Introduction

INTRODUCTION TO THE INDUSTRIAL INTERNET OF THINGS (IIoT)

Devices and machinery have always been connected to some kind of electronic controlsystem. However, low cost cloud computing and ubiquitous networking have for the first time connected these systems to each other and to powerful cloud-based analytics technologies. This has created a deluge of data from sensors, devices, equipment and people. The IoT revolution is this intersection of large scale data with the capacity to rapidly analyze it in real time and create applications that respond to these inputs in a meaningful and contextual way.

This has allowed businesses to create new value, revenue, and impact from their existing infrastructure investments. The fastest, most effective way to take advantage of the IoT revolution is to evolve software development, deployment, and management to specifically address how IoT applications will collect/report data from, prescribe actions for, and control connected devices.

"Every sensor creates a new business"

— Benedict Evans, Partner at venture firm Andreessen Horowitz

IIoT brings several new and unfamiliar challenges in terms of the Volume, Velocity and Variety of data being generated. A data-centric approach is required for IIoT applications, forcing organizations to adapt to these new challenges:

- IIoT data sets are voluminous: individual data points, captured over a long period for a large number of devices create a large volume of data that is not suitable for storage in traditional and expensive data warehouse solutions
- IIoT data flows at high velocity: the aggregate rate at which the data from the large distribution of devices is very high: millions of devices emitting millions of data points create a very high aggregate data throughput that traditional applications have trouble scaling to
- IIoT data is varied and distributed: The diverse nature and distribution of connected devices creates highly diverse types of data sets that resist unification into a common schema, and also originate from diverse sources
- IIoT applications will look to leverage the public internet when appropriate
- IIoT applications will need to deal with intermittent connectivity issues
- IIoT applications and data stores will need data security baked in throughout

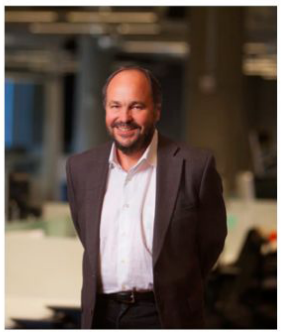
Introduction (Continued)

The large volumes of rapidly flowing data from connected devices require new types of visualizations, analysis, and logic. The fundamental break from the past is that applications have to:

- adapt to new data structures and data handling techniques
- adapt to new data and state management workflows
- adapt to new business and administrative use cases

"I am seeing some major enterprises come to some fairly radical conclusions … **they are willing to build fresh and to leave their existing IT behind.**"

"It's about **catching customers** *& things* **in the act** and being able to provide highly relevant and highly contextual information."

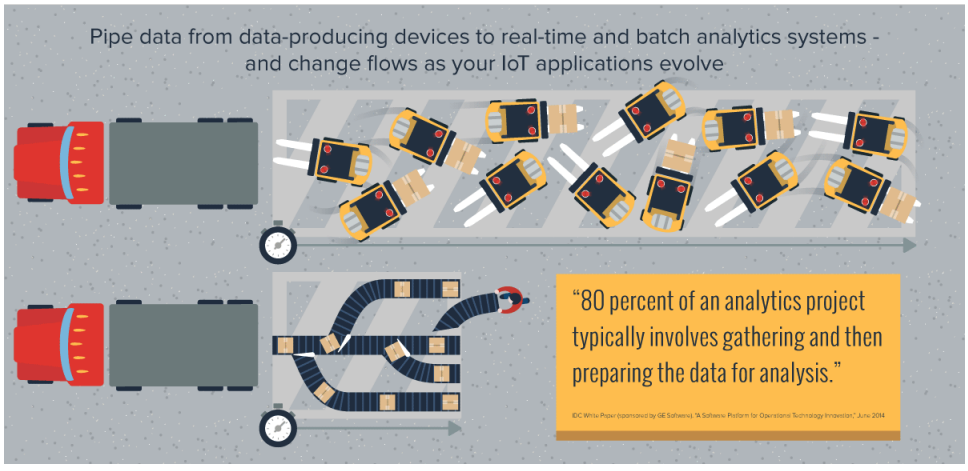


- Paul Maritz, Pivotal Founder & Executive Chairman
GigaOm Structure Data Conference
New York, 2014

As an IIoT application evolves, so do the technologies required to handle IIOT data. The following four modifications to existing data architectures can provide organizations investing in IIOT with new insights while transforming applications:

1. ALLOW ITERATIVE DATA FLOW LOGIC

IDC reports "80 percent of an analytics project typically involves gathering and then preparing the data for analysis." Without an adaptable platform to use myriad data sources, processes, and destinations, organizations face costly rework. IIOT applications designed to evolve as additional requirements emerge save time and money. They can accommodate new data sources (e.g., weather, GPS, sensors, etc.), lters, sorts, and patterns.



