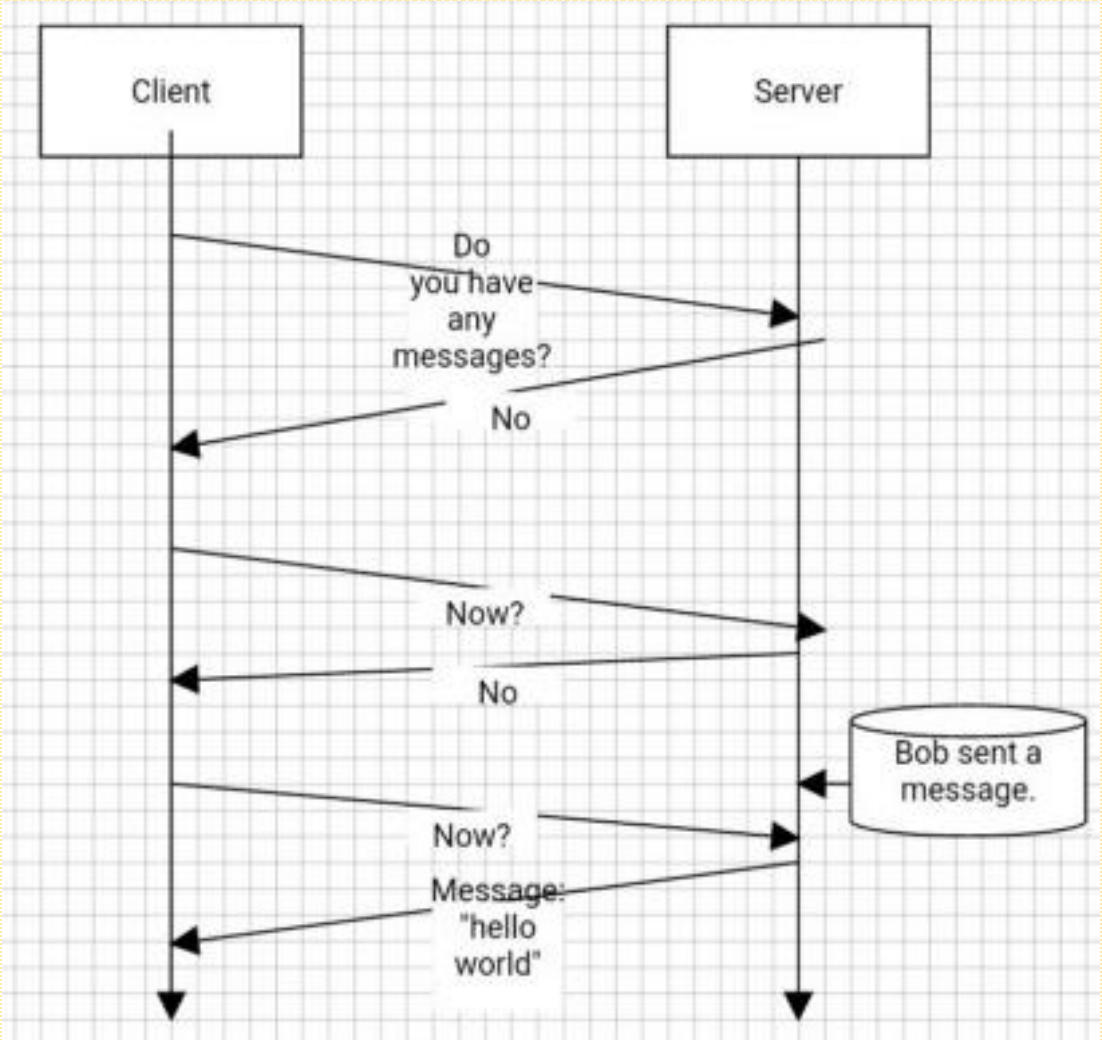
Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

HTTP

- Hyper Text Transfer Protocol that forms foundation of WWW.
- Follow request response model Stateless protocol.
- It was designed for communication between web browsers and web servers, but it can also be used for other purposes.
- HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response.
- HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests.
- Defaults to port **80** (ws://) and **443** (wss://) for secure.
- Applications of HTTP/HTTPs are Web Browsing, RESTful APIs, Web Services, IoT Communication (Limited)

If you are a developer, you probably know what HTTP **(or HTTPS)** is. It is seen every day, in your browser. A request is needed to respond; **you to constantly ask the server if there are new messages in order to receive them.**

You should also know that HTTP allows you to have different types of requests such as post, get or put, each with a different purpose.



Application Layer:

Web Socket:

It allows full duplex communication over a single socket connection. This protocol defines a full duplex communication from the ground up. Web sockets take a step forward in bringing desktop rich functionalities to the web browsers.

The main features of web sockets are as follows −

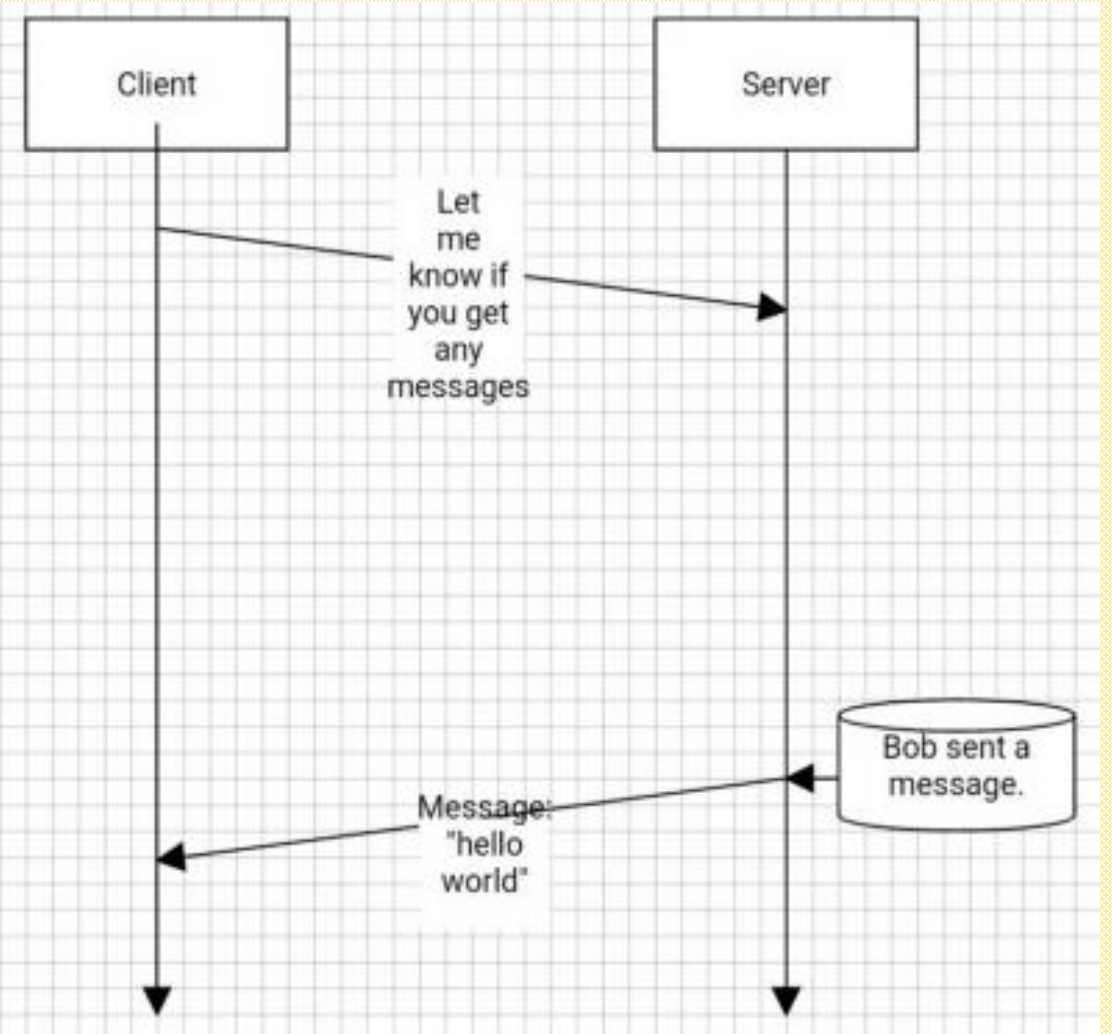
- Real time communication between web servers and clients is possible with the help of this protocol.
- Cross platform standard for real time communication between a client and the server.
- The biggest advantage of Web Socket is it provides a two-way communication (full duplex) over a single TCP connection.
- Both client and server can send/receive messages independently.
- Header Overhead is minimal compared to HTTP; Performance is better than HTTP.
- Defaults to port **80** (ws://) and **443** (wss://) for secure.
- Ideal for real-time applications like chat and gaming.

How WebSocket Works

1. Client sends an **HTTP request** with an Upgrade: websocket header.
2. Server responds with a **"101 Switching Protocols"** response.
3. The connection is upgraded from HTTP to WebSocket.
4. Now, both client and server can **push messages anytime** until the connection is closed.

Web Sockets on the other hand don't need you to send a request in order to respond. They allow bidirectional data flow so you just have to listen for any data.

You can just listen to the server and it will send you a message when it's available.



Web Socket Applications

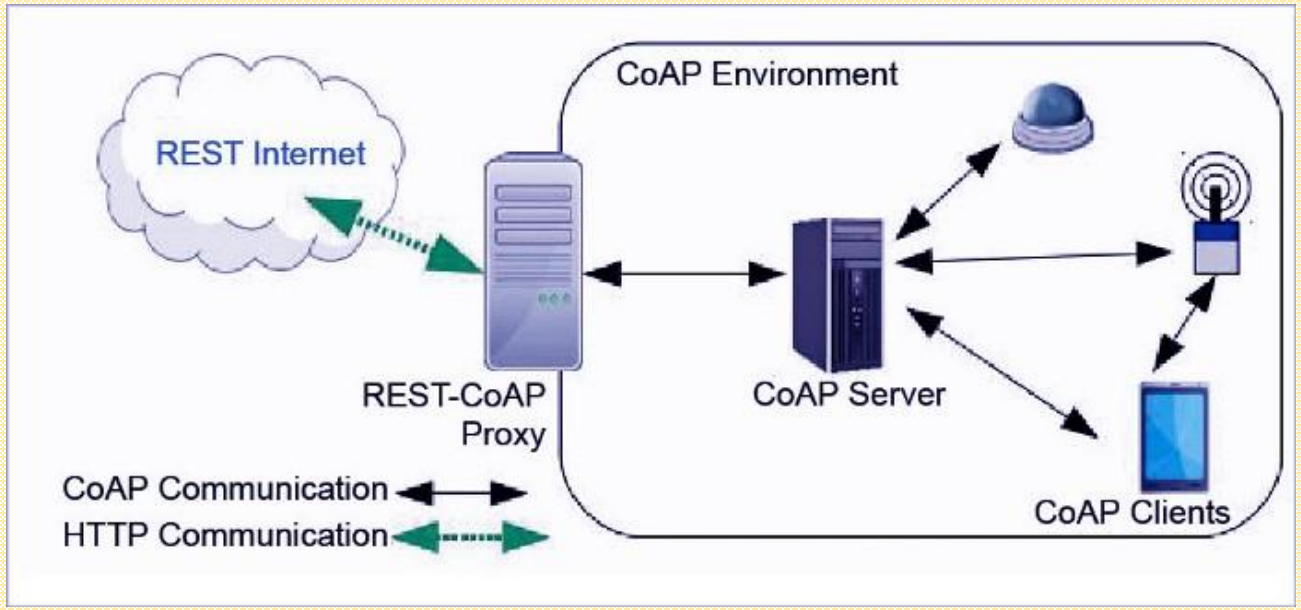
- ❖ Online multiplayer games
- ❖ Live stock market dashboards
- ❖ Real-time GPS tracking
- ❖ Collaborative document editing (e.g., Google Docs)
- ❖ IoT device communication

Application Layer:

CoAP

CoAP (Constrained Application Protocol) is a **lightweight web transfer protocol** designed for use in **constrained devices and networks**, such as **IoT systems**, where resources like power, bandwidth, memory, and processing are limited.

- It's similar to HTTP but optimized for **low-power** and **low-bandwidth** environments.
- Uses client server architecture. Like HTTP, CoAP is a document transfer protocol.
- Unlike HTTP, CoAP is designed for the needs of constrained devices.
- CoAP packets are much smaller than HTTP TCP flows.
- CoAP runs over UDP, not TCP.
- CoAP allows UDP broadcast and multicast to be used for addressing.
- CoAP follows a client/server model. Clients make requests to servers, servers send back responses.
- Clients may GET, PUT, POST and DELETE resources.



- CoAP is designed to interoperate with HTTP and the RESTful web at large through simple proxies. Because CoAP is datagram based, it may be used on top of SMS and other packet based communications protocols.

CoAP vs HTTP

Aspect	CoAP	HTTP
Protocol	UDP	TCP
Payload	Small, binary	Large, text-based
Use Case	IoT, constrained networks	General web applications
Header Size	Very small	Large
Reliability	Optional (confirmable msgs)	Reliable (TCP-based)

Example Application: Smart Home IoT System

Scenario: A smart temperature sensor reporting to a central home controller.

- The controller sends a GET request via CoAP to the temperature sensor.
- The sensor replies with the current temperature value.
- The controller can also **observe** the sensor, receiving updates whenever the temperature changes.

CoAP URI Example:

coap://192.168.0.10/sensors/temperature

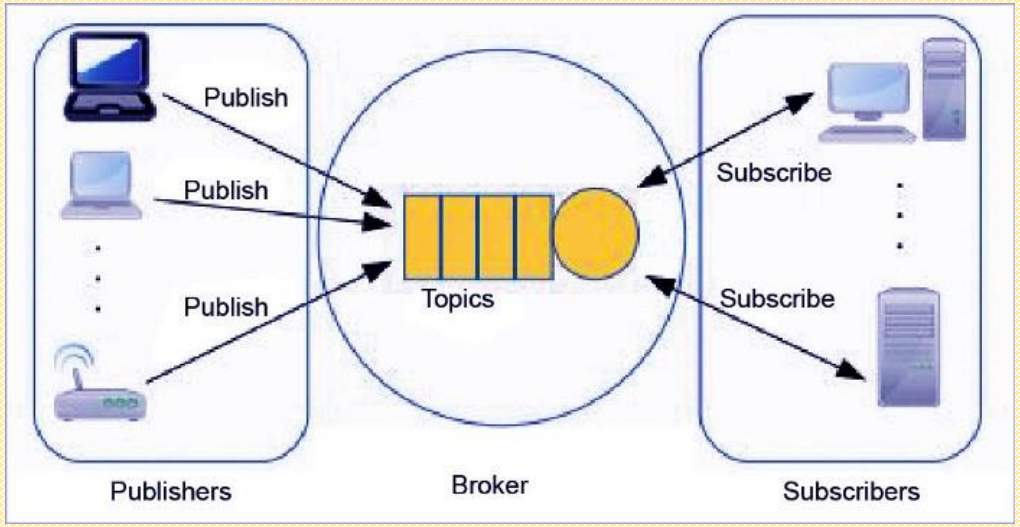
Why Use CoAP in IoT?

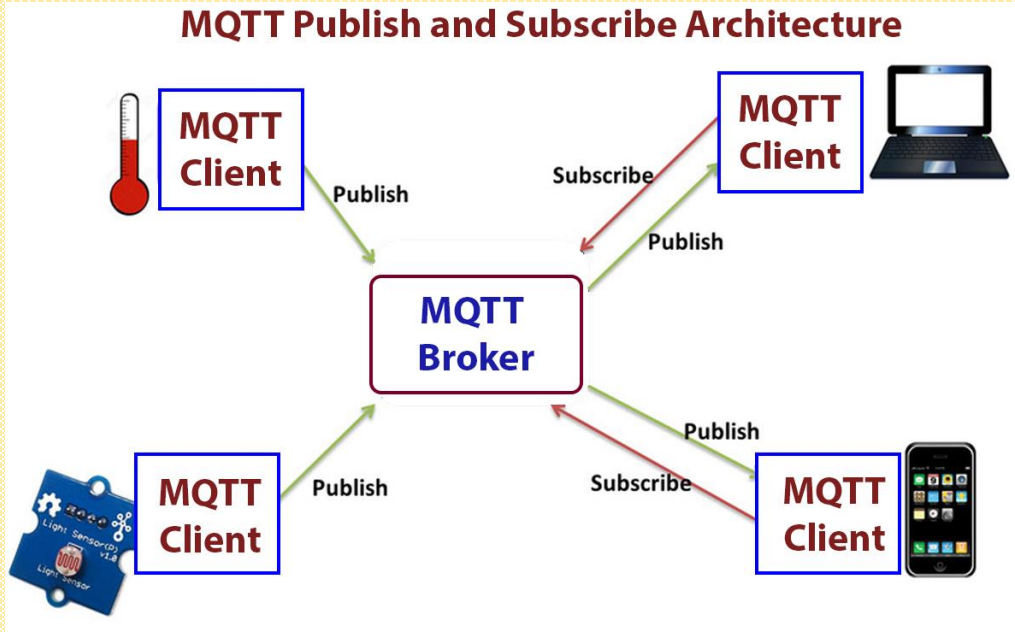
- Ideal for **low-power devices** (battery-operated).
- Suited for **lossy networks** (like LPWAN or 6LoWPAN).
- Enables **efficient communication** with microcontrollers and embedded systems.

Application Layer:

MQTT

- MQTT is a lightweight, publish/subscribe messaging protocol designed for low-bandwidth, high-latency, or unreliable networks — making it ideal for IoT (Internet of Things) applications.
- Uses client server architecture.
- Minimal overhead — ideal for constrained devices.
- Ensures reliable transmission over networks.
- Supports QoS Levels and ensure message delivery as needed (0- At most once, 1-At least once, 2-Exactly once).
- MQTT is a publish/subscribe messaging protocol designed for lightweight M2M communications.
- It was originally developed by IBM and is now an open standard.
- MQTT has a client/server model, where every sensor is a client and connects to a server, known as a broker, over TCP. MQTT is message oriented.





Application Layer:

MQTT Architecture

1. **Publisher**: Sends messages to a topic.
2. **Subscriber**: Receives messages from topics it subscribed to.
3. **Broker**: Central server that handles message routing between publishers and subscribers.

Example Use Case: Smart Agriculture

Scenario: A soil moisture sensor sends data to an MQTT broker. A farmer's smartphone app subscribes to this topic and gets real-time updates on soil conditions.

