Unit # 3 Architecture and Reference Model

IOT Reference model

IOT reference model contains broadly seven layers. Researchers have a different opinion about a number of reference layers it varies from 3 to 7.

Physical Devices and Controllers

This layer has physical devices like sensors for sensing and assembling information about the surroundings which senses some physical parameters or identifies other smart objects in the environment.

Connectivity

In connectivity layer, communications, and processing executed by existing networks. This layer includes connectors like RJ45, ModBus, USB or wireless connectivity.

Computing

The objective of computing layer is for mainly analyzing of data and data formation. Data prepared by encoding, decoding and summarizing for and transformation. This layer is also known as edge computing.

Data Accumulation

Data storage and data functionality are handled in this layer. This considers type of data, processing of data, higher level application data compatibility, combining of data and storage type. In the area of huge Data backend Hadoop, HBase, MongoDB, and Cassandra can be adding for data storage.

Data Abstraction

The data abstraction layer receives data sent by device stored at data accumulation layer need to send to an endpoint to use for the application. The collected data is in many different formats as collected from different sources; need to convert in the same format suitable to a higher-level application. Data indexed, normalized and providing appropriate authentication and authorization for security.

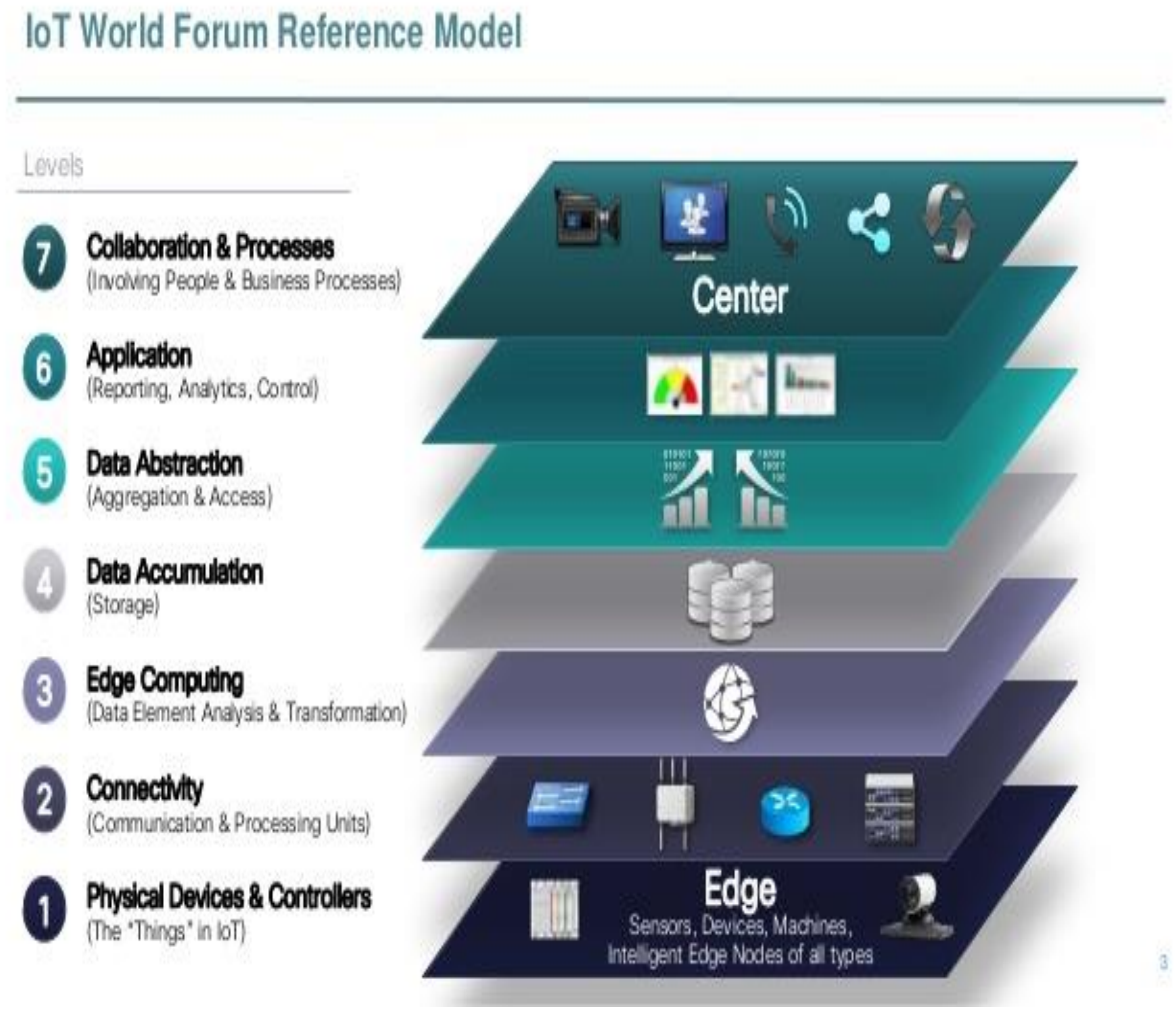
Application (Reporting, Analytics, Control)

The application layer main objective is information interpretation and Software for interactions with previous data abstraction level. Simple communications applications can be handled like Mobile Applications are based on device data, business, programming patterns, and software stacks, operating systems, mobility, application servers, hypervisors, multi-threading, multi-tenancy, etc

Unit # 3 Architecture and Reference Model

Collaboration and Processes (Involving people and business processes)

The information generated in IOT yields to accomplish in this layer by handover to end user and processes. They use applications and associated data for their specific requirements. Sixth layer Applications layer gives the right data, at the right time, so they can do the right thing. People must be able to communicate and collaborate, sometimes using the traditional Internet, to make the IoT useful.


IoT World Forum Reference Model
Levels
7 Collaboration & Processes (Involving People & Business Processes)
6 Application (Reporting, Analytics, Control)
5 Data Abstraction (Aggregation & Access)
4 Data Accumulation (Storage)
3 Edge Computing (Data Element Analysis & Transformation)
2 Connectivity (Communication & Processing Units)
1 Physical Devices & Controllers (The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines, Intelligent Edge Nodes of all types

Unit # 3 Architecture and Reference Model

IoT Architecture

What is IoT architecture?

Because of outstanding opportunities IoT promises, more organizations seek for the inclusion of its products in their business processes. However, when it comes to reality, this brilliant idea appears too complicated to be implemented — given the number of devices and conditions needed to make it work. In other words, the problem of establishing a reliable architecture of Internet of Things inevitably enters the stage.

Among all, to deal with the whole variety of factors affecting IoT architecture, it's easier and more effective to find a reliable provider of IoT solutions. This decision will significantly reduce the number of resources spent on the way. Though it's possible to comprehend the process of creating software, the practical application of its 4 stages contains too many nuances and aspects to be described in simple words. Because of that, use this guide for establishing a proper understanding of what's going on during IoT architecture — but consider referring to the specialist to make this process actually happen. This decision will facilitate getting the needed result and guarantee being a satisfied client of a software development company.

Before revealing the secrets and providing a clear structure of this initiative, it's important to understand what this concept actually means. In essence, IoT architecture is the system of numerous elements: sensors, protocols, actuators, cloud services, and layers. Given its complexity, there exist 4 stages of IoT architecture. Such a number is chosen to steadily include these various types of components into a sophisticated and unified network.

Unit # 3 Architecture and Reference Model

In addition, Internet of Things architecture layers are distinguished in order to track the consistency of the system. This should also be taken into consideration before the IoT architecture process start.