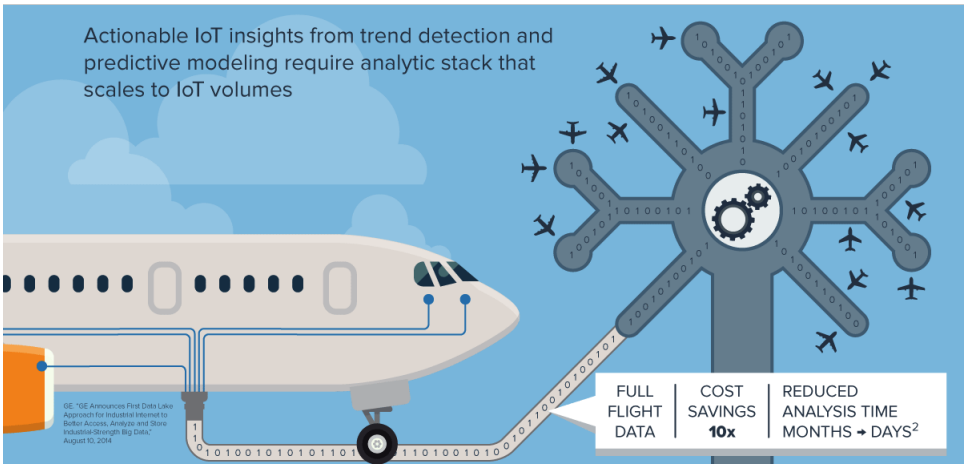
Introduction (Continued)

IIoT requires a unified, distributed, and extensible system for data ingestion, realtime analytics, batch processing, and data export. The development of big data applications that supporting numerous data source types and independently modifiable processing modules needs to be simplified. The time developers spend building and adapting data pipelines for new sources, transformations, and destinations also needs to be reduced. A distributed runtime that is scalable and fault tolerant in highvolume, highvelocity IIOT applications is required, and should use a declarative language that business analysts and data scientists can use without needing to program and compile code.

2. PERSIST DATA FOR TEMPORAL ANALYSIS

The volume of data IIOT produces will increase exponentially, outstripping the growth rate of traditional enterprise data sets. For instance, the data produced by mobile "things" will increase from 18% of the digital universe in 2014 to 27% in 2020, predicts IDC. As sensors connect more devices, the importance of gaining realtime views of IIOT data may equal the imperative to collect data for longerterm analysis, trend detection, and predictive modeling.

Although organizations can now capture much greater amounts of data, many businesses only want to filter and store the data necessary to meet IIOT application requirements. Hadoop has improved the economics of storing highvolumes of data, but actionable insights from trend detection and predictive modeling require an analytics stack. By integrating with SQLonHadoop analytic engines, organizations can easily support IIOT historical analysis using familiar, broadly supported existing SQL skills and tool sets.



The goal of gaining more insight from their data has led industrial enterprises to create the "Industrial Data Lake" approach, which provides data management solutions to enable rapid insights across fleets of assets and industrial processes for datadriven business decisions. In a recent press release, for instance, GE announced it used this approach to "integrate terabytes of full flight data for the first time to produce measurable cost savings of 10x and significantly reduce analysis time from months to days."

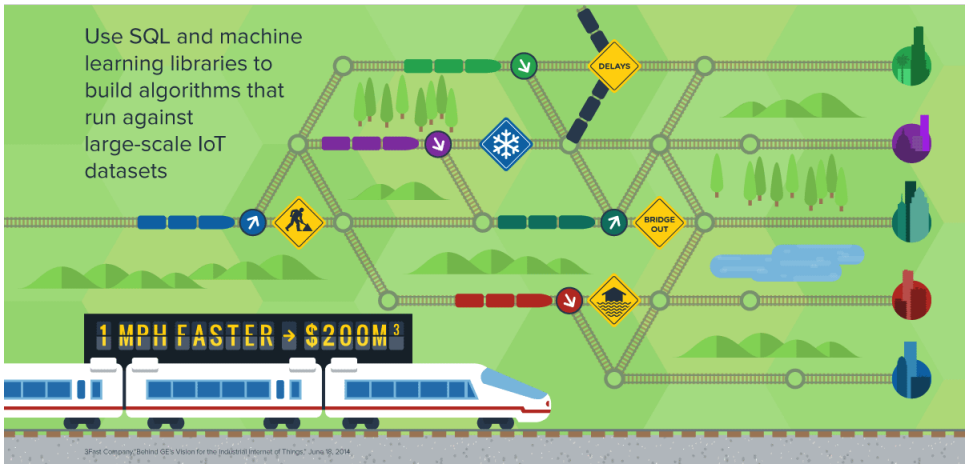
Introduction (Continued)

"Working with Pivotal, we have created a unique industrial data approach that merges information technology with operational technology to better match the productivity and efficiency needs of our customers so they get the most value out of their missioncritical information."

— Bill Ruh, vice president, GE Software

3. USE MACHINE LEARNING FOR PREDICTIVE ANALYTICS AT SCALE

Now that IIOT data can now be easily captured, transmitted, processed, and stored, data scientists can more quickly analyze historical trends and build predictive models with the help of machine learning. These models then become the basis for a new generation of IIOT applications that provide prescriptive recommendations, such as for predictive inventory management or predictive maintenance. The IIOT analytics market is expected to grow at a CAGR of 30% over the next five years. According to IDC, skill set shortages and scalability challenges when applying predictive models to large IIOT data sets can slow implementation and the realization of business benefits.

