White Paper

The Cisco Edge Analytics Fabric System

A new approach for enabling hyperdistributed implementations

Contents

Executive Summary

This white paper describes a software system that provides the framework for the Internet of Things (IoT). It dramatically simplifies the task of creating sophisticated IoT systems.

This system's key capabilities include:

- A framework for edge and fog processing.
- High performance.
- Reusable microservices for collecting data from, and providing control over, devices and machines, as well as processing the data prior to delivery to its destination.
- Different options for reliable transport of data through the system, encompassing both batch and real-time streaming options.
- Flexible mechanisms for integration with IT systems, reporting, and analytics.
- An architectural framework to extend fog processing to multiple tiers: east–west (fog to fog) and north–south (hierarchical processing leveraging network topology).
- Easy-to-use GUI tools to simplify development, deployment, and operation for all aspects of the system.
- A pervasive control paradigm and flow of information back to microservices, devices and machines for management, control, optimization and specific actions.
- A completely open and polyglot system, where third parties can provide devices, processing storage, software modules, analytics, applications, or any combination thereof.

This is the technology that makes IoT approachable, and leads to much faster industry adoption of the vision of IoT.

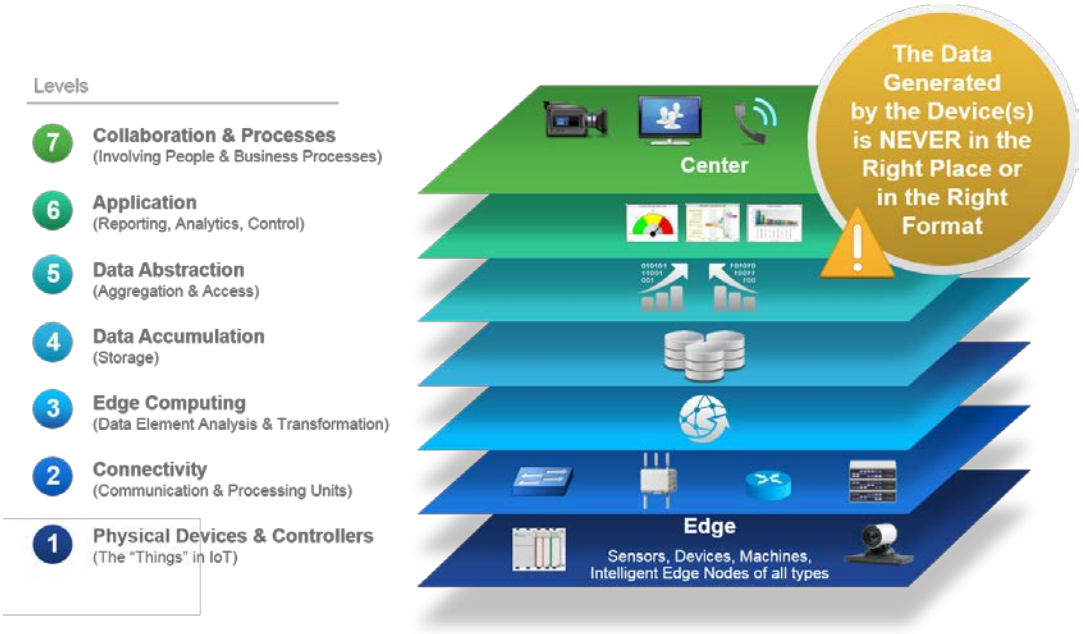
Introduction

IoT is a combination of data generating devices, communications, and data processing that effectively leverages machine generated information for business advantage, including analytics and integration with existing IT systems.

Figure 1 is a conceptual model that depicts the different parts of an IoT system in a layered approach. This illustrative model has been adopted by the IoT World Forum as its reference model[1], with the goal of taking the first step in converting from a concept to a tutorial diagram.

[1] Building and Internet of Things: An IoT Reference Model, http://www.slideshare.net/Cisco/building-the-internet-of-things-an-iot-reference-model

Figure 1. IoT World Forum Reference Model



IoT is something that the industry has never seen before. It has been described in numerous ways, such as:

- The next generation of the Internet.
- An evolution of the networking industry enabling everything to become interconnected, with IoT focused on machine communications.
- The bridging of operational technology (OT) and information technology (IT).
- The capture of information from machines that enable analytics that were never before possible.
- Pervasive control of highly distributed actuators.

Despite the different perspectives on this industry movement, one thing is very clear: IoT requires infrastructure software unlike that which currently exists. The handling of machine-generated data, combined with the scope of very large-scale systems (in terms of geography, number of data generating devices, diversity of manufacturers, frequency of data generation, and overall data volume) including machines, IT, and analytics, is new to the industry. A new form of IoT enablement middleware is required.

IoT System Basics

To meet this need, Cisco has created a new type of software system that complements its advanced networking and computing hardware. This system is driven by a complex set of requirements:

- Flexible: It handles a wide variety of IoT needs.
- Repeatable: Once configured to handle a manufacturing cell, an oil well, a parking lot lighting system, or other systems, it can be easily replicated.

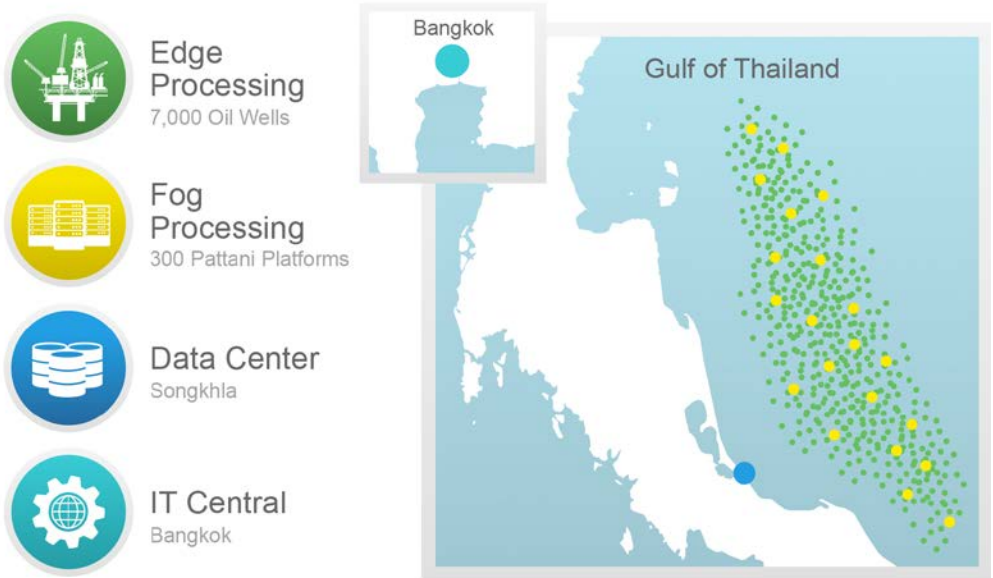
- Scalable: Some IoT systems generate more data than Big Data. Many terabytes of data can be generated quickly. The ability to effectively handle this load requires sophisticated engineering.
- Robust: Many industrial IoT systems must be monitored, managed, secure, and highly available. Using a well-tested system, a validated design, and a proven methodology yields better results.
- Control and action: Most industrial IoT solutions today are focused on data collection and actuation. As previously mentioned, the system described in this document accomplishes those tasks in a unique, flexible, and repeatable manner. Further, control and automation can be implemented with a pre-defined policy architecture.
- Multi-purpose: For example, the value and use of data changes over time. Initially collected data is valuable for monitoring and control where latency and responsiveness may be crucial, later analytics can create optimization and production improvements or cost savings, and in the long term IoT data may be required for compliance assurance.