Basically, there are three IoT architecture layers:

1. The client side (IoT Device Layer)

2. Operators on the server side (IoT Getaway Layer)

3. A pathway for connecting clients and operators (IoT Platform Layer)

In fact, addressing the needs of all these layers is crucial on all the stages of IoT architecture. Being the basis of feasibility criterion, this consistency makes the result designed really work. In addition, the fundamental features of sustainable IoT architecture include functionality, scalability, availability, and maintainability. Without addressing these conditions, the result of IoT architecture is a failure.

Therefore, all the above-mentioned requirements are addressed in 4 stages of IoT architecture described here — on each separate stage and after completing the overall building process.

An Overview of the Main Stages in the IoT Architecture Diagram

In simple terms, the 4 Stage IoT architecture consists of

Unit # 3 Architecture and Reference Model

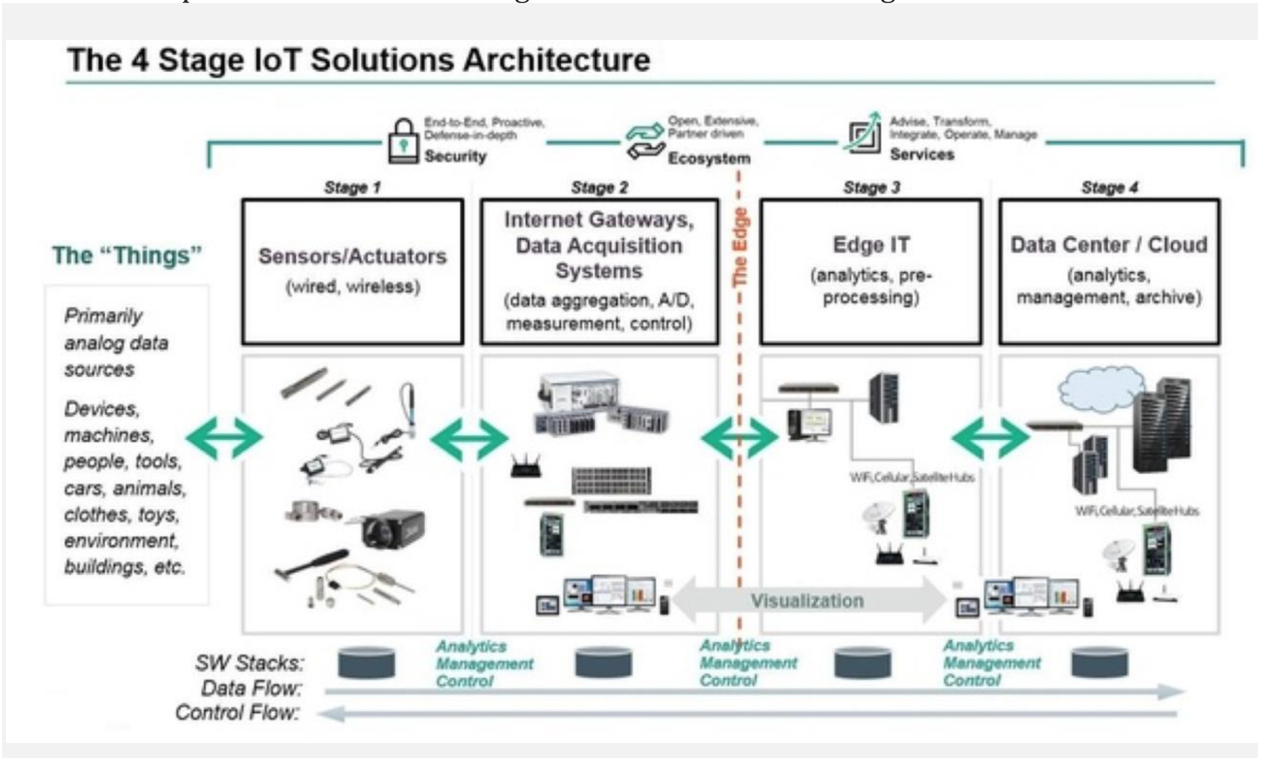
1. Sensors and actuators

2. Internet getaways and Data Acquisition Systems

3. Edge IT

4. Data center and cloud.

The detailed presentation of these stages can be found on the diagram below.


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"
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)
Primarily analog data sources
Devices, machines, people, tools, cars, animals, clothes, toys, environment, buildings, etc.
WiFi,Cellular,SatelliteHubs
WiFi,Cellular,SatelliteHubs
Visualization
SW Stacks:
Data Flow:
Control Flow:
Analytics Management Control
Analytics Management Control
Analytics Management Control

To get the proper understanding of the main actions and the importance of each stage in this process, refer to the detailed reviews presented below.

Stage 1. Networked things (wireless sensors and actuators)

Unit # 3 Architecture and Reference Model

The outstanding feature about sensors is their ability to convert the information obtained in the outer world into data for analysis. In other words, it's important to start with the inclusion of sensors in the 4 stages of an IoT architecture framework to get information in an appearance that can be actually processed.

For actuators, the process goes even further — these devices are able to intervene the physical reality. For example, they can switch off the light and adjust the temperature in a room.

Because of this, sensing and actuating stage covers and adjusts everything needed in the physical world to gain the necessary insights for further analysis.

Stage 2. Sensor data aggregation systems and analog-to-digital data conversion

Even though this stage of IoT architecture still means working in a close proximity with sensors and actuators, Internet getaways and data acquisition systems (DAS) appear here too. Specifically, the later connect to the sensor network and aggregate output, while Internet getaways work through Wi-Fi, wired LANs and perform further processing.

The vital importance of this stage is to process the enormous amount of information collected on the previous stage and squeeze it to the optimal size for further analysis. Besides, the necessary conversion in terms of timing and structure happens here.

In short, Stage 2 makes data both digitalized and aggregated.

Stage 3. The appearance of edge IT systems

During this moment among the stages of IoT architecture, the prepared data is transferred to the IT world. In particular, edge IT systems perform enhanced analytics and pre-processing here. For example, it refers to machine learning and visualization technologies. At the same time, some additional processing may happen here, prior to the stage of entering the data center.

Likewise, Stage 3 is closely linked to the previous phases in the building of an architecture of IoT. Because of this, the location of edge IT systems is close to the one where sensors and actuators are situated, creating a wiring closet. At the same time, the residing in remote offices is also possible.