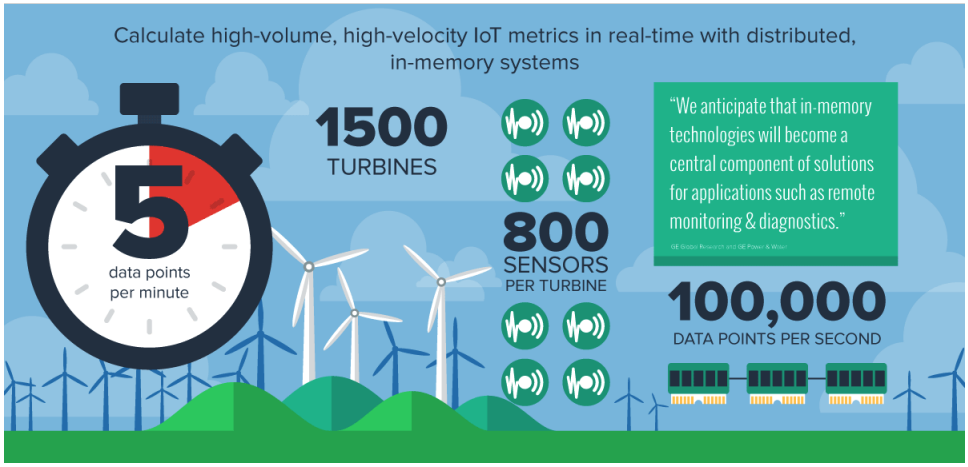


Integration with SQLonHadoop engines enable organizations to leverage existing SQL skills and tools to analyze IIOT data stored in lowcost, largescale commodity data stores. Data science helps ensure IIOT applications are more accurate and effective. Data scientists can drive company and sector level transformations that pave the way for data to become a new source of competitive differentiation. Data scientists can review architecture, business goals, data science methodology and tool use, as well as paths to operationalization. They can further accelerate adoption of analytics tools by building agile, analytic machinelearning models for customers across industries.

Introduction (Continued)

4. APPLY REAL-TIME ANALYSIS AT HIGH-VELOCITY AND HIGH-VOLUME

IIOT applications rely on highspeed access to incoming data for realtime analysis. Defined predictive models allow IIOT data to be scored in realtime so businesses can apply historical data insights and meet incoming data requirements.



The increasing numbers of sensors per device, and the total number of connected devices, have begun to tax legacy data approaches. Organizations are having to make unnecessary speed and capacity tradeoffs that they can't afford.

New tools, such as distributed inmemory data grids, that can ingest over 100,000 times series data points per second and store terabytes of data in memory are required. During typical operation, an IIoT monitoring center will receive a realtime flow of multiple time series data points per minute from millions of unique sensors across over thousands or tens of thousands of global installations — requiring both near realtime and batch analytics. In contrast to a fragmented architecture that processes smaller subsets of data, IIoT enterprises need fast, fleetwide analytics to detect issues and optimize operations.

The Need for a Platform

IIoT is the next iteration of connected machinery. Prior iterations of connected machinery generally formed a category called M2M (Machine-to-Machine). M2M was mostly about point-to-point solutions focused on improving operational efficiency.

IIoT is materially different from prior generations due to the following reasons:

- **Digitization and Connectivity:**
- instead of simple process control loops that connect sensors and actuators in a local feedback loop, IIoT digitizes the output and shares it not just with the immediate vicinity of the device, but also with capable proximal and cloud based systems. These systems allow real time processing of vast amounts of data and the generation of insights from the entire system. Therefore, a single device is not alone: it is part of a fully integrated analytic stack.
- **Customer Experience:**
- Customers are demanding better, more responsive experiences in working with highly complex and highly connected machinery. They expect connected machinery to offer a certain level of intelligence and resilience. The changing nature of the workforce has led to ease of use being directly linked to lower overall TCO and has become a key competitive differentiator. Companies that are focused on improving customer experience need to reexamine the way in which internal and customer applications are designed and built.
- **Operational Efficiency:**
- This is one of the most important goals of IIoT. Industry consolidation, margin pressure, and regulatory forces are demanding that industrial processes be highly optimized. Additionally, the large role of software in industrial processes demands that efficiencies spread to software development, deployment, and operation as well.

Software is the key differentiator in IIoT. This software has to be built by internal application teams with the right security framework, using a robust platform, a strong ecosystem of partners, and sometimes even in collaboration with customers. Building IIoT scale applications is simply not possible without a common platform that provides a core set of infrastructural capabilities.

Building IIoT applications on an IIoT platform enables agility (for development, operations, and data science), which is a fundamental building block of creating relevant software that operationalizes insights from big data and machine learning.

