Digital Twins IN ACTION

Greg Biegel

MEAP

MANNING

MEAP Edition
Manning Early Access Program

Digital Twins in Action

Version 3

Welcome

Thank you for purchasing the MEAP for *Digital Twins in Action*.

Digital twins have been a hot topic for several years now and there is a lot of excitement, interest, and opportunity in the field. A lot of this is because the technology that enables you to build a digital twin has become accessible to just about anyone now, opening up endless possibilities for innovation.

Despite all the buzz, there's surprisingly little practical guidance available. Google "digital twins" and you'll find countless glossy brochures, white papers, and marketing materials filled with futuristic 3D renderings but often lacking real substance. When you try to go deeper, you quickly discover there's no comprehensive, practical guide explaining what a digital twin actually is and, more importantly, how you can build one yourself. This gap isn't surprising. Unlike well-established technologies like the Java programming language or relational databases which have dozens of excellent books dedicated to them, a digital twin isn't a single technology but rather a system integrating multiple advanced disciplines: the Internet of Things, computer graphics, databases, artificial intelligence, and machine learning, each complex enough to warrant its own book.

I'll admit it's a bit of a cliche, but I've genuinely tried to create the book I wish I'd had when starting my digital twin journey. You'll explore all the essential topics for building a digital representation of a physical system—from creating a simple sensor to measure real-world changes, all the way

through programming an AI agent to make autonomous decisions. I'll provide practical examples throughout and take you along as I build a digital twin of my own home.

While you'll need some familiarity with Python to follow the practical examples, and TypeScript for the frontend components, I believe you'll gain valuable insights even if you choose not to follow every line of code! I primarily use AWS for cloud services, but I'll explore other providers whenever they offer compelling alternatives.

Your feedback is crucial to making this the best book possible, and I encourage you to share your thoughts in the liveBook Discussion Forum. I'm genuinely excited to hear your perspectives and answer any questions as we build this resource together.

Thanks again for your interest and trust in purchasing the MEAP!

-Greg Biegel

Brief contents

1 Bridging the physical and digital worlds

This chapter covers

- Defining what a digital twin is
- Different levels of digital twin maturity
- What digital twins are good for
- How digital twins are used across industries
- Considerations when embarking on building a digital twin

Digital twins are virtual models of physical systems that continuously synchronize with real-world data to monitor current conditions and predict future behavior. NASA pioneered this approach decades ago with simulators for the Apollo program. Yet for years, digital twins remained accessible only to large, well-funded organizations that could afford the scarce, expensive sensors, computing capacity, and advanced analytics required to build them. That’s all changed.

Today, digital twins can be built by almost anybody, with an array of low-cost sensors, high-performance, pay-as-you-go computing, and powerful artificial intelligence and machine learning tools widely available.

Some think of digital twins as detailed 3D models. while others see them as glorified dashboards or traditional simulation systems. In this book, we define a digital twin as a system that combines elements from all these interpretations, but is not restricted to any single technology.

What exactly counts as a digital twin? Let’s start with a clear definition before exploring why they’ve become so popular.

1.1 What is a digital twin?

In his 1991 book *Mirror Worlds*, computer scientist David Gelertner imagines building a physical model of a city on your living room floor containing miniature models of buildings and cars, combined with blackboards radiating information related to other things people care about, like the city budget, the air quality, and the waiting time in the emergency room. An army of people would gather data in the city and pass it back to your living room to update this model, allowing you to see, at a glance from your sofa, what is going on throughout the city and use this to influence your own decisions. He then imagines this model living in software, allowing countless people to observe the city simultaneously, each at their preferred level of detail.

Around the same time, Michael Grieves was developing similar concepts in the manufacturing world. In 2002, while teaching at the University of Michigan, Grieves presented what he called the "Mirrored Spaces Model" and later the "Information Mirroring Model"—a framework for creating virtual representations of physical products throughout their lifecycle. His work laid the groundwork for what would eventually be called digital twins in industrial settings.

What Gelertner and Grieves envisioned has become reality with the emergence of digital twins. The term has been used in many contexts since then and can be defined in different ways, but for this book, a digital twin is defined as:

> *A digital representation of a physical system that is updated with real-world data. It helps users understand current conditions, monitor performance, and simulate scenarios to make better decisions based on clearly defined objectives. Digital twins can also act on the physical system, sending instructions to adjust or optimize its behavior.*

This definition emphasizes that digital twins should not be built merely because the technology is available, but should serve specific business, operational, or research goals. By providing a computer-based model that mirrors something physical in reality, they help users to comprehend both the state of the object or system, and its structure—how components are organized and connected. By remaining synchronized through continuously receiving updates about current conditions, digital twins allow users to track the behavior of the physical system while also predicting the effect of changes without changing the real system.

For most of human history, you had to be physically present to understand how something worked. Digital twins have changed that, and today, countless people can simultaneously monitor, analyze, and run simulations on the same physical system from opposite sides of the globe.

1.2 Technology enabling digital twins

Digital twins first gained traction in capital-intensive industries like manufacturing, utilities, and energy. These industries require massive investments, often operate in remote or hazardous environments, and face strict safety and regulatory requirements. Since equipment costs are high, even small improvements in efficiency or reductions in downtime quickly justify technology investments.

While these industries have used operational technology (OT) systems like programmable logic controllers (PLC) and supervisory control and data acquisition (SCADA) for monitoring and control since the 1960s, digital twins represent a major leap forward. By integrating IoT sensors, cloud computing, and AI/ML, modern digital twins can process vast amounts of data, run sophisticated predictive models, and provide actionable insights that traditional control systems alone could never deliver.

The convergence of these technologies enables three key capabilities: virtual modeling during design to reduce physical prototyping, predictive maintenance to prevent expensive failures, and real-time optimization to maximize operational efficiency.

Consider a wind farm where engineers first use the digital twin to virtually test turbine designs under simulated conditions. Once deployed, IoT sensors stream data to the cloud where AI algorithms detect early signs of bearing wear. The digital twin then simulates maintenance scenarios and schedules repairs, while simultaneously adjusting blade pitch across the farm to maximize energy capture based on real-time weather patterns.

1.2.1 Internet of things (IoT)

Not too long ago, measuring things such as temperature, vibration, location, or air quality required bulky and expensive equipment. The affordable and compact sensors available today enable users to capture detailed data about virtually any physical environment or system, opening unprecedented possibilities for monitoring and control. Hobbyists can now build sophisticated home automation systems that measure everything from air quality to energy usage, while startups can deploy sensor networks to monitor

equipment performance or environmental conditions at a fraction of historical costs.

> ***NOTE***
>
> The DHT22 sensor that I use to measure temperature and humidity in my home costs around $5 today. Three decades ago, comparable digital measurement required industrial sensors and data acquisition systems costing thousands of dollars, accessible only to well-funded organizations.

Large enterprises leverage these same technologies to enable monitoring of manufacturing lines, predictive maintenance of critical equipment, and optimization of resource usage. The sensors already embedded in smartphones demonstrate the power of this technology, with users able to measure motion, orientation, light levels, and even perform 3D spatial mapping, turning everyday devices into powerful data collection tools. Figure 1.1 shows output from an iPhone's three-axis accelerometer while the phone is carried in a pocket. The distinctive pattern in this data allows us to infer the physical state of the system, namely that the person carrying it is walking. This democratization of sensing technology means that anyone can now gather the rich, continuous data streams necessary to build digital twins that were once only available to organizations with massive budgets and specialized expertise.

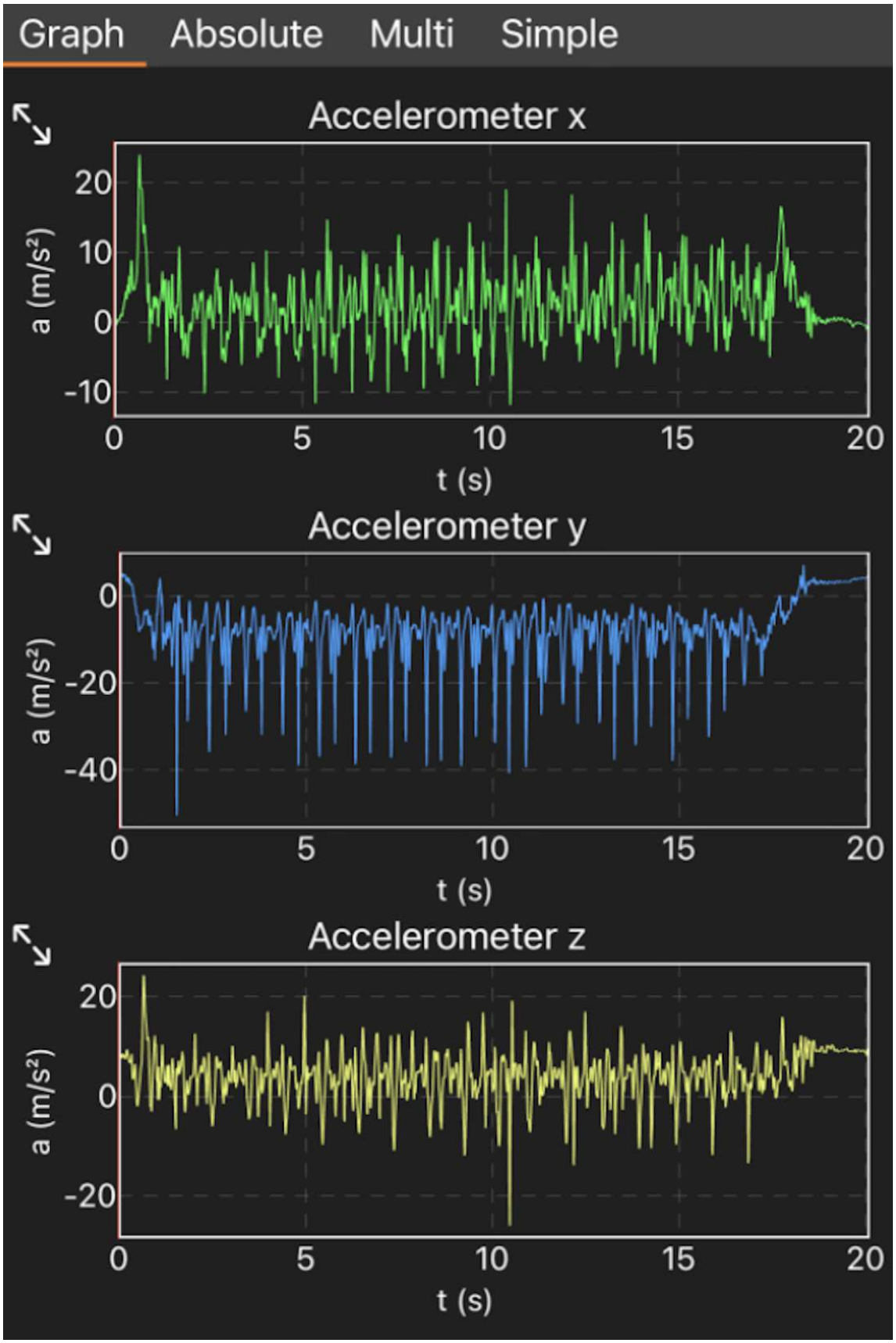

Graph
Absolute
Multi
Simple
Accelerometer x
a (m/s²)
20
10
0
-10
0
5
10
15
20
t (s)
Accelerometer y
a (m/s²)
0
-20
-40
0
5
10
15
20
t (s)
Accelerometer z
a (m/s²)
20
0
-20
0
5
10
15
20
t (s)

Figure 1.1 A screenshot shows the output from a 3-axis accelerometer in a modern iPhone in the phyphox app (https://phyphox.org/) showing data indicating the owner is walking with their phone in their pocket. Such sensor data enables a digital twin to mirror and interpret real-world motion and behavior in real time.

Gathering measurements is only half the story; the data must be sent somewhere useful. Advances in wireless communications have matched this sensor revolution. Fifth-generation (5G) cellular networks offer high speeds for high-bandwidth data. In contrast, Low-Power Wide Area Networks (LPWANs) like LoRaWAN can transmit over 10 km (compared to Wi-Fi's 50–100 meters) using 10–20 times less power than Wi-Fi. This efficiency enables sensors to run for years on a single battery, making them ideal for small, resource-constrained devices deployed globally.

IoT is the combination of these widely available commodity sensors and actuators ("things") embedded into objects and connected to communications networks. As you move through the book, you'll learn how to wire up a fleet of LoRaWAN-based sensors, route their data through a cloud pipeline, and use it to update a digital twin model in near real time.

1.2.2 Cloud computing

In the past, deploying a digital twin was restricted to large industrial players due to the massive upfront investment required for hardware (data storage, analysis, simulation, and 3D rendering). Today, modern public cloud computing provides the essential infrastructure needed to create and operate sophisticated digital twins without that barrier.

Users can access scalable compute resources on-demand for complex simulations, elastic storage for continuous sensor data streams, and managed services for data processing, analytics, and machine learning. This pay-as-you-go model

eliminates the traditional constraints of purchasing expensive servers and specialized software licenses, allowing rapid prototyping and scaling.

Edge computing complements the cloud by processing data closer to where it's generated. For industrial equipment or remote assets, edge devices perform local analytics, filtering, and decision-making in milliseconds, reducing latency for time-critical operations and minimizing bandwidth costs. For example, a security camera can run a machine learning model to detect defects or safety violations on a production line and transmit only the alerts and metadata rather than streaming hours of raw video footage to the cloud.

In this book, we will look at how you can use some of the same cloud services that industrial organizations use today to help you build a digital twin.

1.2.3 Artificial intelligence (AI) and machine learning (ML)

The massive datasets from IoT sensors must be filtered, stored, indexed, and analyzed to generate actionable insights. Recent advances in AI and ML, partly enabled by specialized GPU hardware, have shifted these technologies into widespread use. Combined with IoT and cloud computing, they are essential for modern digital twins.

AI/ML capabilities are crucial for analyzing continuous sensor streams and making predictions about future system behavior. Cloud providers offer user-friendly tools that make these advanced capabilities accessible. Automated platforms handle the complex process of building, training, and deploying models—pre-processing sensor data, selecting

algorithms, and optimizing performance without requiring deep technical knowledge.

For specific industries, specialized platforms offer pre-built solutions. When real-time responses are critical or cloud connectivity is unreliable, edge computing services allow AI processing to happen locally, enabling immediate decision-making. These tools collectively lower the barrier for organizations to add intelligent prediction, anomaly detection, and automated decision-making capabilities to their digital twins.

Furthermore, these capabilities are not only in the cloud. We will look at how you can download pre-trained models for common tasks such as object-detection, and timeseries forecasting, that can be customized and run locally, giving you both control over your data and the ability to operate when internet connectivity is unavailable.

1.2.4 Agentic AI and autonomous decision-making

Agentic AI represents a shift toward systems composed of software agents that autonomously reason, plan, and execute complex workflows across digital twins. These agents can monitor systems, learn from data, make decisions, and perform actions to achieve goals like reducing costs or optimizing operations, either individually or in collaboration with other agents.

Digital twins feed agentic AI the data it needs, while the AI enhances the twin through autonomous optimization and decision-making.

1.3 What makes a good digital twin?

When you hear "digital twin", you might picture a sleek 3D visualization of a complex machine. While visualizations are a common component, they are not the whole story. In fact, some digital twins operate as *headless twins*—systems focused purely on data processing, analytics, and automated decision-making without any visual component at all.

Digital twins range from simple monitoring dashboards to complex predictive and autonomous models. It is useful to categorize them by maturity to help identify the right goal for your needs and measure the expected business value. This book follows the five-level maturity model developed by Verdantix in a 2019 report titled "Smart Innovators: Digital Twins For Industrial Facilities", which categorizes digital twins from basic descriptive systems to advanced autonomous operations, as shown in figure 1.2.

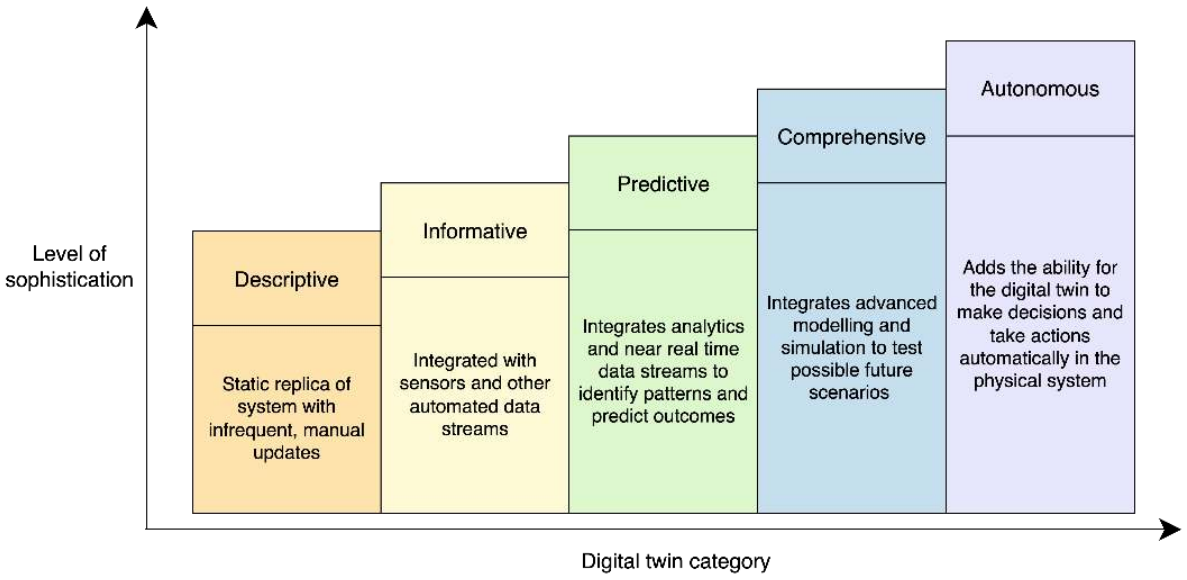


Figure 1.2 The five categories of digital twin.

1.3.1 Descriptive digital twin

At the most basic level is the *descriptive digital twin*. This twin provides a reasonably static digital representation of a

physical object or system, without making real-time predictions or responding to changes. This representation might be a simple engineering diagram or a fully rendered 3D visualization.

REAL WORLD EXAMPLE: ONLINE MAPS

A familiar example of a descriptive digital twin is Google Maps. What you see when you use this product is a digital model of real-world cities containing roads, buildings, and parks and generated from cartographic maps, satellite imagery and other sources, as shown in figure 1.3.

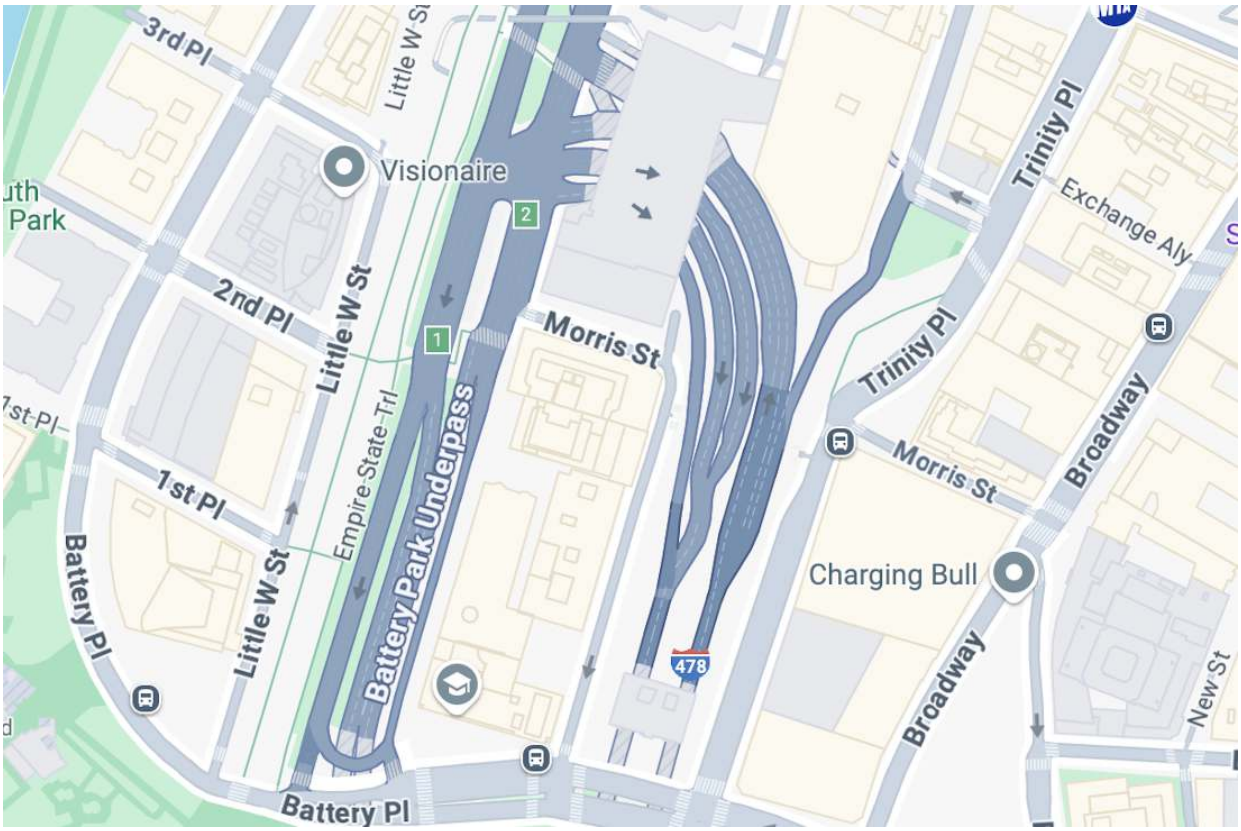


Figure 1.3 Google Maps view of lower Manhattan, a familiar example of a descriptive digital twin. Map data © 2025 Google. Google Maps is a trademark of Google LLC.

In recent years, Google Maps has evolved from simple two-dimensional maps to rich, photorealistic 3D renderings of the

built environment, as shown in figure 1.4. These visuals include buildings modeled through photogrammetry, allowing users to "fly through" cities from the other side of the world, almost as if they were there in person. As one of the largest efforts ever undertaken to capture and represent the physical environment digitally, it is not only a descriptive digital twin itself, but also provides an unprecedented digital foundation for others to build spatially accurate digital twins on top of.

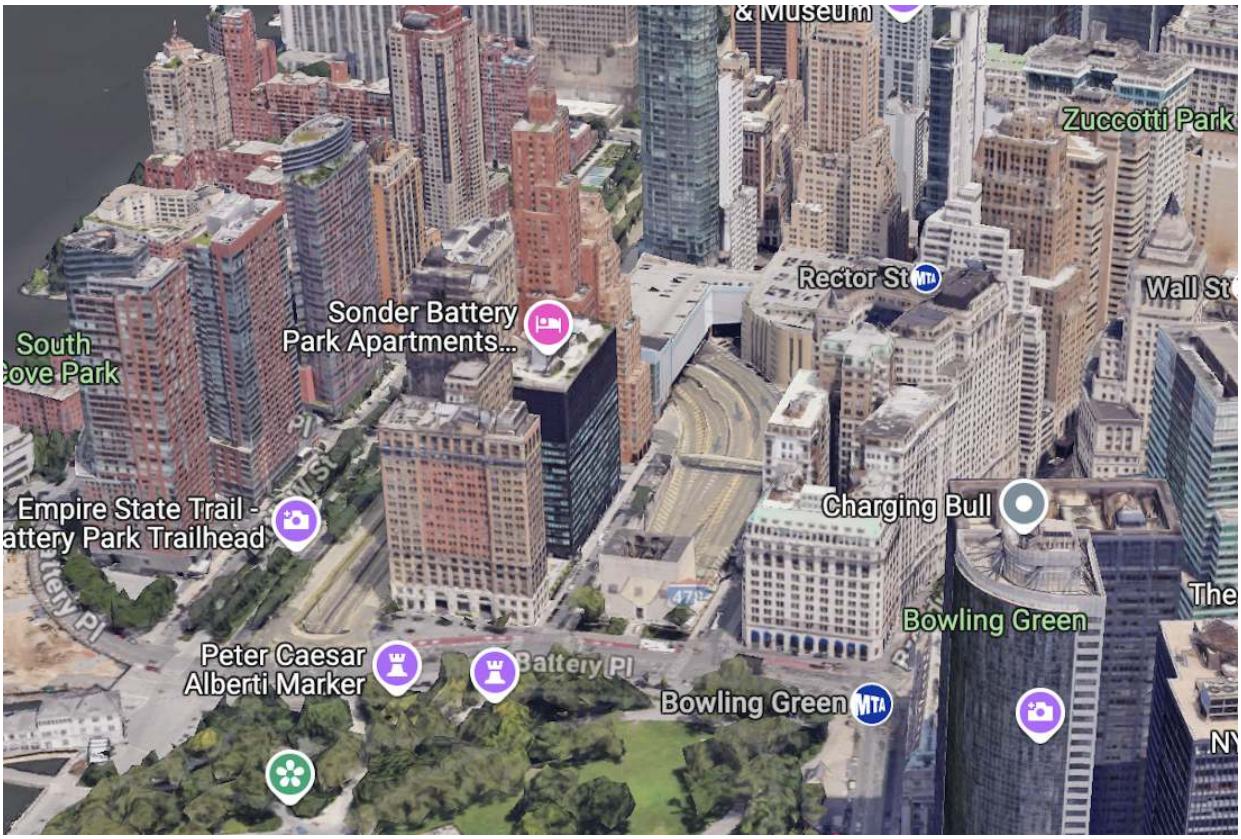


Figure 1.4 Google Maps view of lower Manhattan showing a photorealistic 3D view of the built environment provides a more detailed example of a descriptive digital twin. Imagery © 2025 Google, Map data © 2025, Map data © 2025 Google.

However, the visual representation is only part of the story. Behind the scenes, Google Maps relies on a graph-based model of the road network. Roads are represented as edges in a graph, and intersections are nodes, as shown in figure

1.5. This data structure is what enables turn-by-turn navigation, route optimization, and even traffic prediction. In other words, the visual part of Google Maps is just the surface and underneath is a rich digital model of the physical infrastructure.

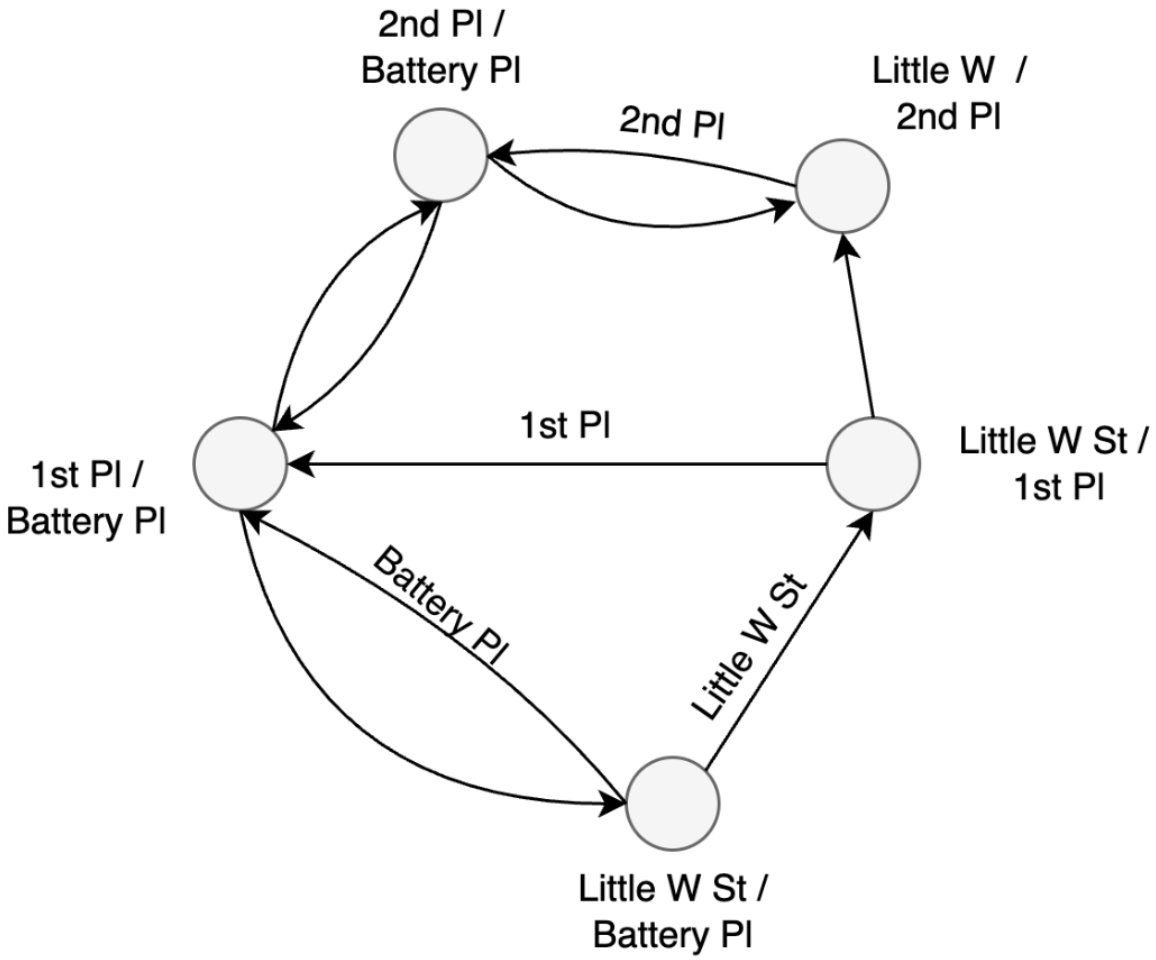


Figure 1.5 A directed graph model of a small subset of lower Manhattan as represented in Google Maps, with vertices representing intersections and edges representing roads with direction of travel.

In the context of digital twins, especially descriptive twins, graph-based models allow us to go beyond what we see and start structuring the world in a way that supports higher-level capabilities like simulation, prediction, and real-time analytics. As you move deeper into digital twin maturity, this structural backbone becomes increasingly important.

1.3.2 Informative digital twin

An *informative digital twin* integrates real-time data streams from the physical system to update the digital representation. This allows users to visualize the current state of the system and make better decisions.

In Google Maps, adding traffic data creates an informative digital twin at a global scale. This traffic data, gathered from fixed sensors and mobile devices, is rendered by color-coding streets according to traffic density and speed as shown in figure 1.6. This pattern of overlaying data layers on map views is one of the most successful design patterns in informative digital twin development.

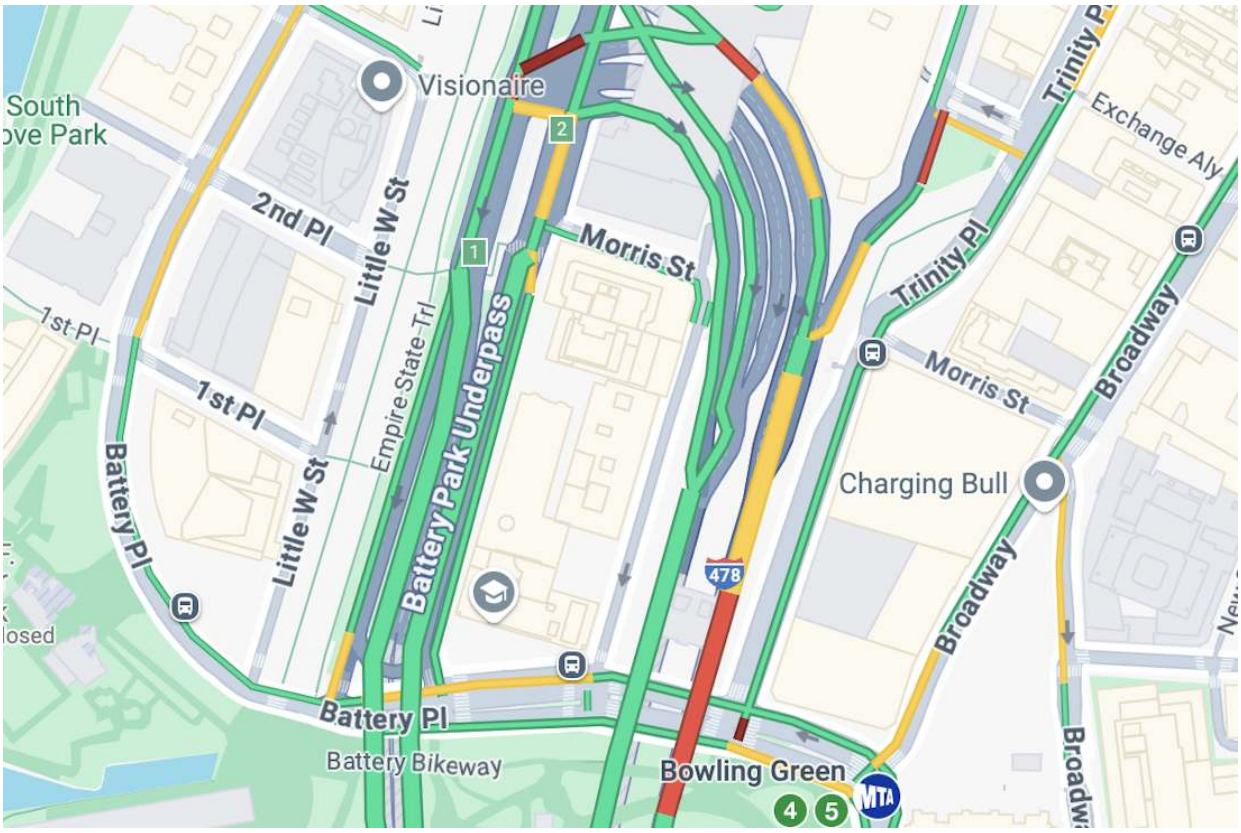


Figure 1.6 Google Maps view of lower Manhattan with realtime traffic congestion data overlaid is an example of an informative digital twin. Imagery © 2025 Google, Map data © 2025, Map data © 2025 Google.

REAL WORLD EXAMPLE: TEMPERATURE MONITORING

Other informative digital twins offer laid-out dashboards with graphs, gauges, dials, and other components. Figure 1.7 shows one displaying temperature fluctuations over several days inside and outside a building and how the indoor temperature is correlated to that outdoors. Behind this simple visualization lies a network of IoT sensors streaming data into the cloud that is then contextualized (for example, linking an individual sensor to a specific room). It is the combination of data acquisition, processing, storage, contextualization, and visualization that makes informative digital twins so valuable.

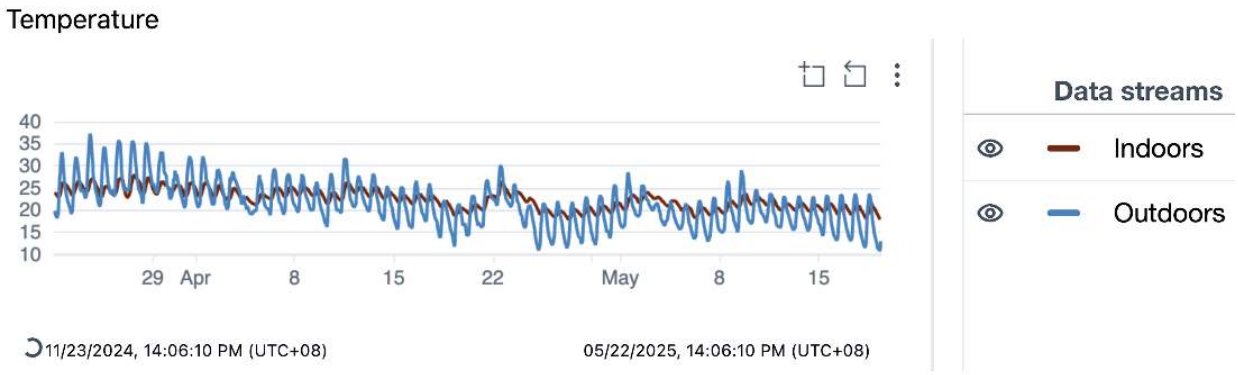


Figure 1.7 Information about the physical environment represented in a dashboard forms the basis of many informative digital twins.

1.3.3 Predictive digital twin

A *predictive digital twin* forecasts what the state of the physical system might be in the future and does not only reflect its current state. It uses historical data to anticipate future states of a physical system, based on the assumption that future states and behavior will be consistent with historical patterns. These predictions can range from simple rules, like alerting when a temperature reading crosses a dangerous threshold, to complex machine learning models trained on months or years of operational data.

Predictive digital twins are particularly valuable in industries that require extensive and costly maintenance of equipment, as they can help them move from periodic (replace parts every 6 months) or reactive (replace parts when the machine breaks) maintenance to predictive maintenance—replacing parts when they start to exhibit characteristics that may indicate failure.

NOTE

Predictive digital twins can transform scheduled or reactive maintenance into proactive optimization, fundamentally changing how organizations manage their physical assets and unlock new value-added services. Companies can now offer predictive maintenance subscriptions, performance guarantees, or outcome-based contracts, turning operational insights into revenue streams.

REAL WORLD EXAMPLE: PREDICTING EQUIPMENT FAILURE

Imagine an industrial pump equipped with a temperature sensor. Over time, it starts showing rising temperatures. A predictive digital twin monitoring this pump might be using a threshold-based rule (for example, "if the temperature exceeds 60 degrees Celsius, then trigger a warning") or a more advanced anomaly detection model trained on patterns from pumps that previously failed. The trained machine learning model knows that when the temperature of the pump has exceeded 60 degrees for more than two hours previously, this has caused a failure in the bearing. The goal is to forecast problems before they happen, avoiding costly breakdowns and unplanned downtime.

REAL WORLD EXAMPLE: PREDICTING TRAVEL TIME

One of the most widely used predictive digital twins is likely in your pocket right now. Open Google Maps, and it will estimate how long it'll take to drive anywhere you choose, not just based on current traffic, but on what traffic is likely to look like 30 minutes from now, as shown in figure 1.8. How does it do that?

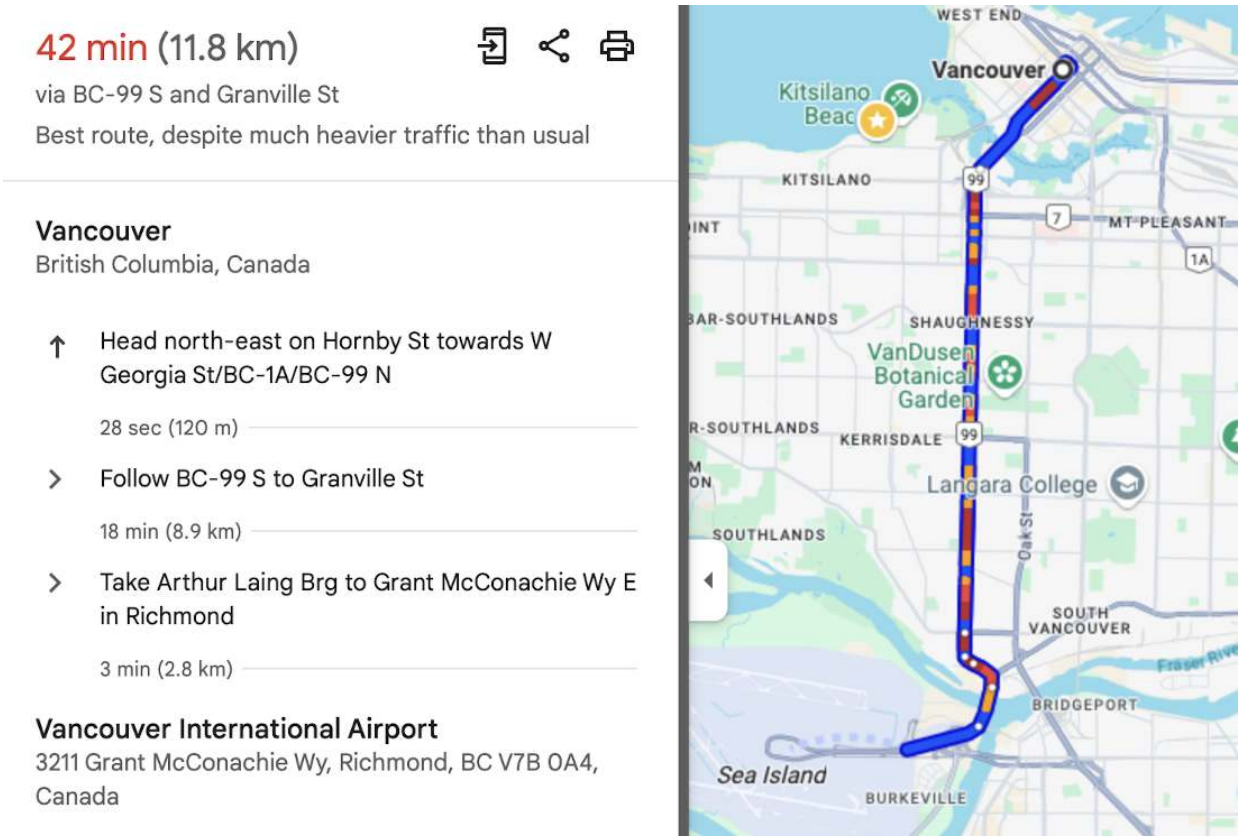


Figure 1.8 An example of a predictive digital twin provided by Google Maps showing predicted travel time. Map data © 2025 Google.

