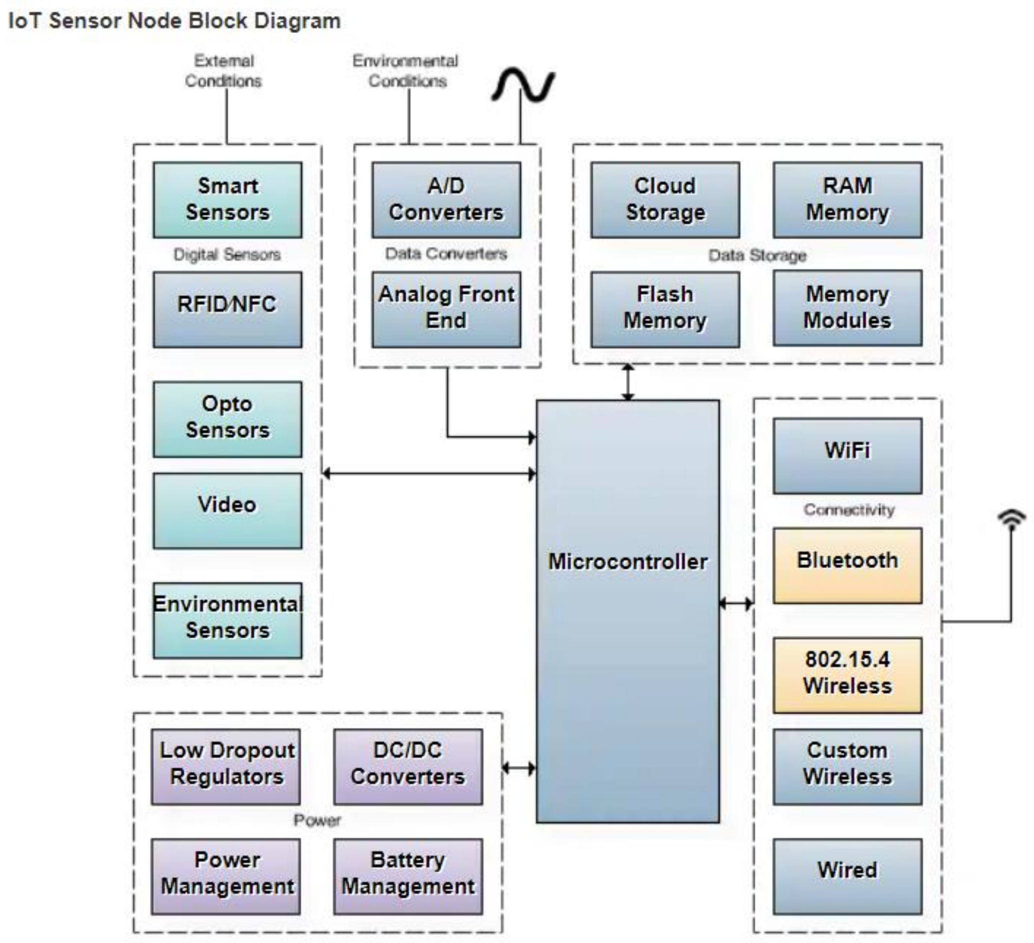
IOT MOCK TEST 1

Ans 1.

IoT Sensor Node Block Diagram

The Internet of Things (IoT) is about interconnecting embedded systems, bringing together two evolving technologies: wireless connectivity and sensors. These connected embedded systems are independent microcontroller-based computers that use sensors to collect data. These IoT systems are networked together usually by a wireless protocol such as WiFi, Bluetooth, 802.11.4, or a custom communication system. The networking protocol is selected based on the distribution of nodes and the amount of data to be collected.

This data is sent over the network to the main hub or computer. This main computer collects and analyzes the data, storing it in memory and even making system decisions based on the results of the analysis.



Ans 2.