The system described in this document is the infrastructure for Cisco® IoT solutions, is used by Cisco Advanced Services for custom implementations, and is available to Cisco partners and third parties as a stand-alone offering to enable them to convert their IoT concepts into reality in record time.

IoT System Flexibility and Scalability

Because customers have a wide variety of IoT use cases, the system is designed to handle complexity, as well as satisfying the needs of simpler problems. One complex example might be the need to bring data from thousands of off shore oil wells and reliably transport it half way around the world to a data center where it is integrated with multiple applications. Another, simpler example might be a supervisory control system completely contained in a manufacturing cell, with only one user.

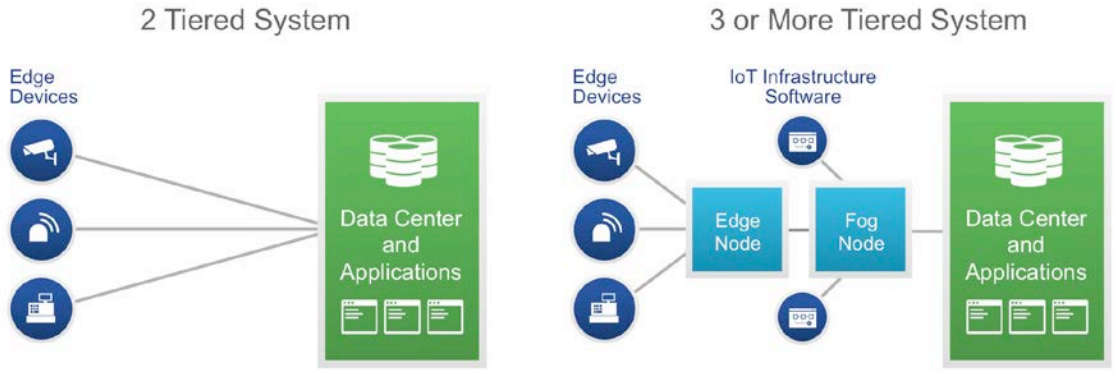
Figure 2. Oil and Gas Example for Data Processing on Platforms, at the Regional Data Center, and the Main Data Center



To accommodate the wide range of system scope, topologies, and geography, the system is modular. It can be used in scenarios where computing is executed in two, three, or four places. Processing is completed where most appropriate, and data filtering, aggregation, and compression is performed at the edge, in the fog, or at the center to optimize performance and scalability, and minimize networking costs.

The diagram in Figure 3 depicts some of the basic options.

Figure 3. Data Computing Tiers: Optional Fog Nodes for Better Performance



Dealing with Complexity

Implementing a solution quickly, with a solid scalable architecture and in a minimum amount of time, is a significant accomplishment. To accelerate the widespread adoption of IoT systems, the time required to create complex systems must be reduced from years to weeks. The system contains not only a sophisticated infrastructure that meets the requirements previously listed, but also a set of GUI tools that replace coding, dramatically decreases skill requirements, dramatically shortens the time to operation, and dramatically increases the quality of the resulting system. These tools achieve several objectives:

- Ease of development and operation provided by these tools is a major improvement over pure open source products.
- Flexibility to adjust a solution quickly to changing needs is critical.
- Incremental complexity of a growing system that changes over time and is contained through the use of tools that support incremental changes and allow monitoring of the entire system.

Example development tools are depicted in Figures 7 through 11.

IoT System Structure

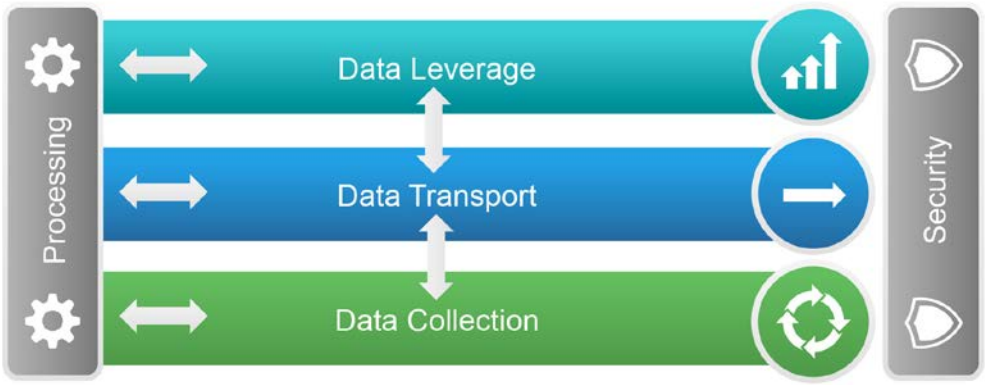
Different scenarios require different approaches. The system is built with different hardware and software modules. With modularity in mind, the system is divided into different functional subsystems (Figure 4):

- **Data Acquisition**: Interacting with devices (many times through a non-IP protocol), receiving data from the device, and handling the initial processing of the data.
- **Data Retention**: Data can be stored, transformed, and retrieved on demand from all locations within the system.
- **Data Transport**: Reliable delivery and transformation of data, assuring that the data is delivered to the destination application and/or data repository guaranteed, exactly once, in order.
- **Data Processing**: Trade-offs are made regarding the location of the processing of telemetry data obtained from devices. In some cases, edge processing is required. In other cases, aggregation of data from a wide variety of sources is necessary. And in other cases, the data cannot be fully leveraged until it reaches a data center or a cloud. All options are available with this system.
- **Data Leverage**: Enabling the consumer of the data to leverage it quickly and effectively. Examples include: reporting or business intelligence; creating an environment for analytics; responding appropriately to an event; or providing data to existing IT systems. Each of these four categories requires different data manipulation. A single system may encompass more than one of these categories, and they may occur in different tiers.
- **Control**: The system must be secure, extensible over time, changeable while running, and it must be monitored as it operates. All of these aspects are implemented or enabled by the system.

The modularity of the system provides independence and decoupling between the different subsystems, and adds great value to the customer through the lifetime of the system by allowing for growth using new modules from Cisco or third parties, or combining Cisco software with other third-party software as needed. This is an open system supporting both proprietary and standard interfaces, with metadata management throughout the system.

