Leveraging MongoDB to create Asset Administration Shells and enable Industry 4.0

White paper

Jan. 2023

MongoDB.

Table of Contents

Abstract

In order to achieve digital manufacturing through Industry 4.0, it is critical to aggregate and manage the huge amount of data generated in the factory by Industrial Internet of Things (IIoT) sensors and devices. Having connected machines, workers and systems is just the start of the digital transformation journey. To extract real value from this connectivity, a digital twin of machines and processes have to be modeled in a standardized manner which will help aggregate heterogeneous data and provide valuable insights to drive productivity and efficiency. The interoperability and flexibility of different assets is an important challenge due to the volume and variety of data that needs to be modeled. This paper will go over Industry 4.0 compliant data modeling principles using the MongoDB Smart Factory testbed as an example. We will model the data using MongoDB Atlas in a way that fulfills requirements and needs of Industry 4.0: interoperability, flexibility and scalability.

Introduction

The fourth industrial revolution is penetrating heavily into the manufacturing industry. Manufacturing companies are investing heavily to create digital transformation strategies and implement new digital technologies. These technologies include but are not limited to the Industrial Internet of Things (IIoT), cloud computing, scalable data infrastructure, and workforce enablement applications. Industry 4.0 (I4.0) enables smart manufacturing, improved supply chain performance, higher organizational efficiency, and business productivity for manufacturing companies.

Through the adoption of Industry 4.0, tangible business benefits can be enabled. Given this attractive prospect, most of the manufacturing companies around the world have plans for Industry 4.0 adoption in the near future. It would be amiss of us to not consider the impact that COVID 19 has had on manufacturing companies' long-term strategies. Companies are emerging from the pandemic with a renewed focus on digital transformation. With people at the center of the digital agenda, long term strategies and roadmaps are being crafted for achieving end-to-end visibility of plant operations, digitally upskilling the remote workforce and creating digital data sharing relationships with ecosystem partners.

A recent McKinsey survey of over 400 manufacturing companies suggests that Industry 4.0 technologies have helped 94 percent of the companies in keeping their operations running during the pandemic. 56 percent of the companies have suggested that these technologies had been critical to their crisis responses. It is also interesting to note that companies that had invested in Industry 4.0 use cases prior to COVID-19 found themselves better positioned to respond to the crisis.

Industry 4.0 generally comprises many complex components and has broad applications in all manufacturing sectors. The digital economy is demanding that manufacturing applications become smarter, drive better customer experiences, surface insights, and take intelligent action directly within the application using live operational data – in real time. Smart Factories implementing Industry 4.0 require a flexible and adaptable manufacturing process to satisfy a market requiring an increasing demand for customization and high

efficiency. One of the criteria for achieving Industry 4.0 is that smart factories must satisfy horizontal and vertical integration requirements through seamless connectivity between assets.

I4.0 Reference Architecture and Asset Administration Shells

In 2013, Platform Industrie 4.0 was created to promote research in Industry 4.0 for the German manufacturing industry. They offered an influential Reference Architectural Model Industrie 4.0 (RAMI 4.0), helping organizations to identify and classify areas of Industry 4.0, building foundations for further technological development. RAMI 4.0 represents the most important Industry 4.0 elements in a three-dimensional layer model. It also ensures that all participants involved in the product life cycle share a common perspective and develop a common understanding of the manufacturing process and how industry 4.0 helps in optimizing the processes.

RAMI 4.0 may look similar to the seven-layer OSI model for network protocols, but since Industry 4.0 involves all manufacturing processes and aspects, RAMI 4.0 representation needs to be more complex. To account for this increased complexity, it is depicted as a cube in Figure 1 that shows the elements and concepts of Industry 4.0 and how they relate to one another.

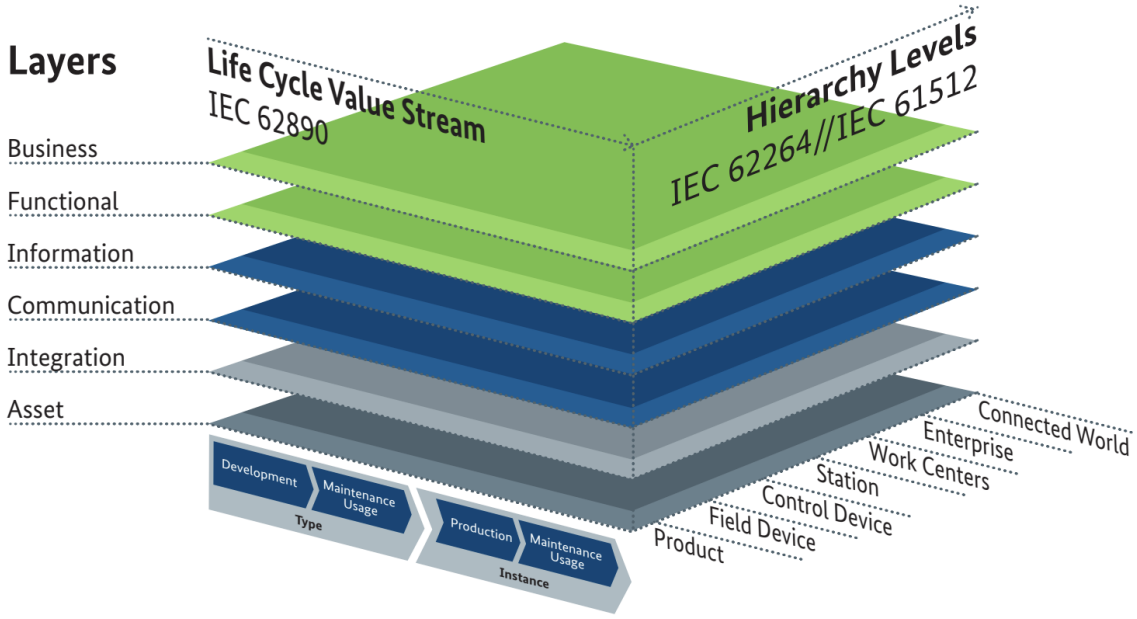

Figure 1. Reference architecture model Industry 4.0 (RAMI 4.0) (Source)

The cube is structured in three dimensions:

The first dimension (left horizontal axis) represents the product life cycle and value stream. The left horizontal axis is from IEC 62890 and represents the product life cycle management. A "type" becomes an "instance" when the development and prototype production is completed and the actual product is manufactured in the production department.

The second dimension (right horizontal axis) represents the hierarchy levels. The hierarchy level is from IEC 62264 (internal series of standards on the integration of company IT and control systems) and represents the different functionalities within the factory or plant.

The third dimension (vertical axis) represents the interoperability layers. The two horizontal axes are brought together with the help of the six layers on the vertical axis of the RAMI cube. The representation in layers comes from information and communication technology.

The second dimension showing the factory hierarchy can be considered Industry 3.0 (or the old world). In such a rigid model, the hardware that is placed on the shopfloor defines the structure, and manufacturing functions are linked to that hardware. The communication is from one level to another and the product is isolated. In the new world or Industry 4.0 world, the idea is to have flexible systems and machines where functions are distributed throughout the network. There is a connected asset network where all assets can interact across hierarchical levels. This transformation is depicted in Figure 2. It is also important that the product is treated as an asset and is part of the network. At the end of the day, manufacturing companies who are looking to achieve Industry 4.0 wish to transform their operations from a rigid hierarchical model to a connected "asset" model where there is seamless communication between all "assets" in a factory, including product, machine, and operators.

For many manufacturing companies, the implementation of RAMI 4.0 and the connected asset model is challenging, since 1) the RAMI 4.0 cube is too abstract and 2) manufacturing processes are complex and generate variable data from different sources, diverse machines and through various communication capabilities. Therefore it is challenging to create a platform where all assets can talk to each other.

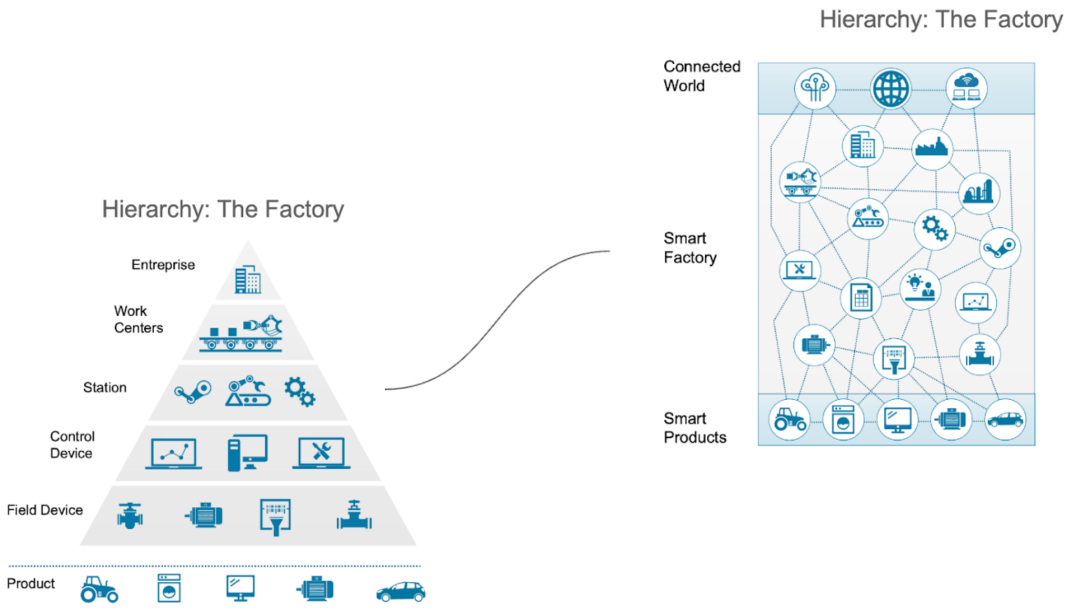

Figure 2: Transformation of a traditional hierarchical model into fully connected model (Source)

Let's define an “asset” in a more structured manner. An asset is anything with a unique ID and a mechanism for sending and receiving data. By this definition, machines, machine components, products, technical drawings, operations, supervisors, and even job orders are all assets. As an example, the operator needs to consume information about jobs from the Manufacturing Execution System (MES) and has to push data into the machine through the use of the Human Machine Interface (HMI). The HMI and MES are the mechanisms for the operator to send and receive data.

