influxdata®

AN INFLUXDATA CASE STUDY

How RTI Remotely Monitors IIoT Systems Using InfluxDB and DDS

Lynne Canavan

Director of Ecosystems, RTI

Kyoungho An

rti

NOVEMBER 2019 (REVISION 1)

Company in brief

Real-Time Innovations (RTI) is a connectivity middleware market leader. It is the largest embedded middleware vendor* and has over 70% of the commercial DDS market share. RTI is also a leader in standards activity, active in 15 standards bodies and a Data Distribution Service^{TM} (DDS) standard author. RTI serves on the boards of prominent industry organizations such as Object Management Group® (OMG®) Board of Directors and Industrial Internet Consortium (IIC) Steering Committee. RTI is a mature leader, with 1,350+ commercial designs and 500+ research projects.

The RTI Connext® databus is a software framework that shares information in real time, making applications work together as one integrated system. It connects across field, fog and cloud. Its reliability, security, performance and scalability are proven in the most demanding industrial systems. Deployed systems include medical devices and imaging; wind, hydro and solar power; autonomous planes, trains and cars; traffic control; oil and gas; and robotics, ships and defense. RTI is privately held, with headquarters in Sunnyvale, CA and Granada, Spain.

Case overview

RTI needed to build a monitoring solution that would meet the unique architectural requirements of monitoring Industrial Internet of Things (IIoT) data for its industrial customers. The IIoT is transforming industries across the board by infusing and connecting billions of connected devices with digital intelligence. Through low-cost computing and networking, today's systems enable far greater amounts of data, at far greater speeds, to be connected throughout distributed systems. As the amount of data increases exponentially, however, there needs to be a new way to monitor it in real time without overwhelming existing systems.

Building a monitoring architecture for IIoT requires a collective software stack for data collection, storage, analysis and visualization. In today's connected world, however, the software components of the monitoring framework may be deployed in a distributed manner and over geographically-dispersed locations, requiring them to be seamlessly integrated over multiple networks.

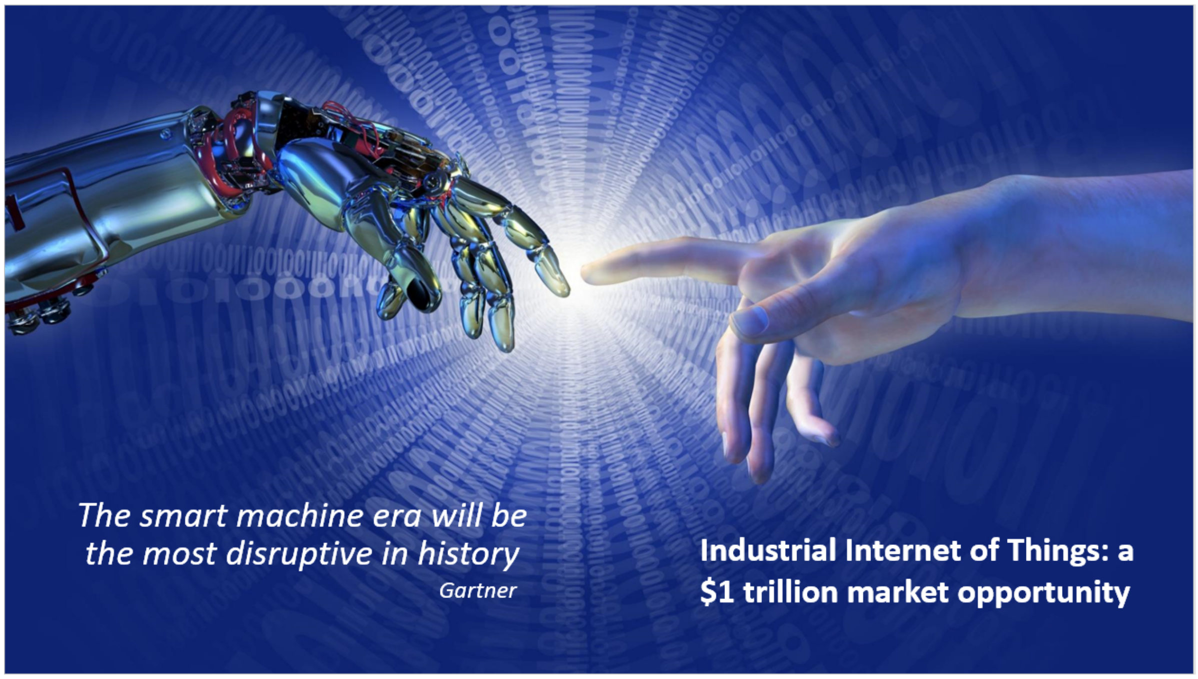
RTI uses InfluxDB in IIoT system monitoring. They created an architecture that meets the software framework requirements by combining the capabilities of the The Object Management Group (OMG) Data Distribution Service™ (DDS) standard for real-time data exchange with InfluxDB. DDS is a middleware protocol and API standard that provides data connectivity, extreme reliability and a scalable architecture to meet IIoT application requirements.

The time series monitoring architecture that RTI built over WAN uses the Telegraf plugins they developed for Connext DDS. The Telegraf plugin monitors and provides alerts throughout distributed IIoT systems.

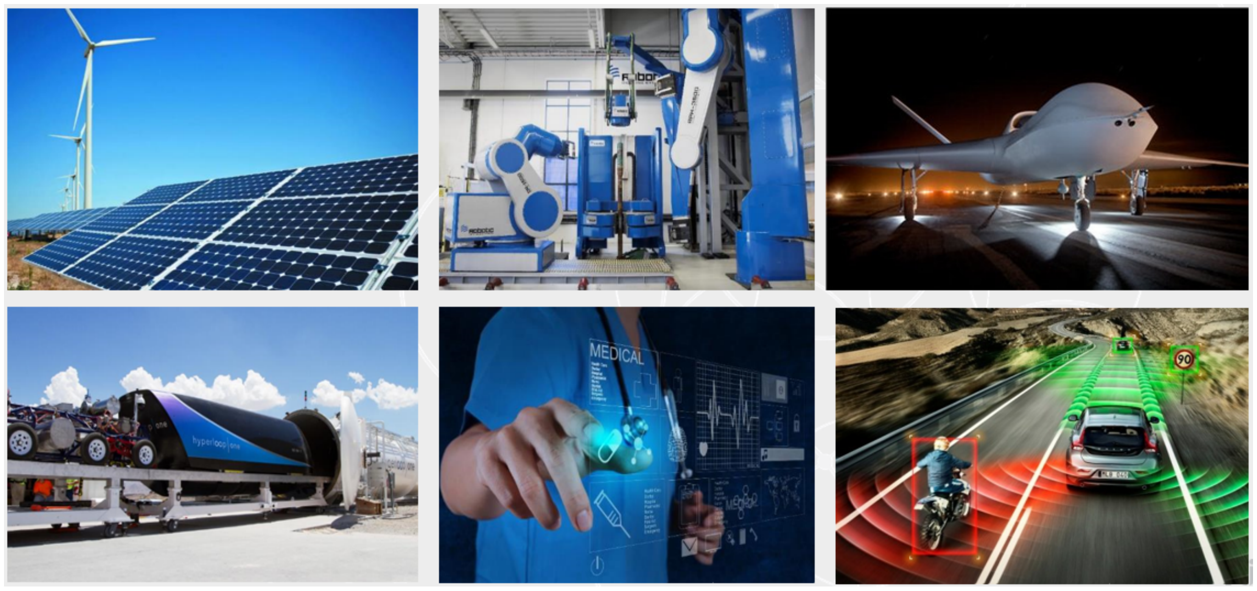
> *“IIoT requires a scalable, data centric approach. Data Distribution Service (DDS) is the standard published by the Object Management Group that accommodates this.”*
>
> ***Lynne Canavan**, Director of Ecosystems, RTI*

The business problem

RTI, known as the Industrial Internet of Things (IIoT) connectivity company, needed to solve the IIoT data challenges of its industrial clients across various sectors. As the IIoT disrupts our world, it brings with it the challenges of data processing at scale, security and precision. Called by some the fourth industrial revolution or the era of smart, connected machines, IIoT is a trillion-dollar market opportunity that’s going to be large and impactful.


The smart machine era will be the most disruptive in history
Gartner
Industrial Internet of Things: a $1 trillion market opportunity

IIoT spans different industries – from healthcare to energy, from manufacturing to transportation – that all share some common attributes with regard to their IIoT implementations.


IIoT use cases across sectors share certain attributes

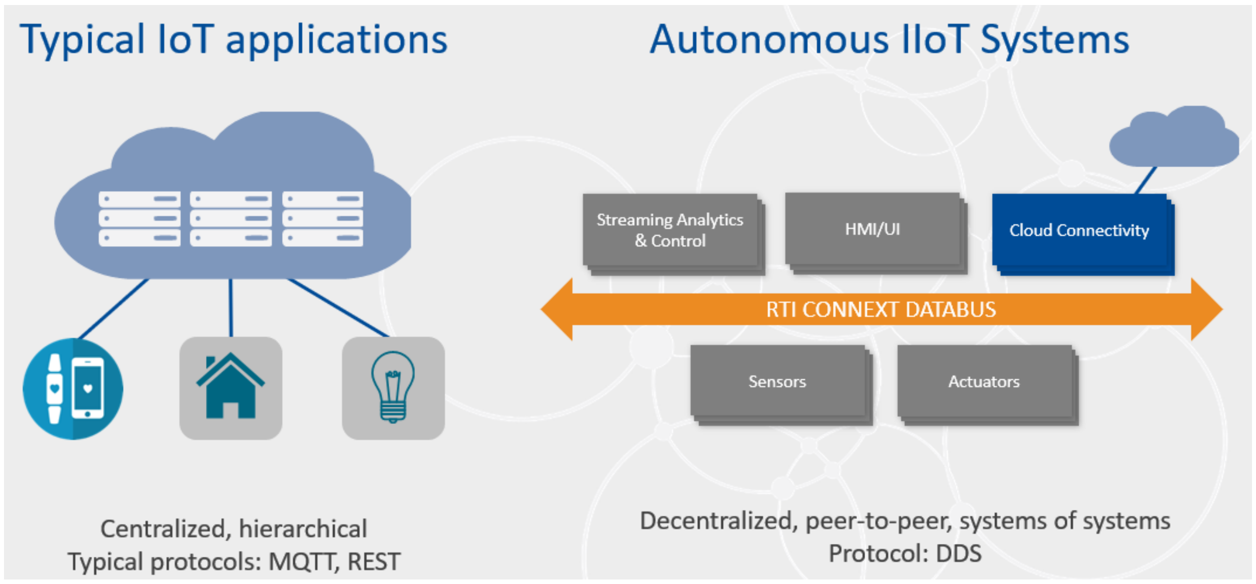
IIoT solutions need to be:

- **Robust and reliable** — there's no room for error in IIoT environments.
- **Secure** — to protect not only from external threats, but also against internal errors.
- **Able to process massive data in real-time** — they require a high-performance architecture that is scalable, secure and reliable with very low latency.

