In-Memory Computing SUMMIT NORTH AMERICA 2019

Build and Deploy Digital Twins on an IMDG for Real-Time Streaming Analytics

Dr. William L. Bain, Founder & CEO
ScaleOut Software, Inc.
November 13-14, 2019

About the Speaker

Dr. William Bain, Founder & CEO of ScaleOut Software:

- Email: wbain@scaleoutsoftware.com
- Ph.D. in Electrical Engineering (Rice University, 1978)
- Career focused on parallel computing – Bell Labs, Intel, Microsoft
- 3 prior start-ups, last acquired by Microsoft and product now ships as Network Load Balancing in Windows Server

ScaleOut Software develops and markets **In-Memory Data Grids**, software for:

- Scaling application performance with in-memory data storage
- **Operational intelligence**: analyzing live data in real time with in-memory computing

ScaleOut Software

14+ years in the market; 450+ customers, 12,000+ servers

Agenda

- **Goals and challenges** for stream-processing
- What are **real-time digital twins**? Why use them?
- **Advantages** in comparison to traditional approaches
- Target **use cases**
- Using **in-memory computing** to host digital twins
- **New APIs** designed for building digital twins & code sample
- **Implementing digital twin models** on an in-memory data grid (IMDG)
- Deploying real-time digital twin models in a **cloud service**
- **Demo**

Goals of Stream-Processing

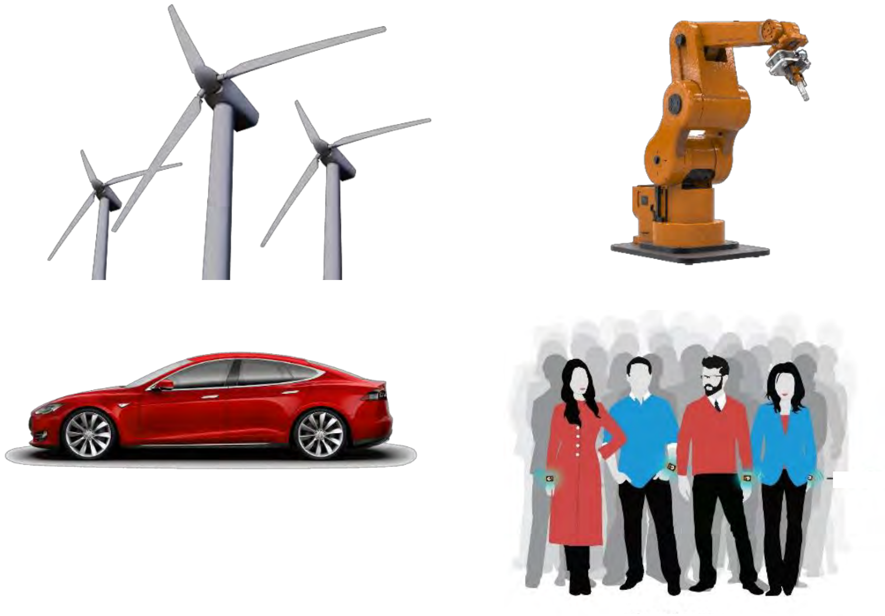
Goal: **maximize situational awareness & real-time control**

How:

- Process incoming data streams from many thousands of devices.
- Analyze events for patterns of interest.
- Provide timely (real-time) feedback and alerts.
- Provide aggregate analytics to identify patterns.

Many applications in IoT and beyond:

- Medical monitoring
- Logistics & manufacturing
- Disaster recovery & security
- Financial trading & fraud detection
- Ecommerce recommendations



Event Sources

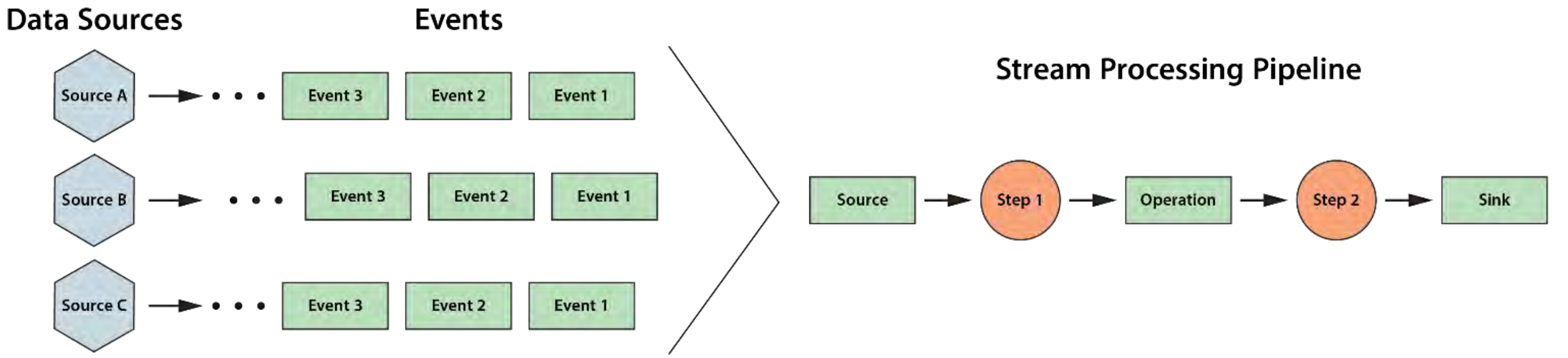
Quick Example: Medical Refrigerators

Cloud-based streaming service monitors 7000+ medical refrigerators:

- Refrigerators hold highly important tissue samples, embryos, etc.
- Service receives periodic telemetry:
  - Temperature
  - Power consumption
  - Door position, etc.
- Must predict failure before it occurs:
  - Notify user to migrate contents to another refrigerator.
  - Avoid false positives.
  - Identify widespread power outages.

Challenges for Stream Processing

Popular software platforms (Flink, Storm, Beam) are **pipeline-oriented**.



Creates **complexity challenges**:

- Difficult to: correlate events by each data source, track state, embed analytics

Creates **performance challenges**:

- Difficult to: respond with low latency, scale for thousands of data sources

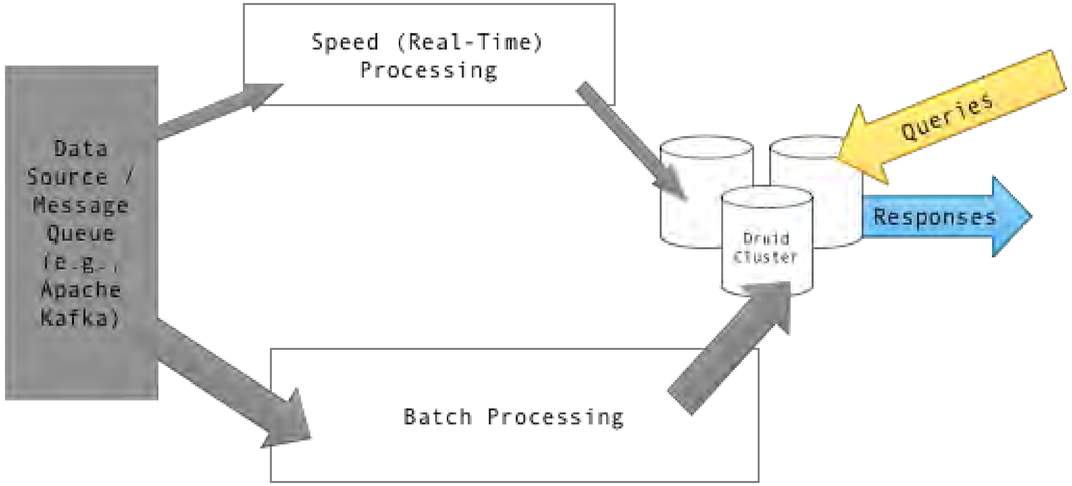
Requires **aggregate analytics** to be performed **offline**.

Typical Approach: Lambda Architecture

Adds complexity to applications that provide real-time analytics:

- Separates real-time processing ("speed layer") from data-parallel analytics ("batch layer").
- Allows only rudimentary analysis and response in real time.
- Defers aggregate analysis to offline processing (e.g., Spark, database query).
- Limits real-time introspection.

Is there a better approach?

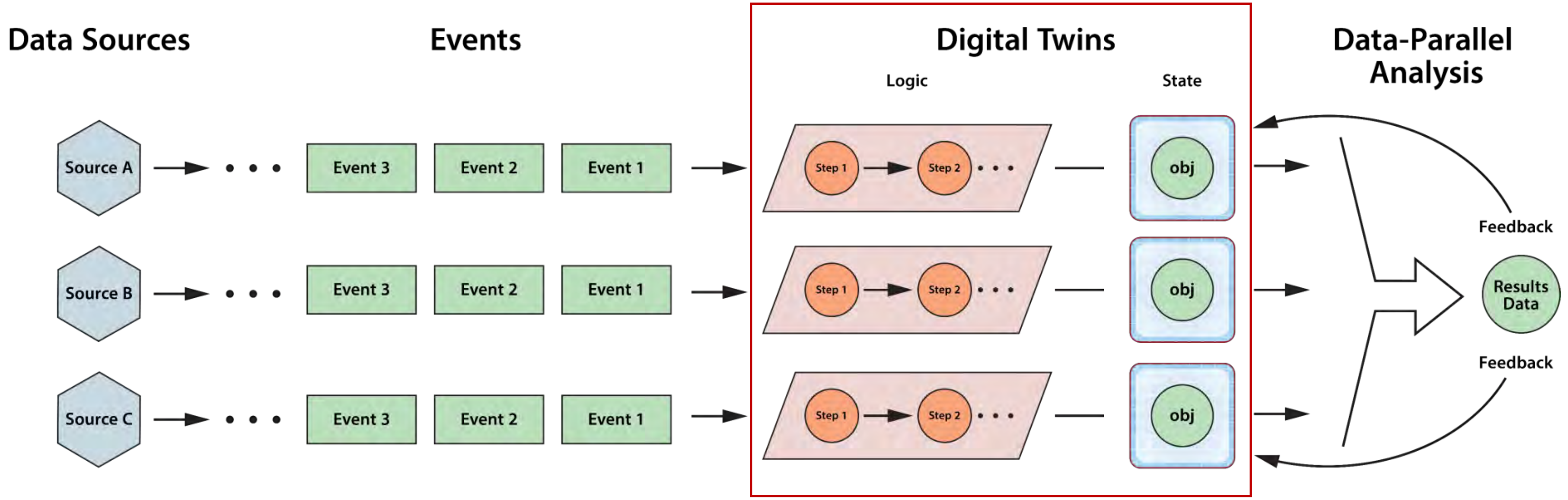


https://commons.wikimedia.org/w/index.php?curid=34963987

Real-Time Digital Twins

A **new software technique** for stream-processing:

- Automatically **correlates telemetry** from each device or data source.
- **Tracks dynamic state** for each data source.
- Provides a **software framework** for hosting application logic (e.g., rules, ML).
- Enables **real-time aggregate analysis** in place.



Other Uses of the Term “Digital Twin”

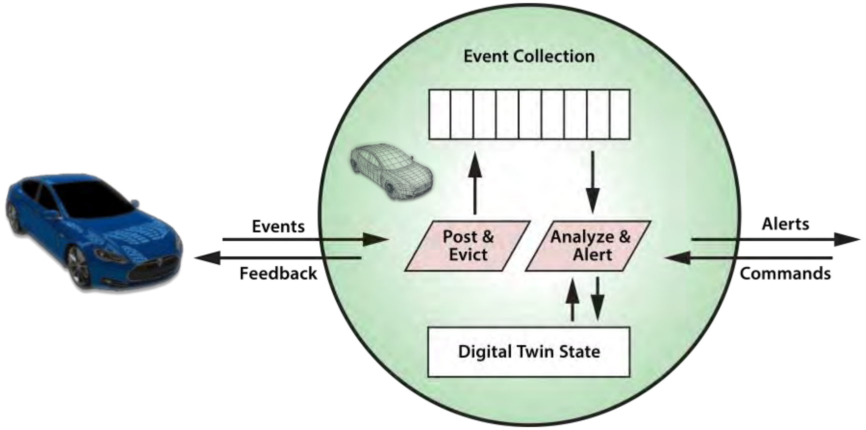
- Created by Michael Grieves for product design and life cycle management (PLM); popularized by Gartner:
  - A virtual version of a physical entity
  - Also, context to interpret telemetry streaming back from the field

- Also:
  - **AWS device shadow**: cloud-based repository for per-device state information with pub/sub messaging
  - **Azure IoT device twin**: JSON document that stores per-device state information (metadata, conditions)
  - **Azure digital twin**: spatial graph of spaces, devices, and people for modeling relationships in context
- These uses are *not* for **real-time stream processing**.