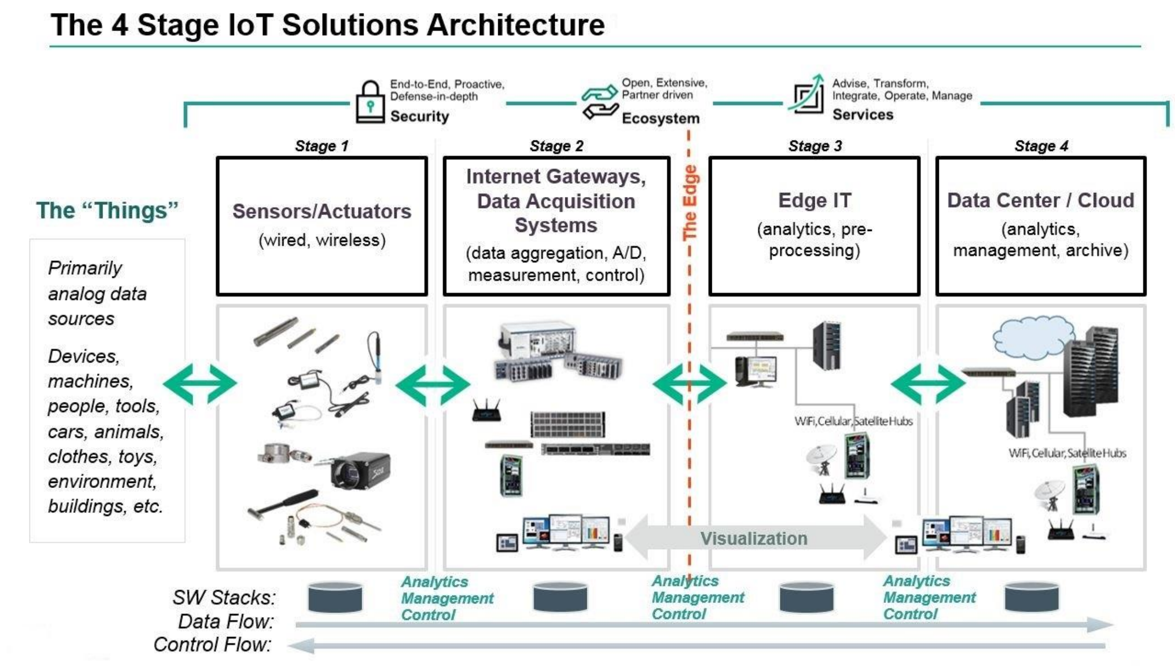
Basically, there are 4stages architecture of an IOT system:

Stage 1 of an IoT architecture consists of your networked things, typically wireless sensors and actuators.

Stage 2 includes sensor data aggregation systems and analog-to-digital data conversion

In **Stage 3**, edge IT systems perform preprocessing of the data before it moves on to the data center or cloud.

Stage 4, the data is analyzed, managed, and stored on traditional back-end data center systems.


The 4 Stage IoT Solutions Architecture
End-to-End, Proactive, Defense-in-depth
Security
Open, Extensive, Partner driven
Ecosystem
Advise, Transform, Integrate, Operate, Manage
Services
Stage 1
Stage 2
Stage 3
Stage 4
The "Things"
Primarily analog data sources
Devices, machines, people, tools, cars, animals, clothes, toys, environment, buildings, etc.
Sensors/Actuators
(wired, wireless)
Internet Gateways, Data Acquisition Systems
(data aggregation, A/D, measurement, control)
The Edge
Edge IT
(analytics, pre-processing)
Data Center / Cloud
(analytics, management, archive)
WiFi, Cellular, Satellite Hubs
WiFi, Cellular, Satellite Hubs
Visualization
SW Stacks:
Data Flow:
Control Flow:
Analytics
Management
Control
Analytics
Management
Control
Analytics
Management
Control

Stage 1. Sensors/actuators

Sensors collect data from the environment or object under measurement and turn it into useful data. Think of the specialized structures in your cell phone that detect the directional pull of gravity—and the phone's relative position to the "thing" we call the earth—and convert it into data that your phone can use to orient the device. Actuators can also intervene to change the physical conditions that generate the data. An actuator might, for example, shut off a power supply, adjust an air flow valve, or move a robotic gripper in an assembly process.

Stage 2. The Internet gateway

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

Stage 3. Edge IT

Once IoT data has been digitized and aggregated, it's ready to cross into the realm of IT. However, the data may require further processing before it enters the data center. This is where edge IT systems, which perform more analysis, come into play. Edge IT processing systems may be located in remote offices or other edge locations, but generally these sit in the facility or location where the sensors reside closer to the sensors, such as in a wiring closet.