All these various assets must be identifiable and all assets must be able to read and understand any other asset’s type, operational and technical data, status, and other asset-specific information. To accomplish this, RAMI 4.0 introduces the asset administration shell (also referred to simply as the “administration shell” or “AAS″). At the core of Industry 4.0 implementation, every asset should be modeled via an AAS. It is a knowledge structure that provides a description of the technical functionality of the physical object and its interactions. The aforesaid can be rationalized as the data representation of the physical world object as a digital twin. AAS as presented in its specification document by Platform Industrie 4.0, is envisioned to contain a number of views that represent different aspects, where each specific view is exposed depending on who is interacting with it. According to the specification, “the asset administration shell is thus made up of a series of submodels. These represent different aspects of the asset concerned; for example, they may contain a description relating to safety or security, but could also outline various process capabilities such as drilling or installation”.

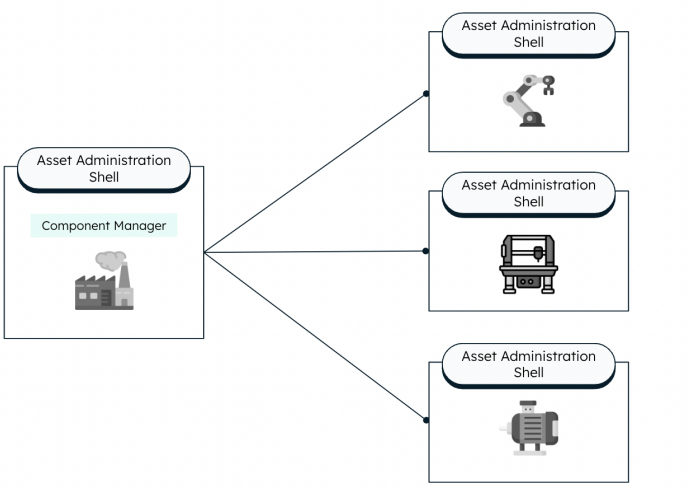

Figure 3: The Asset Administration Shell Concept

In other words, the AAS contains all the information relevant to a specific asset, its lifecycle, technical functionalities, and functions. AAS is an abstract structure that can be described by JSON, UML, XML, AutomationML, and OPC UA information models. All the assets in the factory can be modeled as individual AAS in the central data hub standardizing the information model and the interface, coping with all the heterogeneous systems available in the industrial environment (industrial silos).

Structure of Asset Administration Shell

AAS has two parts: the header and body. The header contains information for the identification, administration, and usage of the asset. It also contains important information about the asset subcomponents. The body part stores submodels which contain information about properties of the asset. These properties contain features that refer to the functions and data related to the asset. Apart from properties, the body of AAS has a manifest that lists all the submodels in the AAS. Finally, the body also contains a component manager which is responsible for linking the information in AAS to the IoT or connected world.

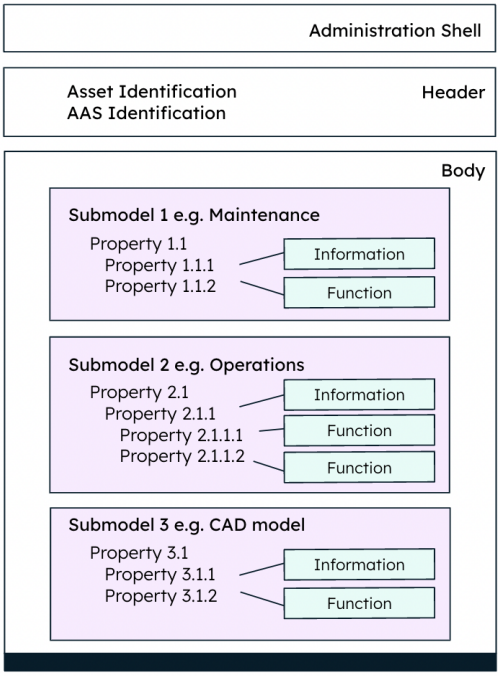

Figure 4: The structure of AAS as described in the specification document (the component manager is omitted here for sake of simplicity)

Let us look at an example of a Computer Numerical Control (CNC) machine and build an AAS model of that machine. In our example, the CNC machine is equipped with sensors and through its controller, it can communicate with external systems for information exchange. Figure 5 shows the AAS structure for a CNC machine.

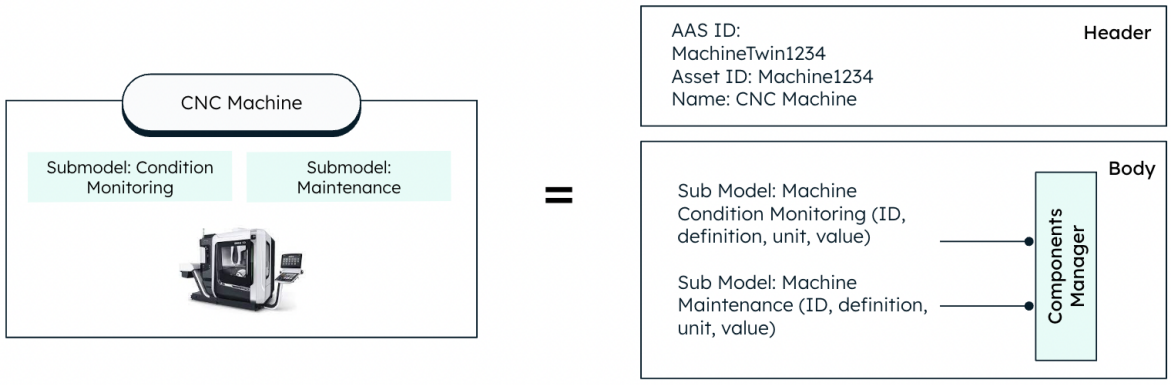

Figure 5: AAS Structure of a CNC machine

AAS Header contains the asset and AAS ID. These unique IDs will help the machine manufacturer, installer, service technician, or anybody seeking technical details related to the machine to perform their task. Since a factory will have multiple CNC machines all connected to an IIoT network, a unique identifier will help ensure the right machine is accessed via the component manager. Here, we can also call the AAS ID as the Asset Digital Twin ID since we are collecting all the information about the asset that is required for mimicking its behavior.

AAS Body contains two submodels, machine condition monitoring, and machine maintenance.

Machine condition monitoring: This submodel will contain information regarding machine status (on/off/running etc.), machine subsystems temperature, vibration, current consumption, number of running hours, etc. This data is generated by means of in-built sensors or external sensors installed on the machine.

Machine maintenance: This submodel will contain all the maintenance-relevant information as provided by the original equipment manufacturer as well as the machine maintenance history and logs. This submodel can be used to facilitate the maintenance service by eliminating the need of going through lengthy service and maintenance manuals. The information stored in this submodel will be related to things such as machine subsystem dismantling, repair, general testing, etc.

Let's try to write this AAS structure in JSON format. JSON, or JavaScript Object Notation, is a human-readable data interchange format, specified in the early 2000s. Even though JSON is based on a subset of the JavaScript programming language standard, it's completely language-independent. In JSON, the AAS will look like as shown in the code snippet below. It has to be noted that we can have many additional submodels for our machine AAS, but for the sake of simplicity we are limiting it to two in this paper.

AAS Header

```
{
  "Header": {
```

```
    "HeaderID": "100",
    "HeaderName": "CNC Machine Header",
    "AssetIdentification": {
      "Name": "CNC Machine",
      "AASID": {
        "IDType": "URI",
        "IDSpec": "https://company.org/AAS/MachineTwin1234"
      },
      "AssetID": {
        "IDType": "URI",
        "IDSpec": "https://company.org/Assets/M1234"
      }
    }
  },
```

AAS Body

```
"Body": {
    "BodyID": "101",
    "BodyName": "CNC Machine Body",
    "SubModels": [
      {
        "Name": "ConditionMonitoring",
        "Version": 1,
        "Revision": 1,
        "ModelID": {
          "IDType": "URI",
          "IDSpec": "https://company.org/models/M1234/ConditionMonitoring"
        },
        "SubmodelElementsCollection": [
          {
            "Name": "DataAcquisition",
            "SubmodelElementProperties": [
              {
                "Name": "SpindleVibration",
                "ExpressionLogic": "EQUAL",
                "ExpressionSemantic": "MEASUREMENT",
                "PropertyID": {
                  "IDType": "URI",
                  "IDSpec": "https://company.org/models/M1234/ConditionMonitoring/Vibration"
                },
                "View": "OPERATIONS",
                "Value": 2000,
                "Unit": "kHz"
              },
              {
                "Name": "MachineTemperature",
                "ExpressionLogic": "EQUAL",
                "ExpressionSemantic": "MEASUREMENT",
                "PropertyID": {
                  "IDType": "URI",
                  "IDSpec": "https://company.org/models/M1234/ConditionMonitoring/Temperature"
                },
                "View": "OPERATIONS",
                "Value": 42,
                "Unit": "C"
              },
              {
                "Name": "MachineRunning",
                "ExpressionLogic": "EQUAL",
                "ExpressionSemantic": "MEASUREMENT",
                "PropertyID": {
```

```
                  "IDType": 0,
                  "IDSpec":
"https://company.org/models/M1234/ConditionMonitoring/MachineRunning"
                },
                "View": "OPERATIONS",
                "Value": true
              }
            ]
          }
        ]
      },
      {
        "Name": "MachineMaintenance",
        "Version": 1,
        "Revision": 1,
        "ModelID": {
          "IDType": "URI",
          "IDSpec": "https://company.org/models/M1234/MachineMaintenance"
        },
        "SubmodelElementsCollection": [
          {
            "Name": "MaintenanceStatus",
            "SubmodelElementProperties": [
              {
                "Name": "LastMaintenanceDate",
                "ExpressionLogic": "EQUAL",
                "ExpressionSemantic": "MEASUREMENT",
                "PropertyID": {
                  "IDType": "URI",
                  "IDSpec":
"https://company.org/models/M1234/MachineMaintenance/LastMaintenance"
                },
                "View": "MAINTENANCE",
                "Value": "2022-10-14",
                "Unit": "Date"
              },
              {
                "Name": "NextMaintenanceDate",
                "ExpressionLogic": "EQUAL",
                "ExpressionSemantic": "MEASUREMENT",
                "PropertyID": {
                  "IDType": "URI",
                  "IDSpec":
"https://company.org/models/M1234/MachineMaintenance/NextMaintenance"
                },
                "View": "MAINTENANCE",
                "Value": "2022-11-14",
                "Unit": "Date"
              }
            ]
          },
          {
            "Name": "MaintenanceDocumentation",
            "SubmodelElementProperties": [
              {
                "Name": "TechnicalDocumentation",
                "ExpressionLogic": "EQUAL",
                "ExpressionSemantic": "MEASUREMENT",
                "PropertyID": {
                  "IDType": "URI",
                  "IDSpec":
"https://company.org/models/M1234/MachineMaintenance/Documentation"
                },
                "View": "MAINTENANCE",
                "Value": "/Machine1234Maintenance.pdf",
                "Unit": "File"
              }
            ]
          }
        ]
```

```
        }
      ]
    }
}
```