Under the hood, Google uses a Graph Neural Network (GNN) that operates on the road network, treated as a graph, as illustrated in figure 1.6. It blends live traffic data with historical trends to simulate how congestion might evolve. Google Maps combines a static representation of the physical road infrastructure with real time positional data and models trained on historical data to achieve remarkably accurate

travel time predictions. The accuracy of these models became apparent during the Covid 19 pandemic when lockdowns shifted traffic patterns overnight, necessitating Google to urgently retrain them on shorter historical traffic data to retain their accuracy.

NOTE

Read more about how Google DeepMind partners with Google Maps to predict traffic patterns with Graph Neural Networks here https://deepmind.google/blog/traffic-prediction-with-advanced-graph-neural-networks/.

REAL WORLD EXAMPLE: A DIGITAL TWIN OF YOU

It is not just machinery and infrastructure that can be modeled in the digital world, but ourselves too. Every time you browse a streaming service or shop online, you're providing data about yourself to systems that are essentially personal predictive digital twins. These systems build and continuously update a profile of your preferences, beliefs, habits, and behaviors. Based on this profile, they predict which item to recommend on Amazon, what music to queue on Spotify, what search results to return to you in Google, or what you might want to watch next on Netflix as shown in figure 1.9.

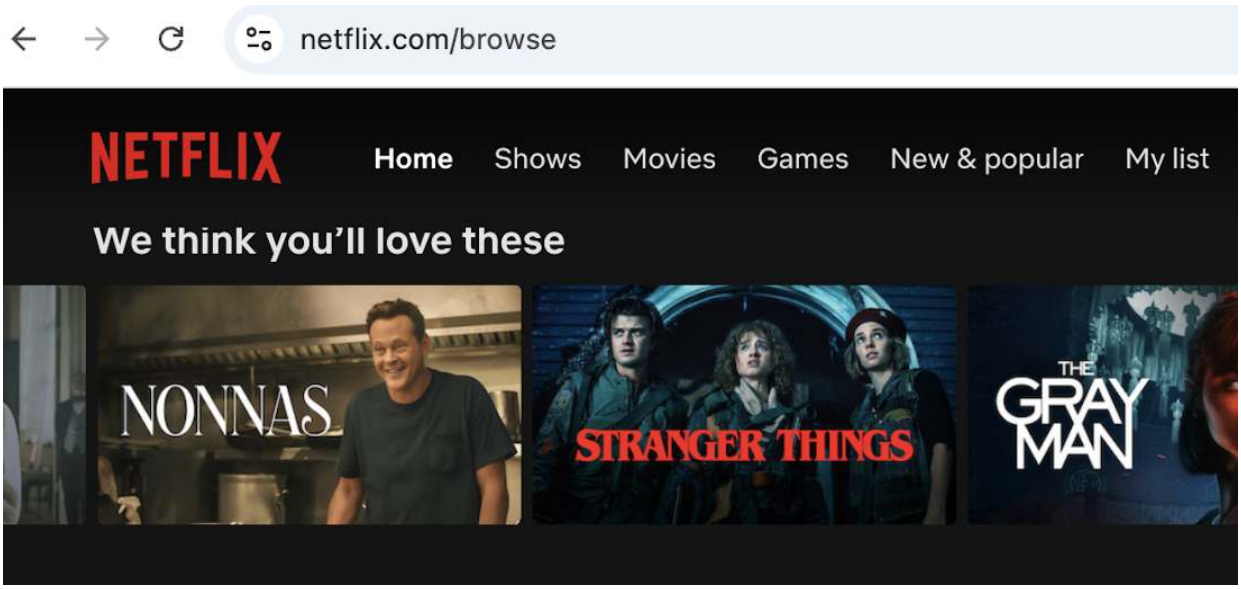


Figure 1.9 What Netflix thinks I would like to watch next based on its representation of my preferences learned through past shows I have watched.

The proliferation of these human-centric twins makes robust privacy protocols and clear ethical guidelines essential to prevent misuse, manipulation, or bias in how our digital reflections are managed and acted upon.

NOTE

Companies such as Delve.ai (https://delve.ai) market software that offers to build a so-called *digital twin of a customer (DToC)* and a *digital twin of an employee (DToE)*—with the promise of building a digital replica of a person to make predictions about their performance, behavior, and potential.

These predictive digital twins are constantly learning—from your clicks, views, likes, purchases, and even how you move your mouse over the screen. They're not based on models of physics and are more focused on patterns in data, but the

principles are the same: observe the past, understand the present, and forecast the future.

1.3.4 Comprehensive digital twin

A *comprehensive digital twin* takes things a step further. It actively simulates different possible futures in addition to monitoring a system and making forecasts. This lets us explore questions like, "What happens if we tweak this parameter?", "What if we push the system beyond its normal limits?" The goal is to understand how a physical system might behave under various conditions, even if we never touch the real thing.

MODELING MEETS REALITY

Simulation and modeling of complex systems isn’t new. Engineers and scientists have been doing this for decades, using mathematical models to represent everything from electrical grids to climate systems. Comprehensive digital twins extend traditional modeling by dynamically integrating real-world data, often through techniques such as state estimation or *data assimilation*. Data assimilation is the process of combining real-world observations with mathematical models to create the most accurate possible representation of a system’s current state.

> ***NOTE***
>
> Data assimilation is like giving your model a regular dose of reality. It compares what the model thinks should happen with what actually happened, then adjusts accordingly.

This makes the model not just a static representation of the system that predicts outcomes based on sensor data, but a living, evolving digital counterpart.

REAL WORLD EXAMPLE: PREDICTING THE WEATHER

Imagine you are planning to go on a hike at the weekend. You will probably look at your favorite weather app to see whether rain is in the forecast for the day you plan to go. Beyond the simple icon of clouds and raindrops lies a sophisticated chain of models, simulations, and real-world data—a comprehensive digital twin of the global weather maintained by institutions like the European Center for Medium-Range Weather Forecasts (ECMWF). The ECMWF maintains multiple numerical models of the atmosphere, known as general circulation models, that it combines with recent sensor observations from global weather stations and remote satellite data (using data assimilation) to get the best possible estimate of the current state of the atmosphere. The updated model becomes the foundation for predicting upcoming weather patterns and whether rain is likely.

The ECMWF is currently implementing an earth-scale digital twin called Destination Earth (https://destine.ecmwf.int/) that will provide a planetary-scale digital replica based on observations and simulations. The first two capabilities include multi-decade simulations of climate change, and on-demand simulations of weather-induced extremes, with the intent being to more accurately predict environmental disasters and mitigate resultant crises. Example output from a simulation run within the climate change adaptation digital twin that is used to inform wind farm design is shown in figure 1.10.

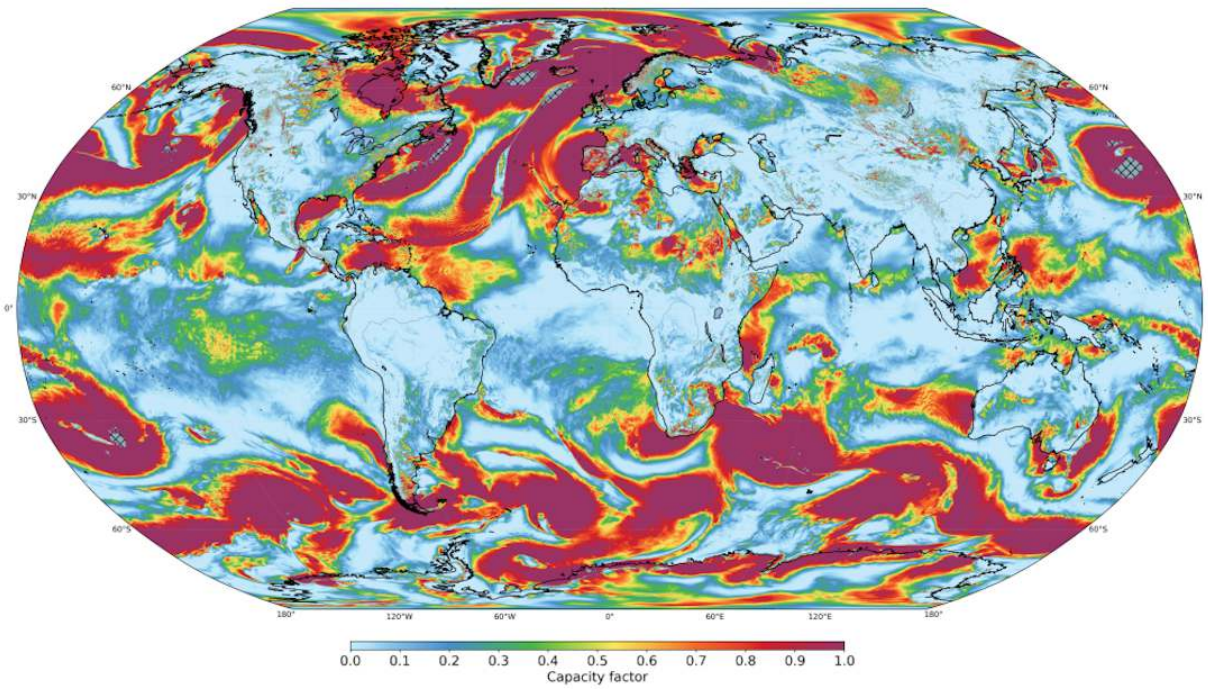


Figure 1.10 An example of a comprehensive digital twin—a view of indicators relevant to wind energy including hourly wind speed distribution with its changes at the multi decadal scale to help improve wind farm design generated with the Climate Change Adaptation Digital Twin, part of Destination Earth. Image © ECMWF. Licensed under CC BY 4.0.

1.3.5 Autonomous digital twin

An *autonomous digital twin* extends beyond purely modeling the physical system and closes the loop between the physical and digital realms by taking real world actions based on information, predictions, or simulations in the digital system. These actions can range from direct physical actuation—for example, opening a pressure release valve when tank pressure is predicted to exceed safe limits—to electronic actuation that ultimately results in an action such as the raising of a notification on a user's device.

Systems resembling autonomous digital twins have traditionally been found in industrial processes, but it's important to distinguish between different types of automated systems. Closed-loop process control systems, such as automated plant control (APC) systems managed by distributed control systems (DCS), use data from SCADA systems to control PLCs through pre-programmed rules and heuristics. They respond to sensor inputs according to fixed logic and a bounded set of conditions but do not learn, reason, or adapt their behavior based on new or unforeseen experiences outside of their defined envelope. Furthermore, these traditional OT systems typically operate in isolation, use proprietary protocols, and are focused primarily on real-time control and maintaining stability within set parameters, rather than holistic, future-oriented optimization of the entire asset or enterprise. The autonomous digital twin, in contrast, integrates agentic AI to reason about multiple, conflicting objectives, predict future states, and generate novel control strategies that adapt over time.

REAL WORLD EXAMPLE: MAINTAINING YOUR HOME TEMPERATURE

You don't need to walk through an industrial plant to see an autonomous digital twin in action. In fact, you might already live with one—and it's probably hanging on your wall.

Let's say you've installed a smart thermostat in your home. When you first turn it on, it behaves like any other thermostat, where you configure the temperature that you want, and it turns on the heat or the air conditioning to bring the temperature as close to that as possible, very much like a traditional closed loop process control system.

But after running for a few days, things start to change. It learns that you tend to lower the temperature at around 10

p.m. when you go to bed, and then turn it up again at 7 a.m. when you wake up in the morning. It also reads the local weather data and starts to form a correlation as to how the external conditions change your home's climate. It even communicates with your mobile phone to determine your physical location and uses this to infer when you are not at home. Eventually, it starts to make changes to the temperature settings on its own as shown in figure 1.11.

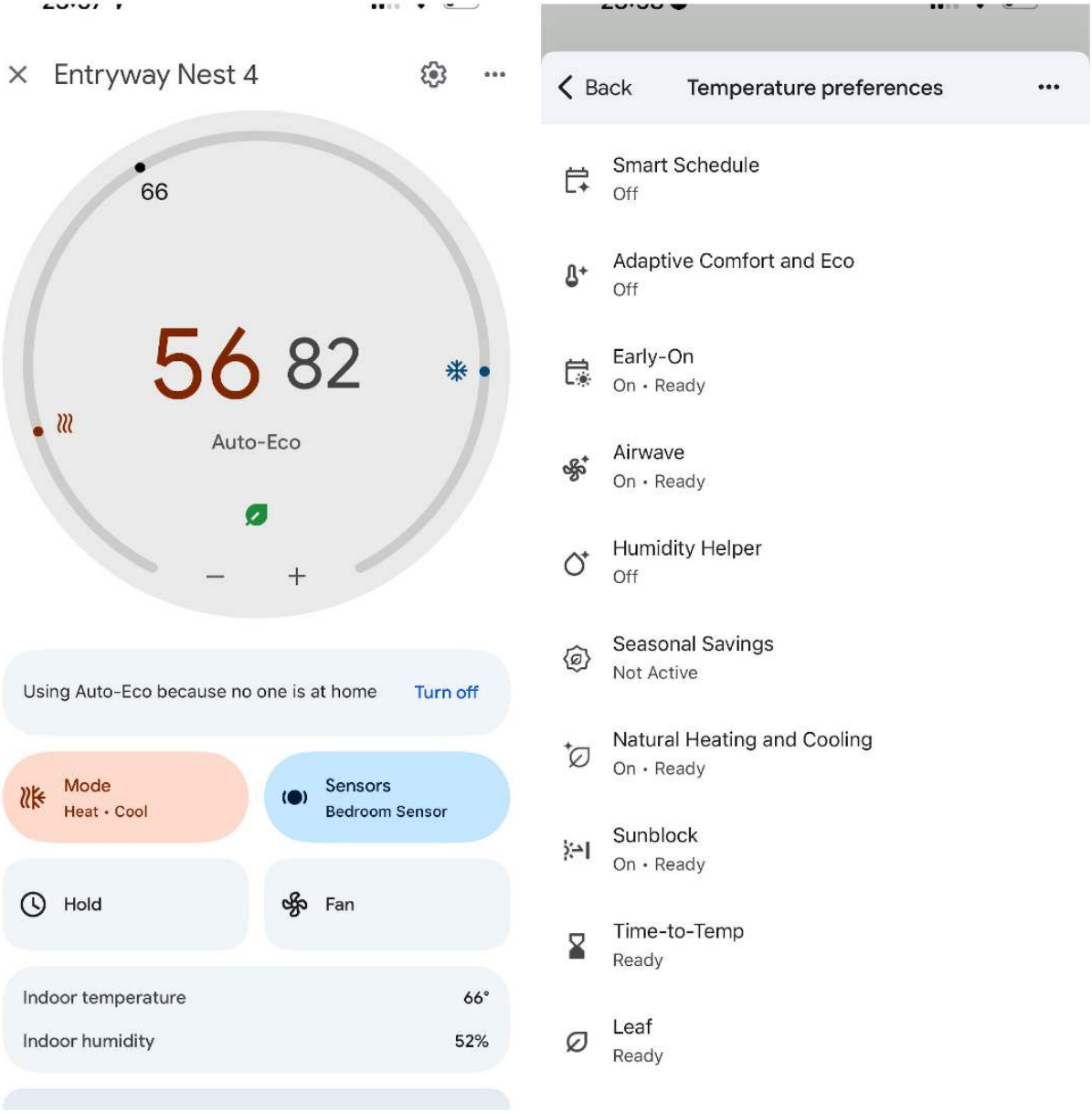


Figure 1.11 The interface to a Nest smart thermostat showing how it takes action based on your physical location. Eco mode is enabled based on presence sensing via the location of mobile phones in the household. Image © Thomas Smailus, Ph.D. P.E. Reproduced with permission.

At this point, your thermostat is more than just a remote control for your AC system. It's running a simplified digital twin of your home's thermal behavior. It knows how long it takes to warm up in the morning, how sunlight through your west-facing windows affects the living room, and when nobody's home. It's constantly comparing what should be

happening (according to its model) with what is happening (from real-time sensors), and it adjusts accordingly, all without your input.

What makes it an autonomous digital twin is that it’s not just logging data or reacting blindly. It’s assimilating sensor data into its model and acting autonomously to improve comfort and save energy. You’re actually living with a low-key autonomous digital twin right there in your home.

1.4 What are digital twins good for?

Hopefully, you are starting to form a picture in your mind as to how you might currently benefit from the digital twins we have touched on so far, but as someone looking to build and deploy a digital twin within their business, what are the potential benefits that they offer? Let’s explore some of these key advantages.

1.4.1 Accelerating product development

This pre-construction model allows engineers to test and simulate countless outcomes and "what-if" scenarios in a risk-free environment. By running simulations, developers can quickly identify performance limitations, predict specific failure modes, and refine their designs. This approach alters the traditional product development cycle, moving away from expensive, iterative physical prototyping. Instead, it enables the rapid testing of many design variants at a far lower cost, allowing physical tests to be focused only on the most promising designs and predicted critical failure points, thereby accelerating innovation.

REAL WORLD EXAMPLE: SIEMENS GAS TURBINE DEVELOPMENT

When Siemens developed the SGT-A65 gas turbine, they adopted a digital twin known as ATOM (Agent-based Turbine Operations and Maintenance). ATOM modeled the fleet operations of Siemens gas turbines, including engine characteristics, supply chain logistics, maintenance facility operations, and customer operations. Replacing a set of Excel-based forecasting tools and representing the entire system, ATOM reportedly provided enhanced analytical capabilities and enabled decision making that considered the entire system of turbine operations.

1.4.2 Reducing costs through predictive maintenance

There is a good reason that digital twins are often found today in industries that operate complex machinery. In such operations, the failure of a machine that runs continuously can have major consequences financially and environmentally, in terms of health and safety, or regulatory repercussions. Being able to gain the insight that a machine is starting to show anomalous behavior allows the operator to examine and repair it before it fails, thereby minimizing the risk of downtime in an approach known as *predictive maintenance*.

REAL WORLD EXAMPLE: ROLLS ROYCE INTELLIGENT ENGINE

You may have experienced the frustration of a flight delay where you are stuck in a terminal with no information as to what has happened and when you may be on your way again. Unplanned maintenance is one of the leading causes of flight delays so being able to predict when an engine might require unscheduled maintenance benefits both airlines and their customers. Rolls-Royce created a program known as the "Blue Data Thread" that provides bidirectional

flow of information—a digital representation of the physical jet engine—between the company and the airlines operating their engines. These digital twins of aircraft engines, known as the "Intelligent Engine", are used to predict what maintenance will be required on which engines at what time, and ensure that repair capabilities and capacity will match these requirements. When engines have been sold for many years under long term maintenance contracts and the cost of engine maintenance is transferred back to the company, they are incentivized to optimize this process as much as possible.

1.4.3 Optimizing performance and operational efficiency

The increased visibility into the operations of plant and equipment that having a digital replica brings, together with the ability to predict and simulate the impact of changes, is a key factor in being able to improve operations across many different industries.

REAL WORLD EXAMPLE: THE NOKIA NETWORK DIGITAL TWIN

Nokia’s Network Digital Twin takes data about the performance of a communications network, including latency, throughput, signal strength and signal to noise ratio from telemetry sent directly from devices on the network and performs threshold monitoring on this representation of the state of the network against key performance indicators (KPIs). In the case of a threshold being breached, such as a change in topography reducing network coverage, network planners are alerted so that they can perform actions to ensure network connectivity is maintained.

1.4.4 Supporting the full infrastructure asset lifecycle

The preceding advantages coalesce in a single, evolving digital twin that supports physical infrastructure throughout its entire lifecycle. The twin starts as a planning and design tool, allowing engineers to simulate different scenarios, test performance, and optimize configurations before any physical work begins. As the asset moves into the construction phase, the twin transitions into a verification and tracking system, recording progress, validating installations against initial designs, and capturing "as-built" information. Finally, once the asset is operational, the digital twin becomes a real-time monitoring and optimization platform, continuously ingesting live sensor data to track performance, predict future maintenance needs, and enable adaptive, data-driven management for the rest of its service life.

REAL WORLD EXAMPLE: CROSS RIVER RAIL PROJECT

The cross river rail project in Brisbane Australia, was a multi billion dollar infrastructure project designed to remove bottlenecks in the existing public transport system. The project mandated a common data environment underpinning a digital twin, to which all contractors were obliged to contribute right from the planning stage, through construction, to operation. The benefits of the resulting digital twin included early identification of design issues, reconciliation across multiple major works packages, the ability to brief stakeholders effectively, and overcoming environmental issues by being able to travel through as-yet built environments, first-person and at scale.

1.4.5 Training and simulation

Modern digital twins may provide immersive, virtual reality environments where teams can participate in active learning (particularly safe practice in hazardous environments), which can lead to better learning outcomes at a lower cost. The original purpose of the lunar and command module simulators setup by NASA during the Apollo space program was to provide a replica of the physical system which astronauts performed extensive training and simulation in.

REAL WORLD EXAMPLE: ANATOMICAL DIGITAL TWIN

Researchers at Curtin University's medical school in Western Australia used photogrammetry to produce 3D reconstructions of human anatomical specimens. These 3D models, as a digital representation of internal organs, decrease the need for anatomy students to have access to costly laboratory facilities with increasingly scarce physical resources, where they may also be at risk of exposure to chemicals as they study physical specimens.

1.4.6 Digital twins across industries

modeling the real world in software so that you can observe, understand, perform experiments, and predict future outcomes without changing the physical system is valuable and has advantages that apply to just about any industry that exists today, but we find digital twins mostly being applied in some key industries.

MINING, ENERGY AND INDUSTRIAL

Mining, energy production, and steel and metal processing are all examples of industries that are characterized by large-scale, capital-intensive operations that use complex processes to extract and transform raw materials into the

essential components of our modern life. Within these industries, even small efficiency improvements to complex workflows and large operational costs can translate to significant financial, safety, and environmental outcomes. Digital twins contribute to process optimization in such industries by combining and contextualizing disparate process data and enabling the identification of opportunities to remove bottlenecks and improve throughput. Digital twins also see extensive use in the training of personnel in these industries to safely operate in hazardous environments.

AUTOMOTIVE

The automotive industry has been an enthusiastic adopter of digital twins across many parts of the industry for many years. Digital twins are used in the product development and engineering phase to simulate aerodynamic performance, reducing the need to build physical prototypes for testing. As the transition to electric vehicles gathers pace, with highly instrumented and connected vehicles with complex onboard software, digital twins are increasingly being used to maintain a digital representation of the state of each vehicle, as well as changing its state through software updates.

AGRICULTURE

As the United Nations predicts the global population to peak at around 10.3 billion people around 2080, up from approximately 8 billion people at the time of writing, agriculture and food production will need to become more efficient, improving yields and reducing waste. There are many ways in which complex agricultural processes can benefit from the living digital representation that a digital twin provides. Accurate and up to date data about plant, animal, and machine health allows farmers to more

efficiently allocate resources. Digital twins of global weather conditions, such as the Destination Earth digital twin being developed by the ECMWF, contribute to more accurate forecasting of weather events that are so important in food production.

NOTE

The United Nations provides extensive datasets related to global population here https://population.un.org/wpp/.

INFRASTRUCTURE

Digital twins are being deployed in critical infrastructure projects, from individual facilities to entire regional networks. Power systems use digital twins to balance supply and demand in real-time, integrating data from generation facilities, transmission lines, and smart meters to prevent outages. The Netherlands employs a digital twin of its flood protection infrastructure to simulate storm scenarios and optimize barrier operations protecting millions of residents below sea level. Digital twins enable infrastructure operators to orchestrate entire systems rather than just individual assets, ensuring the reliability and resilience that modern society depends upon.

SMART BUILDINGS, CITIES, AND STATES

Digital twins are being used across multiple scales in the built environment, from individual building systems to entire cities and states. Within individual buildings, digital twins are used to optimize energy consumption against the competing priorities of energy use and occupant comfort by integrating data from occupancy, weather, and other types of sensors.

Saudi Arabia, with an ambitious plan to house 9 million people in a footprint of 34 square kilometers, uses a digital twin to simulate the operation of lifts within high-rise buildings to optimize the design and construction process.

1.5 Building your first digital twin: key considerations and challenges

You might be wondering where you would even start building a digital twin. Whatever level of digital twin you are planning, the components of the system are as shown in figure 1.12.

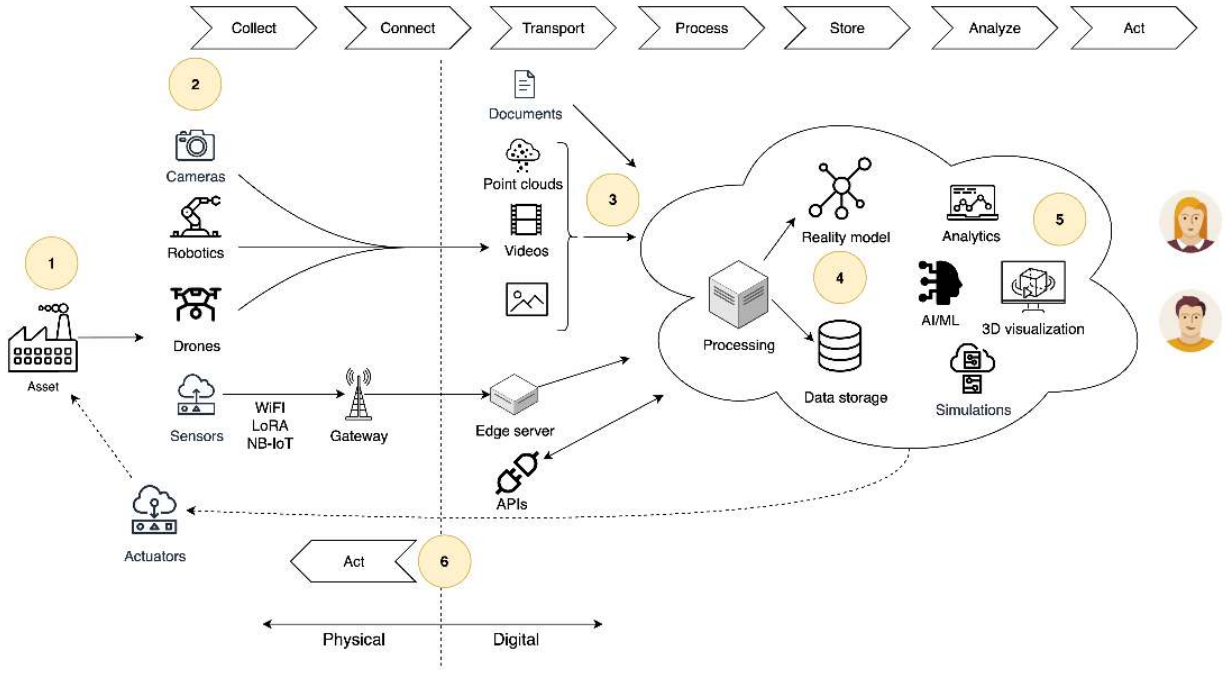


Figure 1.12 High level architecture of a digital twin showing how data about the real world is collected, stored, and processed to make decisions and affect outcomes.

This figure illustrates the complete digital twin data flow, showing how information moves from physical assets through various collection and processing stages to enable intelligent decision-making and automated actions. The

numbered elements trace the journey from initial data capture to final control actions.

1. Understand the object or system that you plan to build the twin of, and critically, what specific objectives you hope to achieve.
2. Determine what data you will need to collect to create a digital model that will support your desired objectives. Begin with static data sources, including documents and still images, before connecting sensors, cameras, and external systems via APIs.
3. Connect these data sources to your digital environment and transport the data into it.
4. Process data sources to build a model of the physical system within the digital environment. This model may include visual models, mathematical models that define your system, and a model of entities and their relationships persisted in a range of data stores.
5. Run analytics and simulations over the model.
6. Close the loop from the digital environment back into the physical system via some form of action.

Over the course of this book, we will explore each of these components in greater detail. We will work through an example of how you can build a functioning digital twin of perhaps your most important asset—your home—and see how building a digital representation can provide value to you in your everyday life. While a home digital twin might seem modest compared to industrial applications, the same architectural patterns, software components, and data flows apply directly to factories, power plants, and smart cities.

1.5.1 A clear definition of success

Before writing a single line of code or creating any rich interactive visualizations, you should be able to clearly articulate what you hope to achieve by building a digital twin. In early stages, this might mean establishing a systematic approach to data collection, identifying relationships between variables, and gaining a clearer understanding of system behavior. These exploratory outcomes provide the foundation for defining more specific targets.

As your digital twin matures, success should be defined through measurable outcomes that drive behavioral change, not just numerical outputs. Rather than vague aspirational priorities like "improving operations" or "reducing cost", articulate how the digital twin will change decisions and actions. For example, "reduce the London hotel’s electricity consumption by 10% within 6 months by enabling facility managers to optimize HVAC control based on real-time occupancy data" is a goal that combines quantifiable results with the behavioral shift that achieves them.

NOTE

Best practice when implementing digital twins is to start with a small pilot project rather than a full-scale implementation. This approach allows you to validate your ideas and gain stakeholder buy-in with minimal risk and investment upfront.

1.5.2 Data quality

It will come as no surprise that creating an effective digital representation of a physical object or system depends on having accurate, complete, and consistent data about the real world. Yet, although modern organizations are

reportedly drowning in data, poor data quality sabotages even the most sophisticated digital twins. As Herbert Simon observed, "a wealth of information creates a dearth of attention", and this abundance often masks underlying quality issues that can undermine digital twin effectiveness.

INCOMPLETE AND UNRELIABLE DATA

Unless you are designing and building your system from scratch, be it a new building, electric vehicle, or iron ore mine, you will likely be dealing with aged assets and infrastructure with varying levels of instrumentation and information available. Just getting operational data streams from your OT network into the IT space where your digital twin will be deployed can be a significant challenge, and the resulting data is often both incomplete and of questionable quality.

Data quality issues can arise from sensors drifting out of calibration over time, connectivity problems that create gaps in time-series data, and equipment failures introducing noise and anomalies. The data that is available could use different measurement standards, communication protocols, and sampling frequencies, that make it difficult to achieve the consistency and accuracy that effective digital twins require. Some of the data may be locked away behind vendor contracts that do not even give you access to data from your own equipment.

LACK OF DATA CONTEXT

Any data that you collect about the physical world has limited value in isolation, without appropriate contextualization that links it with other related data and entities. Consider a data stream from an IoT sensor that measures temperature. Without additional data about what

location or equipment the measurements relate to, it is impossible to make any decisions based on that data.

NOTE

Data contextualization refers to enriching raw data with additional relationships and information to characterize its situation, including relationships to what the data represents, the location of the data, and the type or classification.

Many organizations have historically struggled to match and integrate the multiple disparate data sources that they maintain. With multiple systems that use different identifiers for the same physical asset, customer or event, different terminologies, and different measurement scales, it becomes very difficult to analyze and derive insights to take action on.

1.5.3 Skills gap

Building or deploying a digital twin requires not only extensive technical skills across technologies, including device hardware, networking, software and data engineering, and data science but also deep knowledge of the domain and business in which you operate, in order to understand where a digital twin can bring value to the business. The importance of this combination of broad technical and domain expertise is critical to the digital twin project, delivering on the desired outcomes.

1.5.4 Build vs. buy

As with any new software system, one of the most important considerations is whether to build a custom system or

purchase an off-the-shelf solution. Some of the pros and cons of each approach are shown in table 1.1.

Table 1.1 The pros and cons of building your own solution versus buying off the shelf.

Option	Pros	Cons
Custom build	Ultimate flexibility; perfect alignment with unique business requirements; ensures complete data ownership	Requires extensive in-house technical expertise; takes longer to implement; demands ongoing maintenance
Buy off-the-shelf	Proven product; rapid deployment; reduced need for in-house development	Limited to existing features; requires adapting business processes to the software; difficult to differentiate your business; may compromise data sovereignty if operational data resides in vendor-controlled environments
Hybrid (build on PaaS / customization)	Faster time-to-market than full custom build; leverages existing framework for core functions; allows for critical customization to differentiate	Dependency on the platform vendor; customizations can be costly and create upgrade difficulties; requires a mix of in-house and vendor expertise

The decision is ultimately a strategic choice based on your short-term needs versus long-term goals, available technical expertise, how critical the digital twin will be to your organization, and your tolerance for external data dependencies. Given the breadth of technology involved, the most pragmatic approach is often a hybrid one of buying standard solutions for commodity components while building

custom components that offer competitive differentiation or require strict data control.

1.6 Summary

- Recent advances in IoT, cloud computing, and AI/ML have made the technology required to build a digital twin widely available.
- A descriptive digital twin provides a static digital representation of reality.
- An informative digital twin integrates data streams from the real world, regularly updating the digital representation.
- A predictive digital twin forecasts what the state of the physical system might be in the future, based on an understanding of the past.
- A comprehensive digital twin simulates possible future states of the physical system, using data assimilation to update mathematical models with data from the physical system.
- An autonomous digital twin closes the loop between the physical and digital realms by taking actions in the physical world based on analytics, predictions, or simulations in the digital representation.
- Before building a digital twin, you must be clear about the outcomes you are looking to achieve, what skills you will need, and whether you intend to build it from scratch, buy off the shelf capabilities, or a combination of the two.

2 Mapping physical systems to a digital representation

This chapter covers

- Deciding what aspects of the physical world to capture digitally
- Choosing information sources to build a digital model
- Extracting and digitizing information from these information sources
- Relating objects to each other spatially

Building a digital twin begins with the fundamental questions of what aspects of your physical world you should represent digitally, and how accurately you need to represent them. This decision shapes everything that follows, from the sensors you deploy and the data you collect, to the models you build and the insights you can extract. The challenge lies in translating the rich, multi-dimensional complexity of physical systems into digital representations that are both technically feasible and provide business value. Getting this translation right is important because it determines whether your digital twin will be a powerful tool for optimization and prediction, or an expensive collection of data that fails to deliver meaningful results.

This chapter addresses the first step in any digital twin implementation—understanding what to digitize and how you can map the physical world to a digital representation. The core decisions covered here establish the basis for the

technical implementations that enable effective data collection and system representation. Without a clear understanding of what you’re trying to represent digitally, even the most sophisticated digital representation of your physical system will fail to create an effective digital twin.

Before diving into the technical approaches for digital representation, you must first establish clear objectives that will guide every subsequent decision about what to digitize, how to represent it, and which details matter most for your specific use case.

2.1 Defining your objectives

One of the most common mistakes when building a digital twin is to start with the technology rather than the objectives or strategic goals. You may naturally get excited about creating high fidelity 3D visualizations, or building and deploying sensors before answering the fundamental question of 'what business problem are we trying to solve?'. This failure can lead to wastage and digital twins that do not deliver on the promised value.

The success of any digital twin initiative hinges on clearly defining your objectives before beginning the complex process of capturing and modeling physical systems. This objective-first approach serves as the foundation for all subsequent technical decisions, resource allocation, and validation criteria throughout the digital twin lifecycle. By aligning technical capabilities with business imperatives, you can avoid the common trap of building sophisticated digital models that fail to address real operational challenges or deliver measurable value.

2.1.1 Identify specific challenges and opportunities

Start by documenting concrete business problems that impact your operations rather than leading with available technologies. Imagine a plant superintendent facing 12 hours of unplanned downtime per month from centrifugal pump bearing failures, costing $15,000 per hour in lost production, (or $2.16 million annually). Current time-based, or periodic, maintenance replaces bearings every 6 months, meaning that:

- 40% of the bearings are replaced while they are still in good condition, costing $48,000.
- 15% of bearings fail before the scheduled replacement (causing the downtime).
- When a bearing fails, it can take up to 2 weeks to procure a new one from suppliers, so spare parts must be held in inventory.

The opportunity is to switch from planned to predictive maintenance, servicing equipment based on its actual health. Identifying potential issues early allows for just-in-time ordering of spares and maintenance scheduling, optimizing inventory and labor.

2.1.2 Establish measurable performance indicators and baselines

To illustrate how digital twins can predict and prevent unplanned downtime in a manufacturing plant, we can examine both leading indicators (early warning signals) and lagging indicators (business impact metrics) that work together to provide comprehensive operational insight as shown in table 2.1.

Table 2.1 Leading and lagging indicators defined for predictive maintenance of centrifugal pumps.

Indicator type	Description	Baseline	Target
Leading (Early warning)	Fluctuations in pump vibration and temperature (precursors to bearing failure)	Vibration: ±0.5 mm/s^2. Temp: ±1°C weekly	Fluctuation alerts at >2.0 mm/s^2 and >3°C weekly
Lagging (Business outcome)	Reduction in unplanned downtime and unnecessary replacements	12 hours/month downtime. 40% unnecessary replacement	3 hours/month downtime (75% reduction, $1.62 million saving) 60% reduction in unnecessary replacements ($29K saving)

2.1.3 Map decision-making improvements

Next, you should articulate which specific decisions will be enhanced and how actions taken based on those decisions will deliver business value. Document current decision-making processes, identify information gaps, and define how the digital twin provides better, faster, or more confident decisions, as shown in table 2.2 for the example of pump maintenance.

Table 2.2 An example of what current pump maintenance decisions will be improved and how.

Current decision	Improved decision via digital twin
Time-Based: If 6 months have passed, replace the bearing	**Predictive**: When trends indicate an 80% probability of failure within 4 weeks, order a bearing and schedule replacement in 3 weeks
Fixed Schedule: Maintain equipment based on a rigid schedule	**Risk-Based**: Optimize resource allocation by prioritizing maintenance on equipment with the highest risk of failure

2.1.4 Define a minimum viable digital representation

Based on decision-making requirements, determine the simplest digital representation that enables those improved decisions. Identify which physical attributes, behaviors, and relationships are essential and which can be simplified or omitted. For the pump example, the minimum viable digital representation requires capturing:

1. Vibration frequency spectrum (to detect bearing wear).
2. Temperature of bearing housings (to detect friction-induced heat).
3. Pump motor current signature (to detect mechanical load changes).
4. Operating hours and duty cycles (to understand usage).

This focused approach ensures every aspect of the digital representation serves a clear business purpose and can be validated against measurable results. Only after establishing these foundations should you proceed to capturing and modeling your physical systems.

2.2 A digital twin of the home

Let’s take the concepts introduced for defining your objectives, and create a practical worksheet that can be used to effectively map a physical system to its digital representation. To illustrate these concepts, consider the example of creating a digital twin of a home. For someone like myself, living in one of the driest cities in the world with moderately high electricity prices, the primary objectives might be to reduce water and electricity consumption. The following example works through these objectives based on the process defined earlier, starting with identifying opportunities together with the key data needed to enable better decisions, as shown in table 2.3.

Table 2.3 Start by defining your objectives in terms of the opportunities for improvement your digital twin will offer and what data is required.

Opportunity	Annual cost	Current decision	Better decision	Key data needed
More effective use of electricity produced by solar PV	$2,160	Run appliances when convenient	Schedule high-power loads during optimal solar production	Real-time electricity usage, solar PV output, appliance scheduling
Reduce excessive water usage by over-irrigation	$456	Water based on a fixed schedule	Irrigate based on soil moisture & weather forecast	Soil moisture sensors, rainfall forecast, and irrigation flow rates
More efficient cooling/heating decisions	$1,800	Adjust temperature for immediate comfort	Optimize timing with solar production and occupancy	Indoor/outdoor temperature, occupancy detection, HVAC energy usage
Mitigate risk of fire & flood	$50,000 potential	Only smoke detectors	Leak detection	Moisture, water flow measurement

These opportunities then lead to defining my metrics for success. For my home digital twin, these are the modest metrics listed in table 2.4. These metrics are the lagging indicators that I will measure to determine if I have met my objectives. Depending on the scope of your proposed digital twin, you may have significantly more metrics—the key is to understand what they are, the baseline measure, and your proposed target.

Table 2.4 Define measurable performance indicators and your current baseline.

Metric	Current value	Target value	Timeline
Electricity bill	$330/month peak	$300/month peak (10% reduction)	12 months
Water bill	$100/month average	$85/month average (15% reduction)	12 months

The leading indicators are the data points that I can measure on the way to achieving my objectives and are shown in table 2.5.

Table 2.5 Define the leading indicators that you will measure.

Indicator	Current method	Target method	Frequency
Energy & water use	Quarterly bills	Daily dashboard & alerts	Daily
Irrigation timing	Fixed schedule	Dynamic scheduling	Daily
Load scheduling	None	Peak solar/off-peak timing	Continuous
Leak detection	None	Leak monitoring	Continuous

2.2.1 Mapping my home to a digital representation

Now that I have defined my objectives and how I will measure these, together with the types of data that I will need to build a digital representation of my home, I start to

think about the minimum viable representation that will support my objectives.

I will need to gather and digitize data about electricity consumption, water usage, solar production, and occupancy to keep my digital model synchronized with changes in the physical system. My home is old, without any smart-home infrastructure and I don’t have any of this data available digitally yet, so I will need to digitize historical data (to understand the baseline), and improve how I gather this data in the future to make better decisions. Like most people, I find being able to place data in its spatial context helps me to understand it, and so I will need a model of my home that will show the data in context, as well as a floorplan to map out future sensor placement.

2.2.2 Start with what you have

I know that I will need to add a range of sensors to my home to gather the key data about the physical environment that will be required to support my objectives. But before I rush out and buy sensors, I can use a range of information sources that I already have to build my digital representation and to establish a digital baseline of the metrics of interest. This is just as true in many organizations looking to build a digital twin as it is in my example. The physical system you are building a twin of is likely already captured in many sources that you can use to build your digital representation. Lets look at what these sources of information are—not all of them will be relevant for my home digital twin, but as you look to larger and more complex systems, they will be increasingly important.