Because IoT data can easily eat up network bandwidth and swamp your data center resources, it's best to have systems at the edge capable of performing analytics as a way to lessen the burden on core IT infrastructure. If you just had one large data pipe going to the data center, you'd need enormous capacity. You'd also face security concerns, storage issues, and delays processing the data. With a staged approach, you can preprocess the data, generate meaningful results, and pass only those on. For example, rather than passing on raw vibration data for the pumps, you could aggregate and convert the data, analyze it, and send only projections as to when each device will fail or need service.

Stage 4. The data center and cloud

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

Ans. 3

Logical Design of IoT

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

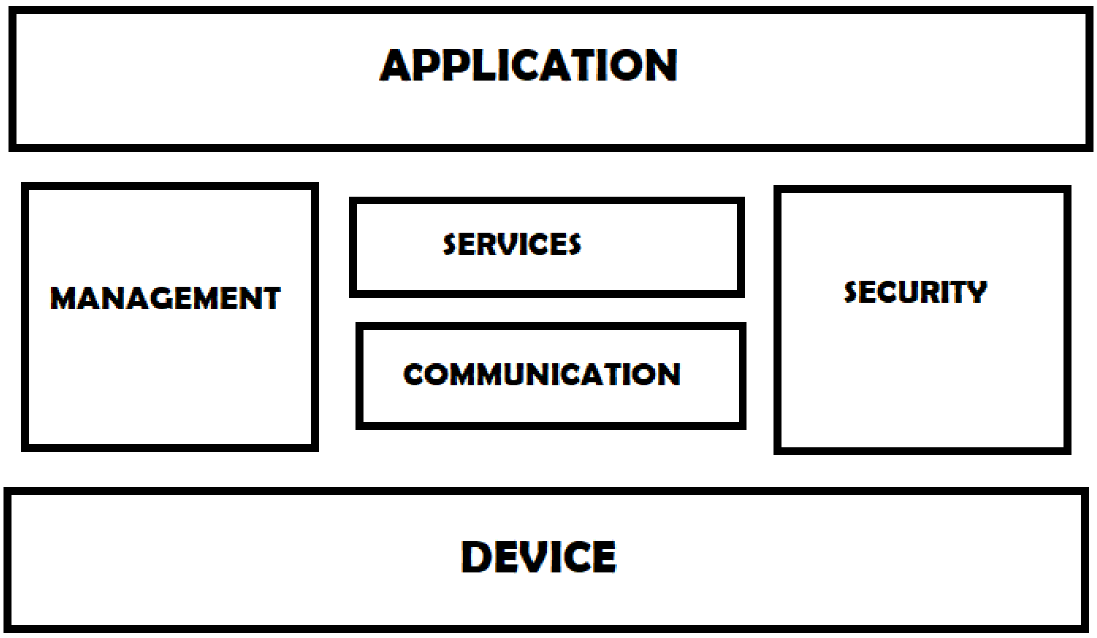
- IoT Functional Blocks
- IoT Communication Models
- IoT Communication APIs

1. IoT Functional Blocks

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

functional blocks are:

Device: An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
Communication: Handles the communication for the IoT system.
Services: services for device monitoring, device control service, data publishing services and services for device discovery.
Management: this blocks provides various functions to govern the IoT system.
Security: this block secures the IoT system and by providing functions such as authentication , authorization, message and content integrity, and data security.
Application: This is an interface that the users can use to control and monitor various aspects of the IoT system. Application also allow users to view the system status and view or analyze the processed data.