- **Publisher**: Soil moisture sensor
- **Broker**: Mosquitto MQTT broker (e.g., running on Raspberry Pi or cloud)
- **Subscriber**: Farmer's mobile app or dashboard

Common MQTT Brokers

Eclipse Mosquitto (Open source), **HiveMQ, EMQX, AWS IoT Core** (MQTT supported), **Google Cloud IoT**

Application Layer:

MQTT vs CoAP

Aspect	MQTT (Message Queuing Telemetry Transport)	CoAP (Constrained Application Protocol)
Protocol Type	Publish/Subscribe messaging protocol	RESTful request/response protocol (like HTTP)
Transport Layer	TCP/IP	UDP/IP
Message Model	Asynchronous, event-driven (via Broker)	Synchronous (client-server), Optional async via Observe
Architecture	Centralized (Broker-based)	Decentralized (peer-to-peer capable)
Security	TLS/SSL	DTLS (Datagram TLS)
Overhead	Medium (lightweight, but uses TCP)	Very Low (binary over UDP)
Reliability	Built-in with QoS (0, 1, 2)	Optional (Confirmable/Non-confirmable msgs)
Resource Usage	More suitable for slightly higher-resource devices	Ideal for very constrained devices
Best Use Cases	Telemetry, real-time messaging, remote monitoring	Resource-constrained IoT (e.g., sensors in agriculture)
Multicast Support	No	Yes (built-in support)

Application Layer:

XMPP

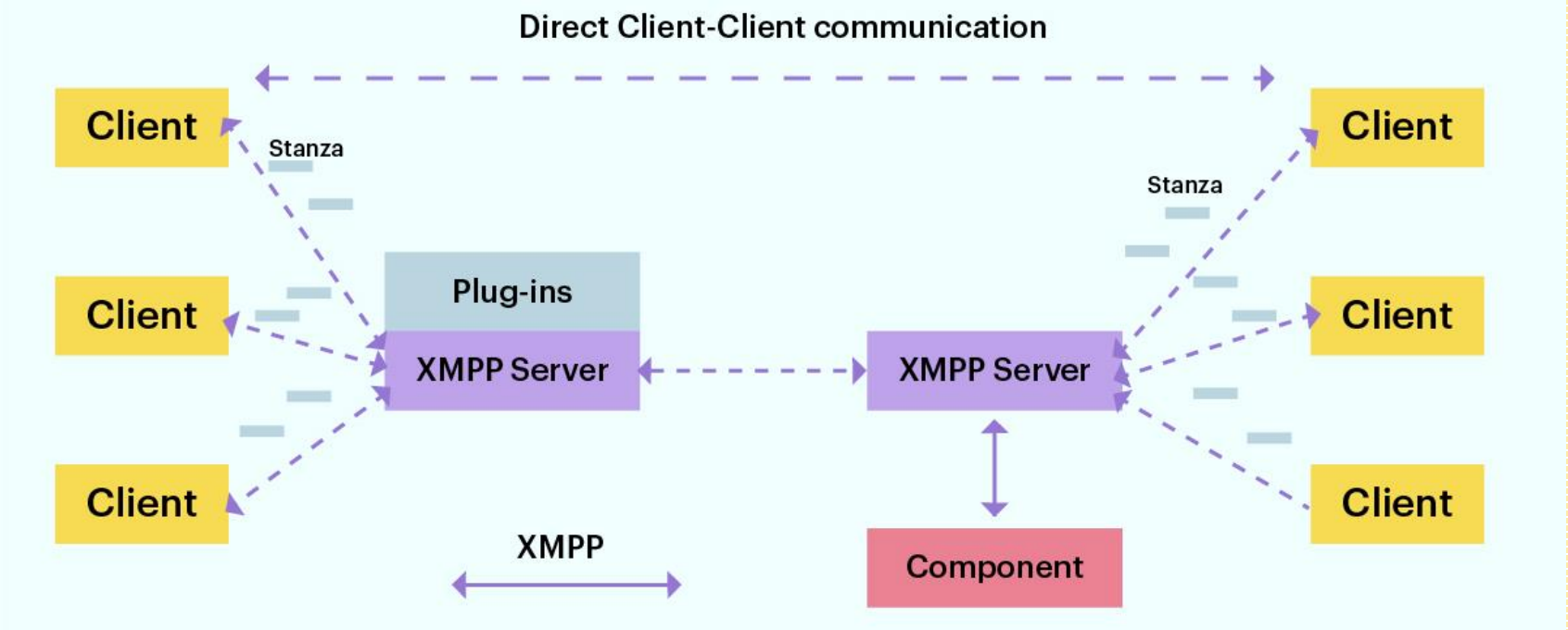
- Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- The XMPP protocols are free, open, public, and easily understandable (Defined by the **IETF** in RFCs like **RFC 6120, 6121, 6122)**
- **It is based on Client–Server** model with support for server-to-server communication.
- It runs over **TCP**, uses **port 5222** (for client) and **5269** (for server).
- New features can be added without breaking the core functionality.
- XMPP applications:
  - Instant Messaging (Google Talk (legacy), WhatsApp (initially), Jabber)
  - IoT Communication (Lightweight messaging for constrained devices)
  - Enterprise Messaging (Private chat servers (e.g., Openfire, ejabberd))
  - Gaming (Real-time player communication

  and remote systems monitoring, web services, lightweight middleware, cloud computing, and much more.

Application Layer:

How XMPP Works (Simplified):

1. Client connects to XMPP server (e.g., user@domain.com)
2. Server authenticates and maintains a session
3. Users exchange messages using XML stanzas (message, presence, iq)
4. Presence updates and subscriptions are handled in real time



Application Layer:

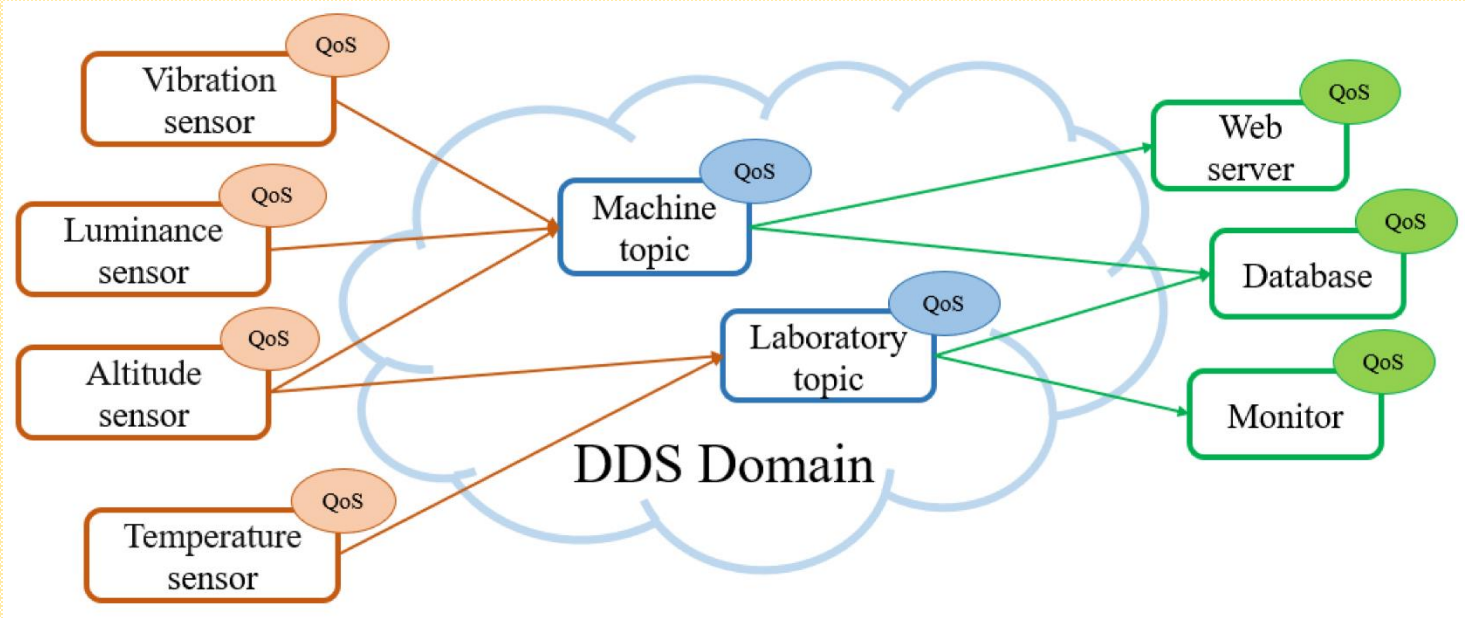
DDS

- **DDS (Data Distribution Service)** is a middleware **communication protocol** designed for **real-time**, **scalable**, and **high-performance** data exchange between devices and applications in distributed systems.
- The protocol uses broker-less architecture in the Internet of Things, unlike MQTT and CoAP protocols.
- It is an IoT protocol developed by Object Management Group for Machine to Machine Communication.
- It uses a publish-subscribe methodology for exchanging data.
- It uses multicasting to bring high-quality QoS to the applications.
- It follows a **publish-subscribe model** and is widely used in systems where **low latency**, **deterministic behavior**, and **quality of service (QoS)** are critical — such as autonomous vehicles, industrial IoT, robotics, aerospace, and defense.

Application Layer:

Key Features of DDS:

- **Real-time Communication:** Offers low-latency data delivery with predictable timing.
- **Publish-Subscribe Model:** Decouples data producers (publishers) and consumers (subscribers).
- **Quality of Service (QoS):** Users can define reliability, durability, latency, deadline, etc.
- **Decentralized Architecture:** No central broker needed; peers discover and communicate directly.
- **Automatic Discovery:** Nodes automatically detect and connect with compatible publishers/subscribers.
- **Scalability:** Supports thousands of nodes/devices and large data volumes.
- **Built-in Security:** Supports authentication, encryption, and access control.



Application Layer:

How DDS Works – Step by Step:

1. **Domain Participants:** Applications join a communication domain by creating a domain participant.

2. Publishers & Subscribers

- Publishers send data; Subscribers receive data.
- Both define the **topic** they deal with (e.g., TemperatureSensorData).

3. Data Writers & Data Readers

- Inside a publisher: **DataWriter** pushes data to a topic.
- Inside a subscriber: **DataReader** receives topic data.

4. Topic-Based Communication: Communication happens over **topics**, defined by name and data type.

5. Discovery Mechanism

- DDS uses **Real-Time Publish-Subscribe (RTPS)** protocol to automatically discover data producers and consumers.

6. Quality of Service (QoS) Policies

- Each communication entity defines its QoS needs (e.g., reliable delivery, minimum delay).
- DDS matches compatible QoS between publishers and subscribers.

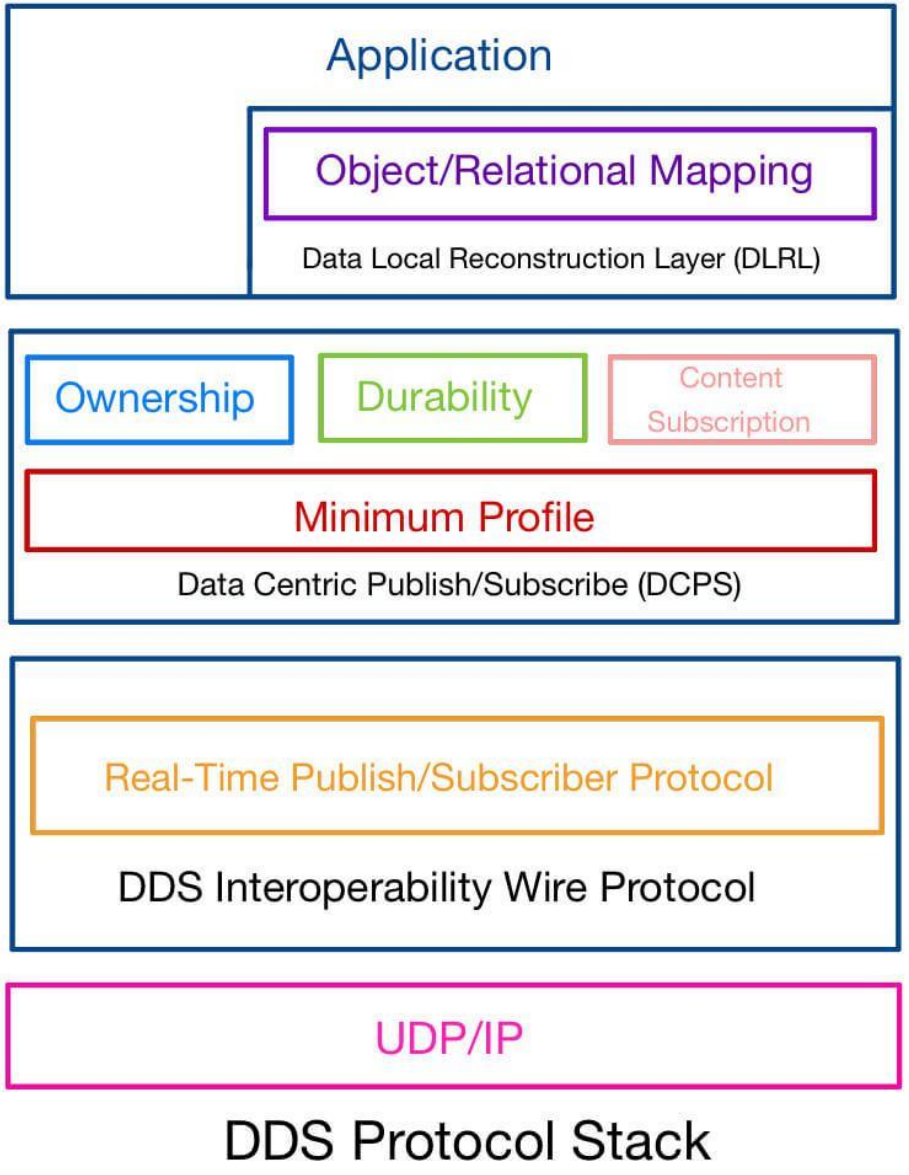
Application Layer:

DDS V1.2 API standard is language, operating system, and hardware architecture independent.

The DDS protocol stack. It consists of two primary layers:

- **DCPS (Data Centric Publish Subscribe) layer:** This layer handles the delivery of information to subscribers. The DCPS is a standard API for data-centric, topic-based, real-time publish/subscribe operations.
- **DLRL (Data Local Reconstruction Layer):** The DLRL provides an interface to DCPS functionalities. It allows for the sharing of distributed data among IoT-enabled devices. The DLRL is a standard API for creating object views from a collection of topics.

DDSI/RTPS V2.1 is a standard wire protocol, ensuring interoperability between different implementations of the DDS standard.



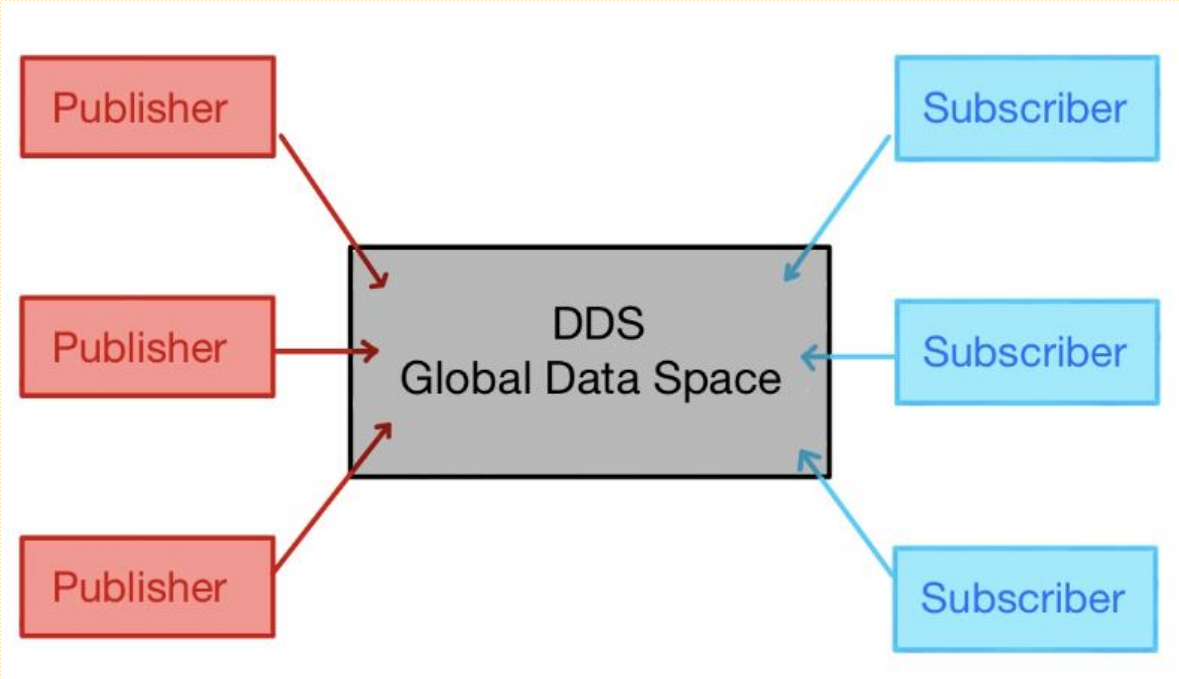
DDS Protocol Stack

Application Layer:

DDS Working

Data Distribution Service (or DDS) is a fully distributed Global Data Space, which makes it fully distributed to avoid introducing a single point of failure or bottleneck.

1. Applications can autonomously and asynchronously read as well as write data in the Global Data Space.
2. Since publishers and subscribers are dynamically discovered, they can join or leave the Global Data Space at any time.
3. Publishers and Subscribers show their intention to produce or consume a specific type of data, such as topics.
4. Subscriptions are matched by taking into account the topics with details like name, data type, etc.
5. All the subscriptions are dynamically matched, and the data flows from the publisher to the subscribers.



Application Layer:

DDS addresses the needs of applications like:

- Aerospace and defence,
- Air-traffic control,
- Autonomous vehicles,
- Medical devices,
- Robotics,
- Power generation,
- Simulation and testing,
- Smart grid management,
- Transportation systems, and other applications that require real-time data exchange.

Application Layer:

AMQP:

- AMQP stands for Advanced Messaging Queuing Protocol. It is a standard for cross platform messaging.
- Open-standard application layer protocol for message-oriented communication.
- Designed for reliable, secure, and interoperable messaging between distributed systems.
- Commonly used in enterprise and cloud applications.
- Supports both point-to-point and publish-subscribe model.
- Enterprise applications are being written in a number of dynamic languages such as Ruby, Perl and Python.