Let us look at the standard fields used in above example and their explanation:

Field	**Explanation**
ID	Identifiers can be defined as URIs
Name	A name consisting of one or more words that is assigned to a data type
Version number	Number to distinguish the version of a submodel or property type
Revision number	Number to distinguish the version of a submodel or property type
Value	Current value that can be specified through an instanced submodel element collection query
Unit of measure	Specifies the unit in which the value of a qualified data element type must be given
Expression logic	Specifies which function should be used if different expression logics are to be compared with one another, for example Equal, Greater Than, Less Than, Between the Value limits etc.
Expression semantic	Specifies which role the property plays in an interaction, i.e. which expression the provider of the property intends. Some example are: ● Requirement (for requests that are to be confirmed or rejected) ● Confirmations (for responses to requests that describe the capability of an asset) ● Measurement (if a measured or actual value is provided)
View	Indicates which business function/view(s) that the property is to be associated with, for example Maintenance, Operations

Do note that there can be more properties added to the template as well as more submodels. For some implementations of AAS, a sampling rate for properties is also required. The key takeaway is that the JSON structure for AAS implementation helps with flexibility and scalability as the number of submodels grows.

MongoDB was designed from its inception to be a database focused on delivering great development experiences. JSON ubiquity made it the obvious choice for representing data structures in MongoDB's document data model. However, JSON only supports a limited number of basic types and does not have support for date and binary data which is important for complex use cases. In order to increase the database performance while keeping it general purpose, Binary JSON or BSON was invented to bridge the gap. BSON's binary structure encodes type and length information, which allows it to be traversed much more quickly compared to JSON. It also adds some non-JSON-native data types such as dates and binary data. MongoDB stores data in BSON format both internally, and over the network. However, anything that can be represented in JSON can be natively stored in MongoDB and retrieved easily as JSON. The MongoDB drivers take care of converting the data to BSON and back when querying the database. At the time of this writing, MongoDB provides over 10 official libraries and drivers that can be used to start developing powerful applications.

MongoDB Developer Data Platform

MongoDB Atlas

MongoDB Atlas is a multi-cloud developer data platform. At its core is our fully managed cloud database for modern applications. MongoDB Atlas is the best way to run MongoDB and the MongoDB's document model is the fastest way to innovate because documents map directly to objects in your code. As a result, they are much easier and more natural to work with. You can store data of any structure and modify your schema at any time as you add new features to your applications.

MongoDB's unified query API is the most natural way to work with data in any form. MongoDB Atlas extends MongoDB's flexibility and ease of use to build full-text search, real-time analytics, and event-driven experiences.

The MongoDB Atlas database is available in 80+ regions across AWS, Google Cloud, and Azure. This flexibility also allows you to take advantage of multi-cloud and multi-region deployments, allowing you to target the providers and regions that best serve your users. This best-in-class automation and proven practices guarantee availability, scalability, and compliance with the most demanding data security and privacy standards.

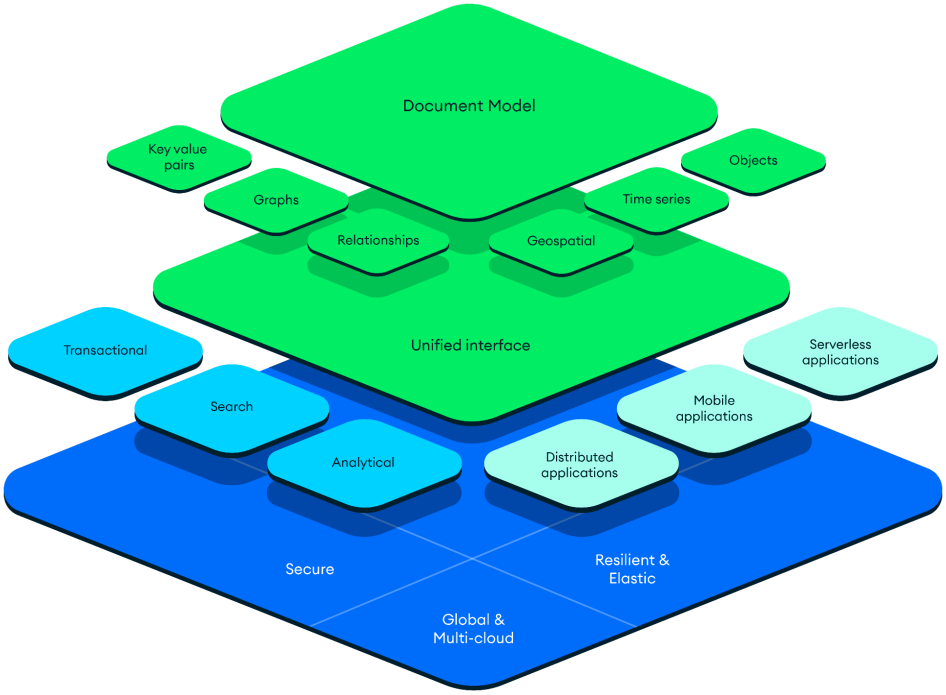

Figure 10: MongoDB Atlas Developer Data Platform

Database Requirements for Industrial IoT

Delivering Industrial IoT (IIoT) and Industry 4.0 at scale requires the ability to extract, interpret, and harmonize data from disparate systems that were not designed to work together. It involves many devices publishing a variety of data and often involves multiple database technologies. Data siloed in different technologies needs to be kept in sync with each other.

The data generated by IIoT devices tends to be large in both volume and frequency, placing a unique strain on the underlying data infrastructure. It is also likely to be time series based as seen in this paper's use case.

While many databases can be used to store IIoT data, some are just better suited to IoT data than others. Due to the polymorphic nature of IIoT sensor data, the database you choose needs to support flexible schemas, make it easy for developers to work with the data, and ensure that your IIoT applications are resilient to future changes.

An IIoT database needs to handle the following challenges:

Data variety: Data produced by IIoT sensors can take many different forms. As the IIoT ecosystem grows, you need a database that can easily evolve its data schemas without overhead or downtime and at scale.

Scalability: IIoT devices produce massive amounts of data. To avoid outages or performance issues, you need a database that can easily and automatically scale vertically as well as horizontally.

Time series data: In order to reduce disk space and optimize data queries, time series data support is essential. The time series data has to be optimized for storage and indexed for fast analytical queries. Compression ratios over 90% are usually desirable for any scalable IIoT platform.

MongoDB Smart Factory Testbed

The Industry Solutions team at MongoDB demonstrates how easily MongoDB can be integrated to solve the digital transformation challenges of IIoT in manufacturing with its flexible, scalable, developer data platform.

Using the small-scale model of a smart fabrication factory from Fischertechnik, the team collects and sends data via MQTT and processes it in MongoDB Atlas. Similar to a full-scale physical factory, the smart factory model demonstrates how easily IIoT use cases can be built on top of MongoDB's developer data platform to enable and accelerate the digitalization of manufacturing processes in any industry.

