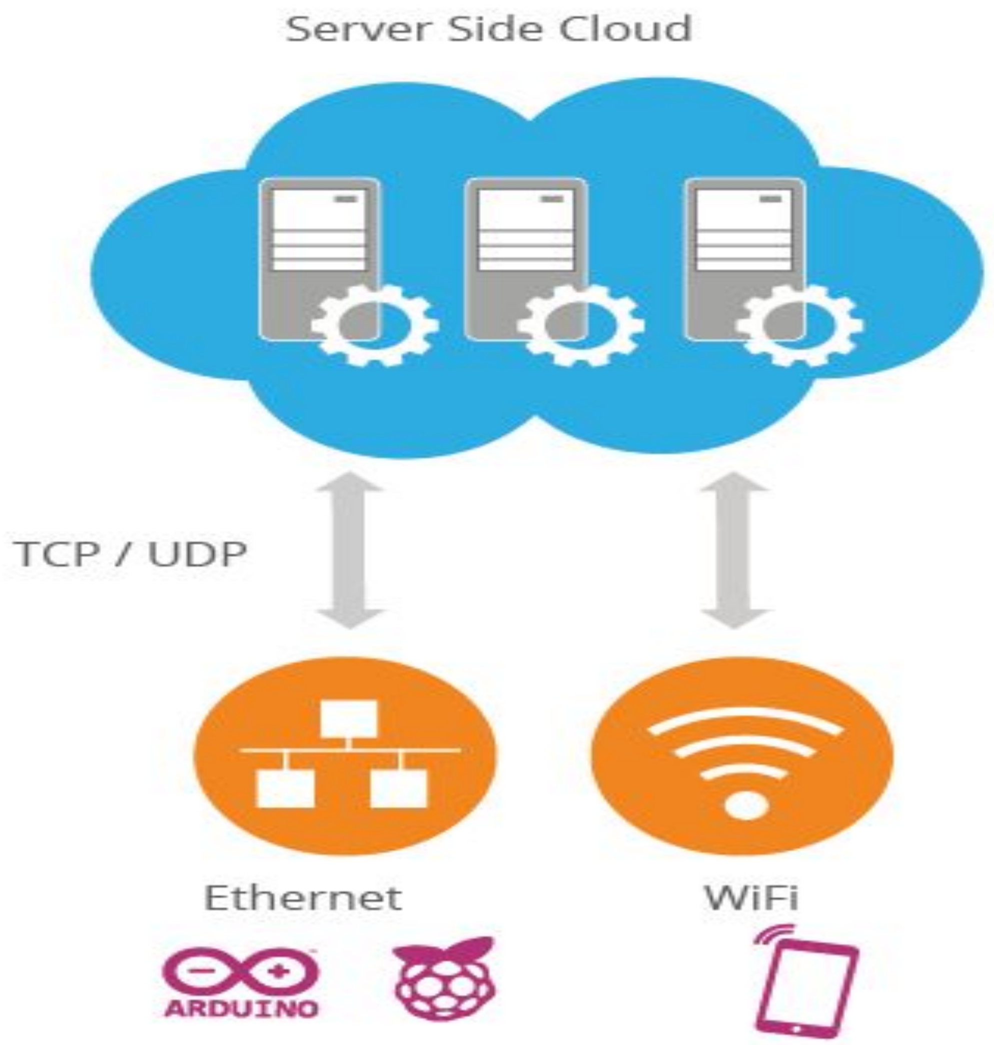
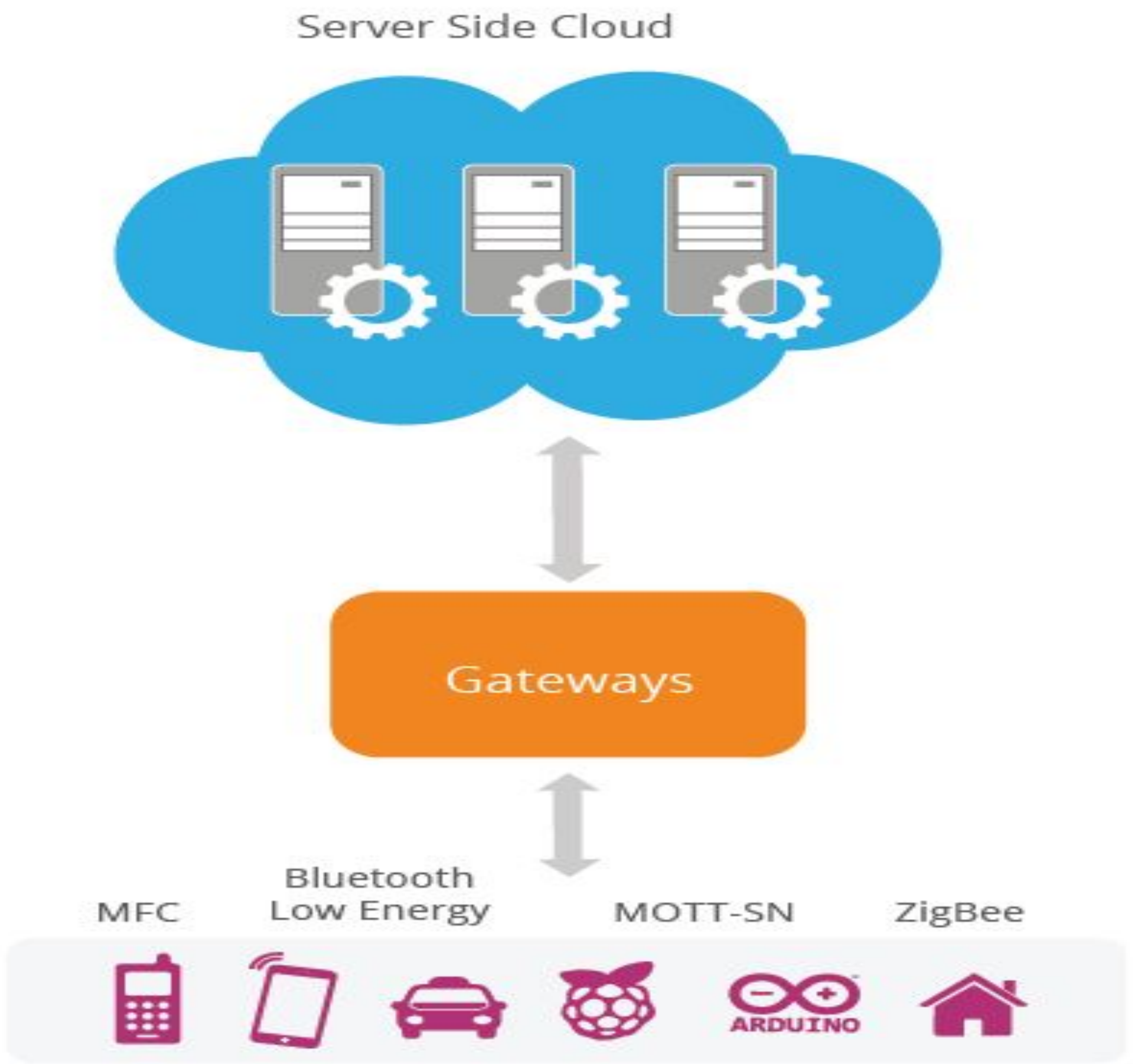
UNIT-2