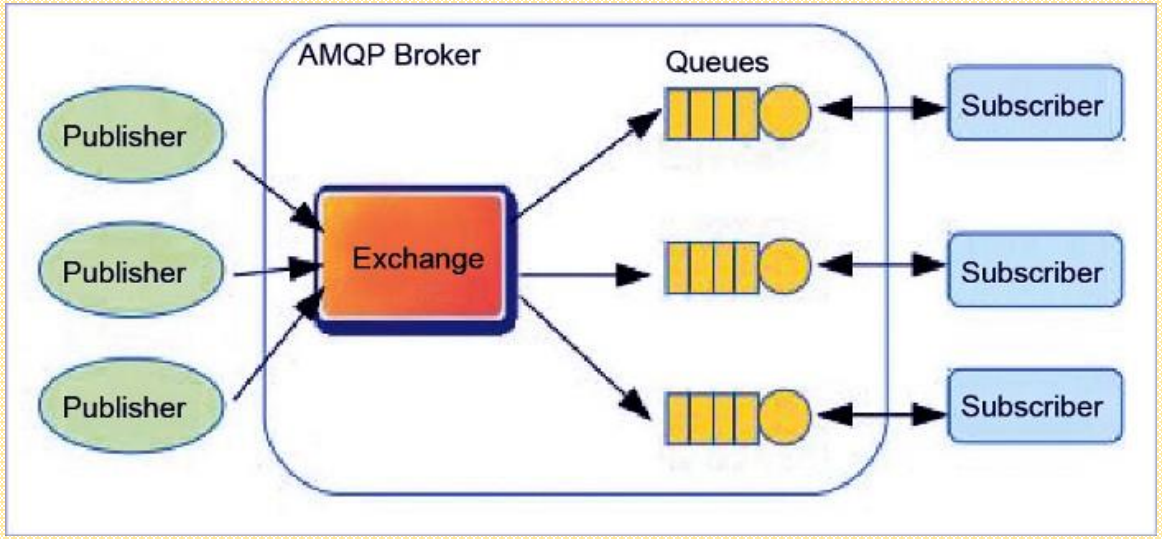
AMQP Key Features:

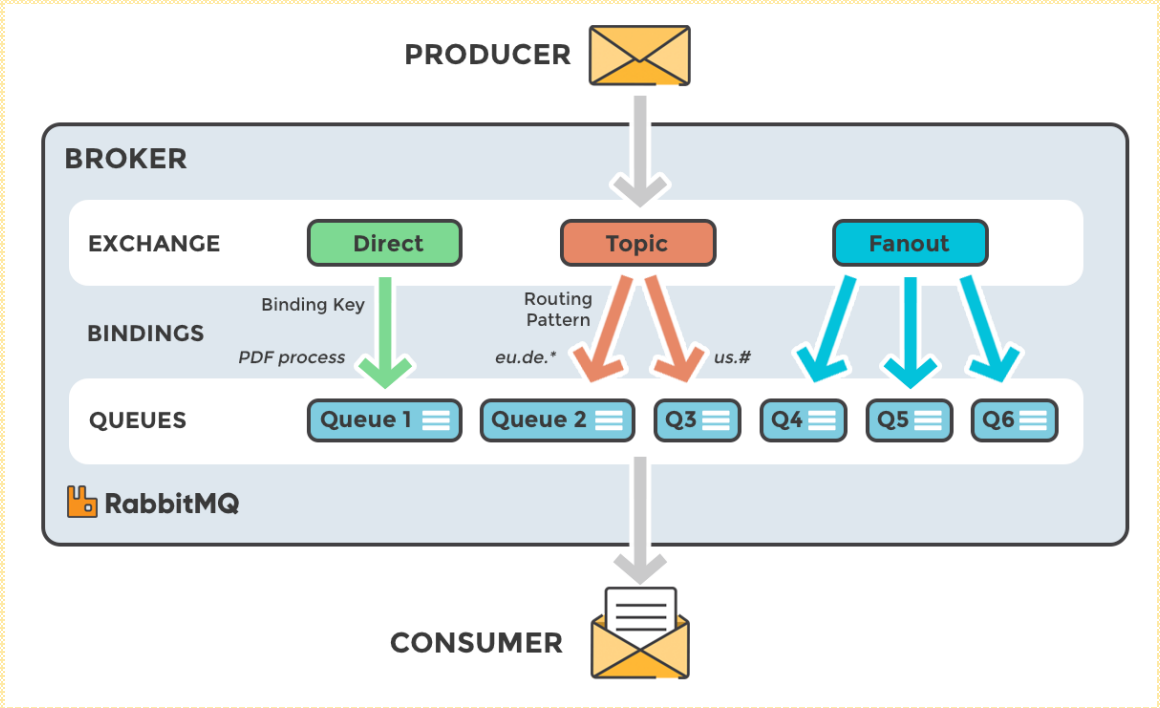
- **Reliable Message Delivery**: Supports message acknowledgment, persistence, and fault tolerance.
- **Interoperability**: Works across different platforms and languages.
- **Secure Communication**: Supports TLS, SASL, and encryption.
- **Routing Capabilities**: Uses exchanges (Direct, Topic, Fanout, Headers) to route messages.
- **Transactional Messaging**: Allows atomic message operations.
- **Queue Management**: Supports durable queues, priority queues, and delayed messaging.
- **Flow Control**: Prevents overload on message consumers.

Application Layer:

Core Components:

- **Producer:** Application that sends the messages.
- **Consumer:** Application that receives the messages.
- **Queue:** Buffer that stores messages.
- **Message:** Information that is sent from the producer to a consumer through RabbitMQ.
- **Connection:** A TCP connection between your application and the RabbitMQ broker.
- **Channel:** A virtual connection inside a connection. When publishing or consuming messages from a queue - it's all done over a channel.
- **Exchange:** Receives messages from producers and pushes them to queues depending on rules defined by the exchange type. To receive messages, a queue needs to be bound to at least one exchange.
- **Binding:** A binding is a link between a queue and an exchange.





Application Layer:

Types of Exchanges in AMQP

- **Direct:** The message is routed to the queues whose binding key exactly matches the routing key of the message. For example, if the queue is bound to the exchange with the binding key *pdfprocess, a message published to* the exchange with a routing key *pdfprocess is routed to that queue.*
- **Fanout:** A fanout exchange routes messages to all of the queues bound to it.
- **Topic:** The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
- **Headers:** Headers exchanges use the message header attributes for routing.

Use Cases:

- Banking and financial transaction systems.
- E-commerce platforms for order and inventory handling.
- Cloud-based backend systems.
- Enterprise micro services integration.
- Secure healthcare messaging systems.

Feature	CoAP	MQTT	AMQP	XMPP	DDS
Full Form	Constrained Application Protocol	Message Queuing Telemetry Transport	Advanced Message Queuing Protocol	Extensible Messaging and Presence Protocol	Data Distribution Service
Model	Client–Server (RESTful)	Publish–Subscribe (Brokered)	Publish–Subscribe (Brokered)	Publish–Subscribe & Presence-based	Publish–Subscribe (Brokerless)
Transport	UDP	TCP	TCP	TCP	TCP, UDP, Shared Memory
Message Format	Binary, compact	Lightweight binary	Binary (AMQP 1.0)	XML-based	Binary
QoS Support	Basic (Confirmable, Non-confirmable)	Yes (QoS 0, 1, 2)	High (Reliable delivery, transactions)	Basic (some via extensions)	Very High (20+ customizable policies)
Security	DTLS	SSL/TLS	SSL/TLS	TLS, SASL	TLS, RTPS security plugins
Best Use Case	Resource-constrained IoT devices	Sensor data transmission	Enterprise-level messaging	Chat, presence, pub-sub in real-time	Mission-critical real-time systems
Architecture Type	RESTful (Web-like)	Centralized broker	Centralized broker	Centralized or federated	Decentralized (peer-to-peer possible)
Scalability	Moderate	High	High	Moderate	Very High
Latency	Very Low	Low	Medium	Medium	Ultra Low
Interoperability	High (Web-compatible)	Medium	High	High	High

Based on Requirements

Requirement	Suggested Protocol
Battery-powered devices, low overhead	MQTT or CoAP
Secure, enterprise-level message queue	AMQP
Real-time performance with QoS tuning	DDS
Chat, messaging, presence-based systems	XMPP
RESTful interface with low power	CoAP

Example Matching Applications

Application	Recommended Protocol
Smart Street Lighting	MQTT / CoAP
Industrial Machine Monitoring	DDS / MQTT
Secure Online Order Processing System	AMQP
Smart Irrigation via Mobile App	CoAP / MQTT
Hospital Patient Alert System	XMPP / MQTT
Autonomous Drones Coordination	DDS

LOGICAL DESIGN OF IOT

The logical design of an IoT system refers to an abstract representation of entities and processes without going into the low-level specifies of implementation. it uses Functional Blocks, Communication Models, and Communication APIs to implement a system.

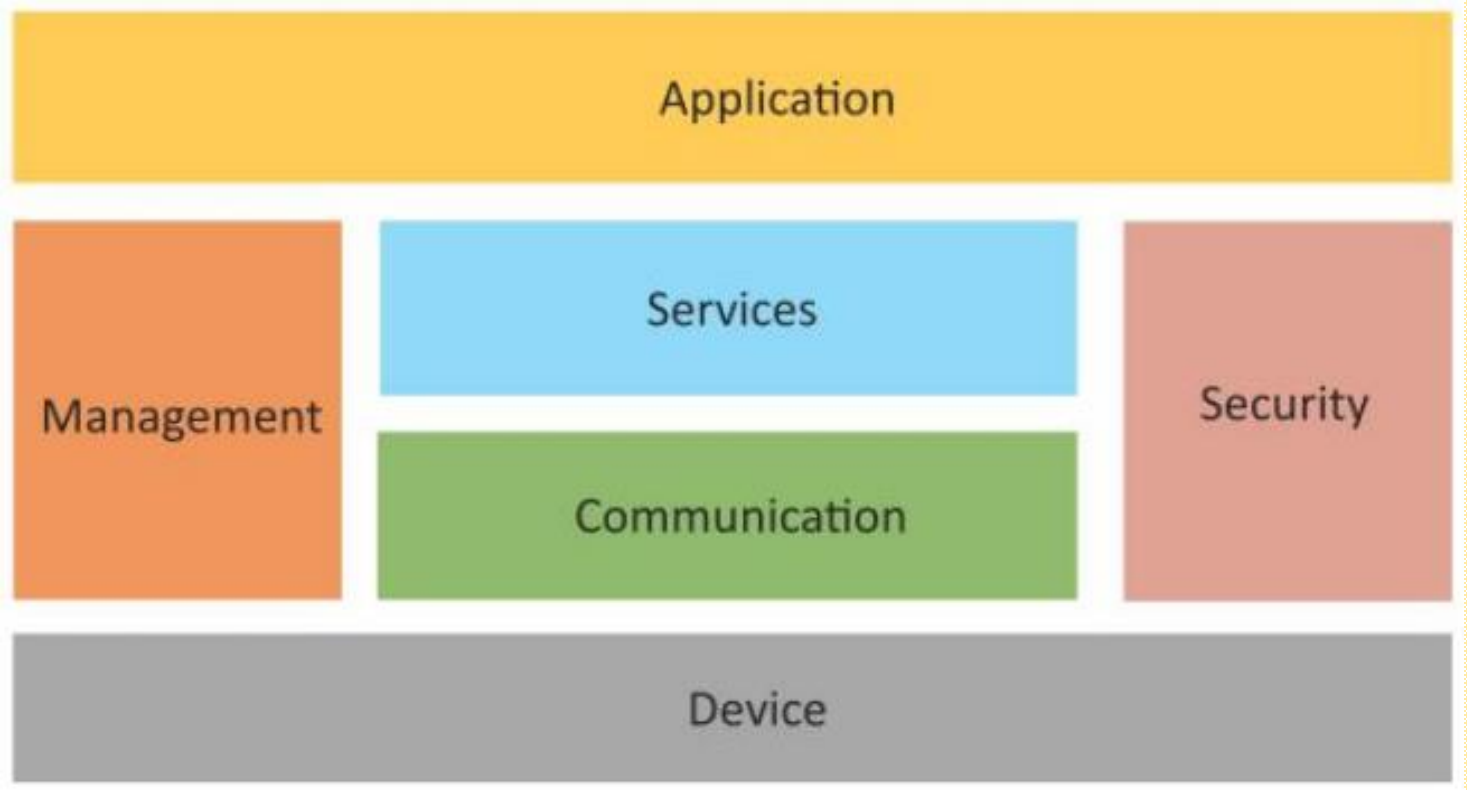
IoT Functional Blocks

IoT Communication Models

IoT Communication APIs

IoT Functional blocks

An IoT system consist number of functional blocks like Devices, services, communication, security, and application that provides the capability for sensing, actuation, identification, communication, and management.



LOGICAL DESIGN OF IOT

These functional blocks consist of devices that provide monitoring control functions, handle communication between host and server, manage the transfer of data, secure the system using authentication and other functions, and interface to control and monitor various terms.

- **Device:** These devices are used to provide sensing and monitoring control functions that collect the data from the outer environment.
- **Communication:** This block handles the communication between the client and cloud-based server and sends/receives the data using protocols.
- **Services:** This functional block provides some services like monitoring and controlling a device and publishing and deleting the data and restore the system.
- **Management:** This functional block provides various functions that are used to manage an IoT system.
- **Security:** This block is used to secure an IoT system using some functions like authorization, data security, authentication, 2 step verification, etc.
- **Application:** It is an interface that provides a control system that use by users to view the status and analyse of system.

LOGICAL DESIGN OF IOT

IoT Communication Models

There are several different types of models available in an IoT system that used to communicate between the system and server like the request-response model, publish-subscribe model, push-pull model, and exclusive pair model, etc.

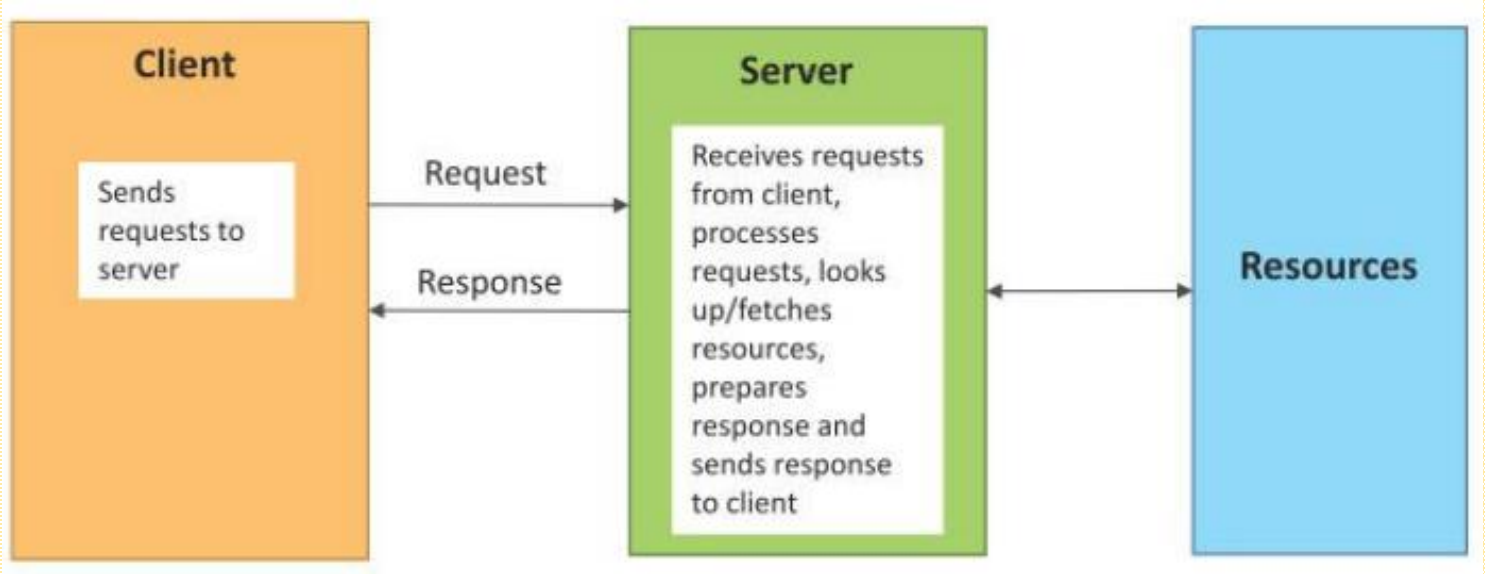
Request-Response Communication Model

This model is a communication model in which a client sends the request for data to the server and the server responds according to the request. when a server receives a request it fetches the data, retrieves the resources and prepares the response, and then sends the data back to the client.

In simple terms, we can say that in the request-response model server send the response of equivalent on the request of the client. in this model, HTTP works as a request-response protocol between a client and server.

Example

When we search a query on a browser then the browser submits an HTTP request to the server and then the server returns a response to the browser(client).



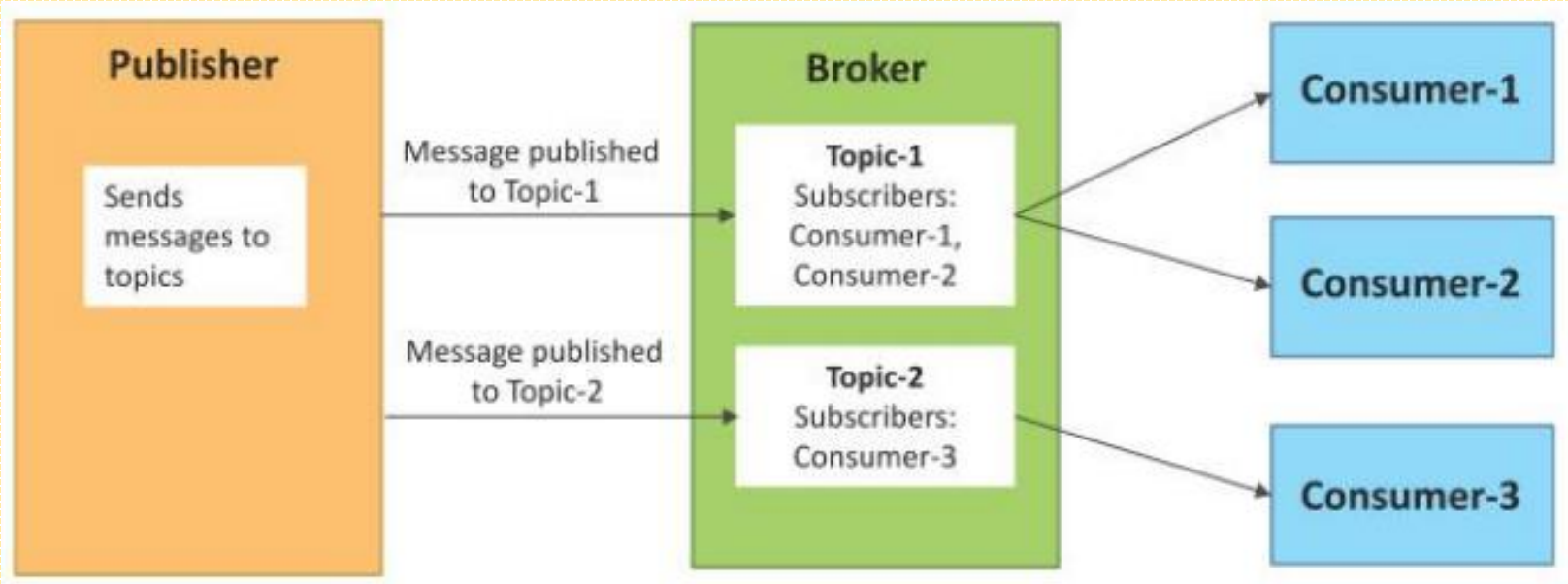
LOGICAL DESIGN OF IOT

Publish-Subscribe Communication Model

- Publish-Subscribe is a communication model that involves publishers, brokers and consumers.
- Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers.
- Consumers subscribe to the topics which are managed by the broker.
- When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

Example

On the website many times we subscribed to their newsletters using our email address. these email addresses managed by some third-party services and when a new article published on the website it directly sends to the broker and then the broker send these new data or post to all the subscribers.