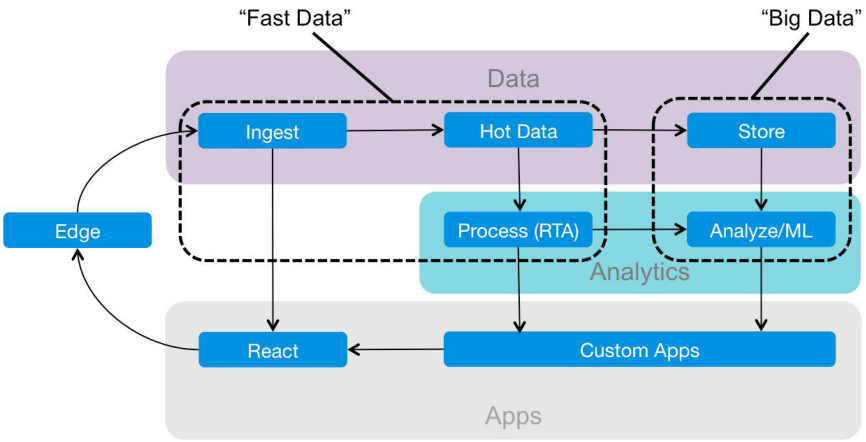
IIoT is still emerging; however, Pivotal has seen patterns emerging over several engagements. These IIoT patterns are heavily influenced by the broader patterns emerging in the data management space especially since IIoT is predominantly about data management and analytics of large amounts of data.

Reference IIoT Platform Architecture

This section examines the reference IIoT Platform architecture as seen by Pivotal. Pivotal will examine UOP use cases by using this reference architecture.

PLATFORM COMPONENTS

Unlike traditional client server applications, IIoT applications are usually built on a split processing paradigm. A fair amount of work happens in the "edge", closer to the actual devices for two reasons: (1) lowering the volume of data transferred, and (2) improved reaction times. The diagram below describes the core elements of the IoT platform architecture.



Lets examine each of the elements in detail.

Reference IIoT Platform Architecture (Continued)

Edge & Gateway

The edge is the closest to the devices being monitored and plays a very crucial part in the data pipeline. New sensors that are created or existing devices with built in sensors are the sources of data. Primary attributes that are needed of the edge are:

- **Local Near Real-time Processing Capabilities**
  The edge is not just an unintelligent source of data; it needs processing capabilities to support the main functions of the device. This includes the ability to filter data, report pattern analysis and take action locally.

- **Extensible**
  IIoT is still in its early stages and customer needs and business models are rapidly evolving. So the choice of technology and hardware platforms must accommodate this requirement for extensibility.

- **Low Cost**
  The edge is where the number of monitored devices can number in the thousands and if cost is not considered then the edge can be an expensive proposition.

- **Powerful**
  Since the edge is not just an unintelligent source of data the components chosen should be powerful enough to handle peaks of local processing requirements.

- **Two-way**
  The edge is not a one way component. While 95% of the data is directed from edge to the backend there is still a need to send messages from the backend to the edge. These are usually control messages or OTA (Over-the-Air) upgrades. However, the downlink use case has to be carefully evaluated in the context of overall requirements and security.

- **Connectivity Options**
  The edge is a complex environment with several connectivity challenges. Sensors should be able to directly connect or utilize gateways. This requires a range of protocols to connect to the backend system. Mesh and Point-to-Point networks both become necessary options. This also means that the intra edge protocols must accommodate these connectivity demands.

In many cases, especially when legacy equipment is involved, the Edge is essentially a Gateway system that collects the data from a wide number and range of sensors (the actual "edge") and does some initial processing/filtering of the data. In this case, the Gateway executes a large number of "offcloud" processes that can include anomaly detection, security breaches, and alarm conditions.

Reference IIoT Platform Architecture (Continued)

Transport

The Transport is the link that connects the Edges/Gateways to the cloud hosted Ingest endpoint. Edges and Gateways need to communicate with the cloud to leverage the full range of machine learning and updated aggregate metrics available from across the connected ecosystem. The transport mechanism is crucial in determining the amount of processing and storage that has to occur at the edge or gateway, and at what frequency is the cloud updated with the actual status of the edge nodes. Higher bandwidth transports permit more work to be offloaded to the platform hosted components, and allows the cloud to have a more realtime view of plant operations. Conversely, lower bandwidth transports require the edge or gateway to make more decisions about data processing, filtering, and reacting to anomalies.

While earlier iterations of IIoT (such as M2M) focused on dedicated wireless networks, current IIoT solutions are looking to leverage the Internet for connectivity. While this opens up concerns for security and privacy, it enables the opportunity for connectivity. Primary options for transport from the edge/gateway to the cloud platform include:

- **Satellite Communications**
  This is an expensive option, but is sometimes necessary for high importance messages. In cases where the installations are extremely remote (e.g. oceanic rigs, or offshore installations), the system might need to use satellite communications to transmit highly critical data. This implies that the local gateway system has to be extremely capable in capturing and filtering device data, and making decision locally.