TOPICS

- Architecture: IoT reference architectures
- Industrial Internet Reference Architecture
- Edge Computing
- IoT Gateways
- Data Ingestion and Data processing pipelines

Architecture: IoT reference architectures





There are several reasons why a reference architecture for IoT is a good thing:

- IoT devices are inherently connected – we need a way of interacting with them, often with firewalls, network address translation (NAT) and other obstacles in the way.
- There are billions of these devices already and the number is growing quickly; we need an architecture for scalability. In addition, these devices are typically interacting 24x7, so we need a highly-available (HA) approach that supports deployment across data centers to allow disaster recovery (DR).
- The devices may not have UIs and certainly are designed to be “everyday” usage, so we need to support automatic and managed updates, as well as being able to remotely manage these devices.
- IoT devices are very commonly used for collecting and analyzing personal data. A model for managing the identity and access control for IoT devices and the data they publish and consume is a key requirement.

Requirements for a Reference Architecture

- Connectivity and communications
- Device management
- Data collection, analysis, and actuation
- Scalability
- Security
- HA
- Predictive analysis
- Integration

Connectivity and Communications

- Existing protocols, such as HTTP, have a very important place for many devices. Even an 8-bit controller can create simple GET and POST requests and HTTP provides an important unified (and uniform) connectivity.
- Firstly, the memory size of the program can be an issue on small devices.
- the power requirements

Device Management

- The ability to disconnect a rogue or stolen device
- The ability to update the software on a device
- Updating security credentials
- Remotely enabling or disabling certain hardware capabilities
- Locating a lost device
- Wiping secure data from a stolen device
- Remotely re-configuring Wi-Fi, GPRS, or network parameters

Data Collection, Analysis, and Actuation

- A few IoT devices have some form of UI, but in general IoT devices are focused on offering one or more sensors, one or more actuators, or a combination of both. The requirements of the system are that we can collect data from very large numbers of devices, store it, analyze it, and then act upon it.
- The reference architecture is designed to manage very large numbers of devices
- If these devices are creating constant streams of data, then this creates a significant amount of data.
- The requirement is for a highly scalable storage system, which can handle diverse data and high volumes.