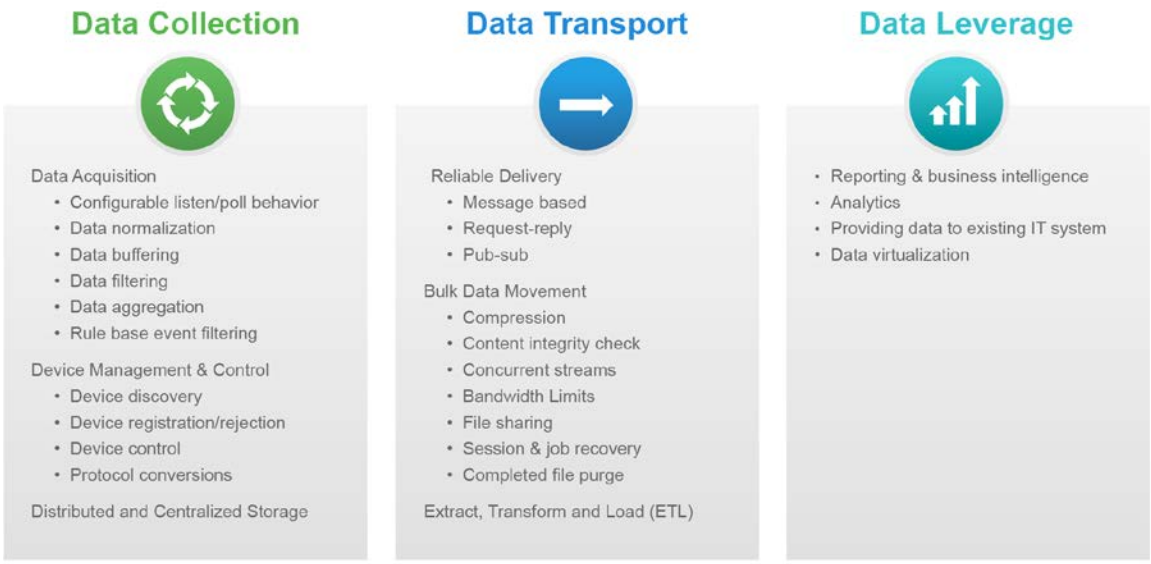
This modularity distributes the processing and logic throughout the system. It can occur at one or more edge nodes, in a fog node, by data center applications, or in a cloud. The software in this hyperdistributed system is packaged as microservices. The system provides a uniform architecture to assure interoperability between microservices, common data encoding across all microservices, and different modes of communicating between them. More details about what defines a microservice, the messaging system between microservices, and the tools for creating microservices are provided in following sections.

Figure 4. Cisco IoT System Capabilities



For each of these functions there are a rich variety of options and processing modules that can be invoked to customize the system to the customers' exact needs (Figure 5). These options are provided by independent software modules that connect to each other using a common architecture, providing the flexibility to add, modify, and upgrade as required over time. This openness allows for many ecosystem partners and developers to add functionality and enhance the system.

Figure 5. IoT System Example: Functional System Processing



Microservices

Processing is distributed throughout the system. For example, data from a device could be inspected, filtered, analyzed, and transformed in an edge node. Eventually, that data (or a derivative of it) can be used for analytics in

the cloud. We therefore need a uniform way to package software modules that perform one or more functions, but may or may not qualify to be called a complete application. These modules are called microservices.

Some key motivations for running microservices at the edge include:

- **Reduce the data at the edge**: Filter and reduce data before being sent to a cloud or on-premises data center. Sometimes this is required due to massive volumes of data and limited network bandwidth.
- **Analyze the data at the edge**: If the consumer of the data is local to the data generating device, such as in a factory control-loop feedback system, analysis of the data is best done local to where it is generated.
- **Time series capture**: If time precision is required as to when the measurement or sensing occurred, the data is typically captured, time-stamped, and stored as close to the edge as possible, usually in a historian database.

Some microservices run at the edge, such as in an edge router or an IoT gateway. Some run in a fog node, perhaps to aggregate data generated by many edge processing nodes. And some run in a data center or cloud. Regardless of location, all microservices in the system need to have certain attributes in common, including:

- Interaction with each other through a common communication system. This system is called a message router system and is described in detail in the Message Router section of this paper.
- Adherence to common data structures and encoding, so as data is passed between them, the value of the data points remains unchanged.
- Share certain capabilities. For example, if one microservice publishes data on a certain topic, all subscribers to that topic will receive copies.
- Compliance with a common security system.
- Able to be monitored in the same way so that the entire system can be monitored.
- Common deployment model, so the complexity of the system grows at a much slower rate than the size of the system.
- Common infrastructure to interact in the same way to system GUI tools that are provided by Cisco as part of the system.

The framework for building and deploying compatible microservices is a major aspect of Cisco's IoT system.

Microservice Implementation

Cisco's microservices framework is designed to be extremely flexible:

- There is no size or structural requirements for a microservice other than its interface to the message router system. A microservice could be a few lines of code, an interface to a database, a user interface, or an application of any size.
- Microservices are not limited to specific functionality. A service could be a Modbus communication service, a data transformation service, a data analytic service, a bridge to move data in or out of a data storage system, or a complete application.
- Services can be developed in a variety of languages, such as C, C#, Java, JavaScript, Ruby, Python, and Dart.

- Services can be developed using familiar software development design patterns.
- All services can interact with each other and pass messages between them.
- Communication between services is typically loosely coupled (asynchronous), enabling unlimited scalability.

The following sections describe the key design aspects of service development.

Service Description

A service description captures the details of service capabilities in a declarative fashion. This includes the RPC/Pub-Sub endpoints, associated message schemas & models. The idea behind this is multi-fold. One, we would like to have certain common service infrastructure (registration, invocation etc.) aspects, while hiding the service implementations from the framework implementation choices (router implementation, for example). Second, this helps service developers focus on one interface to expose their capabilities. The framework and tools can use this service description to generate various interfaces.

Service Dependency

Services can indicate other services they depend on. There are two kinds of dependencies taken into account. One kind is a hard dependency. In this case, a service cannot function unless the dependent services are in place. Typically, this kind of dependency is limited to core services that the framework provides. The second kind is a soft dependency. This means that service startup is not affected by the dependency. Service A can start even when service B (on which it depends) is not running at that instant. To locate a service B, service A could leverage the discovery capabilities provided by the message router.

Service Development

Services can be developed in a variety of programming languages, as previously mentioned. Various language-specific software design kits (SDKs) are provided, along with tutorial information, code samples, and best practices. This provides the services with a common infrastructure.

Common Microservice Infrastructure

The SDK provides the following capabilities to service developers:

- **Service container**: This is a logical container to host the service code. It provides the necessary bootstrapping that enables the service to establish the connection with the message router and take the service description and register the service capabilities with the router on startup. From a deployment perspective, services can be bundled in different ways. On smaller compute nodes, some services can be bundled into a single package and run as a single process. On larger compute nodes, services can be bundled separately and run as independent processes. Service containers provide the necessary abstraction layers such that service code is independent of the deployment aspects previously referenced.
- **Service registration and discovery**: The SDK provides software to enable services to be connected and registered with the message router automatically.
- **Service invocation**: The SDK provides wrapper APIs to invoke other services, using either remote procedure call (RPC) and/or publish/subscribe.
- **Logging and metrics**: A common infrastructure is provided to collect various service specific metrics and expose information to management layers in a consistent way.