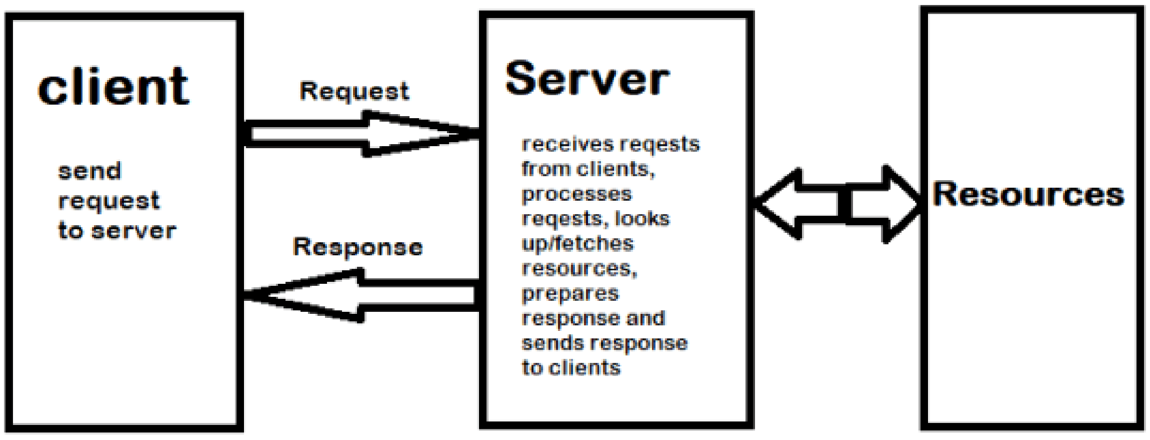
2. IoT Communication Models

Request-Response Model

Request-response model is communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves resource representation, prepares the response, and then sends the response to the client. Request-response is a stateless communication model and each request-response pair is independent of others.

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

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.


client
send request to server
Request
Response
Server
receives reqests from clients, processes reqests, looks up/fetches resources, prepares response and sends response to clients
Resources

Request-Response Communication Model

3. IoT Communication APIs

Generally we used Two APIs For IoT Communication. These IoT Communication APIs are:

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

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 systems's resources and how resource states are addressed and transferred. REST APIs that follow the request response communication model, the rest architectural constraint apply to the components, connector and data elements, within a distributed hypermedia system. The rest architectural constraint are as follows:

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

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

Cache-able – Cache constraints requires that the data within a response to a request be implicitly or explicitly leveled as cache-able or non cache-able. If a response is cache-able, then a client cache is given the right to reuse that repsonse data for later, equivalent requests. caching can partially or completely eliminate some instructions and improve efficiency and scalability.

Layered system – layered system constraints, constrains the behavior of components such that each component cannot see beyond the immediate layer with they are interacting. For example, the client cannot tell whether it is connected directly to the end server or two an intermediaryalong the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.

Uniform interface – uniform interface constraints requires that the method of communication between client and server must be uniform. Resources are identified in the requests (by URIsin web based systems) and are themselves is separate from the representations of the resources data returned to the client. When a client holds a representation of resources it has all the information required to update or delete the resource you (provided the client has required permissions). Each message includes enough information to describe how to process the message.

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

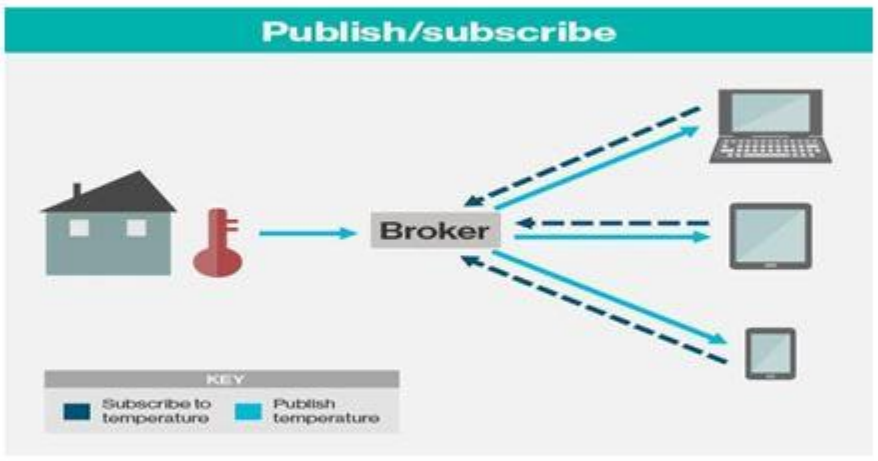
Ans.4

Application layer refers to OSI Level 5, 6 and 7. It is application layer in the TCP-IP model. In IOT architecture, this layer lies above the service discovery layer. It is highest layer in the architecture extending from the client ends. It is the interface between the end devices and the network. This layer is implemented through a dedicated application at the device end. Like for a computer, application layer is implemented by the browser. It is the browser which implements application layer protocols like HTTP, HTTPS, SMTP and FTP. Same way, there are application layer protocols specified in context to IOT as well.