Scalability

- Any server-side architecture would ideally be highly scalable, and be able to support millions of devices all constantly sending, receiving, and acting on data
- An important requirement for this architecture is to support scaling from a small deployment to a very large number of devices.
- Elastic scalability and the ability to deploy in a cloud infrastructure are essential.

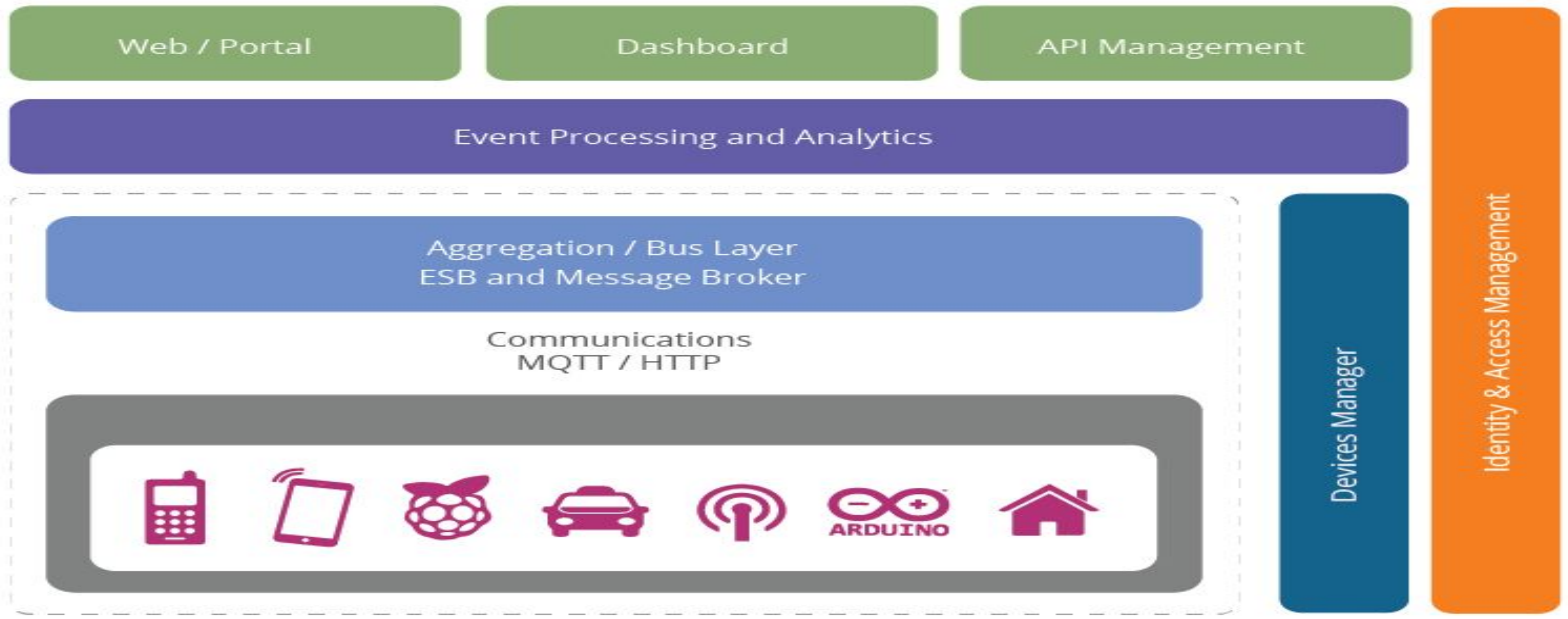
Security

- Risks that are inherent in any Internet system, but that product/IoT designers may not be aware of
- Specific risks that are unique to IoT devices
- Safety to ensure no harm is caused by, for instance, misusing actuators
- The first category includes simple things such as locking down open ports on devices (like the Internet-attached fridge that had an unsecured SMTP server and was being used to send spam).