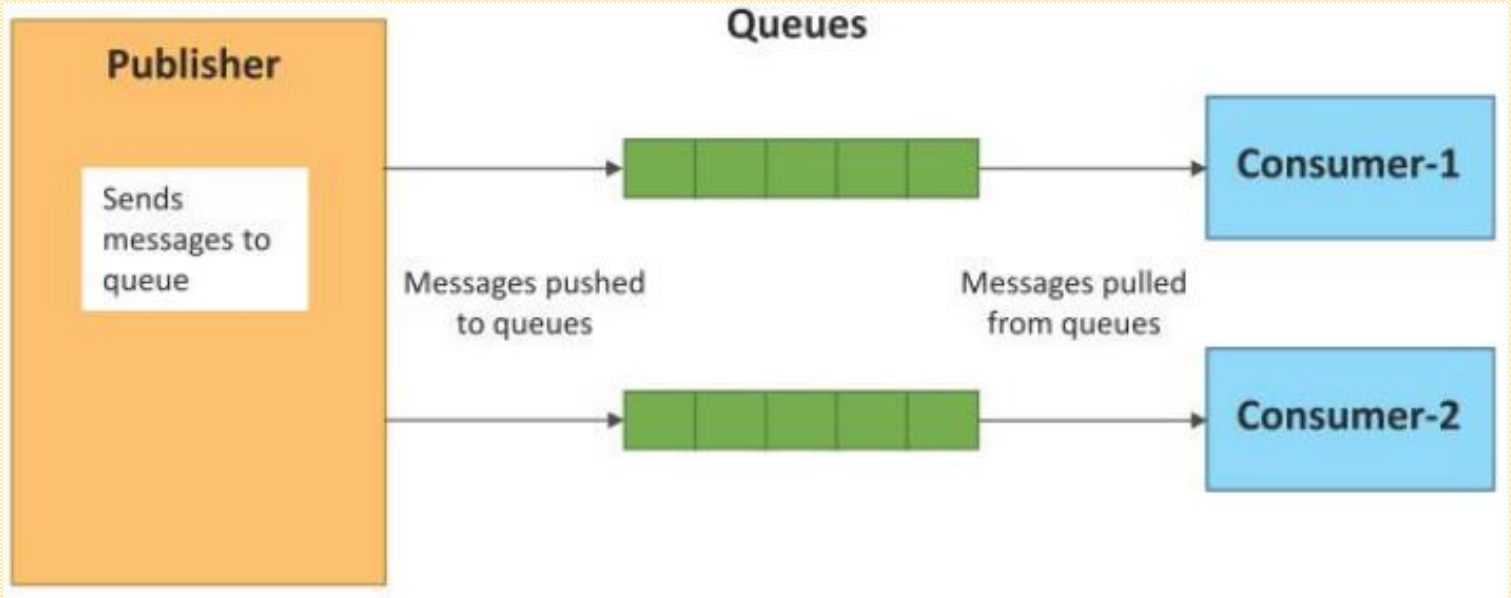
LOGICAL DESIGN OF IOT

Push-Pull Communication Model

- Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.
- Queues help in decoupling the messaging between the producers and consumers.
- Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate at which the consumers pull data.

Example

When we visit a website we saw a number of posts that published in a queue and according to our requirements, we click on a post and start reading it.



LOGICAL DESIGN OF IOT

Push-Pull Communication Model Example

Smart Agriculture System

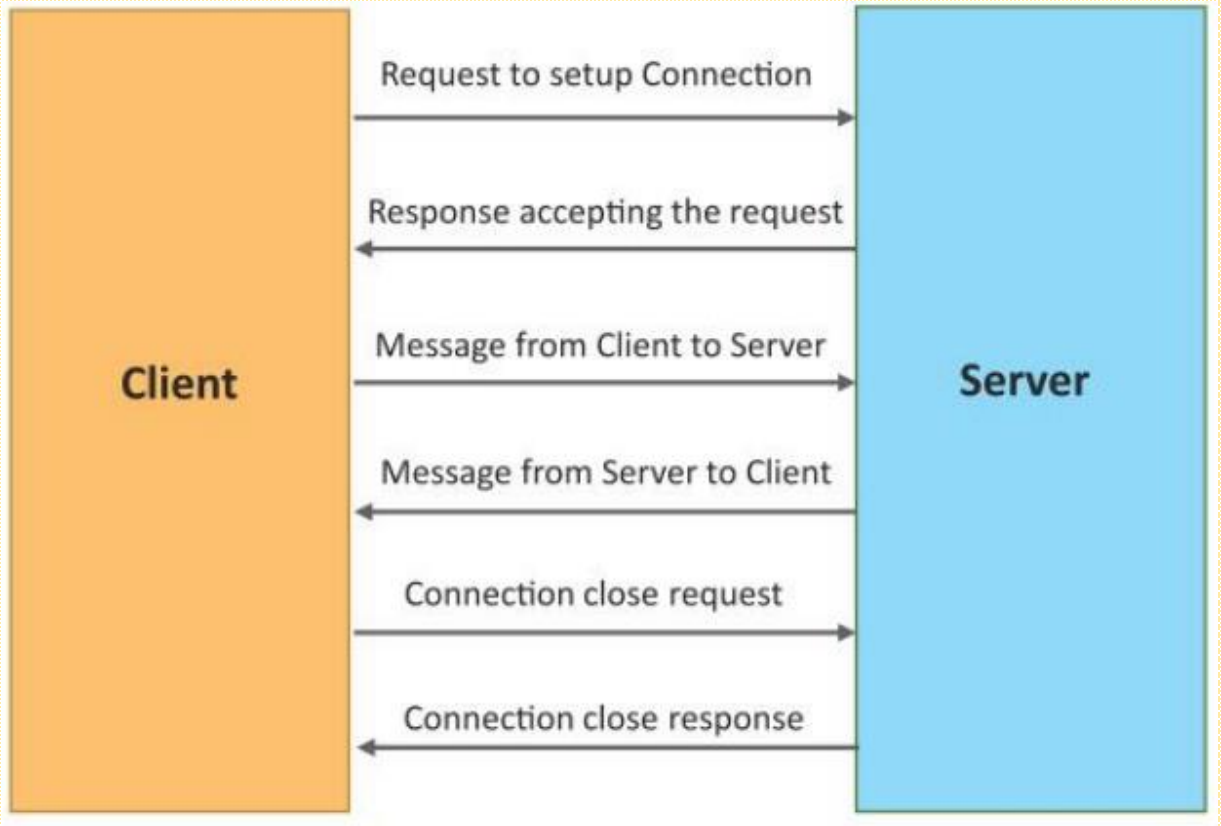
- **Sensors** in a field (e.g., soil moisture, temperature) collect data every 10 minutes.
- These sensors **push** their readings to a **central broker/cloud database** (like AWS IoT or a local edge server).
- A **farmer's dashboard app** periodically **pulls** the data to display graphs or send irrigation commands.
- **Push:** IoT devices **send data automatically** to a broker/cloud (producer role).
- Applications **request the latest data** when needed (consumer role).
- **Example:** Soil sensors pushing data; farmer's app pulling updates on demand.

Function	Technology
Data Push	MQTT, HTTP POST, CoAP
Data Pull	HTTP GET, REST API, GraphQL
Broker Layer	Kafka, RabbitMQ, AWS IoT

LOGICAL DESIGN OF IOT

Exclusive Pair Communication Model

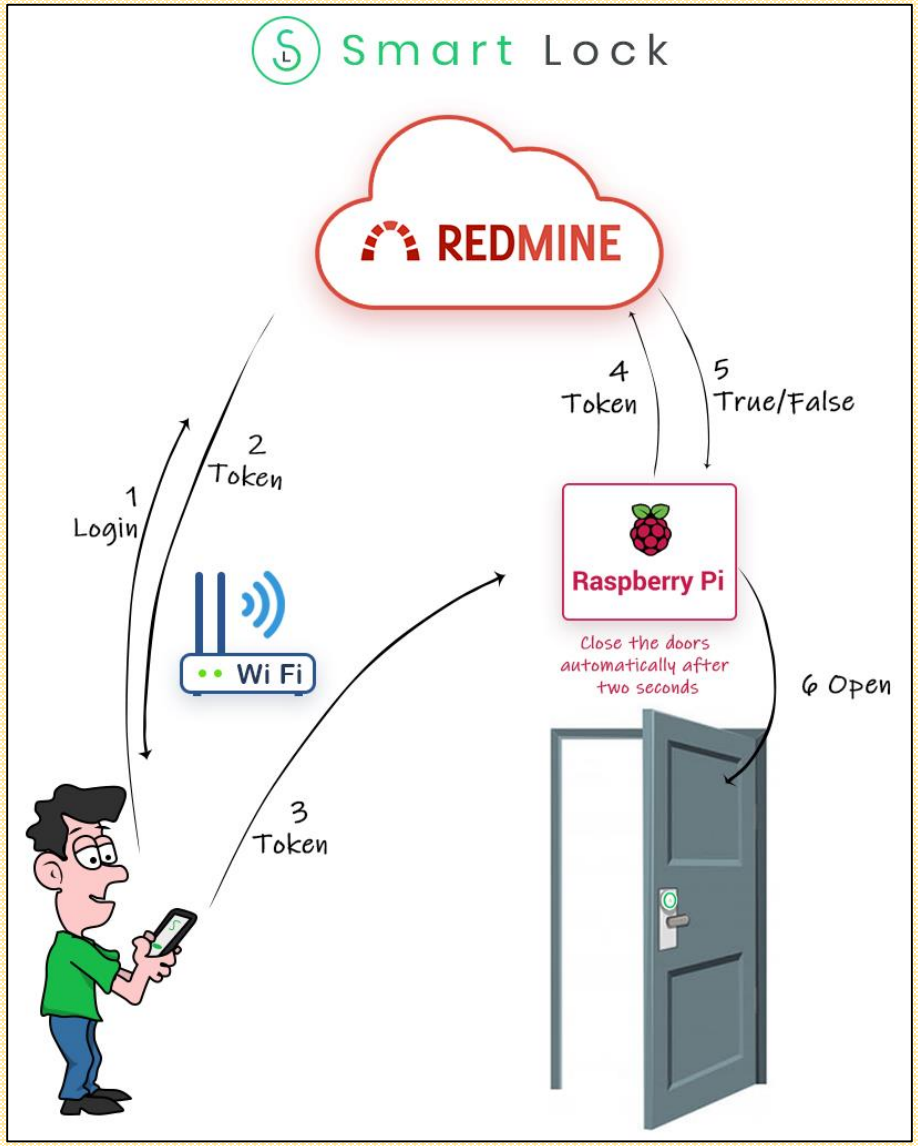
- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.
  - Once the connection is setup it remains open until the client sends a request to close the connection.
  - Client and server can send messages to each other after connection setup.
- Often used in **device-to-device (D2D)** communication
- Ensures **privacy, security, and directness**
- Typically implemented using **persistent socket connections**



LOGICAL DESIGN OF IOT

Example: Smart Lock and Smartphone App

- A **smart lock** at your home is paired exclusively with your **personal smartphone**.
- The communication between the lock and the phone is **encrypted** and **persistent**, meaning:
- Only that smartphone can lock/unlock the door.
- No other device can control the smart lock unless explicitly paired.
- This is often managed using **TLS-secured sockets** over protocols like MQTT/WebSockets or Bluetooth Low Energy (BLE).



LOGICAL DESIGN OF IOT

IoT communication APIs

Generally, we used Two APIs for IoT Communication. These IoT Communication APIs are:

- REST-based Communication APIs
- Web Socket-based Communication API

REST-based Communication APIs

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

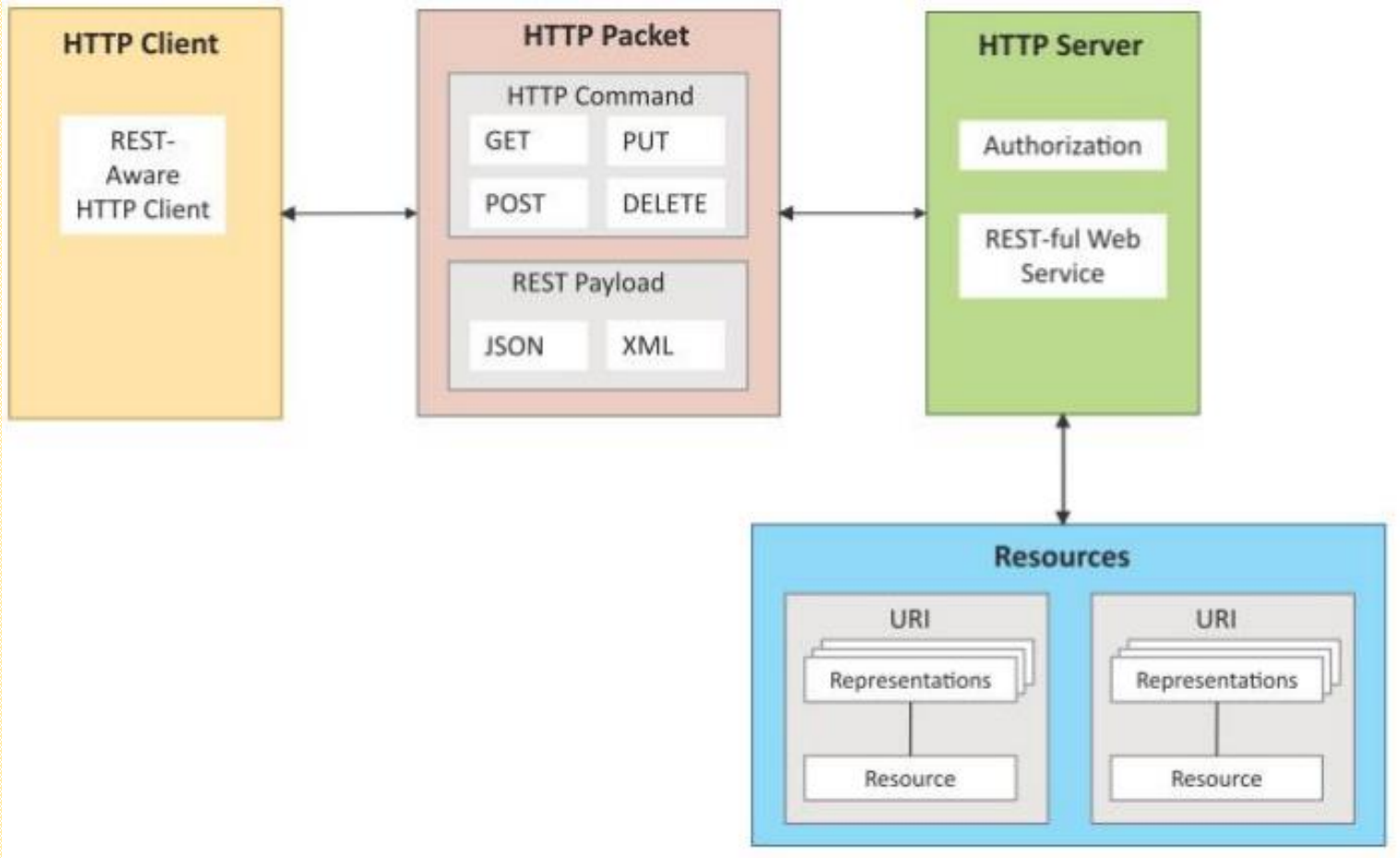
RESTful APIs are designed to be lightweight, stateless, and scalable, using standard web protocols and conventions to make interactions between distributed systems efficient and maintainable.

LOGICAL DESIGN OF IOT

The rest architectural constraint are as follows:

Client-server – A key feature is the client-server architecture, where the client and server are independent. This separation of concerns allows each component to evolve independently, provided the interface between them is maintained.

Stateless – Each client request to the server must contain all the information needed to understand and process the request. The server does not store any client context between requests, which makes the interactions simple and scalable.



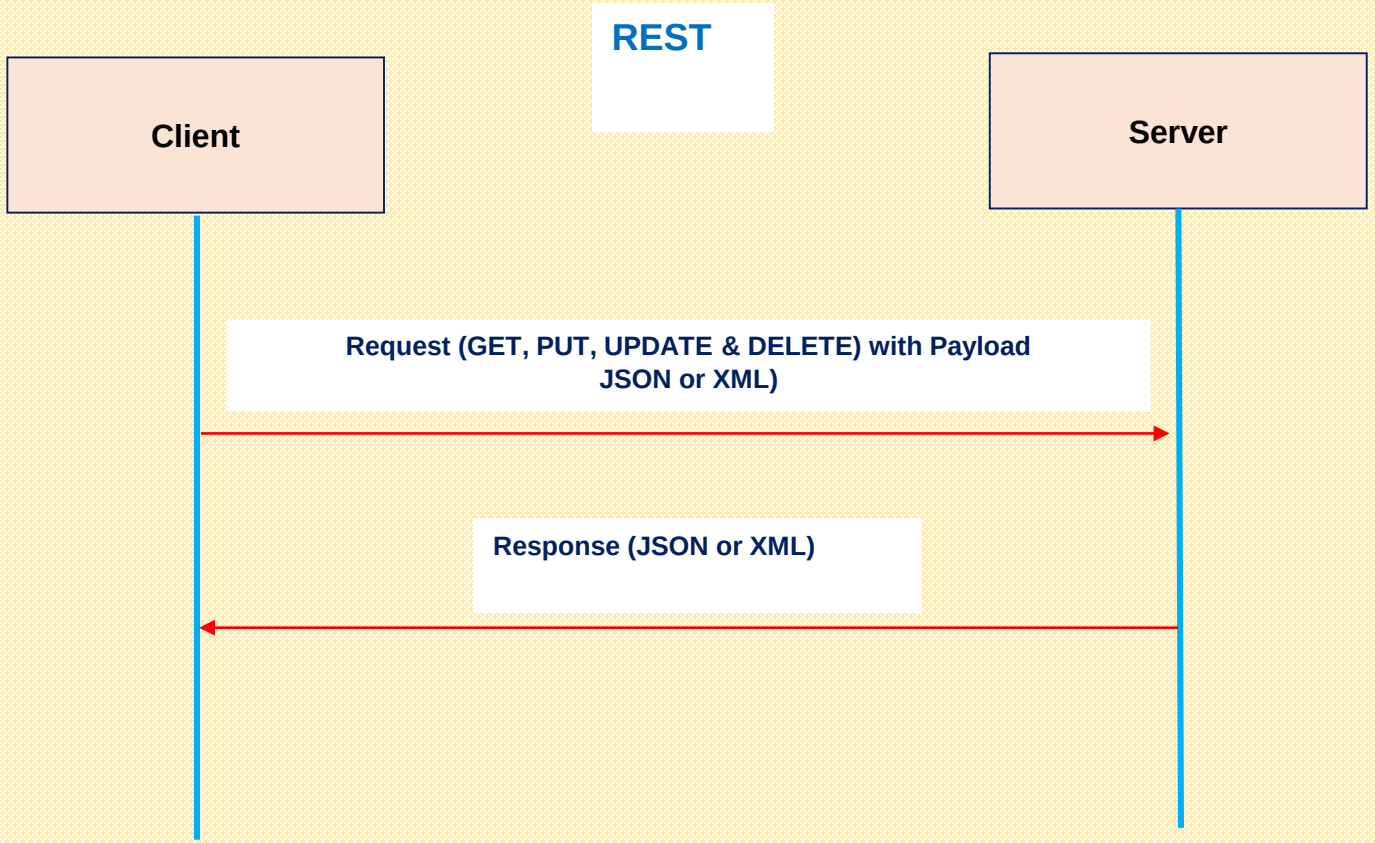
LOGICAL DESIGN OF IOT

- **Cache-able** – It refers to the ability to store responses to client requests in such a way that future, identical requests can be served faster without needing to contact the server again. This helps reduce latency, improve performance, and decrease the load on servers.
- **Layered system** – The use of layered system architecture allows REST APIs to be composed of hierarchical layers, enabling features like load balancing, caching, and proxy servers without the client knowing the underlying implementation.
- **Uniform interface** – It ensures that all interactions between client and server are standardized and follow a consistent structure. This includes the use of standard HTTP methods like GET, POST, PUT, DELETE, and PATCH, and resource identification through URIs.
- **Representation of Resources:** Resources are abstract concepts, and they are represented in a format such as JSON or XML when transferred over the network. For example, a user resource may be represented as a JSON object with fields like name, email, and ID.
- **Code on demand** – Servers can extend client functionality by sending executable code like JavaScript. Although not commonly used, this feature can add flexibility in some use cases.

LOGICAL DESIGN OF IOT

Request Response Model used by REST API:

A RESTful web service is a “WebAPI” implemented using HTTP and REST principles. REST is most popular IoT Communication APIs.



LOGICAL DESIGN OF IOT

HTTP Request Methods and Actions

Uniform Resource Identifier (URI)	GET	PUT	POST	DELETE
Collection, such as https://api.example.com/resources/	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element, such as https://api.example.com/resources/item5	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it does not exist, create it.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.	Delete the addressed member of the collection.