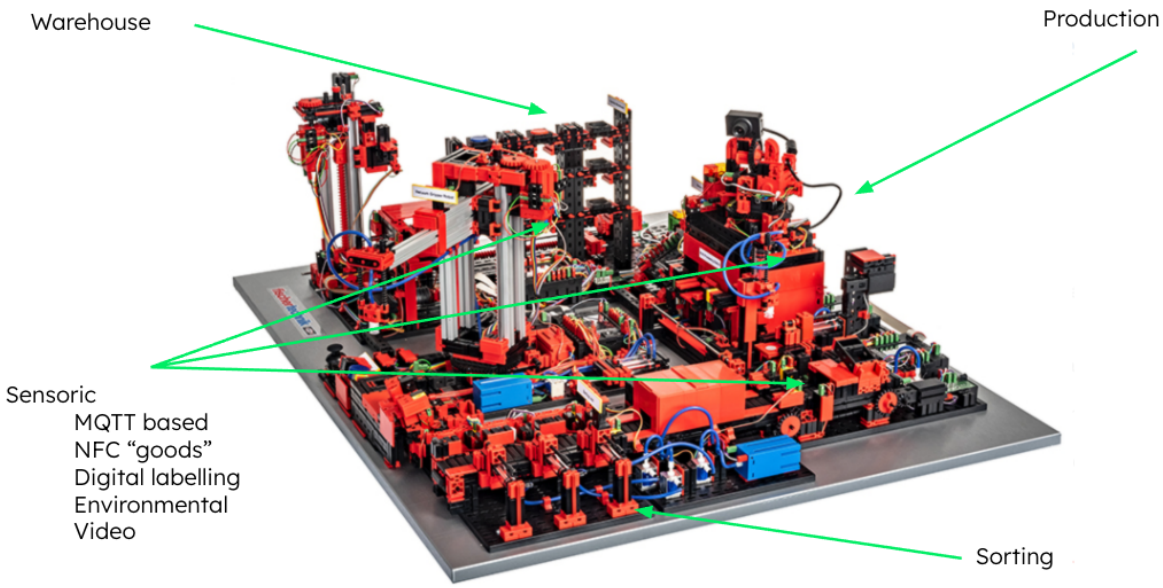

Figure 6: MongoDB Smart Factory Testbed

The testbed is made up of several components: a warehouse, multiprocessing station, and sorting area. The warehouse is where raw material is stacked and stored. When a customer order is received, the raw material is retrieved and moved to processing by a robot with vacuum suction gripper. From there, the items are sorted by color (i.e. red, white, or blue), to be sent to the correct outbound destination. The process covers ordering and storing raw materials to ordering and manufacturing of end products. Throughout these processes, there are multiple sensors detecting the type and color of the items, as well as environmental aspects like temperature and how much inventory is in stock. A surveillance camera detects motion and sends alerts including photos via MQTT. This simulates the wide variety of data a smart factory would emit in real-time for track and trace, monitoring and visualization, alerts and as input for machine learning algorithms. Figure 7 shows the process for the Smart Factory. This paper will focus on creating the asset administration shell of the robot as that is the critical equipment in this whole process.

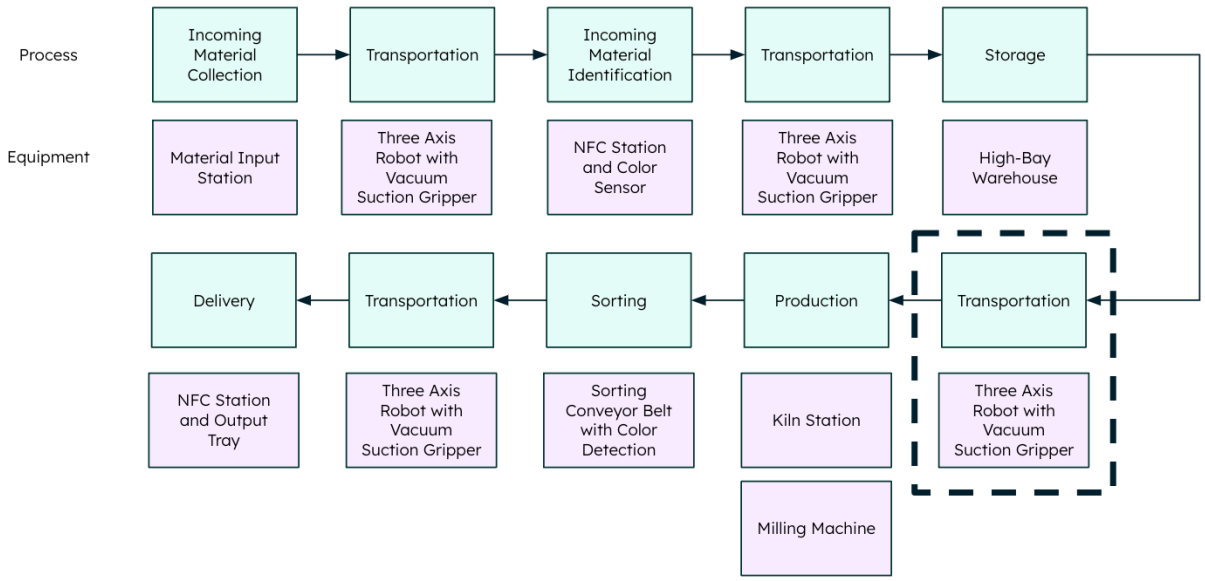

Figure 7: MongoDB Smart Factory process flow and equipment used at each station. This paper will focus on the robot and the robotic process of moving items from warehouse to kiln (highlighted using black border)

The Factory's Infrastructure

The machines of the factory are controlled by TXT controllers, Linux-based computers which use MQTT to communicate between each other, and also with cloud-based applications. There are basically two types of data sent and received via MQTT-commands to trigger an action and streaming of event and time series sensor data. The main TXT controller runs a MQTT broker and replicates selected topics to a MQTT broker in the cloud. From there a Kafka container collects the data streams and inserts them into MongoDB. The data persisted in MongoDB is then visualized via MongoDB Atlas Charts for real-time insights.

To emphasize and explain how MongoDB can be leveraged, we developed additional apps, using JavaScript, ReactJS, and Realm, to integrate and streamline data flows and processes on top of the MongoDB data platform. This includes:

- MongoDB Realm Order Portal: A ReactJS web application to order new products and track the process of orders.

- Data Visualization: A visualization of the different data types collected in MongoDB and visualized via MongoDB Atlas Charts for insights.
- Alert Management App: A mobile app leveraging MongoDB Realm and Realm Sync for alert notification and management offline and online.

A detailed architecture diagram is shown in Figure 8. For more details on the factory and how it sends and receives data, please read our [Manufacturing at Scale: MongoDB and IIoT blog series.]

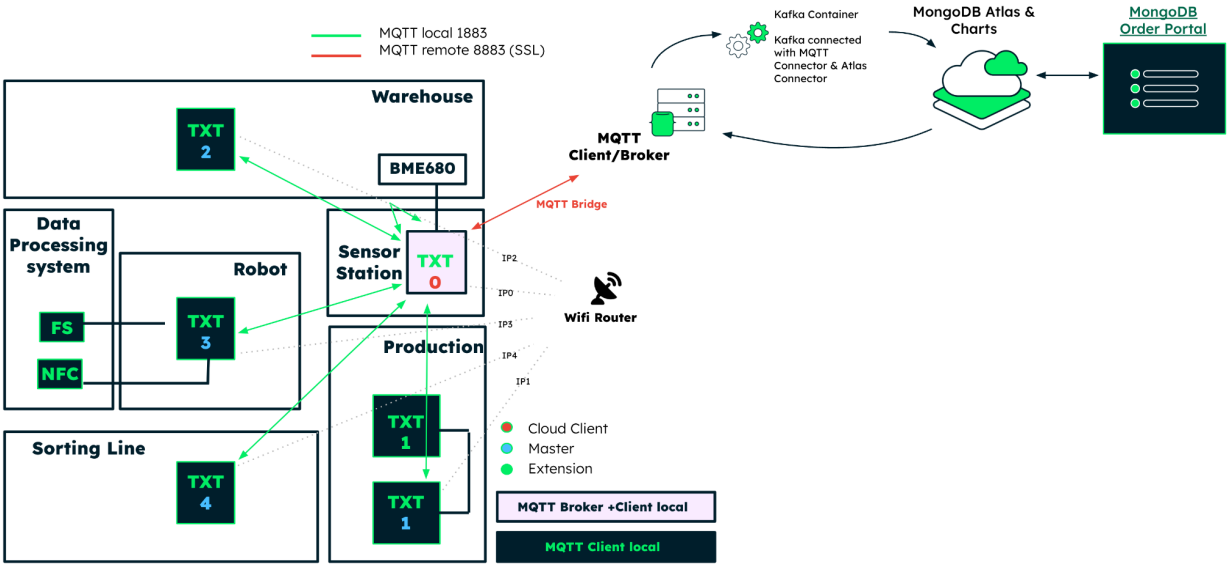

Figure 8: Factory layout connected to data infrastructure

When the product is received at a station, the TXT controller at that station publishes acknowledgement messages through MQTT protocol. These messages are received and stored in MongoDB Atlas in a time series collection. In case of our selected process (robot moving product from warehouse to kiln), the messages flow will look something like this:

1. Robot sending out acknowledgement that it has picked up a product from the warehouse. The code field value 3 indicates that the piece has been picked up successfully.

```
{
 "ts":"YYYY-MM-DDThh:mm:ss.fffZ",
 "code":3,
 "product":{
              "id":"123456789ABCDE",
```

```
                "type":"<BLUE/WHITE/RED>",
                "state":"<RAW/PROCESSED>" }
}
```

2. Robot sending out acknowledgement that it has transported a product to kiln in the production station. Code 7 is indication of successful transfer to the kiln

```
{
 "ts":"YYYY-MM-DDThh:mm:ss.fffZ",
 "code":7,
 "product":{
                "id":"123456789ABCDE",
                "type":"<BLUE/WHITE/RED>",
                "state":"<RAW/PROCESSED>" }
}
```

3. The production station sending out acknowledgement with code = 1 that production is started

```
{
 "ts":"YYYY-MM-DDThh:mm:ss.fffZ",
 "code":1
}
```

Creating Asset Administration Shell for Smart Factory Robot using MongoDB

We can create our robot AAS in MongoDB for a specific use case through a series of five simple steps.

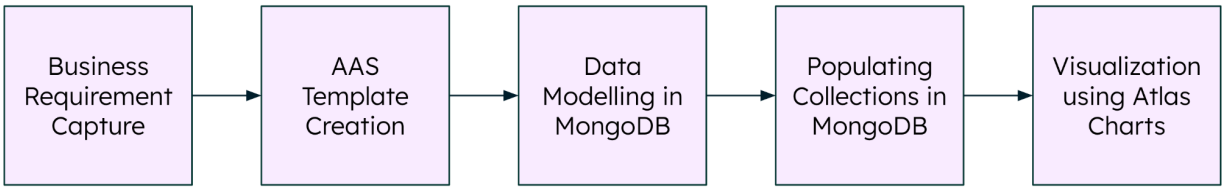

Figure 9: Five step process to create robot AAS

Step 1 - Business Requirements Capture

The whole idea behind creating Asset Administration Shells is to develop a scalable and flexible model that can meet the Industry 4.0 requirements. One of the most important Key Performance Indicators (KPI) that manufacturing companies always strive to improve is Overall Equipment Effectiveness (OEE). OEE measures the percentage of planned production time that is truly productive. As a baseline, OEE can be used to track progress of the factory over time in eliminating top losses. For many companies, the goal is to achieve an OEE greater than 85%. OEE calculation takes into account all the losses (availability, performance and quality loss) resulting in a truly productive manufacturing time.