Anatomy of a Real-Time Digital Twin

A **real-time digital twin model** describes how to process incoming events from a specific type of data source (e.g., a wind turbine).

- Consists of a message processor method and a state object definition:
- **Message processor**:
  - Receives and analyzes events and commands.
  - Encapsulates analysis algorithm.
  - Generates alerts and outbound device messages.
- **State object** holds dynamic, per-device data:
  - Dynamic context for analyzing events
  - Also: time-ordered event lists, cached parameters
  - One **instance** per data source (device)



Comparison: Two Types of Digital Twins

A real-time digital twin is ***not* a PLM model** of a physical device:

PLM Digital Twin	Real-Time Digital Twin
Goal: Aid in product development.	Goal: Aid in real-time streaming analytics.
Models characteristics and behavior of a physical device (simulation model).	Analyzes telemetry streams from a physical device & generates feedback and alerts.
Proactively generates outputs over time and accepts inputs.	Reactively processes telemetry messages and commands.
Implements dynamic state that models device behavior.	Implements dynamic state that adds context to help interpret telemetry.
Example: digital twin for a medical refrigerator:	
Models door open/close events, temperature changes over time, power fluctuations, etc.	Analyzes incoming events based on maintenance history, usage, and condition.

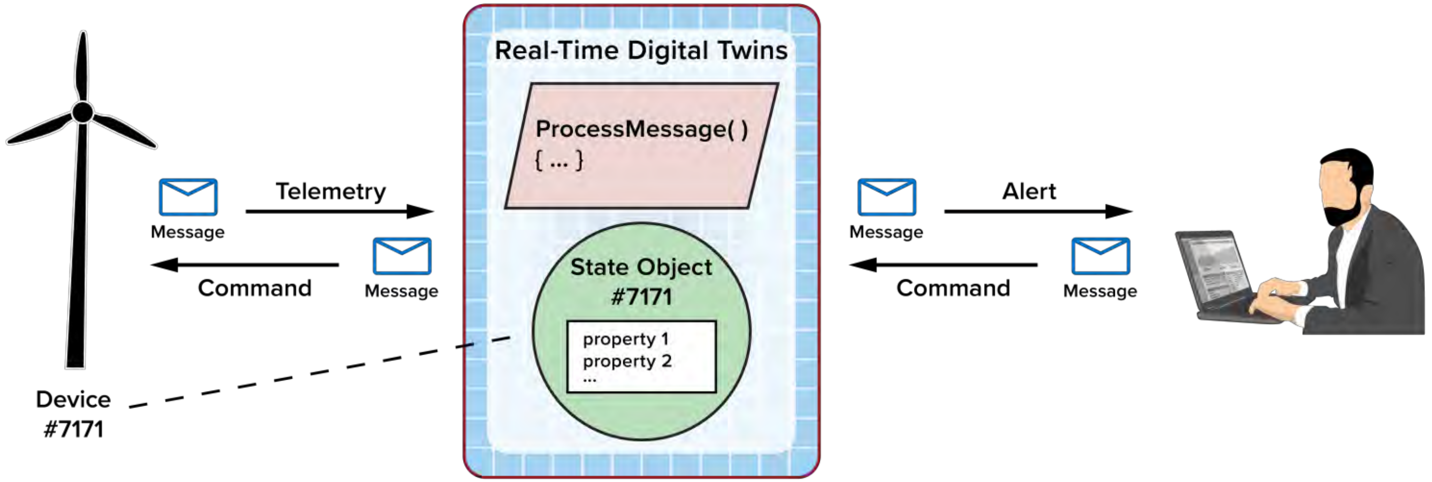
Advantages of Real-Time Digital Twins

Simplifies application design:

- Provides automatic event correlation and access to per-device state.
- Uses an object-oriented approach to encapsulate state and behavior.

Enables **deeper introspection** in real time:

- Dynamically tracks state of each device to help analyze incoming events.
- Provides orchestration for analytics code (e.g., rules engine, ML).
- Enables integrated, aggregate analysis.

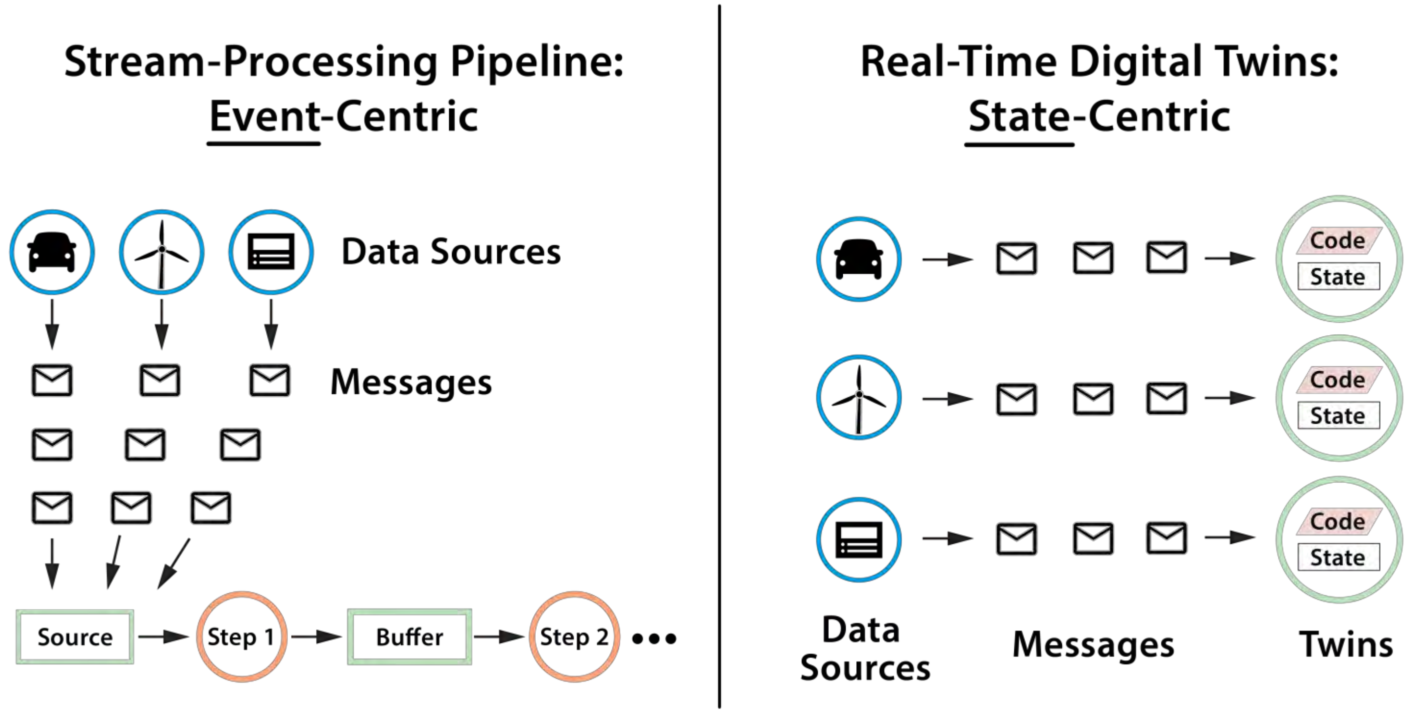


Runs well on IMDGs.

Simplifies Application Design

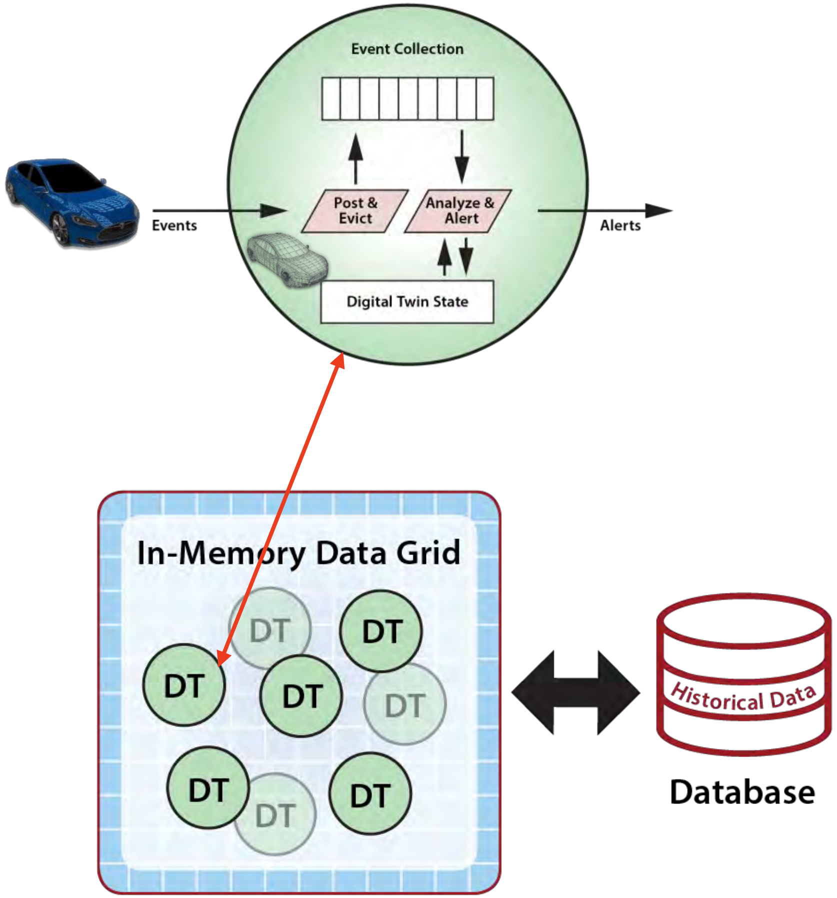
State-centric approach (vs. event-centric):

- Avoids event correlation in the application.
- Avoids need for *ad hoc* state storage.
- Encapsulates analysis logic in one place.
- Provides automatic domain for aggregate analysis.



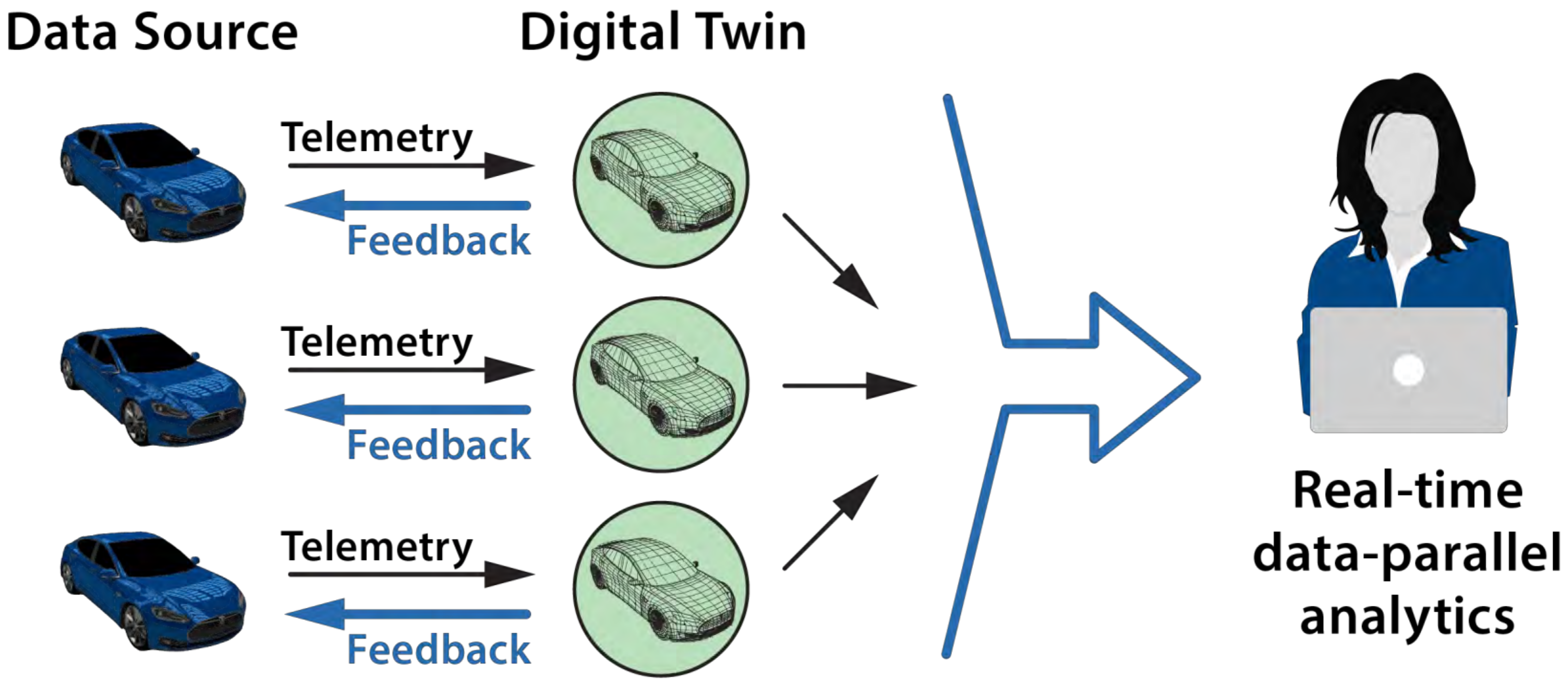
Digital Twins Can Access Historical State