The above requirements are the IIoT requirements that RTI needed to meet for its customers through a suitable monitoring architecture based on DDS.

Technical problem

There are key differences between IoT and Industrial IoT architectures. The architecture that runs a Nest thermostat or personal fitness device simply cannot accommodate the requirements and scale of IIoT. IIoT requires a scalable, data-centric approach. Data Distribution Service (DDS) — for which RTI is the largest commercial vendor — is the standard published by the Object Management Group that accommodates this approach, through a central databus structure.

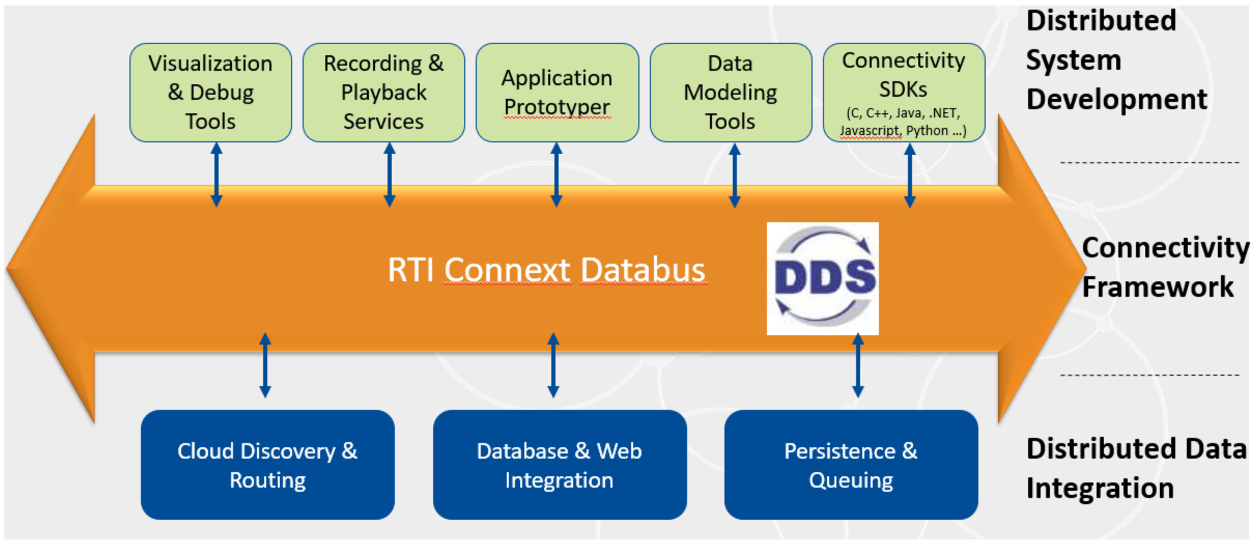

Typical IoT applications
Autonomous IIoT Systems
Streaming Analytics & Control
HMI/UI
Cloud Connectivity
RTI CONNEXT DATABUS
Sensors
Actuators
Centralized, hierarchical
Typical protocols: MQTT, REST
Decentralized, peer-to-peer, systems of systems
Protocol: DDS

IIoT requires scalable, highly reliable data connectivity

RTI Connext Databus is a data-centric framework that develops and runs complex distributed applications. It integrates data in motion from associated applications and tools, which eliminates major bottlenecks and single points of failure (critical for IIoT applications).

Here it's important to note the difference between message-centric and data-centric protocols:

- A traditional message-centric middleware requires writing code that sends messages (a specific code).
- DDS is data-centric — with DDS, the programmers instead write code that specifies how and when to share data, and then directly share data values.


Visualization & Debug Tools
Recording & Playback Services
Application Prototyper
Data Modeling Tools
Connectivity SDKs (C, C++, Java, .NET, Javascript, Python ...)
Distributed System Development
RTI Connext Databus
DDS
Connectivity Framework
Cloud Discovery & Routing
Database & Web Integration
Persistence & Queuing
Distributed Data Integration

RTI Connext Databus: a real-time, data-centric connectivity framework

Rather than managing the complexity in your application code, DDS enables controlled, managed, secure data sharing for specific integration points.

With DDS, applications do not need to deal with data serialization and de-serialized data because they are communicating through data, not messages. Application code is much simpler because all the parts are provided by DDS. DDS architecture is fully distributed peer-to-peer and is designed for real-time and low-latency communication. So there is no broker between the publisher and subscriber.

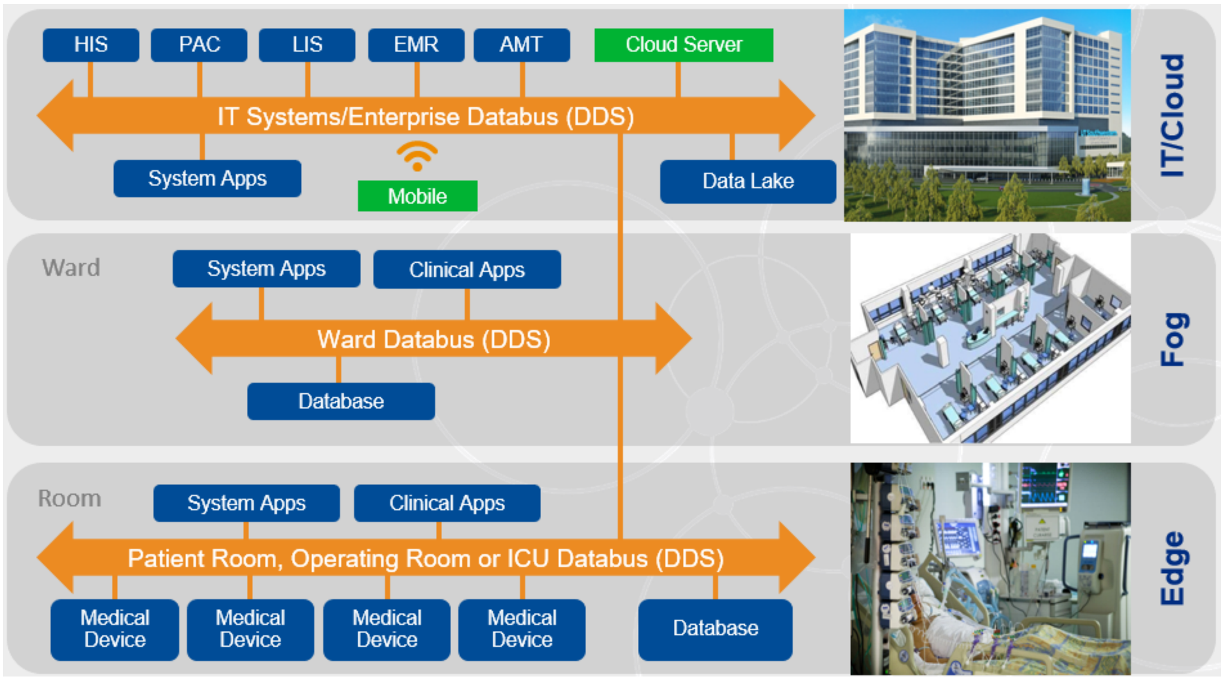
Below are two real-world examples of how a DDS-based connectivity framework can help in a real-world IIoT setting:

- A hospital operating room is typically full of equipment, but even today, with all the tech available — most of those equipment pieces do not communicate with each other. That can lead to latrogenics (hospital error, the 3rd leading cause of death).

Picture taken in an actual operating room

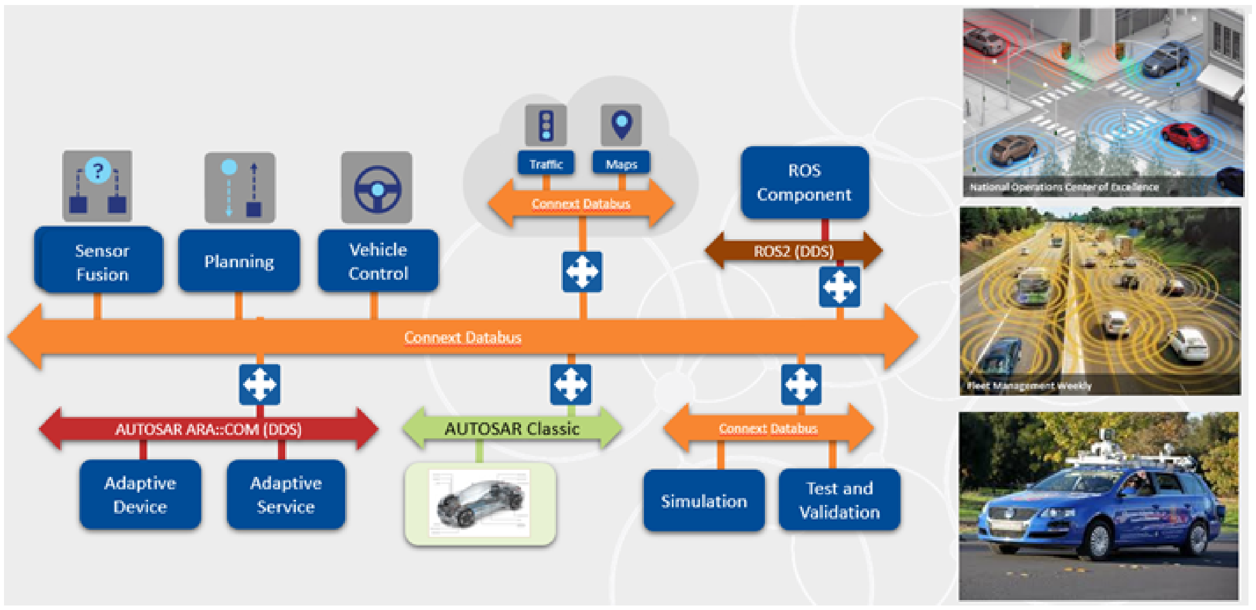
Connected medical devices can help reduce this problem by correlating data from different medical devices; for example, if an infusion of medicine starts, and the patient's blood pressure or oxygen levels drop. Those are separate machines, but through DDS, can be correlated. Connected IIoT systems can flash alarms indicate the need for immediate action rather than manually monitoring single machines.