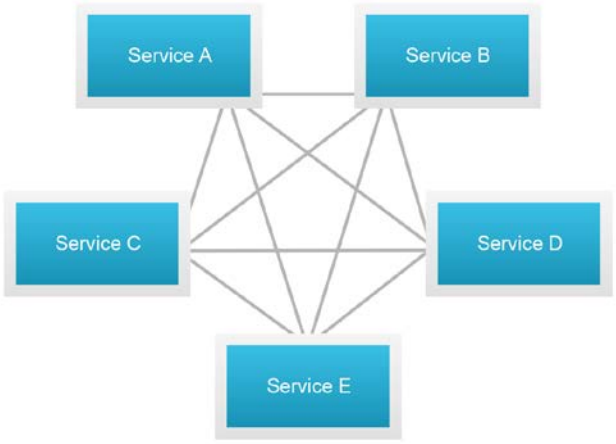
Microservice Lifecycle Management

Since the microservices are interdependent upon each other, management of the lifecycle of the associated services is required.

The Message Router

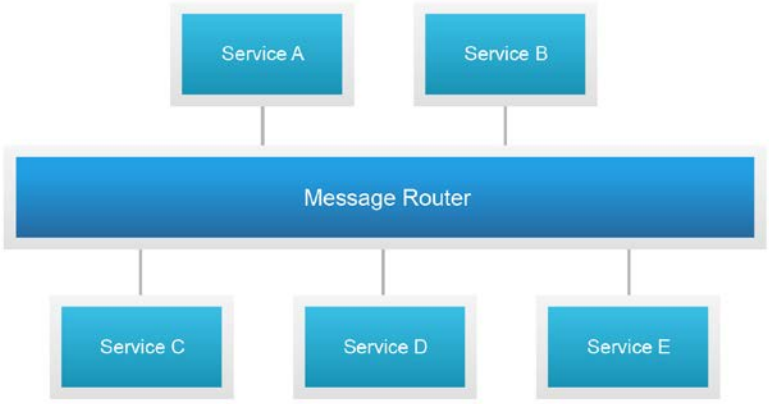
When a system is composed of one or more components, those components must communicate in order to complete a process. Without an intermediate message router, a component would directly communicate with another component by a direct connection(s). These connections between components tend to become component-to-component interaction specific. As the system and components evolve, these connections result in a full mesh topology, with every component tightly coupled with every other component, as shown in Figure 6. This impedes scaling and manageability.

Figure 6. Components in a Full Mesh Topology



Introduction of new components into the system results in more connections and more component-to-component specific interactions. Over a time, components become interdependent and the system as a whole becomes brittle and rigid, making it difficult to evolve. The alternative, which provides better scaling, is a message router. It provides a messaging infrastructure for different services to communicate with each other in a common way. Figure 7 shows how using a message router results in N connections, instead of exponentially increasing connections.

Figure 7. Connections Using a Message Router



The message router enables services to be loosely coupled and evolve independent of other services. It achieves the loose coupling between services by providing the following primary constructs:

- **Messages and message exchange patterns (MEPs)**: This is the ability to define language-independent messages and exchange them using a well-defined set of patterns. These patterns can provide a richer set of interaction options than the basic transmit/receive or get/put capabilities. Options such as publish/subscribe are available.
- **Message transport**: A transport mechanism on which messages are routed between services can provide additional capabilities and quality of services (QoS) beyond those available using TCP or HTTP protocols. These are described in more detail in the Message Router Quality of Service section of this paper.
- **Control Messages**: A set of control/command messages to facilitate services to register and declare their capabilities with the router so that services can communicate with each other.

Microservices expose their service capabilities to the message router, who can then broker communications between compatible interfaces of the participating microservices.

Connectivity from the edge to the fog, data center, or cloud is always IP based. The messaging capabilities described herein are layered on top of TCP/IP.

Message Router Quality of Service

There are many possible failure modes in complex systems: network connectivity might be intermittent, bit error rates might interfere with communications fidelity, power outages or glitches might occur, hardware fails, operating systems crash, or an application might simply be off-line. A robust IoT system requires reliability beyond networking protocols and involves handshaking the information between microservices and applications to guarantee end-to-end delivery and availability to the desired service level. This is a primary requirement of the data transport subsystem.

The data transport subsystem provides protection against two categories of failure modes: failure of a message router (or its communications links); and failure of a microservice (or connected application) that is dependent upon the message router (or its communications link) to deliver information to it.

- Guaranteed delivery guarantees delivery of every message (even in the event of a network or message router failure), providing a higher level of service than TCP or HTTP. This deals with a failure of a message broker.
- Durable connection options assure that a consuming microservice or application receives every message it should receive, even if the application or its connectivity fails intermittently. This deals with a failure of an application or microservice.

Message Router Scalability

A single message router can handle hundreds of thousands of messages per second; in most cases this is sufficient. However, in a few very high volume data scenarios, multiple message routers can be clustered and share the load. With this option, load balancing, high availability (through automatic fail-over), and fault tolerance is achieved. This is an example of horizontal scalability.

There are many systems where message routing is desired in an edge node(s), in a fog node(s), in one or more data centers, and in a cloud(s). In this scenario, there may be multiple message routers. A microservice or application should not have to be aware of how many routers are in a system, or to which routers other microservices are connected, and may need to subscribe to messages from multiple sources that are being routed through different message routers. The system's message routers can be connected together and the multi-router protocol between them will cause them to act as one. A publisher of data can connect to one router, a subscriber to another, and the delivery QoS will be the same as if both were connected to the same router. Traffic can be partitioned amongst multiple brokers when the ability to vertically scale a single broker is exhausted.

Message Router IoT Features

There are numerous special features present in Cisco's IoT message router system that are not available in other messaging systems. Examples include:

- IoT systems typically involve nodes with limited processor and memory capabilities. An IoT message router can run on a very small footprint. For example, the message router described in this paper can run in less than 50MB of RAM.
- IoT systems are frequently geographically dispersed. The multi-router routing capability supports scalability in both the number of systems and their geographic dispersal.
- The communication links within an IoT system are frequently limited. The message router compresses the data as it sends it across network links.
- The physical location of devices connected to an IoT system is sometimes difficult to determine. The message routers can obtain, propagate, and provide meta-data about the devices, including their location.
- Some devices have interfaces that provide many data points. This could result in a very large set of unneeded data being transported across the network. Unlike other systems, this system supports subscription to a very fine granularity—down to individual data elements. For example, a 500 cell battery may provide 500 data points with every request. If a subscriber is interested in only three of these data

points, it can subscribe to only those points, resulting in a much more efficient use of the available network bandwidth.

In summary, enhancements to the state-of-the-art in message brokering have been made to handle the unique properties of IoT systems.