- Digital twins store dynamic state information in memory for fast access.
- Also can retrieve slowly-changing data from a database:
  - Device parameters
  - Maintenance history
- Can update database:
  - Event-message history
  - Significant changes to the device



Enables Aggregate Analysis

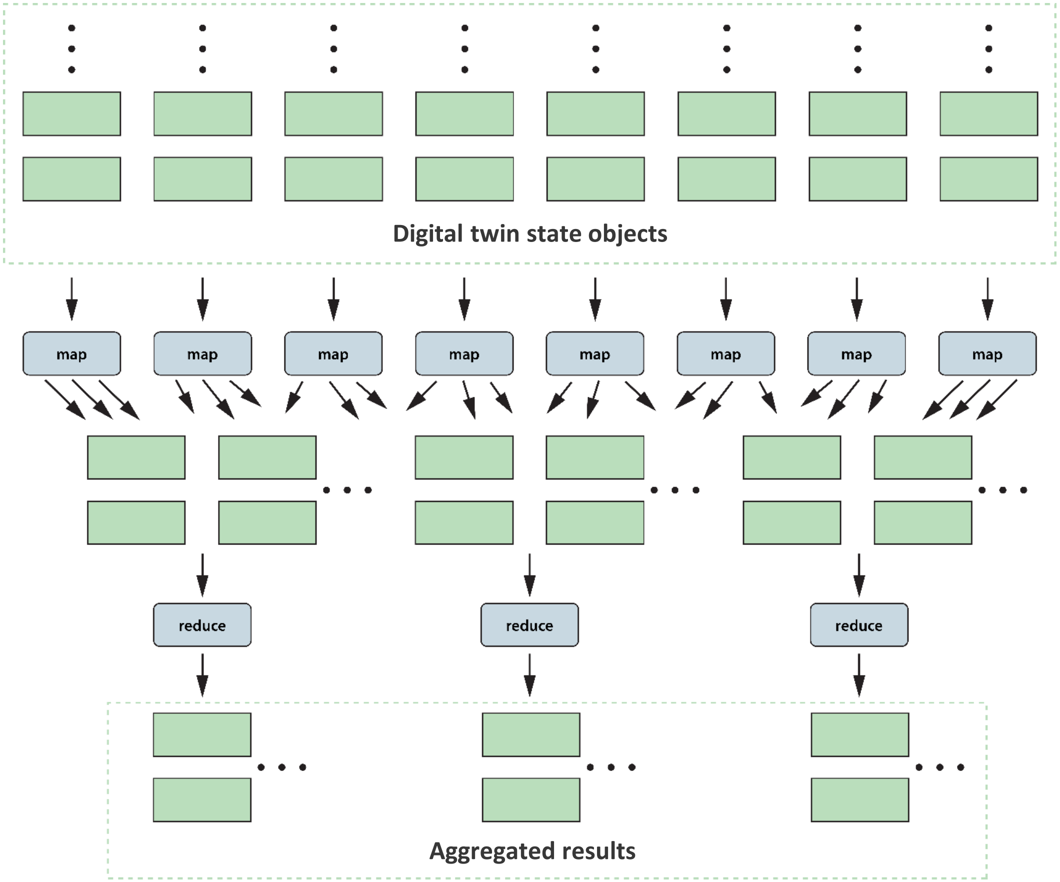
Real-time digital twins create a natural **domain for data-parallel analysis**:



Aggregate Analysis with MapReduce

A well-known, data-parallel technique:

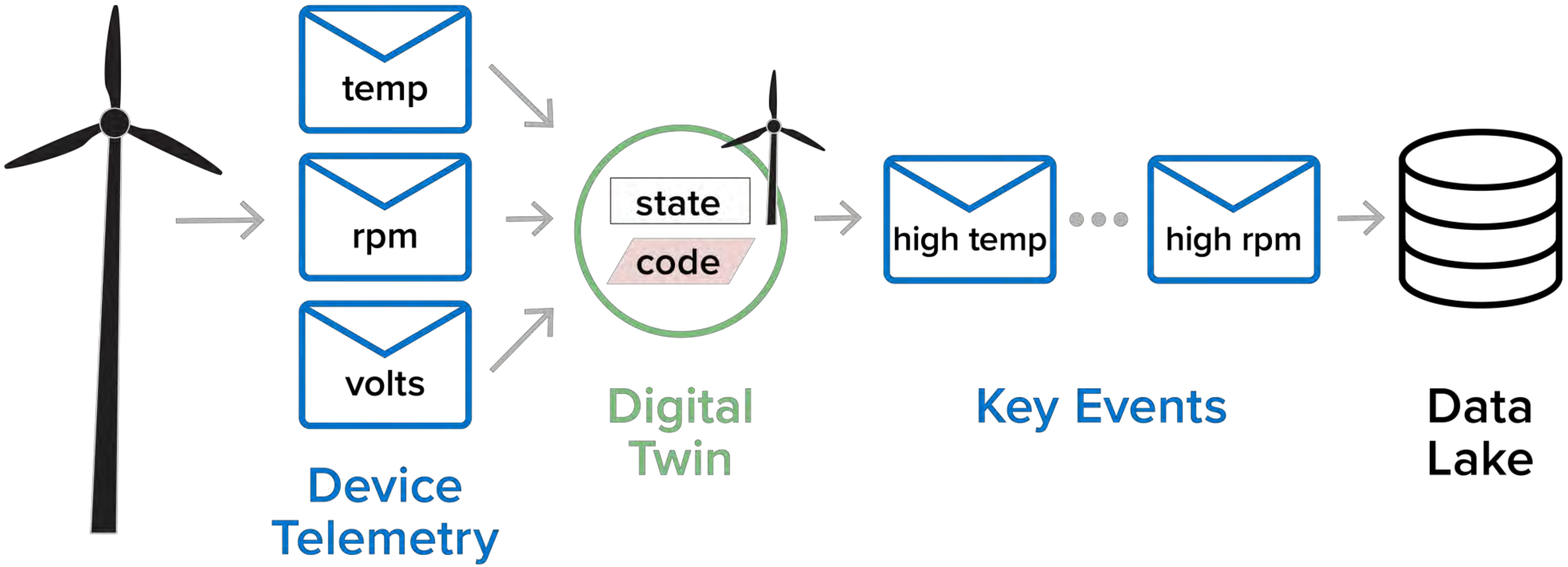
- Aggregates property values across all instances of a model.
- Allows results to be grouped according to the value of another property.
  - Example: Ave. vehicle speed by county
- Runs seamlessly within an IMDG:
  - Runs concurrently with event processing.
  - Avoids network bottlenecks.
  - Avoids delay for offline processing.



MapReduce Data Flow

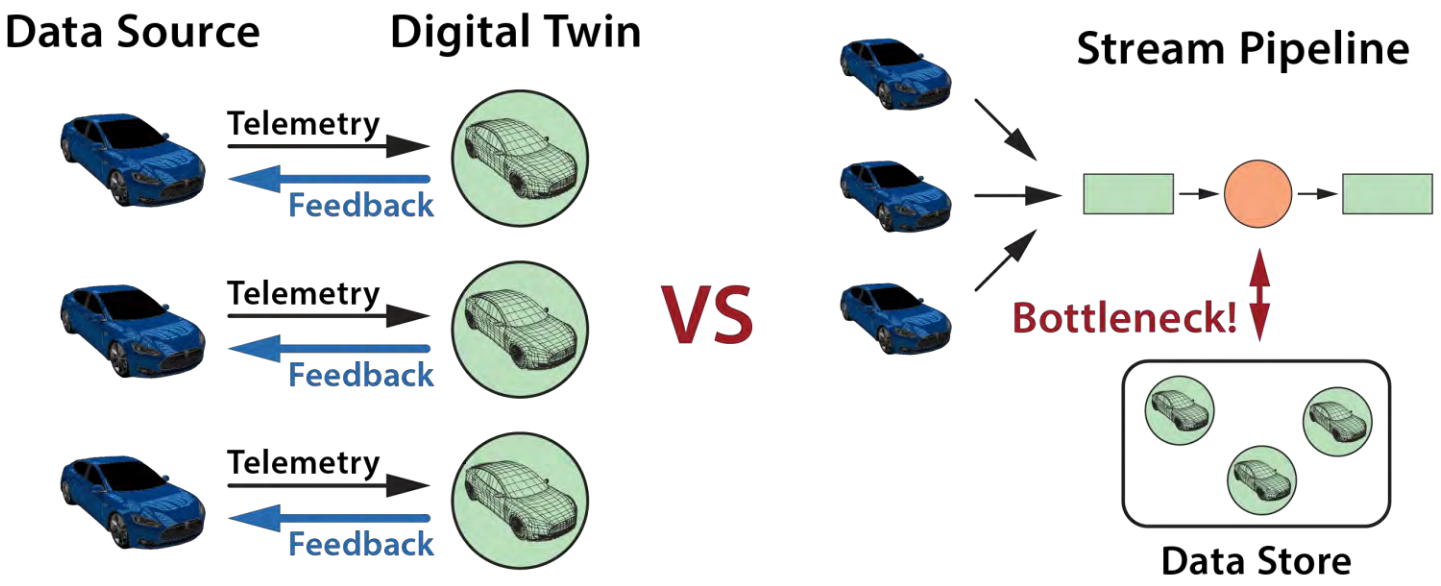
Also Enables Telemetry Filtering

Real-time digital twins can filter events for offline analysis in the data lake:



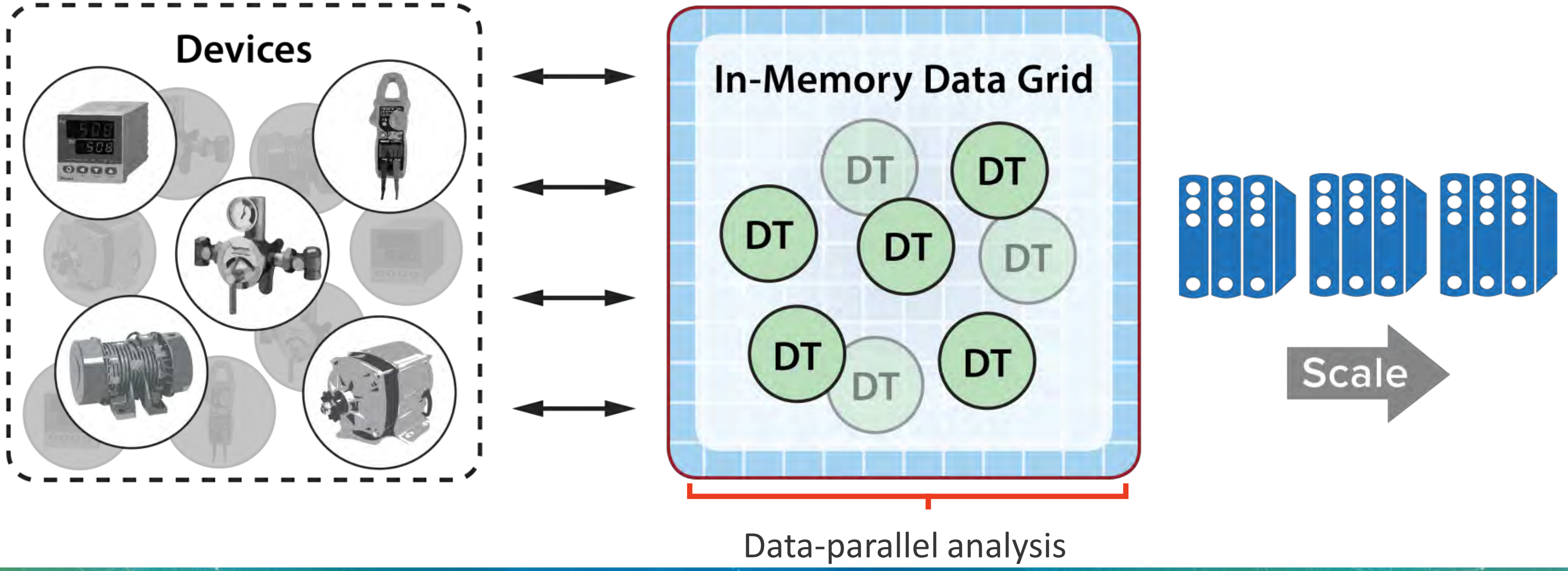
Avoids Network Bottlnecks

- State-centric approach distributes events across state objects.
- Avoids network bottleneck accessing remote data store from event pipeline.
  - Network bottlenecks prevent scalable throughput.