OEE = Availability x Performance x Quality

Availability is calculated as the ratio between actual run time of the machine and total planned production time (or shift time)

Availability = Total Run Time / Planned Production Time

Performance takes into account all the losses in time that makes an equipment perform at less speed than maximum possible. It can be calculated as

Performance = (Ideal Cycle Time x Total Count) / (Planned Production Time - Total Stop Time)

Quality is simple to calculate. It is simply the ratio between good parts (no defects - passed inspection) and total parts products by the machine whose OEE we are trying to calculate

Quality = Good Count / Total Count

These formulae give us an idea about how to design the Asset Administration Shell of the robot and what submodels to include in the model. The AAS should be designed in such a way that it enables easy access to availability, performance and quality metrics of the physical equipment. In conclusion, the use case or the business requirement is to perform **automated robot OEE monitoring for the transportation process between warehouse and kiln**.

Step 2 - AAS Template Creation

Let us look at the type of data that is needed from the robot controller to achieve the use case requirements. The following data is needed

Data	**Description**	**Sensor Data Needed**
Total product count	How many products/items has the robot tried to transport from origin location to target location over the course of planned production time	Message from Robot with acknowledgment code = 3
Good product count	How many products have been transferred successfully during transportation over the course of planned production time (The assumption is that if a product is dropped during the course of transportation, it is damaged and needs to be rejected quality wise)	Message from Robot with acknowledgment code = 7
Ideal cycle time	Theoretical fastest time for robot to complete a process	Nil
Planned production time	Time for one shift (We will ignore scheduled time loss / planned downtime for this example)	Nil
Total run time	Planned production time minus any availability loss	Sum of difference in timestamp between origin and target station acknowledgment messages
Total stop time	Planned production time - Total run time	Nil

With the sensors and data identified, let us list down the submodels for the AAS for our use case

Submodel 1 : Product Data

```
{
  "Name": "Product",
  "Version": 1,
  "Revision": 1,
  "ModelID": {
    "IDType": "URI",
    "IDSpec":
"https://smartfactory.org/models/robot/prod
uct"
  },
  "SubmodelElementsCollection": [
    {
      "Name": "Product Count",
      "SubmodelElementProperties": [
        {
          "Name": "Total Product Count",
          "ExpressionLogic": "EQUAL",
          "ExpressionSemantic":
"MEASUREMENT",
          "PropertyID": {
            "IDType": "URI",
            "IDSpec":
"https://smartfactory.org/models/robot/prod
uct/totalproductcount"
          },
          "View": "OPERATIONS",
          "Value": 100,
          "Unit": ""
        },
        {
          "Name": "Good Product Count",
          "ExpressionLogic": "EQUAL",
          "ExpressionSemantic":
"MEASUREMENT",
          "PropertyID": {
            "IDType": "URI",
            "IDSpec":
"https://smartfactory.org/models/robot/prod
uct/goodproductcount"
          },
          "View": "OPERATIONS",
          "Value": 90,
          "Unit": ""
        }
      ]
    }
  ]
}
```

Submodel 2 : Production Data

```
{
  "Name": "Production",
  "Version": 1,
  "Revision": 1,
  "ModelID": {
    "IDType": "URI",
    "IDSpec":
"https://smartfactory.org/models/robot/prod
uction"
  },
  "SubmodelElementsCollection": [
    {
      "Name": "Cycle Time",
      "SubmodelElementProperties": [
        {
          "Name": "Ideal Cycle Time",
          "ExpressionLogic": "EQUAL",
          "ExpressionSemantic":
"MEASUREMENT",
          "PropertyID": {
            "IDType": "URI",
            "IDSpec":
"https://smartfactory.org/models/robot/prod
uction/idealcycletime"
          },
          "View": "OPERATIONS",
          "Value": 10,
          "Unit": "sec"
        },
        {
          "Name": "Planned Production
Time",
          "ExpressionLogic": "EQUAL",
          "ExpressionSemantic":
"MEASUREMENT",
          "PropertyID": {
            "IDType": "URI",
            "IDSpec":
"https://smartfactory.org/models/robot/prod
uction/plannedproductiontime"
          },
          "View": "OPERATIONS",
          "Value": 480,
          "Unit": "min"
        },
        {
         "Name": "Total Run Time",
         "ExpressionLogic": "EQUAL",
         "ExpressionSemantic":
"MEASUREMENT",
         "PropertyID": {
         "IDType": "URI",
         "IDSpec":
"https://smartfactory.org/models/robot/prod
uction/totalruntime"
         },
         "View": "OPERATIONS",
```

```
          "Value": 450,
          "Unit": "min"
        },
        {
          "Name": "Total Stop Time",
          "ExpressionLogic": "EQUAL",
          "ExpressionSemantic":
"MEASUREMENT",
          "PropertyID": {
                 "IDType": "URI",
                 "IDSpec":
                 "https://smartfactory.org/models/rob
                 ot/production/totalstoptime"
          },
          "View": "OPERATIONS",
          "Value": 30,
          "Unit": "min"
        }
      ]
    }
  ]
}
```

The full AAS template looks as follows

```
{
  "Header": {
    "HeaderID": "100",
    "HeaderName": "Robot Header",
    "AssetIdentification": {
      "Name": "Robot",
      "AASID": {
        "IDType": "URI",
        "IDSpec": "http://smartfactory.org/robotaas"
      },
      "AssetID": {
        "IDType": "URI",
        "IDSpec": "http://smartfactory.org/robot"
      }
    }
  },
  "Body": {
    "BodyID": "101",
    "BodyName": "Robot Body",
    "SubModels": [
      {
        "Name": "Product"
        ...
      },
      {
        "Name": "Production"
        ...
      }
    ]
  }
}
```

Step 3 - Data Modelling in MongoDB

The first step in the time series data life cycle is ingesting data from sensors, devices, applications, etc. Manufacturing companies are embarking on digital twin initiatives to create a digital footprint of a physical entity. High-volume data ingestion is a key requirement here. However, many companies are still busy doing the plumbing, so to speak, to ingest that data and it is very time and resource-consuming. So capabilities to

speed up data ingestions and data modeling are key. Time series data also needs to be compressed and stored efficiently. This is where MongoDB time series collections come in.

MongoDB 5.0 introduced time series collections, which automatically store time series data in a highly optimized and compressed format, reducing customer storage footprint, as well as achieving greater query performance at scale. These optimizations eliminate lengthy development cycles, enabling developers to quickly build a schema tuned for the unique performance and analytical demands of time series applications that they're building. When using time series collections, MongoDB automatically creates a clustered index against data ordered by time so you can query data with little latency. We have also expanded the MongoDB query API with a suite of specialized time series queries and analytics. These queries and analytics are specifically window functions and temporal operators to help you uncover hidden patterns quickly. MongoDB also natively supports the entire time series data, lifecycle from ingestion storage, querying real-time analysis and visualization through to online archiving or automated expiration as data ages, all in a single secure, resilient, and scalable enterprise-ready data platform. Finally, MongoDB time series collections sit right next to regular collections in the same database so you can easily blend time series data with enterprise data supporting any workload.

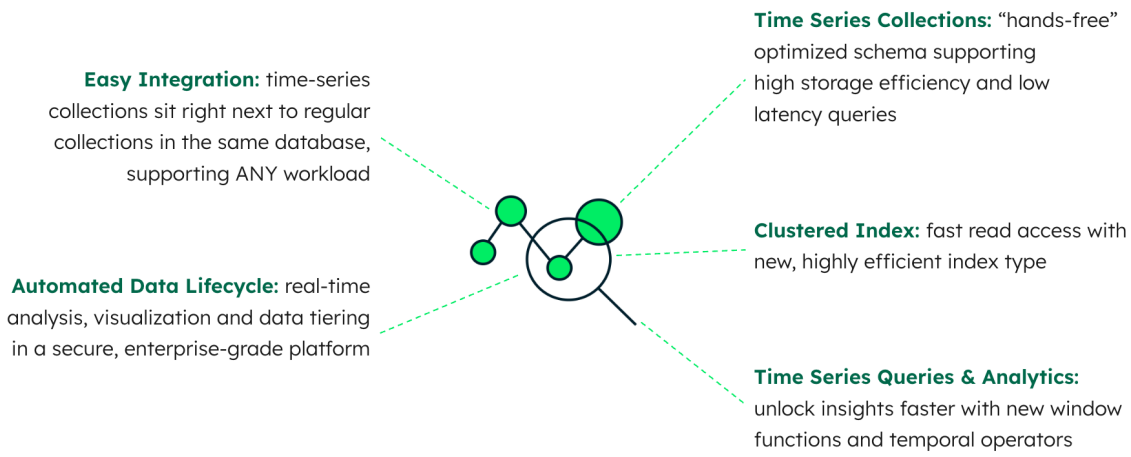

Figure 11: MongoDB native Time Series collections for IIoT data

Time series data is generally composed of these components:

Time when the data point was recorded.

Metadata (sometimes referred to as source), is a label or tag that uniquely identifies a series and rarely changes. In our example, we have one field in metadata which is the robot Id.

Measurements (sometimes referred to as metrics or values), are the data points tracked at increments in time. Generally, these are key-value pairs that change over time. In our case, that would be the code value.

Let us see how the sensor data will look in MongoDB Atlas with time series collections and how the AAS data will look as a normal collection in MongoDB. The AAS data is stored as a JSON structure which not only makes it flexible to add more submodels without worrying about schema changes. It also serves as a template to be applied to other equipment in the future that we would like to calculate OEE of.

Figure 12: Robot Messages and AAS in MongoDB

We can store the product ID and color in the time series collection metadata as well but for simplicity's sake, we can ignore it. The factory runs a strict serial production process.