LOGICAL DESIGN OF IOT

HTTP Request Feature, Methods and Actions

Feature	Description
Stateless	Each request from client to server must contain all the information needed.
Client-Server	Separation between client (user interface) and server (data processing).
Uniform Interface	Uses standard HTTP methods like GET, POST, PUT, DELETE.
Resource-Based	Resources are identified using URIs (e.g., /temperature, /device/1).
Cacheable	Responses can be cached to improve performance.

Method	Description	Example URI	Action Performed
GET	Retrieve resource	/sensor/temperature	Get temperature data
POST	Create new resource	/device	Add a new IoT device
PUT	Update existing resource	/device/123	Update device settings
DELETE	Delete a resource	/device/123	Remove a device

LOGICAL DESIGN OF IOT

REST API in Smart Agriculture System

Feature	Request	Response

GET Request to Read Sensor Data	GET /sensor/moisture HTTP/1.1
	Host: api.smartfarm.com
		{
		"sensor_id": "moisture01",
		"value": "23%",
		"timestamp": "2025-08-05T10:45:00Z"
		}

POST Request to Add a New Device	POST /device HTTP/1.1
	Content-Type: application/json
	{
	"device_id": "pump1",
	"type": "water_pump",
	"location": "Field A"
	}
		{
		"message": "Device added successfully", "device_id": "pump1"
		}

PUT Request to Update Device Configuration	PUT /device/pump1 HTTP/1.1
	Content-Type: application/json
	{
	"status": "ON", "mode": "auto"
	}
		{
		"message": "Pump configuration updated"
		}

DELETE Request to Remove a Device	DELETE /device/pump1 HTTP/1.1	{
		"message": "Device pump1 removed"
		}

LOGICAL DESIGN OF IOT

WebSocket based communication API

- WebSocket is a full-duplex, bidirectional communication protocol that operates over a single TCP connection.
- This API uses full-duplex communication so it does not require a new connection setup every time when it requests new data.
- Unlike HTTP (which is request-response-based and stateless), WebSocket allows real-time, continuous communication between a client (like a browser or IoT device) and a server.
- WebSocket communication begins with a connection setup request sent by the client to the server. The request (called WebSocket handshake) is sent over HTTP and the server interprets it is an upgrade request. If the server supports WebSocket protocol, the server responds to the WebSocket handshake response.
- This type of API reduces the traffic and latency of data and makes sure that each time when we request new data it cannot terminate the request.

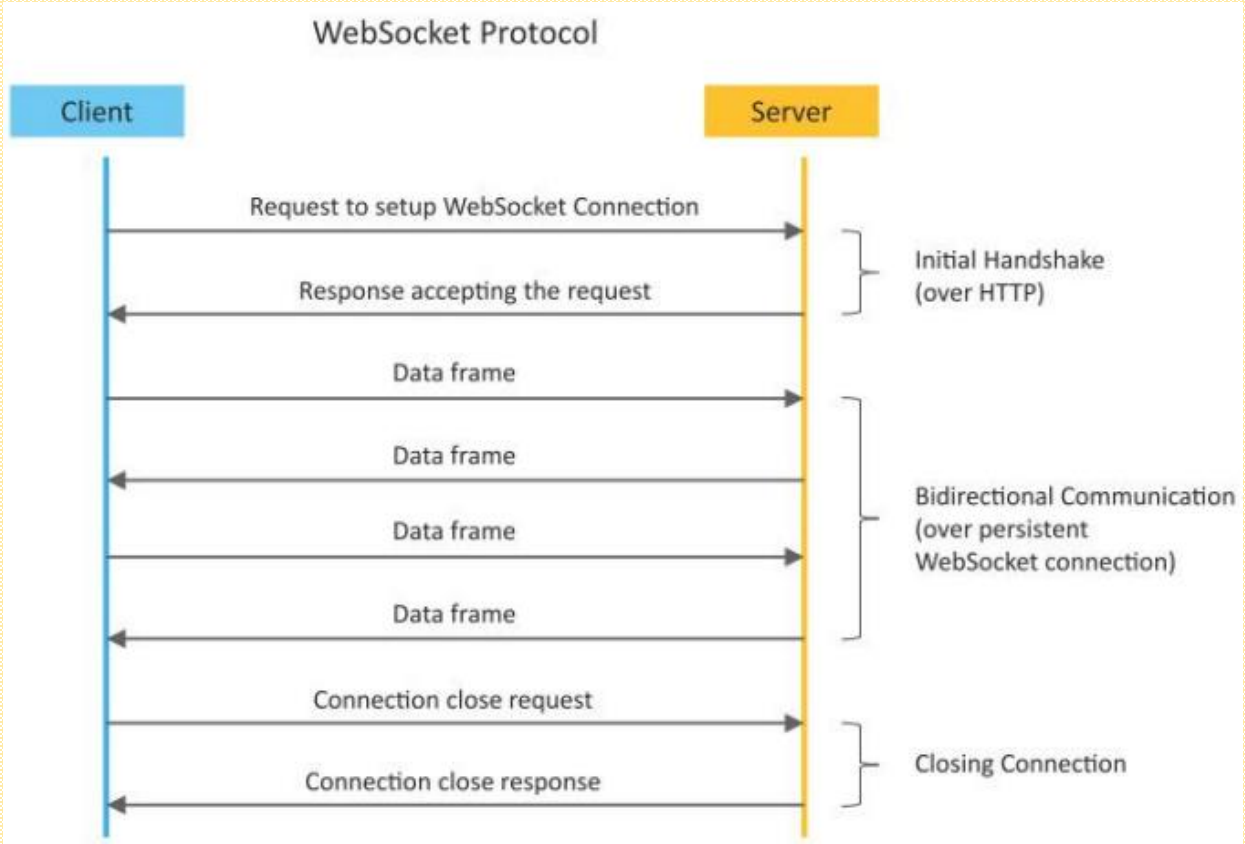
LOGICAL DESIGN OF IOT

Key Features of WebSocket based communication API

- **Full Duplex Connection:** Both client and server can send messages independently at any time.
- **Low Latency (Realtime):** Ideal for real-time communication (chat apps, IoT monitoring, gaming).
- **Persistent Connection:** Maintains a single connection until one side closes it.
- **Efficient:** Less overhead compared to opening new HTTP connections for each message.

How it works:

- **Client initiates a handshake** using HTTP with an Upgrade header.
- **Server accepts** and switches the protocol from HTTP to WebSocket.
- **Bi-directional communication** begins.
- **Either side can close** the connection at any time.



IOT ENABLING TECHNOLOGIES

There are several technologies which play important role in IoT Applications. The technologies are:

- Wireless Sensor Networks
- Cloud Computing
- Big Data Analytics
- Embedded Systems
- Communication Protocols
- Security Protocols
- Web Services
- Mobile Internet
- Semantic Search Engines

IOT ENABLING TECHNOLOGIES

Wireless Sensor Networks

- **A Wireless Sensor Network is a network** of distributed devices with sensors that monitor physical or environmental conditions like temperature, humidity, motion, vibration, etc.
- A typical WSN consists of:
  - **End Nodes**: Devices with sensors attached.
  - **Routers**: Forward data packets from end nodes to the coordinator.
  - **Coordinator**: Collects data from all nodes and acts as a gateway to the Internet.

How WSNs Work:

- WSNs rely on wireless protocols like IEEE 802.15.4.
- ZigBee is a popular wireless technology based on this standard.
  - Operates at **2.4 GHz**.
  - Supports data rates up to **250 KB/s**.
  - Typical range: **10 to 100 meters** (depends on power output & environment).

IOT ENABLING TECHNOLOGIES

Applications of Wireless Sensor Networks in IoT

- **Weather Monitoring Systems**
  - Collect temperature, humidity, and other environmental data.
  - Data is aggregated and analyzed.
- **Indoor Air Quality Monitoring**
  - Measure air quality and gas concentrations indoors.
- **Soil Moisture Monitoring**
  - Measure soil moisture at different locations (useful in agriculture).
- **Surveillance Systems**
  - Detect motion and collect security data.
- **Smart Grids**
  - Monitor electrical grid status at multiple points.
- **Structural Health Monitoring**
  - Monitor vibration in buildings or bridges to check structural integrity.

IOT ENABLING TECHNOLOGIES

Key Advantages of WSNs:

- **Large scale deployment** of low-cost, low-power nodes.
- **Continuous monitoring** of environmental & physical conditions.
- **Self-organizing:** No manual reconfiguration needed when nodes fail or new nodes are added.
- **Robust:** Can reconfigure itself automatically for reliable operation.

IOT ENABLING TECHNOLOGIES

Cloud Computing

- Cloud computing is a transformative technology that delivers **applications, computing power, storage, and networking services** over the Internet. It works on a **“pay-as-you-go”** model — users pay only for the resources they use.
- The “cloud” is a term that simply means “the internet.” Computing involves the infrastructures and systems that allow a computer to run and build, deploy, or interact with information.
- In cloud computing, this means that instead of hosting infrastructure, systems, or applications on your hard drive or an on-site server, you’re hosting it on virtual/online servers that connect to your computer through secure networks.

Key Features:

- **On-demand resources:** Users can provision computing, storage, and networking resources as needed.
- **Self-service:** Users don’t need direct interaction with a human provider to provision resources.
- **Multi-tenancy:** Resources are pooled to serve multiple users simultaneously.
- **Platform independence:** Accessible from different devices (PCs, laptops, tablets, smartphones).
- **Virtualization:** Users access virtual machines and storage, not the underlying physical hardware.

IOT ENABLING TECHNOLOGIES

Cloud Computing Service Models:

- **SaaS or Software as a Service.**
  - SaaS means instead of installing software on your computer, you access the platform online.
  - Provides **ready-to-use software applications** over the Internet.
  - User just uses the app — no need to manage servers, OS, or software updates.
  - Platform independent — accessible via browser on any device.
  - Example: A web-based IoT dashboard that shows real-time data and analytics. Deploying a custom server for storing IoT sensor data.
    - Square, which processes payments online
    - Google Apps such as Google Drive or Calendar
    - Slack, which allows collaboration and chat between other users

IOT ENABLING TECHNOLOGIES

Cloud Computing Service Models:

- **IaaS or Infrastructure as a Service.**
  - IaaS provides infrastructure components such as servers, storage, networking, security, and moreover the cloud.
  - Provides virtual machines, storage, and networking.
  - Users manage: virtual machines, OS, applications.
  - **Billing:** Pay for compute/storage used.
  - Example: Deploying a custom server for storing IoT sensor data. Develop an IoT analytics application using cloud-hosted tools.
    - Dropbox, a file storage and sharing system
    - Microsoft Azure, which offers backup and disaster recovery services, hosting, and more
    - Rackspace, which offers data, security, and infrastructure services.

IOT ENABLING TECHNOLOGIES

Cloud Computing Service Models:

❖ **PaaS or Platform as a Service.**

- PaaS provides computing platforms such as operating systems, programming language execution environments, databases, and web servers.
- Provides a platform to develop, run, and manage applications.
- Includes development tools, APIs, and services.
- Users only manage their applications — not the underlying infrastructure.
- Example: Develop an IoT analytics application using cloud-hosted tools:
  - Google App Engine and Heroku, which allow developers to develop and serve apps

- **Serverless Computing.** Serverless computing (also called simply “Serverless”) is simply using a server on the cloud. This offers more elasticity, easier maintenance, and is often more price effective than hosting servers on-site.

IOT ENABLING TECHNOLOGIES

Big Data Analytics

- Big Data is a massive amount of data sets that cannot be stored, processed, or analyzed using traditional tools.
- **So, what makes data "big"?**
  - Big data is characterized by the five V's: volume, velocity, variety, variability, and value. It's complex, so making sense of all of the data in the business requires both innovative technologies and analytical skills.
- Big data analytics describes the process of uncovering trends, patterns, and correlations in large amounts of raw data to help make data-informed decisions. These processes use familiar statistical analysis techniques—like clustering and regression—and apply them to more extensive datasets with the help of newer tools.

Why Big Data Analytics Matters for IoT

- Big Data Analytics allows IoT systems to:
  - Detect patterns (e.g., predicting machine failures)
  - Make smart decisions (e.g., adjusting temperature automatically)
  - Optimize processes (e.g., reduce energy usage)
  - Provide insights for innovation and improved services.

IOT ENABLING TECHNOLOGIES

Big Data Analytics in IoT

- Big data analytics refers to collecting, processing, cleaning, and analyzing large datasets to help organizations operationalize their big data.
- IoT systems generate massive amounts of data every second from sensors, devices, and machines. Managing this data requires Big Data Analytics, which uses advanced techniques and tools to:
  - **Clean** and organize raw data (data cleansing and wrangling)
  - **Store** data efficiently
  - **Process & analyze** it for patterns, trends, or insights
  - **Visualize** results to support decision-making

IOT ENABLING TECHNOLOGIES

Big Data Analytics

- Analyze Data: Getting big data into a usable state takes time. Once it's ready, advanced analytics processes can turn big data into big insights. Some of these big data analysis methods include:
  - **Data mining** sorts through large datasets to identify patterns and relationships by identifying anomalies and creating data clusters.
  - **Predictive analytics** uses an organization's historical data to make predictions about the future, identifying upcoming risks and opportunities.
  - **Deep learning** imitates human learning patterns by using artificial intelligence and machine learning to layer algorithms and find patterns in the most complex and abstract data.

IOT ENABLING TECHNOLOGIES

Tool/Technology	Purpose/Description
Hadoop	Open-source framework to store and process large datasets on clusters of commodity hardware. Handles structured and unstructured data.
MapReduce	Core component of Hadoop. Performs distributed processing in two steps: Map (distribute/filter data) and Reduce (aggregate results).
YARN (Yet Another Resource Negotiator)	Resource management and job scheduling technology for Hadoop clusters.
Spark	Open-source cluster computing framework. Supports fast batch and stream processing with in-memory computing.
NoSQL Databases	Non-relational databases for unstructured/variable data. Examples: MongoDB (frequent data changes), Cassandra (distributed storage for big chunks of data).
Tableau	End-to-end analytics and visualization platform for prepping, analyzing, and sharing big data insights.
Talend	Tool for data integration and management (ETL – Extract, Transform, Load).
STORM	Open-source real-time computational system for processing data streams instantly.
Kafka	Distributed streaming platform for fault-tolerant, high-throughput data pipelines and real-time messaging.

IOT ENABLING TECHNOLOGIES

Artificial Intelligence & Machine Learning in IoT

- Artificial Intelligence (AI) in IoT makes devices “smart” — they can learn, reason, and make decisions without constant human instructions.
- Machine Learning (ML) is a branch of AI that trains IoT systems to learn from data, recognize patterns, and improve automatically with experience.

How AI & ML Work in IoT

- **Collect Data:**

  IoT devices generate huge amounts of real-time data through sensors.
- **Analyze & Learn:**

  ML algorithms analyze this data to find patterns, detect anomalies, or make predictions.
- **Act Smartly:**

  Based on analysis, IoT devices can automatically adjust, send alerts, or take actions without human intervention.

IOT ENABLING TECHNOLOGIES

Key Benefits:

- Enables automation and intelligent decision-making.
- Reduces human effort and errors.
- Improves efficiency and safety.
- Provides valuable insights from IoT data.

Examples:

- **Smart Home:**

  AI enables voice assistants (like Alexa) to learn user preferences and automate home devices.

- **Predictive Maintenance:**

  ML models predict equipment failures in factories, so maintenance can be done **before** breakdowns occur.

- **Healthcare:**

  Wearable IoT devices use AI to track health data and alert users or doctors about unusual patterns.

- **Smart Vehicles:**

  Self-driving cars use AI and ML to detect objects, read signs, and make driving decisions in real-time.

IOT ENABLING TECHNOLOGIES

Communication Protocols in IoT

- Communication protocols form the backbone of IoT systems.
- They enable connectivity between IoT devices and allow interaction with applications.
- Protocols ensure seamless data exchange over the network.

Purpose of Communication Protocols:

- Allow devices to exchange data reliably and efficiently.
- Enable coupling between sensors, actuators, and cloud/server applications.

Types of Protocols:

- **Link Layer Protocols** – e.g., Ethernet, Wi-Fi, Bluetooth, Zigbee
- **Network Layer Protocols** – e.g., IPv4, IPv6
- **Transport Layer Protocols** – e.g., TCP, UDP
- **Application Layer Protocols** – e.g., HTTP, MQTT, CoAP

IOT ENABLING TECHNOLOGIES

Key Functions:

- **Data Exchange Formats**: Define how data is structured.
- **Data Encoding**: Representing data in machine-readable formats.
- **Addressing Scheme**: Ensures data reaches the correct destination device.
- **Routing**: Guides data from source to destination through the network.

Additional Functionalities:

- **Sequence Control**:
  - Maintains the correct order of packets.
  - Helps in detecting missing or duplicate packets.
- **Flow Control**:
  - Regulates the rate of data transmission.
  - Ensures sender doesn’t overwhelm receiver or the network.
- **Retransmission of Lost Packets**:
  - Enhances reliability by re-sending lost or corrupted data packets.

IOT ENABLING TECHNOLOGIES

What is an Embedded System in IoT?

An Embedded System is a microcontroller or microprocessor-based hardware system with software programmed for a specific function. In the context of IoT (Internet of Things), embedded systems form the core of “smart” devices — they sense, process, communicate, and often actuate in response to environmental data.

Key Components of an IoT Embedded System

Component	Function	Examples
Microcontroller / Microprocessor	Controls the system, executes code	Arduino, ESP32, Raspberry Pi Pico
Sensors	Collect data from the environment	DHT11 (Temp/Humidity), MQ-135 (Gas)
Actuators	Perform physical actions	Motors, Relays, LED, Buzzer
Communication Module	Enables data transfer between device and cloud/network	Wi-Fi (ESP8266), Bluetooth, LoRa
Power Supply	Powers the device	Batteries, Solar cells, Power banks
Memory	Stores code (flash) and data (RAM)	EEPROM, SRAM
Software/Firmware	Embedded code for control logic and communication	C/C++, MicroPython, Arduino IDE

IOT ENABLING TECHNOLOGIES

How Embedded Systems Work in IoT

- **Sense**: Data is collected through sensors (e.g., temperature, light).
- **Process**: Microcontroller processes the data (e.g., apply threshold or logic).
- **Communicate**: Sends data to a server/cloud via communication protocols.
- **Actuate**: Performs an action if required (e.g., turn on fan if too hot).
- **Monitor/Control**: Remote dashboard or app monitors or controls the device.

Example: Smart Street Light System

Component	Details
Microcontroller	ESP32 (Wi-Fi enabled)
Sensor	LDR (Light sensor)
Actuator	Relay module connected to street light
Communication	Wi-Fi + HTTP API to send data to cloud
Power	Solar panel with battery
Functionality	Turns light ON at dusk, OFF at dawn

IOT ENABLING TECHNOLOGIES

Applications of Embedded Systems in IoT

- Smart Home Devices (thermostats, security systems)
- Industrial Automation (monitoring and control)
- Agriculture (soil moisture, irrigation control)
- Health Monitoring (wearable ECG, pulse oximeter)
- Smart Cities (waste bins, traffic lights, pollution control)
- Environmental Monitoring (weather stations, air quality)

Popular Embedded Boards for IoT

Board	Features
Arduino Uno	Easy to use, digital/analog I/O, no Wi-Fi by default
ESP8266/ESP32	Built-in Wi-Fi/Bluetooth, cheap, powerful
Raspberry Pi	Linux-based, powerful for edge computing
STM32	Industrial-grade, real-time control
BeagleBone Black	High-performance embedded Linux

IOT LEVELS: IoT Level 1

What is a Level-1 IoT System?