This layer is responsible for data formatting and presentation. The application layer in the Internet is typically based on HTTP protocol. However, HTTP is not suitable in resource constrained environment because it is extremely heavyweight and thus incurs a large parsing overhead. So, there are many alternate protocols that have been developed for IOT environments. Some of the popular IOT application layer protocols are as follow –

- MQTT
- Websocket
- CoAP
- HTTP

MQTT – Message Queuing Telemetry Transport is a lightweight messaging protocol. It uses publish-subscribe communication way and that's why it is used for M2M (machine to machine) communication. It is based on TCP-IP protocol and is designed to operate in limited bandwidth. In the protocol terminology, the limited network bandwidth is referred as 'small code footprint'. However, the exact meaning of limited network bandwidth is not clear in the specification.
This protocol has been specially designed for sensor networks and wireless sensor networks. MQTT allows devices to send or publish data information on a given topic to a server. There is a MQTT broker (Broker- Mosquitto) in between publisher and subscriber. The broker then transfers the information to the clients that are previously subscribed.
So, MQTT fares well even when connection between the broker and the publisher or broker and the client is broken due to limited network bandwidth. This ability to deal with delay or latency in network makes this protocol quite suitable for wireless networks.



CoAP – Constrained Application Protocol is specifically designed for constrained (limited) Hardware. The hardware that doesn't support HTTP or TCP/IP can use CoAP Protocol. So, basically the designers of this protocol taking inspiration by the HTTP had designed the CoAP protocol using UDP and IP protocol. It is a lightweight protocol that needs low power IOT application like for communication between battery powered IOT devices. Like HTTP, it also follows client-server model. The clients can GET, PUT, DELETE or POST informational resources over the network.

CoAP makes use of two message types – requests and responses. The messages contain a base header followed by message body whose length is specified by the datagram. The messages or data packets are small in size, so that they can be communicated among constraint devices without data losses. The messages can be confirmable or non-confirmable. For confirmable messages, the client need to respond with an acknowledgement after receiving the data packet. The request messages can contain query strings to implement additional functionalities like search, sort or paging.

For security, the protocol uses Datagram Transport Layer Security (DTLS) over UDP. In HTTP, it is very complicated to know the new state on a variable. in HTTP, client has to do polling again and again which means client has to ask every time every second to see if there is any new state of variable it is observing. In CoAP, the CoAP services try to solve the problem by creating an observe flag, so if the original device sends observe flag with GET command, every time the sever or the other devices see that there is a change in the state of the variable then the server or the devices will send the push notification to the original device who is actually finding the observer flag.

The protocol also provides an additional mechanism for resource discovery by the client devices. The server has to provide a list of resources along with meta data which can be accessed as link or application media type. By navigating through the list, a client can find available resources (information or data) and discover their media types.
The sensor nodes in the network themselves act as server instead of clients. The sensors are directly routable and the data communication is one to one between sensor and the client devices. For implementation of this protocols, the sensors must be able to receive data packets and able to respond them.

Websocket – WebSocket JavaScript interface defines a full-duplex single socket connection over which messages can be sent between client and server. The standard simplifies the complexity around bi-directional web communication and connection management.
Reactive Streams – Reactive Streams is a protocol stack for asynchronous stream processing with non-blocking back pressure. It is a specification which may be implemented on Java or JavaScript. It is used for handling live data streams whose volume is not known.

HTTP/2 – HTTP version 2 will be the next version of the HTTP protocol. It will be using header field compression to reduce latency and will allow multiple concurrent exchanges on the same connection.
JavaScript IOT – This is not any standard or protocol. It refers to the use of JavaScript and particularly Node.js (a server side JavaScript) with IOT boards for various applications. Some of the JavaScript based IOT projects include NodeMCU, Jerryscript, Socket.IO, Ruff, NodeRed, KinomaJS, CHIRIMEN, Mosca, Nodebots, heimcontrol.js, Resin, UPM, i2C, node-http2, onoff, node-serialport, mdns, IoT.js, Favor, Serverless, noduino, duino, Pijs.io, etc.
In the next tutorial, IOT Cloud and Services will be discussed.

By- Udit Gupta

2047262

2MCA