- **Cellular Connectivity**
  This option utilizes data transmission over commercial cellular networks. While not as expensive as satellite communication, it still cannot be used to upload raw, unfiltered data from the gateway to the back end: the overall data quantities are far too great to make that economical. This case also requires a capable gateway system that can determine what needs to be filtered and what needs to be transmitted; the gateway, in this type of connectivity, will also apply a fair amount of business logic to determine appropriate alarm conditions and other important information.

- **Wired Internet**
  Landbased installations most likely have a wired Internet connection to the facility. This is the most economical option. Some customers may be willing to allow the use of their Internet connection to transport data from their on-premises devices to the cloud backend. While this has the potential to lower costs, it also could bring up issues with Firewalls and other network items. This would limit the choice of protocols (NAT Traversal) and as a result reduce the overall flexibility. However, this can provide a near real-time view into the plant's operations and can provide the most up-to-date information to train the machine learning models in the cloud.

- **Batch Transport**
  In the case of disconnected systems, it is possible that data will be spooled at the edge/gateway for a period of time and then will be transported via physical media to the backend at regular intervals. For instance, an offshore oil rig might have satellite connectivity for communicating catastrophic events, but may dispatch disks of data via ship or helicopter to be uploaded to the backend from a landbased facility. This is the highest latency process and provides data to the cloud mostly for long term forensics and large scale machine learning.

Reference IIoT Platform Architecture (Continued)

Regardless of the actual means of connection from edge to backend, the communication protocol between the edge and the backend needs to be resilient to poor connectivity, low bandwidth, low power; two-way connectivity and inherently handle retry logic to keep clients simple. It must also provide a high level of security for transmitted payloads. These are less significant for Batch Transport use cases.

Ingest

The firehose of data from all sources ultimately converges at the ingest system. It is a critical piece of the overall platform since this is where all the payloads from all the transports are delivered. This layer needs to have the following attributes:

- **High Availability**
  The Ingest endpoint needs to be highly available: it cannot be down due to server crashes, hardware failure, or scheduled downtime. Globally distributed devices are continuously sending data in, and they cannot get a "not available" signal. The ingest endpoint must always be operational to accept input from edges/gateways, and should have the ability to spool information if other systems downstream from it are experiencing outages or downtime.

- **Scalability**
  The Ingest layer needs to be easily elastic to accommodate for both planned and unplanned spikes in incoming data. Any scale-out design in software should not be encumbered by the choice of the underlying hardware platform. The scale out nodes of the ingest layer should start up quickly and require minimal amounts of bootstrapping to become functional.

- **Stateless**
  The ingest nodes should be stateless so that horizontal scale can be easily achieved without a lot of overhead around managing routing for sessions. Stateless systems are an architectural best practice for scale-out tiers.

- **Asynchronous, Non-critical-path**
  The nodes in the ingest layer should be as lightweight as possible. A fire hose processor works best when it does "just enough" processing. Typical activities done here are decryption, authentication, very basic message validation, storage of data, and queuing the message to the next step. Keeping the ingest layer functionally and technically lightweight has the benefit of not having to update it very often. This adds to its ability to be highly available.

Hot Data

The ingested data needs to be staged in a data grid for rapid querying, and these data "writes" should be done in a manner that doesn't interfere with the ingestion pipeline. These data stores need to exhibit the following attributes:

- **High Speed, High Concurrency**
  The data grid needs to be a high speed, low latency system so that ingestion processes are not waiting for data writes to complete. Also, the data grid needs to accept write operations from potentially millions of edge nodes per second (high concurrency).

Reference IIoT Platform Architecture (Continued)

- **High Availability**
  This tier represents the "current state" of the system and is used by multiple services to query and coordinate their works. It is essential that this tier utilize techniques of data partitioning and redundancy to keep the total amount of data highly available even if individual nodes exhibit temporary failure.

- **Scale Out Architecture**
  The data grid needs to adapt to the ever growing number of edge devices, and also to the increasing amount of data each device will be transmitting over time. These stores should be able to adjust their cluster sizes transparently and without downtime.

- **Event Processing**
  The data grid needs to apply business logic to inbound messages as they arrive and trigger listeners on an event driven basis, as opposed to frequent "polling scans" of data tables to see if certain conditions are being met. It should also be able to keep aggregate measures of inbound values, and provide transactional counters for certain types of events.

- **Flexible Schemas**
  The classes and generations of devices at the edge can change over time. Diverse plants can have diverse types of devices that transmit data in different formats. The data grid should provide means to allow converged storage of these diverse data structures and not have rigid schemas that require a lot of data schema design up front.

In-Memory Object Data Grids are tools of choice for this tier, as they provide high speed and high concurrency due to their fully RAM-based nature, high availability via sophisticated data partitioning techniques, scale out architectures by dynamic cluster management, and event processing through triggers and continuous queries on inbound data. Since these are object stores and not strict schema stores, they can handle a larger variety of data structures without fundamental changes to the storage schemas.

Store

The economics of storage have changed radically in recent years: the advent of scalable, distributed storage on low cost commodity machines has made it possible to store vast amounts of data for long periods of time in a highly resilient manner. Indeed, the large social and mobile networks of today would not exist were it not for the invention of these scale-out commodity storage platforms. These storage platforms are ideal for IoT, since the primary attributes of the archival storage tier are:

- **Ability to Store Everything**
  Every byte of data coming in from the ingest layer should be stored in its native form. This is usually put in an HDFS store that has a very low cost profile and is easily scaled for more storage capacity.