- A Level-1 IoT System is the simplest type of IoT system. It consists of:
- A **single node/device** that handles:
  - Sensing/Actuation
  - Data storage
  - Local processing/analysis
  - Application hosting
- These systems are:
  - Low-cost, low-complexity
  - Suitable for applications where:
    - Data is not very large
    - Processing requirements are not computationally intensive

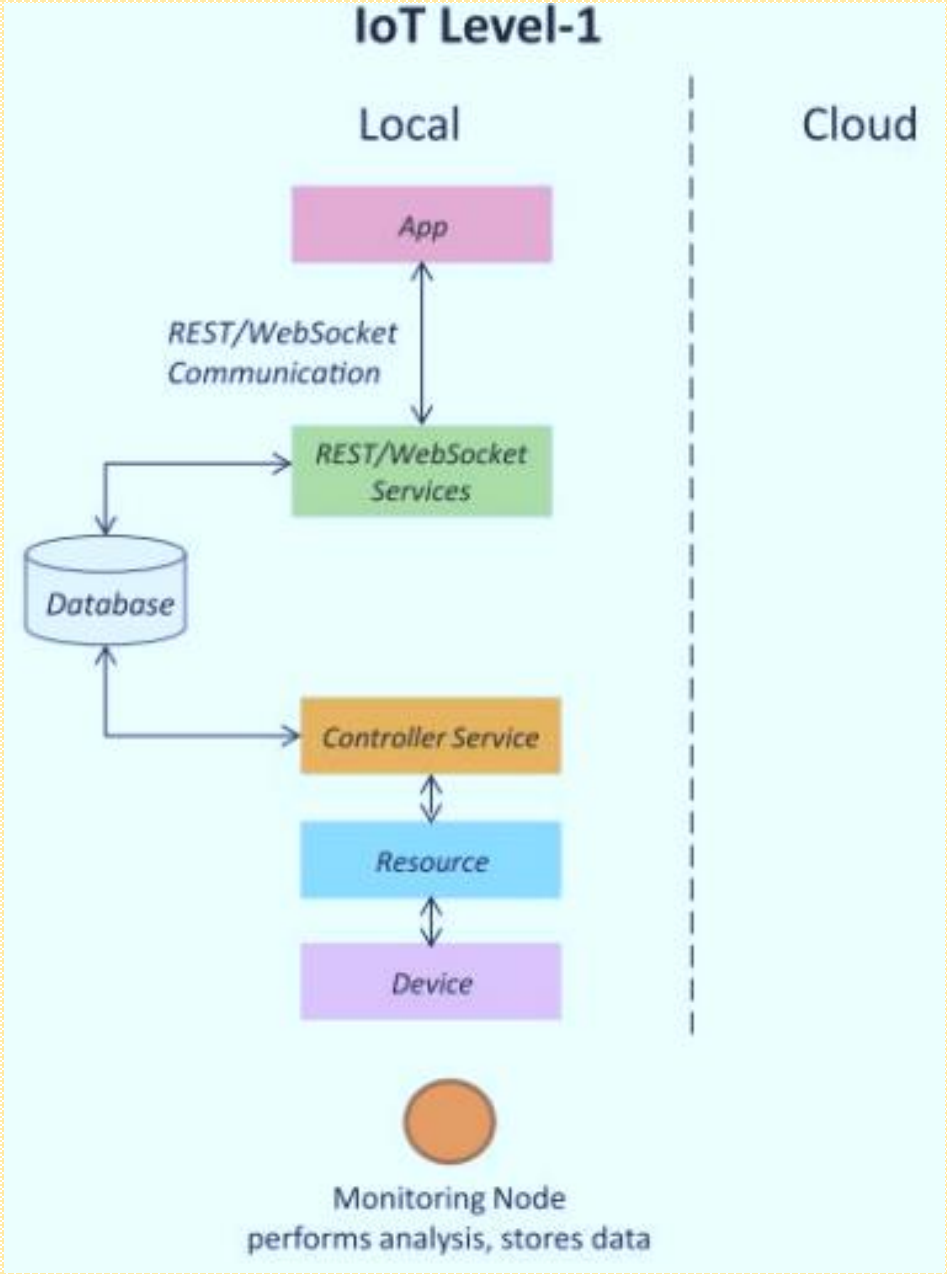
Key Features of Level-1 IoT System:

- **Single Node Function:** All functions (sensing, control, DB, UI) on one device.
- **Local Processing:** No need for cloud or external analytics
- **REST Services:** Allow remote control and automation logic
- **Lightweight & Efficient:** Good for small home setups and educational projects

IOT LEVELS: IoT Level 1

Example Home Automation: Components

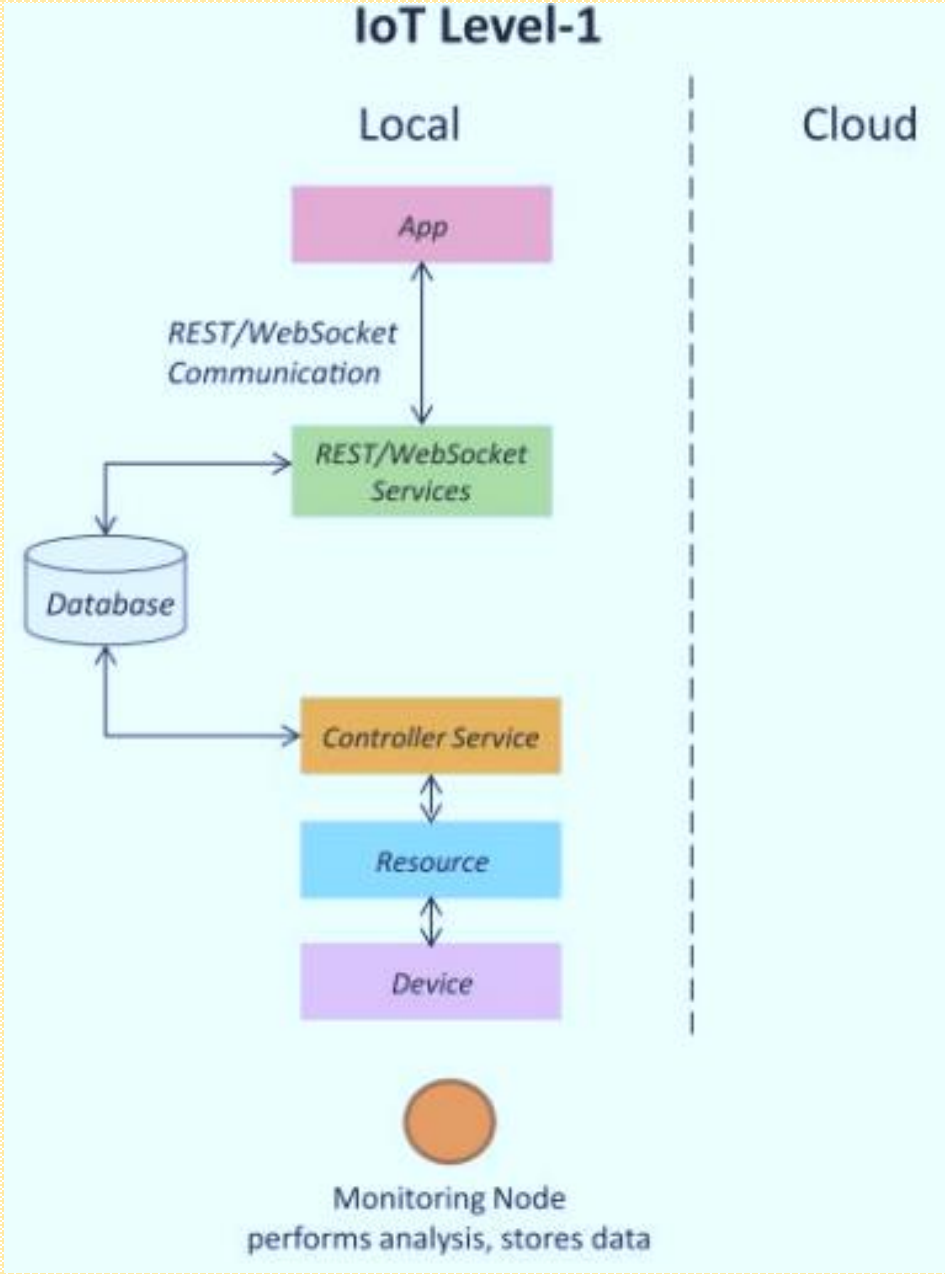
Component	Role
Single Node (e.g., Raspberry Pi)	Performs sensing, control, database storage, and hosting the UI
Sensors (e.g., LDR)	Detect environmental condition (e.g., light level)
Actuators (e.g., relay switches)	Switch lights/appliances ON or OFF
Database (e.g., MySQL)	Stores current state of devices
REST API (e.g., Django REST)	Allows services to retrieve/update device states
Web Application (UI)	User interacts with this to control appliances



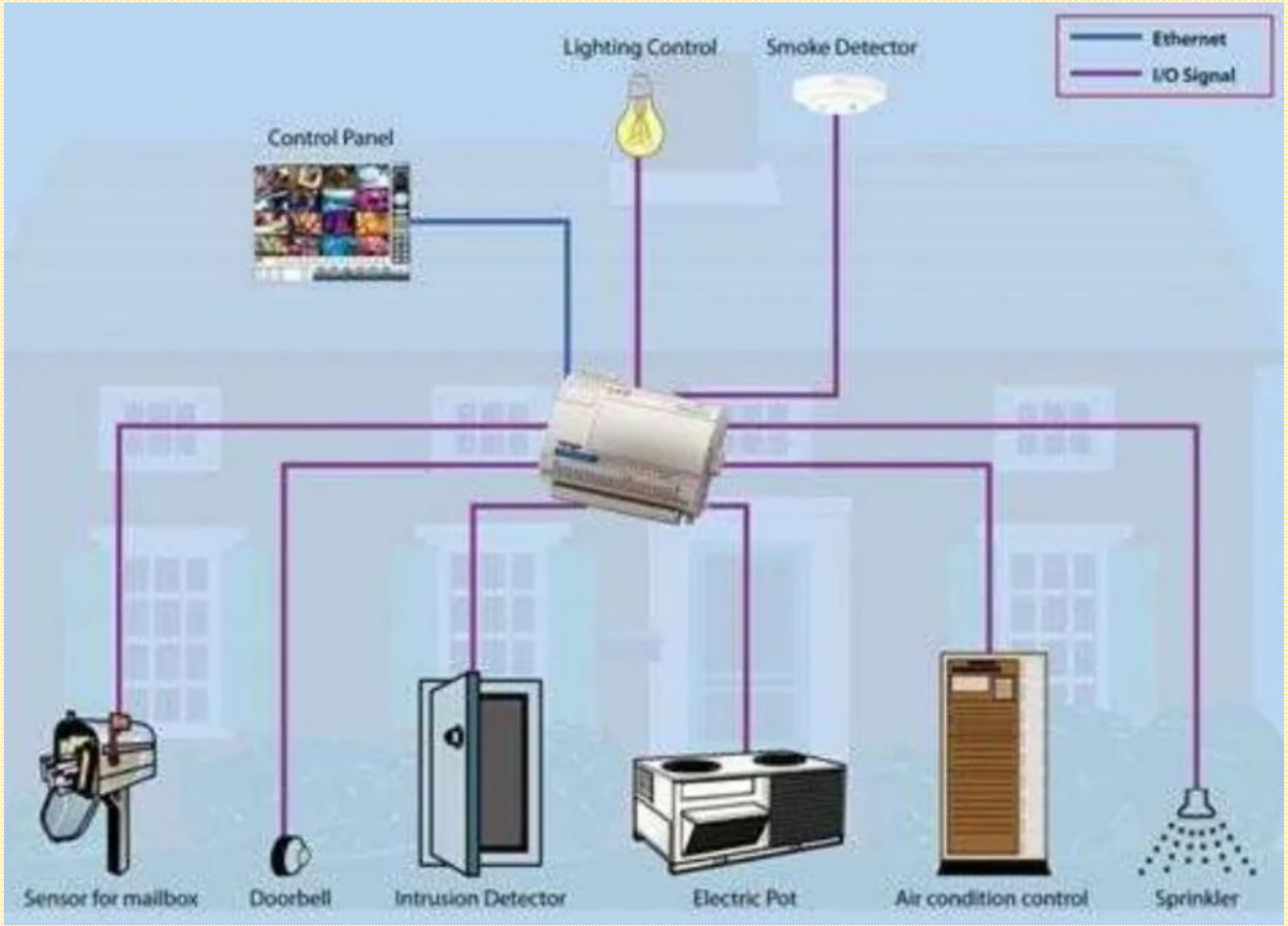
IOT LEVELS: IoT Level 1

How It Works

1. Sensors collect data (e.g., light level in a room).
2. Data is stored in a local database on the device.
3. Controller Service continuously checks the light level:
   - If in auto mode, it turns lights ON/OFF based on light level.
4. Web Application allows:
   - User to see current light state
   - User to control light manually (if in manual mode)
5. RESTful APIs provide services to update/fetch status.
6. Since the device is connected to the Internet, users can:
   - Access the application remotely (e.g., via browser or mobile).



IOT LEVELS: IoT Level 1



IOT LEVELS: IoT Level 2

What is a Level-2 IoT System?

- A Level-2 IoT System is characterized by:
  - A single node/device that handles:
    - Sensing/Actuation
    - Local analysis
  - Cloud storage for data
  - Cloud-hosted application (UI)
- These systems are suitable when:
  - Data volume is large
  - Analysis is light, and can still be done locally

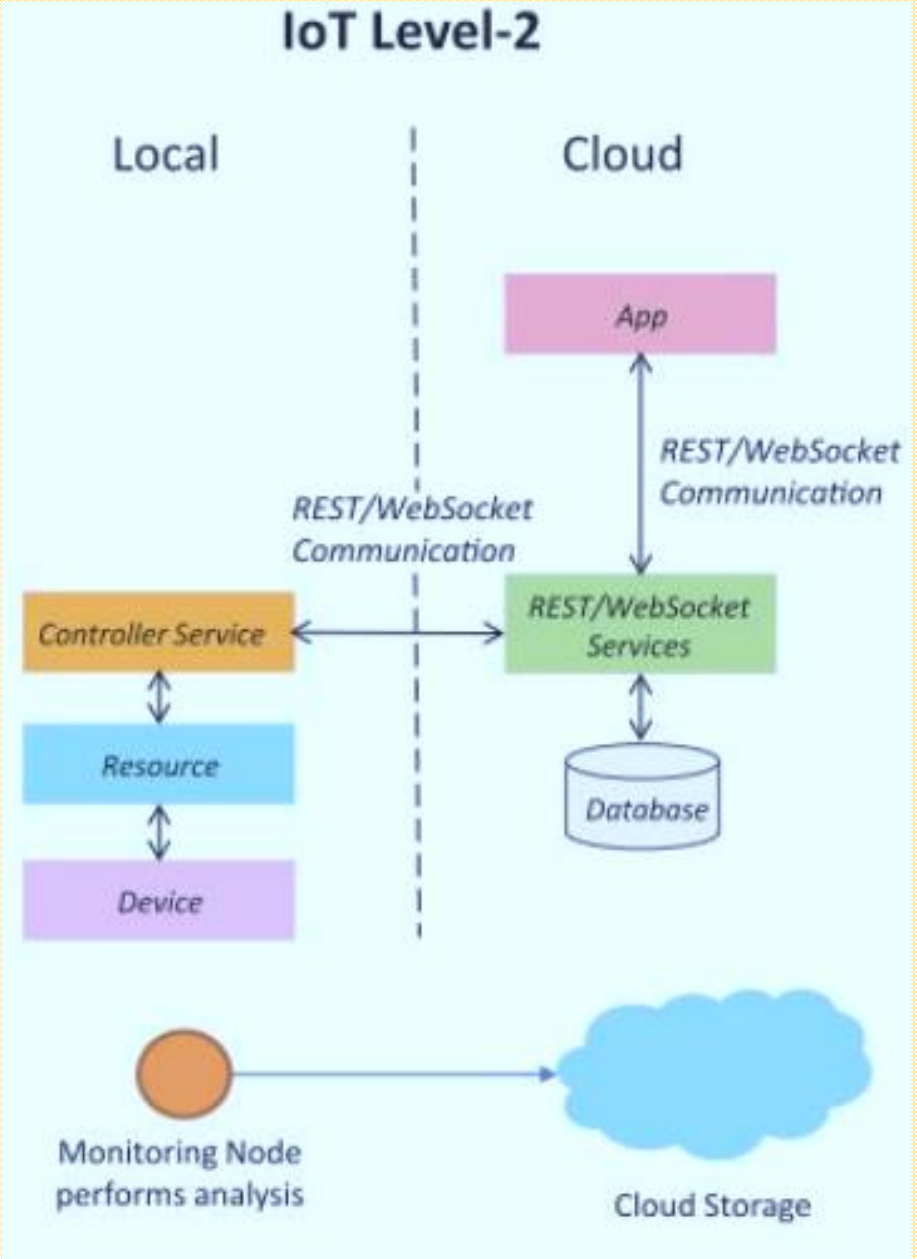
Key Features of Level-2 IoT System:

- Single-node sensing/actuation: Performs local control and sends data to cloud.
- Cloud data storage: Suited for large, historical datasets.
- Cloud-based application: Offers visualization and user-friendly dashboards.
- REST APIs: Enable remote communication with the cloud system.
- Semi-autonomous: Basic intelligence at the edge, enhanced insights from cloud.