2.3 Information sources for digital representation

Most physical systems are extensively documented via drawings, maintenance logs, reports, and photos. This multi-modal documentation holds decades of operational knowledge, forming the foundation of current asset management.

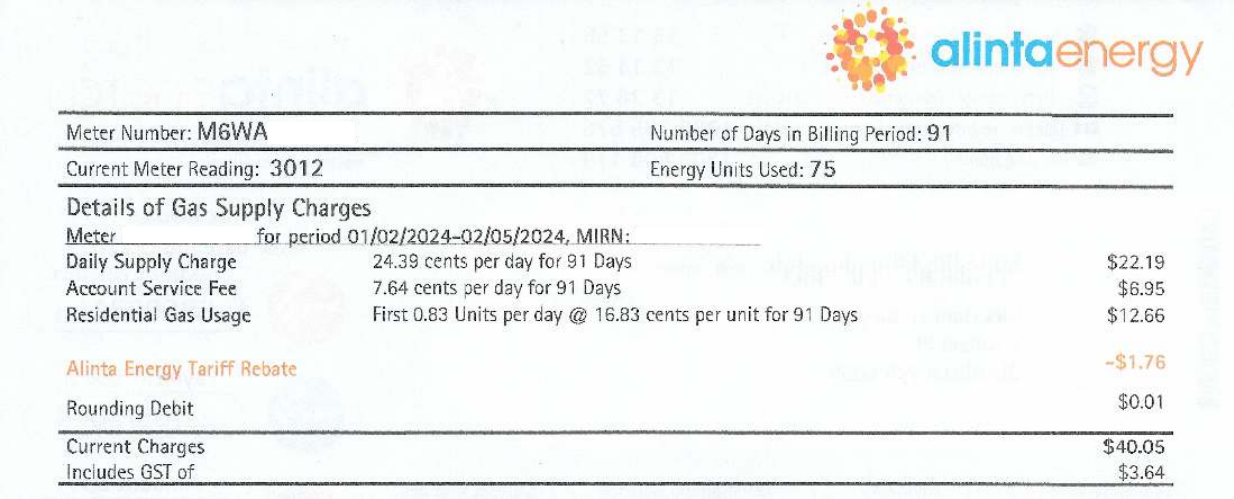
However, much of this information is in formats digital twin platforms can't directly process, for example scanned PDFs, paper records, and legacy files. Transforming these diverse sources into machine-readable formats is an important first step in building comprehensive digital representations without starting from scratch.

2.3.1 Historical records

Text-based documents capturing a system's state over time are essential for digital representation, providing data for change analysis and machine learning training. Examples include:

- A maintenance record that captures work that has been performed on a piece of equipment at some time in the past.
- An incident report or failure record that provides information about system vulnerabilities, failure modes, and recovery procedures.
- A work order that records work to be scheduled in the future.
- A utility bill, as illustrated in figure 2.1 that records a meter reading at a point in time.

Digitizing historical records is important as it allows you to understand how physical systems have performed over time, capture changes made to the physical system, and, importantly, provide the training data necessary for machine learning algorithms.



alintaenergy

Meter Number: M6WA	Number of Days in Billing Period: 91	
Current Meter Reading: 3012	Energy Units Used: 75	
Details of Gas Supply Charges		
Meter for period 01/02/2024–02/05/2024, MIRN:		
Daily Supply Charge	24.39 cents per day for 91 Days	$22.19
Account Service Fee	7.64 cents per day for 91 Days	$6.95
Residential Gas Usage	First 0.83 Units per day @ 16.83 cents per unit for 91 Days	$12.66
Alinta Energy Tariff Rebate		-$1.76
Rounding Debit		$0.01
Current Charges		$40.05
Includes GST of		$3.64

Figure 2.1 A sample gas bill as generated quarterly by my residential gas provider. The current meter reading is provided for a stated date.

TRY IT OUT: UTILITY BILL ANALYSIS

To reduce my home water and energy consumption, I must first establish the current utility usage baseline. A predictive digital twin requires observing the past, understanding the present, and forecasting the future.

Home utilities typically rely on periodic manual meter readings (quarterly for gas/water in this example) or low-resolution digital data (daily for electricity). This fragmented approach creates challenges for unified analysis:

- *Swivel chair integration* - data is scattered across separate utility web portals, lacking a unified digital view.
- *Data format inconsistency* - historical data exists across varying formats (APIs, downloadable CSVs, and non-machine-readable PDF bills).
- *Infrequent updates* - quarterly billing prevents timely behavior modification.

To understand the 10-year consumption baseline (e.g., seasonal usage patterns) and feed historical data to machine learning models for prediction (e.g., testing the impact of weather and solar production), this locked PDF data must be digitized.

The solution is to use *optical character recognition (OCR)*, specifically an open-source library like Tesseract, to extract meter readings and dates from historical PDF bills. This converts the information into a historized, machine-readable format—although not perfect, OCR provides a valuable mechanism for digitizing paper records. The sample code in listing 2.1 uses Tesseract to extract the date and meter reading from the PDF bill shown in figure 2.1. Before running this code sample, ensure you have installed Tesseract (available from https://tesseract-ocr.github.io/tessdoc/Installation.html) on your system and that it is available in your path.

NOTE

All code samples are available in the book's GitHub repository here https://github.com/digital-twins-in-action, together with instructions on how to set up your Python environment to run them.

Listing 2.1 Extracting the meter reading from a utility bill using OCR to digitize consumption data

```
from PIL import Image
import pytesseract, re

def extract_meter_reading(text):
  pattern = r"Current Meter Reading:\s*(\d+)" #1
  match = re.search(pattern, text)
  return int(match.group(1)) if match else None

def extract_date(text):
  pattern = r"(\d{1,2}/\d{1,2}/\d{4})" #2
  matches = re.findall(pattern, text)
  return matches[-1] if matches else None

def read_bill():
  image = Image.open(r"./images/gasBill.png")
  text = pytesseract.image_to_string(image)

  meter_reading = extract_meter_reading(text)
  date = extract_date(text)

  return {
    "meter_reading": meter_reading,
    "date": date
  }
```

#1 A regular expression to extract the meter reading based on the format of the bill.
#2 There are multiple dates in this bill. I want the last date that appears.

Once I have configured the regular expression in the code to extract the data I require (date and meter reading), I can then run this code against the decade worth of PDF bills I have been emailed by my utility providers. The output of this is shown in figure 2.2, which captures my household energy consumption for the past 10 years. I do the same with my water bills so that I have a baseline of consumption before I build my digital twin.

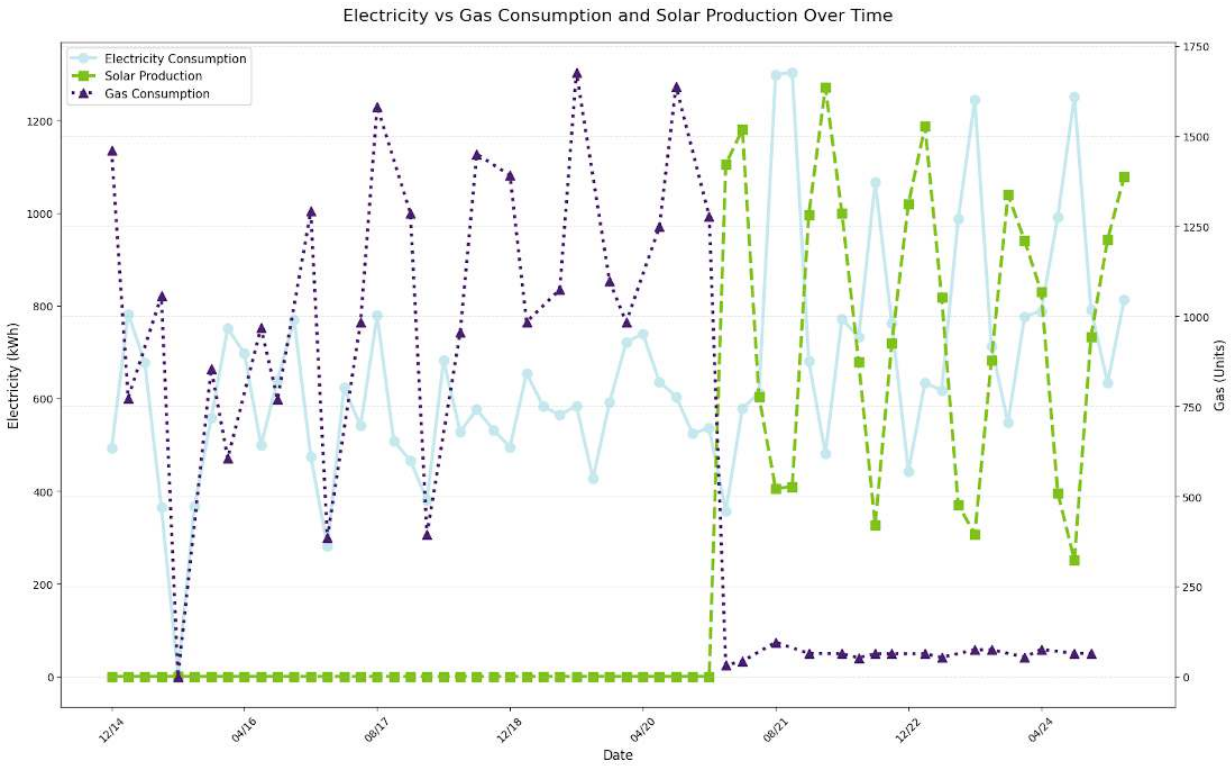


Figure 2.2 Ten years' worth of energy consumption data extracted from PDF gas and electricity bills by OCR.

Having converted all this data from PDF bills into a machine readable format, I can immediately gain insights from the data including seasonal variation on consumption, and the change in consumption when transitioning from a gas hot water system to solar hot water, as well as electricity production from solar panels installed around 2020. I will use this digitized energy data in my digital twin, but will combine it with higher resolution data from sensors moving forward.

2.3.2 Photographs

Photographs are accessible, powerful data sources, often created using just a smartphone. They provide a vital visual record and temporal sequence of changes and many organizations may already maintain extensive photographic records which can be used to reconstruct timelines of

physical changes or correlate visual documentation with operational events. Photographs offer:

1. *Contextual documentation* - a visual record of the physical state at a point in time, useful for tracking degradation or construction progress.
2. *Source of structured data* computer vision can automatically extract information like equipment IDs, valve positions, gauge readings, and corrosion levels, digitizing operational state data.
3. *Metadata* - contains useful information like location, position, and camera pose.

TRY IT OUT: EXTRACT STRUCTURED DATA FROM A PHOTOGRAPH

In the context of building my home digital twin, photographs are an important source of data related to utility consumption. Since utility bills are only generated quarterly, I do not have real time consumption data to inform decisions and actions. I can take photographs of utility meters and then use computer vision to extract data from the analog water and gas meters into a machine readable format and update the state within the digital twin of my home at a much higher frequency than digitizing quarterly bills. This is demonstrated in figure 2.3 using a photo of my water meter taken with a smartphone, and the same library, Tesseract, as I used to extract text from the PDF bill.

Figure 2.3 A photograph of my residential water meter (1), with the area of interest extracted and converted to grayscale in (2) before being inverted in (3) to enable OCR extraction of the digits.

The code, shown in listing 2.2, is slightly more complex due to the need to crop the area in the image where the meter digits are and preprocess the image to get an optimal result. Figure 2.3 shows the progression of the image through the extraction process. This code will extract the number `42678` from the photograph, which can then be associated with the date the photo was taken and it's location, and stored in a database.

Listing 2.2 Example code to extract a meter reading from an image of an analog meter, and digitize it using OCR.

```
import cv2, pytesseract
import numpy as np
from PIL import Image, ImageOps

def extract_meter_reading(image_path, display_results=True):
  img = Image.open(image_path)
  crop_box = (1000, 890, 1700, 1030) #1
  cropped_img = img.crop(crop_box) #2
  gray_img = ImageOps.grayscale(cropped_img) #3
  inverted_img = ImageOps.invert(gray_img) #4
  meter_reading = pytesseract.image_to_string( #5
    inverted_img,
    config='--psm 7 -c tessedit_char_whitelist=0123456789'
  )
  meter_reading = meter_reading.strip() #6

  return meter_reading
```

#1 This is the bounding box of the digits in the image.
#2 Crop the image to the digit bounding box.
#3 Convert the cropped image to grayscale.
#4 Invert the grayscale image. This often helps OCR for dark digits on a light background (like the meter reading).
#5 Use OCR to extract text. Treat the image as a single line of text (often good for meter readings) and restrict OCR to only digits.
#6 Clean up the extracted text (remove whitespace, newlines).

You may wonder if it is worth going to the trouble of taking a photograph of the meter daily and running the image through this process to extract the reading. Why not simply enter the reading into a computer to digitize it? By placing an IoT camera over the analog meter, I can use OCR on the photographs it takes to fully automate taking a daily meter reading.

2.3.3 Videos

Static capture methods provide snapshots, but video streams introduce continuous temporal monitoring, updating

the digital representation of your physical system in near real-time. This captures the dynamic nature of operational environments.

Video data can be integrated into a digital twin using three main strategies:

CONTINUOUS STREAMING

The simplest, but most costly method. The entire video feed is continuously transmitted to the digital twin for real-time monitoring and storage. This demands high bandwidth and storage resources.

CHANGE DETECTION

A more efficient variation where data is only sent or stored when a relevant physical change (e.g., movement or new appearance) is detected in the video stream. This is common in security applications where only event-based fragments are transmitted.

EDGE PROCESSING WITH SELECTIVE TRANSMISSION

The most resource-efficient method. Raw video streams consume massive bandwidth when transmitted to remote servers or the cloud. Instead, video data is processed locally at the edge (close to the physical asset). Only derived, processed data (for example, 'vehicle detected', 'occupancy count is 3') is selectively transmitted, drastically reducing bandwidth and improving efficiency.

NOTE

At the edge refers to processing data and running analytics close to where the physical asset or data is located, rather than transporting the data to a centralized data center or the cloud. It offers benefits in terms of reduced latency, bandwidth efficiency, and the ability to respond to data quickly.

TRY IT OUT: DETECT MULTIPLE OBJECTS IN A VIDEO FEED

Computer vision detects the position and type of objects within video frames, assigning bounding boxes and confidence scores. This allows digital twins to represent the location and status of physical objects like people, vehicles, or packages.

By running object detection at the edge, only the processed information (e.g., 'vehicle detected at entrance') is transmitted, not the full, high-bandwidth video stream.

You Only Look Once (YOLO - https://www.v7labs.com/blog/yolo-object-detection) is a popular, high-speed convolutional neural network model that predicts multiple object bounding boxes in a single pass. You can try the YOLO version 8 model out yourself with the code shown in listing 2.3, run on a computer equipped with a webcam. Whereas this code simply overlays the video feed with the detected objects, in a complete application, you would transmit the detected object events to your digital twin application (at a much lower cost than transmitting the full video feed).

Listing 2.3 Use the YOLO model to demonstrate multiple object detection within a video stream.

```
import cv2
import numpy as np
from ultralytics import YOLO

def run_webcam_detection():
  model = YOLO("yolov8n.pt") #1
  cap = cv2.VideoCapture(0) #2
  cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640) #3
  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

  while True:
    ret, frame = cap.read() #4
    results = model(frame) #5
    annotated_frame = results[0].plot() #6
    cv2.imshow("YOLOv8 Detection", annotated_frame) #7

    if cv2.waitKey(1) & 0xFF == ord('q'):
      break

  cap.release()
  cv2.destroyAllWindows()
```

#1 Use the YOLOv8 nano model, which is optimized for speed and resource efficiency.
#2 Open the webcam.
#3 Set the webcam properties.
#4 Continuously read a frame from the webcam.
#5 Run YOLOv8 inference on the frame.
#6 Visualize the results on the frame.
#7 Display the annotated frame.

Figure 2.4 demonstrates the use of the YOLO model to detect people within a room, which can be used to determine occupancy within a room by detecting people within the frame.

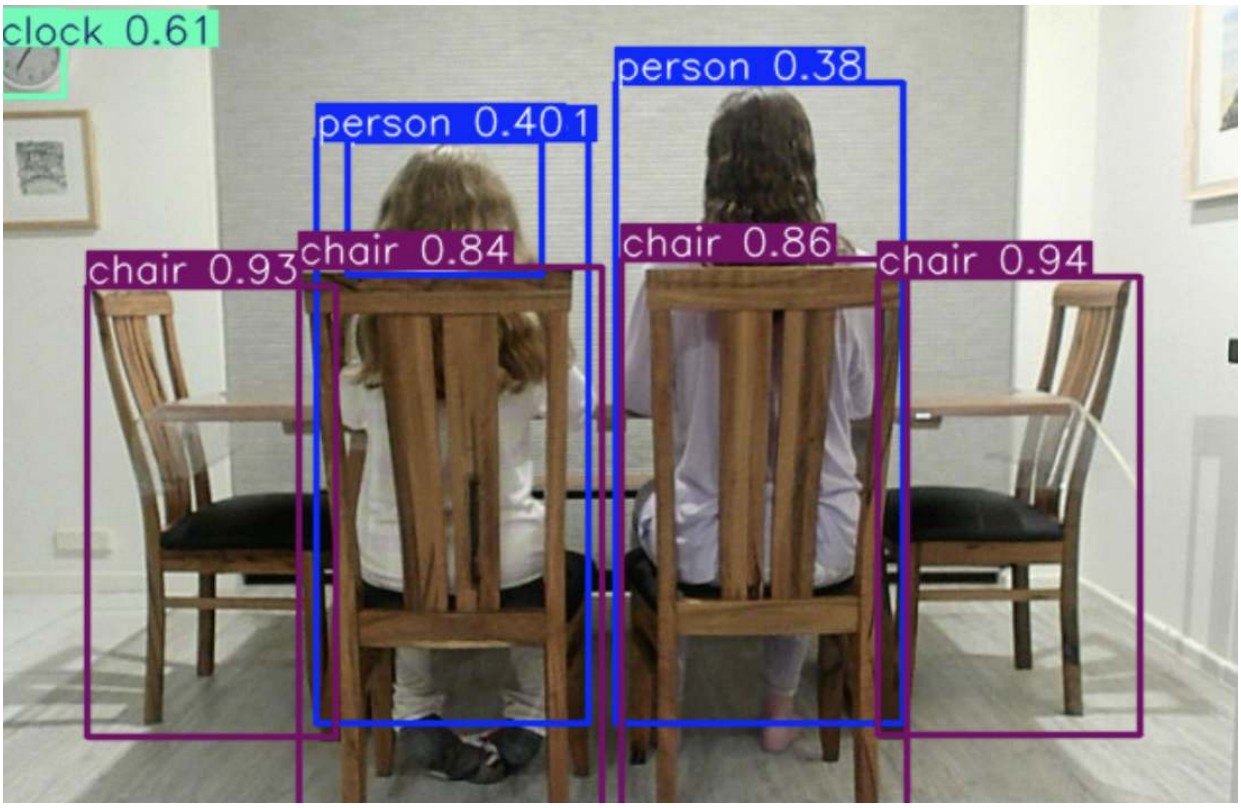


Figure 2.4 An example of using a multi object-detection computer vision model over a video stream using the YOLO v8 model to determine room occupancy. The model draws a bounding box around each object it detects in the stream, and adds a label to the box with a confidence score, which is a combined percentage representing the model's certainty that a detected object belongs to a specific class and that the bounding box contains an object.

In my home digital twin, I can make use of edge object detection in a video stream to detect occupancy levels in the home, and use this data to make decisions about energy or water use.

While the use of video data to capture physical systems for digital representation requires careful consideration of architecture, privacy, and data management, the resulting value extends far beyond simple visualization, creating digital twins that truly mirror the living, changing nature of physical systems. As computational capabilities and computer vision algorithms continue to advance, video will increasingly serve as a foundational data source for digital

twins across industries, enabling levels of fidelity and responsiveness previously unattainable.

2.3.4 Engineering documents

2D drawings remain the foundational language of engineering, construction, and operations across many industries due to their ability to reduce complexity through the use of visual abstractions that convey large amounts of information through standardized symbols and conventions.

An example of this is the fragment of a *process flow diagram (PFD)* shown in figure 2.5. A PFD is commonly found in chemical or process engineering and visually maps the sequence of actions and flow of materials in a process. Through the use of standardized symbols representing a pressure vessel, valve, condenser and heater, together with conventions regarding what the lines in the diagram represent, this compact document conveys a significant amount of information to a skilled process engineer. This diagram represents the flow through a distillation column separating a mixture of pentane and hexane (the feed) into streams of pentane (distillate) and hexane (bottoms), using feedback composition control. It shows how these components are physically connected and how gases and liquids flow through the process.

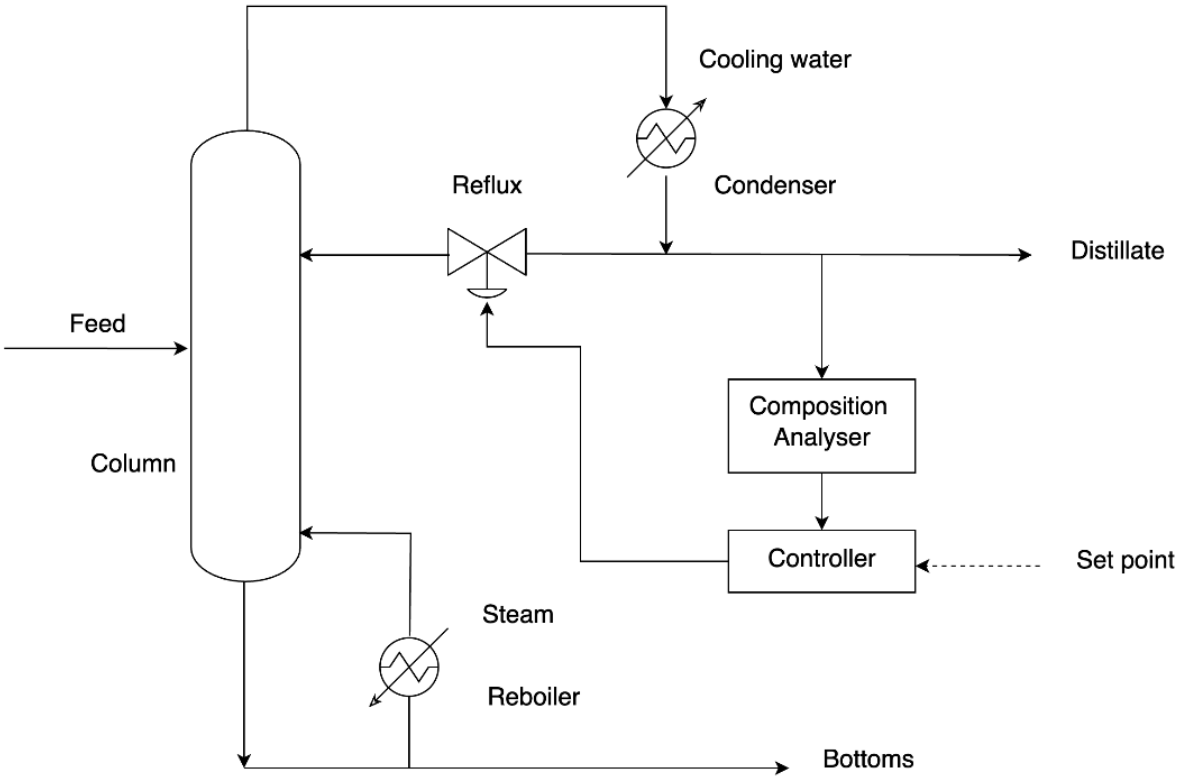


Figure 2.5 A simple example of a process flow diagram (PFD) showing the components of a chemical process using standard symbols to represent physical components and their relationships.

Another type of engineering document that is important within many industrial contexts is a *piping and instrumentation diagram (P&ID)*. These documents represent the engineering view of an industrial process (such as an oil refinery), showing the complex relationships between equipment, piping, and instrumentation, creating a comprehensive schematic blueprint of the physical process, again using standardized symbols to represent the physical arrangement of components and the functional relationships between them. A simple example of a P&ID is shown in figure 2.6 and represents a tank and a pump and the connections between them, including representation of the instruments that measure aspects of this process, including the level of liquid in the tank and the pressure within the pipe from the pump.

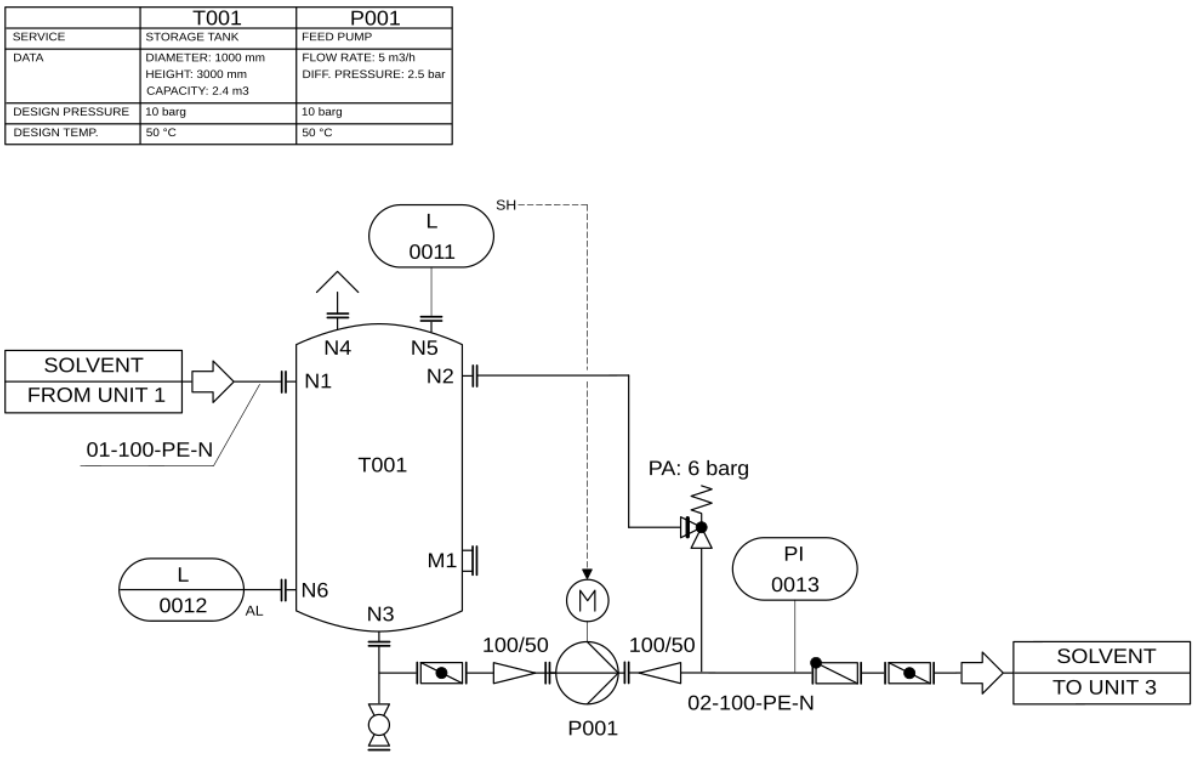


Figure 2.6 A sample piping and instrumentation (PID) diagram showing how complex physical systems can be logically represented in a diagram. Source Wikipedia. Licensed under CC BY-SA 3.0

Both PFD and P&ID documents are critical document based representations of a physical system that are important to consider when looking to capture that system (in this case, a complex industrial process) and represent it digitally. Their importance is due to:

- As documents that define, in great detail, the physical structure and relationships that make up the physical system they can be used to create an accurate digital model of the physical system.
- As documents related to the physical system, a digital twin can contextualize them, linking and relating them to other pieces of information within the organization. This assists in information retrieval and an associated

reduction in the time taken to make high quality decisions.

But as raster graphics (as the PFD in figure 2.5 and the P&ID in figure 2.6 are printed here), a computer cannot understand the information contained in these documents without some further processing. Luckily, computer vision, and more recently, vision capable *large language models (LLMs)* can help to digitize these documents into a structured format that can be used.

TRY IT OUT: EXTRACT INFORMATION FROM AN ENGINEERING DOCUMENT

P&IDs, as shown in Figure 2.6, often exist as legacy documents, either in paper form or scanned PDF documents with multiple revisions, annotations, and different symbols. The diagrams themselves are designed for human, rather than machine, interpretation and contain important information related to industrial sensor locations and identifiers, control logic, and topological information. Digitizing these documents and transforming the information they contain into structured information that can be processed by machines is critical in any digital twin of an industrial process.

LLMs with multi-modal understanding can be used to analyze engineering documents and transform them into machine readable representations that can further be used to create a digital model of the process defined in the diagram.

Listing 2.4 uses Anthropic's Claude Sonnet 4 model and a simple prompt to convert the sample P&ID diagram shown in figure 2.6, to a structured JSON format. Fine-tuning the prompt allows you to customize the structure of the generated JSON. To run this code, you will need an API key

from Anthropic, which you can get here: https://www.anthropic.com/api. The sample P&ID is provided as an image in the books GitHub repository.

Listing 2.4 Transform a PID document to JSON representation using Claude Sonnet 4

```
import base64, requests

def analyze_image(api_key, image_path, prompt):
  with open(image_path, "rb") as f:
    image_data = base64.b64encode(f.read()).decode('utf-8')

  response = requests.post(
    "https://api.anthropic.com/v1/messages",
    headers={
      "x-api-key": api_key,
      "anthropic-version": "2023-06-01",
      "content-type": "application/json"
    },
    json={
      "model": "claude-sonnet-4-20250514",
      "max_tokens": 5000, #1
      "messages": [{
        "role": "user",
        "content": [
          {"type": "text", "text": prompt},
          {"type": "image", "source": {
            "type": "base64",
            "media_type": "image/png",
            "data": image_data
          }}
        ]
      }]
    }
  )
  return response.json()["content"][0]["text"]

if __name__ == "__main__":
  api_key = "sk_***"

  result = analyze_image(
    api_key,
    "images/pid.png",
    "Convert this diagram to JSON" #2
  )
```

#1 This parameter defines the maximum number of tokens the model should generate before stopping. Customize this to the number of tokens you need.
#2 You can fine tune the prompt here to generate output in a different form.

The model is able to extract information from the visual diagram and represent it as structured JSON, as shown in listing 2.5. This structured data can be used to create a digital model of this process in a knowledge graph, as we will see in Chapter 5.

Listing 2.5 Structured JSON representation of the P&SID returned by Claude (truncated for brevity). The full output is available on GitHub.

```
{
  "equipment": [
    {
      "id": "T001",
      "type": "STORAGE_TANK",
      "service": "Storage Tank",
      "data": {
        "diameter": "1000 mm",
        "height": "3000 mm",
        "capacity": "2.4 m3"
      },
      "design_conditions": {
        "pressure": "10 barg",
        "temperature": "50 °C"
      },
      "connections": [
        {
          "id": "N1",
          "type": "inlet",
          "line": "01-100-PE-N",
          "source": "UNIT 1",
          "description": "Solvent inlet from Unit 1"
        }
      ]
    }
  ],
  "instruments": [],
  "lines": [],
  "control_systems": []
}
```

2.3.5 External systems

External systems provide context that is important when building a digital representation of a physical system. In many industries, systems external to your digital twin (but internal to your organization), such as *enterprise resource planning (ERP)* and SCADA systems, will contain information about your equipment, its maintenance history, performance

and so on. Some external systems may offer *application programming interface (API)* access, making integration technically straightforward via RESTful APIs, GraphQL endpoints, and webhook subscriptions that provide real-time or near-real-time data feeds, while others may need significant effort to access the data they hold.

In my home digital twin, information that can be used to achieve the objectives is available from several external systems. I can access weather forecast data generated from the European Centre for Medium-Range Weather Forecasts (ECMWF) models that we saw in chapter 1, which can be used to predict rainfall and automatically adjust the irrigation system. My electricity provider provides daily electricity usage data via their website that can be used to inform the system more regularly than a quarterly bill. This contextual information is what I will use to move from reactive problem solving towards predictive optimization.

TRY IT OUT: GET FORECAST RAINFALL FROM THE ECMWF

A subset of ECMWF real-time forecast data is made available to the public and can be accessed for free up to ten times a day from the Open-Meteo service (https://open-meteo.com). The following URL, when pasted into a browser, will return the rainfall forecast over the next 7 days from the ECMWF's Integrated Forecasting System (IFS) for the city of Perth.

```
https://api.open-meteo.com/v1/forecast?latitude=-31.9522&longitude=1
15.
➥8614&hourly=rain&models=ecmwf_ifs025&forecast_days=7
```

I will integrate a daily call to this API by my digital twin to provide information that I can use to make dynamic

decisions about whether to irrigate the garden or not.

2.4 Spatial and geometric representations

Think back to the first time you heard the term 'digital twin'. Chances are, you pictured a rich 3D visual model that closely represents reality. There is a good reason for this: many modern digital twins incorporate 3D models of the physical system they mirror, for a number of reasons

- A 3D model provides an immersive spatial context that allows people to visualize a complex system as it actually exists in the physical world, aiding in their understanding of the size, shape, location, and relationship of physical objects.
- Overlaying a 3D model of a built asset over that of the design provides a powerful mechanism to see where the built asset differs from the design view.
- The models can be used to simulate physical phenomena in the real world, such as the flow of air over a newly designed vehicle.
- They are well suited to the contextualization of data received about the physical environment via sensors, allowing it to be spatially located, which then aids in the consumption and understanding of the data.

In addition to the immersive realism experience offered by 3D visualizations, 2D geometric representations that show layouts and measurements are no less important when capturing a digital representation of reality. Just like engineering documents and drawings, 2D geometric representations of physical spaces often employ domain-specific visual abstractions that clearly convey information

about the physical world without the complexity that a 3D spatially accurate model may bring.

2.4.1 2D geometric models

Long before computer graphics enabled photorealistic 3D models, maps, floorplans, and blueprints allowed engineers, architects, planners, and operators to understand, design, and manage complex spatial relationships in the real world. In digital twin implementations, these traditional formats retain their importance for the reasons that they are information dense, universally understood representations that are, in many cases, already integrated with existing workflows.

MAPS

As we saw in Chapter 1, the 2D geometric representation of a physical area that is provided by a geographical map forms the basis of many digital twins. Maps are spatial reference frameworks that provide geographic context, location relationships and environmental understanding that many people are familiar with, having been taught to use them from a young age.

Soil maps, geological surveys, and environmental hazard zones documented in specialized maps provide critical information for understanding foundation conditions, seismic risks, and environmental constraints that affect system design and operation. This contextual information is essential for creating digital twins that accurately represent how physical systems interact with their environmental setting.

Geographic Information Systems (GIS) serve as the authoritative foundation for spatially-referenced digital

twins, providing standardized frameworks for capturing, storing, analyzing, and visualizing geospatial data with precise coordinate reference systems and established accuracy standards.

FLOORPLANS

Floorplans instantly convey room count, size, and layout using simple, scaled diagrams and standard architectural symbols. While 3D models contain the same information, floorplans deliver it more efficiently, providing a clear, uncluttered view of spatial logic.

These precise measurements enable floorplans to serve as powerful computational tools. Digital formats allow path-finding algorithms (like A* and Dijkstra's) to calculate optimal routes between any two points. This is crucial for comprehensive digital twins that simulate movement, such as tracking people in an office or robots in a warehouse.

For large-scale implementations, 2D floorplans provide the essential foundation, serving two dual purposes:

- *Instrumentation mapping* - laying out sensor and equipment locations.
- *Navigation* - creating intuitive interfaces that allow users to zoom into detailed 3D views of specific areas.

The enduring value of a floorplan lies in its clarity, which translates directly into both computational efficiency and user understanding, making it indispensable for digitally navigating and representing physical spaces. Figure 2.7 shows a floorplan that I have created of my home that I will be using in my digital twin as the basis of a digital representation of the physical environment.

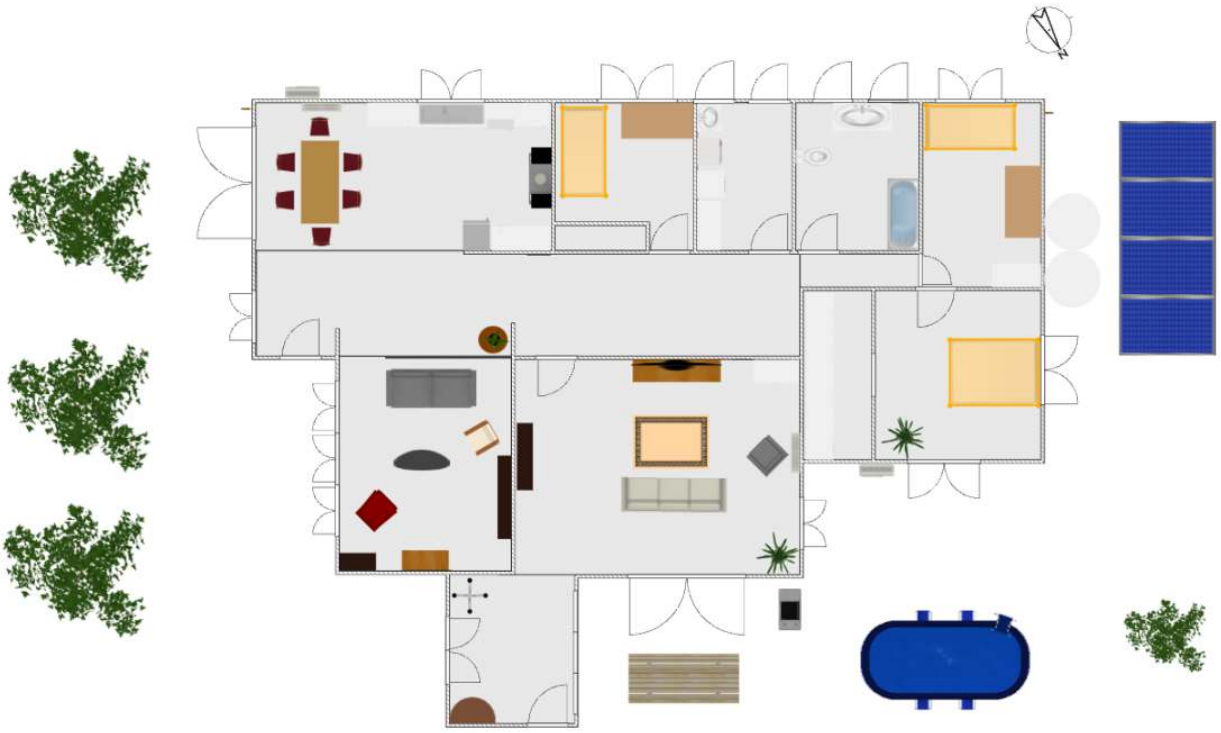


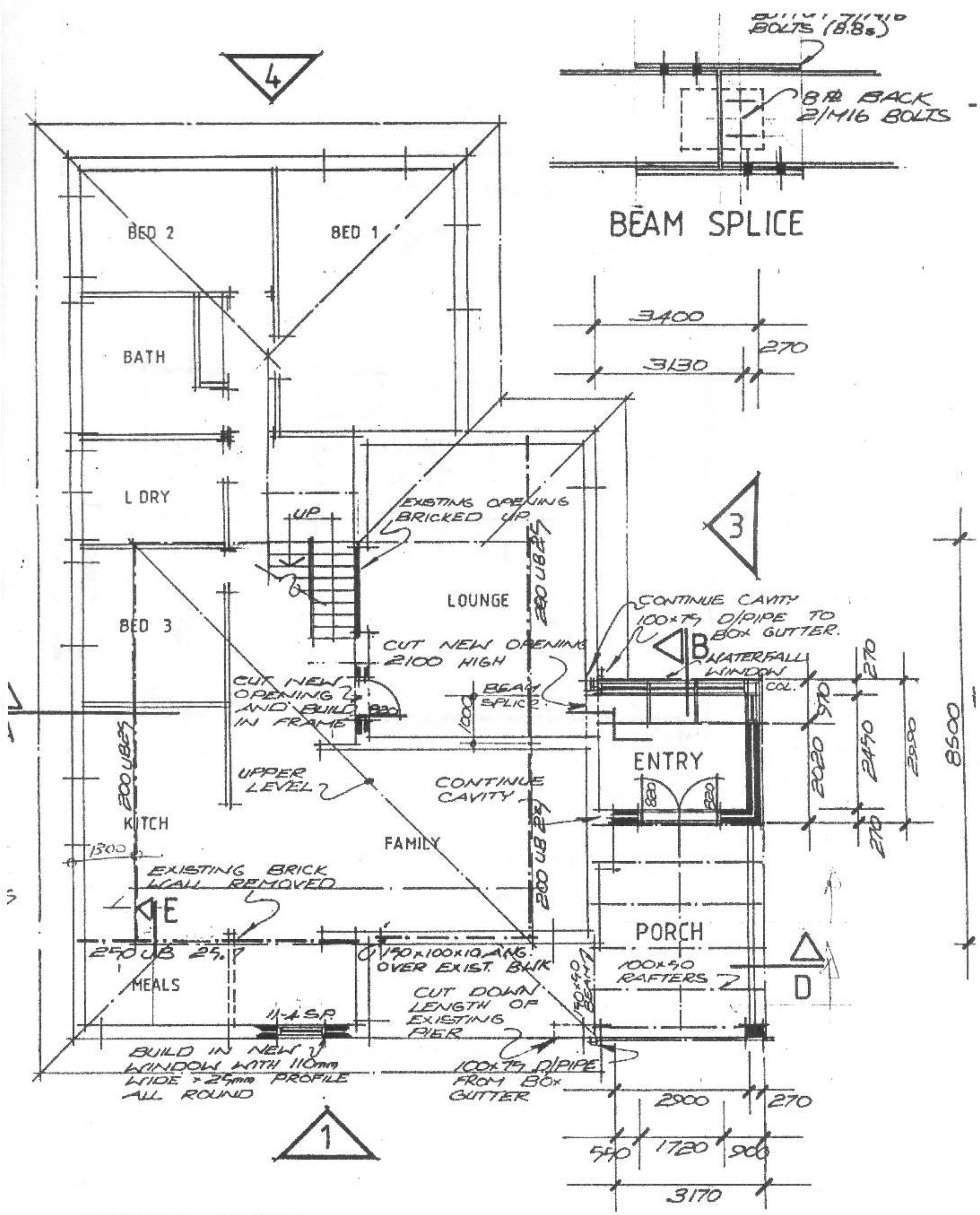
Figure 2.7 A floorplan of my home that I will use in my digital twin to map sensor placement, and as the basis to build a 3D model of the building.