- Two very important specific issues for IoT security are the concerns about identity and access management. Identity is an issue where there are often poor practices implemented. For example, the use of clear text/ Base64 encoded user IDs/passwords with devices and machine-to-machine (M2M) is a common mistake.
- Encryption on devices that are powerful enough;
- A modern identity model based on tokens and not userids/passwords;
- The management of keys and tokens as smoothly/remotely as possible; and
- Policy-based and user-managed access control for the system based on XACML.

The Architecture

- The reference architecture consists of a set of components. Layers can be realized by means of specific technologies,



Layer are

- The layers are
- Client/external communications - Web/Portal, Dashboard, APIs
- Event processing and analytics (including data storage)
- Aggregation/bus layer – ESB and message broker
- Relevant transports - MQTT/HTTP/XMPP/CoAP/AMQP, etc.
- Devices

The cross-cutting layers are

- Device manager
- Identity and access management

The Device Layer

- The bottom layer of the architecture is the device layer. Devices can be of various types, but in order to be considered as IoT devices, they must have some communications that either indirectly or directly attaches to the Internet.
- Arduino with Arduino Ethernet connection
- Arduino Yun with a Wi-Fi connection
- Raspberry Pi connected via Ethernet or Wi-Fi
- Intel Galileo connected via Ethernet or Wi-Fi Examples of indirectly connected device include
- ZigBee devices connected via a ZigBee gateway
- Bluetooth or Bluetooth Low Energy devices connecting via a mobile phone
- Devices communicating via low power radios to a Raspberry Pi

The Communications Layer

- The communication layer supports the connectivity of the devices. There are multiple potential protocols for communication between the devices and the cloud. The most wellknown three potential protocols are
- HTTP/HTTPS (and RESTful approaches on those)
- MQTT 3.1/3.1.1
- Constrained application protocol (CoAP)

- The two best known are MQTT[6] and CoAP[7].
- MQTT was invented in 1999 to solve issues in embedded systems and SCADA. It has been through some iterations and the current version (3.1.1) is undergoing standardization in the OASIS MQTT Technical Committee[8].
- MQTT is a publish-subscribe messaging system based on a broker model.
- The protocol has a very small overhead (as little as 2 bytes per message), and was designed to support lossy and intermittently connected networks. MQTT was designed to flow over TCP.
  In addition there is an associated specification designed for ZigBee-style networks called MQTT-SN (Sensor Networks).

- CoAP is a protocol from the IETF that is designed to provide a RESTful application protocol modeled on HTTP semantics, but with a much smaller footprint and a binary rather than a text-based approach. CoAP is a more traditional client-server approach rather than a brokered approach. CoAP is designed to be used over UDP.
- For the reference architecture we have opted to select MQTT as the preferred device communication protocol, with HTTP as an alternative option.

- Better adoption and wider library support for MQTT;
- Simplified bridging into existing event collection and event processing systems; and
- Simpler connectivity over firewalls and NAT networks

The Aggregation/Bus Layer

- The ability to support an HTTP server and/or an MQTT broker to talk to the devices;
- The ability to aggregate and combine communications from different devices and to route communications to a specific device (possibly via a gateway)
- The ability to bridge and transform between different protocols, e.g. to offer HTTPbased APIs that are mediated into an MQTT message going to the device.

- The aggregation/bus layer provides these capabilities as well as adapting into legacy protocols. The bus layer may also provide some simple correlation and mapping from different correlation models (e.g. mapping a device ID into an owner’s ID or vice-versa).
- Finally the aggregation/bus layer needs to perform two key security roles. It must be able to act as an OAuth2 Resource Server (validating Bearer Tokens and associated resource access scopes). It must also be able to act as a policy enforcement point (PEP) for policy-based access.

The Event Processing and Analytics Layer

- This layer takes the events from the bus and provides the ability to process and act upon these events. A core capability here is the requirement to store the data into a database.
- Highly scalable, column-based data storage for storing events
- Map-reduce for long-running batch-oriented processing of data
- Complex event processing for fast in-memory processing and near real-time reaction and autonomic actions based on the data and activity of devices and other systems
- In addition, this layer may support traditional application processing platforms, such as Java Beans, JAX-RS logic, message-driven beans, or alternatives, such as node.js, PHP, Ruby or Python.

Client/External Communications Layer

- The reference architecture needs to provide a way for these devices to communicate outside of the device-oriented system.
- Firstly, we need the ability to create web-based front-ends and portals that interact with devices and with the event-processing layer.
- Secondly, we need the ability to create dashboards that offer views into analytics and event processing. Finally, we need to be able to interact with systems outside this network using machine-to-machine communications (APIs).
- These APIs need to be managed and controlled and this happens in an API management system.