Reference IIoT Platform Architecture (Continued)

- **Tiered Storage**
  The needs for data processing and analytics vary widely in terms of volume, velocity and latency of data retrieval, depending on the use case. So any storage for IoT has to be in a "tiered" fashion. While the large volume of data is handled by HDFS, the high velocity and low latency need to be handled using potentially SSDs backing the HDFS, and most certainly require some form of in-memory technologies that front the archival store. DAS, SSD and Inmemory are the various forms of tiered storage.

- **Storage and Access Paradigms**
  Traditionally, data is stored in a relational format whilst being retrieved using SQL as common mechanism. However, IoT data cannot generally fit the rigid requirements of a traditional SQL schema. This means that the relational format may inhibit new IoT data models. Use of NoSQL technologies helps ensure that as new devices and business models emerge, the relational data model is not inhibiting innovation. IoT requires flexible data structure storage mechanism that can supply "schema on read" instead of "schema on write".

- **Massively Parallel: Move Code, Not Data**
  The multiple petabytes of data that accumulate with IoT ingestion cannot be scanned with traditional "row at a time" systems common in business intelligence platforms. Data will accumulate faster than these systems can process it. Also, a lot of time and bandwidth will just be spent in shuttling data from the datastore to the processor core. The new generation of IoT scale data storage systems allow for massive parallelism: they execute queries in the data storage nodes themselves, and collate the results. This allows small pieces of code to be moved quickly to where the data is, and large amounts of data can be processed in a fraction of the time it would take a linear system.

Traditional data warehouse technologies, using highly proprietary systems and costing tens of thousands of dollars per gigabyte of storage are wholly inadequate and unfeasible for IoTscale storage. With potentially petabytes of storage required, these systems are not economical, and with rigid SQL type storage, these systems are not flexible enough. Also, since traditional data warehouses provide highly linear SQLstyle querying, they are not suitable for the exploratory machine learning and data science that is essential in extracting value from IoT data.

Process and Analytics

The key difference between traditional online applications and the next generation IoT applications is the ability of the system to react in near real time to information as it flows in from the edge. This processing has two primary components:

- **Real Time Analytics (RTA)**
  This process is applied to the data as it comes in; it inspects each new payload and applies some form of categorization or classification to it, and triggers some downstream process based on a model or rule that the system has around processing of these messages. This is a fast and low latency process.

Reference IIoT Platform Architecture (Continued)

- **Machine Learning (ML)**
  this process is applied over large sets of data at periodic intervals (batches) and it helps create the models/business rules that will be used by the real-time analytics system for decision making. ML is the application of statistical methods and predictive algorithms to large sets of historical or source data to create finely tuned models that can be applied to new data as it comes in. This allows for customized models to exist on a per device, per plant, and per use case basis.

IoT platforms are ones that can specifically take the models developed via machine learning and "feed them back" into the real time analytics system so that new and updated learning can be continuously applied to new data without scheduled downtime or code changes to any tier.

Primary Considerations for RTA and ML

- **Reliable Queue**
  Synchronous architectures result in tightly coupled, brittle, and resource intensive systems. Asynchronous "actor" style processing and programming models are therefore commonplace now. The heart of an asynchronous processing system is a reliable queuing infrastructure. The power of queues enables messages to have multiple consumers, each operating either independently or in collaboration using the queues as a place to collaborate in conjunction with the data store to exchange data. The queues are distributed to enable highly available, high throughput data processing.

- **Distributed Event Stream Processing (DESP)**
  One of the consumers of the queue is usually a stream processor. Stream processing performs many tasks like counting, filtering, aggregating, joining etc. Since IoT is an edge computing paradigm, a fair amount of stream processing happens locally. Stream processing for IoT in the backend is usually looking at much longer windows of time and it also correlating events across many nodes in the edge.

  Example: Inspect the output from temperature sensor for long periods of time to watch for increasing trends that may trigger a threshold if the pattern continues. Another example could be looking at two similar sensors at different nodes in the edge and comparing the two to watch for alerts. This way a user is not specifying thresholds but the IoT system is smart to look for patterns automatically.

- **Complex Event Processing (CEP)**
  CEP is looking not just at temporal patterns from one stream but actually at multiple streams. Traditional CEP solutions have not been built on a distributed paradigm making them not scale well for large volumes of data. Recent developments have been in combining CEP solutions on top of distributed stream processing solutions. In the realm of IoT some amount of CEP needs to happen at the edge and with the backend again looking at larger time windows and data from across many nodes in the edge.

  Examples of this are looking at combination of temperature and humidity sensors that also could be across multiple nodes in an edge.

Reference IIoT Platform Architecture (Continued)

- **Business Rules (BR)**
  Business Rules engines tend to be much simpler than DESP and CEP solutions. While business rules engines can be deemed redundant with advanced DESP solutions, the current crop of DESP solutions seem to slow down as more rules/facts are layered on them. This could be an issue in IoT architectures so there could be a need for a RETE algorithm based robust BR engine to be integrated with the DESP solution.