BLUEPRINTS AND ARCHITECTURAL DRAWINGS

Blueprints are specialized, detailed geometric drawings used by architects and engineers to ensure a building is constructed to specific standards. They contain significantly more detail than a floorplan, including millimeter-accurate dimensions, material lists, and construction specifications, detail that can be difficult to achieve through simple scanning.

A digital twin uses blueprints to create a precise digital model of a physical structure, enabling instant comprehension of spatial relationships, adjacencies, and exact dimensions. This is crucial for twins requiring high-accuracy spatial modeling. Alternatively, the digital twin can provide a spatial or temporal link to a digital blueprint repository, offering quick access to the source

documentation when needed. Figure 2.8 shows a blueprint of my home that I will use to build a digital model of my home to be used in the digital twin.



GROUND FLOOR

1:100

Figure 2.8 An example of a blueprint of my home, showing dimensions and material specifications that are an important source of spatial information when modelling physical structures digitally.

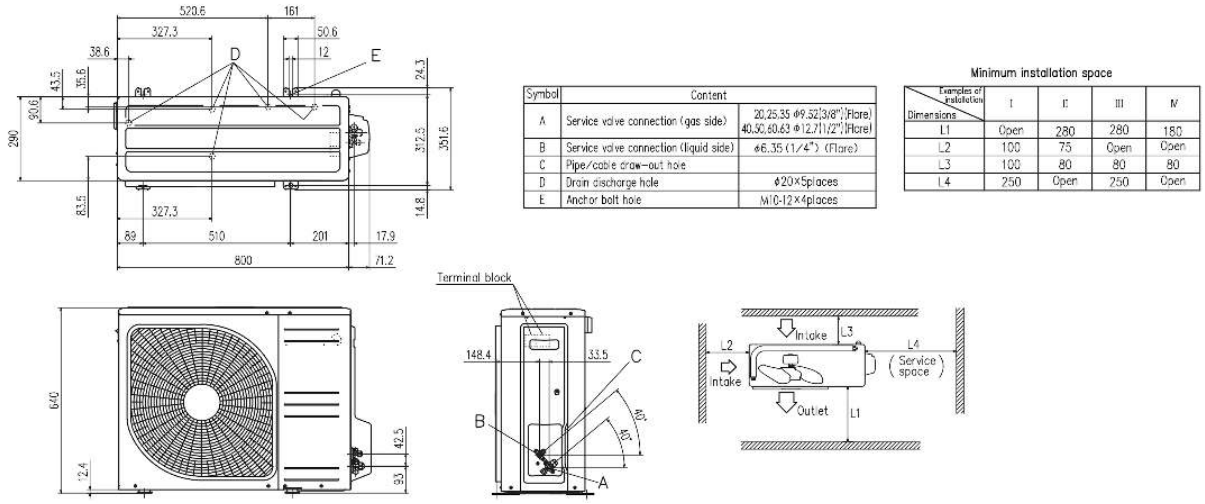
ENGINEERING DRAWINGS

Engineering drawings are the definitive expression of engineering intent, providing a universal standard that uses symbols, conventions, and annotations to define an object's design in precise detail. They provide essential information throughout the object's lifecycle.

The typical layered structure of these drawings supports selective information extraction. Layers for dimensions, annotations, and geometric features can be processed independently to isolate data specific to modeling needs.

Crucially, drawing revision systems provide historical tracking of design changes. Understanding this design evolution ensures that the digital representation matches the system's current configuration, accounting for field modifications or design improvements.

The engineering drawing in figure 2.9 shows the geometry and dimensions of an air conditioning unit I have installed in my home. The drawing conveys the geometry and dimensions of the physical object in a multi-view projection. This information can be used both to plan the placement of the physical device as well as to create an accurate 3D model of the device to be placed in a 3D visualization.



Symbol	Content	
A	Service valve connection (gas side)	20,25.35 ø9.52(3/8")(Flare) 40,50,60.63 ø12.7(1/2")(Flare)
B	Service valve connection (liquid side)	ø6.35 (1/4") (Flare)
C	Pipe/cable draw-out hole	
D	Drain discharge hole	ø20×5places
E	Anchor bolt hole	M10-12×4places

Minimum installation space

Examples of installation / Dimensions	I	II	III	IV
L1	Open	280	280	180
L2	100	75	Open	Open
L3	100	80	80	80
L4	250	Open	250	Open

Figure 2.9 An engineering drawing of an external reverse cycle air conditioning unit. © Mitsubishi Heavy Industries.

2.4.2 3D geometric models

While 2D models convey layout logic, many digital twins require the depth and detail provided by 3D geometric models. These models form the visual and spatial backbone of digital twin applications, transforming abstract data into intuitive, interactive representations.

The core value of 3D models is that they:

1. Enable exploration of complex systems from multiple perspectives.
2. Show data within a spatial context, making it meaningful and actionable.
3. Support collaborative design before construction.

Applications require different 3D approaches, each offering distinct advantages: mesh-based models prioritize performance, parametric designs capture engineering intent, and point clouds preserve precise measurements.

BUILDING INFORMATION MODELS

A *building information model (BIM)* is a 3D model of a building or infrastructure asset that also contains rich information about the building's components, materials, and performance characteristics. BIM models typically contain detailed information about structural elements, mechanical systems, electrical networks, plumbing infrastructure, and architectural features, along with their material properties, performance specifications, and maintenance requirements. This integrated approach enables architects, engineers, and facility managers to analyze building performance, simulate different scenarios, and coordinate complex construction and renovation projects with unprecedented accuracy.

The *industry foundation classes (IFC)* are an ISO standard (https://www.iso.org/standard/84123.html) that serves as the primary open file format for exchanging BIM data. IFC files contain not only the geometric representation of building elements but also their semantic meaning—a wall object understands that it's a wall, knows its thermal properties, structural capacity, and relationships to adjacent spaces and systems.

You can navigate to https://3dviewer.net/#model=assets/models/haus.ifc to view a simple BIM model defined as an IFC file as shown in figure 2.10, showing both the geometric model and the properties associated with the selected wall.

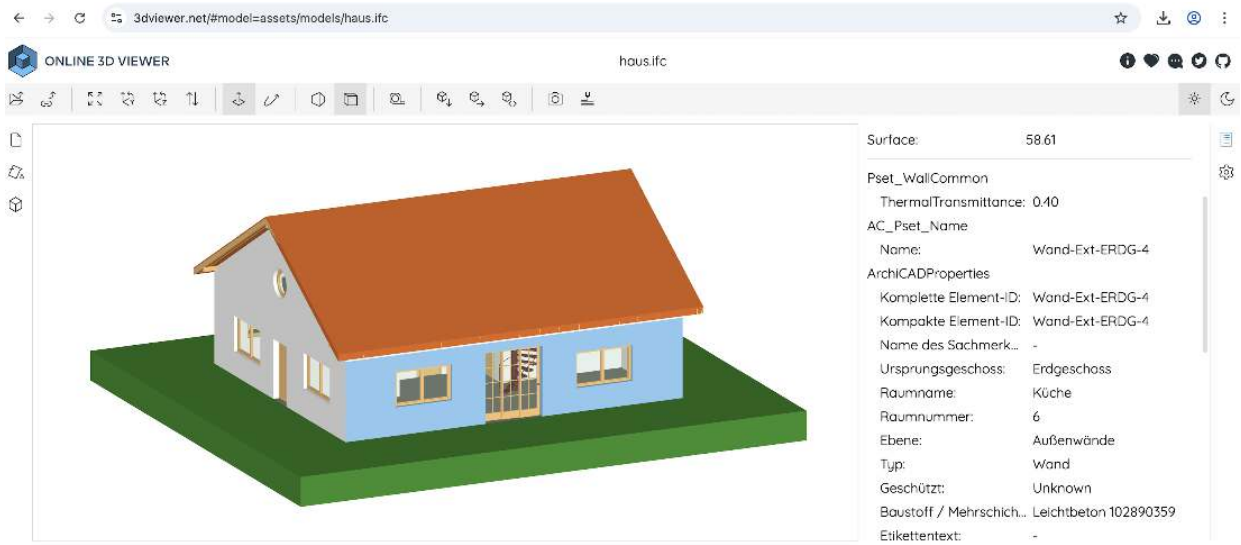


Figure 2.10 An example of a BIM model in IFC showing the 3D model and additional information stored within it. © Viktor Kovacs. Data licensed under MIT.

MESH-BASED 3D MODELS

Mesh-based 3D models offer an accessible entry point into digital twin visualization. They can be created and updated with relatively basic skills compared to other 3D modeling approaches, making them particularly attractive for digital twins where speed of deployment matters more than absolute precision.

Geometric meshes enable efficient 3D visualization that can serve hundreds or thousands of users simultaneously. You can view the mesh-based 3D model of the city of Melbourne in Australia, shown in figure 2.11, by navigating to https://cityofmelbourne.maps.arcgis.com/apps/webappviewer3d/index.html?id=b555219a327b4535a89d8ec6e97780cf.

Figure 2.11 A mesh-based 3D model of the city of Melbourne illustrating an efficient digital representation of a large physical environment. Data © City of Melbourne. Licensed under CC BY 4.0

This accessibility comes with trade-offs. Mesh models approximate complex shapes through networks of points and connecting lines, inherently sacrificing precision for performance. Increasing mesh density improves accuracy but demands greater computational resources, creating a constant balance between visual fidelity and system responsiveness. Despite these limitations, mesh-based models excel in their primary role of intuitive visualization. They prove especially valuable in browser-based digital twin applications, where efficient rendering of realistic environments enables users to quickly understand and navigate complex physical assets without requiring specialized software or high-end hardware.

PARAMETRIC 3D MODELS

Parametric 3D models define geometry through mathematical relationships, parameters, and constraints rather than explicit coordinate data like mesh based models. A cylinder in a parametric model may be defined by its radius and height, and the relationship between these, for example, that the height of the cylinder is three times the radius. The dimensions of the geometric object may change, but this relationship will be maintained in a parametric model. This type of model can easily be iterated on by simply modifying parameters without having to rebuild the whole model, while maintaining the intent of the design during modifications. Parametric modelling is popular and essential in many industries where the accuracy and precision of complex designs are essential, like aerospace, automotive, and mechanical engineering. Most modern professional *computer aided design (CAD)* software is primarily parametric.

Parametric 3D models provide capabilities within digital twins that extend beyond those offered by mesh based geometric modelling. If you are building a design driven digital twin where it is important to model the underlying engineering design that governs a structure or system, or you need to precisely model constraints such as clearances, alignment, and dimensions, then parametric models are essential. Parametric models also have the ability to be modified dynamically based on sensor readings or reconfigured to execute different simulations. Furthermore, parametric models can represent entire product families, automatically generating variants for different applications. A CAD model of an offshore oil platform rendered in the Cognite Data Fusion™ web-based digital twin platform is shown in figure 2.12.

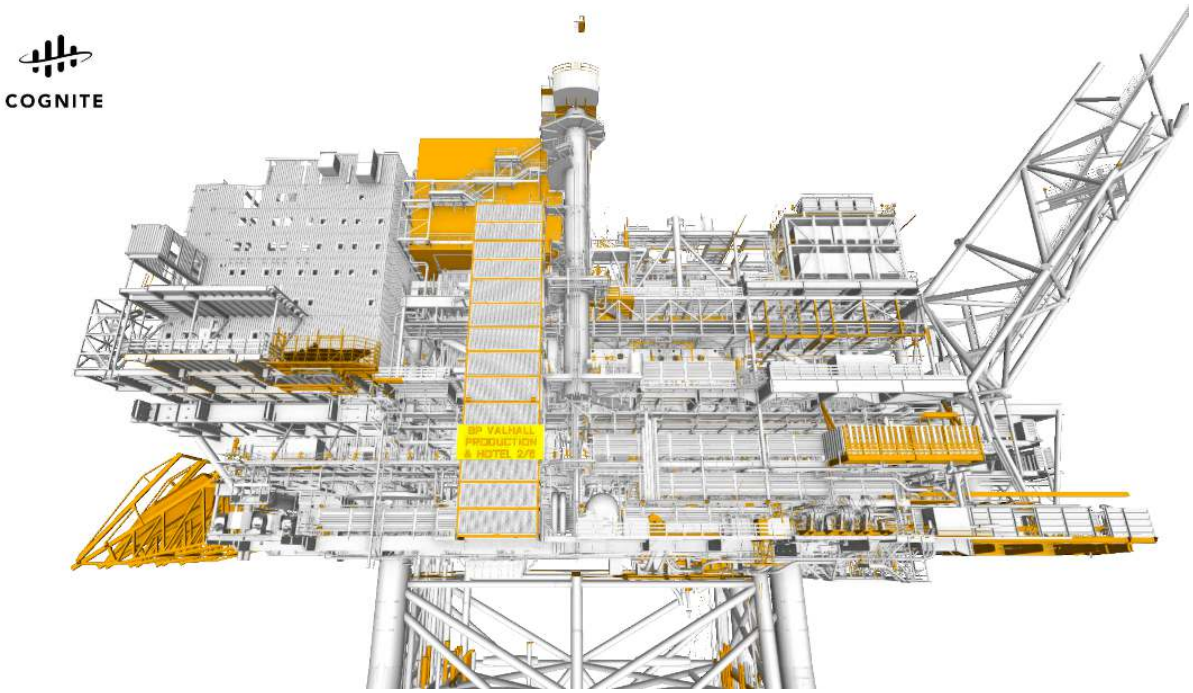


Figure 2.12 A 3D model derived from a CAD model of the Valhall production platform—a complex industrial facility located in the North Sea—as pictured in Cognite Data Fusion™, an industrial digital twin platform. Image © Cognite™ (https://www.cognite.com/en), reproduced with permission.

POINT CLOUDS

A point cloud is a collection of often millions or billions of points in 3D space, with each point having spatial co-ordinates (x,y,z) and color, timestamp, and intensity information. The huge number of discrete points rendered in 3D space collectively represent the geometric shape of physical objects at the time they are captured. This remarkably simple data structure, generated with laser or light scanning, or structure from motion, is a powerful mechanism for capturing and recreating physical objects within a digital twin, particularly large scale environments like buildings, infrastructure, and terrain.

Point clouds are particularly useful for tracking changes over time (with the appropriate equipment, laser scans can be

performed quickly), and identifying wear or damage, particularly with complex or irregularly shaped structures. Unlike mesh based geometric models, point clouds have no inherent surface or volume representation, and are relatively storage intensive. They also support taking precise measurements within the 3D model since the position of every point is stored with a high degree of precision, being based on the calculation of how long a beam of light from a laser takes to bounce off the physical object. Figure 2.13 shows a point cloud of the Morro Bay power plant in California, and how a measurement of the height of the stack can be taken in the model. You can explore this point cloud yourself by navigating to https://viewer.copc.io/?q=https://3d.dtia.site/pointcloud/ept.json.

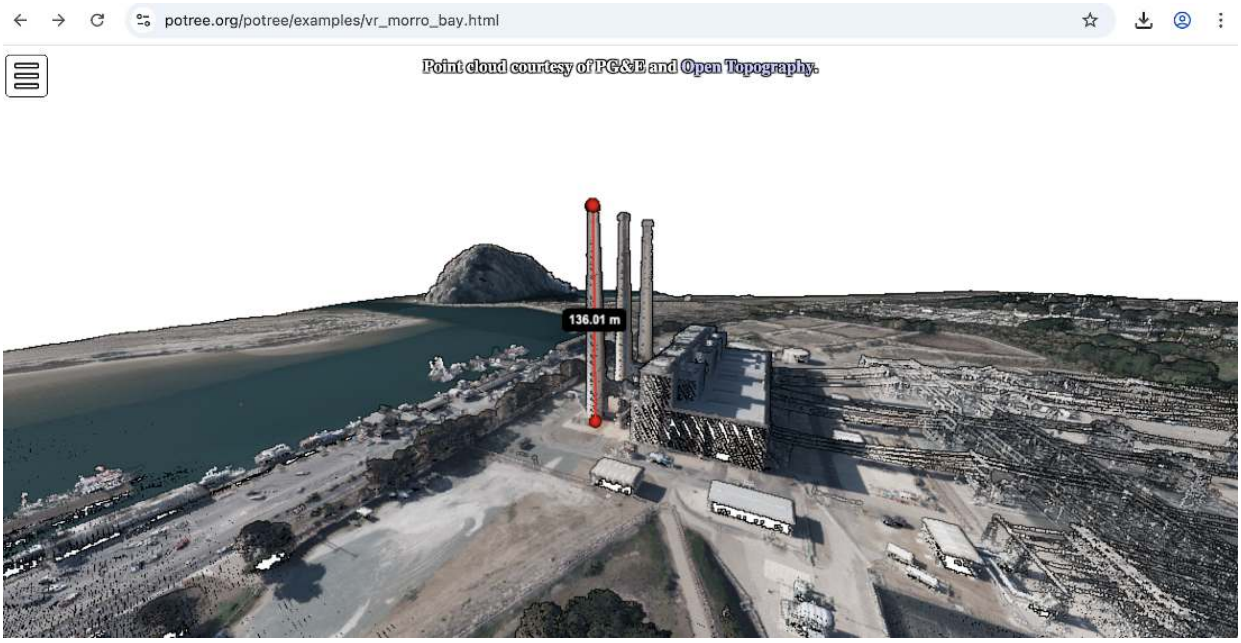


Figure 2.13 A point cloud visualization of the Morro Bay power station, showing how measurements can be taken within the point cloud. © PG&E Diablo Canyon Power Plant (DCPP): San Simeon and Cambria Faults, CA. Distributed by OpenTopography. https://doi.org/10.5069/G9CN71V5

OTHER TYPES OF 3D MODELS

Some domains require fundamentally different approaches to spatial representation. Medical imaging, materials science, and non-destructive testing have developed sophisticated techniques for capturing and modeling internal structures that remain invisible to conventional 3D scanning methods. Among these, computed tomography (CAT or CT scanning) and voxel-based modeling represent powerful paradigms that are also finding applications beyond their traditional medical origins. A voxel (volume pixel) represents the 3D equivalent of a pixel, defining a discrete unit of volume within a 3D space. Unlike traditional geometric models that define surfaces and boundaries, voxel models partition space into a regular grid where each cell contains specific properties or material characteristics. This difference in representation makes voxels particularly suited for modeling heterogeneous materials, internal structures, and phenomena that vary continuously through volume rather than existing only at surfaces.

2.5 Spatial mapping and reference systems

We can see that digitally representing physical assets and objects as 2D representations, 3D models, and digitized data from documents is a powerful aid to understanding, but there is another dimension to this representation, and that is the position of the objects in the real world. Spatial reference systems are systems that allow us to precisely measure locations on the surface of the earth and with many different types in use, it is crucial to understand how these systems work to create an accurate spatial representation of reality where we can precisely position objects in virtual space and determine spatial relationships between objects.

2.5.1 Coordinate systems and projections

When modelling physical objects, there are a number of categories of reference systems that you need to be familiar with. Often, you will need to transform between each of these different systems when moving between representations, and it is important to understand what each type of reference system is, what it is used for, and how you may convert between each type.

Table 2.6 Different types of coordinate reference systems that are important when building digital representations of a physical system.

System	Description	Key use	Integration challenges
Local coordinate systems	Site-specific frameworks using an origin point and axes aligned to project features (for example building grids)	Engineering drawings and CAD models; intuitive for site design and calculations (for example a facility layout)	Coordinates are only meaningful locally, complicating integration with external data
Geographic coordinate systems (GCS)	Spherical systems representing points on Earth's surface (spheroid) using Latitude and Longitude	GPS data (often using the WGS84 datum / EPSG:4326), satellite imagery, and global asset tracking	Angular nature complicates engineering calculations (distances, areas) due to Earth's curvature
Geocentric coordinate systems	3D Cartesian coordinates with the origin (0,0,0) at the center of the Earth	Earth-Centred, Earth-Fixed (ECEF) / EPSG:4978 is common. Used for direct calculation of 3D distances and vectors between any two global points	N/A
Projected coordinate systems (PCS)	Techniques that flatten the Earth's spheroid onto a 2D plane	Web Mercator (online mapping standard, Google Maps) and Universal Transverse Mercator (UTM)	PCS inherently distort shape or area; for example, Web Mercator exaggerates size away from the Equator

Figure 2.14 shows the same point on the earth's surface represented in a geographic coordinate system (latitude, longitude) on the left, and in a projected coordinate system (EPSG:54050). Notice that the projected coordinate system coordinates are presented as meters from an origin point on a 2D plane, making calculations simpler than those required with with the angular nature of the spherical geographic coordinate system.

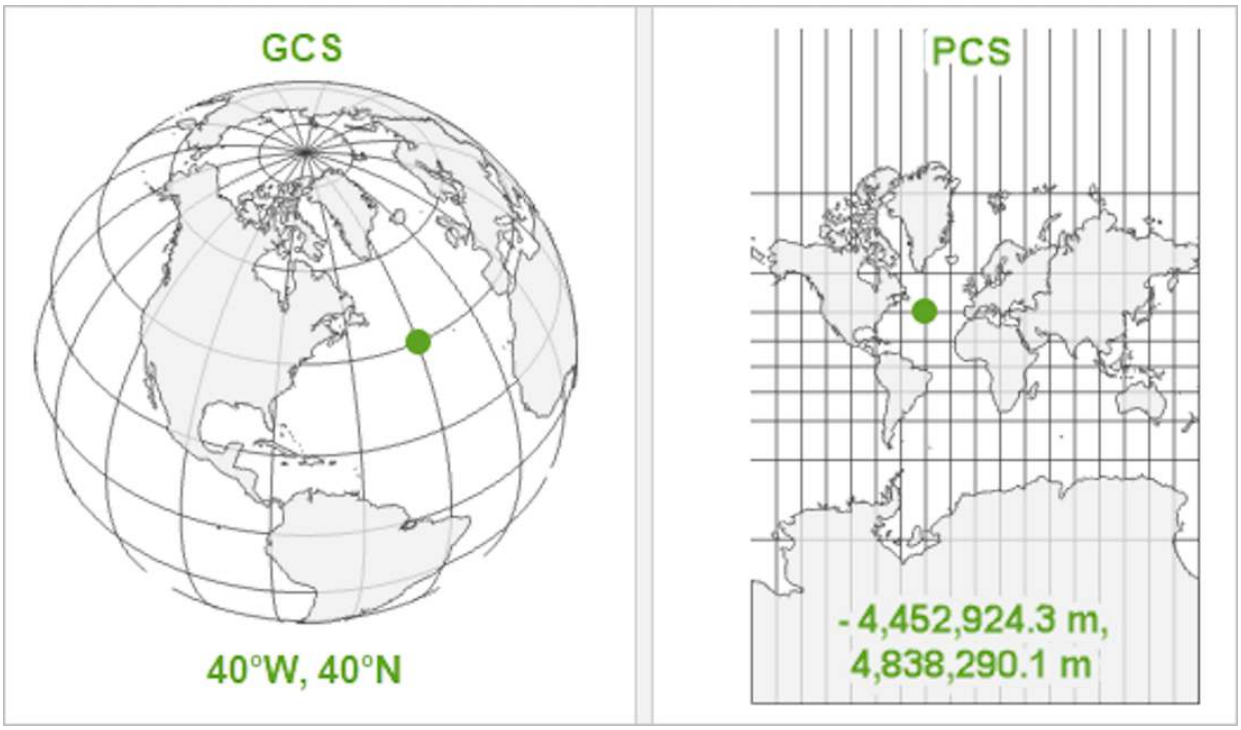


Figure 2.14 An example of the same point represented in a geographic coordinate system and a projected coordinate system. Copyright © 2025 Esri and its licensors. All rights reserved.

NOTE

The European Petroleum Survey Group (EPSG) originated a public registry of datums, coordinate reference systems, and earth ellipsoids known as the EPSG Geodetic Parameter Dataset, assigning a unique code to each entity, known as the EPSG code. For the WGS84 geographic coordinate reference system (the coordinates that you may be familiar with in GPS), the EPSG code of the 2D geographic coordinate reference system is **EPSG:4326**.

TRY IT OUT: EXTRACT GEOGRAPHIC COORDINATES FROM A PHOTOGRAPH

Photographs are an important way to digitally represent a physical system, and many contain geographic coordinates embedded in them via *exchangeable image file format (EXIF)* metadata. Listing 2.6 demonstrates how geographic coordinates can be extracted from a photograph to show where on the earth's surface the photograph was taken, and plot this on Google Maps, showing how spatial context can be extracted from a digital image of a physical object.

Listing 2.6 Extract geographical coordinates from a photograph and display the location the photograph was taken in Google Maps.

```
from PIL import Image
from PIL.ExifTags import TAGS, GPSTAGS
import webbrowser

def main():
  exif = Image.open("./images/rockPaintings.jpg")._getexif() #1
  gps = next((dict((GPSTAGS.get(k, k), v) for k, v in val.items())
  for tag, val in exif.items() if TAGS.get(tag) == "GPSInfo"), {})

  def convert_to_degrees(value): #2
    d = float(value[0])
    m = float(value[1])
    s = float(value[2])
    return d + (m / 60.0) + (s / 3600.0)

  lat = convert_to_degrees(gps['GPSLatitude'])
  lat *= -1 if gps['GPSLatitudeRef'] == 'S' else 1

  lon = convert_to_degrees(gps['GPSLongitude'])
  lon *= -1 if gps['GPSLongitudeRef'] == 'W' else 1

  webbrowser.open(f"https://www.google.com/maps?q={lat},{lon}") #3

if __name__ == "__main__":
  main()
```

#1 Extract GPS coordinates from EXIF data in the photograph.
#2 Convert GPS coordinates to degrees in float format.
#3 Open Google Maps with a pin at the location where the photograph was taken.

2.5.2 Coordinate transformations and conversions

So why is it so important to understand the different types of spatial reference systems when capturing physical systems for digital representation? Real-world digital twin projects frequently encounter coordinate system complications, including mixed data sources using different

reference systems, incomplete transformation documentation, and legacy systems with poorly defined coordinate frameworks. Field survey data might use local construction coordinates, while satellite imagery uses geographic coordinates and facility drawings reference an arbitrary site grid.

When we work with local coordinates in a floorplan or blueprint, we specify an origin point (0,0) and everything in the floorplan has local coordinates relative to that origin. To position this floorplan on a real-world map using global projected coordinates like the Universal Transverse Mercator (UTM) projection, we need to transform each point's coordinates. This transformation from 2D local coordinates to projected global coordinates involves two operations. First, if the floorplan is not oriented towards true North but rotated towards a 'site North', the coordinates must be rotated by the offset angle θ. Then, they must be translated by the offset from the target reference system's origin. To transform local 2D coordinates to UTM coordinates, a 3x3 matrix can be applied where (tx, ty) are the UTM coordinates of the origin point (easting, northing).

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & tx \\ sin(\theta) & cos(\theta) & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Listing 2.7 shows how we can perform this coordinate transformation in code by using matrix multiplication.

Listing 2.7 Using a coordinate transformation matrix to convert local coordinates to projected global coordinates in UTM zone 36S (EPSG:32736).

```
import numpy as np

def create_transformation_matrix(origin_easting,
    origin_northing, rotation_angle_degrees):
  theta = np.radians(rotation_angle_degrees)

  matrix = np.array([ #1
    [np.cos(theta), -np.sin(theta), origin_easting],
    [np.sin(theta), np.cos(theta), origin_northing],
    [0, 0, 1]
  ])

  return matrix

def transform_coordinates(local_coords, transform_matrix):
  homogeneous_coords = np.hstack( #2
    [local_coords,
      np.ones((local_coords.shape[0],
      1))]
  )

  utm_coords = (
    transform_matrix @ homogeneous_coords.T).T #3

  return utm_coords[:, :2] #4

if __name__ == "__main__":
  origin_easting = 471519 #5
  origin_northing = 7977628

  rotation_angle = 30 #6

  transform_matrix = create_transformation_matrix(
    origin_easting, origin_northing, rotation_angle
  )

  local_coords = np.array([ #7
    [0, 0],
    [100, 0],
    [0, 100],
```

```
    [100, 100]
])

utm_coords = transform_coordinates(local_coords, transform_matrix)
print(utm_coords)
```

#1 Create the transformation matrix
#2 Add a column of ones to make the coordinates homogeneous
#3 Apply the transformation as a matrix multiplication
#4 Return just the easting and northing (drop the homogeneous coordinate)
#5 UTM coordinates of the origin point
#6 Rotation angle in degrees of the site offset from true North
#7 The array of local coordinates

Figure 2.15 illustrates the coordinate transformation performed in listing 2.8, showing the original local coordinates in the 0,0 origin based reference grid, and then the transformed coordinates in the UTM Zone 36S grid, assuming that the site is offset by 30 degrees from true North.

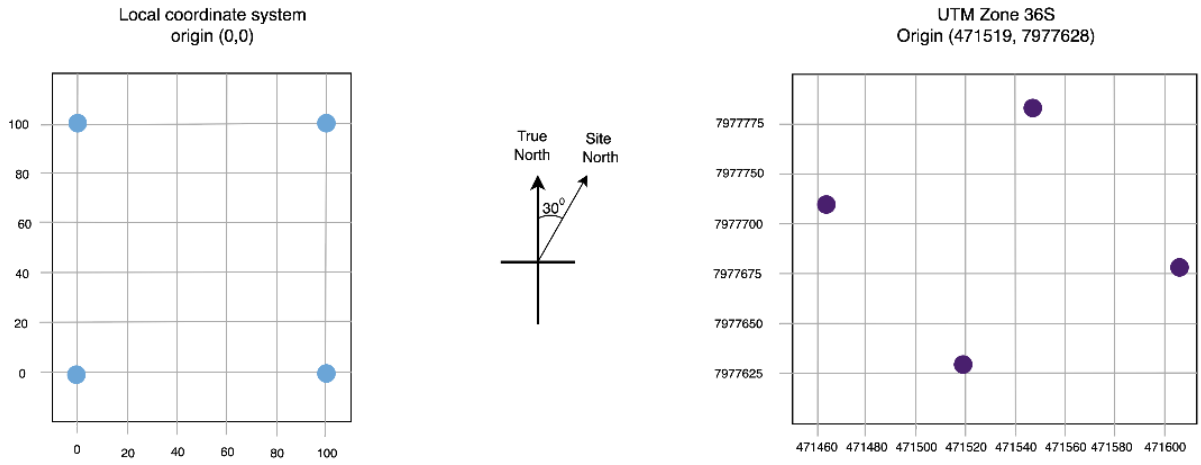


Figure 2.15 Visualization of the output of the code in listing 2.8 that converts local coordinates to UTM coordinates in zone 36S (EPSG:32736) with an origin point at 471519, 7977628 and a site offset of 30 degrees from true North.

While it is important to understand the mathematics of how coordinate transformations work, in practical digital twin applications, you can leverage higher level libraries to

perform coordinate transforms for you. Listing 2.8 shows the use of the Python version of the PROJ library (https://proj.org), a popular library for geospatial coordinate transformation to convert between UTM coodinates and WGS84 with a single line of code.

Listing 2.8 Using PyProj to transform UTM coordinates to WGS84 is a lot simpler than handling the transformation from scratch.

```
from pyproj import Transformer

transformer = Transformer.from_crs(
  "EPSG:32736",
  "EPSG:4326",
  always_xy=True)

easting, northing = 471519, 7977628
lon, lat = transformer.transform(easting, northing)
print(lon, lat)
```

2.6 Deciding what you need

We have covered many sources of information about physical systems and spaces that you can use to build your digital representation, but how do you decide which of these to use? You may have hundreds of engineering documents or hours of video of your operations, but before investing in OCR or object detection, you should map each potential information source back to your objectives and understand whether that source supports your outcomes, and what specific decisions it will enable. Understanding how accurate and complete each potential information source is, together with how easily it can be integrated into your digital twin, will also help you assess whether it is useful or not.

In table 2.7 is a checklist where I have mapped which information sources I will use to build a digital representation of my home.

Table 2.7 Information source checklist for my home digital twin.

Source	Home digital twin	Priority	Effort	Notes
Historical records				
• ❑ Utility bills	Electricity/water bills for baseline	High	Low	Use OCR to extract data from PDFs
• ❑ Maintenance records	Appliance service history & warranties	Low	Low	Scan receipts, create digital maintenance log
Visual documentation				
• ❑ Current photographs	Meter readings, home condition	High	Low	Use a smartphone camera with GPS metadata
• ❑ Aerial/drone imagery	Roof condition, solar panel placement	Low	Medium	Can be used to generate a photorealistic 3D mesh model
Engineering Documents				
• ❑ Building blueprints	Original house plans with room dimensions	High	Medium	Scan and build a 3D mesh model
External system data				
• ❑ Weather services	Local forecast and historical climate	High	Low	Free APIs available

	data			
Spatial representation				
• ❑ Floor plans	Room layouts for sensor placement planning	High	Medium	Generate from blueprint
• ❑ 3D mesh model	Visual representation for data contextualization	Low	High	Use photogrammetry or 3D modeling software
• ❑ Coordinate reference system	Local coordinates sufficient	Low	High	Precise global coordinates not required

Classifying each information source with its importance versus the effort it will take to incorporate into your twin helps you to prioritise what to focus on first. I will start with those sources that have a high priority and relatively low effort, before moving on to those with high priority but higher effort, before finally tackling those with lower priority.

2.7 Summary

- When building a digital twin and looking to capture physical systems for digital representation, start with objectives, not technology. Define specific, measurable business outcomes before deciding how to capture physical systems for digital representation, and at what level of fidelity.
- Historical documentation that captures designs, drawings, and the state of systems can hold critical data

that can be extracted into a machine readable format using OCR and computer vision, providing the temporal context essential for predictive digital twins.

- Photographs and videos are an important data source when representing physical systems, both as representations of the physical environment at a point in time and as a source to build more complex 3D models or extract information from.
- Three main 3D representation approaches serve complementary purposes: mesh-based models provide efficient visualization for web browsers, parametric models enable precise engineering and design iteration, and point clouds capture complex geometries and track physical changes over time.
- Understanding coordinate systems (local, geographic, projected) and transformation techniques ensures that diverse data sources from GPS tracking to CAD drawings can work together seamlessly.
- Match representation fidelity to decision requirements and start with the minimal possible representation of the physical world in your digital twin. Prioritize what aspects of the physical environment you will capture to support your objectives.

3 Sensing the real world

This chapter covers

- The role of sensors in updating a digital twin
- Building a complete sensing system for real-world applications
- Processing sensor data
- Managing sensors at scale

Digital twins depend on maintaining an accurate and timely connection to reality. While 3D models show you what a system looked like when designed, a true digital twin evolves continuously with its physical counterpart through systematic sensing of the physical world. Sensors bridge the physical and digital worlds, continuously capturing changing conditions such as temperature fluctuations, equipment vibrations, fluid flows, and human activities and converting them into data streams that keep digital twins synchronized with reality.

This data foundation allows digital twins to not just monitor, but to optimize system performance by adjusting parameters in the real system based on simulated outcomes. As digital twins mature from descriptive (what happened) through predictive (what will happen) to autonomous capabilities (automatically adjusting system behavior), the demands for rich, timely data increase dramatically.

This chapter walks you through building a complete sensing architecture using the home digital twin project to illustrate common challenges in any digital twin project: selecting appropriate sensors, handling infrastructure constraints, and balancing cost with capability. You'll learn how to design sensing systems that capture the right data at the right frequency, connect diverse sensors through multiple

communication protocols, and process sensor data into formats suitable for digital twin applications. By the end of this chapter, you'll understand how to architect sensing systems that provide the data foundation necessary for digital twin applications across any domain.

3.1 How sensors work

Modern sensors enable digital twins to monitor various aspects of physical systems. Building management requires environmental monitoring (temperature, humidity, occupancy) to optimize heating, ventilation and air-conditioning (HVAC) systems. Equipment health monitoring relies on motion and vibration sensors (accelerometers, gyroscopes) to detect anomalies before failure. Environmental quality monitoring uses chemical sensors to track air pollutants and water conditions. Energy management depends on electrical sensors measuring current, voltage, and power consumption. Spatial monitoring employs optical sensors (cameras, light detectors) and ultrasonic sensors for occupancy and distance measurement. Effective digital twins require strategic sensor selection, deploying only sensors that provide data necessary for the specific decisions and actions the system must take.

What distinguishes a digital twin from traditional monitoring systems is both purpose and scale. Traditional monitoring systems passively track metrics and alert when thresholds are breached. Digital twins go further by enabling active control based on continuous feedback. They require:

- *Data density*—comprehensive streams that enable predictive capabilities through machine learning.
- *Simulation capabilities*—rich historical data for scenario modeling and what-if analysis.
- *Timely synchronization*—frequent updates that keep the digital model current with the physical system.

- *Contextual understanding*—multiple data streams that reveal relationships between system components.

No matter which type of sensor you choose, they all follow a similar path from physical measurement to digital data, as shown in figure 3.1.

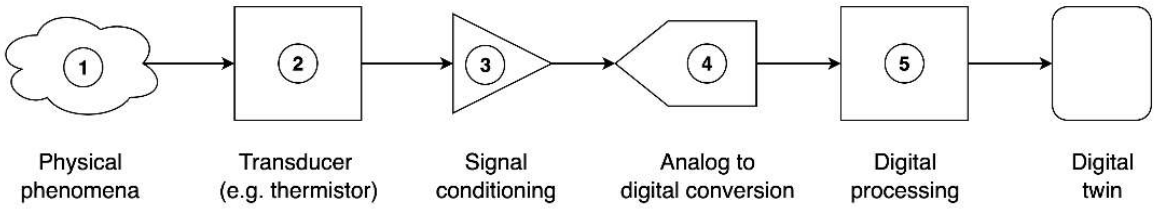


Figure 3.1 The sensor signal chain by which a change in the physical environment makes its way to the digital twin representation via an electronic sensor.

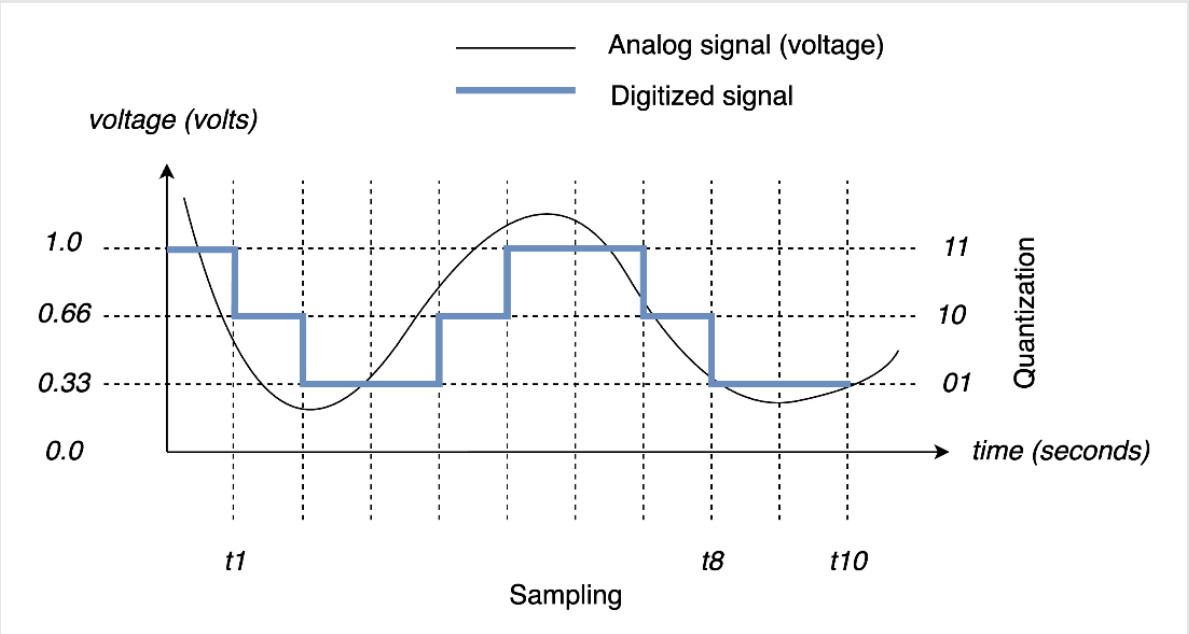
1. *Physical change*—something in the environment changes (temperature rises, pressure increases, object moves).
2. *Transduction*—the sensor's sensing element converts this physical change into an electrical signal, typically a voltage change.
3. *Signal conditioning*—the raw signal gets amplified and filtered to remove noise.
4. *Analog-to-digital conversion*—the voltage is sampled and converted to digital values.
5. *Processing*—digital signal processing performs offset and gain correction, linearization and digital filtering.
6. *Communication*—formatted data is transmitted to your digital twin platform.