- The first is that it provides a developer-focused portal (as opposed to the userfocused portal previously mentioned), where developers can find, explore, and subscribe to APIs from the system. There is also support for publishers to create, version, and manage the available and published APIs;
- The second is a gateway that manages access to the APIs, performing access control checks (for external requests) as well as throttling usage based on policies. It also performs routing and load-balancing;
- The final aspect is that the gateway publishes data into the analytics layer where it is stored as well as processed to provide insights into how the APIs are used.

Device Management

- Device management (DM) is handled by two components.
- A server-side system (the device manager) communicates with devices via various protocols and provides both individual and bulk control of devices. It also remotely manages software and applications deployed on the device. It can lock and/or wipe the device if necessary.
- The device manager works in conjunction with the device management agents. There are multiple different agents for different platforms and device types.
- The device manager also needs to maintain the list of device identities and map these into owners. It must also work with the identity and access management layer to manage access controls over devices

Data Ingestion and Data processing pipelines

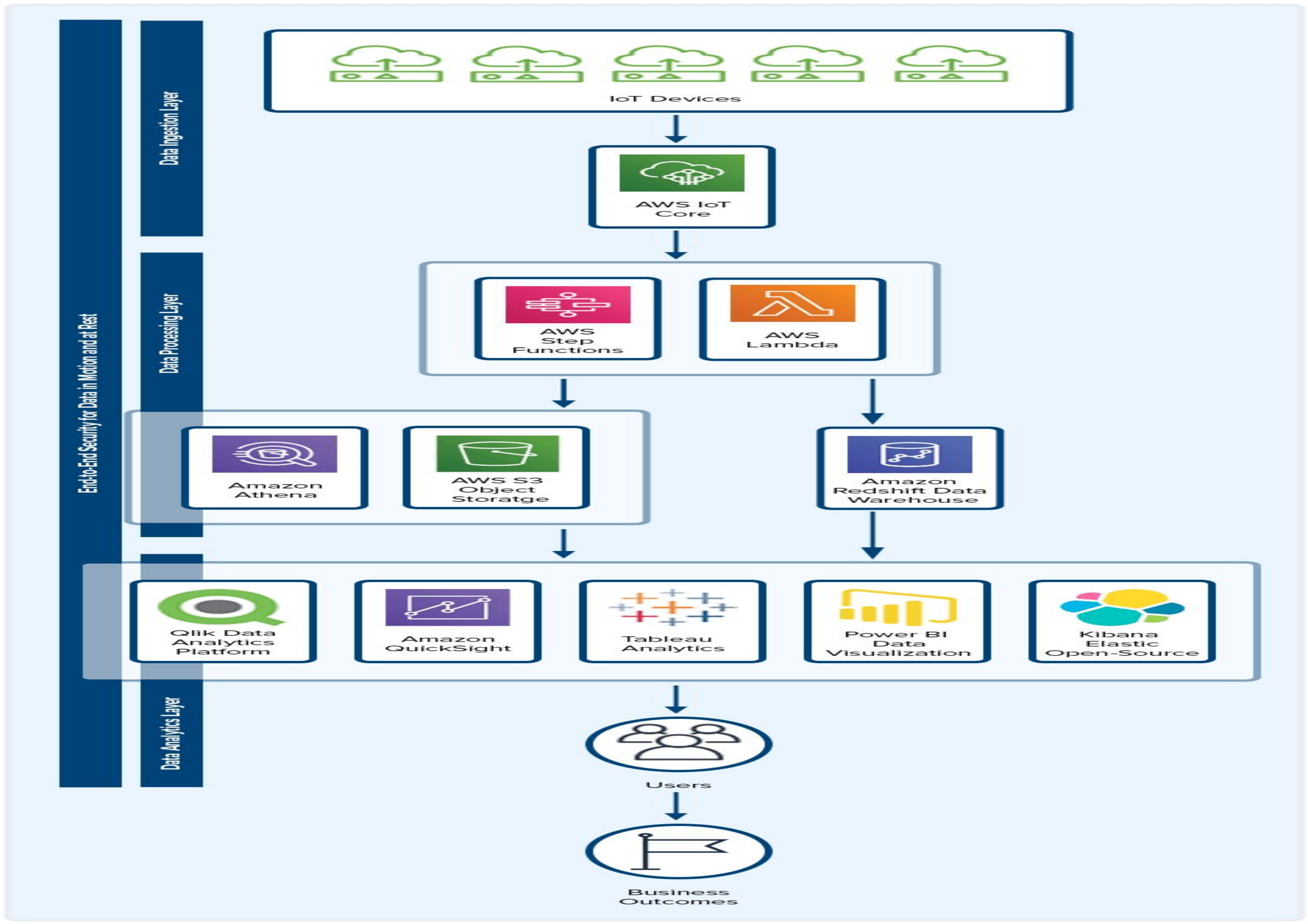
- IoT data pipelines are composed of three layers: data ingestion, data processing, and data analytics. Data ingestion is where device data is generated and brought into the cloud.
- The data processing layer is where your IoT data is acted upon to take it from something raw and turn it into something useable
- The analytics layer is where data is visualized and made available to decision-makers, and where the real value of an IoT data pipeline is realized.
- Data ingestion relies upon the interplay of connected devices and AWS IoT Core. IoT Core allows you to connect and manage billions of IoT devices and is the gateway to the cloud for IoT device data