- **Data Storate (Revisited): Colocation for DESP**
  For DESP, CEP, and BR engines to be effective, they all need access to data as they perform specific actions. Some of the data lies in the stream itself but additional data needs to be joined/referenced to take actions.

  For example as sensor data comes in for a device it may become necessary to look at additional details about the device (for example; specs etc) to take the right decision. Customer specific data may also need to be referenced. As streams of data are being processed very fast the data access also needs to need low latency high speed. This calls for an in In-Memory Data Grid that stays colocated with the DESP system. Ideally the data in the data grid is partitioned to ensure that all data requests from the DESP are local only and not resulting in expensive network calls.

- **Large Scale Data Analytics and Machine Learning**
  A majority of the IoT platform components described above pertain to near-real time processing. As data volumes grow, large scale analytics and ML will be required. The idea is to build a ML model via training, based on large data sets and inject the built model back into stream processing. This injection would happen on both sides: Edge as well as the backend. This is classic model scoring just that it happens twice with different scope applied at both places.

Applications & Business Models

This is the services component of an IoT architecture. It is where applications are hosted for deployment and are managed with an automated system that guarantees their uptime and day-to-day operations.

- **Business Applications, Services, APIs and PaaS (Platform as a Service)**
  All the rich IoT processing semantics and analytics are insignificant without services, APIs and applications that expose the rich data pipeline to various stakeholders. These IoT services/apps come in different shapes and sizes. For example, some are API only without a User Interface (UI), while some are rich dashboards/portals.

  Microservices are an emerging trend in application development and help improve developer productivity via cleanly defined decoupling between various interconnected pieces of business logic. Microservices play well within the IoT space as small teams of developers can work on services for the different types of devices/sensors.

  A PaaS like framework allows for an agile way to stand up these myriad IoT applications and services. In fact, the rise of Microservices based architectures almost mandates the use of a PaaS to dramatically improve dev/ops ability and agility.

Reference IIoT Platform Architecture (Continued)

- **Operationalizing Insights**
  The application tier is the component where insights from realtime analytics and machine learning are operationalized into functioning business logic. This tier allows RTA and ML to be offered as a value added service to customers via APIs and dashboard applications. This tier can also host applications that conduct exploratory data science to assist with the development of machine learning models.

- **Rapid Application Development (RAD) Tools**
  To help accelerate IoT efforts one could leverage RAD tools and off the shelf IoT frameworks that have some of the components that make up the Ingest, Store, Process/Analyze/Apps mentioned above with varying levels of maturity. While RAD tools may seem like a quick and easy way to start, it is hard to sustain them longterm. RAD tools work on a robust application development platform such a PaaS. They not only inherit all the benefits of the core PaaS platform they also extend using robust enterprise class code

- **Multi-Tenancy**
  The application tier must be able to host applications that serve multiple potential external customers. For instance, in an OEM model, the manufacturer may choose to provide valueadded services by providing custom or customized applications for a particular customer. The OEM might also host the customer's proprietary applications as a means to get nonessential services off the customers' hands.

Business Model Evolution

As OEMs move towards evolving their business models from onetimesales to longer term subscription services, the application and software component of the IoT platform can be a key enabler of this new model. It can allow OEMs to develop, deploy, scale, and manage new products quickly, and can increase the value of their existing edge hardware by adding intelligence from the cloud and making it uniformly available to their customers.

A Platform layer (PaaS) can help OEMs transition to being software driven companies that differentiate themselves substantially in the marketplace by providing highly value-added services on a purely cloud hosted model, i.e. it gives OEMs the ability to add a SaaS component to their core hardware offerings.

React

The react system transmits instructions back to the edge/gateway. Depending on the use case, the react system can be close to the edge (or even be part of the gateway).

- **On-Site React**
  This is often found in highly secure industrial IoT applications where outside instructions cannot be fed into the Process Control Loop (PCL). The react in this case is part of the edge/gateway hardware and generally does not change often. Updating the react requires a hardware change. This is the fastest, most immediate reaction system, but can also be the most inflexible.

Reference IIoT Platform Architecture (Continued)

- **Remote React**
  This is when the cloud directly sends instructions about state changes or information about relevant events to the edge/gateway. The edge/gateway receiving the update may or may not be the same as the one that originated the event. For instance, a motion sensor might trigger an update on a gate actuator somewhere else.

  In some cases, the react can be an informational message, such as a mobile push notification that is transmitted to one or potentially hundreds (plant personnel), thousands (company-wide alerts), or millions (city-wide notifications) of mobile devices. For instance, a catastrophic event at a plant might trigger a large scale warning alert that is transmitted to city dwellers warning them of the impending danger.

- **Disconnected React**
  The react in this case is not updated automatically from cloud based instructions, but is managed by human intervention. However, the update packages that human operators apply to the onsite react can be built in the cloud based on the data transmitted from the plant. This has the advantage of being fast, as the react is part of the edge/gateway, but is also flexible, since it is software driven. With the correct security protocols around instruction payloads, etc., many of the security concerns with remotely provided instructions can be alleviated.