Application/Microservice Interactions

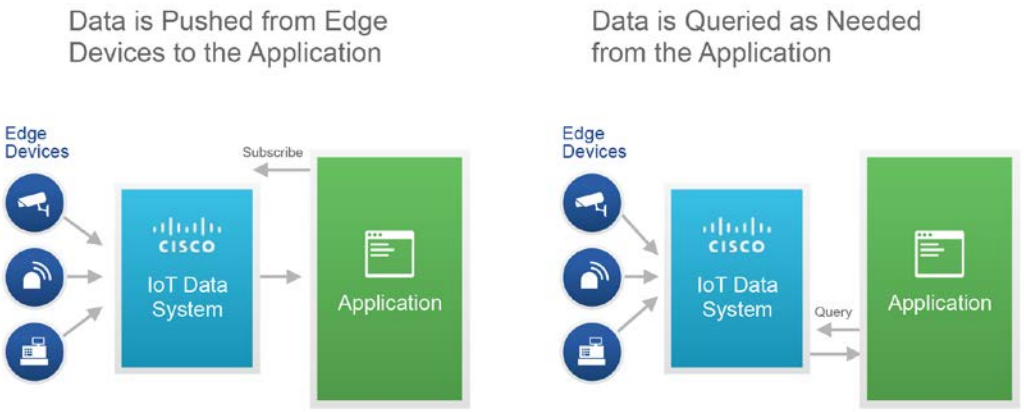
There are multiple computing models which require different approaches of the data transport subsystem. A detailed treatment of this topic is very technical and beyond the scope of this document. A simplified explanation is that sometimes the data is pushed to the application and sometimes the application pulls the data (Figure 8).

When the data is pushed through the system to the application, the assumption is made that the application can keep up with the stream of data. To use this model, an analysis is required to assure that the timing always works and that data is not lost. The data transport/storage system supports both real-time streaming processing (enabling event stream processing and analytics by an appropriate microservice), as well as periodic transfer of bulk data supporting batch processing and/or analytics on at rest data.

When the data is pulled, it is first captured from the device using the data collection subsystem. Then it must be stored until the consuming microservice or application requests it. There are options for storing the data in its interim state, as it is being transported. This is covered in more detail in the next section.

In summary, the data transport system supports multiple application models, and is easily configurable with tools, enabling sophisticated and robust systems without programming.

Figure 8. Data Flows From Devices to the IoT Data System Supporting Multiple Models



Messages

Services communicate by exchanging messages. The message router provides mechanisms to register different message types and exchange them between services.

The messages can be encoded using several options, including:

- Binary
- Text

- JSON – text based serialization
- MsgPack – binary serialization of JSON

Message Transport

Some examples of message transport interfaces are:

- TCP
- HTTP
- WebSockets
- Unix Domain Sockets

Message Exchange Patterns

Message Exchange Patterns (MEPs) define a sequence of messages exchanged between two entities in order to complete an interaction. There are two common MEPs that are of interest:

- Request–Reply: An interaction in this pattern consists of two messages, request and reply. This MEP is commonly used with RPC style applications, where a client sends a request to a server and the server responds with a reply.
- Publish–Subscribe: An interaction in this pattern consists of more than one message. An initial message (subscription request) from a subscriber expressing the interest to receive one or more messages that are published by one or more publishers. This MEP is typically used when two or more entities need to exchange a set of messages.

Microservices Decoupling

There are cases where data is generated by devices so quickly that the consuming microservice or application across the network cannot keep up. This could be a limitation in either network bandwidth or application processing performance. In these cases, the producer and the consumer of the data must be de-coupled so they can each run at their own performance level. Note that no amount of decoupling will solve the problem where the consumer/network cannot keep up with the average publish rate. In this case, the queue grows over time and eventually exhausts system resources. Decoupling/queueing provides two capabilities: the ability to overcome burst scenarios when message volumes are temporarily greater than the ability of the network to transmit them or the microservices to consume them; and the ability to provide resilience when the receiving microservice is temporarily unavailable. Decoupling is accomplished by storing the data temporarily until the consumer can pull the data (refer to previous section on the pull model). This is accomplished in the message router embedded queue, acting on behalf of all of the data generation sources.

There are alternative QoS options that can be used with the message router. If the guaranteed delivery option is used, the message router provides temporary storage (referred to as queuing) on a disk to handle the flow control of data. Persisting the data in this manner, plus logging and recovery routines in the event of a router failure or a power outage, provides very high reliability. The message router guarantees that all data will be delivered, to all subscribers, at least once, in the order the router receives the data.

This option is handled entirely within the router. Microservices are not encumbered with any additional complexity to take advantage of this capability.

Edge/Fog Database

There are several excellent reasons why you might need a specialized database at the edge to temporarily store captured data:

- The leverage of the data might also be at the edge, as in a local closed-loop supervisory system. In this case the data never needs to leave the local domain.
- The data from the devices may need to be cleaned prior to being moved away from the data generating device.
- There are cases where filtering the data is dependent upon previous samples of the data, such as in the case of filtering for outliers, where outliers are defined by a deviation from normal.
- In some cases, the data must be time stamped so that a time series analysis can be performed. Time accuracy is best achieved if the time stamp is recorded as close to the generating device as possible. An example is where a time line is created from the telemetry so that a trend (over time) can be understood. Since the devices themselves may not provide a time stamp, this is accomplished by the historian database.

IoT places very demanding requirements on this database. Examples include:

- It must support very fast insert rates. It cannot let queries interfere with insertions, since devices may not hold new data while waiting for the database to free up for an insert.
- It must be more flexible than a typical relational database management system (RDBMS) database, with regard to the structure of the data from devices. For example, if we have a battery with 500 cells, the desired database schema would have 500 columns. High column count with sparse data is typical in IoT settings, and leads to poor retrieval performance with standard RDBMS products.
- It must support standard American National Standards Institute (ANSI) SQL for compatibility with data consuming products.
- Given the volume of data present in some installations, it must process queries very quickly. The Cisco ParStream database has been measured as responding to queries against one billion rows of data in less than one second.

In a fog node, the requirements may differ. One of the typical functions of a fog node, which sits between the edge nodes and a data center or cloud, is that it has to gather and union data from multiple edge nodes, and prepare the data for northbound transmission. Thus, the database could be located in the fog in addition to or instead of the edge node. In a fog node there are unique requirements, including:

- In the event where the fog node has not already collected and aggregated the data from the edge nodes, gathering this data from multiple edge nodes upon request can be a cumbersome issue, and result in low performance. The alternative is a single query that expands to a geo-distributed query, resulting in the fog node propagating the query to all of the edge nodes (or other fog nodes), quickly executing the query at the edge nodes on the locally stored data, gathering the intermediate result sets from all relevant edge nodes with a union, and returning a single result set to the requester.

- Frequently data coming from the devices is not in the desired format for the consumer of the data. In many cases, the best place to transform the data is in the fog node. The database should have the capabilities of running data transformations, data reduction, and even data analysis, inside the database to provide optimum performance and simplicity in an easily deployable software package.

Regardless of the location of the historian database, it is implemented in the system as a special form of a microservice, and interacts with other microservices through the message router, taking advantage of the router's features.

Intermediate Processing Between Microservices

In some cases, microservices can be simplified if the system inserts processing between them. This event stream processing enhances the overall system by offloading processing from many microservices and centralizing it into a software module designed to execute streaming processing very effectively.

This processing can become complicated, and is best completed by a special purpose engine that is designed to handle all of the unique requirements, such as:

- Evaluating streaming data against rules
- Comparing streaming data with stored data (to compute trend lines or identify outliers)
- Comparing data across multiple different streams
- Executing complex algorithms against data in motion

The Cisco Connected Streaming Analytics (CSA) product can accomplish these tasks, and is easily programmable with scripting to accomplish almost any need.