Consider measuring temperature, which is something I will do throughout my home. A thermistor in a temperature sensor I have installed varies its resistance based on the ambient temperature, producing a tiny proportional voltage in a circuit. This millivolt signal needs amplification to a usable level. An analog-to-digital converter (ADC) then samples this voltage, converting each sample to a digital number. Firmware in the

sensor applies calibration curves to convert raw ADC counts to actual temperature values, then formats this data for transmission.

HOW IS A PHYSICAL CHANGE CONVERTED TO A DIGITAL VALUE?

The analog to digital conversion at the core of the sensor signal chain is illustrated below. The continuously varying voltage caused by temperature fluctuations is *sampled* once every second (or at a frequency of 1Hz), and is *quantized* with a 2-bit converter. *Quantization* refers to the process of mapping the continuous output value (the voltage from the sensor), to a smaller set of values. In a 2-bit converter, the continuous voltage is mapped to 2^2 (4) possible values represented by 2 bits. The figure shows that mapping the continuous voltage to one of 4 possible values every second means that some information is lost between time ***t8*** and time ***t10***, where the fall in voltage below 0.33V is not translated to the digital signal. If it's important to your digital twin to be able to detect that fall in temperature, you need to use a sensor with different conversion characteristics.



Listing 3.1 shows how analog to digital conversion works to convert a voltage reading to a calibrated moisture percentage in a 10 bit converter.

Listing 3.1 A simple example of how a continuously varying voltage is quantized to a discrete value by the process of analog to digital conversion.

```
def adc_read(voltage, bits=10, vref=3.3): #1
  max_val = 2**bits - 1
  return int((voltage / vref) * max_val) #2

def to_moisture(adc_value, dry=850, wet=400):
  if adc_value >= dry: return 0
  if adc_value <= wet: return 100
  return 100 * (dry - adc_value) / (dry - wet)

print(f"{to_moisture(adc_read(1.5)):5.1f}%")
```

#1 ADC function takes in in the voltage, the number of bits, and the maximum possible voltage.
#2 Conversion formula from analog to digital.

3.1.1 Key sensor characteristics

Selecting the right sensors for your digital twin is rarely as simple as finding those that measure the parameters you're interested in. You must understand the key characteristics that determine how well a sensor will perform in your specific environment and use case, and evaluate your sensor choice against these.

- *Temporal resolution* refers to how frequently the sensor can provide measurements. A manufacturing line might need temperature readings every second, while a building's occupancy sensor might only need updates every few minutes. Match the sampling rate to your system's dynamics and decision-making needs.
- *Measurement range and sensitivity* defines the limits every sensor operates within. A humidity sensor designed for indoor use might fail in an industrial dryer where localized humidity spikes beyond normal ranges. Similarly,

sensitivity determines the smallest change the sensor can detect, which is important when monitoring small changes and gradual trends.

- *Accuracy* is how close your measurement is to the true value, while *precision* is how repeatable your measurements are. A sensor might consistently read 22.3°C when the actual temperature is 25.0°C, so it is precise but not accurate. For digital twins, you often need both, but sometimes you can compensate for systematic accuracy errors through calibration.
- *Environmental tolerance* refers to the constraints under which your sensor operates. Industrial environments might expose sensors to extreme temperatures, vibration, dust, or corrosive chemicals. Outdoor sensors need weatherproofing. Some environments require intrinsically safe designs to prevent explosion risks, while others demand non-contaminating sensors that won't compromise sample purity in pharmaceutical, food, or clean room applications.
- *Power consumption* is the primary constraint that can shape your entire sensing strategy. Wired power is not always available, forcing you to trade off between power and functionality. When considering battery power for sensors, you will need to carefully consider factors such as sampling rate and the cost of wireless communication to minimize the operational costs of replacing batteries across your sensor fleet.

3.2 Selecting the right sensors for your digital twin

Choosing the right sensor involves more than matching measurement type to physical parameter. A temperature sensor that works perfectly in a home environment might fail in an industrial setting, while a highly accurate sensor becomes useless if it can't communicate reliably with your digital twin

platform. Deciding what to measure, as well as how and when to measure it, must be driven by the objectives you have defined based on the decision-making improvements you aim to achieve. Here are factors to consider when selecting sensors for your digital twin.

3.2.1 Define what you're measuring

Working backwards from the objectives of your digital twin, start by identifying the specific physical parameter that best indicates the condition you're trying to monitor. This isn't always obvious, for example, equipment overheating might manifest as elevated temperature, increased vibration, or changing electrical current draw.

Once you've identified the right parameter, establish the expected operating range. For my home's irrigation system, soil moisture typically varies between 20% (too dry) and 80% (saturated), with optimal levels around 40-60%. Understanding this range helps you select sensors with appropriate measurement spans and avoid paying for unnecessary precision.

Finally, consider the rate of change—how quickly does this parameter fluctuate? Soil moisture changes gradually over hours or days, so sampling four times daily provides adequate coverage. In contrast, electrical power consumption can spike within seconds when appliances turn on, requiring more frequent sampling to capture these changes.

Standard questions to help define what you're measuring include:

- *What specific outcome or failure mode are you trying to detect or prevent?* Different failure modes require monitoring different parameters. For example, bearing failure in a motor might be detected through rising

temperature, increased vibration, or abnormal acoustic patterns, each requiring different sensors and thresholds.

- *Which physical parameters change first when this condition occurs?* Early warning indicators give you time to plan maintenance, order parts, and schedule downtime whereas late-stage indicators only tell you when something has already changed. In an HVAC system, refrigerant pressure variations might signal compressor issues days before temperature control degrades, giving maintenance teams time to respond proactively rather than reactively.
- *How quickly does this parameter change during normal operation?* This determines your required sampling rate, data storage needs, and the complexity of your analysis algorithms. A wind turbine's blade vibration might need monitoring at thousands of samples per second to detect bearing defects, while monitoring a building's daily energy consumption pattern requires only hourly readings.
- *How much historical data do you need to establish baseline behavior?* Without a proper baseline, you can't distinguish normal variations from abnormal conditions. A manufacturing line might require three months of production data to understand typical variation patterns across different products, shifts, and seasonal conditions before anomaly detection becomes reliable.
- *What is the normal operating range for this parameter under typical conditions?* This defines the ranges of the thresholds you will monitor. For instance, a commercial freezer might normally operate between -18°C and -22°C, so alerting at -15°C provides early warning of cooling system degradation before product spoilage occurs at -10°C.
- *What is the acceptable delay between a change occurring and your digital twin being updated to reflect the change?* This drives not only your choice of sensors but the entire architecture that will ingest and process the data from them. A safety-critical pressure relief system might require

sub-second response times with edge processing, while tracking monthly water consumption for billing purposes tolerates daily updates transmitted over low-power networks.

3.2.2 Consider the environment

The physical environment where your sensor will operate often eliminates more options than any other factor. Start with the basic indoor/outdoor distinction: outdoor sensors need weatherproof enclosures (IP65 or higher), UV-resistant materials, and wider operating temperature ranges—my garden sensors must survive both 45°C summer days and wet winter days. Industrial environments amplify these concerns when sensors near manufacturing equipment must withstand constant vibration that would destroy consumer-grade devices within days, while dust can wreak havoc on sensitive electronics. Mounting and access is also important to consider; a sensor in a cramped ceiling space needs to be reliable enough to minimize maintenance visits, while one mounted on vibrating equipment needs secure attachment and possibly shock-dampening mounts. In figure 3.2 I use the floorplan of my home to plan the precise placement of my sensors indoors and outdoors.

NOTE

IP65 refers to an *ingress protection* code of 65, which indicates how well a device is protected against water and dust. The first digit represents solid particle protection, with **6** indicating it is protected against ingress of dust, whilst the second digit represents liquid ingress protection, with **5** indicating it can withstand powerful jets of water in any direction.

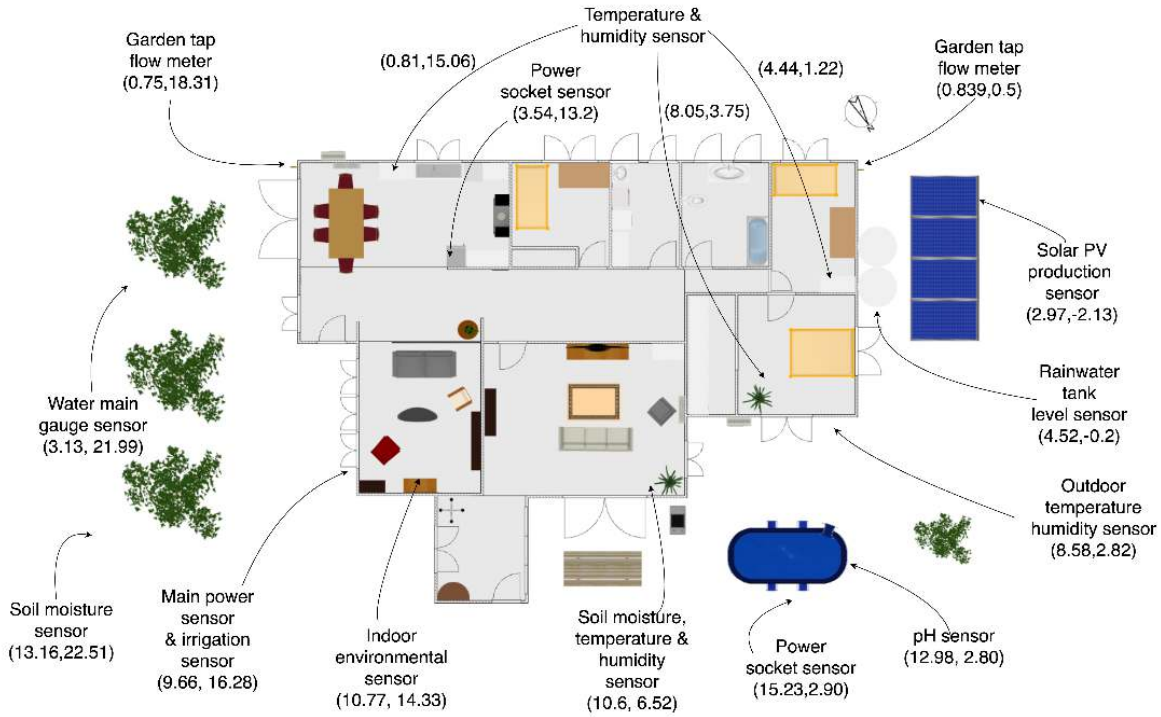


Figure 3.2 Using a floor plan to select the optimal placement of sensors based on what parameters need to be measured, as well as the location of power outlets and taps. The local coordinates of each sensor relative to the origin point (close to the compass symbol at the top right of the image) in meters are also shown.

3.2.3 Determine required performance

Match sensor capabilities to what your digital twin actually needs to deliver value and resist the urge to specify the highest performance possible. For accuracy, ask yourself what decisions the data will drive. My home's temperature sensors only need ±0.5°C accuracy because my HVAC system can't control more precisely than that anyway. Paying extra for ±0.1°C accuracy would be wasting money on precision I can't use. Similarly, sampling frequency should match your system's dynamics and control capabilities. While my pool pump could report power consumption every second, I only sample hourly because that's sufficient to detect when it's running and calculate daily energy usage. More frequent sampling would just drain batteries faster without enabling better decisions. *Latency*, the delay between measurement and availability in

your digital twin, matters most for real-time control. My irrigation system can tolerate long delays because soil moisture changes slowly, but a leak detection system needs sub-minute response to prevent damage. Understanding these requirements early prevents both *over-engineering* (expensive sensors generating data you can't use) and *under-engineering* (missing important events because you sampled too slowly). Table 3.1 shows examples of when higher performance sensors might be required.

Table 3.1 A comparison of situations where higher performance is required across the dimensions of accuracy, sampling rate, and latency, and when more moderate performance will do.

Dimension	High performance required	Moderate performance sufficient
Accuracy	**Leak detection: ±0.01 L/min**. Small flow changes indicate leaks; false positives waste investigation time, false negatives allow damage	**HVAC control: ±0.5°C** System can't control more precisely; ±0.1°C accuracy would waste money on unused precision
Sampling rate	**Vibration monitoring: 10,000 Hz** bearing defects create high-frequency signatures; slower sampling misses critical failure indicators	**Energy monitoring: 1 sample/hour** Sufficient to detect when equipment runs and calculate daily usage; faster sampling drains batteries without enabling better decisions
Latency	**Pressure relief: <1 second** Safety-critical response to prevent equipment damage or injury requires immediate action	**Soil moisture: minutes to hours** Changes occur slowly; irrigation decisions tolerate significant delays without impact

3.2.4 Evaluate practical constraints

Real-world constraints often dictate sensor choices more than technical specifications, with power availability usually being the primary constraint. Mains-powered sensors can transmit

continuously over WiFi, while battery-powered locations demand low-power wireless protocols and infrequent transmissions. Most industrial sites require certified electricians for any powered installation, adding hundreds of dollars of labor cost to each sensor. Maintenance access is equally important when considering battery-powered devices. A sensor located somewhere difficult to access needs a multi year battery life to minimize maintenance costs. My home digital twin takes a typical approach of investing in quality sensors for important measurements (power monitoring) while using $20 consumer sensors for less essential data (spare bedroom temperature). It is important to understand your constraints before purchasing incompatible equipment.

3.2.5 Plan for integration

Before purchasing any sensor, verify it will actually work with your planned infrastructure. Check specific requirements like 2.4GHz vs 5GHz bands, WPA2 vs WPA3 security for WiFi and whether your chosen platform supports the device's specific implementation. Data format mismatches cause significant frustration and extra work. For example, some sensors output raw ADC values requiring complex decoding, while others provide pre-formatted JSON that integrates immediately. My sensors each use different payload formats, requiring custom decoders for every device type, which is manageable for a few sensors but a maintenance nightmare at scale. Some sensors require regular calibration, like my pool pH sensor which must be calibrated monthly using a buffer solution. Understanding these integration requirements before deployment prevents the scenario of having perfectly functional sensors that can't talk to your digital twin platform or produce data in unusable formats.

3.2.6 Choose sensors for a home digital twin

Based on my objectives to reduce energy and water consumption, I need to identify which physical parameters can

provide actionable insights. This means thinking beyond obvious measurements like "electricity usage" to understand what specific data enables optimization decisions. This led me to choose the range of sensors shown in table 3.2.

Table 3.2 Sensors I have chosen to capture the physical environment for representation in my home digital twin.

Measurement	Location	Sensor Model	Power	Update rate
Current flow (total)	Meter box	PowerPal Smart Meter Monitor	Battery	1 min
Current flow (applicances)	Power outlets	Netvox R809A & Milesight CT101	Mains	30 min
Water flow (mains)	Meter	Dragino AIS01-LB	Battery	Daily
Water flow (garden)	Garden taps	Dragino SW3L	Battery	5 mins
Water flow	Laundry	YF-S201 + ESP32	Mains	5 mins
Water storage	Rainwater tank	Dragino DDS20	Battery	Daily
Indoor temperature & humidity	Indoor rooms	Dragino LHT52	Battery	30 mins
Indoor environment	Living area	Milesight AM319	Mains	10 mins
Outdoor temperature & humidity	Outdoors	Dragino LHT65S	Battery	30 mins
Occupancy	Living area	Coral board with edge TPU	Mains	Continuous
Water quality	Pool	Manual pH meter	Battery	Daily

I have selected sensors from a number of different manufacturers, in different form factors. With these sensors, I

can not only target specific appliances and taps, but I can also measure parameters that will influence electricity and water usage, such as the level of my rainwater storage tanks, and environmental parameters that impact power use such as room occupancy, temperature, and humidity. Figure 3.3 shows my most comprehensive sensor that allows me to measure a total of nine environmental parameters with one device.

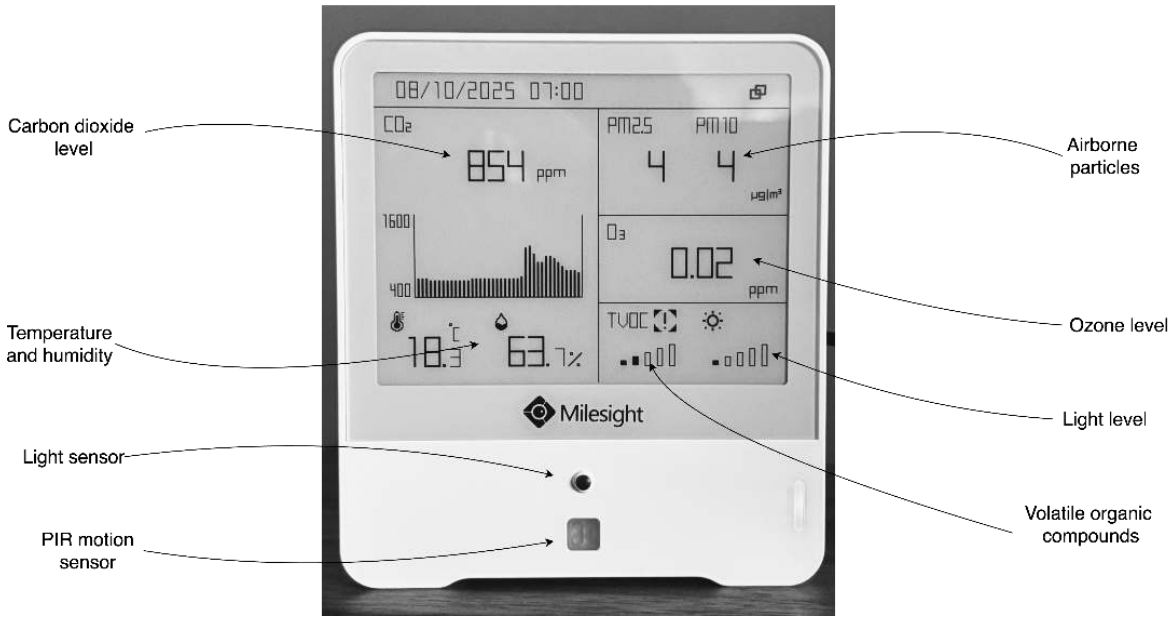


Figure 3.3 An environmental sensor that measures nine different parameters, displaying some on an E Ink display, as well as transmitting them over LoRaWAN.

Sometimes you will not be able to find the sensor that you need available to purchase off the shelf and will need to build your own. In appendix B, I show an example of how you can build an IoT sensor that measures temperature, humidity, and soil moisture using low cost electronic components.

3.3 Connecting sensors: communication technologies

Once you've established what types of sensors you require, you will need to get their data to your digital twin platform. Moving data between two points becomes surprisingly complex when

multiplied across dozens or hundreds of sensors in diverse locations. The choice of communication technology significantly impacts your sensor selection, battery life, deployment cost, and system reliability. A temperature sensor might cost $20, but if it requires a $1,000 cellular modem and $10 monthly service fee to transmit its data, it might be beyond your budget. Conversely, choosing a low-cost communication protocol might limit you to sensors from specific manufacturers or require deploying your own network infrastructure. These dependencies can have wide ranging impact; selecting LoRaWAN for its long range and low power might mean accepting lower data rates, which in turn limits you to simple environmental sensors rather than vibration monitors that generate rich frequency data. The communication landscape has evolved dramatically over the past decade, moving from expensive proprietary protocols to a rich ecosystem of standards-based options. This abundance creates its own challenges of how to choose from the available range of viable technologies.

3.3.1 Understanding the communication landscape

Wireless communication technologies exist on a spectrum, trading off data rate, range, and power consumption, as shown in figure 3.4:

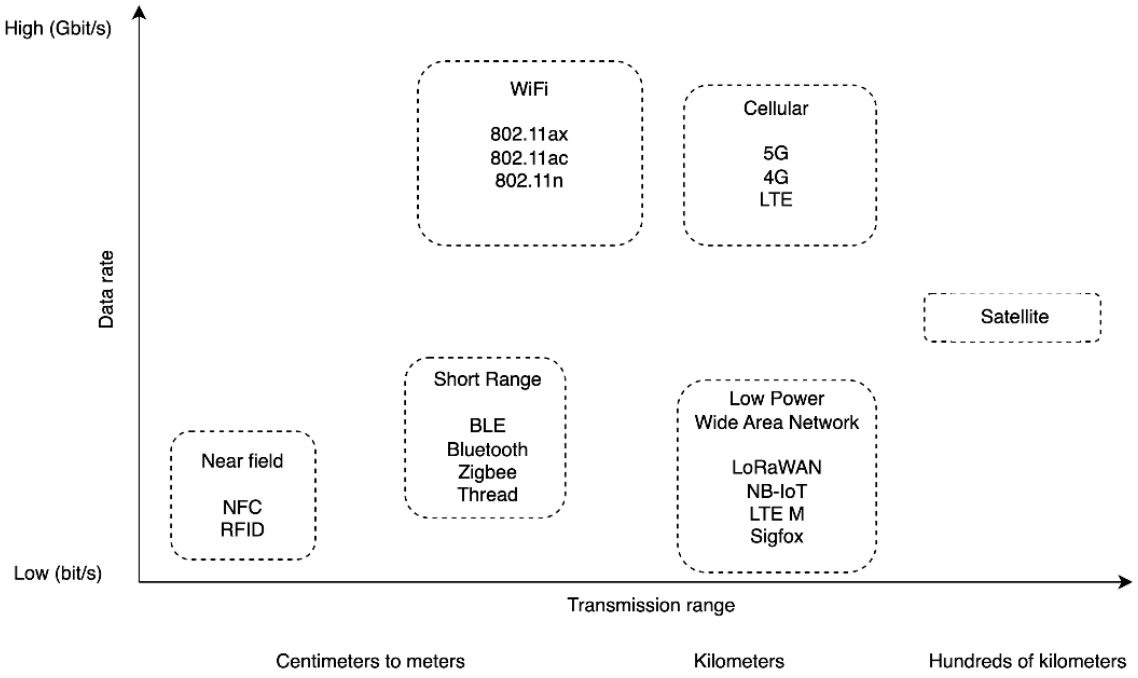


Figure 3.4 Wireless communication technology exists on a spectrum of data rate, transmission range, and cost. Higher data rates work over shorter distances and require more power, whereas longer range communication typically means accepting lower data rates.

No single technology excels at all three since the laws of physics impose fundamental tradeoffs. High data rates require more power and work over shorter distances whereas long range communication at low power means accepting lower data rates.

3.3.2 Short-range technologies (<100m)

When sensors are relatively close to your data collection point and you need frequent updates or rich data, short-range technologies excel. The three dominant short-range technologies shown in table 3.3 each evolved to solve different problems. WiFi emerged from the need for high-speed computer networking and is fast and ubiquitous, but power-hungry. Bluetooth Low Energy (BLE) is the opposite, optimizing for battery life at the expense of range and throughput. Zigbee and Thread (together with the Matter interoperability standard)

occupy a middle ground, designed specifically for sensor networks that need to scale beyond simple point-to-point connections by adopting a mesh-based network topology.

Table 3.3 Popular short-range wireless communication technologies compared on data rate, range, and power draw.

Technology	Data rate	Typical range	Power draw	Infrastructure needs	Home use case
WiFi	54Mbps-1Gbps+	30-100m	High (constant power)	Existing router	Indoor environmental sensors near outlets
Zigbee/Thread	250kbps	10-100m	Low	Dedicated hub	Considered but not selected
BLE	2Mbps	10-30m	Very low	Smartphone/hub	Electricity meter pulse counter

3.3.3 Long-range technologies (>100m)

When sensors are scattered across large areas, lack access to wired power supplies, or there is significant interference (for example, structural steel in an industrial plant), long-range technologies become essential. Table 3.4 provides a comparison of wireless communication technologies that have ranges in excess of 1000 meters. Each of these technologies makes different tradeoffs to achieve their extended range. LPWAN devices have a range of kilometers on battery power, but data rates are measured in bits per second. Cellular operates on licensed frequencies with guaranteed service levels but requires monthly subscription fees, while low-earth orbit satellite services like Starlink have made communication practical in the middle of the ocean, though at a premium price.

Table 3.4 Popular long-range wireless communication technologies compared on data rate, range, battery life and cost model.

Technology	Data rate	Typical range	Battery life	Cost model	Home use case
LPWAN (LoRaWAN)	0.3-50kbps	2-15km urban	5-10 years	One-time hardware	Selected for indoor and outdoor sensors
Cellular	100kbps-50Mbps	National coverage	6-12 months	Monthly fees	Too expensive for my needs
Satellite	25Mbps+	Global coverage	Hours	High monthly fees	Unnecessary for a suburban home

3.3.4 Choosing the right technology

In my home deployment, the choice became clear once I mapped out sensor locations. Indoor sensors near power outlets could leverage my existing WiFi network; there was no need for new infrastructure where I had solid coverage. The electricity meter presented a unique challenge since I could not find an affordable LoRaWAN sensor. Instead, I used a low cost pulse counter that syncs with a smartphone app via BLE, making it the pragmatic choice despite requiring manual data export to transfer the data to my collection hub. Given its popularity in home automation, I seriously evaluated Zigbee for its mesh networking capabilities, but ultimately decided that deploying a separate Zigbee network made little sense when LoRaWAN could handle all my battery-powered sensors in the required range with a single gateway. Figure 3.5 shows a LoRaWAN flow meter sensor that I have installed on one of my external taps.

Figure 3.5 A battery powered LoRaWAN flow meter installed on a garden tap and sending data to my home digital twin.

When looking to select a specific communication technology for the sensors that will feed data to your digital twin, you should consider the following factors:

- *Coverage requirements*—the distance between sensors and gateways, and what technology will support that distance.
- *Power constraints*—whether you have access to mains power or if batteries are a better choice, and how long batteries might last.
- *Data requirements*—how much data, and at what rate, will you need to communicate.
- *Infrastructure*—what already exists versus what you will need to deploy.
- *Total cost*—including devices, gateways, and ongoing service fees.

Given that most of the sensors I have chosen communicate over LoRaWAN, I needed to configure a private LoRaWAN network. Appendix A goes into the details of how I have done this, using a consumer gateway (the Dragino LPS8N indoor gateway) and leveraging a managed cloud service (AWS IoT Core for LoRaWAN) to provide a managed network server. Where I have access to mains power, I use WiFi for communications. This creates a hybrid home sensor network (figure 3.6) where most sensors are battery-powered and communicate over LoRaWAN, while mains-powered sensors in WiFi coverage areas communicate directly with the message broker.

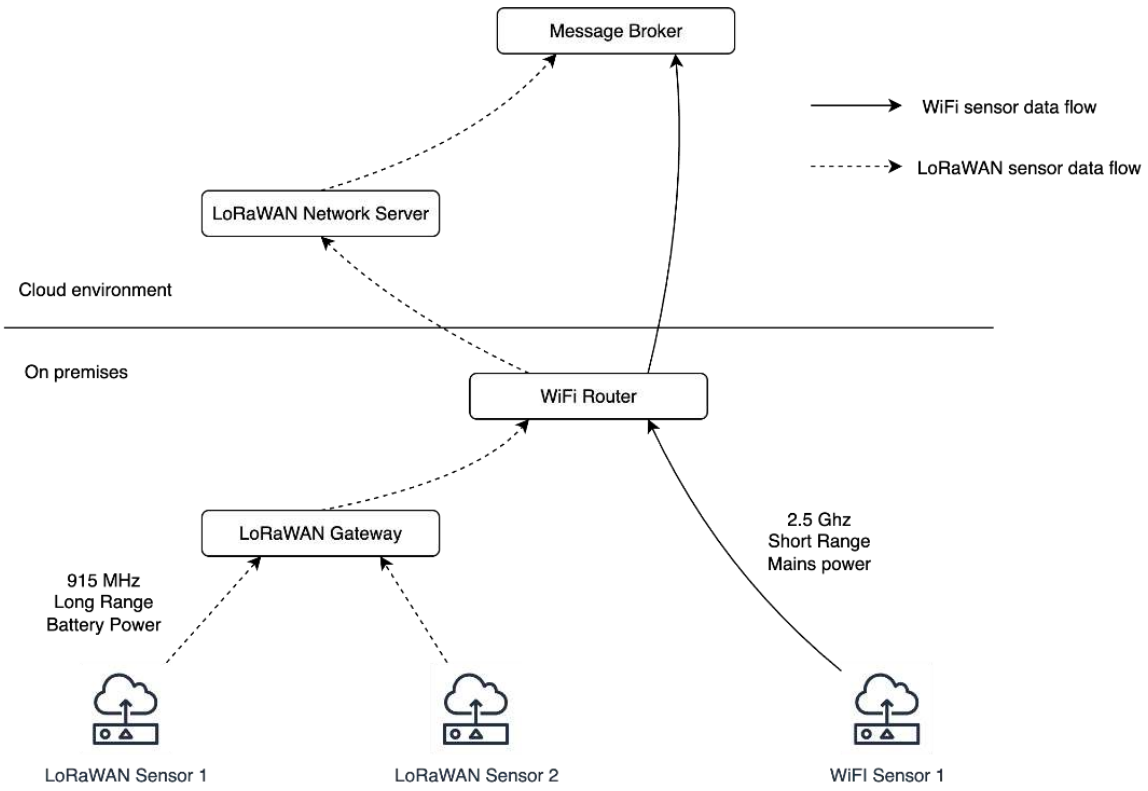


Figure 3.6 A hybrid sensor network that uses LoRaWAN and WiFi to send electronic sensor readings to a common message broker.

NOTE

You may rightly be wondering why not select Home Assistant combined with Zigbee sensors for a home digital twin? Whilst this would be the right choice for a purely home automation project, I will demonstrate technologies that can be extended to industrial, agricultural, and other use cases.

3.3.5 Universal data transport architecture

The data that comes from IoT sensors, whether it is carried over WiFi, LoRaWAN, or other communication technologies, must then be transported to a digital twin application by an application layer protocol that defines the interface between the application and the underlying network over which the data travels (with all the complexities of physical network

communication abstracted away). *Message queuing telemetry transport (MQTT)* is an open source, lightweight messaging protocol that has become the default application protocol for IoT. MQTT uses a loosely coupled, publish/subscribe communication model, which is highly flexible and adaptable and decouples devices and applications.

I use MQTT to transport data from all the sensors that I am integrating to a digital twin application server, which serves as a collection hub for all incoming data. It is not only data from networked electronic sensors that can be transported over MQTT—I also use it to transport manually collected data into my digital twin.

Whatever the source of data, the transformation to MQTT involves several steps:

1. *Protocol translation*—convert from sensor-specific protocols to MQTT messages. The managed network server from AWS handles this conversion to MQTT for LoRaWAN sensors.
2. *Data organization*—assign meaningful hierarchies for data organization.
3. *Payload standardization*—transform diverse formats into consistent structures.
4. *Metadata preservation*—maintain context about data origin and quality.

MQTT IN PRACTICE

Traditional point-to-point communication creates rigid dependencies between systems. If a temperature sensor needs to send data to a monitoring dashboard, a database, and an alerting system, it must know about and maintain connections to all three consumers. When requirements change, such as adding a new analytics service or removing an old system, the

sensor’s code must be modified. This tight coupling makes it difficult to extend and evolve such communication systems.

> ***NOTE***
>
> In publish-subscribe systems, a *topic* is simply a named channel or category that acts as the routing address for messages. Publishers send messages to specific topics without knowing who will receive them, while subscribers express interest in topics without knowing who will send them.

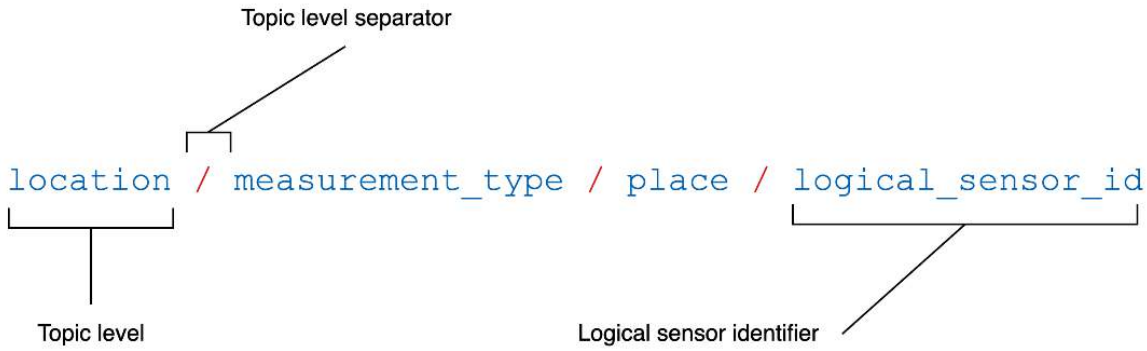
MQTT’s publish/subscribe pattern eliminates these dependencies through the abstraction of a topic, where publishers and subscribers do not need to be aware of each other. The MQTT broker acts as an intelligent middleman, routing messages between parties that remain unaware of each other’s existence.

For my home digital twin, MQTT provides a unified message bus that can carry data from all sensing sources into my digital twin application. Sensors can be added and removed at any time without impacting consumers of their data.

- LoRaWAN sensors will publish data to MQTT via the LoRaWAN gateway and the AWS IoT Core for LoRaWAN managed network server.
- ESP32 sensors connected by WiFi will publish data directly to MQTT.
- Data from sources other than networked electronic sensors will be captured, transformed, and published to MQTT.

It is important to design a functional topic hierarchy within MQTT to support logical subscription selections and support the future growth of your sensing network. A hierarchical structure that reflects the physical structure of your asset enables you to

use wildcards in your subscription patterns. In my home digital twin, I adopt the topic naming convention shown in the following figure, with four topic levels.



Which leads to topics named as follows:

```
home/electricity/electricitymeter/power_meter_1
home/electricity/washingmachine/power_meter_2
home/electricity/refrigerator/power_meter_3
home/water/washingmachine/power_meter_4
home/water/washingmachine/flow_meter_1
home/water/garden/flow_meter_2
home/environment/bedroom1/temp_sensor_1
home/environment/bedroom2/temp_sensor_2
home/environment/outdoors/temp_sensor_3
home/environment/indoors/moisture_temp_sensor_1
```

This enables me to use powerful subscription patterns with MQTT wildcards—for example, subscribing to `home/water/#` captures all water flow readings. By adopting logical sensor identifiers in the topic naming rather than physical sensor identifiers, I can switch out sensor hardware without having to rename any topics, and I can use the same topics regardless of whether the data travelled via LoRaWAN or WiFi.

TRY IT OUT: SEND AND RECEIVE DATA ON MQTT

You can try out publishing data to an MQTT topic and subscribing to messages from MQTT with the code shown in

listing 3.2, which uses a public MQTT broker to send a dummy temperature reading as if it were coming from a temperature sensor in my bedroom. The program subscribes to the same topic to read the published data to the console. Since `test.mosquitto.org` is a public MQTT broker, if you change the code to subscribe to ALL topics and not just the one that you publish to (in MQTT, you can subscribe using the multi-level wildcard `#` to get data on all topics), you will see a deluge of messages of all types that people all over the world are publishing to this free broker.

Listing 3.2 A simple MQTT publish and subscribe example using a public MQTT broker.

```
import paho.mqtt.client as mqtt
import time

BROKER = "test.mosquitto.org"  #1
TOPIC = "home/environment/bedroom1/temp_sensor_1"

def on_message(client, userdata, message): #2
  print(f"Message received: {message.payload} on topic {message.topi
c}")

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2) #3
client.on_message = on_message
client.connect(BROKER)

client.subscribe(TOPIC) #4
client.loop_start() #5

for i in range(5): #6
  temp = 20 + i
  client.publish(TOPIC, json.dumps({"temperature": temp}))
  time.sleep(2)

client.loop_stop() #7
client.disconnect()
```

#1 Use a free public MQTT broker.
#2 Define the message handler to run when a message is received.
#3 Create a new MQTT cleint, define the message handler, and connect to the broker.
#4 Change this to be "#" to see all data on this broker.
#5 Start the client listening.
#6 Publish 5 messages to the topic.
#7 Cleanup.

3.4 Real world data collection strategies

Not every aspect of the physical world can be captured by electronic sensors. Despite the proliferation of IoT devices and ever-decreasing sensor costs, significant gaps remain between what we need to measure and what can be practically automated. Cost, technology limitations, and practical

constraints often require combining automated sensing with manual data collection and third-party data sources to create a complete picture of your physical system. In my home digital twin, fully automating every measurement would cost thousands of dollars and require extensive modifications to existing infrastructure. Industrial digital twins face similar trade-offs at larger scales: a chemical plant might have thousands of manual gauge readings collected during operator rounds, while a smart city might rely on citizen-reported issues to supplement its sensor network. The key to success is to strategically combine different data collection methods based on the value of the information, the cost of automation, and the required update frequency. High-value, rapidly changing parameters justify expensive automated sensing, whereas stable parameters that change slowly can rely on periodic manual collection. Environmental conditions affecting your system may be better sourced from external providers than duplicating measurement infrastructure. By thoughtfully mixing these approaches, you can build a digital twin that delivers insights without breaking budgets or requiring impractical deployments.

3.4.1 Edge processing

Rather than transmitting raw sensor data to remote servers for analysis, edge devices perform real-time processing locally, extracting meaningful insights and transmitting only relevant information. This approach dramatically reduces bandwidth and power requirements and can enable autonomous decision-making even when network connectivity is intermittent or unavailable. My main water meter provides a perfect example of where edge processing is valuable in sensing the real world.

EXAMPLE: ANALOG METER READING

I can convert my water meter from analog to digital by taking a photograph of it and using OCR to extract the current

reading. The challenge, shared in many real world scenarios, is that the use of low power, low bandwidth, long range battery powered sensors means that transmitting every image to the cloud for processing is often not feasible.

The solution I have put in place is to deploy a camera that can run a machine learning model on the sensor device itself to extract the reading from an image, and to transmit only this reading to the cloud. The camera, shown in figure 3.7, uses LoRaWAN to communicate and has a range that allows me to mount it on my water meter on the street (200 meters from my communication gateway), powered by a battery that will last over a year. Processing the camera image to extract the digits reduces a 100KB image to just a 10 byte reading, a 10,000x reduction in transmission size. I can still configure the sensor to periodically send the raw image (it breaks it into smaller data packets for transmission) so that I can verify the edge processing.

Figure 3.7 An edge AI camera installed on my main water meter and sending data to my home digital twin

3.4.2 When electronic sensing isn’t enough

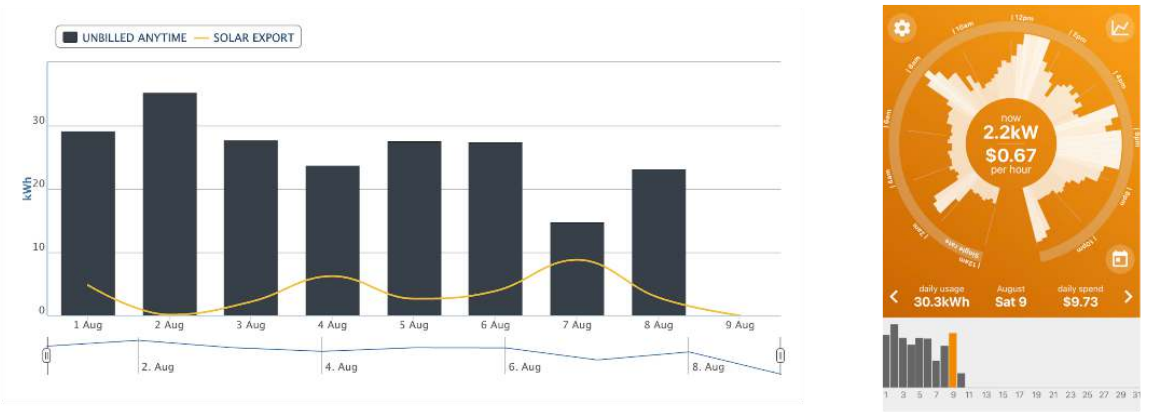
The limitations I encounter in building my home digital twin mirror challenges found in industrial deployments at every scale. Whether you’re monitoring a single home or a sprawling manufacturing facility, you’ll face the same constraints, including that some measurements are too expensive to fully automate, some equipment lacks sensor interfaces, some data is locked in proprietary systems, and some parameters require human observation or interpretation. The difference lies not in the nature of these challenges but in their magnitude—an industrial plant might have thousands of analog gauges instead of one, or deal with vendor lock-in across dozens of equipment suppliers rather than a single utility company. Understanding these constraints through simple home examples helps you recognize and plan for similar situations in more complex environments:

- *Cost constraints*—a comprehensive electrical monitoring system with current transformers on every circuit would cost thousands of dollars. Rather than make this substantial investment, I monitor whole-home consumption with a single optical sensor on my electricity meter, supplemented by smart plugs or current transformers on major appliances.
- *Technology gaps*—I am unable to attach a sensor to my solar inverter to measure electricity production. The inverter does host a simple API available over my local home network that I can query for production data.
- *Existing infrastructure*—my smart electricity meter collects detailed consumption data every 30 minutes, but my utility only provides daily summaries through their web portal. The detailed data exists, but isn’t accessible through any API.
- *Complex measurements*—pool water chemistry requires measuring pH, chlorine levels, alkalinity, and other parameters. Automated sensors for all these

measurements would cost more than my entire digital twin project.