The amount of data coming from the patient — multiplied for every bed and every surgery unit in the hospital — becomes massive. The below figure shows a layered databus – taking the databus presented above, and adding in the layers that connect from the edge to the cloud. On the bottom layer, in one room, the DDS databus allows the medical devices to talk to each other. That is then connected to the nurses' station, and up to the IT / cloud layer at the hospital. The data flow is contextually appropriate and scalable — with no single point of failure.



Healthcare IIoT: Smart, Connected, Reliable

The same applies to autonomous vehicles — massive data that needs to be communicated rapidly, to and from lots of sources. The vehicle is receiving LiDAR data (LiDar is a surveying method that measures distance to a target by illuminating the target with laser light and measuring the reflected light with a sensor). The vehicle is also receiving traffic data. A lot of data is coming into the vehicle and has to arrive in real-time and to be reacted to quickly. Through the central Connext Databus, you connect to all that is happening in the car and around it. Below is a sample auto architectural databus.



Autonomous vehicles: sensor-to-cloud connectivity

Each vehicle needs a databus for communication between sensors and vehicle drivers, and between vehicles themselves.

The solution

> *"InfluxDB supports built-in time series functions that can select and aggregate or even predict metrics. These time series functions are essential to normalize and analyze an enormous amount of data collected for monitoring and alerting."*
>
> ***Kyoungho An**, Senior Research Engineer, RTI*

Why InfluxDB and Telegraf?

RTI decided to use the InfluxDB stack for its RTI Connext DDS. They use the InfluxDB time series database (for IIoT system monitoring) and InfluxDB's agent Telegraf (for collecting metrics and events). Both are open source and publicly available for download and deployment.

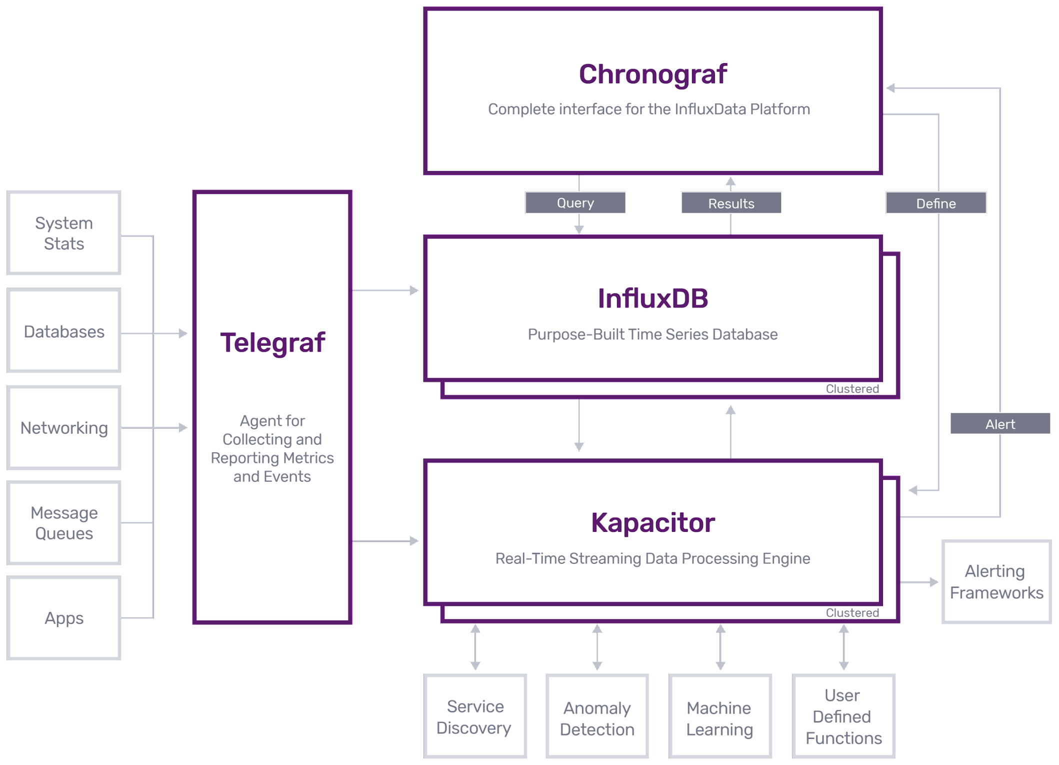
RTI came to know InfluxDB while exploring time series monitoring technology for a research project funded by the Department of Energy to develop system management capabilities for the Microgrid standard called Open Field Message Bus. At the time, they looked into other microgrid technologies, but decided to use InfluxDB because it is purpose-built for time series data, and since monitoring data is time series data.

These are the reasons why they decided to use Telegraf:

- Push-based metric collection, which is a better fit for DDS (which uses event-based pub/sub model)
- Many out-of-the-box plugins for system monitoring (which makes it easy to collect from diverse sources for system value chain and easy to extend)
- Mature and widely adopted open source technology with commercial offering/support
- They realized they could easily build a Telegraf plugin for DDS without paying InfluxDB or having to wait for their roadmap to accommodate building one. This is why open systems are important.

InfluxDB integrates with Connext DDS through the Telegraf plugins for Connext DDS that RTI built.

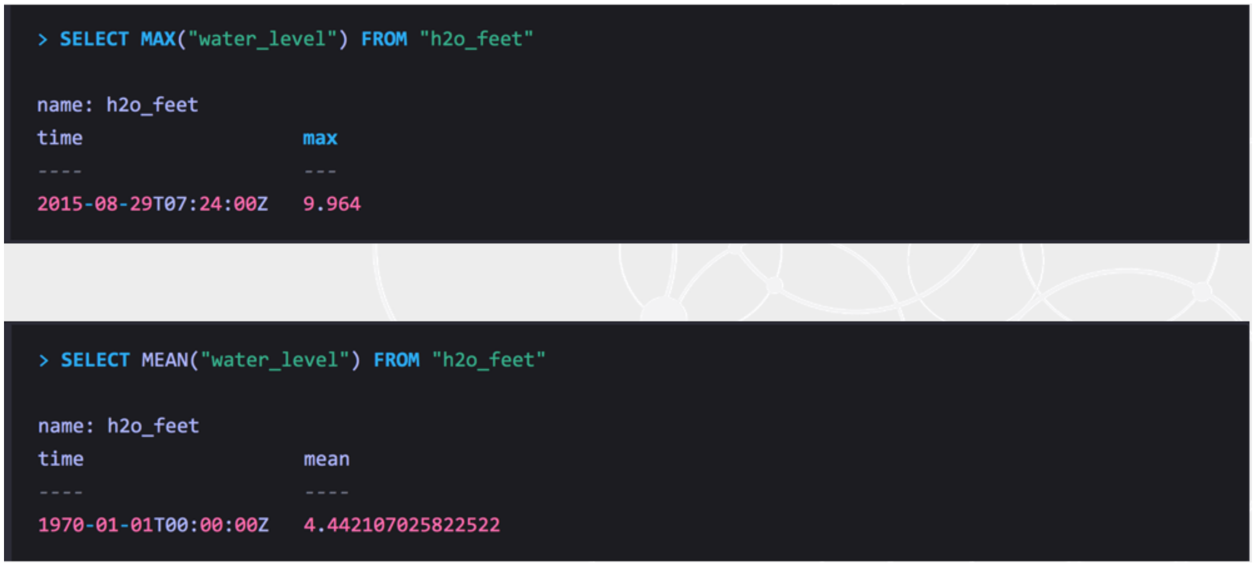
Benefits of using DDS and InfluxDB



InfluxDB platform architecture

Telegraf provides 200+ plugins contributed by the community and has a very strong ecosystem. You can collect metrics from diverse sources and provide collective metrics to many destinations.

Additionally, InfluxDB supports built-in time series functions that can select and aggregate or even predict metrics. These time series functions are essential to normalize and analyze an enormous amount of data collected for monitoring and alerting.