Stage 4. Analysis, management, and storage of data

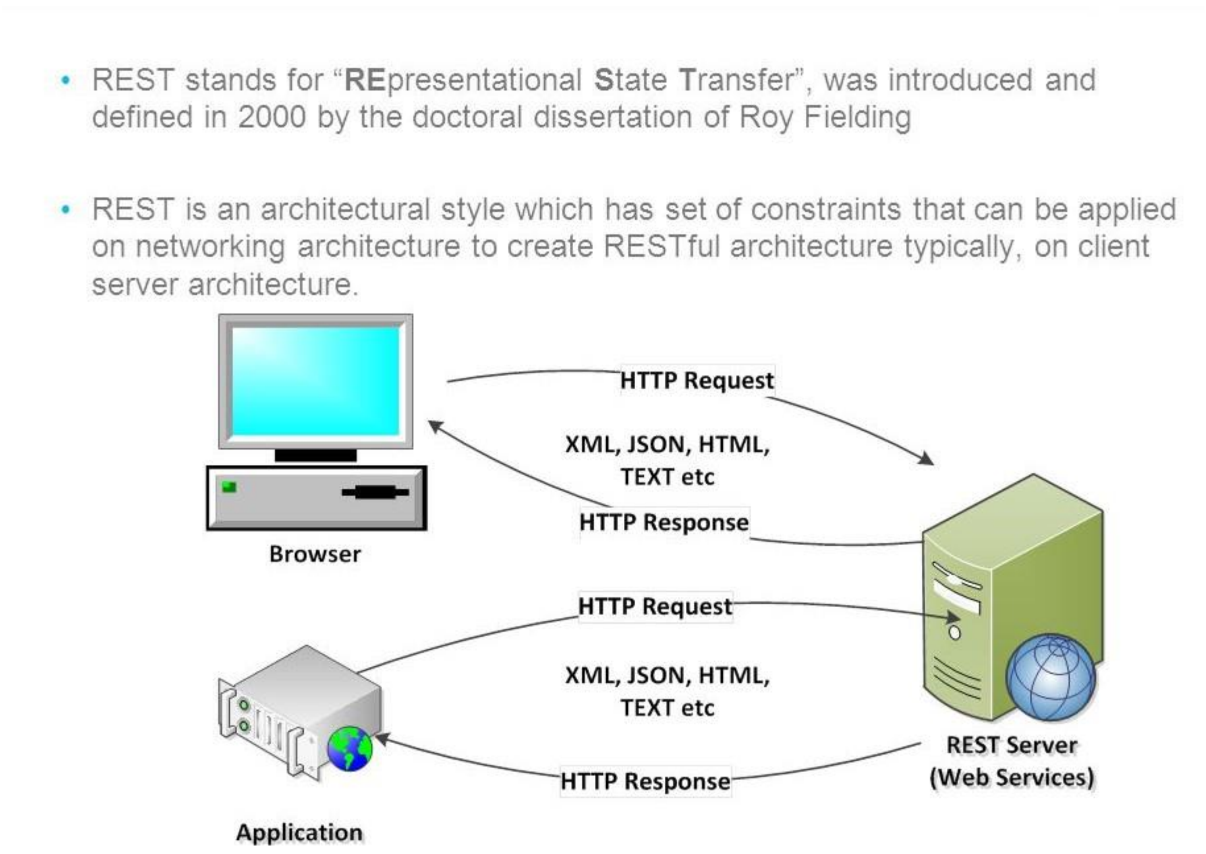
The main processes on the last stage of IoT architecture happen in data center or cloud. Precisely, it enables in-depth processing, along with a follow-up revision for feedback. Here, the skills of both IT and OT (operational technology) professionals are needed. In other words, the phase already includes the analytical skills of the highest rank, both in digital and human worlds. Therefore, the data from other sources may be included here to ensure an in-depth analysis.

After meeting all the quality standards and requirements, the information is brought back to the physical world — but in a processed and precisely analyzed appearance already.

Stage 5 of IoT Architecture?

In fact, there is an option to extend the process of building a sustainable IoT architecture by introducing an extra stage in it. It refers to initiating a user's control over the structure — if only your result doesn't include full automation, of course. The main tasks here are visualization and management. After including Stage 5, the system turns into a circle where a user sends commands to sensors/actuators (Stage 1) to perform some actions.

REST Architecture



- REST stands for "**RE**presentational **S**tate **T**ransfer", was introduced and defined in 2000 by the doctoral dissertation of Roy Fielding
- REST is an architectural style which has set of constraints that can be applied on networking architecture to create RESTful architecture typically, on client server architecture.

HTTP Request
XML, JSON, HTML, TEXT etc
HTTP Response
Browser
HTTP Request
XML, JSON, HTML, TEXT etc
HTTP Response
Application
REST Server (Web Services)

Representational State Transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating web services.

RESTful services allow the requesting systems to access and manipulate textual representations of resources by using a uniform and predefined set of stateless operations. A stateless protocol operation does not require the server to retain session information or status about each communicating partner for the duration of multiple requests.

Unit # 3 Architecture and Reference Model

REST is not a protocol. It is about manipulating resources, uniquely identified by URIs. A resource is stateful and contains a link pointing to another resource. All the actions on resources are done through a Uniform Interface.

As REST is an architecture style, it can be mapped to multiple protocols such as HTTP, CoAP, etc…

Six guiding constraints define a RESTful system. These constraints restrict the ways in which the server can process and respond to client requests

- **Client-server**: separation of concerns is the principle behind the client-server constraints.
- **Stateless server:** request from client to server contains all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- **Cache:** the client can reuse response data, sent by the server, by storing it in a local cache.
- **Layered system:** allows an architecture to be composed of hierarchical layers. It enables the addition of features like a gateway, a load balancer, or a firewall to accommodate system scaling.
- **Code-on-demand:** (optional) REST allows client functionality to be extended by downloading and executing code in the form of scripts (e.g. JavaScript).
- **Uniform interface**
    - **Identification of resources:** resource identifier enables the identification of the particular resource involved in an interaction between components.
    - **Manipulation of resources through representations:** resource representations are the state of a resource that is transferred between components.
    - **Self-descriptive messages:** contain metadata to describe the meaning of the message.
    - **Hypermedia as the engine of application state or HATEOAS:** Clients find their way through the API by following links available in the resource representations.

Unit # 3 Architecture and Reference Model

IoT Architecture Basics

So what are we looking for in an "end-to-end" or complete IoT architecture anyway? Here are some important requirement :

- **Concurrent Data Collection** – support for collection, analysis and control from a large number of sensors or actuators
- **Efficient Data Handling** – minimize raw data and maximize actionable information
- **Connectivity and Communications** – provide network connectivity and flexible, robust protocols support between sensors/actuators and the cloud
- **Scalable** – scale individual elements in the system using the same architecture
- **Security** – end to end encryption and monitoring
- **Availability and Quality of Service** – minimal latencies and fault tolerant
- **Modular, Flexible and Platform-independent** – each layer should allow for features, hardware or cloud infrastructure to be sourced from different suppliers
- **Open Standards and Interoperable** – communication between the layers should be based on open standards to ensure interoperability
- **Device Management** – enable automated/remote device management and updates
- **Defined APIs** – each layer should have defined APIs that allow for easy integration with existing applications and integration with other IoT solutions

Common Architectures:

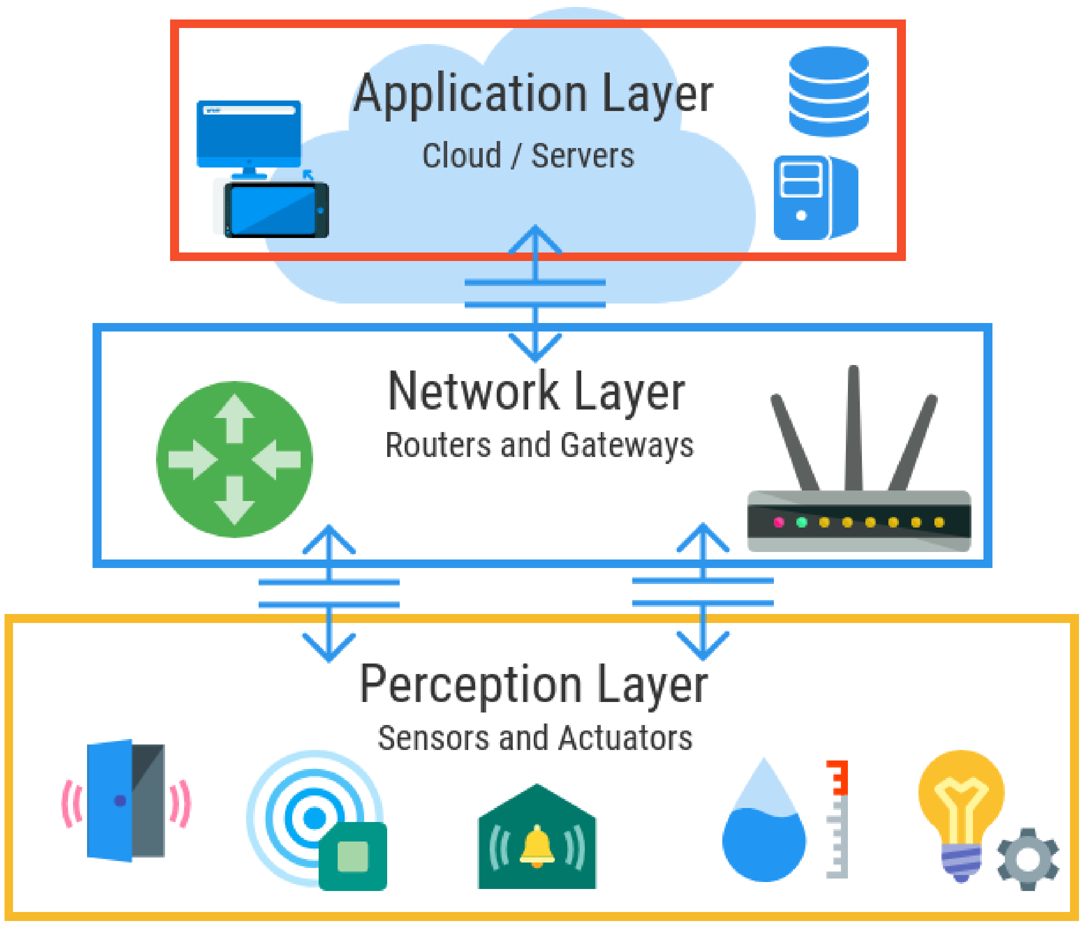
While we can't cover all of the possibilities and permutations, the following group of architectures should give you a greater understanding of the core design considerations and typical primary functional layers in an end-to-end IoT stack.

Three Layer (Tier) IoT Architecture

While there are myriad bits that build a complete end-to-end IoT architecture, this architecture simplifies it down to three fundamental building blocks.

1. Perception layer – Sensors, actuators and edge devices that interact with the environment
2. Network Layer – Discovers, connects and translates devices over a network and in coordination with the application layer
3. Application Layer – Data processing and storage with specialized services and functionality for users

Unit # 3 Architecture and Reference Model



The fundamental Three Layer IoT Architecture

Devices make up a physical or perceptual IoT layer and typically include sensors, actuators and other smart devices. One might call these the "Things" in the Internet of Things. Devices, in turn, interface and communicate to the cloud via wire or localized Radio Frequency (RF) networks. This is typically done through gateways. Often times IoT devices are said to be at the "edge" of the IoT network and are referred to as "edge nodes". When selecting a device, it is important to consider requirements for specific I/O protocols and potential latency, wired or RF interfaces, power, ruggedness and the device's overall sensitivity. It is critical to determine how much device flexibility your architecture should have.

Many newer devices are IoT ready right out of the box (e.g. are sold with low power bluetooth or are Ethernet enabled). However, most sensors, actuators and legacy devices still interface via conventional "pre-IoT" methods such as analog or serial connections. It is common practice to connect one or more of these conventional devices to microcontrollers, systems on modules (SOMs) or single-board computers (SBCs) with the necessary peripherals (e.g. Arduino, NetBurner, or Raspberry Pi). At a minimum, such collectors provide network connectivity between the edge nodes and a master gateway. In some instances they may be capable of being configured as a gateway as well.

IoT Gateways are are an important middleman element that serves as the messenger and translator between the cloud and clusters of smart devices. They are physical devices or software programs that typically run from the field in close proximity to the edge sensors and other devices. Large IoT systems might use a multitude of gateways to serve high volumes of edge nodes. They can provide a range of functionality, but most importantly they normalize, connect and transfer data between the physical device layer and the cloud. In fact, all data moving between the cloud and the physical device layer goes through a gateway. IoT gateways are sometimes called "intelligent gateways" or "control tiers".

Today, gateways also support additional computing and peripheral functionality such as telemetry, multiple protocol translation, artificial intelligence, pre-processing and filtering massive raw sensor data sets, provisioning and device management. It is becoming common practice to implement data encryption and security monitoring on the intelligent gateway so as to prevent malicious man-in-the-middle attacks against otherwise vulnerable IoT systems. NetBurner devices can be used as robust IoT Gateways, as well as IoT Device Collectors, as mentioned above.

Certain gateways offer an operating system that is specialized for use in embedded and IoT systems along with optimized low-level support for different hardware interfaces, such as NetBurner's SOMs with our custom Real Time Operating System (RTOS) and interface libraries. Managing memory, I/O, timing and interface is not a trivial task. According to Google Cloud, "Generally these abstractions are not easy to use directly, and frequently the OS does not provide abstractions for the wide range of sensor and actuator modules you might encounter in building IoT solutions." Libraries are typically available based on standard protocols. Oftentimes, the most optimized libraries will be part of commercially available development kits and SDKs (as is the case with NetBurner for a multitude of protocols and hardware types).

The **Cloud** is the application layer. It communicates with the gateway, typically over wired or cellular internet. The "Cloud" might be anything from services like AWS or Google Cloud, server farms, or even a company's on-premises remote server. It provides powerful servers and databases that enable robust IoT applications and integrate services such as data storage, big data processing, filtering, analytics, 3rd party APIs, business logic, alerts, monitoring and user interfaces. In a Three Layer IoT Architecture, the "Cloud" is also used to control, configure, and trigger events at the gateway, and ultimately the edge devices.

Five Layer (Tier) IoT Architecture

The Five Layer IoT Architecture essentially builds upon the three layer approach (see figure above). This is still primarily a cloud-centric IoT architecture, where almost all of the IoT data processing is done on the cloud or a remote server. The difference between the Five Layer IoT Architecture and the Three Layer IoT Architecture are the addition of the following:

- A Business Layer
- A Processing Layer

The top-level Business Layer highlights the various management, business logic, and top-level requirements that need to be coordinated for a sustainable and successful architecture that is able to provide consistent value to the business and end users. It also reinforces the idea that IoT applications may be just a part of an organization's portfolio of interconnected technology and business areas.

The use of a Processing Layer unveils an important element of many IoT systems which need to incorporate numerous layers of processing in their architecture; oftentimes to filter down massive data sets and thus conserve resources. Using the processing layer at more than one point within in a specific architecture may be required for particular systems.

Business Layer	Manages the entire IoT system, its functionality, applications, and business models.
Applications Layer	Provides application specific services to users.
Processing Layer	Analyses, stores, and processes large data sets. Might use databases, cloud computing, big data processing resources.
Transport Layer	Convert and transfers sensor data between layers and through networks such as 3G, LAN, Bluetooth, LoRaWAN etc. A typical IoT gateway.
Perception or Physical Layer	Sensors gather data from the environment; actuators turn things on or off, or set values.