We need one more collection to store the OEE, availability, performance and quality values. This can be set up as a simple normal collection in Figure 13.

OEE

```
_id: ObjectId('6371cfab70dec388bf31fd1d')
oee: 77
availability: 90
performance: 95
quality: 90
date: 2022-11-12T00:00:00.000+00:00
```

Figure 13: OEE Collection in MongoDB

As more machines get connected to MongoDB Atlas, having a good indexing strategy is crucial to ensuring that your MongoDB database returns your results in the most efficient way. Indexes organize the data in a specific order so the engine knows where to look for it. In our example, we can create an index based on AAS ID as almost all of our writes and reads are to be based on a specific AAS ID.

Consider the scenario where an organization wants to create digital twins of all the equipment in all their factories globally. The vision is to create AAS collections for all their products and machines which can range into hundreds of thousands and there are concerns regarding latency issues with a single location data server. Database systems with large data sets or high throughput applications can challenge the capacity of a single server. There are two ways to scale up a database cluster, vertically and horizontally. Vertical scaling or scaling up means increasing the capacity of a single server using a more powerful CPU, adding more RAM or increasing the amount of storage space. There is a practical maximum for vertical scaling and cloud providers generally have hard ceilings on available hardware configurations.

MongoDB supports horizontal scaling through sharding. Sharding is a method for distributing data across multiple machines. Using this method, MongoDB supports deployments with very large data sets and high throughput operations. For more information on sharding, please refer to the MongoDB documentation on sharding.

MongoDB uses the shard key to distribute the collection's documents across shards. The choice of shard key affects the creation and distribution of chunks across the available shards. We can add a new body property called location to each AAS document and then utilize a zoned sharding approach which will segment data by location and distribute it to the appropriate shard. Zones can help improve the locality of data for sharded clusters that span multiple data centers.

Step 4 - Populating AAS and OEE Collection

After modeling and storing smart factory sensor data in MongoDB Time Series collections, we need to transform the raw acknowledgment messages and populate the robot AAS collection. The updated values in the collection will be used to calculate OEE and update OEE collection documents. We can calculate all the desired OEE values using the time stamps in the acknowledgment messages from the robot.

MongoDB Atlas provides many options to achieve our goal. Let us look at those options one by one.

Using MongoDB Atlas Triggers

MongoDB Atlas Triggers allow you to execute server-side logic in response to database events or according to a schedule. MongoDB Atlas provides two types of triggers: Database and Scheduled triggers. Triggers listen for events of a configured type. Each trigger links to a specific MongoDB Atlas function. When a trigger observes an event that matches your configuration, it "fires." The Trigger passes this event object as the argument to its linked Function.

In our case, we can set up a trigger based on the production shift timing schedule (Figure 14) so that it fires once every day at the end of the shift. When the trigger is fired, it will run our custom function and not only calculate the product counts and total run time but also update the AAS and OEE collection automatically. The formulae for performance, quality, and availability can be set up inside the custom function. Therefore reducing the dependency on creating a HTTP client outside of MongoDB Atlas.

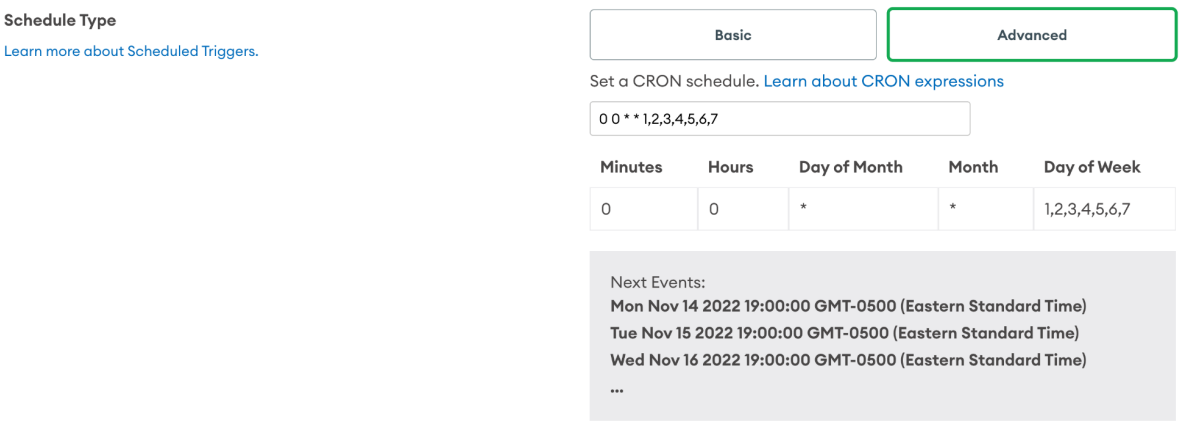

Figure 14: MongoDB Atlas provides the flexibility of setting up scheduled triggers

Inside the custom trigger function, the aggregation pipeline can be used to transform and update the values in the AAS collection. Figure 15 shows an example of how a custom function can be written that runs an aggregation pipeline over the robot message documents and updates the total product count field in the robot AAS document. The custom function can be expanded to include OEE calculations and updating the OEE collection in the same manner. To understand more about the aggregation framework, please visit the MongoDB [Aggregation Reference web page]

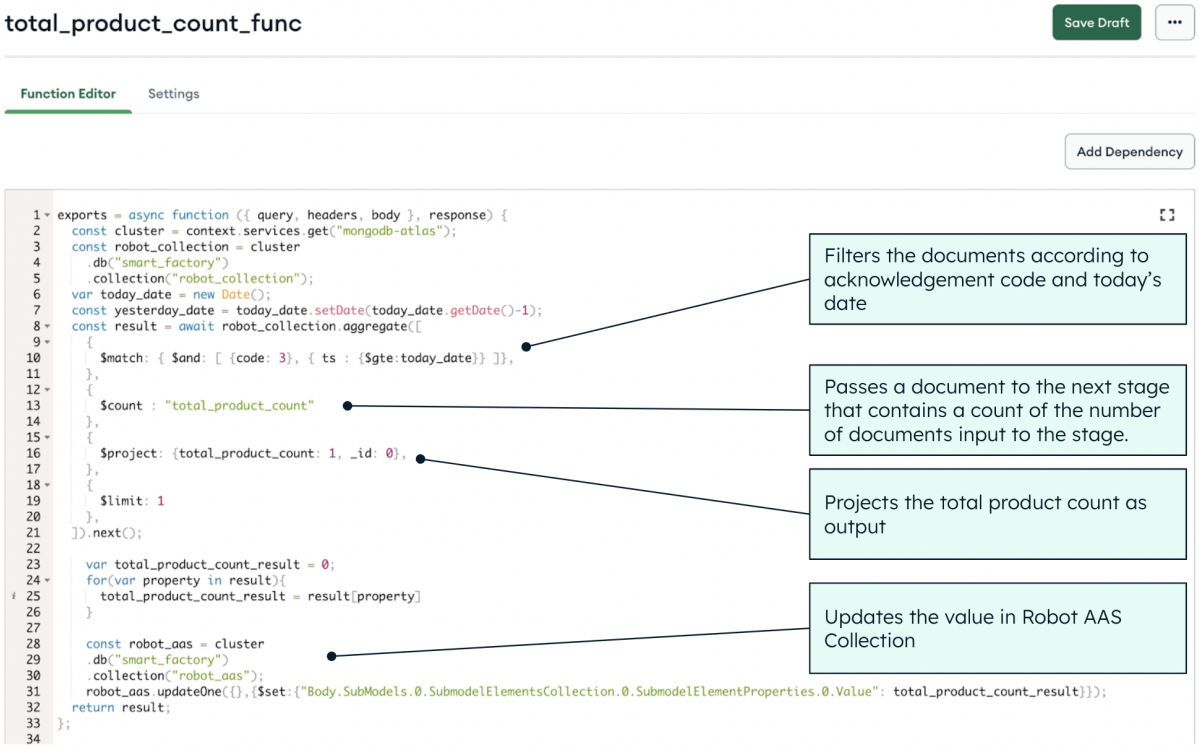

Figure 15: Custom Functions in Triggers

The overall design is shown in Figure 16, where the robot messages are streamed into MongoDB Atlas through Kafka streams and then using the scheduled trigger the collections are updated with necessary data.