Data Ingestion and Data processing pipelines

- The data processing layer is where raw IoT data is transformed into something that can be used by your analytics, AI, or machine-learning applications, and secured while in motion and at rest.
- he processing layer is also responsible for routing data to the right storage location. For data pipelines that take advantage of federated data storage architecture, structured data is sent to an Amazon S3 data warehouse, and unstructured data sent to an Amazon S3 datalake.

Data Ingestion and Data processing pipelines

- The final layer of the data pipeline is the analytics layer, where data is translated into value. AWS services such as QuickSight and Sagemaker are available as low-cost and quick-to-deploy analytic options perfect for organizations with a relatively small number of expert users who need to access the same data and visualizations over and over.


End-to-End Security for Data in Motion and at Rest
Data Ingestion Layer
Data Processing Layer
Data Analytics Layer
IoT Devices
AWS IoT Core
AWS Step Functions
AWS Lambda
Amazon Athena
AWS S3 Object Storatge
Amazon Redshift Data Warehouse
Qlik Data Analytics Platform
Amazon QuickSight
Tableau Analytics
Power BI Data Visualization
Kibana Elastic Open-Source
Users
Business Outcomes

- **What is an IoT data pipeline?**
- A data pipeline is any set of processes designed for two things:
- To define what data to collect, where and how;
- To extract, transform, combine, validate, and load the data for further analysis and visualization.
- There are a few categories in which the data pipeline is divided into:
- Ingestion
- Transport
- Storage and Management
- Processing and Visualizing

IoT data pipeline architecture looks like:

- Data Ingestion Layer
- * Data collection layer
- * Data Processing Layer
- * Data storage layer
- *Data query layer
- *Data visualization layer
- **The other way to look at the stages according to Laurenz Da Lus, Solutions Architect at Cloudera are these five elements:**
- **Acquire** — Acquire raw data from source systems
- **Parse** — Convert raw data into a common data model and format
- **Enrich** — Add additional context to the data
- **Profile** — Analyse entity behavior across time (sessionisation)
- **Store** — Store data to support access and visualization

Data Ingestion Layer

- *Data ingestion* involves procuring events from sources (applications, IoT devices, web and server logs, and even data file uploads) and transporting them into a data store for further processing.
- It is about moving **unstructured data** — from where it is originated, into a system where it can be stored and analyzed. Data ingestion is the first step in building the data pipeline.
- At this stage, data comes from multiple sources at variable speeds in different formats. Hence, it is very important to get the data ingestion right in any IoT pipeline.

- Data can be streamed in realtime or ingested in batches. When data is ingested in real time then, as soon as data arrives it is ingested immediately. When data is ingested in batches, data items are ingested in some chunks at a periodic interval of time. Ingestion is the process of bringing data into Data Processing system.
- **Popular data ingestion tools:**
- * Apache Flume
- *Apache Kafka
- *Apache Nifi
- *Google pub/sub

Data Transport/Collection Layer

- Data transport overlaps somewhat with data ingestion, but “ingestion” revolves around getting data extracted from one system and into another, while “transport” concerns getting data from any location to any other.
- Message brokers are a key component in data transport; their purpose is to translate a message from a sender’s protocol to that of a receiver, and possibly transform messages prior to moving them.