Fog Computing IoT Architecture

Fog computing is a newer convention that moves certain IoT services, like monitoring and pre-processing, closer to the edge to enable faster local decision making and automation. The Fog Layer resides between the Physical Layer (sensors and devices) and Transportation Layer (gateway) in what might be called the local network for that IoT cluster. In Fog architectures, computational and storage resources are typically provided by what is called a “Smart IoT Gateway” or “Fog Node”, which can also be laterally networked to other fog nodes.

According to NIST’s 2018 Fog Computing Definition Draft, “Fog nodes may be either physical or virtual elements and are tightly coupled with the smart end-devices or access networks. Fog nodes typically provide some form of data management and communication service between the peripheral layer where smart end-devices reside and the Cloud. Fog nodes, especially virtual ones, also referred as cloudlets, can be federated to provide horizontal expansion of the functionality over disperse geolocations.”

Fog architectures can have many benefits. By pre-processing sensor data, they can reduce bandwidth requirements between the gateway and the cloud while reducing the resource consumption on the cloud. They also can lead to significantly improved real-time performance. Using the Fog Architecture makes a lot of sense for use cases where the above reasons hold value. It is sort of like refining a raw material closer to the source and only transporting the refined product to market. Fog nodes may even be configured to communicate directly with other fog nodes to create a mesh that can bypass the cloud completely.

Fog computing architectures attempt to address requirements surrounding real-time performance, security and efficiency. The figure below illustrates the added layers entailed with the Fog Architecture.You can see how four new layers sit on the margin between the Physical and Transport Layers and the value that these new layers can provide. That said, this also increases the complexity of the architecture, and the new layers introduce additional steps and data conversions which can lead to more points of failure.

Business Layer		Manages the entire IoT system, it’s functionality, applications, and business models.
Applications Layer		Provides application specific services to users
Processing Layer		Analyses, stores, and processes large data sets. Might use databases, cloud computing, big data processing resources
Transport Layer		Transfers sensor data between layers and through networks such as 3G, LAN, Bluetooth, LoRaWAN etc. A typical IoT typical gateway.
Fog Layer –	Security Layer	Encrypt / decrypt data.
	Storage Layer	Store files or data with localized relevance.

Smart IoT Gateway	Pre-Processing	Filter, processes, analyze and reduce edge data or process commands or subscriptions from the cloud.
	Monitoring	Monitor power, resources, responses, and services, access.
Perception or Physical Layer		Sensors gather data from the environment; actuators turn things on or off, or set values.

Table**: A Fog Computing Architecture**

Edge Computing Architecture

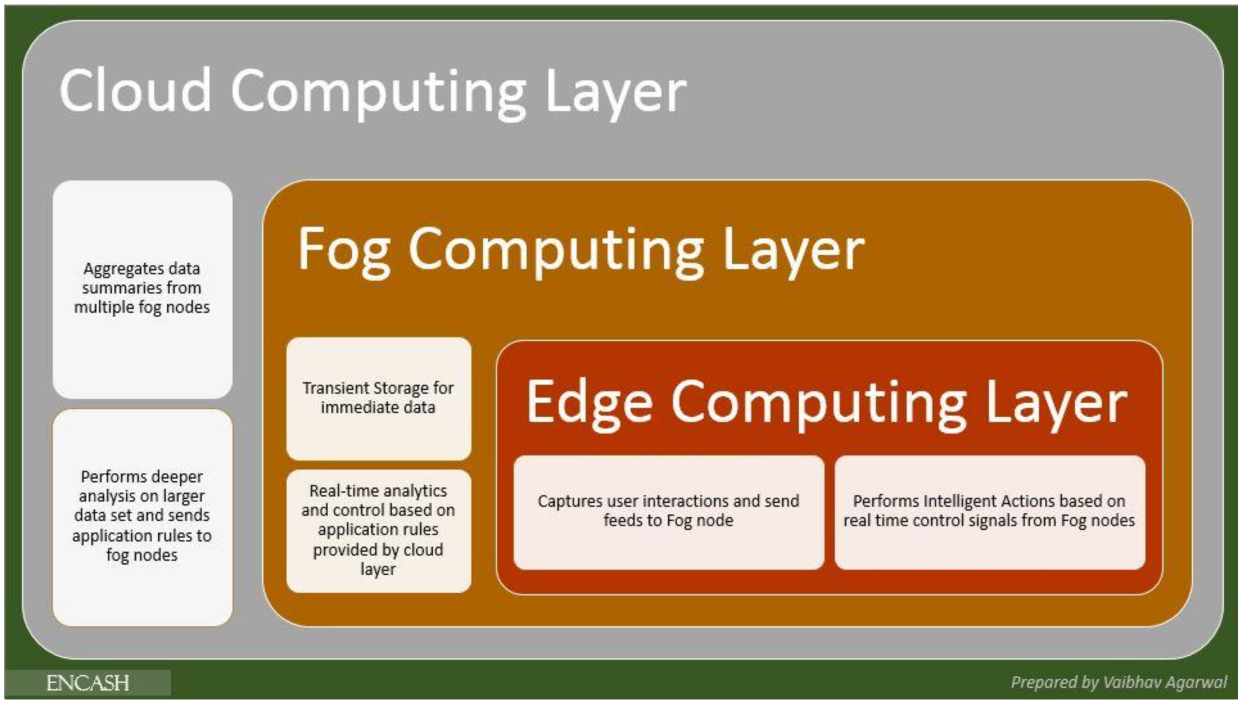
Edge computing is closely related to fog computing, where the goal is to keep certain processing capabilities and functionality closer to the edge nodes. It can be particularly useful in reducing sense and respond latencies for applications that require real-time performance. In edge computing, processing takes place at the physical perception layer directly on a smart device or on an IoT device collector as discussed earlier in this article. It can work independently or in any combination with fog and cloud computing.

If the edge node's processing unit were powerful enough, we could even extend layers for transport, security, storage, pre-processing, and monitoring, thereby reducing the work and functionality required between it and the cloud. According to info.opto22.com, "Edge computing saves time and money by streamlining IoT communication, reducing system and network architecture complexity, and decreasing the number of potential failure points in an IoT application. "Conversely, there are many reasons why having too much computing at the edge can be inefficient and overkill.

In this modality, edge computing provides similar benefits as fog computing but with an even greater capability to reduce localized latencies. Edge computing potentially allows for decentralization of computing, even greater data privacy, and allows for mesh networking off of the cloud. We go into more detail on edge computing in our piece called, "Computing from the Edge." Whether or not the benefits of this architecture outweigh its potential cons will depend on the specific requirements of the system and applications in question.

Hybrid Cloud-Fog-Edge Architecture

As hinted to earlier, fog and edge architectures can be hybridized with cloud-centric IoT architectures if deemed to be a good fit for a project's requirements and business objectives. The below diagram portrays one combination which uses a nested configuration.


Cloud Computing Layer
Aggregates data summaries from multiple fog nodes
Performs deeper analysis on larger data set and sends application rules to fog nodes
Fog Computing Layer
Transient Storage for immediate data
Real-time analytics and control based on application rules provided by cloud layer
Edge Computing Layer
Captures user interactions and send feeds to Fog node
Performs Intelligent Actions based on real time control signals from Fog nodes
ENCASH
Prepared by Vaibhav Agarwal

Example of hybridizing Iot Architectures

Mist Computing Architecture

Mist! Really?! Without being too judgmental about the naming trend going on here, let's just touch on this fairly young IoT topology. Mist is an optional layer and a subclass of fog computing. Inspired by the proliferation of IoT and introduction of IoT social applications (e.g. crowd sensing) the Mist Computing Architecture attempts to optimize for scalable, cost-efficient platforms, distributed data analytics, allocation and provisioning of limited resources, and reduced response times.