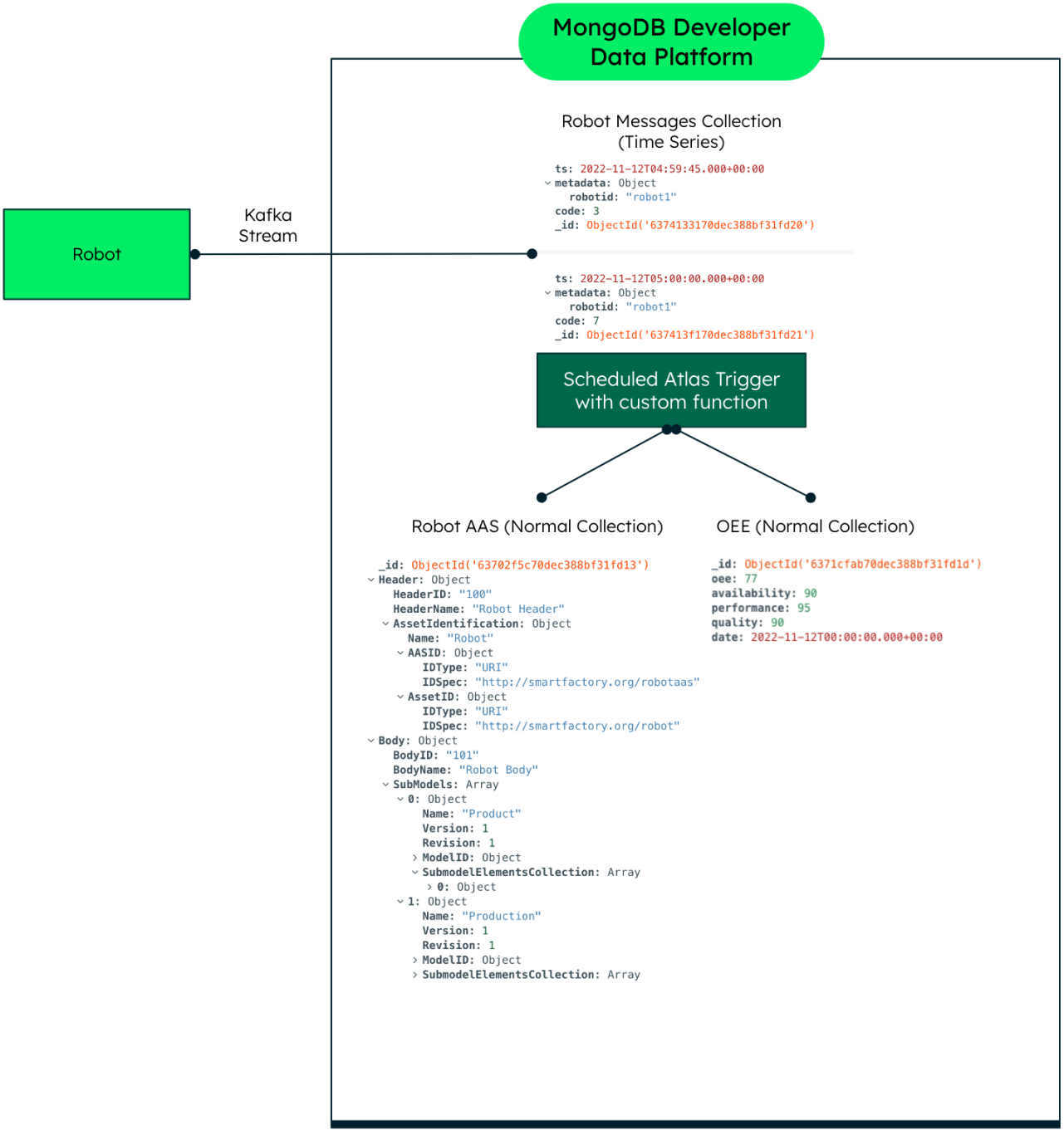

Figure 16: Updating collections using MongoDB Atlas Triggers

Using Data API and MongoDB Drivers

The MongoDB Atlas Data API lets you read and write data in MongoDB Atlas with standard HTTPS requests. To use the data API, all you need is an HTTPS client and a valid API key.

The data API uses JSON and EJSON (MongoDB Extended JSON) formats to represent data in requests and responses. This makes it straightforward to send and receive data from MongoDB Atlas because anything that you can represent in JSON can be natively stored in MongoDB, and retrieved just as easily in JSON.

The data API supports two types of endpoints:

Data API Endpoints are automatically generated endpoints that each represent a MongoDB operation. You can use the endpoints to create, read, update, delete, and aggregate documents in a MongoDB data source.

Custom HTTPS Endpoints are app-specific API routes handled by functions that you write. You can use custom endpoints to run your app's backend logic or as webhooks that integrate with external services.

Similar to triggers, we can use custom HTTPS endpoints and MongoDB aggregation framework inside a custom function to extract data from acknowledgment messages collection and calculate total and accurate product counts for a specific date. This calculation can be implemented using window functions. Window functions allow you to run a window across a sorted set of documents, producing calculations over each step of the window, like a rolling average or count.

The custom HTTPS endpoint, whenever called from an external application will respond with the total product count as an integer value. The AAS application only needs to perform 3 GET API calls at the end of the shift to extract all information necessary for the updates in the AAS document. Similarly, the OEE collection can be updated after calculating the values using the formulae in Step 1. To insert the returned values back into the AAS collection, a POST request can be used along with a custom function.

This approach is only useful if you wish to set up an external application to retrieve data from and into MongoDB Atlas. MongoDB Atlas Data API key use cases include:

1. Serverless development
2. Integration with other cloud apps and services
3. Environments where MongoDB drivers aren't an option

MongoDB supports twelve official libraries to connect your application to the database. These libraries enable you to perform CRUD operations, run aggregation pipelines, manage authentication and even perform encryption activities on fields from client applications. There are multiple community supported libraries as well.

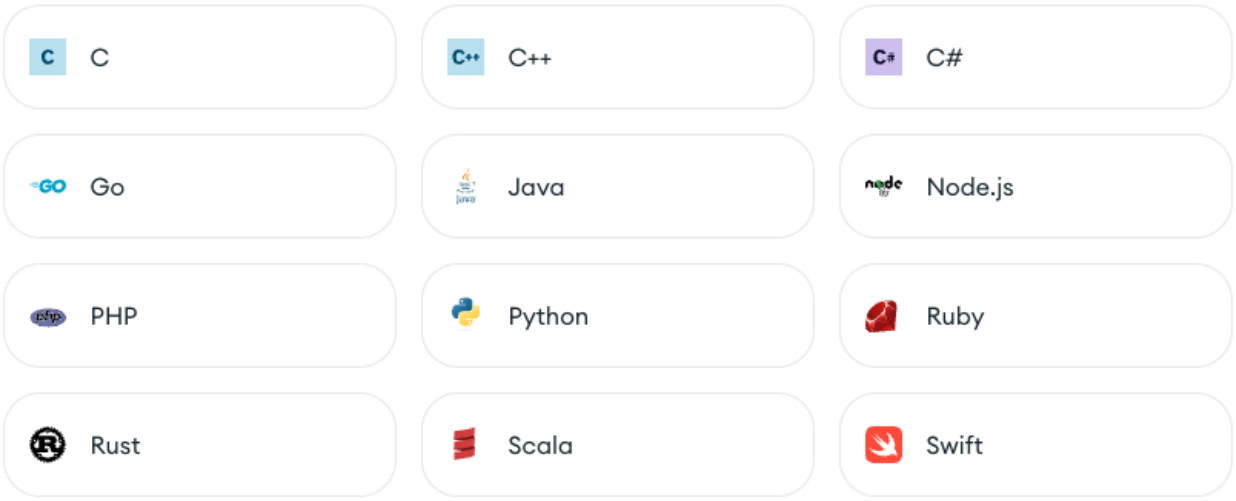

Figure 17: MongoDB Official Driver Libraries

Regardless of whichever option is used, the key point is that MongoDB Atlas provides inherent flexibility to the developers to build their applications.

Step 5 - Visualization of Robot Performance using MongoDB Atlas Charts

We have talked about pushing data into MongoDB, transforming it, and updating it automatically using Triggers and through external applications using Data API, but our implementation of AAS is not complete yet as we want to visualize the performance of our robot using charts. Through impactful visualizations, we can find actionable insights quickly and easily and build data-driven collaboration and knowledge-sharing practices. There are two classes of charting tools available on the market: 1) commercial Business

Intelligence (BI) tools ,for example, Tableau or PowerBI and 2) visualization libraries for example D3 or Charts.js. These two classes share similar problems and challenges. They are time consuming, complex, and costly. Commercial tools require development time to set up connectors and ETLs and they often lack real-time data refreshes. Licenses can be expensive and costs add up. In the case of visualization libraries, you need highly skilled resources to build and maintain code. While there usually is no paid license required, the cost of development time and maintenance can add up over time.

MongoDB Atlas Charts is the best way to visualize MongoDB data. It is built for the document model and visualizing JSON data so you can do more faster. It is integrated with MongoDB Atlas so there are no setup or ETL requirements. You can even embed charts and dashboards quickly via iFrame or with rich customization through the embedding SDK.

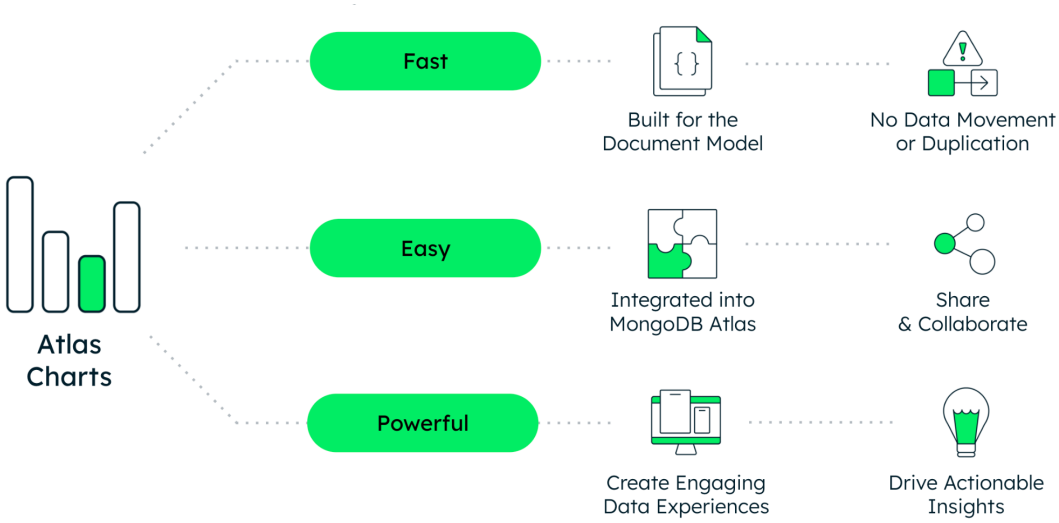

Figure 18: MongoDB Atlas Charts benefits

We can set up an OEE dashboard in minutes using MongoDB Atlas Charts. The first step is to identify the data collection which in our case is the robot OEE collection. Once done, then all we need to do is to choose the right widget to display our data, link that widget to our field and we can have a neat visualization as shown in Figure 19 below. To understand more about how to use widgets in MongoDB Atlas Charts, please refer to the detailed documentation.