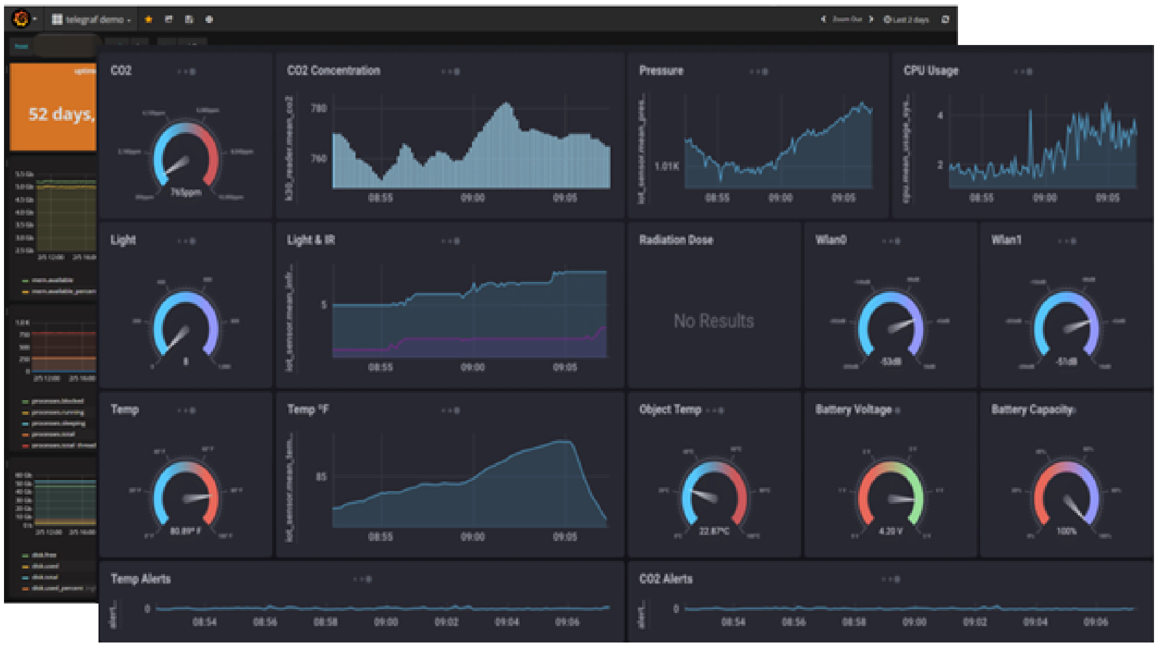


```
> SELECT MAX("water_level") FROM "h2o_feet"

name: h2o_feet
time                  max
----                  ---
2015-08-29T07:24:00Z  9.964
```

```
> SELECT MEAN("water_level") FROM "h2o_feet"

name: h2o_feet
time                  mean
----                  ----
1970-01-01T00:00:00Z  4.442107025822522
```

Example of built-in time series functions in InfluxDB

Lastly, InfluxDB is well-integrated with visualization tools, like Grafana and Chronograf. Specifically, with Grafana, there are many freely available dashboards for InfluxDB and Telegraf.



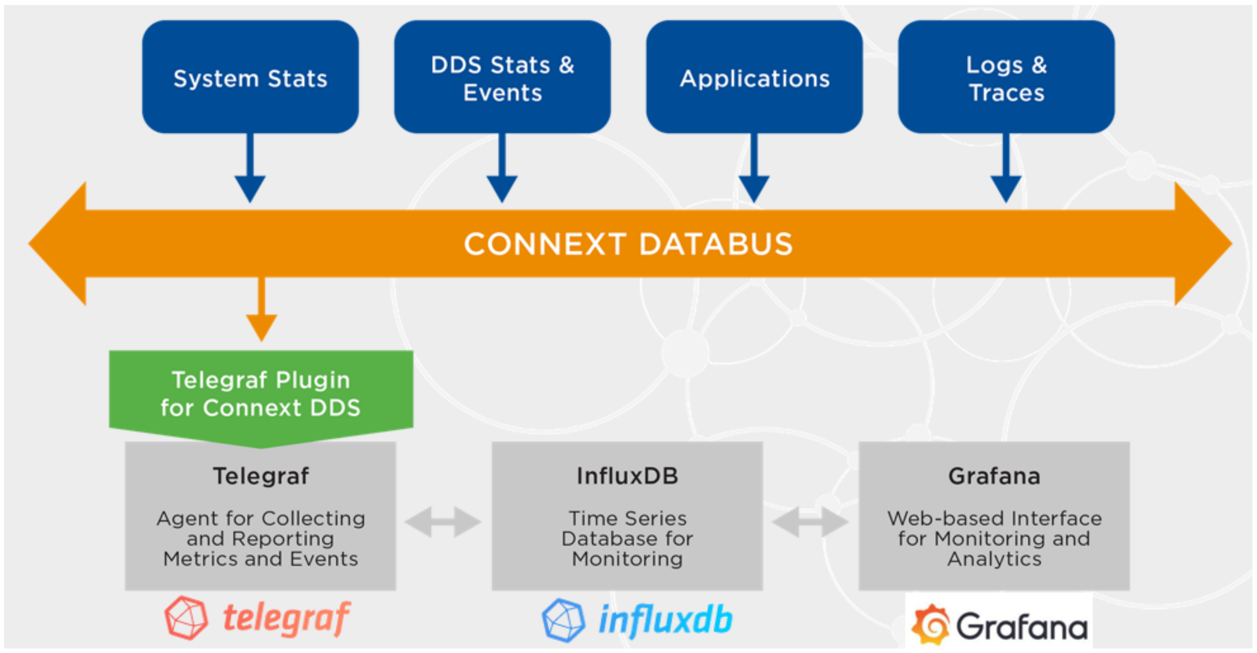
RTI Connext DDS is the first connectivity framework designed for the demanding requirements of the IIoT. Based on the Connext Databus, RTI's software allows applications to exchange data in real time and provides the non-stop availability and security essential for mission-critical systems.

RTI Connext DDS software includes the world's leading implementation of the Object Management Group (OMG) Data Distribution Service (DDS) standard. DDS is the only open standard for messaging that supports the unique needs of both enterprise and real-time systems. Its open interfaces and advanced integration capabilities slash costs across a system's lifecycle, from initial development and integration through ongoing maintenance and upgrades.

Technical architecture

> *"The Telegraf plugin for Connext DDS enables the monitoring architecture with DDS and InfluxDB."*
>
> ***Kyoungho An**, Senior Research Engineer, RTI*

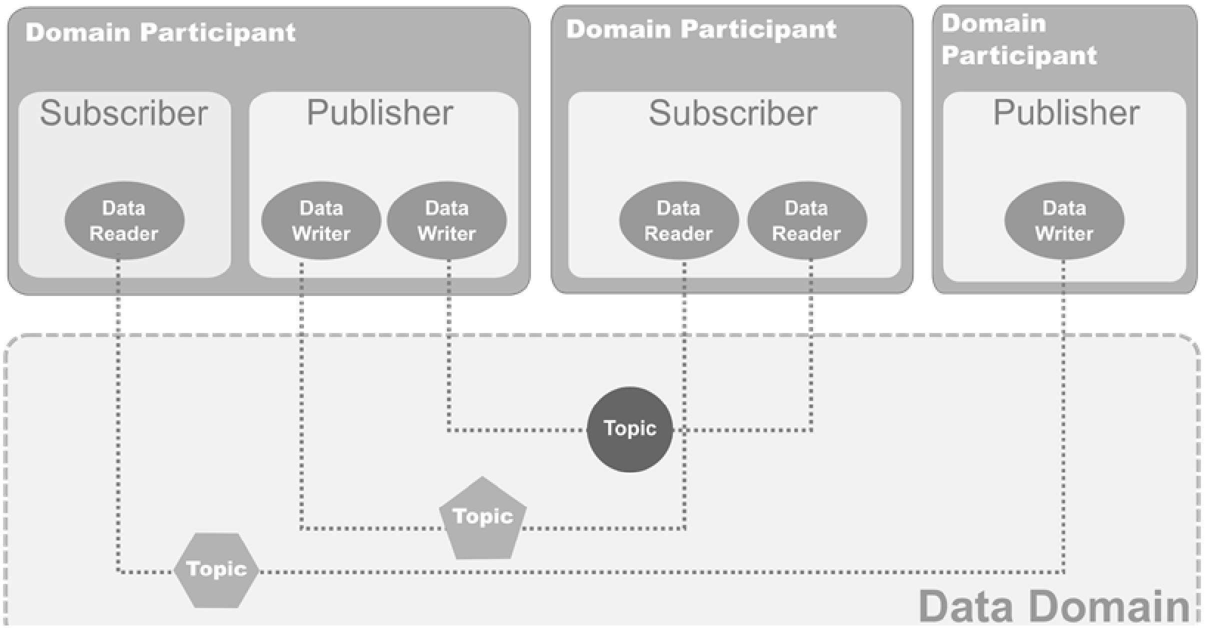
The architecture with RTI's Connext DDS and InfluxDB


System Stats
DDS Stats & Events
Applications
Logs & Traces
CONNEXT DATABUS
Telegraf Plugin for Connext DDS
Telegraf
Agent for Collecting and Reporting Metrics and Events
InfluxDB
Time Series Database for Monitoring
Grafana
Web-based Interface for Monitoring and Analytics
telegraf
influxdb
Grafana

The above figure shows the architecture of the integration of Connext DDS and InfluxDB. The key component to realize this architecture is the Telegraf plugin for Connext DDS. With this architecture, metrics and logs for systems and applications can be sent and received over Connext DDS databus. Then, the received metrics can be provided to InfluxDB for visualization and alert tools like Grafana.

DDS overview and terminology

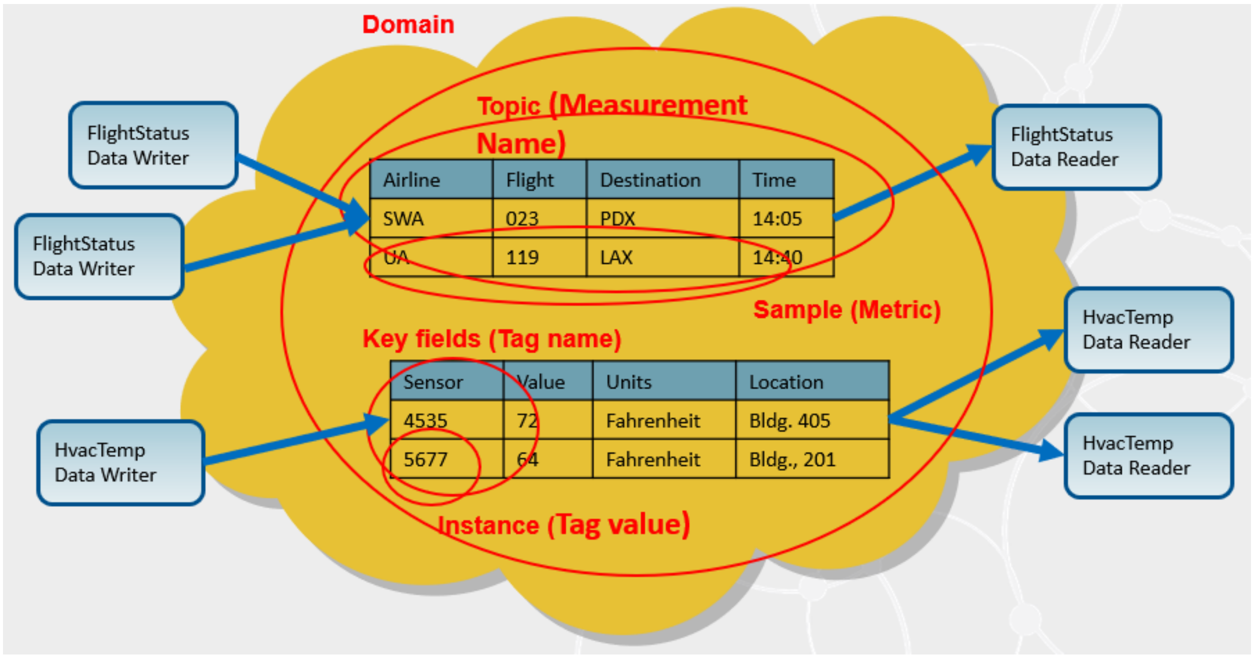
The below diagram describes the key architectural entities of DDS. Since DDS adopted the publish/subscribe model, DDS applications can communicate through DDS published and subscribed entities, which are data writers for publication and data readers for subscription. Here's how this communication works.