IOT LEVELS: IoT Level 2

Example Smart Irrigation System

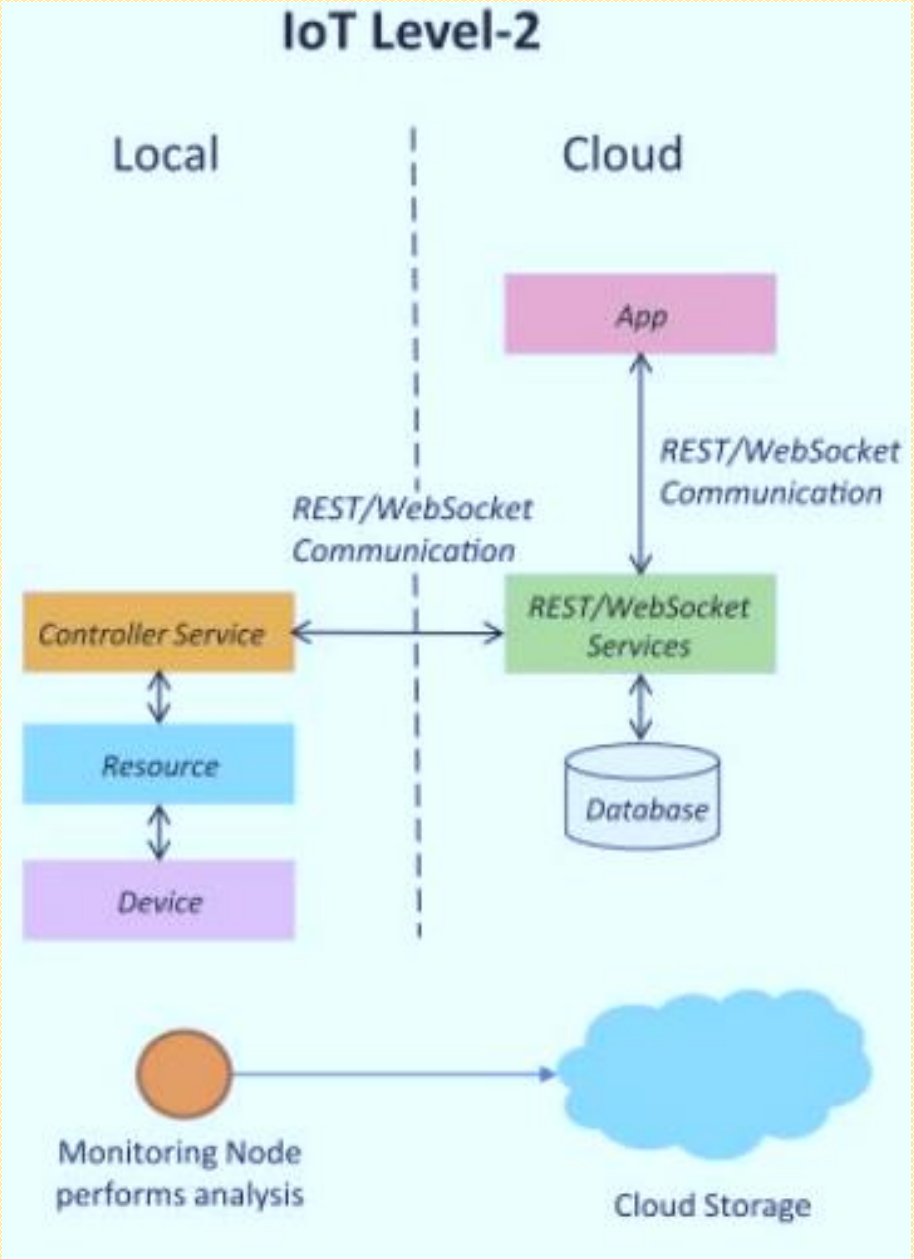
Component	Role
Single Node (e.g., Arduino or Raspberry Pi)	Collects soil moisture data and controls irrigation
Sensors (e.g. Moisture Sensors)	Measure the moisture content in soil
Actuator (e.g., solenoid valve)	Turns irrigation system ON/OFF based on moisture level
Cloud Database	Stores historical sensor data
Cloud REST API	Used for uploading/downloading moisture readings
Cloud-based Application	Provides visualization and analytics (moisture trends, decisions)



IOT LEVELS: IoT Level 2

How It Works

1. Sensors read soil moisture levels.
2. The controller service monitors the readings:
   - If moisture < threshold → sends command to actuator to irrigate.
3. Sensor data is also sent to the cloud.
4. Cloud REST API stores data in cloud database.
5. A web/mobile application (hosted in the cloud) provides:
   - Helps in making data Visualization of moisture trends
   - Dashboard to track irrigation history
6. Helps in making data-driven irrigation decisions.



IOT LEVELS: IoT Level-3

What is a Level-3 IoT System?

- A Level-3 IoT system is designed with a single node (device) that performs sensing and/or actuation, but data storage, analysis, and application hosting are entirely cloud-based.
- This level is suited for applications where:
  - Large volumes of data are generated
  - Computational analysis is intensive
  - Centralized processing is preferred

Why Use Level-3 IoT?

- When real-time monitoring and complex analysis are needed.
- For applications where centralized cloud access is critical.
- For scalable and globally accessible systems.

Key Features of Level-3 IoT System:

- **Single-node sensing/actuation:** Performs local control and sends data to cloud.
- **Cloud data storage:** Suited for large, historical datasets.
- **Cloud data analysis**: enables scalable, real-time processing, storage, and intelligent insights from sensor data using cloud platforms and services.
- **Cloud-based application:** Offers visualization and user-friendly dashboards.
- **REST APIs**: Enable remote communication with the cloud system.

IOT LEVELS: IoT Level-3

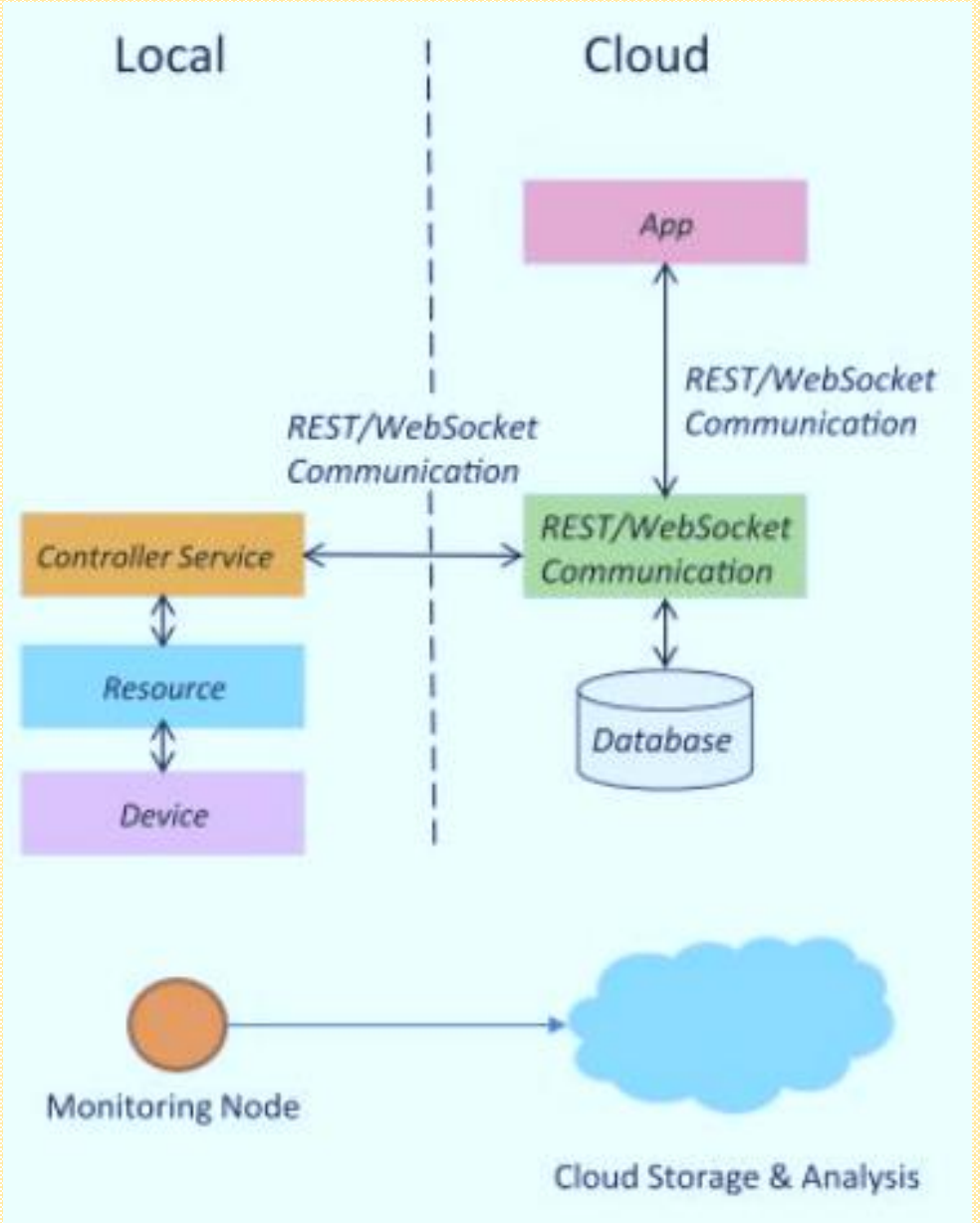
Example Package Handling System

Category	Component	Function/Role
IoT Device	Microcontroller/SBC (e.g., ESP32, Raspberry Pi)	▪ Reads sensor data and handles network communication
Sensors	Accelerometer (e.g., MPU6050)	▪ Measures vibration and impact forces during transit
	Gyroscope (e.g., MPU6050)	▪ Detects angular motion and orientation changes
Communication	WebSocket Client	▪ Sends data from device to cloud in real-time
	Wi-Fi Module (built-in or external)	▪ Enables Internet connectivity for the device
Cloud Platform	WebSocket Server (e.g., Node.js, AWS IoT Core)	▪ Receives real-time data from the device
	Cloud Database (e.g., Firebase, AWS DynamoDB)	▪ Stores vibration data and timestamps
Application	Web Dashboard (ReactJS / Angular)	▪ Displays vibration events, graphs, and package history
	Alert System (via Email/SMS/API)	▪ Notifies stakeholders if vibration exceeds safe threshold
Analytics	Rule-based or ML Engine	▪ Analyzes patterns in vibration data to detect potential damage or mishandling
Power Supply	Battery Pack / USB Power	▪ Powers the IoT device during shipment

IOT LEVELS: IoT Level-3

How It Works: Package Handling System

- **Sensors Used**: Accelerometer & Gyroscope on the package
- **Monitoring**: Real-time tracking of vibration levels during shipment
- **Data Flow**:
  - Sensor data → Sent in real-time to the cloud via WebSocket
  - Stored in cloud storage
  - Visualized in a cloud-based application
- **Cloud Services**:
  - WebSocket: Real-time communication (faster than REST)
  - Cloud Database: Store vibration levels
  - Cloud App: Visualize and analyze data, trigger alerts

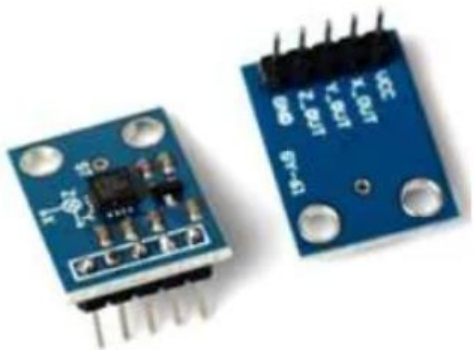


IOT LEVELS: IoT Level-3

Sensors used

Accelrometer

sense movement or vibrations



Gyroscope

Gives orientation info

IOT LEVELS: IoT Level-4

What is a Level-4 IoT System?

- The Level-4 IoT system is designed for complex, large-scale, and computation-intensive applications that require multiple data-collecting nodes and both local and cloud-based processing. This level supports distributed data collection and centralized cloud analytics, enabling robust and scalable IoT solutions.

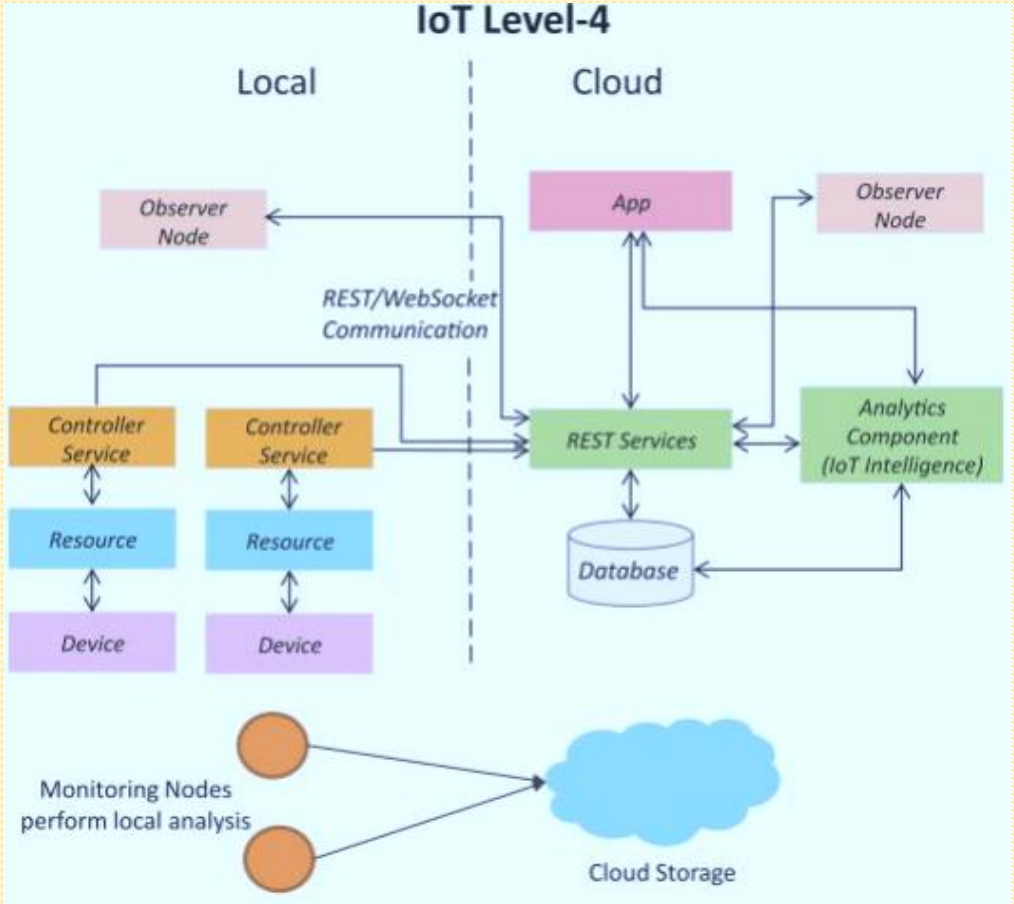
Use Cases for Level-4 IoT Systems

- **Smart City Applications** (e.g., traffic noise, pollution tracking)
- **Environmental Monitoring** (air quality, water levels, forest fire detection)
- **Distributed Industrial Systems**
- **Precision Agriculture with Distributed Sensor Clusters**
- **Large-Scale Surveillance Systems**

IOT LEVELS: IoT Level-4

Characteristics of Level-4 System

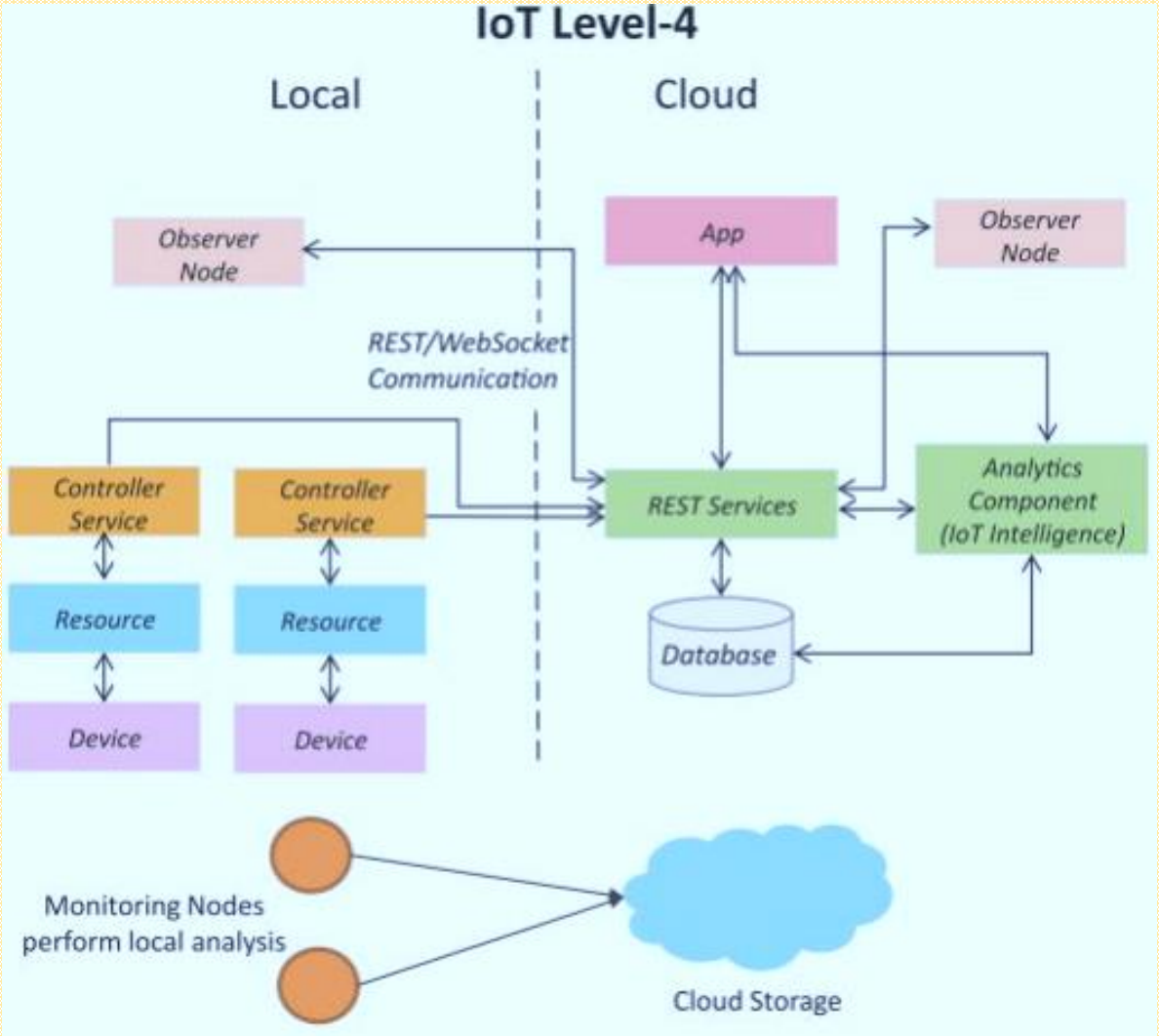
Feature	Description
Nodes	Multiple IoT nodes deployed across different locations
Local Processing	Each node can process data locally using built-in logic or controller services
Cloud Integration	All nodes send their data to the cloud for further analysis and storage
Application Hosting	The main application (UI and logic) resides in the cloud
Scalability	Highly scalable due to the distributed nature of data collection and centralized cloud handling
Data Volume	Suitable for applications involving large amounts of data
Computational Load	The system supports computationally intensive analysis and aggregation in the cloud
Control Scope	Each node can control devices locally; observer nodes only observe and analyze data



IOT LEVELS: IoT Level-4

Architecture Overview

- **Multiple Nodes**: The system uses several independent IoT nodes, each equipped with sensors and controllers. These nodes operate autonomously and perform local analysis on the collected data.
- **Cloud Storage**: The processed or raw data from each node is sent to the cloud, where it is stored in a centralized database.
- **Cloud-Based Application**: A web or cloud-based application aggregates, analyzes, and visualizes data collected from all the nodes.
- **Observer Nodes**: These are cloud-based nodes that subscribe to data feeds from IoT devices. While they process information, they do not control any physical devices.



IOT LEVELS: IoT Level-4

Example: Level-4 IoT system for environmental noise monitoring:

- **System Setup:**
  - Nodes are placed in multiple locations (e.g., street corners, construction sites).
  - Each node consists of:
    - A microcontroller or single-board computer (e.g., Raspberry Pi)
    - A sound sensor (microphone module or noise-level sensor)
    - A controller service to collect and pre-process noise data
- **Local Functionality:**
  - Each node monitors sound levels in real time.
  - It processes the data locally (e.g., to determine decibel levels or detect patterns).
- **Cloud Functionality:**
  - Each node sends data to the cloud.
  - Data is stored in a cloud database and analyzed for trends or thresholds.
  - A cloud-based web application is used to visualize aggregated noise data, view heatmaps, or generate alerts.
- **Observer Nodes:**
  - These are analytics or decision-making systems that subscribe to cloud data for further processing (e.g., to optimize urban planning), but do not control any field device.

IOT LEVELS: IoT Level-5

What is a Level-5 IoT System?

- A Level-5 IoT system consists of multiple end nodes and a coordinator node.
- The end nodes perform sensing and/or actuation, and send data to the coordinator node, which aggregates the data and transmits it to the cloud for storage, analysis, and visualization.
- The application is cloud-based, and the analysis is typically computationally intensive, suited for wireless sensor networks (WSNs) with large-scale, distributed data.