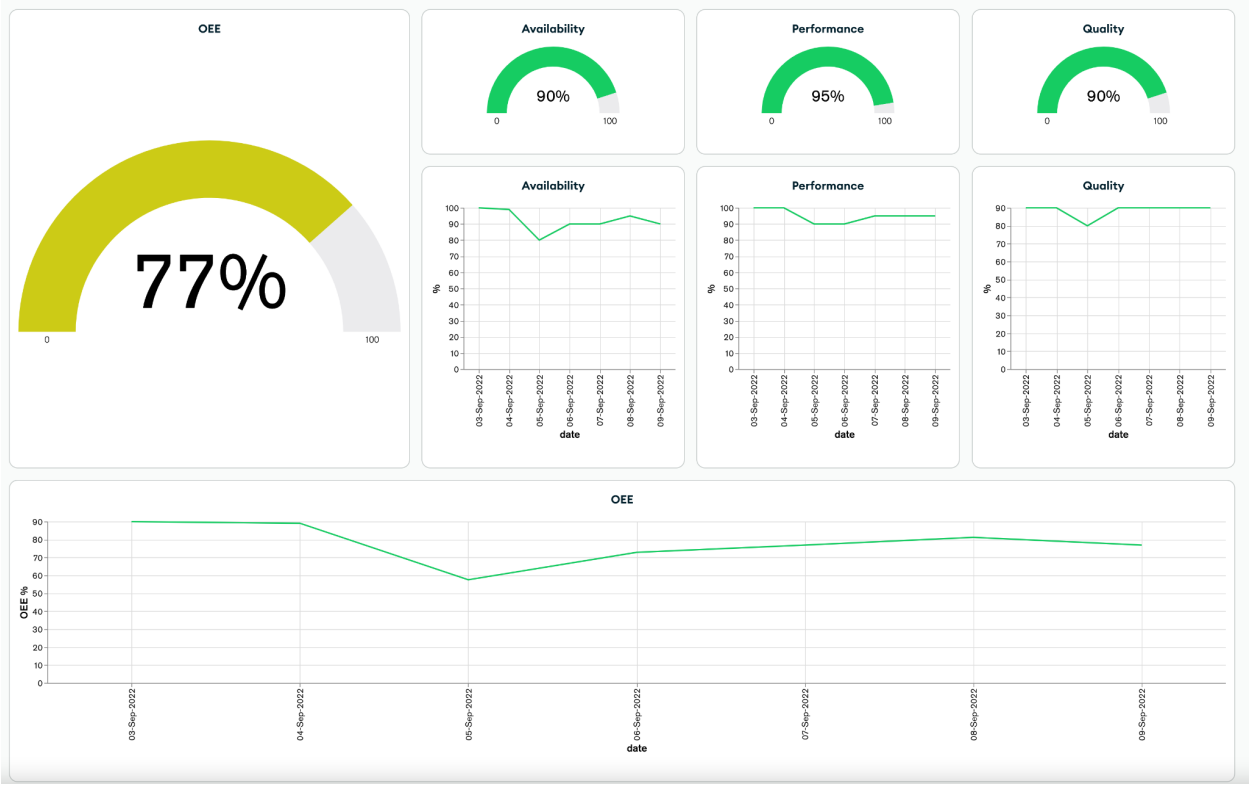

Figure 19: Robot OEE Dashboard using MongoDB Atlas Charts

The complete architecture of our use case looks is shown below:

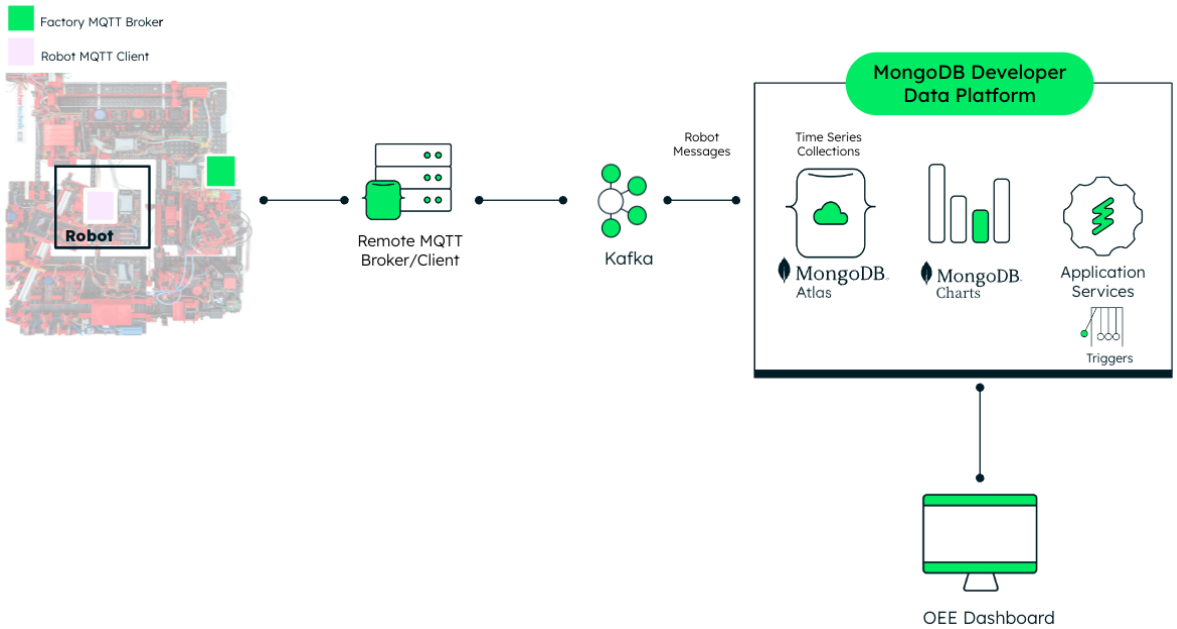

Figure 20: Complete Architecture Diagram with MongoDB Developer Data Platform (using Triggers)

Conclusion

MongoDB is strongly positioned to implement Asset Administration Shells and enable Industry 4.0 capabilities for the manufacturing industry, with a strong set of features and functionality that cover the entire lifecycle of IIoT data. These capabilities allow MongoDB to be in a unique position to fast-track the digital transformation journey of our manufacturing clients.

Get in touch with the MongoDB Team to find out how we can support your digital transformation vision come to life

About MongoDB

MongoDB empowers innovators to unleash the power of software and data. Whether deployed in the cloud or on-premises, organizations use MongoDB for trading platforms, global payment data stores, digital end-to-end loan origination and servicing solutions, general ledger system of record, regulatory risk, treasury and many other back-office processes. At the core of our developer data platform is the most advanced cloud database service on the market, MongoDB Atlas, which can run in any cloud, or even across multiple clouds to get the best from each provider with no lock-in.

To learn more about MongoDB, visit MongoDB.com

About the author

Humza is a Principal in the Industry Solutions Team at MongoDB looking after Manufacturing and IoT use cases. Prior to joining MongoDB, he was working at Ernst & Young Canada as a Senior Manager, in digital operations business consulting practice. Humza did his PhD at Nanyang Technological University, Singapore, and worked with the Singapore manufacturing industry for a number of years on Industry 4.0 research and implementation. He has spent most of his career enabling smart and connected factories for many manufacturing clients. In 2020-2021, He established a multi-year strategic roadmap for Singapore's smart supply chain initiatives. His book on Industry 4.0 is the first book of its kind detailing the real-world implementation of concepts related to digital manufacturing.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.
Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.com)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)
MongoDB Enterprise Download (mongodb.com/download)
MongoDB Realm (mongodb.com/realm)

Legal Notice

This document includes certain "forward-looking statements" within the meaning of Section 27A of the Securities Act of 1933, as amended, or the Securities Act, and Section 21E of the Securities Exchange Act of 1934, as amended, including statements concerning our financial guidance for the first fiscal quarter and full year fiscal 2021; the anticipated impact of the coronavirus disease (COVID-19) outbreak on our future results of operations, our future growth and the potential of MongoDB Atlas; and our ability to transform the global database industry and to capitalize on our market opportunity. These forward-looking statements include, but are not limited to, plans, objectives, expectations and intentions and other statements contained in this press release that are not historical facts and statements identified by words such as "anticipate," "believe," "continue," "could," "estimate," "expect," "intend," "may," "plan," "project," "will," "would" or the negative or plural of these words or similar expressions or variations. These forward-looking statements reflect our current views about our plans, intentions, expectations, strategies and prospects, which are based on the information currently available to us and on assumptions we have made. Although we believe that our plans, intentions, expectations, strategies and prospects as reflected in or suggested by those forward-looking statements are reasonable, we can give no assurance that the plans, intentions, expectations or strategies will be attained or achieved. Furthermore, actual results may differ materially from those described in the forward-looking statements and are subject to a variety of assumptions, uncertainties, risks and factors that are beyond our control including, without limitation: our limited operating history; our history of losses; failure of our database platform to satisfy customer demands; the effects of increased competition; our investments in new products and our ability to introduce new features, services or enhancements; our ability to effectively expand our sales and marketing organization; our ability to continue to build and maintain credibility with the developer community; our ability to add new customers or increase sales to our existing customers; our ability to maintain, protect, enforce and enhance our intellectual property; the growth and expansion of the market for database products and our ability to penetrate that market; our ability to integrate acquired businesses and technologies successfully or achieve the expected benefits of such acquisitions; our ability to maintain the security of our software and adequately address privacy concerns; our ability to manage our growth effectively and successfully recruit and retain additional highly-qualified personnel; the price volatility of our common stock; the financial impacts of the coronavirus disease (COVID-19) outbreak on our customers, our potential customers, the global financial markets and our business and future results of operations; the impact that the precautions we have taken in our business relative to the coronavirus disease (COVID-19) outbreak may have on our business and those risks detailed from time-to-time under the caption "Risk Factors" and elsewhere in our Securities and Exchange Commission ("SEC") filings and reports, including our Quarterly Report on Form 10-Q filed on December 10, 2019, as well as future filings and reports by us. Except as required by law, we undertake no duty or obligation to update any forward-looking statements contained in this release as a result of new information, future events, changes in expectations or otherwise.