Tight Coupling

As an alternative to the edge and fog processing previously described, the data transport subsystem could deliver the data directly from the device to the center (data center or cloud) where it is immediately processed by an application or stored in a repository and made available for query by an application, BI reporting tool, or analytic tool. Note that the system supports Kafka, RDBMS, ParStream, and Hadoop based repositories, the selection of which is dependent upon the format and data type, volume of data, and query response time requirements. This simpler pattern, where the device is tightly coupled with processing in the data center or cloud, is used when the amount of data is limited and protection against some of the failure modes is not required.

Development and Management Tools

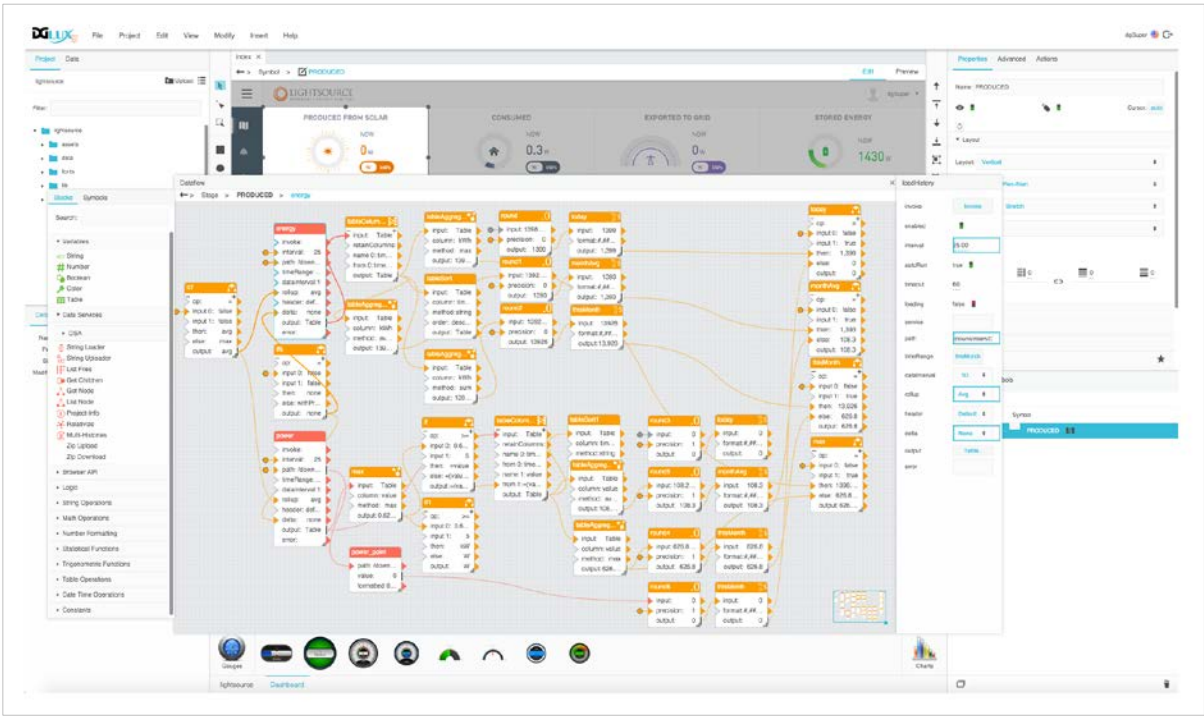
Productivity in developing the IoT system, flexibility in adjusting it after deployment, monitoring the system over its lifetime, and managing the different aspects of a running system, are all important aspects to consider. It is important that the system is designed so that exceptional skills are not required to develop and manage it.

The system contains several graphical tools that can be used in both the development and operational phases of the system (Figure 9).

Development tools include:

- A software design kit (SDK) and a development toolchain to develop microservices and applications from scratch in multiple languages
- The ability to find all of the microservices and applications connected to the system
- The ability to introspect the interfaces of these microservices
- The ability to chain microservices together, thus defining the data flow between the microservices

Figure 9. Development Tool for Specifying Data Flows Simplifies IoT System Development



Management tools include:

- Large scale central management of the lifecycle of applications, microservices, and data
- A cloud based application repository to simplify discovery and distribution of applications, microservices, and their dependent components
- The ability to manage security of the system, and determine which devices are authorized to generate data
- The ability to monitor the system as it is operating, and observe the data flow through the various nodes and microservices comprising the system

Figures 10 and 11 show some of the management tools available.

Figure 10. Lifecycle Management Tool Allowing Applications and Microservices to be Deployed to Edge and Fog Nodes