DDS Terminology

- If data writers and readers use the same topic, then they can exchange a structured topic data.
- Multiple writers and readers can be grouped under a publisher or a subscriber.
- A participant can create multiple publishers and subscribers.
- DDS data domain is a logical communication environment used to isolate network communications of DDS participants – only participants using the same domain_ID can communicate with each other.

DDS concepts can be explained better with some InfluxDB concepts. Within a DDS domain, topics exist. A topic can be a database table, that can be written or read by a data writers or data readers. It can be a similar concept to the measurement name in InfluxDB. Within a DDS topic, it has a sample of data and has a set of fields, like metric data in InfluxDB. Within a single topic, samples can be grouped and identified by key fields, which can be equivalent concepts to a tag name. DDS instance is the identifier of sample streams in a topic and can be a similar concept to a tag value. This analogy helps to understand DDS concepts from InfluxDB's perspective.



Airline	Flight	Destination	Time
SWA	023	PDX	14:05
UA	119	LAX	14:40

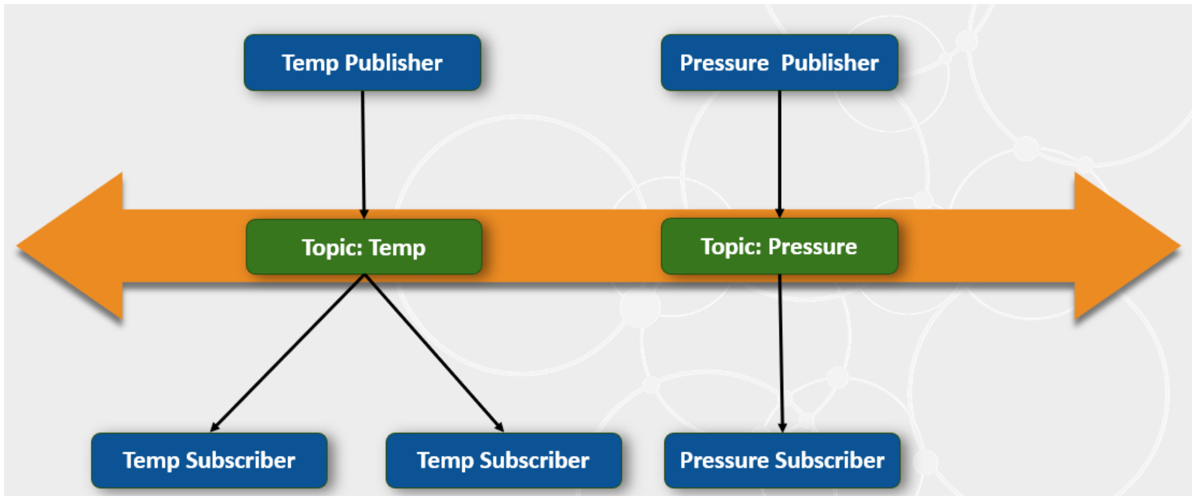
Sensor	Value	Units	Location
4535	72	Fahrenheit	Bldg. 405
5677	64	Fahrenheit	Bldg., 201

Mapping of DDS and InfluxDB Concepts

DDS design and features

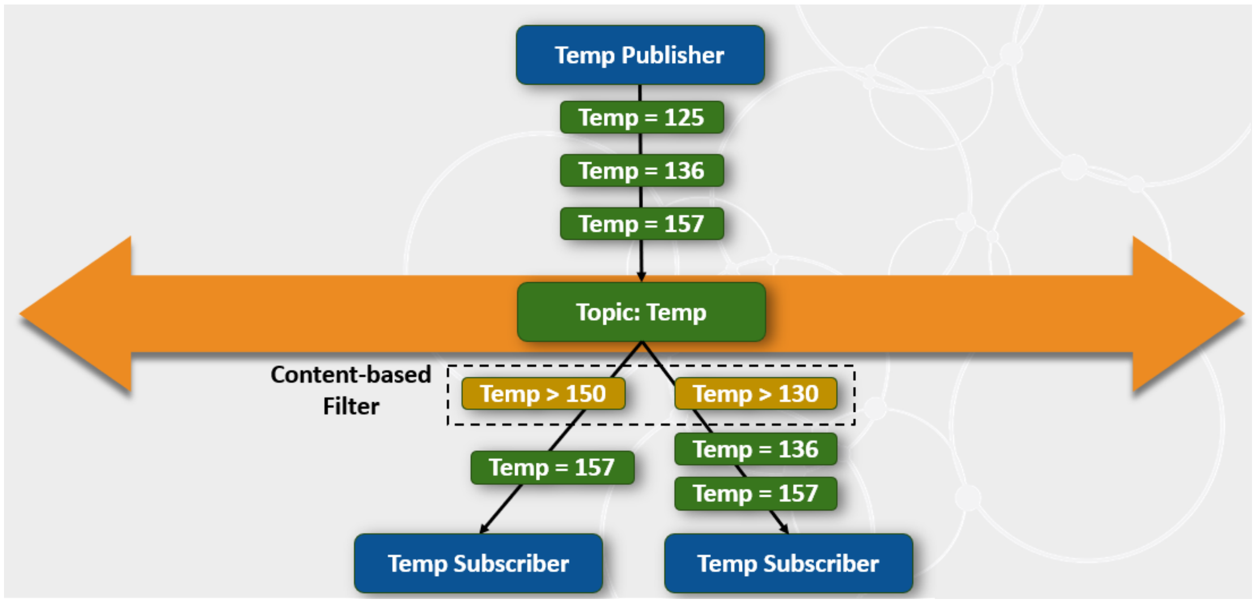
First, DDS is fully distributed (there's no broker between publishers and subscribers) and supports decoupled interactions. So the publisher and subscribers directly communicate with each other. This allows achieving very high throughput and low latency, without a single point of failure or bottleneck.

As shown in the figure below, DDS applications declare intents to publish or subscribe to a topic. This does not require prior knowledge of each other. So publishers and subscribers can join anytime from any place. This decoupling can greatly reduce integration effort.



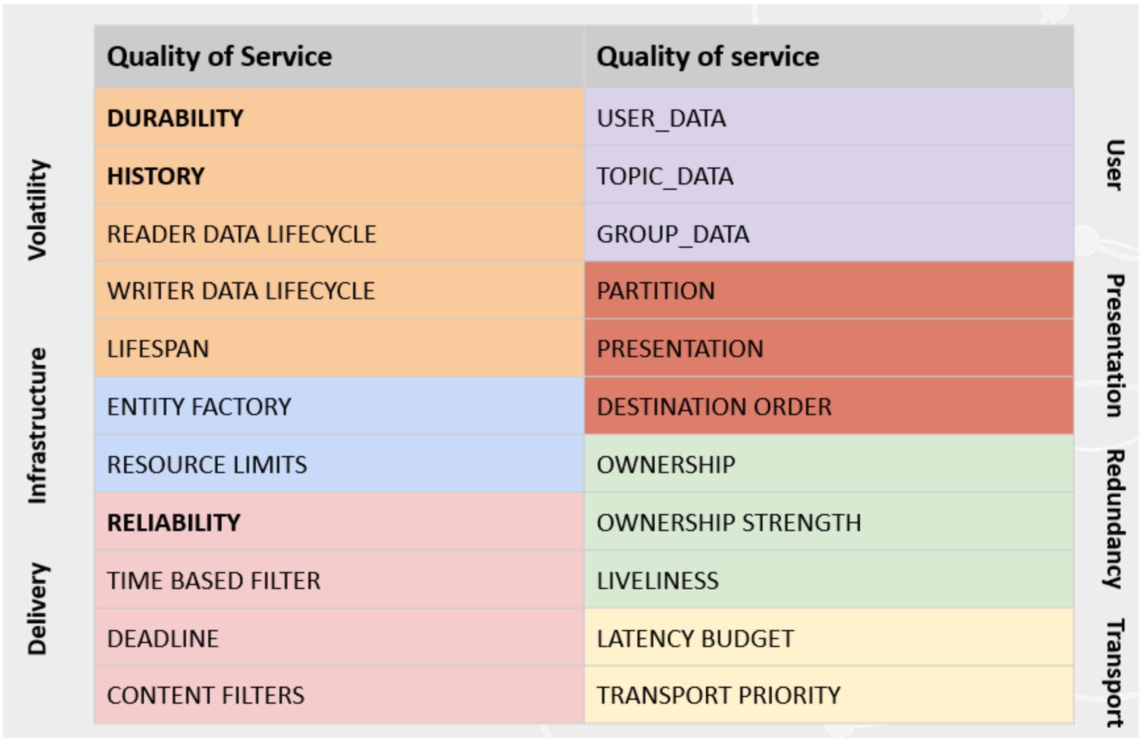
DDS is fully distributed and supports decoupled publish/subscribe communications

DDS is data-centric. This means that publishers and subscriber exchange strongly-typed data, not messages. With that, DDS can support content-based filtering on the publisher side. For example, a subscriber can declare content filters; such as wanting to receive temperature data above a certain threshold. Then a publisher delivers the data only within that threshold. This will be quite useful to improve efficiency in an environment having a limited network bandwidth.