Data Transport/Collection Layer

- Apache Kafka is a high-throughput distributed messaging system for consistent, fault-tolerant and durable message collection and delivery. Kafka producers publish streams of records or topics to which consumers subscribe. These streams of records are stored and processed as they occur.
- Kafka is typically used for a few broad classes of applications:
- Real-time streaming data pipelines *between* systems or applications;
- Real-time streaming applications that *transform* streams of data;
- Real-time streaming applications that *react* to streams of data.

Data Processing Layer

- In this layer, our task is to do magic with data, as now data is ready we only have to route the data to different destinations.
- In this main layer, the focus is to specialize Data Pipeline processing system or we can say the data we have collected by the last layer in this next layer we have to do processing on that data.

Data Processing Layer

- **The type of Data processing differs between batch and stream data types.**
- *** Batch:**
- ***Stream:**
- There are three steps in real-time stream processing for Analytics:
- Transformation
- Data enrichment
- Storing of data
- **Transformation** — It includes the conversion of the data which is collected from the IoT device. After this conversion, the resulting data is transferred for further analytics.
- **Data Enrichment** — Data enrichment process is the operation in which the sensor collected raw data is combined with the other data-set to get the results.
- **Storing Data** — This task includes storing the data at the required storage location.

Data Storage/Management Layer

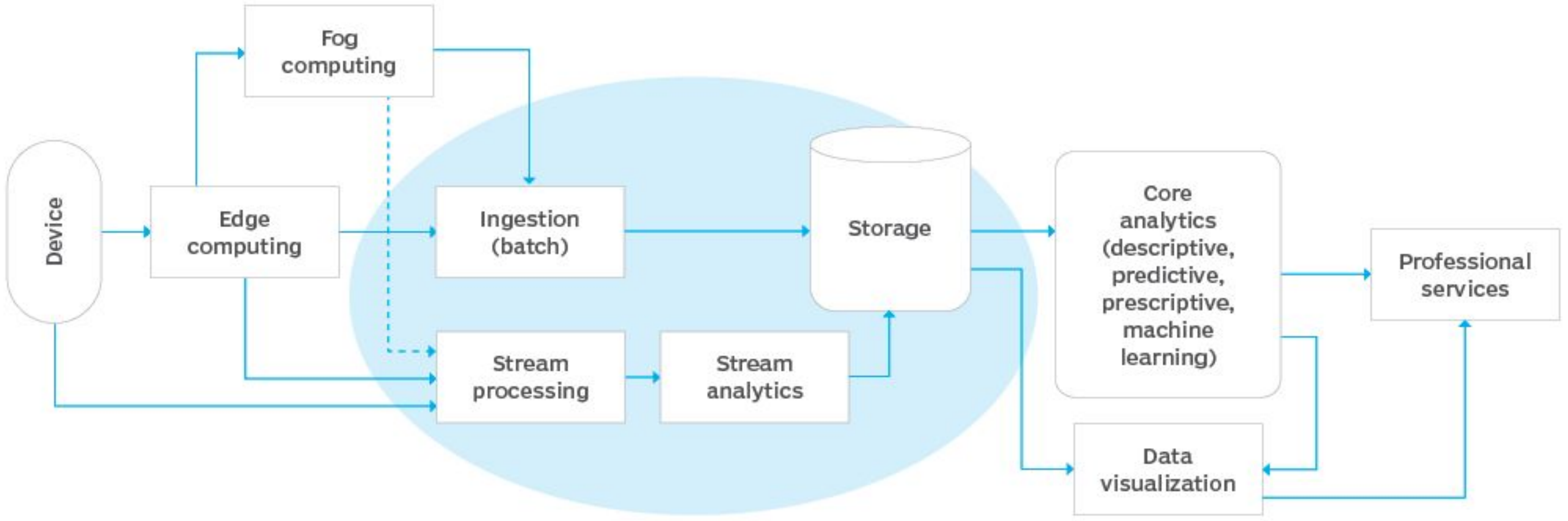
- **PostgreSQL**
- PostgreSQL is an open-source object-relational database management system emphasizing extensibility and standards compliance that has been around so long, it’s become a standby for companies ranging from manufacturing to IoT.
- **Redis**
- Redis is a super fast variant of the NoSQL database known as a key-value store. As such, it’s an extremely simple database that stores only key-value pairs and serves search results by retrieving the value associated with a known key.
- Redis’s speed and simplicity make it well-suited for embedded databases, session caches, or queues. In fact, it’s often used in conjunction with message brokers, or as a message broker itself. The Aiven Redis service can be found here.
- **Cassandra**

- If you’re working with large, active data sets, and need to tweak the trade-off between consistency, availability and partition tolerance, then Apache Cassandra may be your solution. Because data is distributed across nodes, when one node — or even an entire data center — goes down, the data remains preserved in other nodes (depending on the consistency level setting).