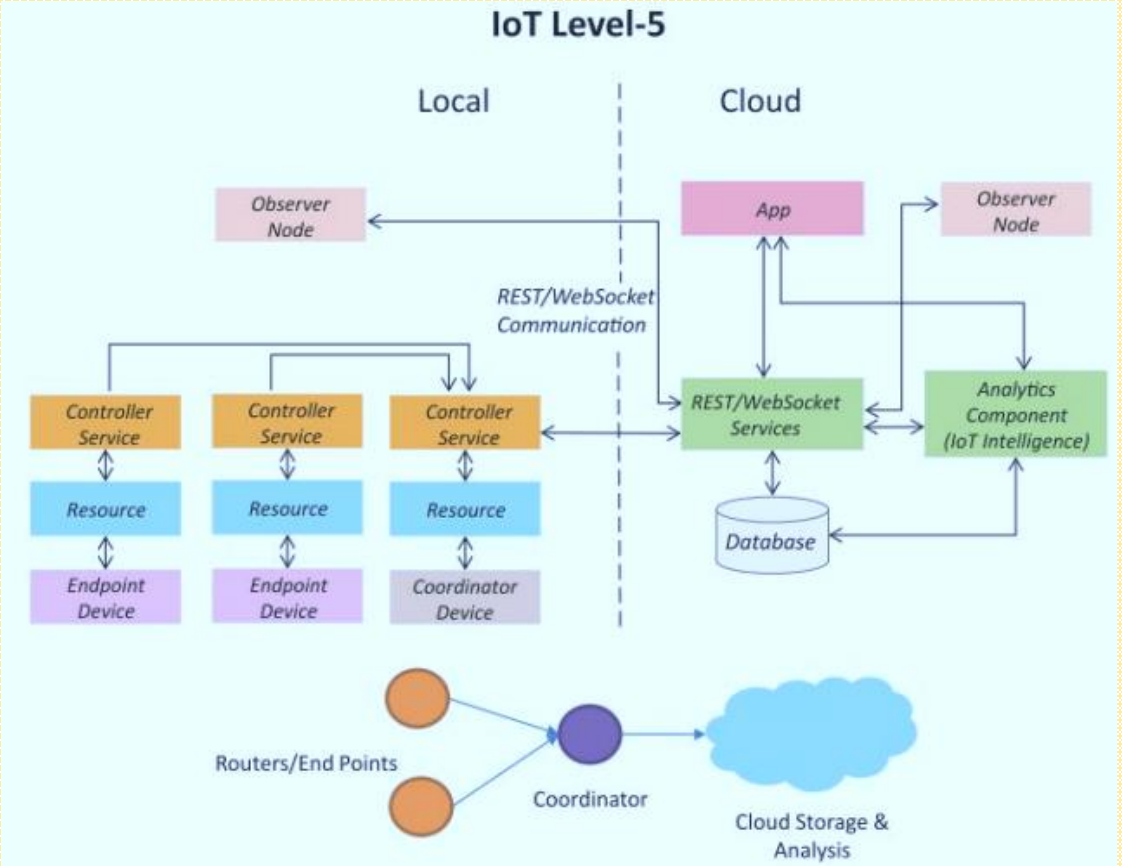
Key Features

- **High scalability** (handles many sensors).
- **Distributed sensing, centralized analysis**.
- **Real-time alerts** for fire risk based on sensor thresholds.
- Suitable for **large-scale environmental monitoring** applications.

IOT LEVELS: IoT Level-5

Example: Forest Fire Detection System (Components Used)

Component Type	Component Name/Description
End Nodes	Microcontrollers with sensor interface (e.g., Arduino, ESP32)
Sensors	Temperature sensor (e.g., DHT11), Humidity sensor, CO_2 sensor (e.g., MG-811)
Coordinator Node	Raspberry Pi / Edge Gateway Device
Communication Protocols	LoRa, Zigbee (for node-to-coordinator), Wi-Fi/Ethernet (coordinator to cloud)
Cloud Platform	AWS IoT, Google Cloud IoT, or Azure IoT Hub
Cloud Database	MongoDB Atlas, Firebase Realtime DB, or Amazon DynamoDB
Cloud App (Dashboard)	Web-based application using tools like Node-RED, Django, or Grafana



IOT LEVELS: IoT Level-5

Example: Forest Fire Detection System (Components Used)

How it Works

1. End Nodes:

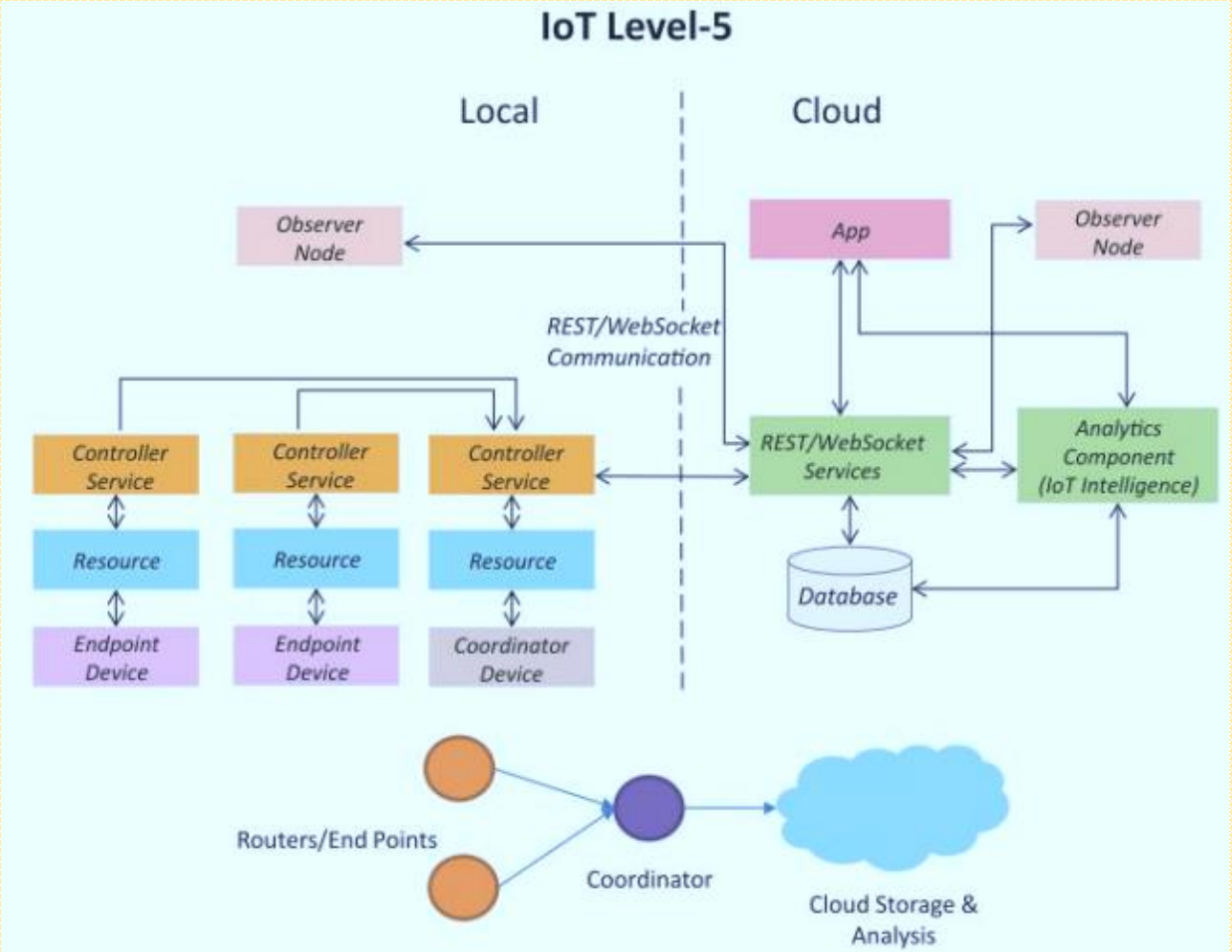
- Deployed in various forest areas.
- Continuously collect data on temperature, humidity, and CO_2 levels.

2. Coordinator Node:

- Gathers sensor data from all end nodes.
- Provides Internet gateway connectivity.

3. Cloud System:

- Stores all data in a cloud database.
- Performs data aggregation and predictive analysis (e.g., fire risk prediction using AI/ML models).
- Displays visualizations through a web dashboard.



IOT LEVELS: IoT Level-6

What is a Level-6 IoT System?

- Multiple independent end nodes (devices) are deployed.
- These end nodes sense (measure) or actuate (control) something.
- They send the data to a cloud platform.
- The cloud stores the data and runs analytics on it.
- Results are shown to users via a cloud-based application (like a dashboard).
- A centralized controller in the cloud keeps track of the status of all end nodes and can send control commands back to them, enabling remote monitoring and control in real-time.

Key Features

- **Real-time monitoring**.
- **Cloud-centric storage and processing**.
- **Remote control** from a central place.
- **Scalable** — you can add more nodes anywhere.

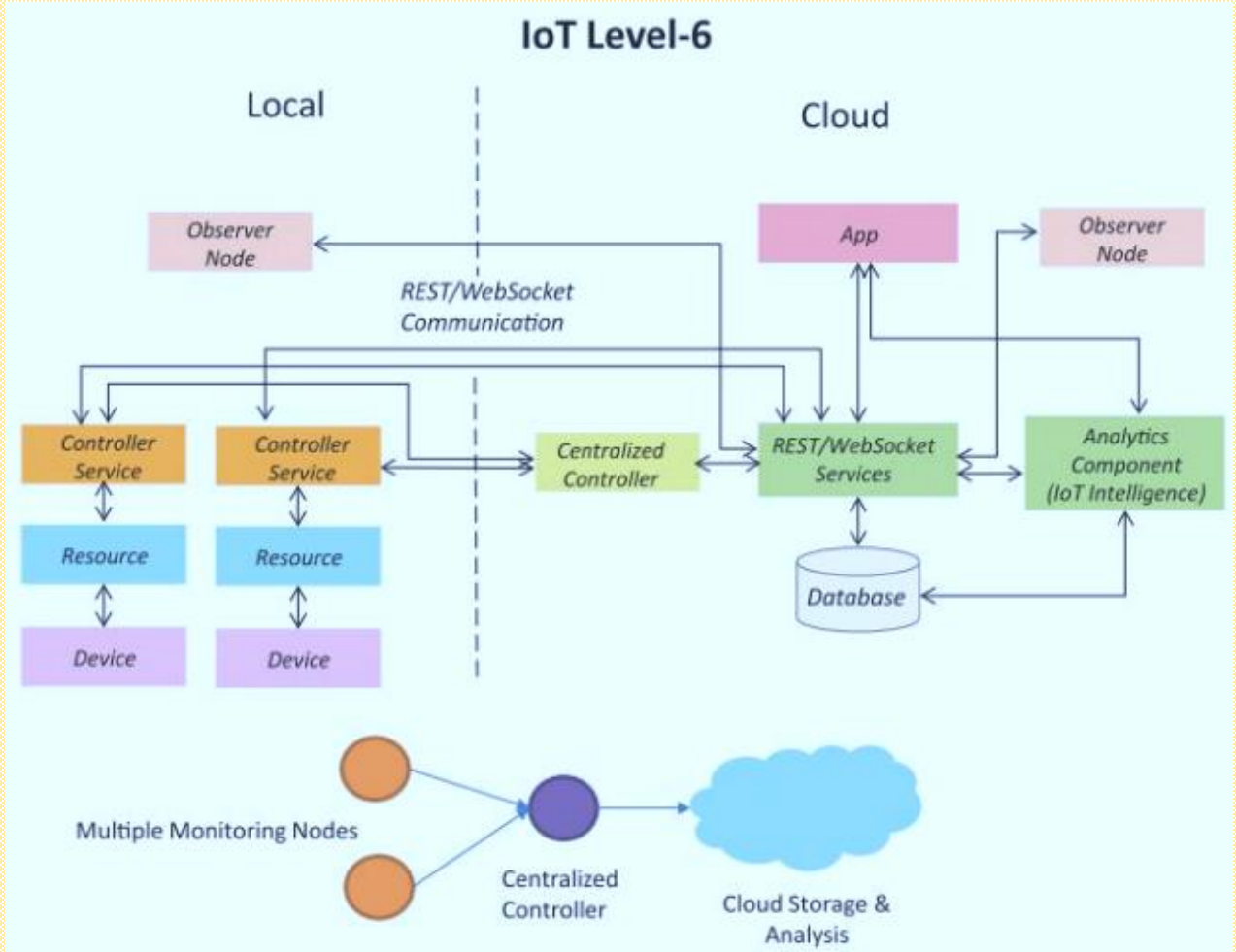
IOT LEVELS: IoT Level-6

Example: Weather Monitoring System

A weather monitoring network with many sensor nodes in different locations, sending real-time temperature, humidity, and pressure data to the cloud, which analyzes and visualizes it for users, while also allowing remote control if needed.

How it Works

- **Sensor Nodes**: Measure data (temperature, humidity, pressure).
- **Cloud**: Stores and analyzes data.
- **Dashboard/App**: Visualizes results for the user.
- **Central Controller**: Monitors all nodes and sends back commands.
- **Actuator Nodes**: Perform actions based on commands (e.g., open a vent).



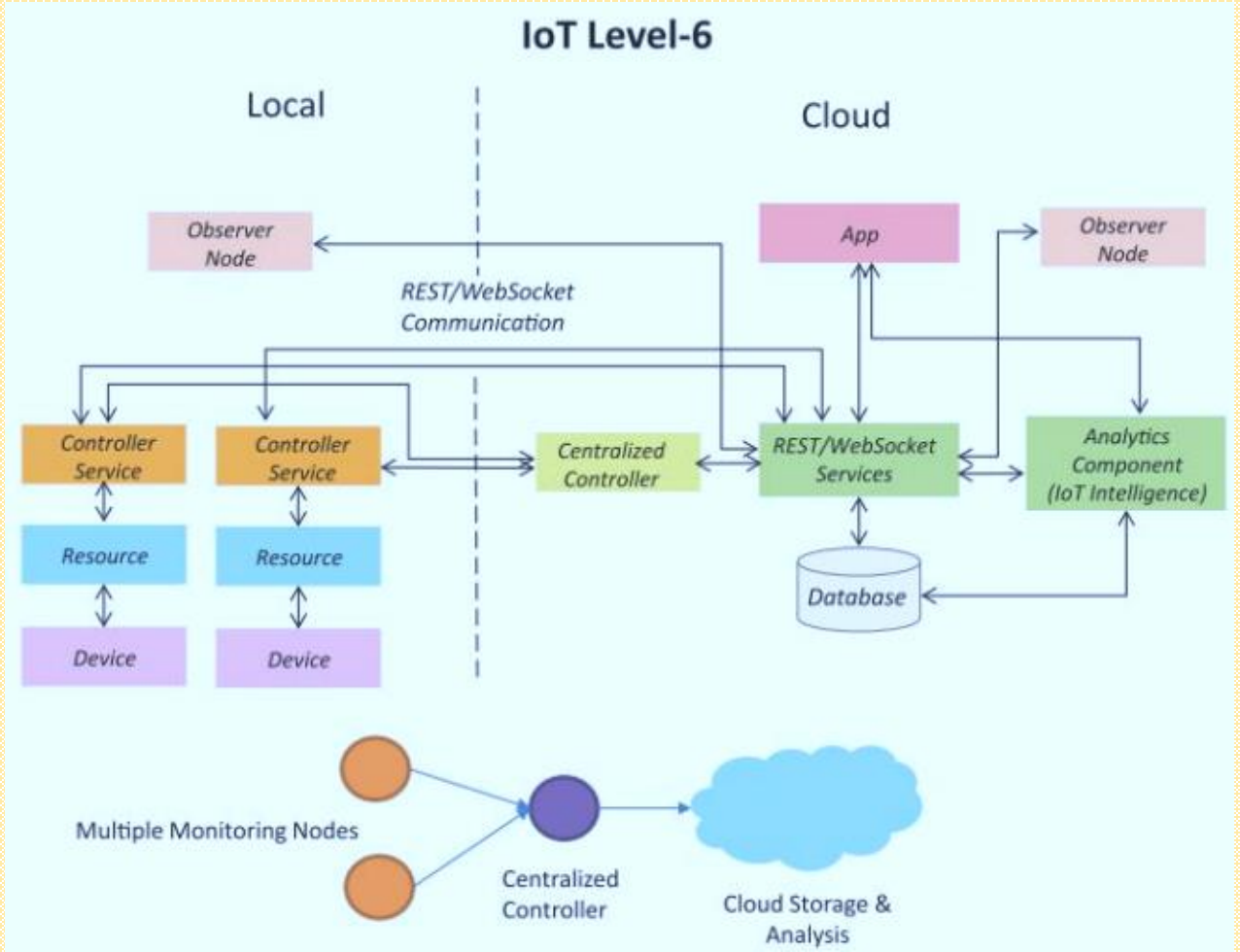
IOT LEVELS: IoT Level-6

Example: Weather Monitoring System

How it Works

Imagine a **city-wide weather monitoring system**:

- **End nodes**: Weather stations in different parts of the city.
- Each station has sensors for temperature, humidity, and pressure.
- These stations send live data to the cloud using WebSocket (real-time).
- In the cloud:
  - Data is stored in a database.
  - Analytics software aggregates data, finds trends, and makes predictions (like tomorrow's weather).
- Users see the results in a web/mobile app.
- If needed, the central controller can send commands (like activating a fan or heater at a remote node).



COMPARISON OF IOT LEVELS (IoT Level 1 – IoT Level 6)

Aspect	IoT Level 1	IoT Level 2	IoT Level 3	IoT Level 4	IoT Level 5	IoT Level 6
System Nodes	Single node	Single node	Single node	Multiple independent nodes	Multiple end nodes + 1 coordinator	Multiple end nodes + 1 Central Controller
Data Storage	Local (on-device)	Cloud	Cloud	Cloud	Cloud	Cloud
Data Analysis	Local	Local	Cloud	Cloud (from multiple nodes)	Cloud (from coordinator)	Cloud (central analytics + control)
Application	Local (on the same device)	Cloud-based	Cloud-based	Cloud-based	Cloud-based	Cloud-based
Communication	REST APIs	REST APIs	WebSocket / REST	REST / WebSocket	REST/WebSocket	WebSocket (real-time)
Control	Local automatic or manual control	Local control + cloud monitoring	Cloud-based control + real-time alerts	No control — observer nodes only	Centralized control via coordinator node	Centralized control in cloud, commands sent to actuators
Example	Home automation (light control)	Smart irrigation system	Package handling system	Noise monitoring (city-wide)	Forest fire detection system	City-wide weather monitoring
Components	Raspberry Pi, LDR, Relay switch	Soil sensors, controller, cloud REST API	Accelerometer, gyroscope, WebSocket API	Sound sensors, multiple nodes	Temp/Humidity/CO_2 sensors, coordinator node	Temp/Humidity/Pressure sensors, actuator nodes, central controller
Analysis Type	Simple status monitoring	Threshold-based actions	Event-triggered, threshold-based	Aggregation, visualization	Aggregation, pattern recognition	Real-time, predictive + control
Scalability	Low	Low–Moderate	Moderate	High	High	Very High
Use Case Type	Low-cost, low-complexity	Low-cost + Cloud storage	Cloud-driven alert system	City-scale sensing + visualization	Environmental monitoring + decision-making	Scalable, real-time monitoring + remote control