DDS is data-centric and supports content-based filtering

DDS provides more than 20 standard configuration parameters for data delivery, resource usage, etc. These parameters are called Quality of Services (QoSs). Among them, the most interesting are reliability, durability, and history QoS.

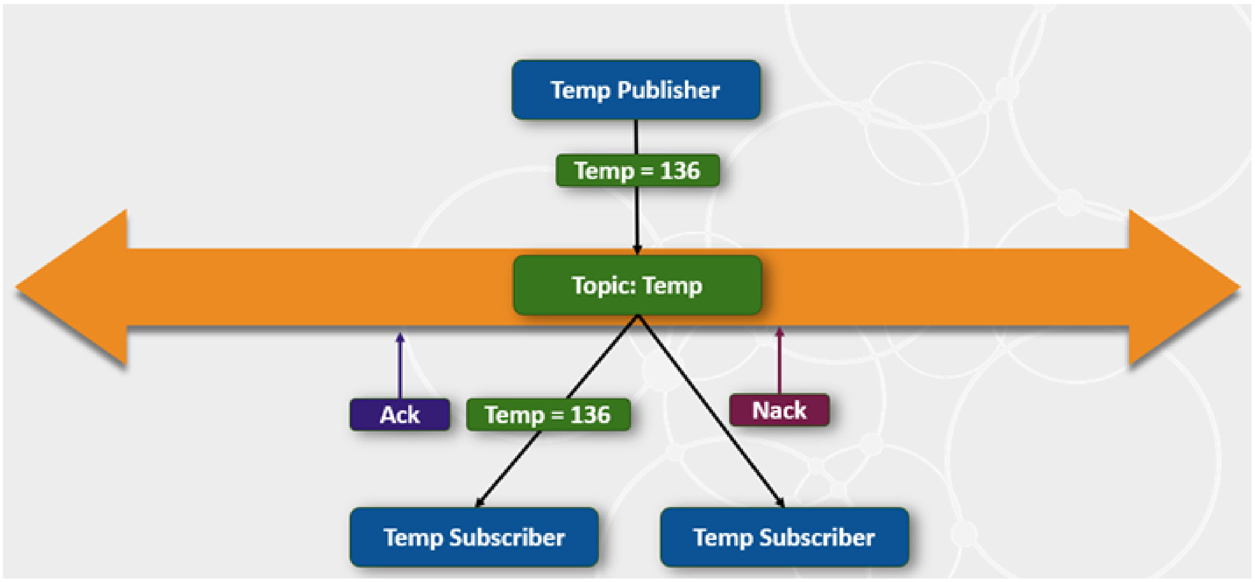

Volatility
Infrastructure
Delivery
User
Presentation
Redundancy
Transport

Quality of Service	Quality of service
DURABILITY	USER_DATA
HISTORY	TOPIC_DATA
READER DATA LIFECYCLE	GROUP_DATA
WRITER DATA LIFECYCLE	PARTITION
LIFESPAN	PRESENTATION
ENTITY FACTORY	DESTINATION ORDER
RESOURCE LIMITS	OWNERSHIP
RELIABILITY	OWNERSHIP STRENGTH
TIME BASED FILTER	LIVELINESS
DEADLINE	LATENCY BUDGET
CONTENT FILTERS	TRANSPORT PRIORITY

DDS can control data flows and resource usage

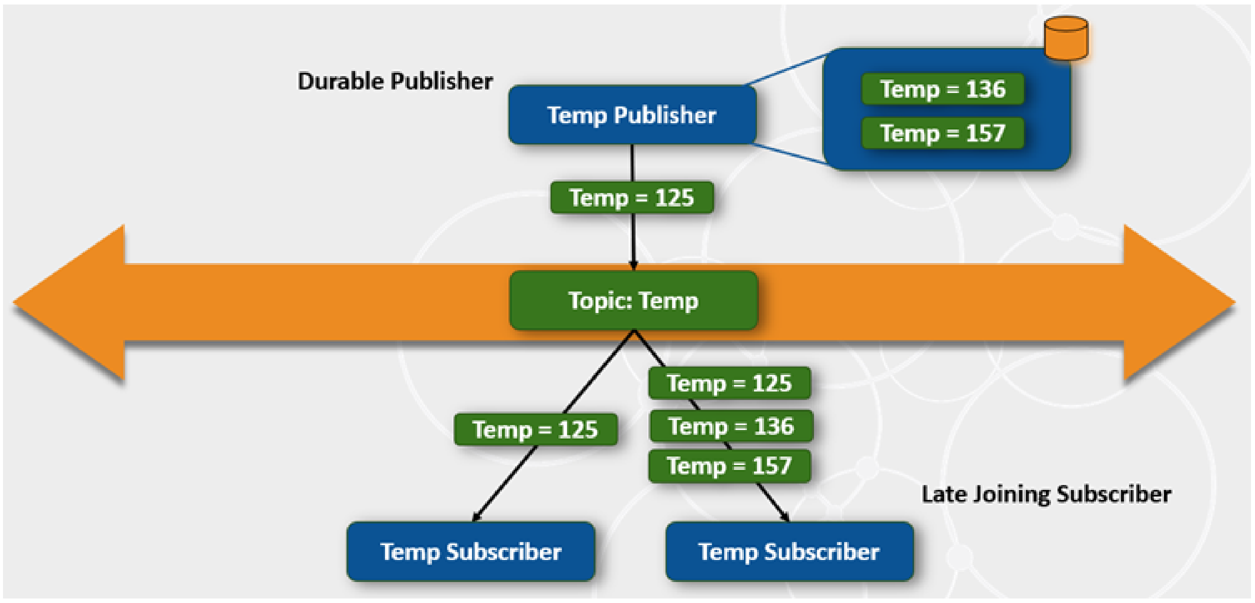
DDS supports multiple transports, including shared memory UDP and TCP. The default transport of DDS is UDP, which does not have a reliable mechanism for data transmission. But with DDS reliability QoS, it can deliver data in a reliable way on top of UDP, if it is set to reliable.

At a high level, with DDS reliability QoS, if data is delivered, a subscriber responds with an `Ack` message to the corresponding publisher. If not, the subscriber responds with a `Nack` message to request to send the data again. Then, the publisher resends the missing data. DDS QoS policies are configurable. You can set it to either, for example, reliable or best effort, depending on your use case.



DDS supports reliable delivery on top of UDP and multicast

DDS can deliver historical data for late joiners with durability QoS. So a persistent publisher keeps historical data. When a late-joining subscriber joins the system, it can deliver the historical data stored in its historical buffer.



DDS can deliver historical data for late joiners

You can configure historical data to be stored either in memory or persistent storage. The size of buffers can also be configured through history QoS, so you can control the amount of historical data that you'd like to persist.

Telegraf plugins for RTI Connext DDS

RTI developed three Telegraf input plugins and one Telegraf output plugin:

Input Plugins

- **DDS Consumer** (can read DDS data in any DDS data types)
- **DDS Consumer with the Line Protocol data model** (can read DDS data with a specific data model, developed based on the Line Protocol format)
- **DDS Monitor** (can read monitoring data for these DDS applications). DDS uses DDS itself to send its monitoring data, such as how many samples were sent or received, or are there any missing or rejected samples. Those types of metrics can be collected. And we can use Telegraf for collecting that data.

Output Plugin

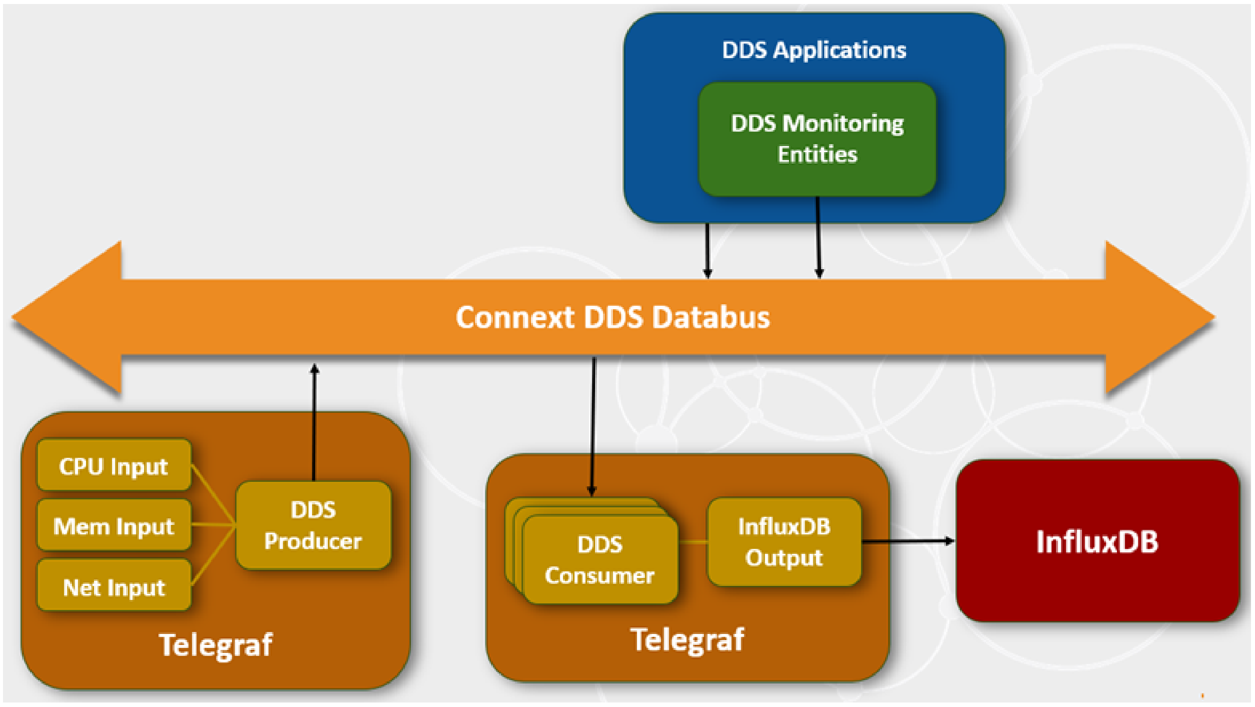
- **DDS Producer with the Line Protocol data** (can get metrics from any Telegraf input plugins and publish them with the data model based on the Line Protocol format)

RTI developed multiple input plugins because DDS data samples are strongly typed, and they wanted to support ingesting data from:

- Any DDS applications (application-specific data types)
- Telegraf plugins (Line Protocol data type)
- DDS monitoring entities (DDS monitoring data type)

As the below figure shows, RTI uses Telegraf to:

1. **Support getting data from DDS applications**, with application-specific data types. DDS Consumer needs to provide a way to define DDS data types by users, and convert this data to the format that Telegraf can understand.
2. **Subscribe to data from DDS monitoring entities**, which are mainly for monitoring DDS-specific metrics.
3. **Send metrics from existing Telegraf input plugins** (such as CPU, memory and network), which required RTI to develop their data type based on the Line Protocol format.