Leverages In-Memory Computing

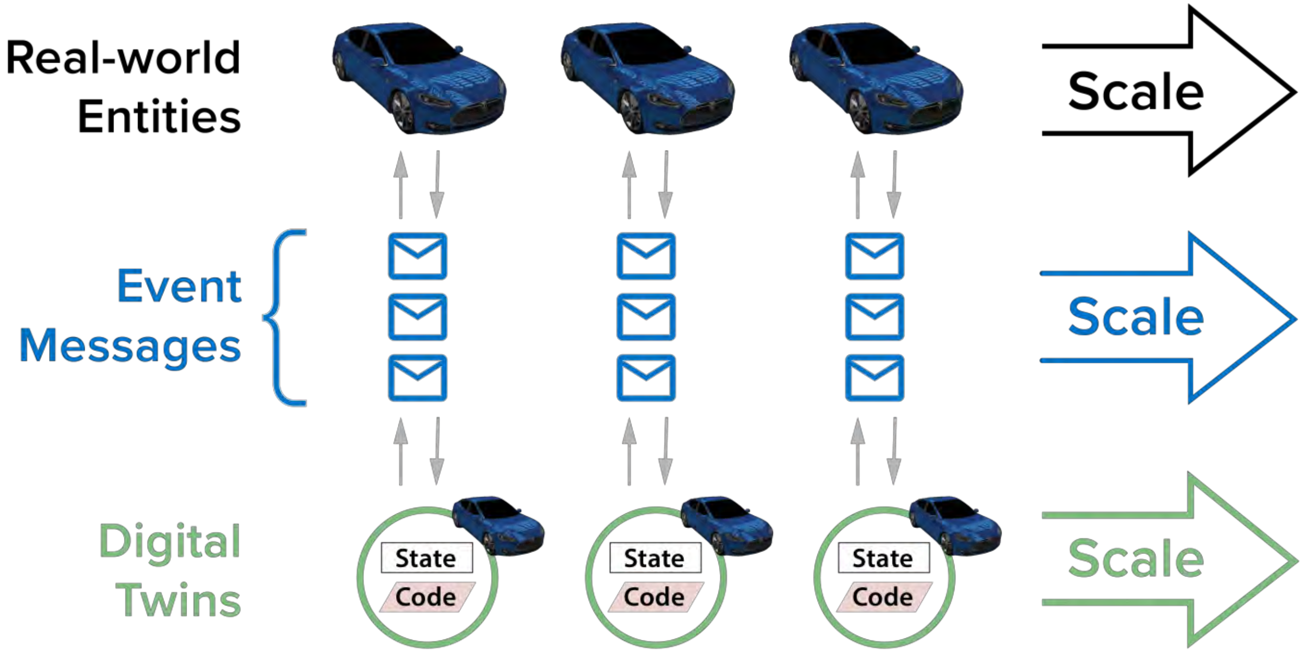
- State objects can be hosted within an in-memory data grid (IMDG).
- IMDG delivers event messages to state objects and runs message processor.
- IMDG can perform data-parallel analysis <u>in place</u> across state objects.



IMDG Delivers Fast, Scalable Performance

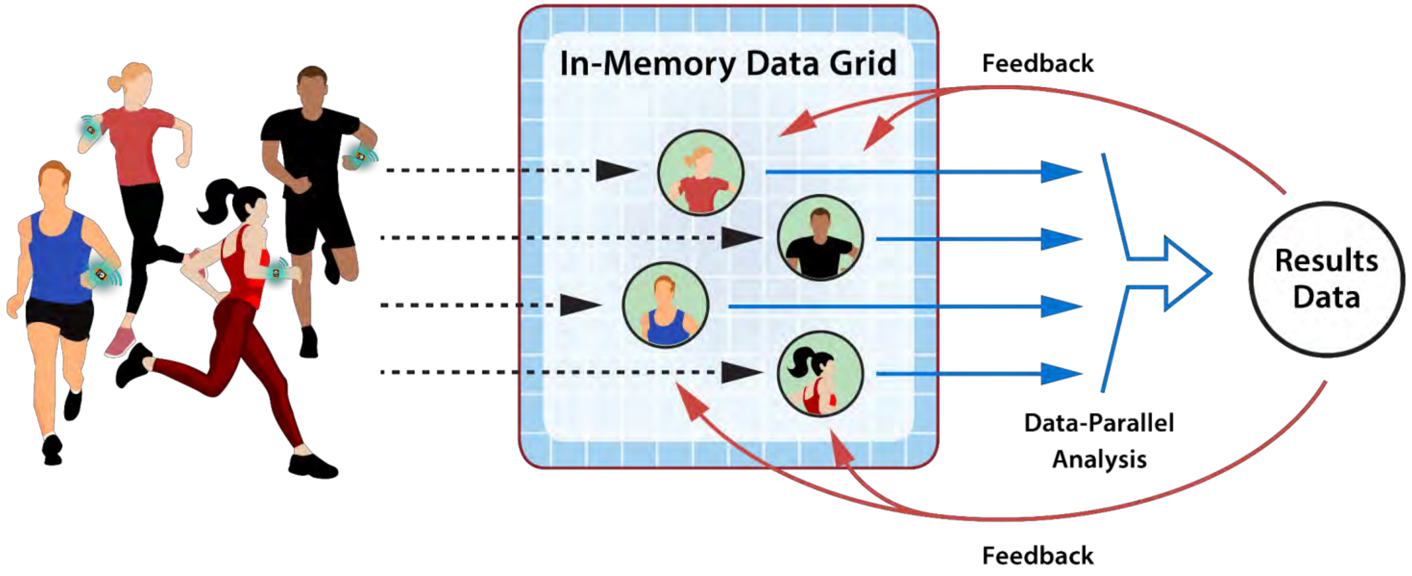
In-memory data grid:

- Processes event message in 1-2 milliseconds.
- Performs typical data-parallel analysis in ~1-5 seconds.
- Transparently scales to handle 100,000+ digital twin instances.



Target Use Cases for Digital Twins

- Useful in applications which require **fast response times** and **situational awareness**
- Benefit from real-time aggregate analysis
- Examples:
  - Health tracking
  - Disaster recovery
  - Security monitoring
  - Fleet management
  - Ecommerce recommendations
  - Fraud detection

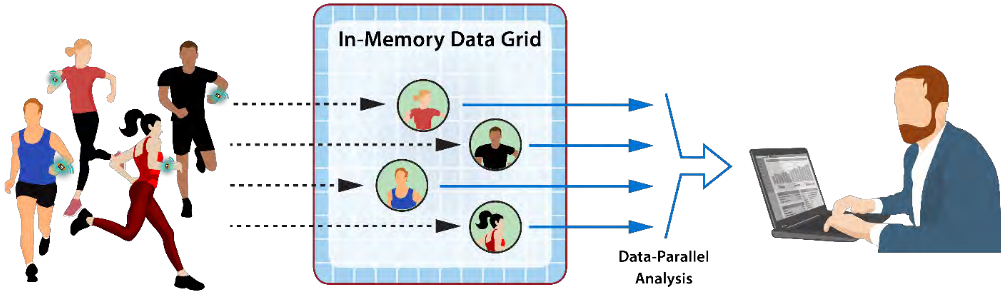


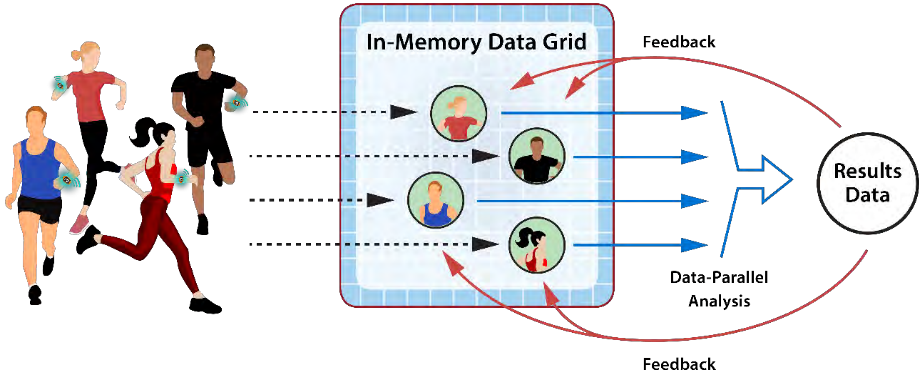
Example: Telemetry and Feedback from Wearable Devices

Real-Time Health Tracking

Digital twins analyze telemetry from health-tracking devices to help ensure safety (predict events):

- Digital twins receive periodic messages with key metrics (heart rate, blood oxygen, etc.).
- State objects track person's health history, medications, limitations, recent medical events.
- Analysis algorithm can integrate dynamic, aggregate results from large populations.



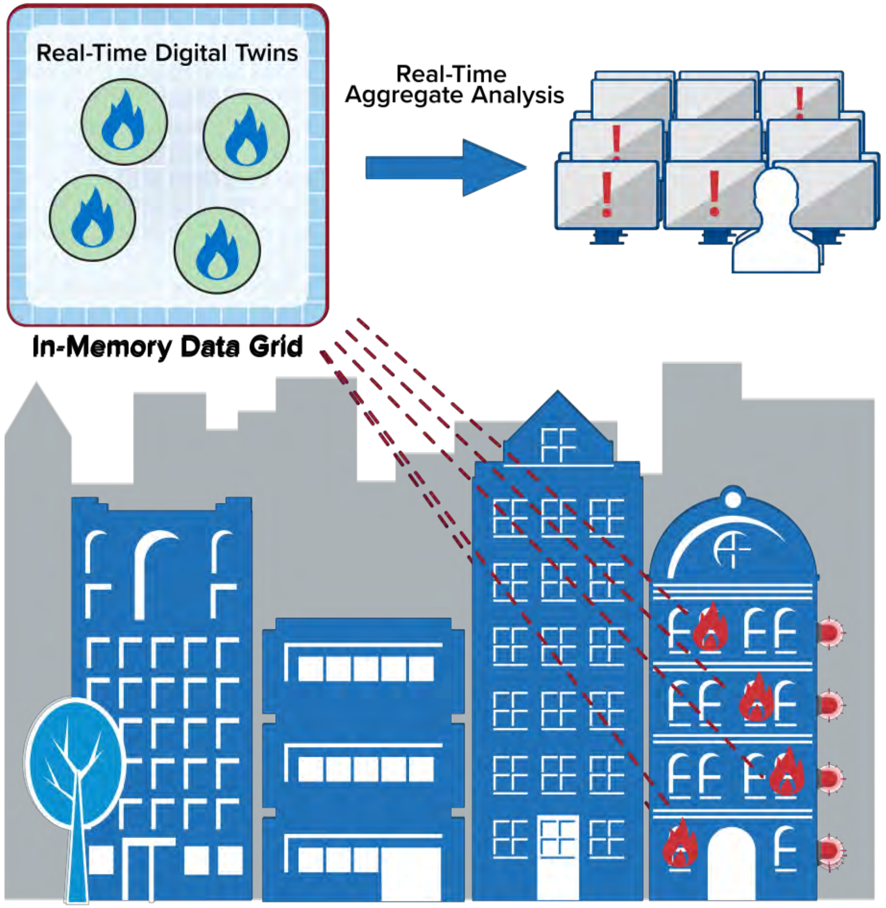


Disaster Recovery

Digital twins analyze telemetry from sensors to determine scope of an incident in real time.