EXAMPLE: ELECTRICITY MONITORING IN THE HOME

Fitting my home out with automated electrical monitoring on every circuit is prohibitively expensive for the objectives of my home digital twin so I have fitted an optical pulse sensor to my electricity meter that records much finer grained data but only transfers the data to my phone. To ingest this data into my digital twin, I must export the data from my phone as a CSV file, and import it to my digital twin. Figure 3.8 shows an example of daily consumption data available to me from my electricity utility via a web portal compared to the data that is available from the new sensor, transmitted to my phone. Notice that my provider portal data is delayed by two days, making it impossible to make meaningful data driven decisions.



Current consumption data

Future state with granular data

Figure 3.8 The electricity consumption data that I currently have available from my provider on the left, compared with the fine grained data I will obtain with an optical sensor, right.

3.4.3 Manual sensing strategies

Manual data collection becomes valuable when automated sensing is impractical, too expensive, or when human observation adds important context. The key to successful

manual sensing is to make manual collection systematic and integrated with your digital twin.

EXAMPLE: POOL CHEMISTRY MONITORING

Electronic pool chemistry monitors cost thousands of dollars, but simple test kits provide accurate measurements for under $50. Here's how I will integrate manual testing into my digital twin:

1. *Standardized process*. To ensure consistency, I manually test the water at the same time each day, from the same location, following an identical procedure.
2. *Digital recording*. I built a simple mobile interface that allows me to enter readings immediately after taking measurements.
3. *Contextual data*. The manual process lets me record qualitative observations like water clarity and debris level that sensors would miss entirely.

This approach will reveal patterns that would be invisible without systematic collection, such as a rise in pH on hot sunny days, changes in chemistry after heavy rainfall and will allow me to optimize the runtime of the filter seasonally to reduce electricity usage. Listing 3.3 shows how easily you can create an interface to capture manual data readings and transmit them to an MQTT broker for processing.

Listing 3.3 A minimal HTML page that uses the client side Javascript AWS SDK in an HTML page to publish an MQTT message containing measured swimming pool pH.

```
<html>
<body>
  <input type="number" id="phValue" placeholder="pH value">
  <button onclick="publish()">Send</button>

  <script src="https://cdnjs.cloudflare.com/ajax/
    libs/aws-sdk/2.1400.0/aws-sdk.min.js"></script> #1
  <script>
    AWS.config.update({ #2
      region: REGION,
      credentials: new AWS.CognitoIdentityCredentials({
        IdentityPoolId: IDENTITY_POOL_ID
      })
    });

    const iot = new AWS.IotData(
      {endpoint: MQTT_ENDPOINT}); #3

    function publish() {
      iot.publish({ #4
        topic: 'home/water/pool/chemistry_sensor_1',
        payload: JSON.stringify({ph: phValue.value})
      }, (e,d) => console.log(e||'Sent'));
    }
  </script>
</body>
</html>
```

#1 Import the AWS SDK for Javascript.
#2 Configure the AWS SDK with your region, and the identifier of your Cognito identity pool.
#3 Instantiate the IoT data client.
#4 Publish the value entered into the input field to the pool water chemistry MQTT topic.

MAKING MANUAL COLLECTION SUSTAINABLE

The biggest challenge with manual data collection is the human factor of ensuring consistent and accurate collection. If the process is difficult or tedious, or the value of the collecting the data is not apparent, it is likely to be abandoned. To create a

sustainable manual data collection process, keep the following in mind:

- *Simplify the data collection process*. Reduce the friction taken for manual data collection by providing an interface that is easy to use and is available in the field where the measurement is taken. Try to eliminate the need to transcribe data by providing a mobile interface where it can be entered directly as the measurement is taken rather than writing it down and then having to input it when back in the office.
- *Provide immediate value*. Providing feedback during the data entry, through alerts when readings are out of normal ranges, or showing trends makes the benefits of performing the data entry tangible and motivates users to continue with the process.
- *Automate reminders*. Consider providing useful automated reminders that prompt users to perform manual collection. If you can make these smarter than just a daily reminder—for example by taking into consideration context such as the amount of time since the last test, or abnormal weather that may impact levels, all the better.
- *Design for scale from the start*. When dealing with hundreds of collectors taking thousands of readings, the challenges multiply. Build in quality controls such as validation rules, duplicate detection, and automated data quality checks to catch errors early. Consider implementing role-based access and workflows that allow supervisors to review and approve submissions before they enter the main dataset. Use dashboards that give managers visibility into collection completion rates across teams, identifying gaps or patterns that need attention.

3.4.4 Integrating third-party data

Many physical parameters affecting your system are already being measured by others. Weather services track

temperature, humidity, rainfall, and solar radiation while utilities monitor grid conditions and traffic services track road conditions. The challenge can be accessing and integrating this data. In my digital twin I integrate with an external weather data source to obtain forecast data, and with the water corporation to obtain data about water restrictions, as well as the state of the metropolitan water supplies.

3.5 Data processing

Sensors produce raw data streams that need processing, decoding, and protocol transformation before they are transported into your digital twin. We will deal with data ingestion and storage in chapter 4, but prior to ingesting the data, we must decode it from sensor specific formats and protocols keeping in mind the unique characteristics of IoT sensor data.

3.5.1 Understanding IoT data characteristics

There are distinct attributes of the sensor data originating from IoT devices that set it apart from traditional data, impacting how it’s handled in digital twin applications. These key characteristics can be summarized in the following way.

VOLUME

IoT devices may generate vast quantities of data, from continuous sensor readings, to extensive log files, requiring scalable storage and processing solutions.

VELOCITY

Data is produced and transmitted at high speeds, often in near real-time. This demands rapid processing to keep digital twins continuously updated and responsive. Often the velocity of

data is variable requiring a processing architecture that can scale elastically.

VARIETY

IoT data comes in diverse formats and structures, originating from various sensor types and systems. Every sensor manufacturer has their own data format. Some send human-readable JSON:

```
{"temp": 23.5, "humidity": 45.2, "battery": 3.2}
```

Others pack data into binary formats to save bandwidth:

```
06 eb 02 9d 7f ff 01 68 62 81 69
```

Even identical sensors might output differently based on firmware versions or configuration. My sensor deployment uses 10 distinct data formats, each requiring specific handling. This necessitates flexible integration and data modeling approaches.

VERACITY

The accuracy and trustworthiness of IoT data can be inconsistent, subject to sensor errors, transmission issues, or device failures. Environmental factors affect sensor accuracy—my outdoor temperature sensors read high in direct sunlight. Occasionally my indoor temperature sensor produces an impossible value of -99°C indicating a malfunction that I must handle.

VALUE

While the ultimate goal is to extract meaningful insights and actionable intelligence from raw data the value can vary dramatically. My soil moisture sensors show noise-level fluctuations between actual watering events. This low value

density means processing must extract signal from noise, identifying genuinely significant changes while filtering meaningless variation.

3.5.2 Message decoding

Sensors transmit data in different formats, using different encoding and protocols that must be decoded and transformed into a format that can be ingested, stored, and processed by a digital twin application. This requires some compute to run to perform the decoding. Since the volume and velocity of sensor data can he highly variable, it makes sense to use serverless compute for message decoding. In my home digital twin I use AWS Lambda functions that activate only when messages arrive, eliminating idle compute costs. For my home digital twin processing approximately 50,000 messages monthly, this translates to mere dollars in compute costs versus hundreds for an always-on server to do the same.

> ***NOTE***
>
> AWS Lambda is a serverless computing service offered by AWS that allows you to run code without provisioning or managing servers. You provide code to run and Lambda executes your code in response to events, managing all underlying compute resources, including server and operating system maintenance, scaling, and capacity provisioning. Microsoft's Azure functions and Google Cloud's cloud functions provide similar serverless compute functionality.

Figure 3.9 shows the architecture of the sensor data collection hub, where each sensor publishes data to an MQTT topic, and a single message decoder function subscribes to all sensor data topics, and is able to decode their payloads.

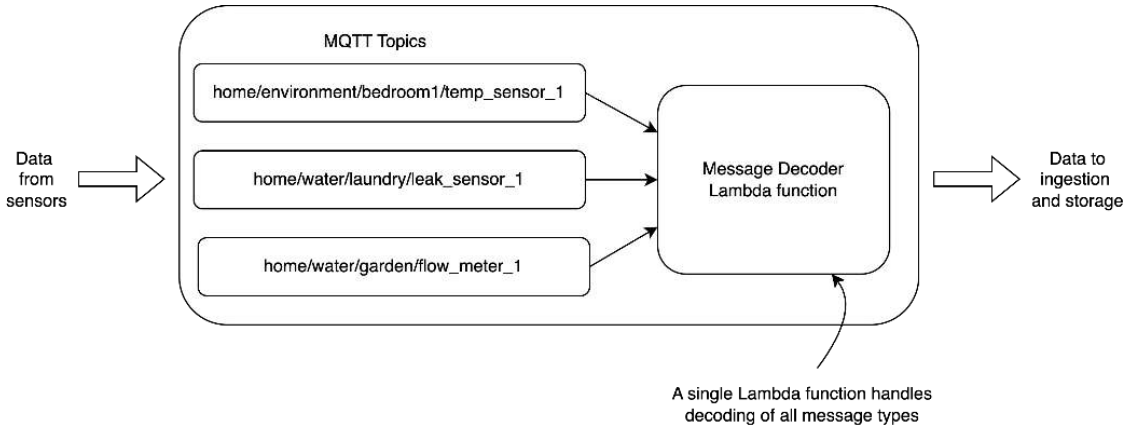


Figure 3.9 The sensor data collection hub architecture where messages from multiple types of sensor are collected, decoded, and routed for storage.

EXAMPLE: DECODING LORAWAN PAYLOADS

My message decoder Lambda subscribes to all MQTT topics that data is published to by sensors, and implements a decoding function for each type of message received. Consider my LHT52 indoor temperature and humidity sensors distributed around the house. When the one in the master bedroom transmits data, the following Base64 encoded hex string is received on the MQTT topic named `home/environment/bedroom1/temp_sensor_1` in AWS:

```
B1QDOH//AWhRYAQ=
```

This Base64 string decodes to 11 hex bytes as follows:

```
07 54 03 38 7f ff 01 68 51 60 04
```

> ***NOTE***
>
> Base64 is a way to represent binary data (like images, files, or any non-text data) using only text characters that are safe to use anywhere—specifically 64 different characters: A-Z, a-z, 0-9, plus two symbols (usually + and /).

According to the device manual, these 11 bytes contain measured temperature and humidity in the first 4 bytes, followed by data from an optional external temperature probe and the measurement timestamp as shown in table 3.5.

Table 3.5 Format of the message received from a Dragino LHT52 indoor LoRaWAN temperature and humidity sensor.

Bytes	Size	Content	Format
0-1	2	Temperature	Signed 16-bit, ÷100
2-3	2	Humidity	Unsigned 16-bit, ÷10
4-5	2	External Temperature	Signed 16-bit, ÷100
6	1	External Sensor ID	Unsigned 8-bit
7-10	4	Unix timestamp	Unsigned 32-bit

The first 2 bytes `0x07 0x54` represent a signed 16-bit temperature value (signed means it can represent temperatures both above and below 0°C, though it never gets below freezing where I live!). Decoding requires bit manipulation as shown in the function shown in listing 3.4.

Listing 3.4 A message decoder function that extracts temperature and humidity from the payload of a Dragino LHT52 sensor.

```
def decode_sensor_message(base64_payload):
  raw_bytes = base64.b64decode(base64_payload) #1

  temp_raw = struct.unpack('>h', raw_bytes[0:2])[0] #2
  temperature = temp_raw / 100.0 #3

  humidity_raw =
    struct.unpack('>H', raw_bytes[2:4])[0] #4
  humidity = humidity_raw / 10.0 #5

  return {
    'temperature': temperature,
    'humidity': humidity,
    'unit': 'celsius'
  }

payload = "B1QDOH//AWhRYAQ=" #6
data = decode_sensor_message(payload)
print(f"Temperature: {data['temperature']}°C")
print(f"Humidity: {data['humidity']}%")
```

#1 Convert Base64 to raw bytes.
#2 Extract temperature (bytes 0-1): signed 16-bit, big-endian.
#3 Convert to degrees Celcius.
#4 Extract humidity (bytes 2-3): unsigned 16-bit, big-endian.
#5 Convert to percentage.
#6 Example encoded payload from sensor.

This transformation serves multiple purposes:

- *Decoding*: converts binary data to human-readable values.
- *Standardization*: creates consistent JSON structure across sensor types.
- *Context addition*: adds units and metadata not present in raw data.

3.5.3 Sensor fusion

Real world sensor data is often noisy and uncertain. Each sensor provides an imperfect view of the true state of the

system, and in a digital twin, relying on a single sensor can lead to an inaccurate representation of the physical system. Sensor fusion combines readings from multiple sensors to estimate a more accurate and reliable representation of the physical system.

One of the most widely used techniques for this is the Kalman filter, which is an algorithm that provides an optimal estimate of the system's true state over time by continuously updating predictions based on both model dynamics and new measurements. It effectively smooths out noise and gives more weight to reliable information as it becomes available.

In a digital twin of a smart building, for example, temperature sensors distributed across a room may all report slightly different readings due to calibration errors or local airflow differences. The Kalman filter can fuse these readings to produce a stable and accurate estimate of the true room temperature.

EXAMPLE: FUSING MULTIPLE TEMPERATURE SENSOR READINGS WITH A KALMAN FILTER

The Python code in listing 3.5 demonstrates how a simple Kalman filter can be used to fuse readings from five noisy IoT temperature sensors into a single, accurate estimate of the true room temperature.

Listing 3.5 A simple implementation of a standard linear Kalman filter in one dimension. The example assumes a steady state temperature, and only estimates a single state.

```
import numpy as np
import matplotlib.pyplot as plt

true_temp = 22.0 #1
timesteps = 50 #2

np.random.seed(42)
sensor_noise = [np.random.normal(0, 0.5, timesteps) for _ in range(5)]
sensors = [true_temp + n for n in sensor_noise] #2
measurements = np.mean(sensors, axis=0) #3

x_est = 20.0 #4
P = 1.0
R = 0.25
Q = 0.01

estimates = []

for z in measurements:
  x_pred = x_est #5
  P_pred = P + Q

  K = P_pred / (P_pred + R) #6
  x_est = x_pred + K * (z - x_pred)
  P = (1 - K) * P_pred

  estimates.append(x_est)

plt.plot(measurements, label="Average Sensor Reading", linestyle="dotte
d")
plt.plot([true_temp] * timesteps, label="True Temperature", linestyle
="--")
plt.plot(estimates, label="Kalman Filter Estimate")
plt.xlabel("Time step")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()
```

#1 Assume a constant real temperature of 22.
#2 Generate 50 readings for each of the 5 sensors, assigning random noise to each reading.
#3 Calculate the average from each of the 5 sensor readings.

#4 Estimate uncertainty (*P*), process noise (*Q*), and measurement noise (*R*)
#5 Prediction step.
#6 Update step.

Figure 3.10 shows the output of the code in lisiting 3.5, plotting the true temperature, the mean of the five temperature sensors, and the temperature estimated by the Kalman filter at each time step. At each step, the filtered estimate is closer to the actual temperature than the mean of the sensor readings.

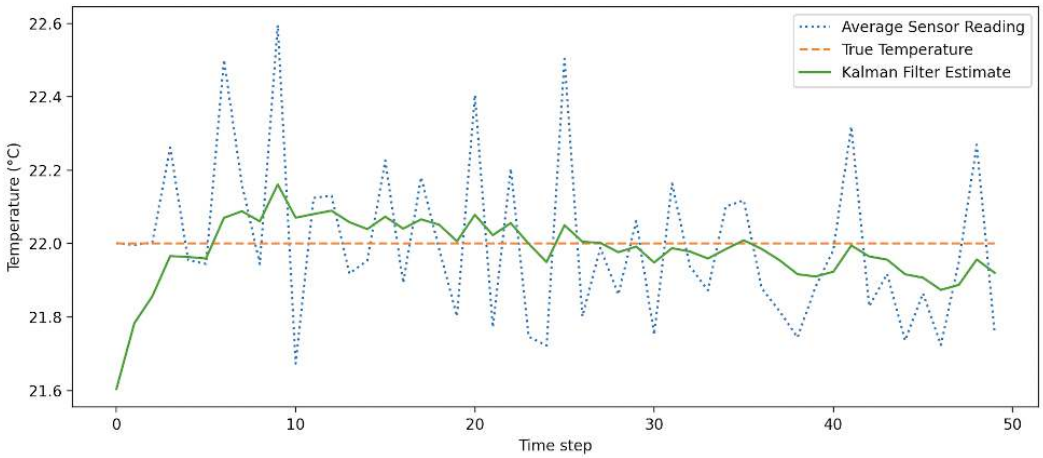


Figure 3.10 Output of the simple Kalman filter running against five noisy temperature sensors as shown in listing 3.5.

3.6 Scalability and device management

Each sensor that you deploy adds multiple management dimensions that compound exponentially as your deployments grow. Deploying the first ten sensors around my home was easy, but as I add more sensors even in the small scale of my home digital twin, managing them manually becomes impossible. I need to keep track of which sensor is deployed where, what version of firmware each is running, and when batteries were last changed. The challenge of the type of large scale IoT deployments required by an industrial scale digital twin is in keeping sensors working reliably at scale. Following

are the challenges that you will face as you scale up sensor deployments for your digital twin.

3.6.1 Battery management

Many battery-powered sensors promise multi-year operation, but that assumes ideal conditions. In reality, batteries drain faster in temperature extremes, with frequent transmissions, or poor network signal quality. Without systematic monitoring, the first indication of a dead battery is missing data only noticed days or weeks later. In my deployment, I've implemented battery voltage reporting in every sensor message, allowing me to predict when batteries will run out and replace them before they do (almost a meta digital twin of the sensor network!). Industrial deployments often standardize on specific battery types and implement replacement schedules based on worst-case scenarios rather than manufacturer estimates.

3.6.2 Firmware updates

Security vulnerabilities, bug fixes, and feature additions require firmware updates throughout a sensor's lifetime. The limited bandwidth of LPWAN networks makes transmitting firmware images challenging, battery-powered devices must manage updates without draining power, failed updates can brick remote sensors requiring expensive site visits, and heterogeneous networks mean different update procedures for each device type. My LoRaWAN sensors require updates during maintenance windows, transmitting firmware in small chunks over days, and maintaining rollback capabilities for failed updates.

3.6.3 Physical asset tracking

Sensors transmit streams of data as soon as they are powered on. Without any knowledge of where that sensor is, or what it

is measuring, that data stream is meaningless or misleading. I maintain both logical locations (what the sensor measures), and the physical location where each sensor is, combined with photographs to show exact mounting positions.

3.6.4 Calibration drift

The edge processing camera I have installed on my water meter requires careful and precise calibration to ensure the ML model extracts the correct digits from the image. Any slight movement on the camera mounting requires me to recalibrate it. My swimming pool pH meter also requires regular calibration using a solution of a known pH. Tracking and coordinating calibration at scale requires careful management.

3.6.5 Security management

Each sensor you deploy represents a potential attack vector into your network. Managing credentials, certificates, and encryption keys becomes increasingly complex with scale. Keys must be rotated periodically, certificates renewed before expiration, and compromised devices quickly identified and isolated. In appendix B, you can work through an example of how AWS IoT Core uses TLS with X.509 certificates for mutual authentication, securing communication between an IoT device in my home, and the cloud.

3.6.6 Practical management strategies

The successful management of large sensor deployments requires a systematic approach that includes the following considerations:

- *Automated monitoring*—implement a system that tracks the health of your sensor network based on factors such as message frequency, battery voltage, and data quality metrics and automatically alerts you of any anomalies. I

have implemented a monitoring system that alerts me when a sensor does not transmit data on schedule.

- *Standardization*—just by limiting the number of manufacturers and device types you need to manage as far as possible can dramatically reduce management complexity. I have standardized my indoor temperature and humidity sensors on a single model that reduces my complexity in managing firmware updates.
- *Batch operations*—if you are managing a large number of sensors, design support for bulk operations. Whether updating firmware, rotating security credentials, or adjusting configuration parameters, the ability to apply changes across device groups rather than individually becomes essential at scale. This requires careful planning of device grouping strategies whether by type, location, criticality, or deployment date. Both AWS and Microsoft offer cloud services for the management of large fleets of IoT devices but my home digital twin has not yet reached a scale where investing in a management strategy makes sense.

3.7 Bringing it all together

The first four phases of a digital twin—collection, connection, transportation, and processing—comprise the foundation that bridges the physical and digital worlds. Figure 3.11 shows how these phases work for my home digital twin.

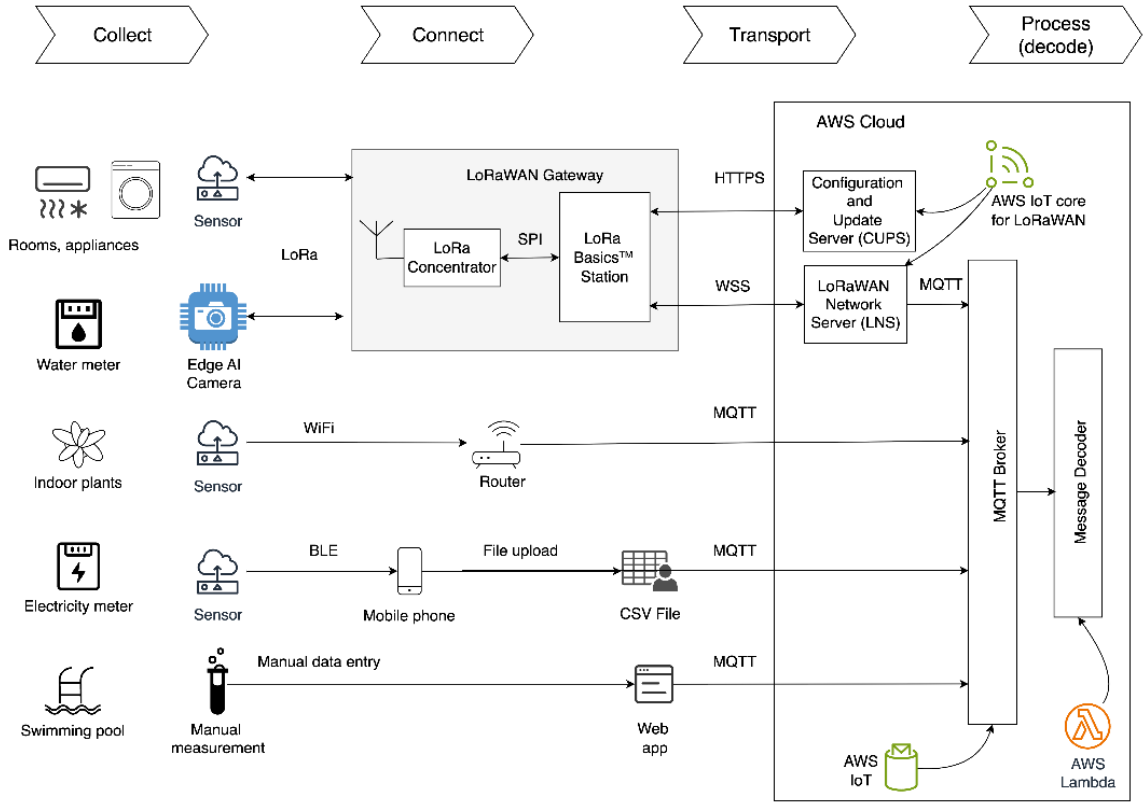


Figure 3.11 Complete sensing system architecture showing collect, connect, and transport layers.

This architecture demonstrates an essential principle of real-world digital twin implementations, that is no single technology solution can address all sensing requirements. Successful digital twins combine multiple approaches to updating their digital representation of the physical world, each optimized for specific requirements and the constraints of the real world.

3.7.1 The collect layer: diverse data sources

The collect layer encompasses all the methods we use to capture information about the physical world. As figure 3.11 shows, this includes four distinct approaches, each addressing different constraints:

- *Electronic sensing*—sensors measuring rooms, appliances, and indoor plants capture environmental parameters directly. These sensors vary in complexity from simple

temperature measurements to multi-parameter devices monitoring air quality, occupancy, and soil conditions.

- *Electronic sensors with edge processing*—the water meter camera represents sensing couple with edge processing where raw data (images) gets processed locally into actionable information (meter readings). This approach solves the bandwidth and power constraints that would make transmitting full images impractical over LoRaWAN.
- *Manual data collection*—the swimming pool represents measurements that are more cost-effective to collect manually than to automate. Pool chemistry testing with a $50 digital meter provides the same accuracy as a $1,000 automated system, making manual collection the pragmatic choice.
- *File-based integration*—the electricity meter shows how existing infrastructure often provides data in formats that require special handling. My optical pulse sensor captures detailed consumption data but only syncs to a mobile phone, requiring CSV export and manual upload to integrate with the digital twin.

This diversity reflects real-world constraints where cost, technology limitations, and practical considerations prevent a uniform approach to data collection.

3.7.2 The connect layer: protocol translation and routing

The connect layer handles the complex task of moving data from diverse sources to a common processing platform. Three distinct communication paths demonstrate how different technologies serve different requirements:

- *LoRaWAN path*—battery-powered sensors in indoor and outdoor locations connect through a LoRa concentrator to a gateway running LoRa Basics Station. This path prioritizes range and power efficiency over data rate, making it ideal

for environmental sensors that transmit small amounts of data infrequently. The gateway handles protocol translation, converting LoRa radio signals to internet protocols.

- *WiFi path*—indoor sensors with access to mains power connect directly through the existing WiFi router. This path offers higher data rates and more frequent transmission, suitable for sensors that need to provide richer data streams or more responsive updates.
- *Manual and file integration paths*—both manual readings and CSV file uploads come through a web application that provides a user interface for data entry and file processing. This application serves as a protocol translator, converting human input and file formats into data that can be published to the data collection hub.

The key is that all paths converge on the same transport protocol, MQTT, regardless of how the data was originally collected or transmitted.

3.7.3 The transport layer: unified data flow

The transport layer creates a unified data stream from diverse sources using MQTT as the common protocol. This design provides several important advantages:

- *Protocol abstraction*—the message decoder receives all data through MQTT subscriptions, regardless of whether it originated from a LoRaWAN sensor, WiFi device, manual entry, or file upload and performs any message decoding required. This abstraction simplifies application logic and enables consistent data processing.
- *Scalability*—adding new sensor types or communication methods requires publishing data to MQTT topics and implementing any message decoding that is required. The core application logic remains unchanged.

- *Reliability*—MQTT's publish/subscribe model provides built-in features for handling connection failures, message persistence, and quality of service guarantees that ensure data reaches the application even when network conditions are poor.
- *Cloud integration*—AWS IoT Core provides managed MQTT infrastructure along with integration to other cloud services for storage, processing, and analysis. The LoRaWAN Network Server (LNS) and Configuration and Update Server (CUPS) handle the complexity of managing LoRaWAN devices while presenting a simple MQTT interface to applications.

3.7.4 The processing layer: message decoding

Initial processing of sensor data is performed in the message decoder function where device-specific data formats from different types of sensors are converted to a common data format before being routed for additional processing and storage.

3.7.5 Data flow walkthrough

To understand how this architecture works in practice, let's trace a sensor reading from physical measurement to digital twin application:

1. *Physical measurement*. As a garden tap is turned on and water flows through a flowmeter, spinning the blades on the impeller that have magnets attached to them. The rotating magnets create a changing magnetic field that is detected by the Hall effect sensor. This sensor outputs a signal, typically a frequency, proportional to the flow rate.
2. *Signal processing*. The sensor's electronics convert soil conductivity to a digital moisture percentage, apply calibration, and format the reading with timestamp and device identifier.

3. *Communication*. The sensor transmits a compact LoRaWAN packet containing the moisture reading, device ID, and metadata.
4. *Gateway processing*. The LoRaWAN gateway receives the radio signal, validates the message, and forwards it over the internet to AWS IoT Core.
5. *Network server processing*. The AWS LoRaWAN Network Server authenticates the device, decrypts the message, and publishes the data to an MQTT topic following the naming convention `home/water/garden/flow_meter_1`.
6. *Message decoding*. The message decoder translates the encoded message to a JSON object to be published for ingestion and storage.

This flow of data from my garden tap, to its eventual destination where it is decoded is shown in figure 3.12.

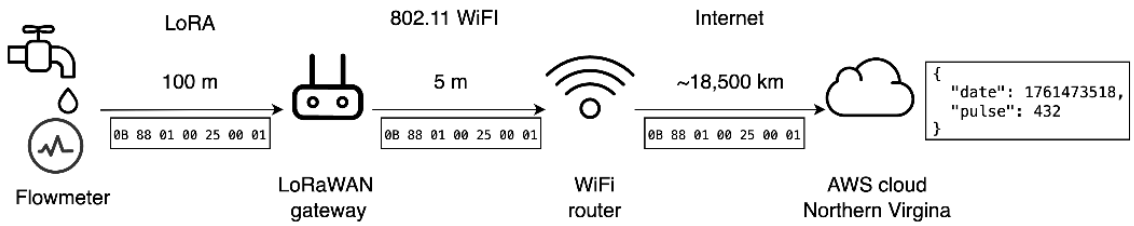


Figure 3.12 The route travelled by a reading from the LoRaWAN flowmeter attached to my garden tap.

This same flow applies whether the data originates from an electronic LoRaWAN sensor or manual pool chemistry testing—the architecture abstracts away the collection and communication details to provide a consistent interface for digital twin logic.

3.7.6 Architectural principles

Several key principles emerge from this architecture that apply to digital twin implementations at any scale.

1. *Embrace diversity*. Rather than forcing all data collection into a single technology, design systems that can accommodate multiple approaches based on practical constraints.
2. *Converge on standards*. While collection methods may vary, standardize on common protocols and data formats for transport and processing to simplify application development.
3. *Plan for evolution*. Design loosely-coupled interfaces that can accommodate new sensor types and communication methods without requiring changes to core application logic.
4. *Balance automation and pragmatism*. Automate what provides clear value, but don't hesitate to use manual collection when it's more cost-effective or reliable.
5. *Design for failure*. Assume that sensors will fail, networks will be unreliable, and data will be missing. Build resilience into every layer of the architecture.

This architecture has proven robust in my home digital twin deployment, handling everything from brief network outages to sensor failures while maintaining continuous operation. The same principles scale to industrial deployments where the stakes are higher but the underlying challenges remain the same.

3.8 Summary

- Sensors are the bridge between the physical and digital worlds, and update a digital twin with changes detected in the real world.
- There are many types of sensors with different characteristics of resolution, sensitivity and accuracy and precision.
- Consider carefully what you need to measure, the performance you require and practical constraints such as

where you must measure and what infrastructure you need when selecting sensors for your digital twin.

- Not every measurement that you need to take can be economically or practically captured by electronic sensors and manual sensing or integrating data that has been sensed by third parties is a critical aspect of capturing changes in the real world.
- Sensors need to communicate their readings at differing rates and distances and there are a variety of communication technologies to choose from.
- Sometimes you may not find the sensor that you require available off the shelf and will need to build your own.
- Operating sensors reliably at scale brings challenges related to battery management, firmware updates, calibration and physical asset tracking that can be addressed by automated health monitoring, standardization and documentation.

4 Data integration and management

This chapter covers

- The types of data typically used by digital twins
- Sources of data and how they are integrated into a digital twin
- Data storage solutions
- Managing data governance and compliance

Data is the lifeblood of a digital twin. Sensors, enterprise systems, external APIs, and human inputs all generate data that must come together in a coherent way if the twin is to reflect reality and provide actionable insights. Without careful attention to how data is collected, cleaned, combined, and stored, a digital twin risks becoming fragmented, inconsistent, or untrustworthy. Data integration and management are therefore not just technical necessities but the foundation upon which a successful digital twin is built.

We've examined how to construct digital representations of physical systems and use sensors to gather data about system changes and feed it into your digital twin. Now we look at the ways in which we can integrate data into the twin from the myriad of places it may be produced, and how we can store and manage this data so that is available in an accurate and timely manner for the twin to consume. Together, these capabilities are what allow a digital twin to evolve from a collection of raw signals into a trusted representation that supports decision-making, prediction, and optimization.

We begin by looking at the different types of data that are important in a digital twin. You'll learn how each type plays a role in shaping the twin, what challenges they pose, and how they can be combined effectively. By the end of this section, you will be able to classify digital twin data sources, understand their unique characteristics, and recognize how they influence integration strategies. This foundation sets the stage for looking at how to store, transform, and govern this data so that your digital twin remains consistent, reliable, and useful over time.

4.1 Types of data

Digital twins must handle diverse data types—both structured and unstructured—that each bring their own characteristics, storage demands, and processing requirements. Understanding these different data formats is essential when designing a digital twin architecture capable of capturing, storing, and analyzing the complex information needed to create accurate representations of physical systems.

4.1.1 Reference data

Reference data represents relatively static information about physical assets, organizational structures, and system configurations. This data provides context for interpreting operational measurements and includes asset hierarchies, device

specifications, maintenance procedures, organizational charts, and configuration parameters.

Unlike operational data that changes continuously, reference data typically remains stable for weeks, months, or years. A pump's manufacturer, model number, and installation date rarely change, while its operational parameters like flow rate and temperature vary continuously. This stability influences both storage requirements and update frequencies.

Reference data often originates from enterprise systems like ERP platforms, asset management databases, and configuration management tools. The master data management challenge lies in maintaining consistency across multiple systems that may store overlapping information about the same physical assets.

4.1.2 Timeseries data

Timeseries data represents the continuous flow of measurements, observations, and state changes that characterize system behavior over time. This operational data is what allows a digital twin to keep its digital representation of the physical system updated and accurate, enabling monitoring, trend analysis, and predictive modeling that drives automated decision-making.

Timeseries data exhibits several key characteristics:

- *Temporal ordering*—Each measurement is associated with a precise timestamp that establishes when the observation occurred, enabling analysis of trends, sequences, and causality relationships.
- *Temporal locality*—Recent data typically requires more frequent access than historical data, influencing storage and indexing strategies.
- *High frequency and volume*—Sensors produce continuous streams ranging from thousands of measurements per second in industrial equipment to periodic readings every few minutes in smart home applications. The continuous generation of data by many sensors creates large volumes of data that require specialized storage and processing approaches optimized for temporal queries and aggregation operations.
- *Immutable*—Historical measurements represent facts that rarely change once recorded, unlike transactional data that frequently updates.

Table 4.1 shows an example of timeseries data produced by a single sensor in my home that measures a range of environmental parameters, showing the characteristics that are typical of this type of data.

Table 4.1 Example of a timeseries data from a Milesight AM319 environmental sensor in my home.

Sensor Id	Timestamp	CO2	Humidity	PIR	PM10	PM25	Pressure	Temperature	TVOC
24e124710b423527	1749945584458	639	67	trigger	6	5	1016.6	16.4	1.2
24e124710b423527	1749946183884	636	66.5	idle	4	4	1016.6	16.4	1.2
24e124710b423527	1749946784664	649	66.5	idle	5	5	1016.6	16.5	1.2
24e124710b423527	1749947383975	646	66.5	idle	5	5	1016.8	16.5	1.2

4.1.3 Unstructured and semi-structured data

Object data encompasses the vast array of unstructured and semi-structured content that provides essential context and documentation for physical systems. This includes technical manuals, maintenance procedures, equipment photographs, inspection videos, engineering documents, and configuration files that don't fit into tabular or temporal formats but remain important for comprehensive system understanding and operational effectiveness.

Two examples of unstructured data that I will store in my home digital twin are shown in figure 4.1. I store photographs of my water meter taken by an IoT camera for validation and calibration of the machine learning model that runs to extract the meter reading from the image, and appliance manuals and other documents that will provide important context when I start to use AI to improve decision making.

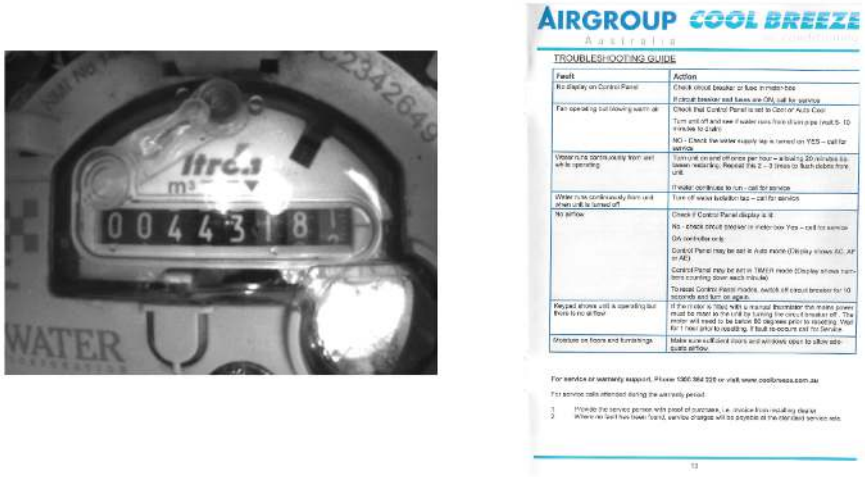


Figure 4.1 Examples of unstructured data stored in my home digital twin, with an image taken by an IoT camera shown to the left, and a device manual on the right.

4.1.4 Spatial data

Spatial data represents geographic coordinates, physical layouts, and spatial relationships that define where assets exist and how they relate to each other in 2D

and 3D space. While other data types describe what systems are doing or how they're performing, spatial data answers the question of where these activities occur.

Spatial data is indispensable for developing location-aware analytics, enabling proximity insights, and supporting geographically optimized decisions. Its applications vary by environment: within industrial facilities, it details equipment and sensor placement; across smart cities, it maps the complex network of roads, utilities, and building locations.

The complexity of spatial data varies significantly. Simple point coordinates require minimal storage and processing, while complex 3D meshes or geographic information system (GIS) data require specialized handling and substantial storage capacity.

Figure 4.2 shows a 3D mesh model of my home that I use in my home digital twin, together with the local x,y coordinates of three sensors that I have deployed, as an example of spatial data that is important in a digital twin. This 3D model serves as the spatial framework that provides geometric context for all sensor data collected throughout the house. By mapping sensor locations precisely within this 3D representation, I can correlate environmental measurements with specific physical spaces, room layouts, and architectural features.

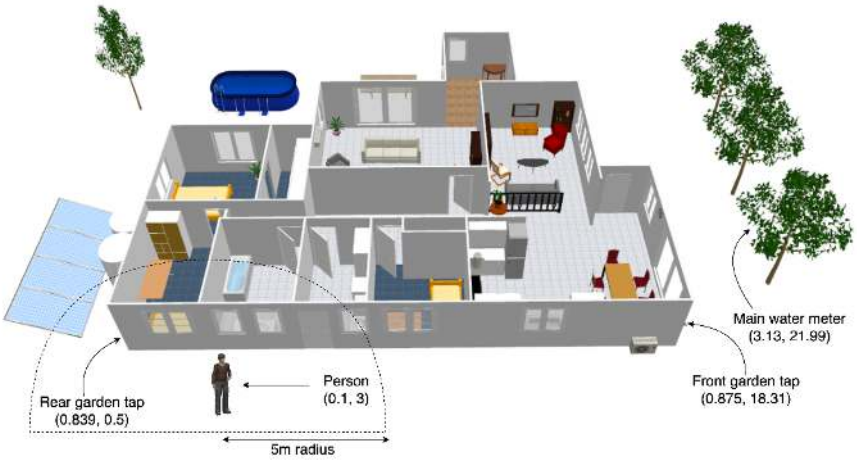


Figure 4.2 A 3D mesh model of my home that I will use in my digital twin with the coordinates displayed for three sensors I have deployed to measure water usage around the home. Listing 4.2 shows how I can use the coordinates to find objects within five meters of the person.

One of the benefits of spatial data is that it gives you the ability to reason spatially within your digital twin, just as you would in the real world. Looking at figure 4.2, given the location of the person I have added to the model, I can use a simple calculation to find sensors within a given proximity of their position. The code in listing 4.1 provides an example of this by using local coordinates to find objects within five meters of a given location using the Euclidean distance formula:

$$d\big((x_1, y_1), (x_2, y_2)\big) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This code will return the rear garden tap which is 2.6 meters from the person.

Listing 4.1 Using spatial coordinates to perform a proximity based search query to find all sensors within five meters of a given location.

```
my_pos = (0.1, 3) #1

sensors = [ #2
    {"id": "flow_meter_1", "name": "Rear garden tap", "pos":
    ➥(0.839, 0.5)},
    {"id": "flow_meter_2", "name": "Front garden tap", "pos":
    ➥(0.875, 18.31)},
    {"id": "water_main_meter", "name": "Main water meter", "pos":
    ➥(3.13, 21.99)},
]

dist = lambda p1, p2: ((p2[0]-p1[0])**2 + (p2[1]-p1[1])**2)**0.5 #3

print("Sensors within 5m:")
[print(f"- {s['name']}: {d:.1f}m") for s in sensors if (d := dist(my_pos,
➥s['pos'])) <= 5]
```

#1 My position in the local coordinate system.
#2 Sensors, and their coordinates in the same reference system.
#3 Use the Euclidean distance formula to calculate the distance in meters.

> ***TIP***
>
> The ability to spatially locate sensors and the data they produce becomes significantly more important and useful the larger the area you model, and the increased density of sensors you deploy.

4.1.5 Derived data

Derived data results from calculations, transformations, or aggregations applied to raw data sources. This processed information transforms basic measurements into actionable insights through feature engineering, statistical analysis, and machine learning techniques.