How RTI Connext DDS uses Telegraf

How the DDS Consumer plugin was implemented

To explain the plugin's implementation, it's important to first introduce the RTI Go Connector, which was used to create and access DDS entities for developing Telegraf plugins.

RTI Go connector is a simplified API for DDS application development in Go programming language. It is built on top of DDS C API, with cgo, which is a way to call C code from go. RTI connector was developed by the RTI research team and is freely accessible in the RTI community GitHub.

The DDS Consumer plugin, which can subscribe to any DDS applications with application-specific data, is developed as a service input plugin.

```
type ServiceInput interface {
    Input
      // Start the ServiceInput.
      Start(Accumulator) error
      // Stop stops the services and closes any necessary channels and
connections
      Stop()
 }
```

Telegraf provides plugin API, and RTI leveraged a service input plugin API to develop the DDS Consumer Plugin because this plugin creates a thread to handle DDS samples when they arrive. A service input plugin API has additional interface functions on top of the input plugin API. They include Start and Stop functions:

- Most of the plugin implementation is included in the `Start` function, which is invoked when the Telegraf plugin gets started.
- The `Stop` function is invoked when the Telegraf plugin is stopped. (In the current example, we have to code for deleting DDS entities that were created for this plugin in the `Stop` function.

The `Start` function then:

- Creates DDS entities defined in our external configuration file (covered further below). The connector function creates an RTI DDS connector object with a participant name and a path for a DDS external configuration file.
- Can get the DDS reader with its name, by calling `GetInput`. It is also defined in an external configuration and file, as well. The created connector and reader object pointers are stored in the DDS consumer object. They are used to read DDS samples in the process functions.
- Starts a service thread for processing received DDS samples.

```
func (d *DDSConsumer) Start(acc telegraf.Accumulator) error {
      // Create a RTI Connector
      d.connector, err = rti.NewConnector(d.ParticipantConfig,
      d.ConfigFilePath)
      // Get a DDS reader
      d.reader, err = d.connector.GetInput(d.ReaderConfig)
      // Start a thread for processing DDS samples
      go d.process()
```

Below is the code for the process function, by which a service thread reads and processes DDS data samples from the reader and then processes and injects metrics to a Telegraf output.

```
// Take DDS samples from a DataReader and ingest them to Telegraf outputs
func (d *DDSConsumer) process() {
  for {
  d.connector.Wait(-1)  // Wait until a new DDS sample arrives
  d.reader.Take()  // Take DDS samples
  numOfSamples := d.reader.Samples.GetLength()
      for i := 0; i < numOfSamples; i++ {  // Iterate the DDS samples
  json, err := d.reader.Samples.GetJSON(i)  // Return a DDS sample in JSON
  metrics, err := d.parser.Parse(json) // Parse the JSON object
  // Add metrics to an output plugin
```

If multiple samples arrive, the function has to iterate them. Within the iteration, the DDS sample is converted to a JSON object. Then the JSON object is parsed to a metric struc, resulting in a metric object that you can pass on to an output plugin.

Thanks to all the functions provided by the plugin API and RTI Go connector, the plugin code is small and simple, consisting of less than 200 lines of code.

How to use the input plugin

The DDS Consumer input plugin, as the other plugins, can be downloaded from the RTI Community GitHub:

- A fork repository from Telegraf
- Installation
- How to use it
- Example configurations

Assuming that you downloaded and have a Telegraf executable, you first have to create an external configuration file for DDS data types and entities.

Here is an example of a DDS external configuration file.

Creating an XML configuration file

```
<dds>
<!-- Data Types -->
<types>
<struct name="ShapeType" extensibility="extensible">
      <member name="color" stringMaxLength="128" id="0" type="string"
      key="true"/>
      <member name="x" id="1" type="long"/>
      <member name="y" id="2" type="long"/>
      <member name="shapesize" id="3" type="long"/>
</struct>
</types>
```

First, the external file will include the definition of data types. For example, here, we have defined a data type for a shape that includes four attributes, “color” in “string”;”x” and “y” coordinates in “long”; and “shapesize” in “long”.

After you have a data type defined, you should define a DDS domain, which includes a `domain_id` you'd like to participate (which is “0” in the below example). Within the domain definition:

- Register the data type you'd like to use. In the example, we registered the “ShapeType” defined in the previous code example.
- Define a topic associated with the data type. You need to define DDS participants, subscribers, and readers you'd like to create.

```
<!-- Domain Library -->
<domain_library name="MyDomainLibrary">
<domain name="MyDomain" domain_id="0">
```

```
      <register_type name="ShapeType" type_ref="ShapeType"/>
      <topic name="Square" register_type_ref="ShapeType"/>
</domain>
</domain_library>
```

In this example, as shown in the code below:

```
<!-- Participant library -->
<domain_participant_library name="MyParticipantLibrary">
<domain_participant name="Zero" domain_ref="MyDomainLibrary::MyDomain">
      <subscriber name="MySubscriber">
            <data_reader name="MySquareReader" topic_ref="Square"/>
      </subscriber>
</domain_participant>
</domain_participant_library>
</dds>
```

- We defined a participant named "Zero", in the domain defined in the code example.
- Under that participant, we defined a subscriber named, "MySubscriber".
- Under that subscriber, we defined a `data_reader` named "MySquareReader" that uses the "Square" topic that was defined in the domain definition.

The above three steps complete recreating an external configuration file for DDS.

The next step is defining a Telegraf configuration file. Below is part of a Telegraf configuration file that uses the DDS Consumer input.

Creating a TOML configuration file

```
[[inputs.dds_consumer]]
  # XML configuration file path
  config_path = "example_configs/ShapeExample.xml"

  # Configuration name for DDS Participant from a description in XML
  participant_config = "MyParticipantLibrary::Zero"

  # Configuration name for DDS DataReader from a description in XML
  reader_config = "MySubscriber::MySquareReader"
```

Like other input configurations, you define the name of the input plugin (in this case, `DDS_consumer`). Under that, there are specific configurations for this plugin.

- `config_path` is for defining the path for the external configuration file (discussed in the previous section).
- `participant_config` is the full name of DDS Participant in the external configuration file. In this example, the full name of the participant is "MyParticipantLibrary::Zero".

- `reader_config` is the full name of DDS Reader in the external configuration file. In this example, the full name of the reader is, "MySubscriber::MySquareReader".

The above configurations are all you need for DDS configuration.

Additionally, as shown below in the code snippet below:

- You can optionally add some fields as tags (In the below example, a color field is added as a tag) and also override the name of the measurement; otherwise, it will use the plugin's name.
- You should use JSON as data format because the plugin code converts DDS samples to only JSON format.

```
  # Tag key is an array of keys that should be added as tags.
  tag_keys = ["color"]

  # Override the base name of the measurement
  name_override = "shapes"

  # Data format to consume.
  data_format = "json"
```

How the DDS Producer/Consumer plugins with Line Protocol data model are implemented

Below is code to show how the DDS Producer/Consumer plugins, that are developed with a data model based on Line Protocol format, are implemented.

```
struct Metric {
      string name;//@key
      sequence<Tag, MAX_TAGS> tags;
      sequence<Field, MAX_FIELDS> fields;
      long long timestamp;
};
```

https://github.com/rticommunity/telegraf/blob/master/plugins/inputs/dds_consumer_lp/line_protocol.idl

```
type Metric struct {
  Name      string    `json:"name"`
  Tags      []Tag         `json:"tags"`
  Fields    []Field     `json:"fields"`
  Timestamp int64   `json:"timestamp"`
}
```

https://github.com/rticommunity/telegraf/blob/master/plugins/inputs/dds_consumer_lp/dds_consumer_lp.go

Like the DDS Consumer input plugin, the DDS producer/consumer plugins with Line Protocol data model initially create DDS entities and then process DDS samples.

```
// Read a DDS sample
                  var m Metric
                  d.reader.Samples.Get(i, &m)

  // Read tags from the DDS sample
                  tags := make(map[string]string)
  for _, tag := range m.Tags {
  tags[tag.Key] = tag.Value
  }

// Read fields from the sample
                  fields := make(map[string]interface{})
  for _, field := range m.Fields {
            switch field.Kind {
            case FIELD_DOUBLE:
              fields[field.Key] = field.Value.D
                  ...
  // Add a metric to an output plugin
                  d.acc.AddFields(m.Name, fields, tags, time.Unix(0,
                  m.Timestamp))
```

How to use the DDS producer/consumer plugins with Line Protocol data

To use the DDS Producer/Consumer plugins with Line Protocol data, there is no need to create an XML configuration file. The plugin source code includes the DDS XML configurations:

```
      /// DDS XML configurations for the Line Protocol plugins
      // Topic name is Telegraf and the type name is Meric
      var xmlString = `
      str://"<dds>
        …</dds>"`
       // Create a Connector object from the XML config
       d.connector, err =
rti.NewConnector("ParticipantLib::TelegrafParticipant", xmlString)
```