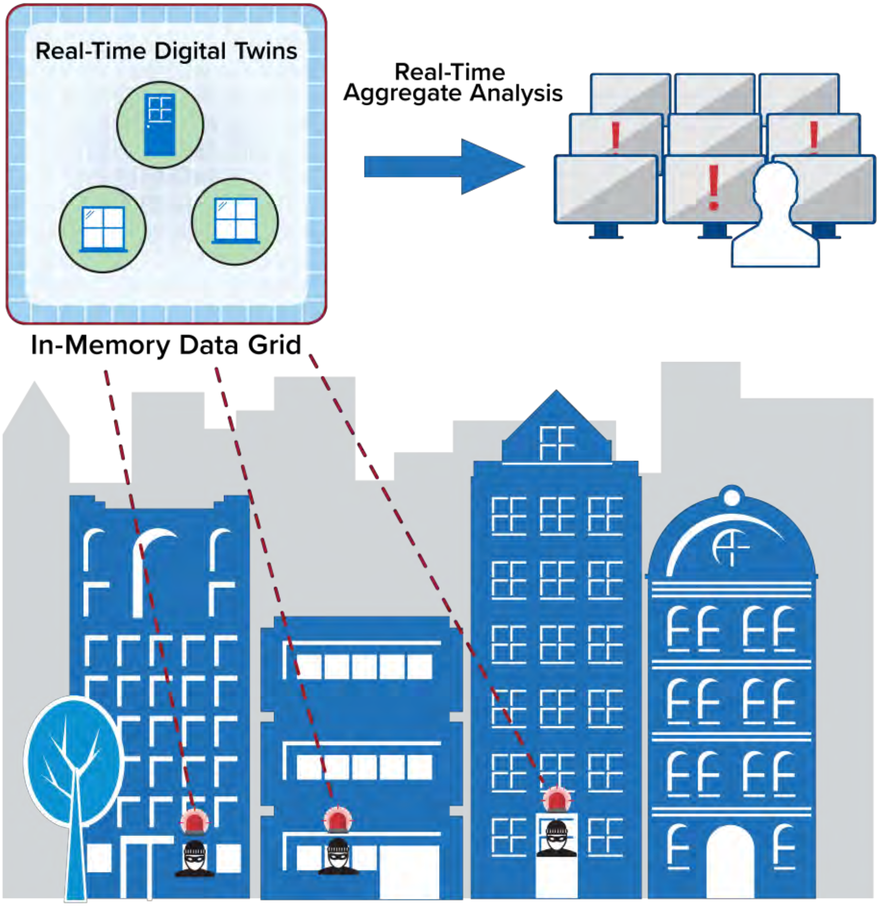
Example: intelligent fire alarm system

- Analysis of sensor telemetry indicates probable or impending fire.
- **Aggregate analysis** of multiple sensors indicates path & extent of fire.
- Enables intelligent evacuation strategy.



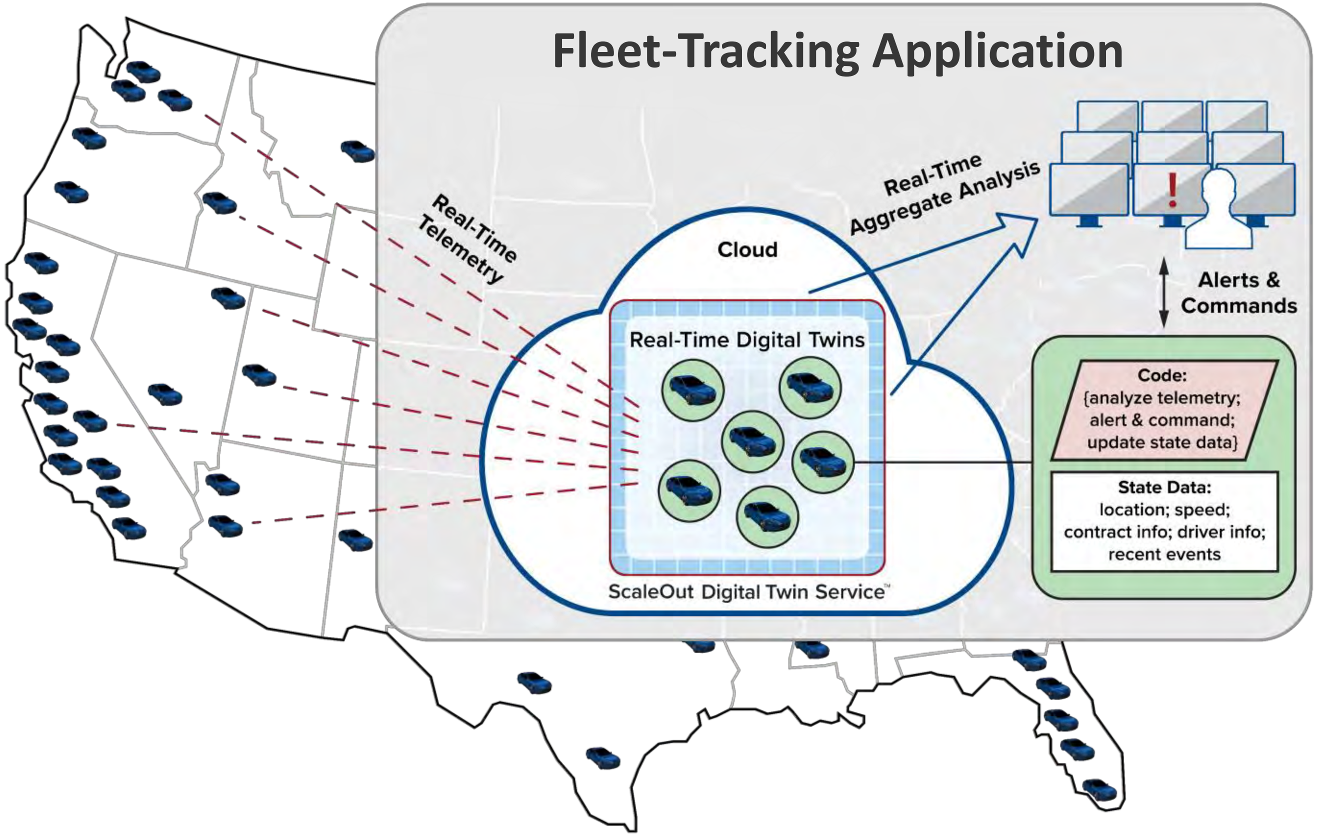
Security Monitoring

- Intrusion sensors analyze telemetry to predict unauthorized access at each location.
- **Aggregate analysis** of perimeter sensors indicates scope of threat.
- Enables focused, real-time response to all critical locations.



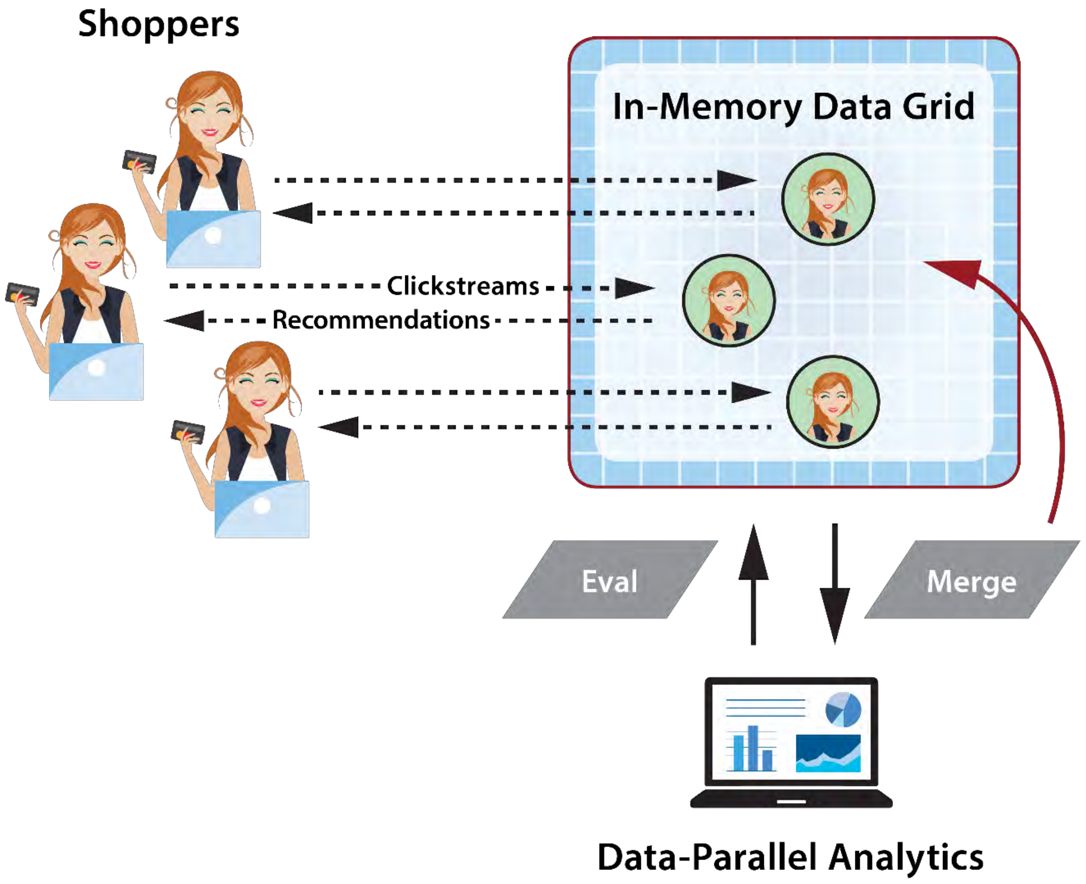
Large Scale Fleet Tracking

- Real-time tracking for a car/truck fleet
  - 100K+ vehicles
- Immediately responds to issues with individual vehicles:
  - Lost driver, engine failure, etc.
- Detects & responds to regional issues within seconds
  - Weather delays, highway blockages
  - Redirects drivers.



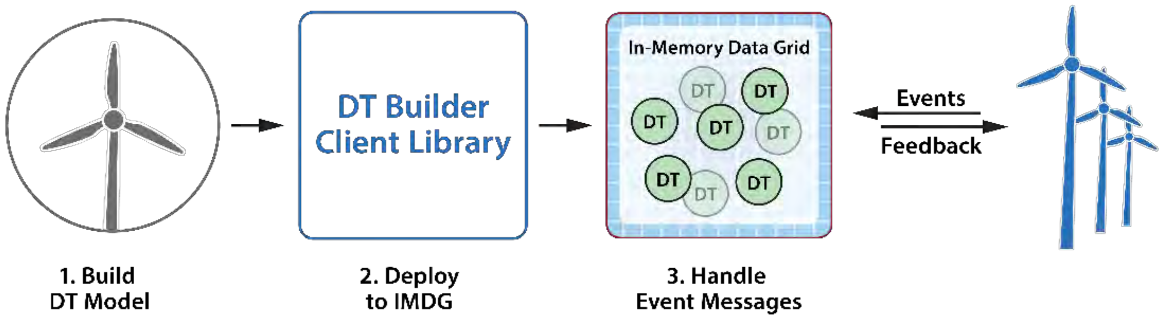
Ecommerce Recommendations

- Ecommerce site may have 100k+ shoppers, each generating a clickstream.
- Digital twin for each shopper:
  - Maintains a history of clicks, shopper's preferences, and purchasing history.
  - Analyzes clicks to create new recommendations in real time.
- Aggregate analysis:
  - Determines collaborative shopping behavior, basket statistics, etc.
  - Enables targeted, real-time flash sales.

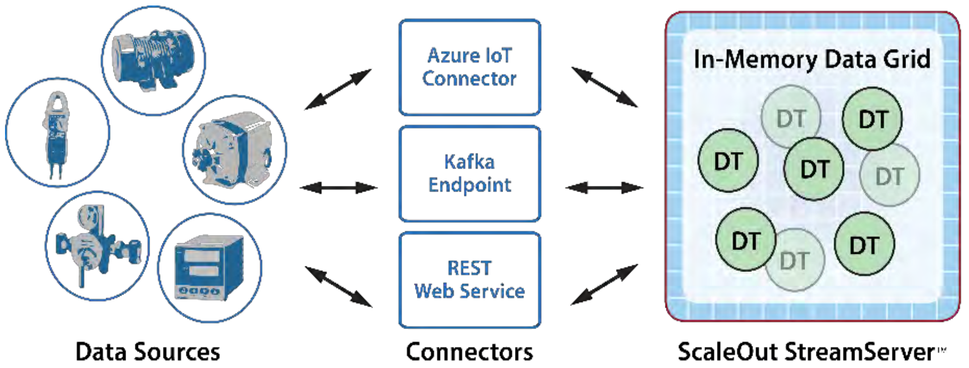


Building and Deploying Digital Twins

- **Step 1**: Build a digital twin model and deploy to the IMDG:



- **Step 2**: Connect the IMDG to a message hub (e.g., Azure IoT Hub, AWS IoT, Kafka, REST, etc.):



Why Use Specific APIs for Digital Twins?