FEATURES

Features are engineered variables that extract meaningful patterns from raw sensor data. A temperature reading of 185°C becomes meaningful when transformed into features like "temperature deviation from optimal", "rate of temperature change", or "time since temperature exceeded safe operating range". These features capture relationships and contextual information that individual measurements cannot express.

Figure 4.3 shows the relationship between raw data stores and features as used both by a data scientist to train an ML model, and also for ML model serving.

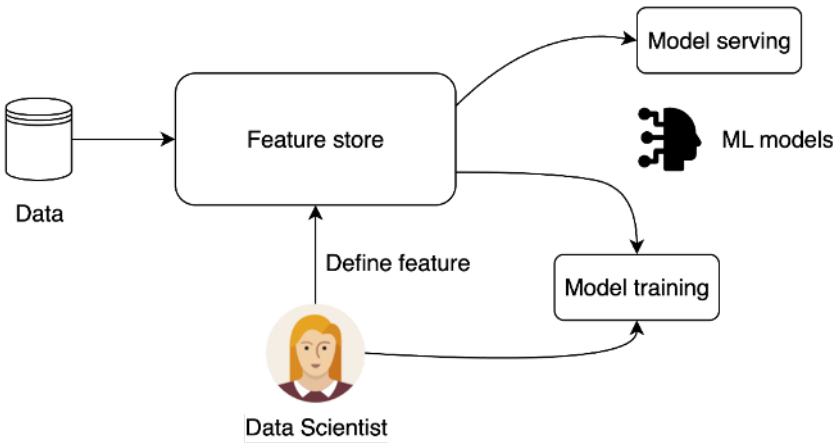
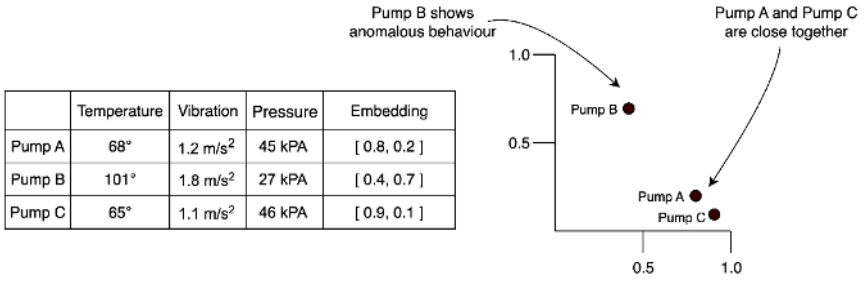


Figure 4.3 The relationship between a raw data store, derived data in a feature store, and the training and serving of ML models.

EMBEDDINGS

Embeddings are dense, numerical vector representations that encode complex patterns, relationships, and semantics from raw data into a compact, mathematically comparable format. Unlike traditional features that extract specific measurable attributes, embeddings capture semantic relationships within high-dimensional data. Figure 4.4 shows a simplified example of how raw data might be represented as a 2D vector, illustrating how sensor readings might be converted into a format where a mathematical operation (distance between two points) corresponds to a real world relationship.



	Temperature	Vibration	Pressure	Embedding
Pump A	68°	1.2 m/s^2	45 kPA	[0.8, 0.2]
Pump B	101°	1.8 m/s^2	27 kPA	[0.4, 0.7]
Pump C	65°	1.1 m/s^2	46 kPA	[0.9, 0.1]

Figure 4.4 An example of how higher dimensionality raw data can be converted to a compact format that encodes the similarity in operating parameters between Pump A and C, while showing the difference in Pump B.

4.2 Data sources

Now that we have looked at the different types of data that are used by a digital twin to create a digital representation of a physical system, we need to understand where all this data comes from.

The data used in a digital twin is typically drawn from three major domains: *operational technology (OT)* systems that monitor and control physical processes, *information technology (IT)* systems that manage business operations and enterprise

data, and external sources that provide contextual information from beyond organizational boundaries.

NOTE

While IoT devices increasingly blur the lines between IT and OT—particularly in smart building and enterprise IoT applications—we consider them in the OT context as they most commonly serve as the sensory layer for physical operations in industrial digital twins. The key distinction is whether the devices are primarily monitoring and controlling physical processes (OT) or supporting business processes (IT).

4.2.1 Operational technology (OT) data sources

Operational technology refers to the hardware and software that directly monitor and control physical devices, infrastructure, and processes commonly found in an industrial environment and are important sources of data for industrial digital twins.

SCADA AND INDUSTRIAL CONTROL SYSTEMS

SCADA and DCS systems form the backbone of industrial data collection. These systems continuously monitor equipment status, and process variables and control the system via PLCs. For a digital twin of a manufacturing line for example, SCADA provides critical timeseries data such as temperatures, pressures, flow rates, motor speeds, and valve positions. This data is stored in a data store known as a *process historian* that is designed to handle the large volume of timeseries data generated by industrial processes over time and provide for analysis of past performance and trends.

Given the high-frequency nature of SCADA data (often multiple readings per second), most digital twins import and store this data locally rather than querying SCADA systems or the process historian directly. Direct queries could impact SCADA performance, and the query interfaces often aren’t designed for analytical workloads. Instead, organizations typically deploy data collectors that subscribe to SCADA data streams, buffer the data, and forward it to a database that maintains a copy of the operational data known as a *mirror*. A digital twin typically sources data from this mirror rather than directly from the process historian itself.

INDUSTRIAL INTERNET OF THINGS (IIOT)

You may be wondering where the IoT sensors that we have looked at fit into the operational technology landscape. While IoT sensors enable increasingly extensive monitoring of the physical environment, my LoRaWAN flow meter is not part of an OT system. There are some key differences between OT systems and IoT sensors:

- *Connectivity* - OT systems traditionally operate on proprietary networks that are isolated from the internet for security and reliability, while IoT devices normally connect via the public internet.

- *Core function* - OT systems are designed for the direct, real-time management and control of physical systems whereas IoT devices are primarily responsible for data gathering, communication, and non real-time control.

Despite these differences the lines between OT and IoT are beginning to blur with IoT devices increasingly being used to supplement operational decision making in OT systems via the *industrial internet of things (IIoT)*. IIoT devices are connected devices specifically designed for industrial environments and are characterized by their rugged build, interoperability with legacy industrial control systems, and secure design.

4.2.2 Information technology (IT) data sources

While OT systems provide the real-time operational data that drives digital twin monitoring and control, IT systems contribute the business context and enterprise intelligence that transform raw operational data into actionable insights. IT systems often support more flexible access patterns than OT systems, allowing digital twins to choose between in-place access and import based on specific requirements.

ENTERPRISE RESOURCE PLANNING

In large corporations *enterprise resource planning (ERP)* systems like SAP contain critical master data about assets, maintenance schedules, procurement records, and organizational hierarchies. This data provides essential context for interpreting operational measurements—knowing that a pump was installed last month explains why its vibration patterns differ from historical baselines. ERP data typically changes slowly and benefits from in-place access through well-established APIs and database connections. Listing 4.2 shows how you can query SAP via an ODATA API to get back sales data. To try this out yourself, you can sign up at https://api.sap.com/ and get an API key that you can use on the publicly available SAP API sandbox.

Listing 4.2 An example of accessing data from a public sandbox of SAP, a common ERP system, to retrieve sales orders.

```
import requests

url = "https://sandbox.api.sap.com/s4hanacloud/sap/opu/
➥odata/sap/API_SALES_ORDER_SRV/A_SalesOrder" #1
headers = {
  "APIKey": "your_api_key_here", #2
  "Accept": "application/json"
}

params = {
  "$top": "5",
  "$select": "SalesOrder,SoldToParty,SalesOrderDate,TotalNetAmount",
}

response = requests.get(url, headers=headers, params=params)

if response.status_code == 200:
  orders = response.json()["d"]["results"]
  print(f"{len(orders)} Sales Orders from SAP:")

  for order in orders:
    so_id = order.get("SalesOrder", "N/A")
    customer = order.get("SoldToParty", "N/A")
    amount = order.get("TotalNetAmount", "N/A")
    print(f"  {so_id}: Customer {customer} - ${amount}")
else:
  print(f"Error: {response.status_code}")
```

#1 Use the free SAP sandbox API and target the sales order API.
#2 Specify your API key as a header.

> ***TIP***
>
> Being able to link data about orders and sales to operational data from sensors allows you to start to answer questions from customers about the quality of goods you have delivered to them for example.

FINANCE, PROCUREMENT, AND BUSINESS INTELLIGENCE

Integrating with financial and procurement systems lets a digital twin factor costs into operational decisions. Understanding financial data such as payment terms and cash conversion cycles can enable, for example, recommendations for inventory levels that will minimize working capital while maintaining operational performance. An autonomous digital twin may automatically place an order for a replacement part from a supplier based on operational data from sensors.

Many organizations have existing investments in business intelligence and analytics tools that contain pre-processed metrics, KPIs, and business rules that digital twins can leverage rather than recreating. These platforms may already compute equipment utilization rates, energy efficiency metrics, or maintenance cost analyses that provide valuable inputs for digital twin algorithms and simulations.

4.2.3 The convergence of IT and OT

Digital twins that are accessed by end users to inspect, simulate, and predict a physical system's behaviors most often execute within the IT network, but rely on data from OT systems. Consider using Google Maps to look at traffic congestion on your route home. You would typically use a web browser to access the map view. The data that controls the congestion coloring within the map may be sourced from a traffic management system such as the *Sydney coordinated adaptive traffic system (SCATS)* which is an OT system deployed in many cities across the world to manage and control traffic signals. Given the threats posed to IT systems (think of viruses and malware that you have encountered on personal computers), these cannot be connected directly to an OT system. Imagine a virus from your home computer infecting the city traffic control system.

CHALLENGES IN OT DATA INTEGRATION

Working with OT data presents unique challenges in simply accessing the data, with industrial systems often operating on physically isolated ('air gapped') networks with strict security requirements to protect physical assets. Many organizations use data diodes or unidirectional gateways that only allow data to flow out of OT networks, making real-time querying impossible and forcing digital twins to work with replicated data.

> ***NOTE***
>
> The Stuxnet computer virus shows how dangerous it can be when industrial control systems are compromised. Considered the worlds first digital weapon, this virus targeted Siemens S7-417 PLCs in SCADA systems and was reportedly responsible for the destruction of centrifuges within Iranian nuclear enrichment facilities in 2009.

OT environments use diverse proprietary protocols that evolved independently across industrial domains. A single facility might use Modbus, Profibus, Ethernet/IP, and OPC-UA simultaneously. Many legacy systems only support polling-based retrieval or require complete dataset downloads rather than selective queries, forcing digital twins to import massive amounts of data simply because source systems cannot filter at the protocol level.

Direct querying of OT systems also risks operational continuity. Industrial control systems prioritize deterministic responses over analytical workloads, and even lightweight queries can introduce dangerous latency. Process historians often lack modern query optimization, causing performance degradation when supporting both real-time ingestion and analytical queries.

These constraints typically force digital twin architectures to implement data mirroring strategies, where collectors replicate OT data to separate analytical environments. While this introduces latency and storage overhead, it protects operational systems while providing digital twins access to rich industrial data.

ARCHITECTURAL PATTERNS FOR CONVERGENCE

The industrial *demilitarized zone (DMZ)* has become the standard pattern for connecting the IT systems, where digital twin applications normally live, to the OT network. Just like a DMZ in military terms, it provides a neutral zone between the trusted OT network and the hostile, untrusted IT network. This zone hosts systems that both domains can access without directly connecting including:

- *Mirror servers* - replicate OT application interfaces for IT access without touching OT systems.
- *Protocol converters* - translate between OT protocols such as Modbus and IT protocols such as MQTT.
- *Security gateways* - inspect and filter traffic, ensuring only authorized data flows between domains.

Figure 4.5 shows a common architecture for IT/OT convergence, known as the *Purdue model*, which organizes industrial systems into hierarchical levels from field devices up to enterprise systems, enabling secure and efficient data flow while maintaining operational safety and security.

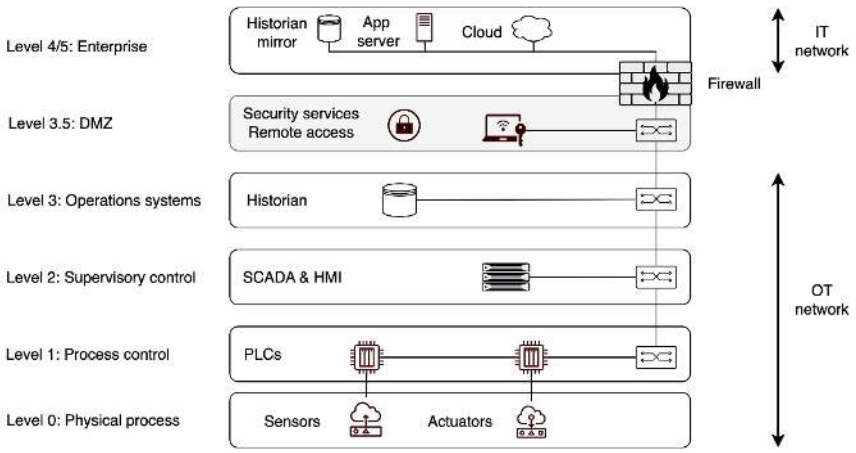


Figure 4.5 The Purdue model of IT/OT convergence segments the network into layers with a DMZ acting as a buffer zone between the OT and IT networks.

4.2.4 External data sources

Digital twins frequently transcend the confines of a single organization, relying heavily on data that originates from external sources beyond your direct control. These external data streams can range from third-party APIs and public datasets to partner organizations' systems and cloud-based services. In many cases, external systems serve as the primary, or even exclusive, source of important information for your digital twin implementation. More commonly, external data supplements and enriches your internal datasets, providing valuable contextual information that enhances the accuracy and comprehensiveness of your digital twin models.

In section 4.4 we will look at different ways that external data can be integrated into your digital twin.

4.3 Data structures

How you structure data for storage and processing significantly impacts query performance, storage efficiency, and analytical capabilities. The architectural decisions you make about data organization—whether to prioritize row-based or columnar storage, how to partition datasets, what indexing strategies to employ, and how to handle schema evolution—will determine not only how quickly you can retrieve information but also what types of analysis become feasible at scale. Different structural approaches optimize for different access patterns and use cases, creating trade-offs between write performance and read efficiency, storage compression and query flexibility, or real-time processing and historical analysis.

4.3.1 Relational structures

Relational structures organize data into normalized tables with defined relationships between entities. This approach excels at maintaining data consistency, supporting complex queries across multiple entities, and handling updates to discrete records.

The relational model's strength lies in query flexibility—you don't need to anticipate every possible question when designing the schema. Unlike other approaches that optimize for specific access patterns, relational databases excel at ad-hoc analytical queries that weren't considered during initial design. This flexibility becomes invaluable as digital twin requirements evolve and new analytical needs emerge.

Figure 4.6 shows a simplified example of how maintenance tasks for an industrial asset can be represented as a relational data model, with each table holding rows representing instances of an entity and key references between the tables representing connections between these entities. In this way, however many workorders and tasks are created for an asset, there will only ever be one asset record, with all other entities related to it.

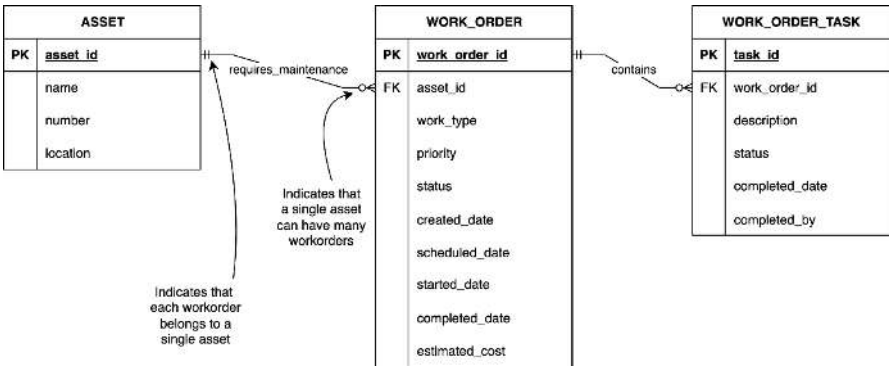


Figure 4.6 A relational model of a work order showing tables and properties representing entities, and their relationships.

This type of data store allows ad-hoc analytical queries to be executed, such as "show me all the maintenance tasks performed on-time by John Smith on the printing press with identifier F1" as shown in listing 4.3. The database for this example is provided alongside the code in GitHub.

Listing 4.3 An example ad-hoc query written using SQL against data in a SQLite database with the schema shown in figure 4.6.

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('maintenance.db') #1

query = '''
    SELECT WT.DESCRIPTION, WT.COMPLETED_DATE, WT.COMPLETED_BY
    FROM WORK_ORDER_TASK WT, WORK_ORDER W, ASSET A
    WHERE WT.STATUS = 'COMPLETED'
    AND WT.COMPLETED_BY = 'John Smith'
    AND WT.COMPLETED_DATE <= W.SCHEDULED_DATE
    AND WT.WORK_ORDER_ID = W.WORK_ORDER_ID
    AND W.ASSET_ID = A.ASSET_ID
    AND A.asset_id = 'F1'
'''

print("Tasks completed by John Smith on/before scheduled date for asset F1:")
results = pd.read_sql_query(query, conn) #2

print(results.to_string(index=False))

conn.close()
```

#1 SQLite stores its data in a file on the local filesystem.
#2 Pandas takes SQL query string and a connection to the database, and loads the results of the query to a dataframe.

Many software applications that model physical objects and systems use relational data models due to the integrity and consistency such a model enforces on the data as well as the flexibility in querying them via a standardized query language. You can use a relational model to represent data that benefits from ACID compliance and transactional integrity. If your digital twin integrates with other systems such as ERP, finance, or supply chain systems, many of these store their data in a relational model, and you will often find yourself querying this type of data.

4.3.2 Columnar structures

Columnar storage organizes data by measurement type (column) rather than by individual records (rows), which contrasts with traditional row-based databases that store complete records together.

Columnar storage offers several key advantages for analytical workloads and data processing. By storing data column-wise, it enables superior compression ratios since similar data types are grouped together, allowing compression algorithms to work more effectively. This structure dramatically improves query performance for analytical operations that typically access only a subset of columns—instead of reading entire rows, the system can retrieve just the specific columns needed for analysis. Columnar storage also optimizes for aggregation operations like sums, averages, and counts, as these calculations can be performed directly on compressed column data without decompressing entire datasets. Additionally, this approach supports efficient parallel processing, where multiple columns can be processed simultaneously across different CPU cores, making it particularly well-suited for large-scale data analytics and reporting scenarios.

The way in which timeseries data is stored as rows is shown as follows:

```
Row 1: [2024-01-15 14:30:00, temp=22.5, humidity=65, pressure=1013.2]
Row 2: [2024-01-15 14:31:00, temp=22.7, humidity=64, pressure=1013.1]
Row 3: [2024-01-15 14:32:00, temp=22.6, humidity=66, pressure=1013.3]
```

And how the same data would be stored in a columnar format:

```
Temperature:  [22.5, 22.7, 22.6, 22.8, 22.4, ...]
Humidity:     [65, 64, 66, 65, 67, ...]
Pressure:     [1013.2, 1013.1, 1013.3, 1013.0, ...]
Timestamps:   [14:30:00, 14:31:00, 14:32:00, ...]
```

4.3.3 Graph structures

Graph structures represent entities and their relationships as nodes and edges, providing natural models for interconnected systems. Digital twins often need to understand complex relationships between assets, locations, systems, and processes that graph structures handle more intuitively than relational models.

Graph structures excel at traversal queries that follow relationships across multiple entities. Finding all sensors within a building, identifying equipment dependencies, or tracing supply chain relationships becomes straightforward with graph traversal algorithms. Figure 4.7 shows a simple example of how a graph structure is used to represent a house, it's rooms, and their devices. In this example finding the smart bulbs in the house at 742 Evergreen Terrace involves finding the node that represents the address, then traversing the outbound edges with the label `hasRoom`, to get the rooms of the house, followed by traversing the outbound edges with label `hasDevice`, to find the devices, and finally looking for the nodes of type `Smart Bulb`.

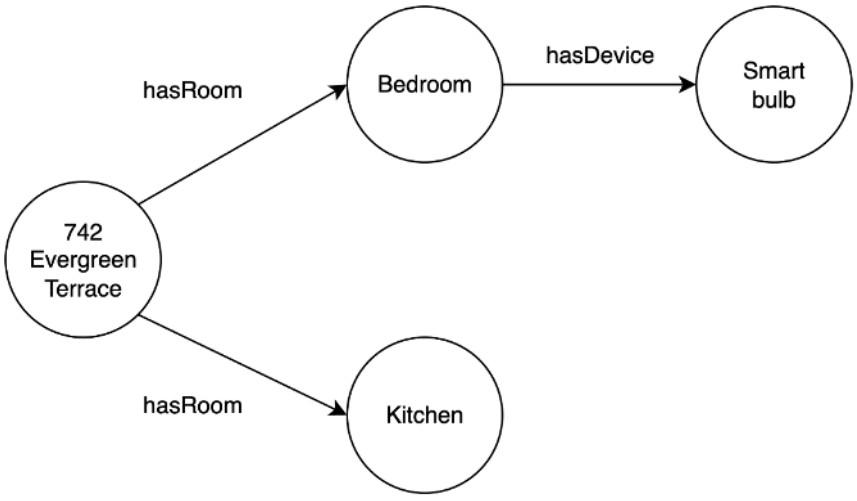


Figure 4.7 A house, its rooms, and their devices modelled as a graph structure.

4.3.4 Document structures

Document structures store data as flexible, schema-less documents that can contain nested structures and variable attributes. This approach works well for semi-

structured data from APIs, configuration files, or sensor data where different device types report different sets of measurements. A simple example of how we might model a house, it's rooms, and their devices as a document record is:

```
{"id": "_496756956732423",
  "name": "742 Evergreen Terrace"
  "rooms": [
    { "name": "Bedroom",
      "area": 10,
      "devices": [
        { "name": "Smart bulb"
          "address": "00:1B:63:84:45:E6"
        }
      ]
    },
    { "name": "Kitchen",
      "area": 20
    }
  ]
}
```

Document structures provide schema flexibility that accommodates evolving data requirements without requiring database migrations. However, this flexibility can complicate queries that span multiple documents or require joins across different data types. In the preceding example, finding all devices in a given home would be easy, but finding what home a particular device is in would be significantly harder.

4.3.5 Key-value structures

Key-value structures provide the simplest data model, storing data as pairs of unique identifiers and their associated values. This approach optimizes for high-performance lookups and scales horizontally across distributed systems.

While key-value structures sacrifice query flexibility, they excel at use cases with predictable access patterns and high-performance requirements. Caching frequently accessed reference data or storing session information are common key-value applications.

4.4 Data ingestion

While accessing data in place offers simplicity and reduces duplication, digital twins often require dedicated data ingestion to meet their operational and analytical requirements. The decision to import rather than query external systems may be driven by the need to transform and enrich source data, run advanced analytics that might overwhelm transactional systems, or maintain operational independence from source systems. Lets look at some of the ways in which a digital twin can ingest data.

4.4.1 Batch data ingestion

Batch ingestion processes data in discrete chunks at scheduled intervals, hourly, daily, or on-demand. Batch ingestion is popular for slow moving enterprise data and offers advantages in reliability, cost efficiency, and processing complex transformations. It is an important mechanism for a digital twin because it efficiently

handles large volumes of historical, contextual, and less time-sensitive data, providing a comprehensive and stable foundation for analysis, modeling, and long-term insights. This type of data is used by digital twins to understand past behavior in order to predict future outcomes.

TRY IT OUT: INGEST MILLIONS OF TAXI TRIPS TO MODEL CITY TRANSPORTATION

Imagine that you are building a city scale digital twin of New York. There are over 13,000 iconic yellow cabs in the city making millions of trips per year. Understanding these taxi movements in the real world enables your digital twin to gain insight into the functioning of the city's transportation network, enabling congestion management, infrastructure planning, and simulation and testing of new traffic management strategies. The New York City Taxi and Limousine Commission provides open taxi trip record data published monthly at https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page. We can ingest this data in monthly batches, cleansing it on ingestion, and store it in a relational database within our digital twin. Once the data has been ingested, we can run analytical queries over it, or use it to retrain a demand prediction ML model for each monthly batch.

Listing 4.4 shows this in action by downloading the June 2025 yellow cab trip data, performing some simple cleansing and loading it to a SQLite database for analysis. Through this analysis we learn that this month nearly four million cab trips were made, with an average distance of 3.6 miles.

Listing 4.4 Batch ingestion of the close to four million yellow cab journeys made in June 2025 for analysis via SQL.

```
import pandas as pd
import sqlite3, requests

db = sqlite3.connect('taxi.db') #1
db.execute('''CREATE TABLE IF NOT EXISTS trips (
    pickup_time TEXT, distance REAL, fare REAL, tip REAL, total REAL)''')

def download(): #2
  url = 'https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_
  ➥2025-06.parquet'
  with open('taxi.parquet', 'wb') as f:
    f.write(requests.get(url).content)

def process(): #3
  df = pd.read_parquet('taxi.parquet')
  clean = df[
    (df['fare_amount'] > 0) & (df['fare_amount'] < 500) &
    (df['trip_distance'] > 0) & (df['trip_distance'] < 100)
  ][['tpep_pickup_datetime', 'trip_distance', 'fare_amount', 'tip_amount',
  ➥ 'total_amount']]

  clean.columns = ['pickup_time', 'distance', 'fare', 'tip', 'total']

  clean.to_sql('trips', db, if_exists='append', index=False) #4
  print(f"Complete: {len(clean):,} trips loaded")

def summarize(): #5
  stats = pd.read_sql_query("""
    SELECT COUNT(*) trips, ROUND(AVG(distance),1) avg_miles,
      ROUND(AVG(fare),2) avg_fare, ROUND(SUM(total)) revenue
    FROM trips""", db)
  print("\n", stats.to_string(index=False))

download()
process()
summarize()
```

#1 Create a SQLite database to store the data.
#2 Download the data for June 20025.
#3 Load the Parquet file to a dataframe and remove records with no distance or fare.
#4 Write the cleansed data to the SQLite database.
#5 Use a SQL query to summarize the data.

4.4.2 Streaming data ingestion

Streaming data ingestion involves the continuous flow of data into a digital twin enabling it to be a dynamic and live representation of its physical counterpart. When working with streaming data, you often don't just want raw sensor readings, but trends. A common technique is to compute aggregates such as averages, minimums, or maximums over a sliding window of recent data. Instead of waiting for a batch to complete, the twin continuously updates these values as new data points arrive.

For example, suppose a temperature sensor emits one reading every second. Rather than acting on every single value, you could maintain a 30-second sliding average that smooths out noise while still responding quickly to changes. As each new

reading arrives, the oldest one falls out of the window, and the average is recalculated. Figure 4.8 shows how a sliding window over a data stream works.

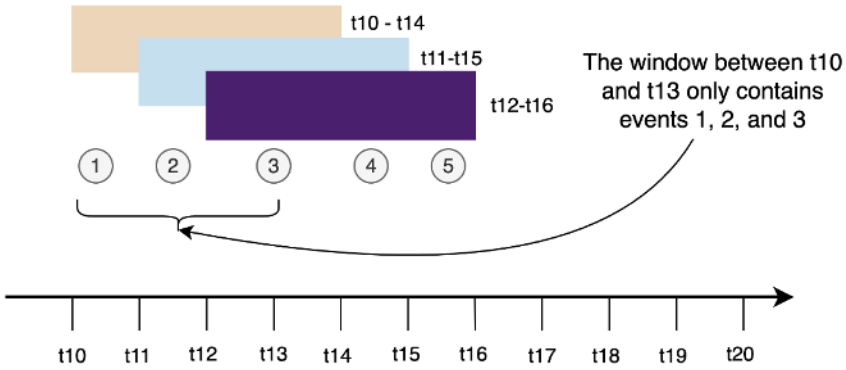


Figure 4.8 A sliding window of length four over a data stream. As the window slides through time, older readings fall out, and new readings are added.

This kind of in-stream processing is valuable because it reduces noise in high-frequency signals, preserves timeliness by avoiding long batch delays, and enables real-time alerts when trends cross defined thresholds.

TRY IT OUT: STREAM LIVE TRAIN MOVEMENT DATA

The National Rail network in the UK sees over 1.5 billion passenger journeys yearly and is a key part of the British economy. A recent report from the Department for Transport estimates an £850 million potential economic benefit from an integrated network management digital twin through reduced congestion, journey optimization and maintenance management. As you can imagine, understanding the state of trains on the physical network at any point in time would be an important part of a digital representation of an integrated transport network. The operator of the rail network across the UK publishes a range of open data feeds including train movement data that provides a real time stream of train positioning and movement event data across the network. You can easily subscribe to this public stream of data and try ingesting it into your own application to get an idea of ingesting streaming data.

We can add a sliding window over the stream as we read it to calculate the percentage of train movements that are late in a five minute period as shown in listing 4.5. GitHub includes the full code and instructions on how to run this example.

Listing 4.5 Calculating the percentage of late train movements across the UK rail network in a five minute sliding window.

```
cutoff = datetime.now() - timedelta(minutes=5) #1
while window and window[0][0] < cutoff: #2
  window.popleft()

for m in (json.loads(frame.body) if isinstance(json.loads(frame.body), list)
➥ else [json.loads(frame.body)])[:5]:
  msg_type = m.get('header',{}).get('msg_type','')
  body = m.get('body',{})

  if msg_type == '0003':  #3
    status = body.get('variation_status','')
    train_id = body.get('train_id','')
    print(f"{status} {train_id}")
    window.append((datetime.now(), train_id, status == 'LATE'))

    count += 1
    if count % 25 == 0:  #4
      late_count = sum(1 for _, _, is_late in window if is_late) <>
      on_time_rate = (len(window) - late_count) / len(window) * 100 if
      ➥window else 100
      print(f"5min window: {len(window)} movements, {on_time_rate:.1f}%
      ➥on-time")
```

#1 Set the window to be five minutes in length.
#2 Remove events from the window that are older than five minutes.
#3 Only process train movement events.
#4 Calculate aggregates every 25 events.

4.4.3 API integration

APIs provide structured, on-demand access to data from both internal enterprise systems and external services, making them essential components of digital twin data ingestion strategies. Unlike batch or streaming ingestion that push data on predetermined schedules, APIs enable digital twins to pull specific data when needed, supporting both scheduled collection and event-driven integration patterns. Digital twins typically integrate with:

- *Representational state transfer (REST)* APIs are widespread across many systems due to their simplicity and provision of predictable, cacheable access to well-defined resources.
- *GraphQL* APIs allow a digital twin to request exactly the data that it needs in a single query which is valuable when integrating with systems that expose rich data models when you only need a few attributes.
- *gRPC* APIs provide high-performance, type-safe communication particularly valuable for real time operations. It's binary protocol and code generation from schema definitions reduce integration complexity while improving performance over HTTP-based alternatives.
- *WebSocket* APIs enable bidirectional, real-time communication between systems. Unlike request-response patterns, WebSockets maintain persistent connections that support instant data updates and control commands.

TRY IT OUT: ACQUIRE SATELLITE IMAGERY FROM A REST API

The NASA visible infrared imaging radiometer suite (VIIRS) instrument is a satellite mounted sensor that collects imagery of the earth's surface for monitoring and investigating changes in surface vegetation, and water and land use, taking daily images. You can understand how this information would be useful to a digital twin that needs to represent geophysical properties of part of the earth. NASA publishes a RESTful API as part of its global imagery browse service (GIBS) service, that you can call to get an image of the earth. The sample code in listing 4.6 retrieves an image, taken by VIIRS, of the largest man-made lake in the world which allows tracking the impact of drought on this important reservoir.

Listing 4.6 Use the NASA GIS RESTful API to retrieve an image of lake Kariba taken by VIIRS from the previous day.

```
import requests
from datetime import datetime, timedelta

bbox = (26.5, -18.5, 29.0, -16.0) #1
yesterday = (datetime.now() - timedelta(1)).strftime('%Y-%m-%d')

params = {
    'SERVICE': 'WMS',
    'VERSION': '1.1.1',
    'REQUEST': 'GetMap',
    'LAYERS': 'VIIRS_SNPP_CorrectedReflectance_TrueColor',
    'SRS': 'EPSG:4326',
    'BBOX': f'{bbox[0]},{bbox[1]},{bbox[2]},{bbox[3]}',
    'WIDTH': '1024',
    'HEIGHT': '640',
    'FORMAT': 'image/png',
    'TIME': yesterday,
    'TRANSPARENT': 'true'
}

response = requests.get( #2
  'https://gibs.earthdata.nasa.gov/wms/epsg4326/best/wms.cgi',
  params=params)

filename = f'lake_kariba_{yesterday}.png'
with open(filename, 'wb') as f:
    f.write(response.content)
```

#1 Define the area of interest on the surface of the earth as a bounding box using EPSG:4326 (WGS*4) coordinates.
#2 Query the RESTful API with an HTTP GET request.

4.4.4 ETL and data transformation

Extract, transform, and load (ETL) processes bridge the gap between diverse source systems and standardized digital twin storage. ETL operations ensure data consistency regardless of source format, apply business rules and validation, and aggregate and optimize data for analytical queries.

The transformation stage handles data cleansing, type conversion, aggregation, and enrichment. Raw sensor readings might be validated against expected ranges, converted to standard units, and enriched with contextual information from reference data systems.

TRY IT OUT: USE APACHE SPARK TO AGGREGATE AND TRANSFORM POWER CONSUMPTION DATA

The optical sensor I fitted to my main electrical meter in chapter 3 allows me to export minute by minute power consumption data as CSV. I would like to aggregate that data into weekly consumption and cost for a home dashboard. Rather than running aggregation queries every time the dashboard loads, I can use Apache Spark to transform and aggregate the raw CSV data once, as shown in listing 4.7. Apache Spark's distributed processing capabilities ensure this approach scales to very large data sets.

Listing 4.7 An example of using Apache Spark to extract minutely power consumption readings, convert their units, and aggregate to weekly totals.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import date_trunc, sum as spark_sum, avg, col
from pyspark.sql.types import *

spark = SparkSession.builder.appName("WeeklyUsage").getOrCreate()

schema = StructType([ #1
    StructField("datetime_utc", TimestampType(), True),
    StructField("datetime_local", TimestampType(), True),
    StructField("watt_hours", DoubleType(), True),
    StructField("cost_dollars", DoubleType(), True),
    StructField("is_peak", BooleanType(), True)
])

df = spark.read.option("header", "true").schema(schema).csv(
  ➥"powerpal_data.csv")

weekly_usage = df.groupBy( #2
    date_trunc("week", col("datetime_local")).alias("week_start")
).agg(
    (spark_sum("watt_hours") / 1000).alias("total_kwh"),
    spark_sum("cost_dollars").alias("total_cost_dollars")
).orderBy("week_start")

print("Weekly Usage (kWh) and Cost:")
weekly_usage.show()

spark.stop()
```

#1 Define the structure of the data stored in the CSV file.
#2 Spark's SQL functions group and aggregate the data in the CSV file.

Modern *ELT* approaches load raw data first and perform transformations within the target system. This pattern works well with cloud data warehouses that provide powerful transformation capabilities and can handle raw data volumes efficiently.

4.5 Storage solutions

Selecting the right storage solution is a key factor in the success of your digital twin meeting your objectives. Digital twins require diverse data types, from real-time sensor readings and historical logs to design blueprints and simulation outputs, each

with different storage requirements for capacity, performance, scalability, and security.

These requirements are often in conflict with one another. Real-time streams need high-throughput writes and low-latency access, while historical analytics require efficient bulk reads across years of data. Metadata benefits from relational integrity, media files need object storage for large files, and ML models require feature stores for scalable predictions and retraining. No single storage technology handles all requirements effectively. Modern digital twin architectures use *polyglot persistence*, deploying multiple storage systems optimized for specific data types and access patterns. This maximizes performance while controlling costs, though it requires understanding each storage technology's strengths and limitations.

4.5.1 Timeseries databases

Timeseries databases form the backbone of many digital twins, being optimized for the continuous stream of measurements about the changing nature of the physical system that arrive over time. There are many specialized timeseries databases available today, including InfluxDB, Kdb+, and Amazon Timestream, which is available as a managed service within AWS. TimescaleDB is an open source extension to the PostgreSQL relational database optimized for timeseries data. Regardless of the specific product these databases share a set of core features:

- *Columnar storage* - organizes data by measurement type rather than by time, enabling efficient compression and analytical queries across timeseries data.
- *Compression* - timeseries data has a number of characteristics that make it easy to compress, including the fact that timestamps always increase and often at regular intervals, consecutive sensor readings may stay the same for a period of time, or only change by a small amount and sometimes gaps in data collection lead to many null values. Techniques such as *delta encoding* stores only the difference between consecutive values, rather than the values themselves. For example, if temperature readings are `25.1, 25.3, 25.2, 25.4`, storing `25.1, +0.2, -0.1, +0.2` might use less space if the deltas are smaller and can be represented with fewer bits.
- *Aggregation* - automatically creates summary statistics at multiple time granularities, transforming raw data using common statistical and temporal functions (mean, min, max) without expensive real-time calculations. Instead of computing daily averages from millions of individual readings, aggregated data provides pre-calculated summaries that enable instant dashboard updates and historical analysis.
- *Time-based partitioning* - distributes data across storage tiers based on age, keeping recent data in high-performance storage while moving historical data to cheaper options.

4.5.2 Analytical data storage

Analytical query patterns in digital twins typically involve scanning large datasets to identify patterns, correlations, or anomalies across multiple dimensions. These queries differ from operational queries that retrieve specific records by key or recent time ranges. For example, analyzing the relationship between equipment age,

maintenance frequency, and failure rates requires scanning years of maintenance records, device specifications, and sensor readings. Storing data in a columnar approach as specialized timeseries databases do also provides significant advantages for analytical queries where only specific columns need to be read rather than entire records.

ANALYTICAL DATABASES

Snowflake, Amazon Redshift, and Google BigQuery are prominent examples of cloud-native columnar data warehouses. They are purpose-built for analytical workloads, meaning they are optimized for complex queries, reporting, and business intelligence, rather than high-volume, low-latency transactional processing.

> ***NOTE***
>
> The separation of storage and compute in modern cloud analytical databases represents one of the most important innovations in cloud databases allowing independent scaling of each resource, reducing costs and improving flexibility. You can scale compute resources for complex queries while maintaining cost-effective storage for large datasets.

The main differences between these and dedicated timeseries databases (like InfluxDB or TimescaleDB) lies in their broader scope. While timeseries databases are focused on temporal data patterns and specific optimizations for time-based queries and compression, analytical data warehouses are designed to handle a wide variety of data types and structures, including structured relational data, semi-structured data, and of course timeseries data from other sources.

COLUMNAR FILE FORMATS

Specialized analytical databases like Redshift offer advantages such as complex SQL query capabilities (window functions, common table expressions), dedicated compute resources, and optimized query planning. However, if your workload doesn’t require these features, the additional cost may not be justified. If you have the need to perform occasional ad-hoc analytical queries from your digital twin, a file based columnar data store such as Apache Parquet is a cost efficient way to store data and still query it via SQL or Python. Another benefit of Parquet is that it is an open standard allowing you to move data between cloud providers or on-premises without vendor lock-in or data migration complexity. Listing 4.8 uses DuckDB to query taxi cab data in Parquet format using SQL, without the need to load the data to a database first.

Listing 4.8 An example of using DuckDB and SQL to directly query data stored in a file in Parquet format with needing to load the data into a database first.

```
import duckdb, requests, time

url = "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_
➥2024-01.parquet"
with open('taxi.parquet', 'wb') as f: #1
  f.write(requests.get(url).content)

start_time = time.time()
result = duckdb.query(""" #2
  SELECT
      COUNT(*) as trips,
      ROUND(AVG(fare_amount), 2) as avg_fare,
      ROUND(MIN(fare_amount), 2) as min_fare,
      ROUND(MAX(fare_amount), 2) as max_fare
  FROM 'taxi.parquet'
""").fetchone()
query_time = time.time() - start_time

print(f"\nResults: {result[0]:,} trips, avg ${result[1]},
➥range ${result[2]}-${result[3]}")
```

#1 Get a Parquet file containing taxi trip data from January 2024.
#2 Use DuckDB to query the Parquet file using standard SQL.

4.5.3 Data lakes and lakehouse architectures

While columnar analytical storage excels for structured data, digital twins generate diverse formats—JSON sensor logs, images, PDFs, and ML model outputs—that don't fit predefined schemas. Traditional analytical databases struggle with this heterogeneity, driving organizations toward data lake architectures. Data lakes shift from "schema-on-write" to "schema-on-read" approaches, preserving raw data in native formats and applying structure only when needed for analysis. This provides flexibility for diverse data types while maintaining cost-effective storage.

Traditional data lakes store diverse data effectively but lack performance and ACID transaction support required for analytical applications. Lakehouse architectures address these gaps by adding database-like capabilities to object storage, combining data lake flexibility with data warehouse performance and reliability. Data lakes often store data in an optimized columnar file format such as Apache Parquet, in an object store like S3, but to modify these files requires expensive read, modify, write cycles. Lakehouse storage technologies enable data lakes, using low cost and virtually unlimited object storage, to behave like data warehouses, supporting transactions such as merge and upsert.