A Mist Layer sits on the verge between the edge domain and the fog domain, and it supports the sharing of real-time situational awareness of the environment across nodes within the same local network via mesh connectivity. Mist also supports lightweight computing residing on the edge network fabric itself. Through decentralization of computing power, mist can also help reduce the load on fog node gateways, offer a low cost processing solution, enable localized, autonomous decision making and AI, and provide real-time responses and fault tolerance.

Unit # 3 Architecture and Reference Model

URI (Uniform Resource Identifier)

A URI (Uniform Resource Identifier) is a sequence of characters that identifies a logical or physical resource. Universal Resource Identifiers are specified in the Internet Engineering Task Force (IETF) Request for Comments (RFC) 3986 and are summarized and extended in documentation for the W3C's Web Architecture, Architecture of the World Wide Web, Volume 1. According to the specifications, resources do not have to be accessible on the Internet. Examples of resources include electronic documents, elevator door sensors, XML namespaces, web pages and ID microchips for pets.

There are two types of URIs, Uniform Resource Identifiers (URLs) and Uniform Resource Names (URNs).

Uniform Resource Locator (URL) – this type of URI begins by stating which protocol should be used to locate and access the physical or logical resource on a network. If the resource is a web page, for example, the URI will begin with the protocol HTTP. If the resource is a file, the URI will begin with the protocol FTP or if the resource is an email address, the URI will begin with the protocol mailto. It is important to remember that URLs are not persistent. This means that if the resource's location changes, the URL also needs to change to point to the resource's new location.

Uniform Resource Name (URN) – this type of URI does not state which protocol should be used to locate and access the resource; it simply labels the resource with a persistent, location-independent unique identifier. A URN will identify the resource throughout its lifecycle and will never change. Each URN has three components: the label "urn," a colon and a character string that serves as a unique identifier.

Every URL is also a URI, but not vice versa.

URI Syntax

The generic form of any URI

is **scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]**

Scheme: The scheme lays out the concrete syntax and any associated protocols for the URI. Schemes are case-insensitive and are followed by a colon. Ideally, URI schemes should be registered with the Internet Assigned Numbers Authority (IANA), although nonregistered schemes can also be used.

While the two slashes shown in the example above are required by some schemes, they are not required by all schemes, including authority components, which are described below.

Authority component: An authority component is made up of multiple parts: an optional authentication section, a host -- consisting of either a registered name or an IP address -- and an optional port number. The authentication section contains the username and password, which are separated by a colon and followed by the symbol for at (@). After the @ comes the hostname, which is in turn followed by a colon and then a port number. It is important to note that IPv4 addresses must be in dot-decimal notation, and IPv6 addresses must be enclosed in brackets.

The path, which contains data, is notated by a sequence of segments separated by slashes. The path must begin with a single slash if an authority part was present. It may also begin with a single slash even if there is no authority part, but it cannot begin with a double slash. Keep in mind that while this part of the syntax may closely resemble a particular file path, it does not always imply a relation to that file system path.

Query (optional): The query contains a string of nonhierarchical data. Although the syntax is not well-defined, it is most often a sequence of attribute value pairs separated by a delimiter, such as an ampersand or a semicolon. The query is separated from the preceding part by a question mark.

Fragment (optional): The fragment contains a fragment identifier that provides direction to a secondary resource. For example, if the primary resource is an HTML document, the fragment is often an ID attribute of a specific element of that document. If the fragment identifies a certain section of an article identified by the rest of the URI, a Web browser will scroll this particular element into view. The fragment is separated from the preceding part by a hash (#).

URI resolution and references

URI resolution is one of a few common operations performed on URIs that are also URLs. It involves determining the proper data access method and parameters needed to locate and retrieve the resource that the URI points to.

A URI-reference is used to determine common usage for a URI. A URI reference may take the form of a full URI, a specific portion of a full URI or an empty string. If there is a fragment identifier, it will identify some portion of the resource referred to by the rest of the URI.

A URI-reference can be a URI, but it can also be what is known as a relative reference. A URI is a relative reference if the URI-reference's prefix does not match the syntax of a scheme followed by its colon separator. In order to determine what components are present and whether the reference is relative, each of the five URI components are parsed for its subparts and their validation.

Unit # 3 Architecture and Reference Model

Challenges of IoT

IoT deployment is diversifying from consumer-based applications such as smart home devices and wearables to mission-critical applications in the areas of public safety, emergency response, industrial automation, autonomous vehicles and the Internet of Medical Things (IoMT).

As these mission-critical applications proliferate, engineers and designers must address important design and test considerations and tradeoffs from early design phase to manufacturing outcomes.

Addressing technical challenges through the 5Cs of IoT

The top five challenges of designing for IoT, the '5C's of IoT', are Connectivity, Continuity, Compliance, Coexistence and Cybersecurity.

Connectivity: Enabling a seamless flow of information to and from a device, infrastructure, cloud and applications, is a top IoT challenge because wireless connectivity is highly complex, and dense device deployments further complicate operations. Yet, mission critical IoT devices are expected to work reliably without fail even in the toughest environments. Fast-evolving wireless standards add to the complexity, and engineers face constant challenges in keeping pace with the latest technologies while ensuring devices can work seamlessly throughout the ecosystem.

Responding to connectivity challenges requires design and test solutions that are highly flexible and configurable, and upgradable to meet future needs. Flexibility is needed to test devices with many radio formats, to assess device performance under actual operation modes, and to support over-the-air (OTA) testing in signaling mode without the need for a chipset-specific driver. The solution should be simple, inexpensive, and able to be used in both R&D and manufacturing to leverage code and minimize measurement correlation issues across the different phases of development.

Continuity: Ensuring and extending battery life, one of the most important considerations for IoT devices. A long battery life is a huge competitive advantage in consumer IoT devices. For industrial IoT devices, a battery life of five or ten years is the common expectation. For medical devices such as pacemakers, device life can mean the difference between life and death. And of course, battery failure is not an option.

To meet IoT battery life requirements, integrated circuit (IC) designers need to design ICs with deep sleep modes that consume very little current and reduce clock speed and instruction sets, as well as implement on low battery voltages. For wireless communications, standards groups are defining new low power consumption operating modes such as NB-IoT, LTE-M, LoRa, Sigfox that offer limited active operation time while maintaining low power consumption. Designers who integrate sensing, processing, control and communication components into a final product, must know how the peripherals behave and consume power, and optimize the product's firmware and software to simplify operation and reduce consumption.

Compliance: IoT devices must adhere to radio standards and global regulatory requirements. Compliance testing includes radio standards conformance and carrier acceptance tests, and regulatory compliance tests such as RF, EMC, and SAR tests. Design engineers often scramble to meet tight product introduction timeline and ensure smooth global market entry while complying with the latest regulations – which are frequently updated

Because compliance testing is complex and time-consuming, it can take days or weeks to complete if performed manually. To keep to a product release schedule, designers can consider investing in inhouse pre-compliance test solutions they can use at every design stage, to fix issues early. Choosing one that is adapted from the test lab compliance system can also help to ensure measurement correlation and reduce risk of failures.

Coexistence:

With billions of devices, congestion in the radio channels is a problem that will only get worse. To address wireless congestion, standards bodies have developed test methodologies to evaluate device operations in the presence of other signals. For instance, in *Bluetooth,* adaptive frequency hopping (AFH) lets a *Bluetooth* device drop channels that experience high data collisions. Other collision avoidance techniques such as listen before talk (LBT) and cooperative collision avoidance (CCA) also improve transmission effectiveness. But effectiveness in a mixed-signal environment is unknown, and when the radio formats don't detect each other, collisions and data losses will occur.