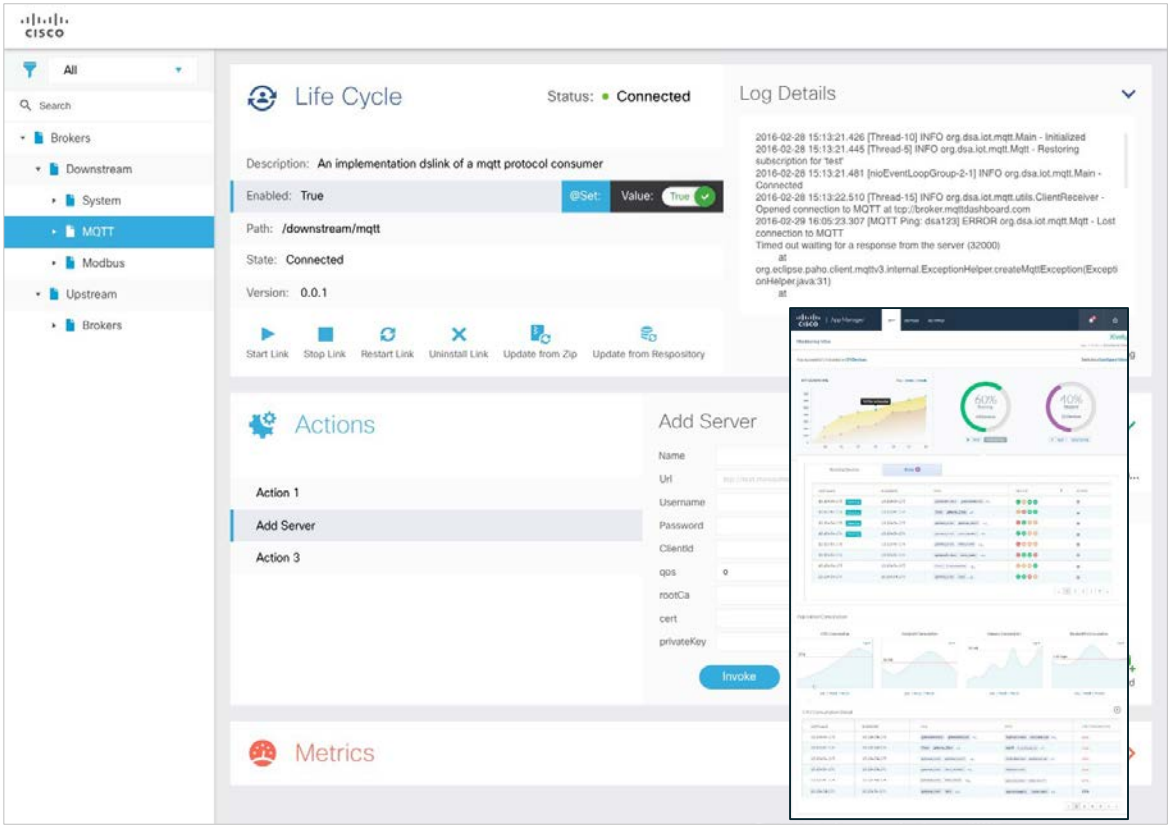
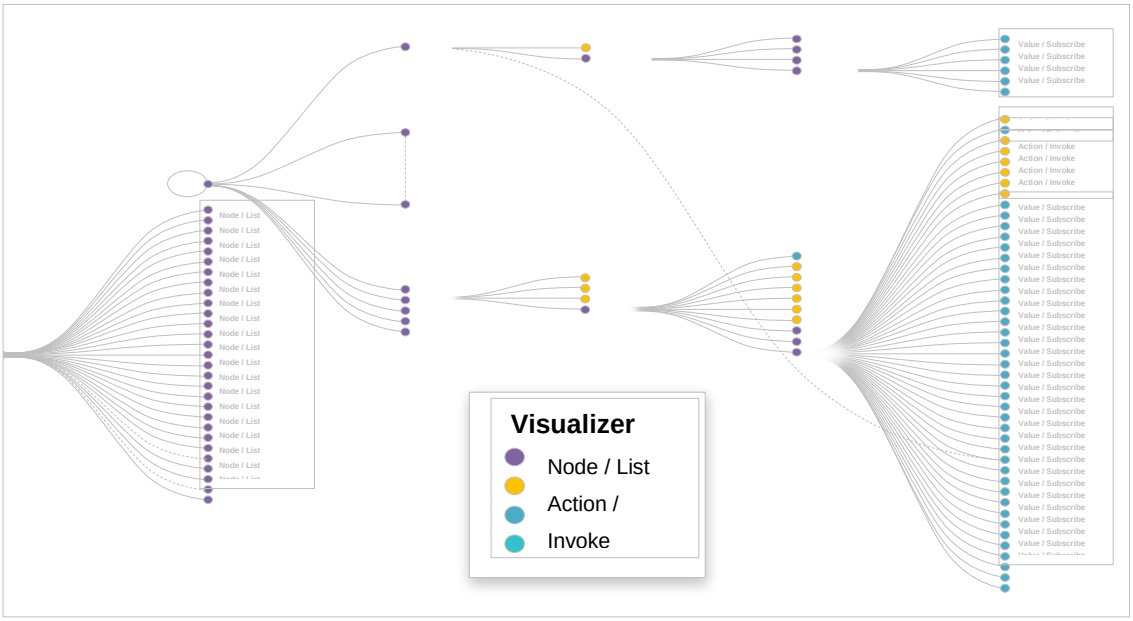


Figure 11. Operational Monitoring Tool Helps Manage the System



Data Leverage

Once the data has gone through edge processing and has been reliably delivered to a destination where more serious computation can occur, optional complementary products from Cisco assist in preparing the data, transforming and storing the data, and presenting it in the optimal way for it to be used.

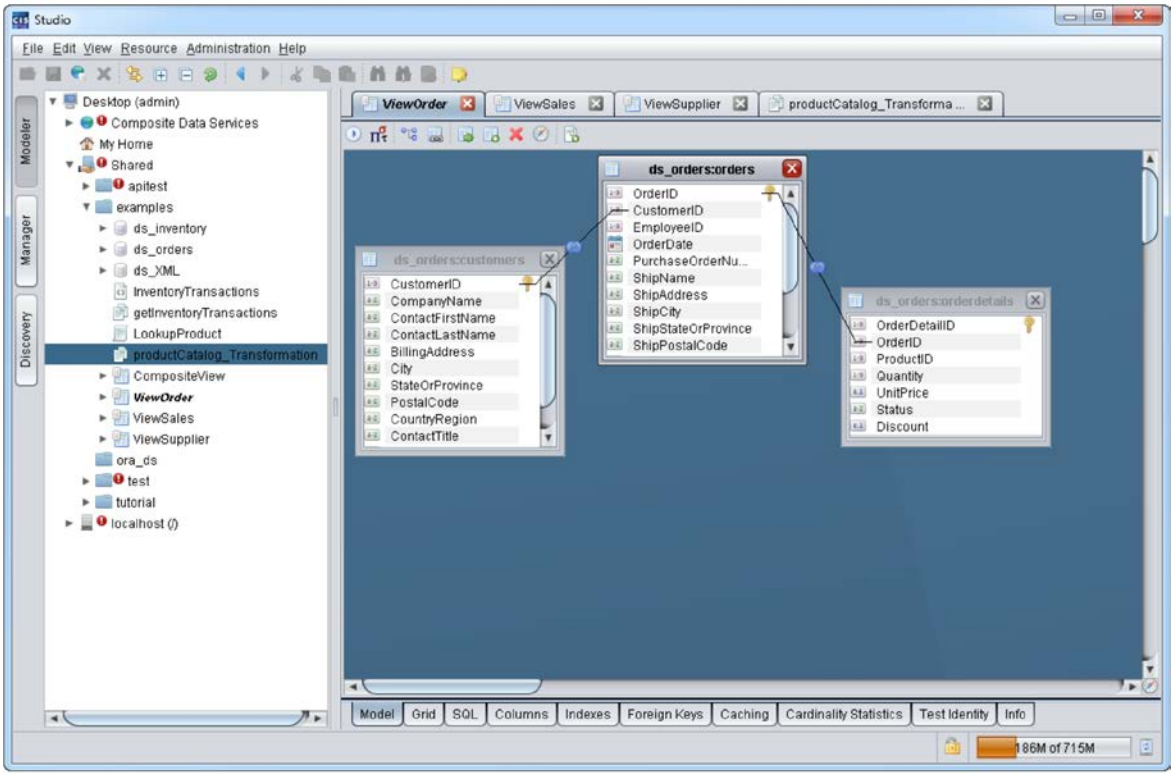
There are different models and activities that can occur at this point; here are some common examples.

Business Intelligence (Reporting)

Once the data transport subsystem has delivered the data, and it is stored in a RDBMS or Hadoop system, reporting tools can be used to visualize the data. However, these tools work better if the data is structured to minimize the number of tables and joins required. Organizing the data appropriately is yet another requirement for leveraging the IoT system.

Many times the report compares the new IoT data (such as today's factory output) to other data (such as the planned output) to gain a perspective. This other (non-IoT) data is typically available in the IT system. In our example, it would come from the Manufacturing Execution System (MES) or the Enterprise Resource Planning (ERP) system. The combining of IoT data and IT data is straightforward using Cisco Data Virtualization.