The react system is crucial for "closing the loop" on the ingest, analytic, and business logic pipeline. It updates the behavior of the edge/gateway with insights gleaned from aggregate analysis of all data coming into the platform.

Security & Trust

TRANSPORT LAYER SECURITY (TLS)

TLS secures the connection between two information exchanging parties so that someone intercepting the information in transit cannot determine the contents of the information. However, it does not prevent bad or spurious information from being sent from one party to another. In other words, TLS is not a means to establish trust between two parties.

TLS is most commonly used on the internet via the HTTPS protocol. It depends on two parties with certificates issued by "well known" certificate authorities exchanging their public keys at the time of establishing a secure connection. Upon receiving an inbound connection request from a client to a server, the server responds with its certificate: the client then checks the certificate's issuer from its database of known issuers, and if the issuer exists in the database, the client accepts the certificate and uses the public key in it to send a symmetric key to the server. The Server and Client will now communicate with each other by encrypting traffic via this symmetric key. The server still does not know who the client is. The client can present its credentials to the server after the initial handshake for the server to vet the authenticity of the connecting party.

MESSAGE LEVEL SECURITY

TLS only secures the tunnel from the outbound edge of the client to the inbound edge of the ingest system. In the case of Industrial IoT, where multiple separate components within the edge perimeter are generating messages, a malicious party can start to generate messages that mimic various process control systems and these can be then transmitted to the server. The server has no way of knowing whether the messages it is receiving via a TLS pipe have been genuinely created by process control units of interest or are simply fake messages.

Each component that is generating messages should generally have a unique component ID and signing key assigned to it. The messages it produces should then be HMAC signed by the signing key. For instance, a log message can look like the following (using JSON format. Other formats like XML can also be used):

```
{
 "componentId":"HQWLKE1341KH",
 "signature":"AZPUQEUHGY5O34HB48G5389ERHAG3JQ4",
 "message":"<<base 64 encoded text of actual message>>"
}
```

The benefit of a signature for each log message is that it allows the receiving end to lookup the signing key from its database, based on the component ID and verify that the log message has indeed been sent by the component. Also, to prevent replay mimicry, the encoded message can also contain the timestamp, forever incrementing, so that the receiving end can check if the message was recently generated, and discard the message if it is older than a certain defined period.

Security & Trust (Continued)

TAMPER-PROOF AUDIT LOGS

One very easy way to create highly tamper resistant audit logs is to use "hash chaining". It is a concept that has also been adopted in systems like the Block Chain for crypto-currencies. The idea for hash chaining is simple yet powerful: the HMAC signature of one log message is based on the contents of the log message and also on the HMAC signature of the message log immediately prior. Therefore, tampering with any one message involves changing every single message that comes after it. Also, the tampering entity needs to have the signing key of message originator.

Therefore, in the above example of a message sample, if the base 64 encoded payloads includes the signature of the prior message (it doesn't have to be from the same component id, if fact it is better that it isn't, not at least on the central recording server) then highly tamper resistant audit logs can be created.

KEY MANAGEMENT INFRASTRUCTURE

For effective implementation of Message Level Security and Anti-Tamper Logs, the system must have a robust key management infrastructure. The key management infrastructure should:

- store keys securely when they're at rest
- cache keys for fast retrieval by ingest endpoints to decrypt message and/or validate their signatures as they stream in
- rapidly provision new keys and deprecate invalidated keys at large scale
- be highly available and scalable to address demand

Various platform components can be used to create this system so it can effectively broker key information to the microservices that require payloads to be validated.

SECURING THE EDGE

Security within IoT (SIoT) is an important & complex topic as not only critical device data is now being potentially sent (or spoofed) over the public Internet, but also now those devices can potentially be controlled from anyone with an Internet connection.

Device authentication is no different from user authentication except in two aspects. The devices need to be securely told their password in advance, and devices authenticate themselves more frequently, essentially on every call to the server. The backend authentication has to accommodate both of those aspects: this implies credential distribution, and large scale credential checking as each call requires authentication (and devices make a lot more calls than humans can). In the case of IIoT, the devices are generally communicating mostly with the gateway that will collect and transmit a filtered set of information. However, in this case also, the devices need to include authentication information that verifies their identity and the correctness of their message. This is to prevent malicious actors (human or software) from spoofing the output of a component.

Security & Trust (Continued)

In the context of IIoT, the edge/gateway is a fairly capable system: it generally encompasses traffic from the entire set of equipment in the plant, or large parts of it. It can therefore include sophisticated encryption techniques to communicate up to the cloud backend, and also verify the instructions it receives from the cloud backend (perhaps, for example, as part of a react instruction).

Edge security should depend on more than just passwords that are sent in as headers with a message (basic auth). When communicating with the cloud, Edge security should incorporate techniques such as certificate exchanges, message hashing, and embedded nonces to increase the security profile of the network traffic.