Data Visualization Layer

- Data visualization tools and dashboards also support managers, marketers, and even end consumers, but there are simply too many such tools, with too many areas of specialty,
- When time-series data needs to be plotted to a graph and visualized — to monitor system performance, say, or how a particular variable or group of variables has performed over time, then a solution like Grafana might be just the ticket.

IoT data pipeline process



SOURCE: ABI RESEARCH

What is Data Streaming?

- **Data streaming** is the process of transmitting a continuous flow of data (also known as streams) typically fed into stream processing software to derive valuable insights. A data stream consists of a series of data elements ordered in time. The data represents an “event” or a change in state that has occurred in the business and is useful for the business to know about and analyze, often in real-time
- Examples of Data Streaming
- Internet of Things: IoT includes a huge number of devices that collect data using sensors and transmit them in real-time to a data processor. IoT data generates stream data. Wearable health monitors like watches, home security systems, traffic monitoring systems, biometric scanners, connected home appliances, cybersecurity, and privacy systems generate and stream data in real-time.

What is Stream Processing? How Does it Work?

- In order to process streaming or live data, you need a process that is quite different from traditional batch processing. A stream processor collects, analyzes, and visualizes a continuous flow of data. And, of course, to process, you need to start with data streaming. Data streaming is at the beginning of stream processing. Stream processing is used to take in the data streams and derive insights from them, often in real-time.

- Low Latency
- A stream processor should work quickly on continuous streams of data. Processing speed is a primary concern due to two reasons. One, the data comes in as a continuous stream, and if the processor is slow and misses data, it cannot go back.
- Scalability
- Streaming data doesn’t always have the same volume. For example, sensors may often generate low volumes of data, but occasionally, there might be a spike in the data. Since the volume of data is unpredictable, the processor should scale up to handle large volumes of data if required.

- Availability
- A stream processor cannot afford long downtimes. The stream data is continuous and arrives in real-time. A processor must be fault-tolerant, meaning it should be able to continue to function even if some of its components fail.

What Are the Major Components of a Stream Processor?

- Datastream Management
- In datastream management, the objective of the stream processing is to create a summary of the incoming data or to build models. For example, from a continuous stream of facial data, a stream processor might be able to create a list of facial features. Another example of this use case is the internet activity logs. From the constant stream of user click data, the stream processor tries to calculate the user's preferences and tastes.
- Complex Event Processing
- Complex event processing is the use case that applies to most IoT data streams. In this use case, the data stream consists of event streams. The job of the stream processor is to extract significant events, derive meaningful insights, and quickly pass the information to a higher layer so that prompt action can be taken in real-time.

Data generation

- The data generation system denotes the various sources of raw data—like sensors, transaction monitors, and web browsers. They continuously produce data for the stream processing system to consume.

Data collection and aggregation

- Each of the above data generation sources is associated with a client, which receives data from the source. These are known as source clients. An aggregator collates the data from several source clients, sending the data in motion to a centralized data buffer.

Messaging buffering

- The message buffers take the stream data from an aggregation agent and store them temporarily before passing into a logic processor. There are two main types of message buffers: topic-based and queue-based. In the topic-based buffers, the incoming data is stored in the form of records called topics. One or more data producers can contribute to a topic. The queue-based message buffer is more of a point-to-point buffering system, reading from a single producer and delivering to a single data consumer.

Message broker

- The data collection, aggregation, and message buffering systems together form a message broker system. The functionality of the message broker is to aggregate the stream data from various sources, format it, and pass it on to a continuous logic processing system.

- Continuous logic processing
- This is the core of the stream processing architecture. The continuous logic processing subsystem runs various predefined queries on the incoming data streams to derive useful insights. The queries may be as simple as ones stored in an XML file. These queries are continuously run on the incoming data. This subsystem may define a declarative command language for the users to easily create these queries. The continuous logic processing system often runs on distributed machines for scalability and fault tolerance. Over the years, the logic processing system has evolved to support dynamic modification of queries and programming APIs for easier querying.

- Storage and presentation
- These are two supporting systems in stream processing. The storage system keeps a summary of the input data stream and can be used for future references. It also stores the results of the queries that are run on the continuous data stream. The presentation system is used for visualizing the data to the consumers. The presentation system may include a higher level of analytical system or alerts to the end-users.