- **Simplifies application design**; avoids complexity of underlying IMDG APIs, including:
  - Explicitly managing and accessing state objects in the IMDG
  - Orchestrating the staging of message-processing code across the IMDG
  - Connecting digital twins to data sources
  - Delivering messages to digital twins and back to data sources
  - Ensuring highly available message handling
- Digital twin APIs and services allow the application to focus on:
  - Defining message-processing code for each type of data source
  - Defining the dynamic state information to be managed for each data source
  - Describing periodic data-parallel analytics to be performed across all digital twins of a given type

Digital Twin Builder APIs

- Application implements a message processor method:

  `ProcessMessage(stateObject, processingContext, messageList)`

- Application defines state object to hold instance properties and optional event lists.
- Processing context defines APIs for sending messages to data source or to other twins.
- Message list contains set of messages that arrived since last call to `ProcessMessage`.
  - Hides latency by handling multiple messages at once.
  - Enables single acknowledgment for a group of messages.

Deployment APIs

- Deploy model to IMDG:

```
builder = new ModelBuilder()
              .AddDependency("code.dll")
              .AddModel<stateObjectType,
                        messageProcessorType,
                        eventMessageType>()
              .Build();
```

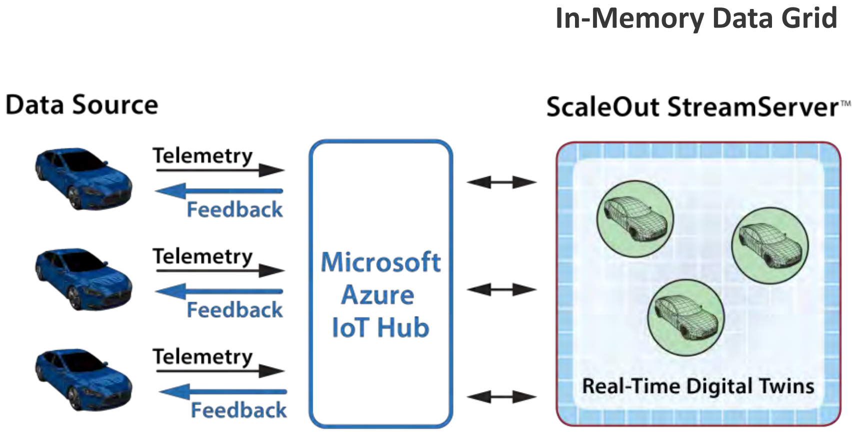
- Deploys model's code to the IMDG.
- Starts message processing.
- Automatically creates a digital twin instance for each new data source id.

Connecting to a Message Hub

- Typical message hubs: Azure IoT Hub, AWS IoT, Kafka, REST
- A connector creates a message path to/from the IMDG and a hub:

```
connector = new XYZConnectionManager(name, connParameters);
```

- Authenticates connection to the message hub.
- Awaits messages from data sources.
  - Uses multiple listeners if supported by the hub.
- Forwards messages to digital twin instances or creates an instance for a new data source.
- Manages acknowledgments for high availability.



Code Sample: Wind Turbine Digital Twin

Goal: Analyze temperature telemetry from a wind turbine.

- Digital twin state object tracks:
  - Parameters: model, pre-maintenance period based on model, max. allowed temperature, max. allowed over-temp duration (normal and pre-maintenance)
  - Dynamic state: time to next maintenance, over-temp condition and its duration
- Message processor:
  - Determines onset of and recovery from over-temp condition.
  - Alerts at maximum allowed duration; logs incidents for time-windowing analysis.

Block Island Wind Farm

Sample State Object (C#)

```
[JsonObject]
public class WindTurbine : DigitalTwinBase
{
    // physical characteristics:
    public const string DigitalTwinModelType = "windturbine";
    public WindTurbineModel TurbineModel { get; set; } = WindTurbineModel.Model7331;
    public DateTime  NextMaintDate { get; set; } = new DateTime().AddMonths(36);
    public const int MaxAllowedTemp = 100; // in Celsius
    public TimeSpan  MaxTimeOverTempAllowed = TimeSpan.FromMinutes(10);
    public TimeSpan  MaxTimeOverTempAllowedPreMaint = TimeSpan.FromMinutes(2);

    // dynamic state variables:
    public bool TrackingOverTemp      { get; set; }
    public DateTime OverTempStartTime { get; set; }
    public int NumberMsgsWithOverTemp { get; set; }

    // list of incidents and alerts:
    public List<Incident> IncidentList { get; } = new List<Incident>();
}
```

Sample Message Processor (Outer Loop)

```
public override ProcessingResult ProcessMessages(ProcessingContext context,
     WindTurbine dt, IEnumerable<DeviceTelemetry> newMessages)
{
    var result = ProcessingResult.NoUpdate;

    // determine if we are in the pre-maintenance period for this wind turbine model:
    var preMaintTimePeriod  = _preMaintPeriod[dt.TurbineModel];
    bool isInPreMaintPeriod = ((dt.NextMaintDate
             - DateTime.UtcNow) < preMaintTimePeriod) ? true : false;

    // process incoming messages to look for over-temp condition:
    foreach (var msg in newMessages) {
        // if message reports a high temp indication, track it:
        if (msg.Temp > WindTurbine.MaxAllowedTemp)
            <track over-temp condition>
        else if (dt.TrackingOverTemp)
            <resolve over-temp condition>
    }
    return result;}
```

Track/Resolve Over-temp Condition

```
// track over-temp condition:
{dt.NumberMsgsWithOverTemp++;

if (!dt.TrackingOverTemp) {
    dt.TrackingOverTemp  = true; dt.OverTempStartTime = DateTime.UtcNow;
    <add a notification to the incident list> }

TimeSpan duration = DateTime.UtcNow - dt.OverTempStartTime;

// if we have exceeded the max allowed duration for an over-temp, send an alert:
if (duration > dt.MaxTimeOverTempAllowed ||
       (isInPreMaintPeriod && duration > dt.MaxTimeOverTempAllowedPreMaint)) {
    var alert = new Alert(); <fill out the alert message>;
    context.SendToDataSource(Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(alert)));
    <add a notification to the incident list> }}
```

```
// resolve the condition and reset our state:
{dt.TrackingOverTemp = false; dt.NumberMsgsWithOverTemp = 0;
 <add a notification to the incident list> }
```

Deploy the Model and Connect to a Hub

- Deploy the wind turbine model:

```
ExecutionEnvironmentBuilder builder = new ExecutionEnvironmentBuilder()
    .AddDependency(@"WindTurbine.dll")
    .AddDigitalTwin<WindTurbine, WindTurbineMessageProcessor,
                    DeviceTelemetry>(WindTurbine.DigitalTwinModelType);
```

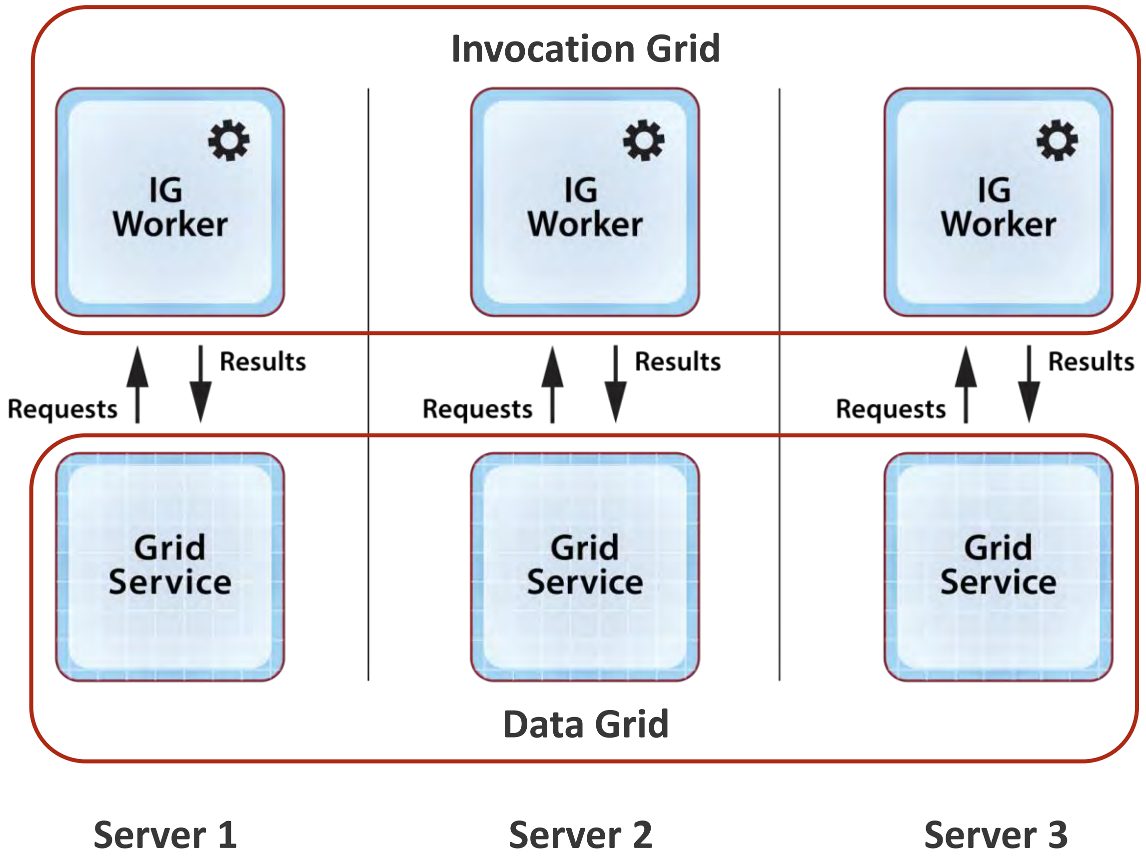
- Connect to Azure IoT Hub:

```
EventListenerManager.StartAzureIoTHubConnector(
 eventHubName                : _eventHubName,
 eventHubConnectionString    : _eventHubConnectionString,
 eventHubEventsEndpoint      : _eventHubEventsEndpoint,
 storageConnectionString     : _storageConnectionString,
 consumerGroupName           : "");
```

How an IMDG Stores Data & Runs Code

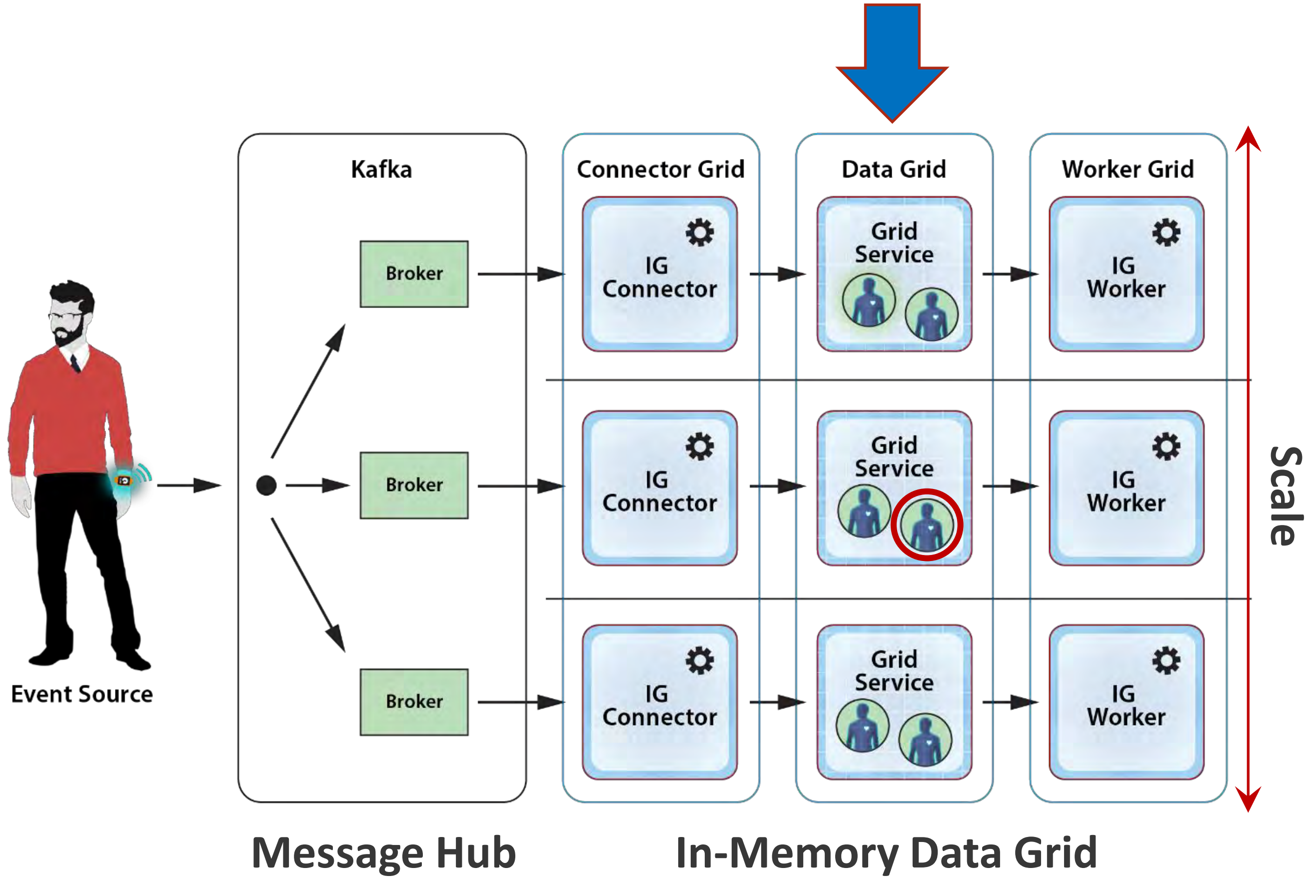
IMDG transparently scales data storage and method execution across multiple servers:

- Stores serialized objects in a **Data Grid**.
- Runs methods in an **Invocation Grid**.
- Each IG Worker process:
  - Hosts a language-specific runtime.
  - Processes requests and accesses objects from its co-located Grid Service process.



How an IMDG Runs Digital Twin Models

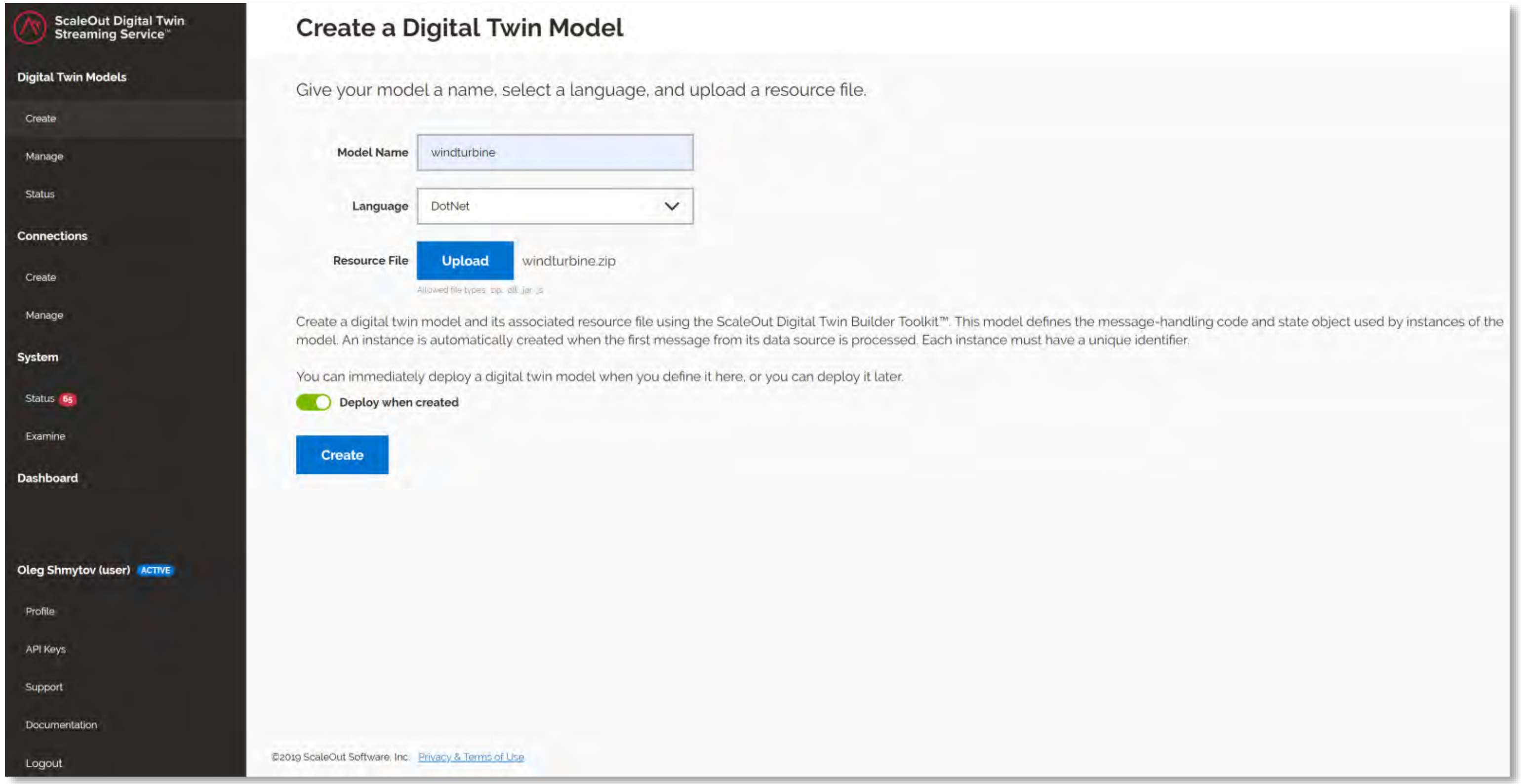
- Digital twin instances are hosted as objects in the Data Grid.
- Digital twin models run in an IG called the Worker Grid.
- Connectors run in an IG called the Connector Grid.
- Connectors invoke message processor on the server hosting the device's instance object.
  - Steers messages to object by id.
  - This minimizes network overhead.



Deploying a Digital Twin to the Cloud

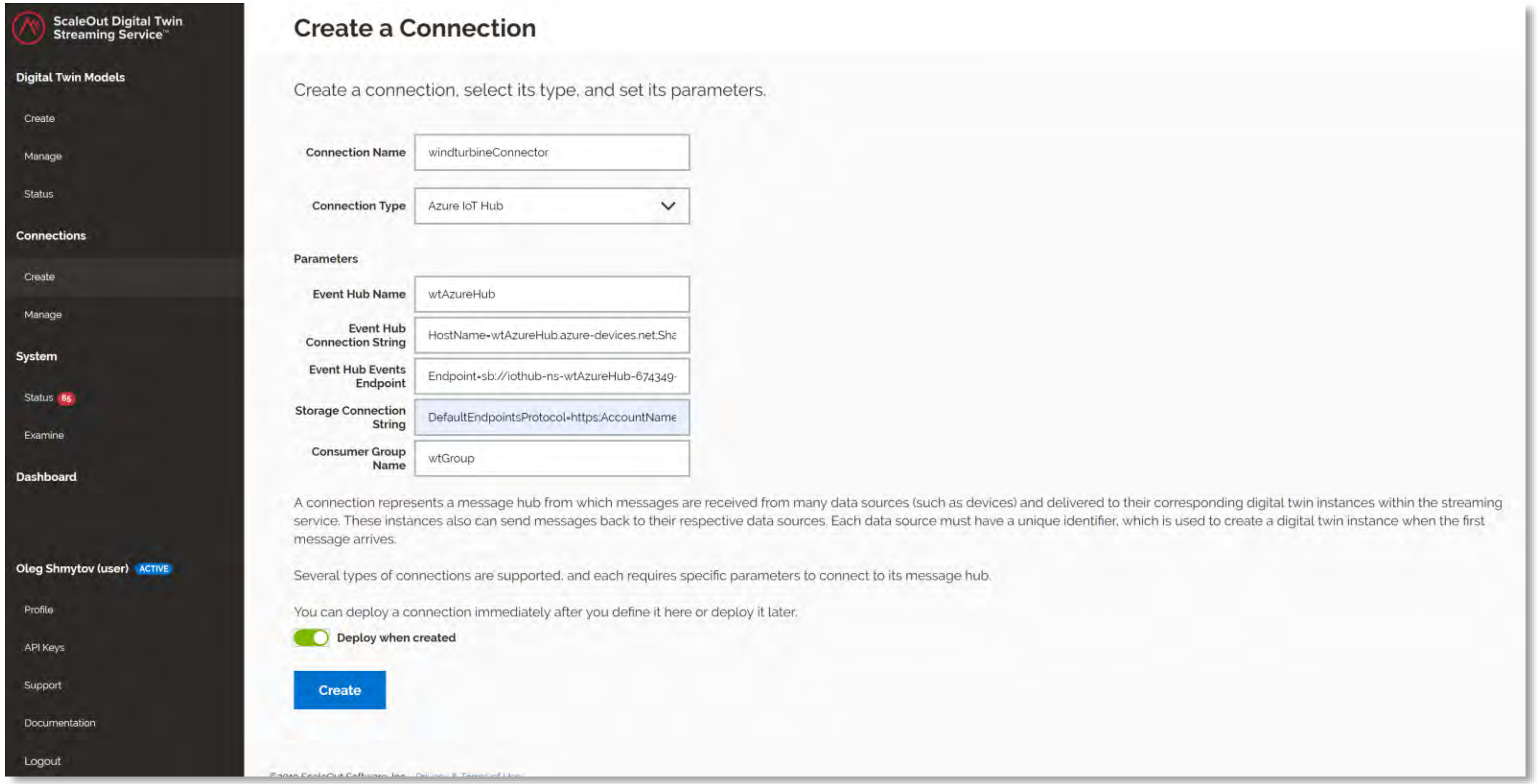
Preview of a UI for a cloud service that hosts digital twins:

- Model is first created using APIs.
- UI uploads code from a resource file.
- UI selects language runtime, such as Java, C#, JavaScript.



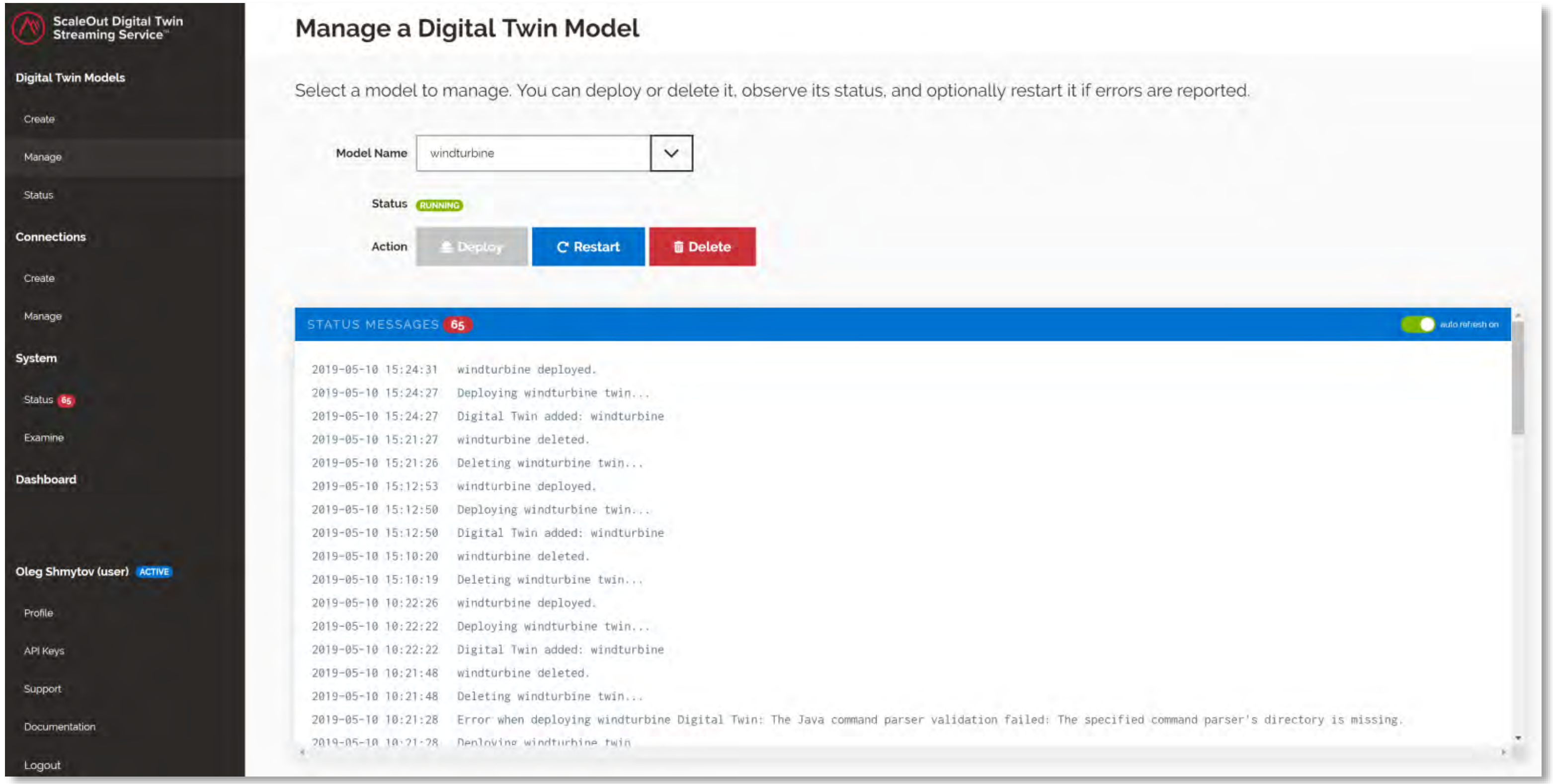
Deploying a Connector to the Cloud

Connectors can be created by specifying the hub type and connection parameters:



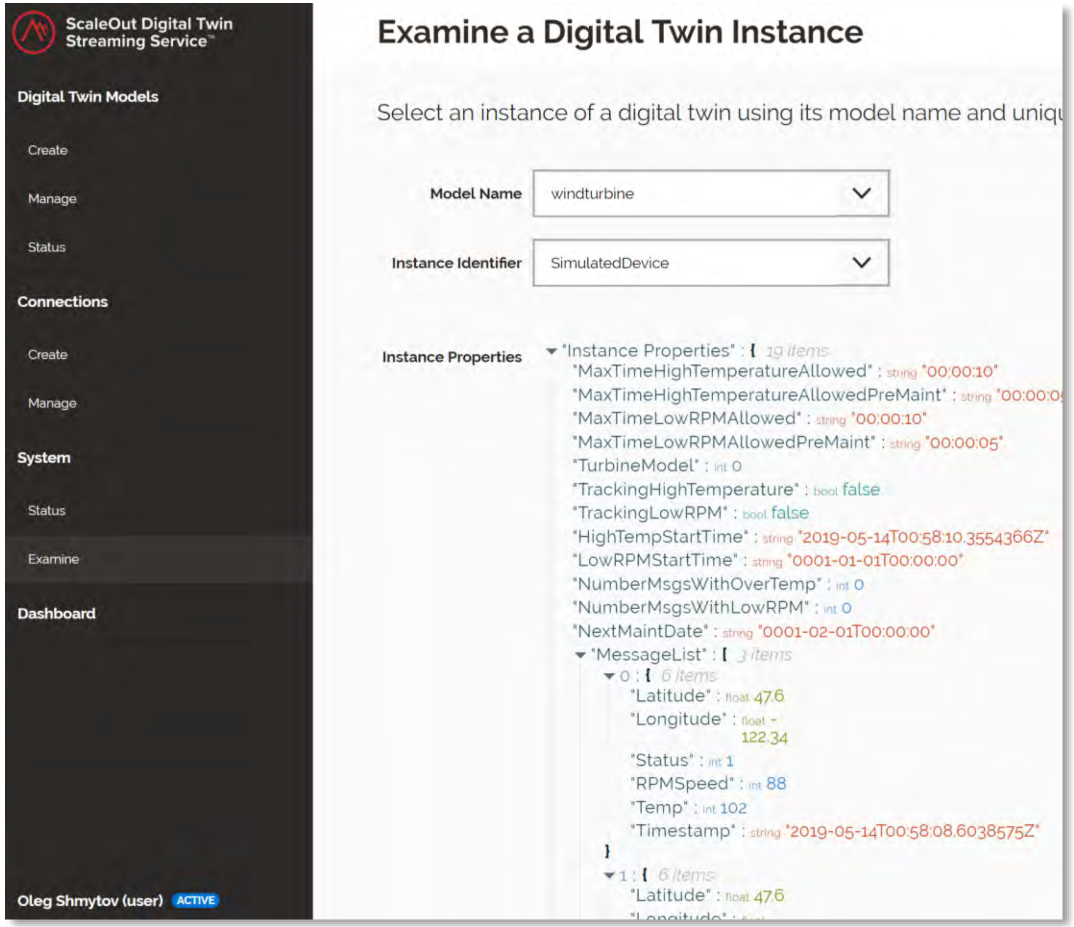
Managing Digital Twin Models in the Cloud

Each model can be independently managed to check status and restart as necessary:



Examining a Digital Twin Instance

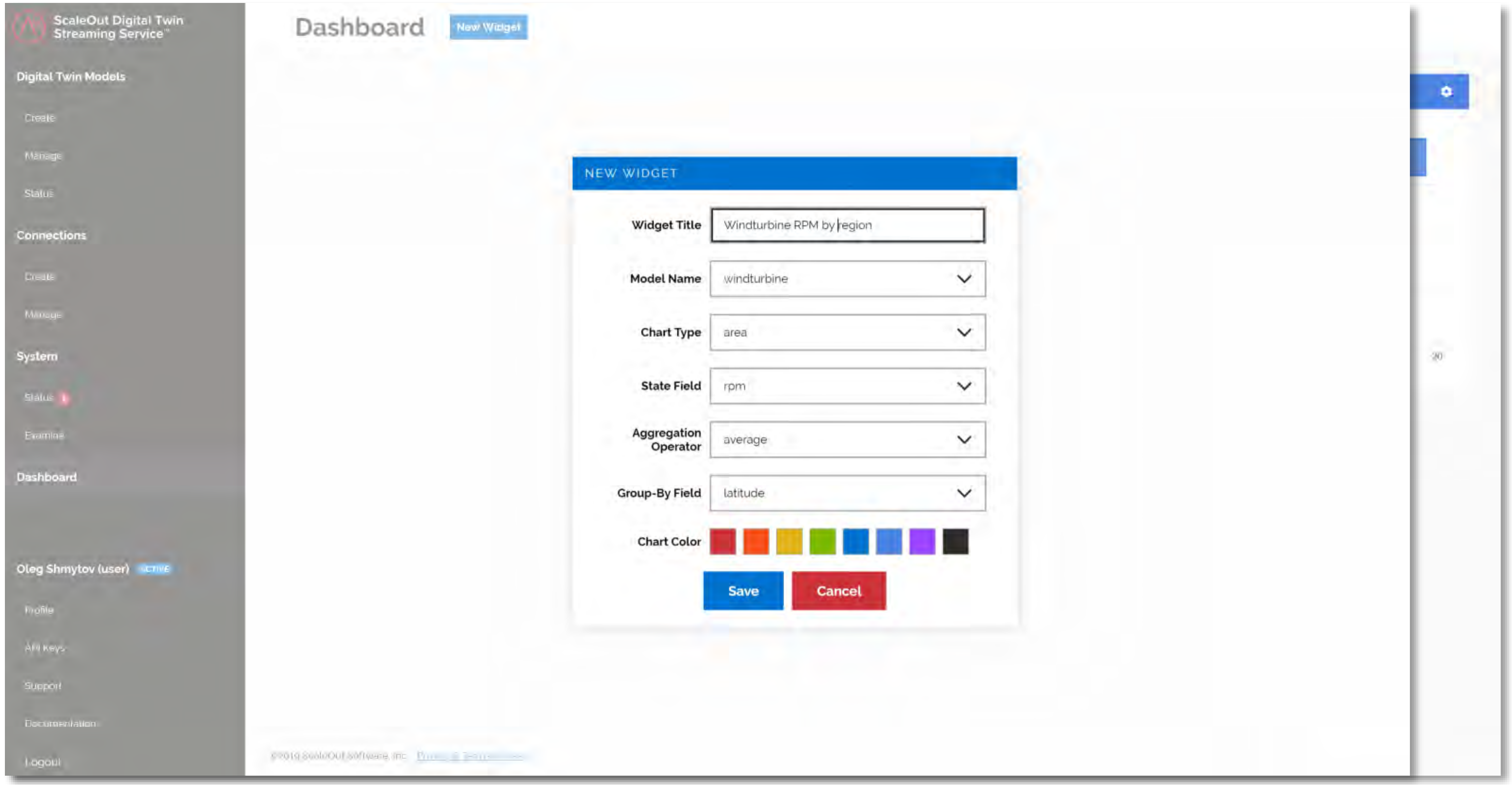
The properties for each digital twin instance (i.e., for each device) can be examined:



Collecting Aggregate Statistics

"Widgets" can be created for digital twin models to display aggregate statistics:

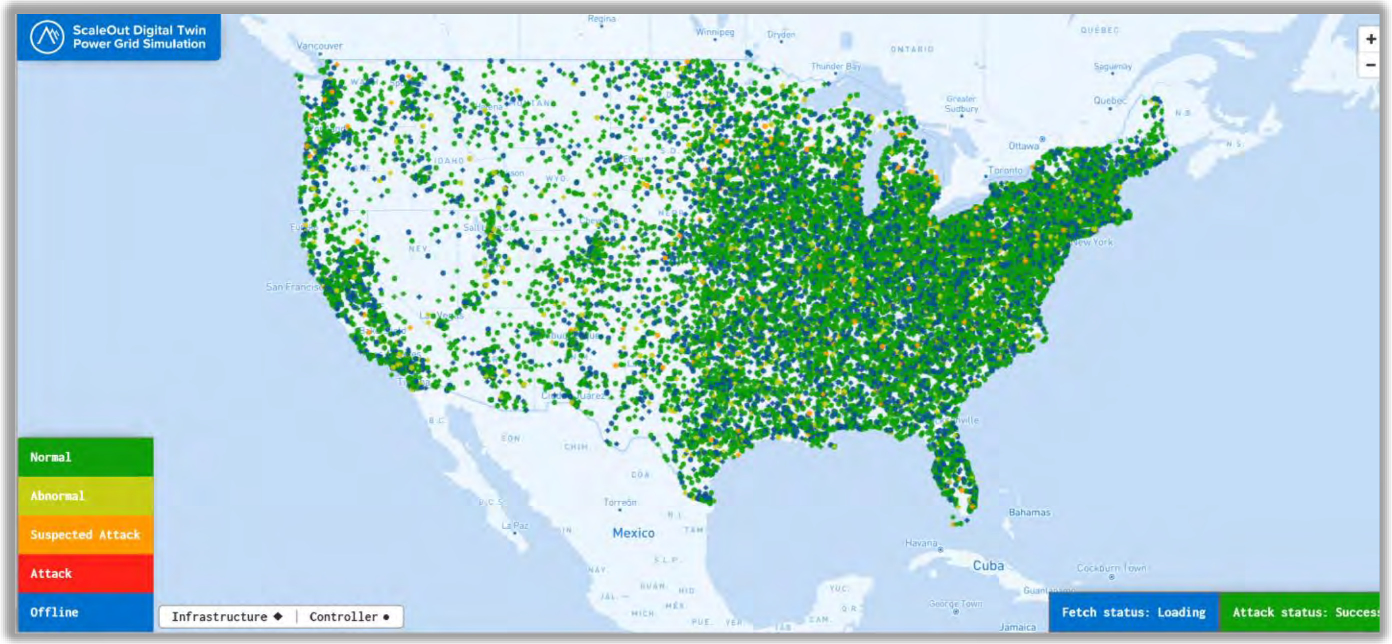
- Performs periodic MapReduce on selected state properties.
- Runs every few seconds.



Demo: Power-Grid Status Tracking

Goal: Maximize situational awareness for grid managers.

- Tracks state of 43K simulated nodes in a power grid with real-time digital twins.
  - Tracks history and characteristics of each node.
  - Generates an assessment of alert level.
- Identifies immediate threats using real-time, aggregate analytics.
  - Displays alert level by region.
  - Refreshes every 5 seconds.



Takeaways

- Real-time stream-processing is challenging.
- Traditional approach (Lambda Architecture) limits real-time processing and cannot perform aggregate analysis in real time.
- Real-time digital twins offer a breakthrough:
  - Deeper introspection in real time
  - Simplified application design
  - Fast, scalable performance
- Enable vastly improved **situational awareness** and **response.**
- In-memory data grid provides a fast, scalable execution platform.