An industrial sensor that loses the control signal, or a medical infusion pump that stops working due to surrounding interference, can have dire consequences. Coexistence testing is therefore crucial, to measure and assess how a device will operate in a crowded, mixed signal environment, and to assess the potential risk in maintaining wireless performance in the presence of unintended signals found in the same operating environment.

Cybersecurity: Most traditional cybersecurity protection tools have focused on network and cloud. Endpoint and over-the-air (OTA) vulnerabilities are frequently overlooked. While mature technologies like *Bluetooth* and WLAN are used in many applications, little has been done to address the OTA vulnerabilities. The complexity of these wireless protocols translates into potential unknown pitfalls in device radio implementations that could allow hackers to access or take control of a device.

According to IDC, 70 percent of security breaches originate from endpoints. Extra care should be taken to safeguard these IoT devices. OTA vulnerabilities and potential point of entries into endpoint devices should be identified, and devices should be tested using a regularly updated database of known threats/attacks to monitor device response and detect anomalies.

Unit # 3 Architecture and Reference Model

Challenges of Designing an Embedded IoT Hardware System

Designing a hardware for embedded devices in the IoT ecosystem requires a deep thoughtful planning. The reason is, there are **several challenges Embedded designers face in designing a hardware system** for IoT enabled devices. Listed below are a few challenges of designing embedded IoT hardware system:

Lack of necessary flexibility for running applications over embedded systems:

With the rising demand for connected devices, embedded systems need to work with heterogeneous devices and adapt to different networking architectures to cope-up with new functionalities and performances in the real-time environment. Due to this situation of increasing technology adoption and deployment of new applications, embedded system designers face several problems in terms of flexibility while developing embedded IoT systems such as:

1. Problems in ensuring smooth integration of new services
2. Difficulty in adapting to new environments
3. Frequent changes in hardware and software facilities
4. Issues in packaging and integration of small size chip with low weight and lesser power consumption
5. Carrying out energy awareness operations, etc.

The security crisis in embedded system design:

All the IoT hardware products need to perform securely in the real-time embedded environment. Since all the embedded components operate in a highly resource-constrained and in physically insecure situations, engineers often face problems in ensuring the security of these embedded components. These systems have to be designed and implemented to be robust and reliable and have to be secure with cryptographic algorithms and security procedures. It involves different approaches to secure all the components of embedded systems from prototype to deployment.

High power dissipation of embedded system design:

Another increasingly aggravating limitation is power dissipation of microprocessor hardware design for getting the best performance out of real-time applications and devices. The persistent challenge is how to deploy an embedded system with an increasing number of transistors and with an acceptable power consumption ratio. There are two causes of high power dissipation in designing low-power embedded systems:

First, because the power dissipation per transistor is increasing with the increase in gate density, the power density of system on chips is set to increase. Thus, the engineers must reduce overall embedded systems' power consumption by using efficient system architecture design rather than relying on process technology alone.

Second, engineers focus on better performance with low power consumption by increasing the frequency of the system, which burns more power. Engineers need to pay more attention to design choices as well.

Problems of testing an embedded system design

For ensuring a reliable product design, conducting in-depth testing, verification, and validation is another challenge.

1. **Embedded Hardware Testing:** This is similar to all the testing types where embedded developers use hardware based test tools. This refers to the embedded hardware tested for the system's performance, consistency, and validation as per the product requirement.
2. **Verification:** Ensuring whether functional verification has been implemented correctly or not.
3. **Validation:** Referring to ensure whether the product matches with the requirement and passes all the quality standards.

Inadequate functional safety of safety-critical embedded systems:

Functional safety is considered as a part of a product's overall safety. Embedded systems are considered as generalized control systems, which perform various control functions that require autonomy, reconfiguration, safety, fault-tolerance and need to eliminate all the unacceptable risks to meet functional safety requirements. These considerations highly influence their use in applications, where many functional loops are competing for the design of computational resources due to which, a number of timing and task-scheduling problems arise.

Increased cost and time-to-market:

Apart from flexibility and security, embedded systems are tightly constrained by cost.

In embedded hardware design, the need originates to derive better approaches from development to deployment cycle in order to handle the cost modeling or cost optimality with digital electronic components and production quantity. Hardware/software code-designers also need to solve the design time problem and bring embedded devices at the right time to the market.

If you are looking for supporting hands to assist with transformational experience in embedded systems design for IoT hardware devices, then connect with eInfochips (An Arrow Company). eInfochips has more than 20 years of experience in providing high quality embedded solutions and services, supporting customers in North America, Europe, and Asia regions.

Unit # 3 Architecture and Reference Model

From silicon design to embedded systems prototyping and from product development to deployment and sustenance, eInfochips maps the journey of its customer by extending hardware and embedded software design services like system modeling and design, rapid prototyping, analytical verification, embedded system deployment, and much more.

Challenges in IoT app development:

Security & Privacy

One of the most controversial issues yet very significant challenges of IoT APP development is privacy & security. Before we move ahead, the security we are talking here is not only the network security but the security of all the components involved in IoT applications. These applications are backed by a network that connects the hardware and software components and involves a huge amount of data traveling through a number of connected devices interfering with the personal spaces of users. With such humungous data online, it is vulnerable to cyber-attacks and hacking. Remember the **2016 Dyn** Cyber-attack- a DDoS attack that caused a slowdown or shut down of major online portals. This always serves as a reminder of security as a challenge while developing IoT applications.

Along with the IoT software development security as a challenge, there are some more key encounters that should be considered while developing an IoT app:

1. Data Exchange Security: This is important to understand that the data transfer from IoT sensors & devices to a platform or gateway and then stored at the cloud. It is essential to ensure data encryption protocol is followed while app development.
2. Physical Security: The IoT devices are usually unattended and hence can be easily tampered by the hackers. Thus, checking whether the security component is added to an IoT device is always a challenge.
3. Cloud Storage Security: Though cloud storage is considered secured it is a challenge for developers to ensure that the IoT platform is properly encrypted, is capable of protecting data and appropriate access & authorization is taken care of.
4. Privacy Updates: Data fetched by IoT devices are always under certain rules & regulations. For example, All the fitness trackers collect user information based on HIPAA guidelines. This means that the information stored on the cloud through the IoT device is compliant to specified regulations. Complying to regulations like HIPAA ensures the privacy of data.

Connectivity

The essence of the Internet of Things application development is the real-time transmission of data. But this becomes a challenge because of poor connectivity or latency. Connected devices that provide useful front-end information is extremely valuable. But poor connectivity becomes a challenge where IoT sensors are required to monitor, process data and supply information. The best example of IoT application suffering due to connectivity is huge manufacturing companies that utilize IoT sensors connected to data platforms cannot perform due to server breakdown.

In fact, connectivity is the foremost concern while connecting devices, applications and cloud platforms. This challenge can be addressed in the design & device application environment. For example- a basic printing machine can be connected with laptops, PCs, mobile devices through WiFi home router but the same is not the case for smart vehicles as they require mobile connected internet. Therefore, developers need to understand the device functions & features and make development decisions accordingly. For IoT devices and platforms to perform well, development teams must ensure that the applications are in good connectivity.