Figure 12. Cisco Information Server (CIS) Studio Allows Data Selection to Create Views from Sources

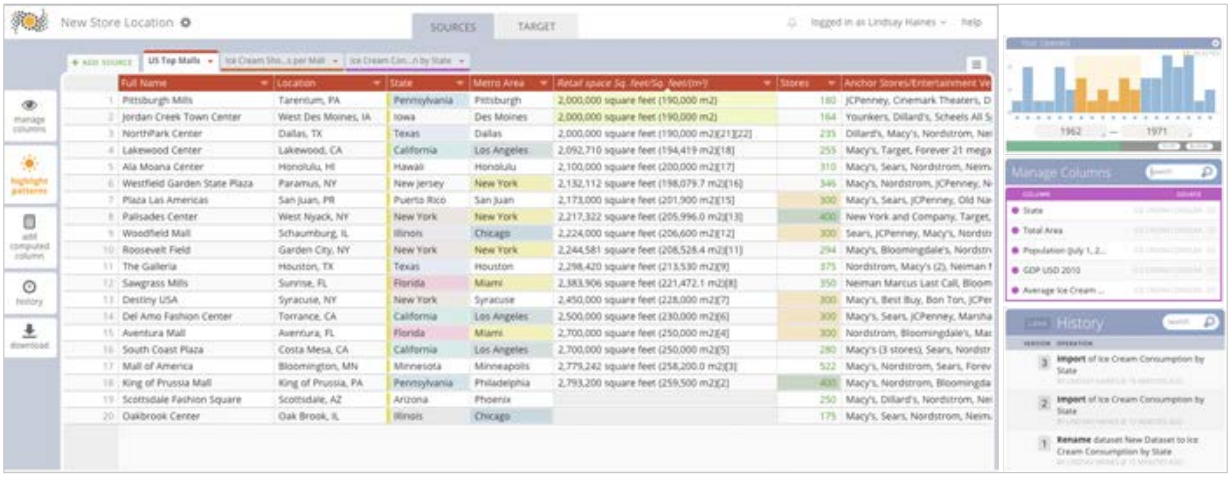


Analytics

Ad Hoc Analytics

Ad hoc analytics are performed by data analysts and data scientists. Since their work is iterative and unplanned, they typically gather the sample of data they want to work with into a sandbox. However, their needs are unique and it is very useful to allow them to prepare the data to assist with their data analysis. The system includes an optional module that can create sandboxes and assist with data preparation prior to statistical analysis. As with other parts of the system, the work is done with a graphical tool. This tool has a highly interactive interface, in the same theme as Microsoft Excel (Figure 13).

Figure 13. Data Preparation Tool



Following the data preparation phase, analytics and statistics can be done with any industry standard analytics product.

Embedded Analytics

Embedded analytics occurs when the analytics are embedded in an application. Cisco provides several embedded analytic applications/systems in areas where it has special expertise. Examples include network analytics, call center analytics, wireless system and location analytics, and event management analytics.

It is straightforward to add an interface to these, and other third party applications, so that the system interacts with them as a microservice.

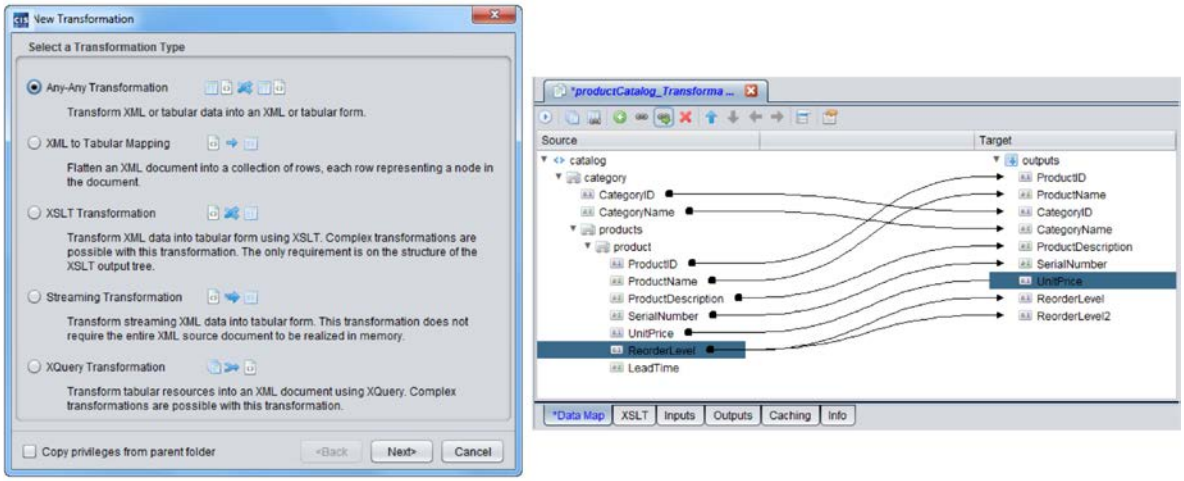
Integration with IT

Almost all IoT systems are put in place by organizations and companies that have invested heavily in IT systems for many years. It is a common objective to integrate the data from IoT sources into these IT systems.

To meet this requirement, the IoT data generated by devices or machines is transported to the destination location and passed to one or more IT applications, typically using a database or Hadoop as a semaphore. In this case, the IoT system must restructure and reformat the data and insert it into the target application's database in the schema defined by the application. This is a well-known technique, and very sophisticated data transformation capabilities

exist in the Cisco Data Virtualization product. Relational, Hadoop key value pairs, XML, SAP, and other data structures can all be converted to that which is needed, without programming (Figure 14).

Figure 14. Data Transformation Using Graphic Development Tools



Control

In the emerging IoT world, automation and control is a key paradigm. Once data is acquired, transported, and leveraged for business improvement, operational insight, and situational awareness, a complete IoT system should provide the agility to take automated actions to alter characteristics of machines/devices, or how their data is processed, based on these data insights. Customers need a control framework to enforce their operational parameters and run books. Broad categories of control types include:

- Control set by an administrator with predefined knowledge, such as operational run book parameters. An example of this control type is a machine running at a certain temperature or torque level; if the settings are change by a local operator, then the IoT control system should enforce the characteristics defined. Similarly, administrators can enforce more efficient energy modes when machines are in idle state.
- Rules and control based on the data collection and consuming applications. Although data collection is independent of the control paradigm, there can be a higher-priority control flow based on real time device and machines monitoring and their surroundings. A real-time change in temperature or humidity in a pipeline could signal a leak or other critical situation. At this instant, this control paradigm should supersede data flow to execute immediate corrective action or alter pipeline characteristics.

The system contains facilities for control and parameter setting of microservices, both during setup as well as during runtime. This topic will be expanded in the next version of this white paper.

Conclusion

Cisco has made significant investments over a period of years, and is well prepared to address the many processing and data needs of the emerging hyperdistributed world. The objective is to bring forward a new form of system for distributed systems, such as IoT, that meets the scalability and quality requirements, is flexible to meet

the customers' exacting needs, comprehensive from the smallest device to the largest Information Technology system, and open so that everyone can extend the system.

We look forward to working with you and helping you to advance your business and achieve the desired outcome.

For More Information

For more information about the Cisco Edge Analytics Fabric system, contact your Cisco account representative, and visit www.cisco.com/go/dataanalytics.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices.**

Printed in USA 06/16