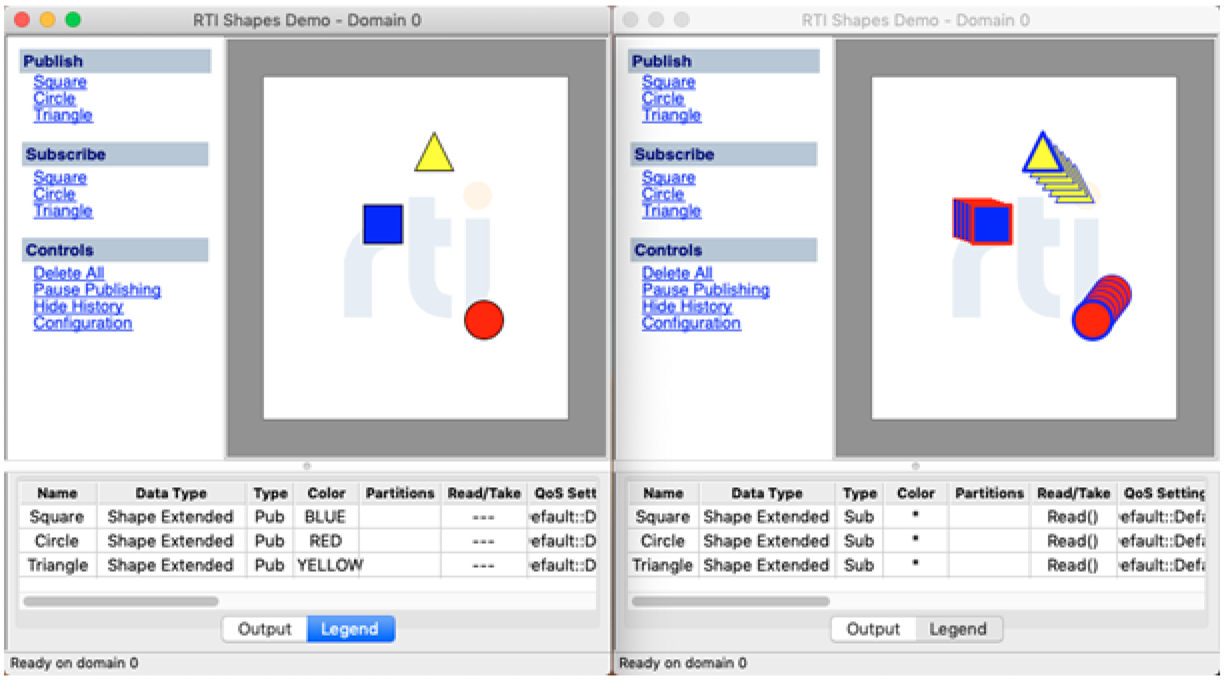
Creating a TOML configuration file

```
# Send metrics over DDS
[[outputs.dds_producer_lp]]
  # DDS Domain ID configuration
  domain_id = "0"
# Read metrics from DDS
```

```
[[inputs.dds_consumer_lp]]
  # DDS Domain ID configuration
  domain_id = "0"
```

RTI Shapes demo

RTI Shapes demo is a tool you can use to learn about the basic and advanced concepts of DDS and to see the Connext DDS and InfluxDB integration in action. You can create a publisher or subscriber with your chosen DDS configurations. The demo shows how the DDS Consumer input plugin works, covering two interesting DDS capabilities: delivering historical data for late joiners and content-based filtering. By watching this demo, you can actually see the data that's working.



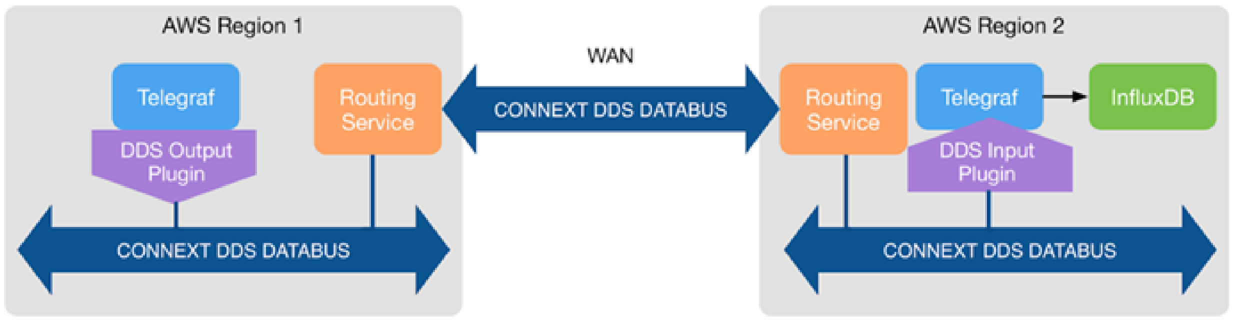
Name	Data Type	Type	Color	Partitions	Read/Take	QoS Sett
Square	Shape Extended	Pub	BLUE		---	efault::D
Circle	Shape Extended	Pub	RED		---	efault::D
Triangle	Shape Extended	Pub	YELLOW		---	efault::D

Name	Data Type	Type	Color	Partitions	Read/Take	QoS Setting
Square	Shape Extended	Sub	•		Read()	efault::Defa
Circle	Shape Extended	Sub	•		Read()	efault::Defa
Triangle	Shape Extended	Sub	•		Read()	efault::Defa

Integration of DDS systems over WAN

Routing Service enables seamless integration of DDS systems over WAN, as shown in the figure below:

- Telegraf in **AWS Region 1** uses System input plugins (e.g. cpu, mem, net) and DDS producer plugin with the Line Protocol format.
- Telegraf in **AWS Region 2** uses DDS Consumer Plugin with the Line Protocol format and InfluxDB output plugin.



Results

> *"RTI Connext Databus enables applications to work together as one, integrated system, significantly reducing development, integration and maintenance costs."*
>
> ***Lynne Canavan**, Director of Ecosystems, RTI*

By developing the DDS plugins, RTI has bridged two important technologies and opened very interesting IIoT possibilities. RTI Connext DDS, which integrates with the InfluxDB stack, has proven success in thousands of mission-critical systems including autonomous vehicles, connected medical devices and next-generation energy systems. Below are some sample IIoT customer use cases made possible by Connext DDS.

- **Enabling large-scale system control**
  RTI Connext DDS reduces complexity for NASA's Launch Control and Data Systems through a DDS data exchange infrastructure that provides fault tolerance through passive process replication, message traceability, persistence profile, and content subscription profile capabilities.

- **Advancing precision science**
  European Southern Observatory's (ESO) new 39-meter Extremely Large Telescope (currently in development) will be the world's largest optical telescope. It precisely synchronizes hundreds of servo mirrors and scientific instruments. RTI Connext DDS coordinates precise control and measurement.

- **Bringing autonomy to deep water**
  TechnipFMC does underwater robotics. Connext DDS allows the autonomous and the semi-autonomous operation to work with things that are happening above sea level, as well. So

that through this instrumentation, there is a reliable, secure — that these robotics instruments do not cause any damage down there but actually do what they're intended to do.

"When you are operating on critical infrastructure that is thousands of meters below the ocean's surface, performance and reliability are crucial because the smallest mistake could lead to a multi-billion-dollar disaster. We rely on RTI to support us with the real-time connectivity required," says Bijou Abraham, Chief Software Engineer at TechnipFMC.

- **Modernizing wide-area surveillance**
  BAE Systems is working with Jindalee Operational Radar Network (JORN) in Australia. JORN is their over-the-horizon radar surveillance network that monitors the air and sea movements across northern coastlines. It plays a vital role in supporting the Australian Defence Force's air and maritime operations, border protection, disaster relief and search & rescue operations. Connext DDS provides that connectivity layer of all the data that's coming in so that it can be integrated and reacted to.

  BAE Systems selected RTI Connext DDS as part of the $1.2 billion AUD Upgrade to Australia's JORN. With Connext DDS, the JORN Phase 6 Upgrade Program will modernize the system, significantly increasing the radar coverage area and information gathered for the Royal Australian Air Force, strengthening the protection of Australia's northern borders.

- **Enabling a safe flying car**
  Airbus Vahana is developing the first certified, electric, self-piloted vertical take-off and landing (VTOL) passenger aircraft. RTI Connext DDS is used to operate those vehicles. It addresses diverse systems with the same technology, greatly simplifying design integration and modularity.

- **Connecting autonomous systems**
  Nextdroid, which develops an on-the-ground transportation system, relies on RTI Connext DDS for autonomous vehicle operation — for systems that need to operate for billions of miles over decades. "Fielding autonomous vehicles to consumers requires not only performance and flexibility today, but the safety, security and reliability to deploy at scale.," says Daniel Gandhi, Director of Autonomous Vehicles.

- **Controlling future transport**
  Virgin Hyperloop One is the only company in the world that has built a fully operational hyperloop system. The RTI Connext Databus connects the Hyperloop One faster-than-sound transport (runs the data that runs the hyperloop for this).

- **Finding and delivering the right data**

Siemens is using Connext DDS in the wind turbine farms (can include 500 turbines with 100-meter blades) to collect, analyze, and react to data. Some of these IIoT applications run from the edge to the cloud. There has to be instant communication to ensure that if something goes wrong, it can be reacted to in real time. Turbine control requires fast local loops and maintenance data collection. RTI-controlled wind turbines today generate ~14GW.

- **Sharing wide-area voice and video**
  Harris provides command & control systems for fire, police and emergency response. They are using Connext DDS for emergency response and control. The distributed communication allows them to respond more quickly and to emergencies, with the correct and updated data. Connext DDS meets their critical scalability, routing and recording requirements. It connects GUIs to servers that route voice and video between mobile and fixed points.

About InfluxData

InfluxData is the creator of InfluxDB, the leading time series platform. We empower developers and organizations, such as Cisco, IBM, Lego, Siemens, and Tesla, to build transformative IoT, analytics and monitoring applications. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by sensors, applications and computer infrastructure. Easy to start and scale, InfluxDB gives developers time to focus on the features and functionalities that give their apps a competitive edge. InfluxData is headquartered in San Francisco, with a workforce distributed throughout the U.S. and across Europe. For more information, visit influxdata.com and follow us @InfluxDB.

548 Market St. PMB 77953, San Francisco, CA 94104