Cross-Platform Compatibility (Hardware & Devices)

IoT applications must be developed keeping in mind the technological changes of the future. Thus, IoT development requires a balance of hardware and software functions. Development teams may concentrate on designing the device that can achieve the finest performance but it might confine updating the product. Also, at the same time, the device allows using new functions, features, help in bug fixing as it has a heavy operating system but may cause a reduction in performance. Hence, it is a challenge for IoT application developers to ensure that device and IoT platform delivers the best performance despite heavy OS, device updates and bug fixings. Many vendors supply with SDKs and APIs for developers to add new functionalities to their already developed application. IoT applications run on the web as well as mobile devices and hence need to be compatible.

As most of the IoT applications are integrated into the legacy systems, it is no less than a challenge for developers to make the system compliant to industry standards and protocols. While developing the IoT application, developers must ensure that the application can be seamlessly integrated without increasing difficulties in the IoT ecosystem (considering both hardware and software).

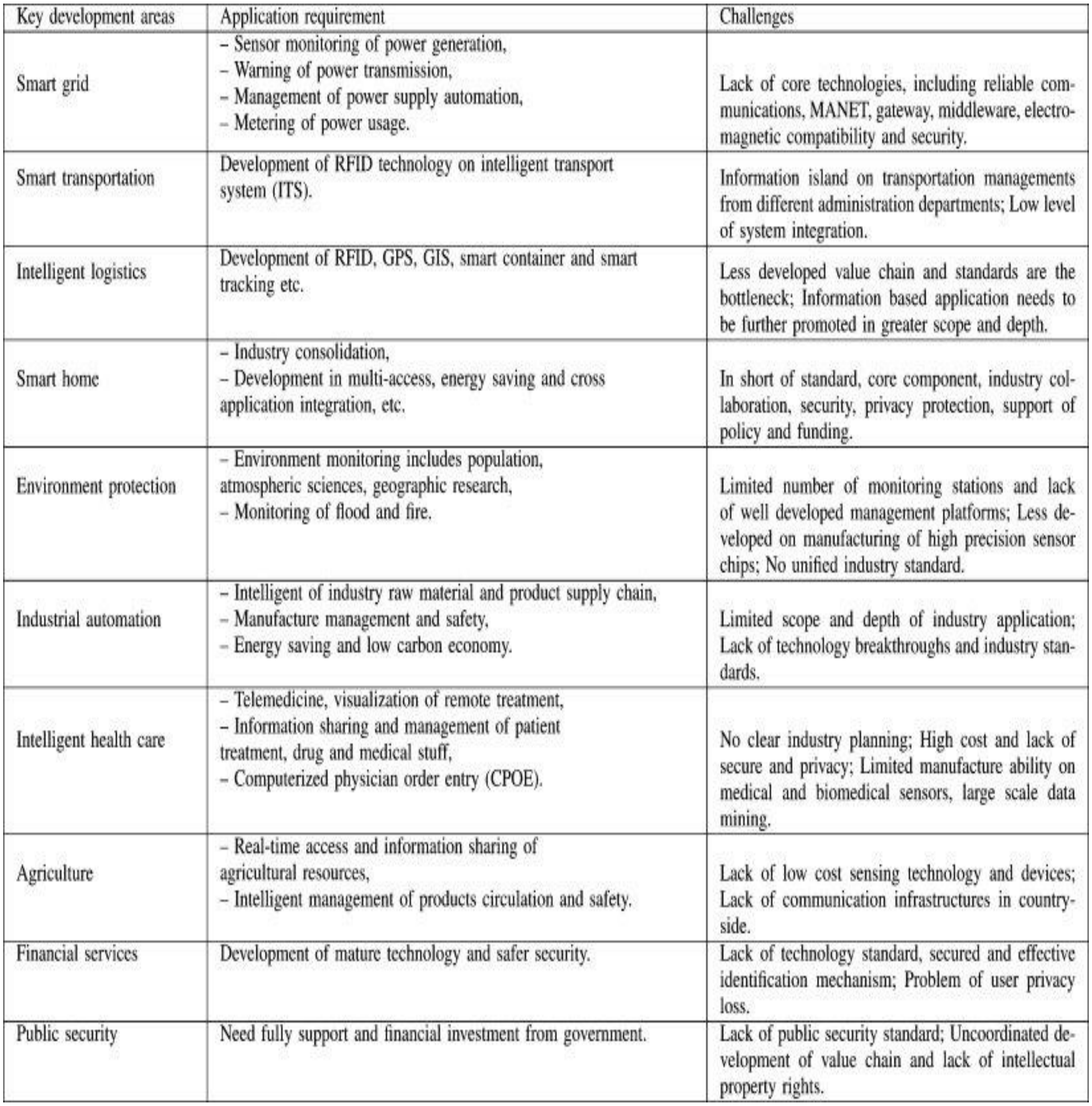
Data Collection & Processing

Since IoT application involves a huge amount of data, data collection & processing the same is a challenge for developers. Along with privacy and security planning, development teams need to ensure that they plan well for the way data is collected, stored or processed within an environment. Also, it is significant to make sure data size that is to be stored on the cloud and comply with platform requirements. To make this certain, IoT application development teams should employ data experts, analytic engineers and machine learning resources to get insights from data stored on the cloud.

In IoT development, data play an important role but what is more crucial here is the processing or usefulness of stored data.

Lack of Skill Set

All of the development challenges above can only be handled if there is a proper skilled resource working on the IoT application development. IoT is a diverse field and thus count on resources that are well aware of software and hardware implementations. A right talent will always get you past the major challenges and will be an important IoT application development asset.



Key development areas	Application requirement	Challenges
Smart grid	– Sensor monitoring of power generation, – Warning of power transmission, – Management of power supply automation, – Metering of power usage.	Lack of core technologies, including reliable communications, MANET, gateway, middleware, electromagnetic compatibility and security.
Smart transportation	Development of RFID technology on intelligent transport system (ITS).	Information island on transportation managements from different administration departments; Low level of system integration.
Intelligent logistics	Development of RFID, GPS, GIS, smart container and smart tracking etc.	Less developed value chain and standards are the bottleneck; Information based application needs to be further promoted in greater scope and depth.
Smart home	– Industry consolidation, – Development in multi-access, energy saving and cross application integration, etc.	In short of standard, core component, industry collaboration, security, privacy protection, support of policy and funding.
Environment protection	– Environment monitoring includes population, atmospheric sciences, geographic research, – Monitoring of flood and fire.	Limited number of monitoring stations and lack of well developed management platforms; Less developed on manufacturing of high precision sensor chips; No unified industry standard.
Industrial automation	– Intelligent of industry raw material and product supply chain, – Manufacture management and safety, – Energy saving and low carbon economy.	Limited scope and depth of industry application; Lack of technology breakthroughs and industry standards.
Intelligent health care	– Telemedicine, visualization of remote treatment, – Information sharing and management of patient treatment, drug and medical stuff, – Computerized physician order entry (CPOE).	No clear industry planning; High cost and lack of secure and privacy; Limited manufacture ability on medical and biomedical sensors, large scale data mining.
Agriculture	– Real-time access and information sharing of agricultural resources, – Intelligent management of products circulation and safety.	Lack of low cost sensing technology and devices; Lack of communication infrastructures in countryside.
Financial services	Development of mature technology and safer security.	Lack of technology standard, secured and effective identification mechanism; Problem of user privacy loss.
Public security	Need fully support and financial investment from government.	Lack of public security standard; Uncoordinated development of value chain and lack of intellectual property rights.