TRY IT OUT: ACID TRANSACTIONS WITH DELTA TABLES

In a traditional data lake that uses Parquet for storage, it is not possible to update data (for example to add a new column) due to the fact that Parquet is immutable. Listing 4.9 shows how delta tables, a lakehouse storage technology, allows data to be updated (this example uses your local disk to store Parquet files). The delta table removes the need to read the Parquet file to memory, make updates, and write it

back to disk when the data is updated. Please note the sample here is truncated for brevity, but the full code is available in GitHub.

Listing 4.9 An example of using delta tables to provide ACID transactions to data stored in Parquet.

```
import pandas as pd
from deltalake import DeltaTable, write_deltalake

def main():
  table_path = "./delta_demo/sensor_data"

  df1 = pd.DataFrame( #1
      {
          "sensor_id": ["temp_001", "temp_002", "humidity_001"],
          "location": ["factory", "warehouse", "factory"],
          "value": [23.5, 18.2, 65.4],
          "timestamp": pd.to_datetime(["2024-01-15 10:30:00"] * 3),
      }
  )

  write_deltalake(table_path, df1) #2
  print("Created Delta table with columns:", list(df1.columns))

  dt = DeltaTable(table_path)

  df_update = pd.DataFrame( #3
      {
          "sensor_id": ["temp_001"],
          "location": ["factory"],
          "value": [24.1],  # Updated temperature
          "timestamp": pd.to_datetime(["2024-01-15 12:00:00"]),
          "unit": ["celsius"],
      }
  )

  dt.merge( #4
      df_update,
      predicate="target.sensor_id = source.sensor_id",
      source_alias="source",
      target_alias="target",
  ).when_matched_update_all().when_not_matched_insert_all().execute()
```

#1 Define some dummy sensor data in a dataframe.
#2 Write the sensor data to a delta table, stored as parquet on the local filesystem.
#3 Update one of the attributes for sensor temp_001.
#4 Merge the updates back to the delta table.

4.5.4 Transactional data storage

Transactional databases manage reference data, configuration information, and operational records that require ACID properties and complex relational queries. These systems handle discrete updates, maintain data consistency, and support the structured data that defines digital twin entities. In contrast to analytical data stores that are designed for efficient querying and processing of large datasets to support reporting and analysis, transactional data stores optimize for fast, consistent data input and modification.

- Relational databases remain essential for reference data, asset hierarchies, and operational records that benefit from normalized schemas and referential integrity. Modern cloud databases provide managed services that reduce operational overhead while maintaining SQL compatibility.
- NoSQL databases emerged to address limitations of relational databases when handling large-scale, diverse data with flexible schemas and horizontal scaling requirements. Digital twins often work with just this type of data from high-volume sensor streams, varying data structures across device types, and unpredictable scaling demands as systems grow from pilot projects to enterprise deployments.

4.5.5 Specialized storage systems

Digital twins often require specialized storage capabilities for specific data types and use cases:

- *Object storage* provides scalable, cost-effective storage for unstructured data including images, videos, documents, and large files. Cloud object storage services offer multiple storage tiers, automatic lifecycle management, and global distribution capabilities.
- *Graph databases* excel at storing and querying complex relationships between entities. These systems use specialized algorithms for traversing connections and identifying patterns across interconnected data.
- *Vector databases* enable semantic similarity searches across high-dimensional data by storing embeddings and providing optimized nearest-neighbor search algorithms. These systems support applications like document similarity, anomaly detection, and recommendation engines.
- *Feature stores* centralize engineered features for machine learning, ensuring consistency between training and serving environments while providing versioning, lineage tracking, and low-latency access for real-time predictions.

4.5.6 Data lifecycle management

The age of the data you collect and store also influences the storage technology that you choose, and data may transition through different types of storage over time. Recent data typically needs fast access for operational decision making, including alerts and dashboard visualization; whereas older data, while still important for analytical purposes, can often tolerate slower query times.

HOT DATA FOR REAL-TIME AND RECENT DATA ACCESS

Hot data includes recent operational measurements, current configuration data, and frequently accessed reference information. This data requires high-performance storage with low-latency access, typically using premium storage tiers and memory caching.

WARM DATA FOR HISTORICAL ANALYSIS AND REPORTING

Data from weeks to months old serves different analytical purposes with relaxed performance requirements. Monthly reports, trend analysis, and model training workloads can tolerate query response times measured in seconds rather than milliseconds. This shift in performance expectations enables cost optimizations through compression, aggregation, and migration to slower but cheaper storage tiers.

COLD DATA FOR ARCHIVAL ACCESS

Historical data beyond immediate operational relevance can be moved to archival storage where access times measured in minutes or hours become acceptable. This cold data primarily serves compliance auditing, forensic analysis, and occasional deep historical studies that justify longer retrieval delays.

Cloud archival services like AWS Glacier or Azure Archive Tier provide cost-effective cold storage with retrieval times ranging from minutes to hours depending on the storage class selected. The dramatic cost reduction—often 10:1 or better compared to hot storage—justifies the performance trade-off for infrequently accessed data. Figure 4.9 shows an example of lifecycle tiering and the effect on storage cost for data stored in Amazon S3.

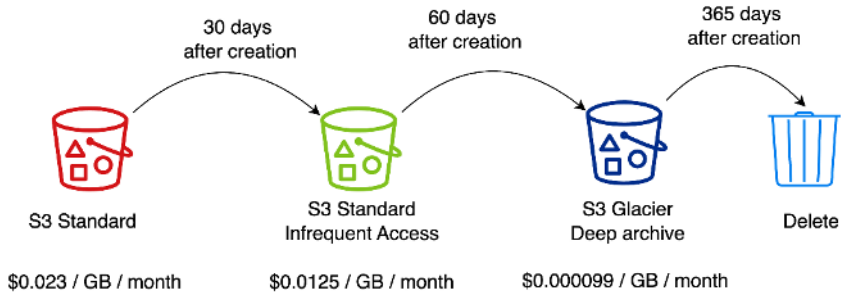


Figure 4.9 An example lifecycle policy for data stored in Amazon S3 that transitions data to cheaper storage and eventual deletion as it ages.

4.6 Data governance and compliance

Digital twins aggregate vast amounts of operational and business data from across organizational boundaries, creating both new analytical opportunities but also significant governance challenges. As digital twins evolve from pilot projects to enterprise-critical systems, establishing robust data governance frameworks becomes essential for maintaining data quality, ensuring regulatory compliance, and managing the risks inherent in large-scale data integration.

4.6.1 Data classification and sensitivity

Establishing clear data classification frameworks enables appropriate protection levels for different information types. Classification schemes typically include:

- *Public data* that can be shared openly without restrictions.
- *Internal data* restricted to organizational use.

- *Confidential data* requiring special handling and access controls.
- *Restricted data* subject to the highest security protections.

Digital twin architectures must enforce these classifications across diverse data sources, ensuring sensitive information receives appropriate protection throughout its lifecycle.

4.6.2 Regulatory compliance

Regulated industries face complex compliance requirements that vary by sector, geography, and data type. Healthcare digital twins must comply with HIPAA requirements, while financial services face SOX and PCI DSS constraints. International regulations like GDPR create additional complexity for systems that cross jurisdictional boundaries. Key compliance requirements often include:

- *Data minimization* - collects only data necessary for legitimate purposes.
- *Data portability* - enables data extraction in standard formats.
- *Right to deletion* - provides mechanisms to remove personal information.
- *Audit trails* - maintain comprehensive logs of data access and modifications.

4.6.3 Access control and authentication

Digital twins require sophisticated access control models that consider data sensitivity, user roles, and operational context. Role-based access control (RBAC) provides a foundation, but complex environments may require attribute-based access control (ABAC) that considers factors like time of access, location, and data age.

Zero-trust architectures assume no implicit trust based on network location, implementing verification for every access request. This approach is particularly important for digital twins that span multiple organizational boundaries and network segments.

4.6.4 Data quality

Poor data quality in digital twins can lead to failures that compromise operational safety and business outcomes. Faulty sensor readings can trigger incorrect automated decisions, while corrupted data feeds may undermine predictive models and analytics. Establishing robust data quality standards requires defining acceptable ranges for each sensor type, validation rules for imported data sources, and standardized procedures for handling missing or anomalous values. These standards should account for both individual sensor constraints and cross-sensor relationships that can validate data plausibility.

4.7 Selecting the right combination

Making optimal decisions about data architecture requires understanding how data types, sources, structures, ingestion methods, and storage technologies work together. Table 4.2 illustrates common types, sources, and integration patterns for the data found in a digital twin.

Table 4.2 Typical sources of different data types in a digital twin and how these are typically structured, integrated, and stored.

Data type	Typical sources	Integration method	Structure	Storage technology	Considerations
Reference data	ERP, asset management	Batch, API	Relational	PostgreSQL, Memgraph	Update frequency, consistency
Timeseries data	IoT sensors, SCADA	Streaming	Columnar	InfluxDB, Timestream	Volume retention, query patterns
Spatial data	GIS, surveys	Batch, API	Specialized	PostGIS, Memgraph	Complexity, query types, accuracy requirements
Derived data	Analytics, ML	Batch processing	Various	Feature stores, S3	Refresh frequency, serving requirements
Unstructured data	Cameras, documents	API, batch	Binary/text	S3, blob storage	Size, access patterns, processing needs

This decision matrix highlights that there is no single 'best' data storage solution; rather, optimal architecture is data type-driven.

Transactional data utilizes relational databases for schema enforcement and consistency, which is crucial for ERP systems. In contrast, analytical data relies on specialized storage technologies optimized for rapid ingestion, high-volume querying, or complex indexing. The handling of unstructured data using simple, highly scalable object storage completes the picture, demonstrating that the sheer volume and access pattern requirements of files often supercede the need for deep structure.

When evaluating architecture options, consider these key factors, but remember the choice of a data storage solution, whether it’s relational, NoSQL, columnar, or specialized, is a balancing act that requires a deep technical and business understanding across several dimensions that are based on the outcomes you plan to achieve with your digital twin:

- *Performance requirements* - real-time applications require low-latency storage and ingestion, while analytical workloads can tolerate higher latency for better cost efficiency.
- *Scale characteristics* - data volume growth rates influence storage technology choices and cost projections. Some systems scale better horizontally while others optimize for vertical scaling.
- *Consistency requirements* - critical operational data may require ACID transactions, while analytical data can often accept eventual consistency for better performance and cost.
- *Cost constraints* - storage costs vary dramatically between technologies and access patterns. Understanding total cost of ownership includes ingestion, storage, compute, and data transfer costs.

- *Integration complexity* - simpler architectures reduce operational overhead but may sacrifice optimization opportunities. Complex polyglot architectures provide optimization but increase management burden.

4.8 Data store for a home digital twin

My home digital twin requires storage for four data categories: timeseries data from sensors and measurements, object data including images and documents, spatial data for floorplans, and relationship data linking sensors, rooms, and content.

I chose cloud storage over local storage for three key reasons. Cloud providers offer superior reliability with automated maintenance, patching, and backups that I cannot match individually. They provide cost-efficient scalability, allowing me to pay only for current usage while accommodating future growth. Finally, they maintain higher security standards than I could reasonably implement myself.

This approach does introduce some risks that I've accepted including a dependency on constant internet connectivity (which I already have), potential egress costs (which are minimal given the relatively small data volumes I anticipate), and vendor lock-in (I mitigate this by choosing established, stable services). The operational simplicity and professional infrastructure of cloud storage justify these trade-offs for this application.

4.8.1 Storing timeseries data

I chose Amazon DynamoDB as the primary time series data store for my home digital twin. While not purpose-built for time series like InfluxDB or TimescaleDB, DynamoDB's serverless architecture eliminates infrastructure management and provides pay-per-use pricing—my most important requirement—while providing single-digit millisecond performance at any scale. DynamoDB's key-value design has the one crucial requirement that you know your access patterns before storing data. This query-first approach demands more upfront planning but delivers highly optimized performance for actual use cases rather than theoretical flexibility. For home digital twins with well-defined patterns and modest data volumes, this trade-off favors DynamoDB's operational simplicity over specialized time series features.

I use a single table design with a simple schema reflecting the sensor data it stores. The partition key is a unique sensor identifier (device EUI for LoRaWAN sensors, descriptive key for manual uploads), the sort key is the measurement timestamp, and attributes contain the relevant readings for each sensor type. This structure accommodates all my home sensors and enables time-window queries, as shown in table 4.3.

Table 4.3 A single DynamoDB table stores all the timeseries data produced by sensors in my home and allows efficient retrieval.

Partition key	Sort key	Humidity	Temperature	CO2	Power	Current	pH
a84041ce41845d13	1749945584458	57.9	18.4	-	-	-	-
a840411971871c86	1749946784664	66.5	20.5	-	-	-	-
24e124710b423527	1749946183884	60.3	16.4	765	-	-	-
00137a1000013507	1749947383975	-	-	-	20	100	-
main_meter_1	1749947383214	-	-	-	2000	-	-
pool_chemistry_1	1749947383569	-	-	-	-	-	7.2

The use of sensor identifier and timestamp as a compound key in the DynamoDB table allows me to efficiently query the data based on sensor and time period, as shown in listing 4.10. In this example I use a Docker container hosting a local version of DynamoDB running on my machine for development and testing, without the need to deploy to AWS.

Listing 4.10 An example of using the partition key of sensor id, and the sort key of timestamp to efficiently query timeseries data in DynamoDB.

```
import boto3

db = boto3.client('dynamodb', endpoint_url='http://localhost:8000',
                  region_name='us-east-1', aws_access_key_id='dummy',
                  aws_secret_access_key='dummy')#1

response = db.query( #2
   TableName='dtia_sensor_data',
   KeyConditionExpression='sensor_id = :sensor AND #ts BETWEEN
   ➥:start AND :end',
   ExpressionAttributeNames={'#ts': 'timestamp'},
   ExpressionAttributeValues={
       ':sensor': {'S': 'sensor_id1'},
       ':start': {'N': '1749946000000'},
       ':end': {'N': '1749947500000'}
   }
)

print(f"Found {response['Count']} readings:")
for item in response['Items']:
   ts = item['timestamp']['N']
   temp = item['temperature']['N']
   humidity = item['humidity']['N']
   print(f"  {ts}: {temp}°C, {humidity}%")
```

#1 Use a local instance of DynamoDB running in a Docker container in this example.
#2 Query the table based on sensor identifier and start and end time.

4.8.2 Storing object data

For my home digital twin, Amazon S3 serves as the object storage layer for unstructured data including images, documents (appliance manuals, floorplans, maintenance records), and historical sensor data exports as infrequently accessed warm data. S3's key advantages are virtually unlimited storage capacity and multiple storage classes that automatically optimize costs based on access patterns—frequently accessed configuration files remain in standard storage while historical exports transition to cheaper Infrequent Access or Glacier tiers.

```
home-digital-twin-bucket/
├── sensors/
│   ├── config/sensor_id1/config.json
│   └── firmware/temp_humidity_esp32-v2.1.0.bin
├── data_exports/
│   ├── daily/2025-08-18/sensor_data.csv
│   └── monthly/2025-08/complete_export.json
├── 3d_models/
│   ├── mesh/home.obj
│   └── tiles/tileset.json
├── media/images/2025-08-18/water_meter_1.jpg
└── documents/
    ├── manuals/MHI_DXK18ZTLA-WF.pdf
    ├── maintenance/2024-01-02_aircon_service.pdf
    └── floorplans/home.png
```

4.8.3 Storing relationship data

For complex relationship data in my home digital twin, I need a database that handles interconnected information efficiently. Graph databases excel here over traditional relational databases by providing natural representation of connections, efficient traversal algorithms, flexible schema evolution, and rich contextual insights that reveal patterns difficult to infer from other data models.

Graph database options include fully managed cloud services (Amazon Neptune, Azure Cosmos DB, Neo4j AuraDB) and self-hosted solutions. While managed services suit large, mission-critical workloads, my small home digital twin model doesn't justify their cost. Instead, I host Memgraph in a Docker container on an AWS EC2 server, providing low-cost operation while maintaining elasticity and future portability to other compute services. In chapter 5, we will look at how I use this graph database to store relationship data in my home digital twin.

> ***NOTE***
>
> Docker is a platform that packages applications and their dependencies into lightweight, portable containers that run consistently across any environment. This eliminates compatibility issues and makes it easy to deploy complex applications.

4.9 Summary

- There are a number of different types of data that are found in most digital twins including timeseries, relational, object, spatial, and derived data which includes features and embeddings.

- Digital twins source data from both internal data stores, and from external sources.
- Both operational systems and information technology systems within the organization are important sources of data that digital twins leverage, with these traditionally separate systems beginning to converge.
- There are different strategies digital twins adopt to ingest data including batch-based, streaming, and API-based integrations.
- There are a range of modern data storage technologies that can be adopted to store the types of data a digital twin uses.
- Digital twins aggregate data from many sources and governance and management of compliance of this data is an important consideration.

5 Modeling reality

This chapter covers

- Contextualizing data and linking it semantically to create a model of reality
- Ontologies as blueprints for semantic understanding of models of reality
- Knowledge graphs as a way to model relationships in the real world
- Serving the model of reality
- The importance of standardization

We have looked at how we can create a digital representation of a physical system, use sensors to measure how that system changes, and the ways in which we can integrate and store this data in a digital twin. But without some way to add meaning to this data, it remains a set of disconnected measurements that offer little insight into the true behavior of the physical system. Raw sensor readings such as temperature values, pressure measurements, or vibration frequencies are merely numbers until they are contextualized within an understanding of what they represent, how they relate to one another, and what patterns or anomalies might indicate about system health or performance.

The transformation from data to actionable intelligence requires layering understanding onto raw measurements. This involves creating relationships between diverse data streams and establishing the semantic frameworks necessary for a digital twin to interpret and reason about information, mirroring human expertise. This crucial step elevates the digital twin from a sophisticated monitoring tool to an intelligent system capable of reasoning and prediction.

We begin by examining how raw measurements become meaningful insights about system behavior. You'll discover how ontologies create semantic meaning, knowledge graphs build interconnected representations, and mathematical models provide quantitative foundations for analysis. By the end, you'll design modeling strategies that transform disparate data into unified representations, select appropriate techniques for different scenarios, and create digital twins that understand and reason about the systems they represent.

5.1 Making sense of data

Imagine that I am not at home, but have access to the sensor data from devices that I have installed in the house. Figure 5.1 is an example of data from an air sensor stored in DynamoDB in my home digital twin. In this table, I see a collection of numbers which are raw data—the number 20.1 is an example. The headings to the table add some meaning to that raw data, so I understand that the the number 20.1 represents a temperature (but is it degrees Celsius, or Fahrenheit?).

Without any additional context around the data, I cannot make any meaningful decisions based on it. I can see that the temperature in some room of the home is around 20°, but I do not know where in my home the sensor with identifier `24e124710b423527` is currently located.

partKey *(String)*	sortKey *(Number)*	co2	humidity	pm25	presssure	temperature
24e124710b423527	1755167296031	1086	54	2	1024.1	20.1
24e124710b423527	1755167896649	988	53.5	2	1024.4	20.1
24e124710b423527	1756463299483	1430	59.5	14	1030.6	20.1
24e124710b423527	1756463899494	1525	60.5	16	1030.7	20.2
24e124710b423527	1756464499979	1222	59.5	18	1030.7	20.2

Figure 5.1 An example of raw data without context from sensors in my home digital twin. Without knowing what the data is related to, I cannot use it to make any meaningful decisions.

I could go and find all the sensors in my home and look at their identifiers until I found the sensor with the identifier of `24e124710b423527` at which point I would know that it is currently in the living room. I also happen to know that this particular sensor transmits readings in degrees Celsius, and that I had set the air conditioner in that room at 24° several hours ago, so I would expect the temperature to be higher. I also know that it has been several years since I had the unit serviced, and that the service manual recommends having the refrigerant gas replaced every two years. I can see another temperature reading in the raw data that I know relates to an outside air sensor that shows the ambient temperature to be only 5°. I'm fairly certain I remember the unit worked well last winter when temperatures fell to freezing. By combining all of these facts about the physical system together in my mind, I decide that it's time to book in a service of my air conditioner unit—I have integrated multiple sources of knowledge together to determine what action I should take.

The challenge is that the additional facts that I have used to determine this action (the unit is set to 24°, the service manual recommends refrigerant gas, my last service was two years ago, the unit has previously handled cool temperatures) are currently all in my mind, and are not represented in a systematic way that my digital twin can reason about.

5.1.1 From measurements to decisions

Digital twins must transform raw sensor readings into actionable decisions. This progression is captured by Ackoff's pyramid, a framework developed by organizational theorist Russell Ackoff that shows how data gains value through successive layers of refinement. As shown in Figure 5.2, the pyramid has four levels: data becomes information when given context, information becomes knowledge when patterns and relationships are understood, and knowledge becomes wisdom when applied to make sound decisions.

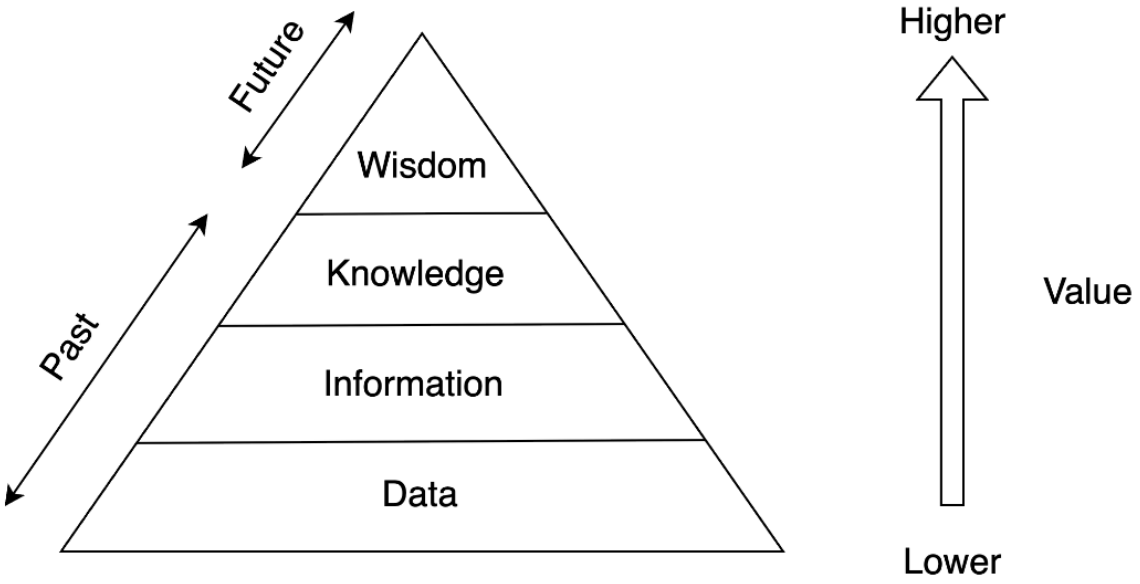


Figure 5.2 Ackoff's pyramid showing how valuable knowledge and wisdom are less available than lower value data and information.

Applying each level of this framework to an example of my swimming pool filter, where the goal is to optimize power consumption, we can see that:

- Data is raw, isolated facts without context. In my home digital twin, this is a simple, un-interpreted number: 2.8 kW.

- Information adds context to the data. The 2.8 kW reading becomes "the pool pump is drawing 2.8 kW of power at 2:15 PM on a Tuesday". This answers the "what, where, and when" questions, giving the data meaning.
- Knowledge reveals patterns and relationships by analyzing information. The digital twin compares the current reading to its historical data and pre-defined rules. The system knows that this pump typically draws 1.2 kW during normal operation, that power draw above 2.0 kW often indicates the filter is clogged and that the filter was last cleaned 2 months ago. Comparing to historical data, it can also tell that power consumption has been gradually increasing over the past three weeks.
- Wisdom applies knowledge to drive action. The digital twin, using its knowledge, can now recommend a course of action: "Since the pool pump is drawing 133% above normal power, indicating likely filter clogging, inspect the filter and backwash if clogged". In an autonomous digital twin, this could include automatically reducing the pump schedule to reduce power use.

5.1.2 The knowledge engineering challenge

While Ackoff’s pyramid provides a good theoretical framework for understanding how data becomes wisdom, digital twin practitioners face the significant challenge of how do you systematically capture and encode the kind of contextual knowledge that I applied intuitively to diagnose my air conditioner problem? In the earlier example, I effortlessly combined multiple pieces of data—history, manufacturer recommendations, environmental context, and past performance patterns—to reach a maintenance decision. This process felt natural because humans excel at contextual reasoning, but translating this capability into digital systems requires deliberate knowledge engineering.

Tacit knowledge is the undocumented, experience-based understanding that expert operators and technicians carry in their heads. This includes intuitive abilities, such as an operator who can "hear" when a pump is developing cavitation (the implosion of vapor bubbles sounds like gravel moving through the pump), or a facilities manager who knows that the east-wing HVAC system always struggles during afternoon sun exposure.

The challenge becomes even more complex in complex industrial settings. A chemical processing plant might have decades of operational knowledge spread across retiring engineers, tribal knowledge embedded in informal procedures, and hard-won insights from equipment failures that happened years ago. Capturing this knowledge enables a digital twin to effectively inform decision making and ultimately autonomous operation in the way a skilled operator can.

NOTE

The risk posed by the wave of retirement of skilled workers in manufacturing and associated loss of knowledge and expertise has already seen companies such as Boeing having to rehire retired engineers to ensure timely delivery from its 737 production line.

As an aging workforce moves to retirement across the industrialized world, the ability to capture, store, and use decades of accumulated knowledge and wisdom offers a huge potential benefit to many organizations.

5.2 Understanding context in digital twins

In the previous section, we explored how the knowledge engineering challenge requires us to capture and systematically represent the contextual understanding that human experts apply naturally. But what exactly do we mean by "context" in digital twin systems, and why is it so critical for creating intelligent, actionable insights?

Consider the example in figure 5.3 where a reading of 85° is received from a temperature sensor. In the case of the reading being related to an industrial furnace, this can be interpreted as being completely normal, based on a number of related pieces of data, most importantly that this is the optimal operating temperature as defined in the engineering specification. Exactly the same reading related to a different object—a cooling pump with a maximum operating temperature of 70° now becomes a critical alert, requiring immediate attention.

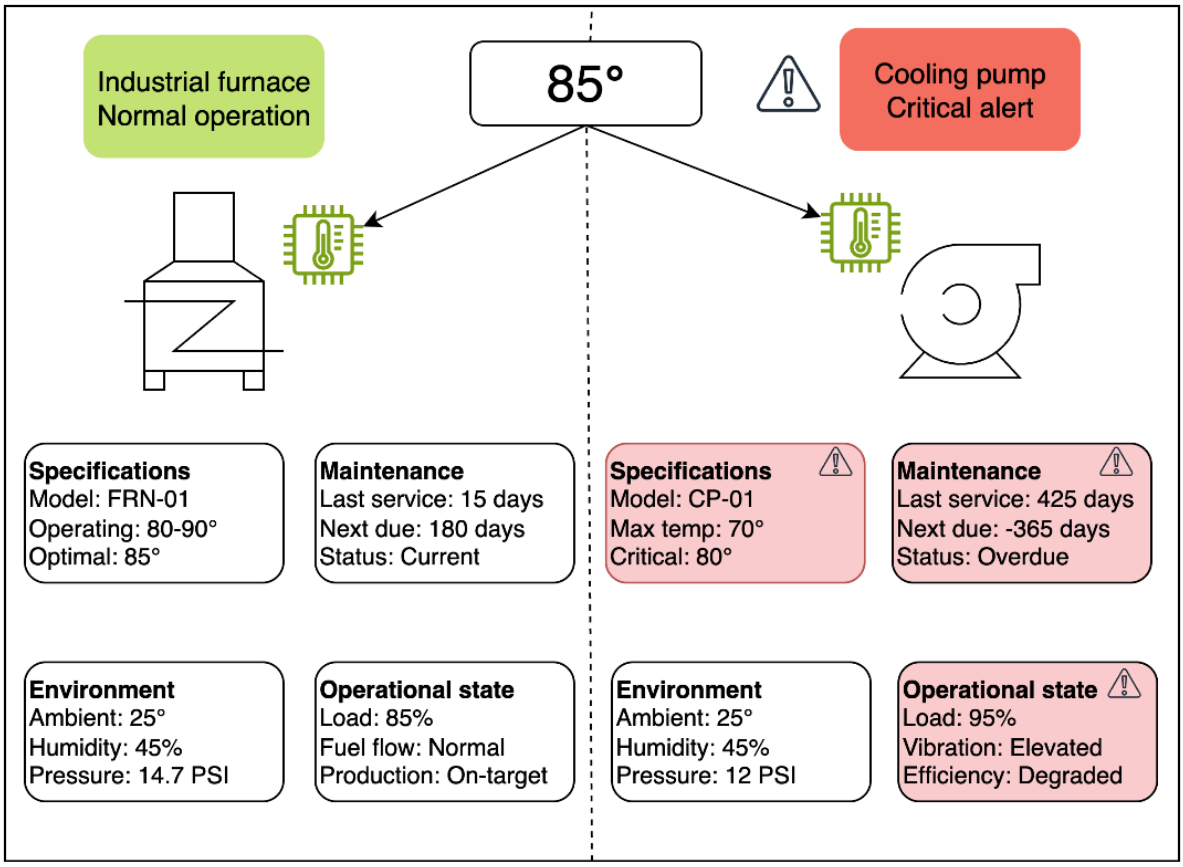


Figure 5.3 The importance of context in interpreting raw data and transforming it into information and knowledge.

A trained operator recognizes this distinction instantly, but their expertise masks significant complexity. Imagine it's your first day and you encounter this 85° pump reading. You must first locate the pump's operating specifications by logging into multiple systems, searching by model or serial number, or hunting through physical documentation. After determining the pump exceeds safe temperature, you need to understand system-wide impacts by consulting piping diagrams stored in a different system. To diagnose the root cause, you must retrieve maintenance history from yet another database—after figuring out how this pump is identified there. This fragmentation across disconnected systems creates dangerous delays when equipment safety demands immediate decisions.

Each step—matching the sensor to its specifications, linking it to system diagrams, connecting it to maintenance records—contextualizes raw data by adding layers of meaning. Yet in most facilities, this happens manually. The effort required to extract, match, and interpret data across siloed systems is expensive and error-prone. Worse, this critical knowledge often exists only as tacit expertise held by senior staff, making it vulnerable to loss and impossible to scale.

Digital twins solve this by encoding context directly into their model of the physical world. By capturing relationships between sensor readings and their associated equipment, specifications, maintenance history, and interconnected systems, a digital twin automatically transforms a measurement like 85° into the right interpretation, whether that's an urgent alert or routine confirmation of normal operation. The contextual framework that experts apply intuitively becomes explicit, queryable, and consistently available.

5.2.1 Types of context

Context transforms raw data into actionable information. Digital twins require four types of context to build a complete understanding of physical systems:

- *Spatial context* is the location of an entity. It links data to physical locations, whether absolute geographical coordinates or relative positions between objects. Knowing that a sensor is 2 meters from a west-facing window (in the southern hemisphere) explains why it shows temperature spikes in the afternoon. Spatial context enables meaningful visualization and reveals physical relationships that affect system behavior.
- *Temporal context* connects each data point to a specific moment in time. A temperature reading of 20°C is just a number; knowing it was recorded at 3:00 PM, rose from

18°C at 2:00 PM, and typically peaks at 22°C by 4:00 PM reveals patterns, trends, and potential anomalies.

- *Relational context* captures how things relate to each other. It answers questions like "which valve controls this pipe?"" or "what maintenance schedule applies to this pump?" These relationships link data points into a web of meaning that enables holistic reasoning.
- *Physical context* describes how components combine into larger systems. A digital twin models the hierarchy from individual parts up to complete assemblies, for example, a temperature sensor is part of an HVAC unit, which is part of a room, which is part of a building. This mirrors real-world structure rather than arbitrary data organization.

Figure 5.4 shows how these contexts work together for a garden moisture sensor. The sensor produces a reading of 15% —raw data without meaning. Temporal context shows moisture has dropped from 35% over three days. Spatial context places the sensor in the southeast corner, three meters from Sprinkler A and eight meters from Sprinkler B. Physical context reveals Sprinkler A connects to Solenoid 2. Relational context groups the sensor and Sprinkler A into "Zone 1" with shared watering rules.

Together, these contexts enable the digital twin to reason: "Zone 1 moisture is critically low and dropping. Activate Solenoid 2 to run Sprinkler A for 20 minutes". Without this contextual understanding, the digital twin sees only disconnected facts like a 15% reading here, and a solenoid identifier there, with no ability to take meaningful action.

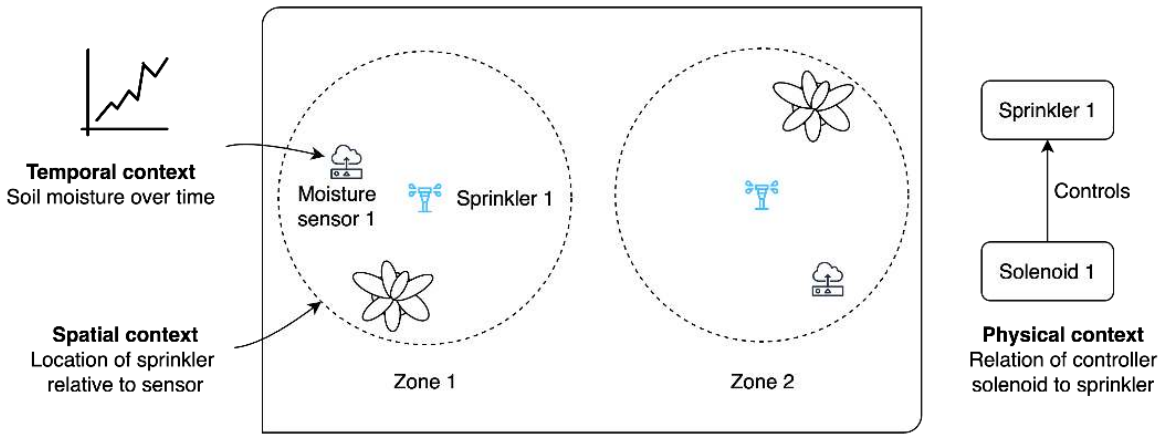


Figure 5.4 An example of how raw data produced by a moisture sensor in my garden can be contextualized with temporal, spatial, physical, and relationship context.

5.2.2 How to contextualize data

Unless you are starting from scratch, building a model of a physical system normally means that you will likely be dealing with many digital representations of the same thing. William Kent's 1978 book *Data and Reality* (just as relevant in 2025 as it was in the 1970's) deals with the challenges of representing reality through data and begins with the following quote

> *Entities are a state of mind. No two people agree on what the real world view is.*

— Apollon Metaxides
as quoted by William Kent

The significance of this is that humans perceive and manage data in different ways, and a data model is really a person's perception of ambiguous information, not objective reality itself.

If this starts to sound too philosophical, lets consider a practical example that you may experience in your daily life. If you have both a checking account and a credit card issued

by your bank, your information is likely stored in (at least) two different systems. You may be identified in both of those systems with a common identifier (say a customer number), but often this is not the case. Your name is probably recorded in both systems, but sometimes spelt differently. In the card system my name could be Gregory, but in the other system it is Greg. In order to obtain a full picture of a customer (the so called customer 360 view), my bank must link all the different digital representations together. The same is true across most industries today—think of your own industry or personal life and all the disconnected digital representations that exist.

The examples we have looked at so far of contextualizing data by linking data and entities together have been easy to do because of the small scale and the fact that I know the relationships. But what happens when you have hundreds of thousands of records spread across multiple systems where you do not know the relationships.

CONTEXTUALIZATION THROUGH DATA MATCHING

In practical terms linking related facts together means matching data stored in different systems and identified differently. This often entails matching different identifiers that are very similar but not identical. Think of the street address '51 Stratford Drive'—when written as '51 Stratford Dr.' a human can easily match these two identifiers as likely being equal, but to a computer they are completely different. When you’re adding context by matching records between different systems, you will often need an approach to match similar identifiers.

TRY IT OUT: MATCH TWO DATASETS

The easiest way to try and find similar items is to compare every item in your data set with every other item. Listing 5.1 shows an approach that uses a technique known as the *Levenshtein distance* which is a measure of the number of edits that need to be made to one string to convert it to another as a way to fuzzily match identifiers. You can use this approach to match items with identifiers that are similar, but not exactly the same.

> ***NOTE***
>
> `The fuzz.token_set_ratio` function in the example is a variation that ignores word order and duplicate words, often providing a better similarity score for identifiers.

Listing 5.1 A simple example of fuzzily matching identifiers which is a common approach to linking data records for contextualization.

```
from thefuzz import fuzz

identifiers = [ #1
    "CUST-001234", "CUSTOMER-001234", "cust_001234",
    "PROD-ABC123", "PRODUCT-ABC123", "Product ABC-123",
    "EMP-789456", "EMPLOYEE-789456", "emp_789456"
]

print("Fuzzy Matches (score >= 80):") #2
for i, id1 in enumerate(identifiers):
  for id2 in identifiers[i+1:]:
    score = fuzz.token_set_ratio(id1, id2)
    if score >= 80:
      print(f"{id1:15} ↔ {id2:15} ({score})")
```

#1 Set of sample identifiers that need to be fuzzily matched.
#2 Find fuzzy matches (identifiers that are similar and likely to refer to the same entity)

This brute-force comparison becomes prohibitively expensive, scaling quadratically, as dataset size grows (for example,

comparing 100,000 items requires billions of comparison operations). For large datasets, techniques like locality-sensitive hashing (LSH) are necessary to avoid this computational overload.

Identifying these matches is more than just a data cleaning exercise; it is the key step in constructing a model of reality in your digital twin, from data held in different systems. When your algorithm determines that two different identifiers likely refer to the same real-world object, you are effectively discovering a semantic relationship.

NOTE

Cognite Data Fusion® uses entity matching algorithms based on string similarity matching in its data contextualization tools.

5.2.3 The importance of context in generative AI

Generative AI (GenAI), especially large language models (LLMs), is transformative for digital twins, but its value is realized only with the right context. Without the rich data from a digital twin, an LLM is a general tool, but with context, it becomes a domain-specific expert.

Context is crucial for two reasons:

- *Grounding (preventing hallucination)*: Context anchors the LLM's responses, preventing it from producing inaccurate or fabricated information (hallucination). A general LLM lacks specific knowledge of a plant's layout or assets. If an operator asks, "What's the optimal pressure for Pump A-7?", a general LLM might give a generic or incorrect answer. By feeding the LLM real-time and historical data from the twin (model, age, maintenance

records), its response is grounded in reality and relevant to that specific equipment.

- *Specialization (enabling expert analysis)*: Context allows a general-purpose model to specialize. A digital twin contains proprietary domain knowledge (for example, engineering specifications, sensor data, tacit expert knowledge) not found in public training sets. Integrating this specialized data enables complex tasks, such as:
  - Analyzing turbine vibration readings.
  - Cross-referencing them with the specific maintenance schedule and operating hours.
  - Generating a detailed diagnostic report predicting a potential bearing failure.

The LLM thus moves beyond simple information retrieval to become an expert system capable of complex analysis and predictive recommendations.

5.3 Ontologies as a blueprint for understanding

To create accurate digital models, computers require a formal structure to organize and define concepts. An *ontology* is a formal, structured framework that defines the essential concepts, relationships, and properties within a given domain. Acting as a blueprint or schema, it precisely maps entities and their interconnections. This formalization allows machines to move beyond simple data processing to reason and infer, transforming raw data into a coherent knowledge map. This approach ensures the clarity, consistency, and shared vocabulary vital for building robust, scalable, and intelligent digital twin systems.

5.3.1 Core components of an ontology for a digital twin

The primary purpose of an ontology in a digital twin is to provide a comprehensive and structured knowledge model of a physical asset, system, or process. It acts as the semantic foundation that enables the digital twin to accurately represent, reason about, and interact with its real-world counterpart. At its core, a digital twin ontology is composed of the following key elements:

- *Concepts and classes* represent the types of entities that exist within the domain. In a digital twin of a factory for example, these might include `Machine`, `Product`, and `Building`.
- *Properties and attributes* define the characteristics of a concept providing the details that describe a specific instance. For example, a `Machine` might have properties such as `serialNumber`, `operationalStatus`, and `lastMaintenanceDate`.
- *Relationships* define how different concepts are interrelated, mapping the complex dependencies that exist in the physical world. For example, a relationship might state that a `Product` is `processedBy` a `Machine`. These connections allow the digital twin to understand the contextual relationships between data.
- *Rules and constraints* govern the behaviour and valid states of entities and relationships, allowing for automated reasoning and validation. For example a rule might state that a `Product` cannot be `processedBy` a `Machine` whose `operationalStatus` is `offline`.

5.3.2 Defining ontologies for digital twins

To define an ontology, we use a formal language that is both human-readable and machine-interpretable. In the context of digital twins, a leading example of such a language is the Digital Twin Definition Language (DTDL) used by platforms like Microsoft's Azure Digital Twin service. DTDL provides a

way to define the interfaces of a digital twin, specifying the models and relationships that a twin instance can have. It uses a JSON-LD syntax, which makes it easy to read and write. The language is structured to precisely capture the concepts (known as interfaces), properties (telemetry, properties), and relationships of a real-world entity.

> ***NOTE***
>
> JSON-LD (JSON for linking data https://json-ld.org/) is a method of encoding linked data using JSON designed to add semantic meaning to regular JSON by linking it to shared vocabularies and ontologies on the web.

For example, an interface for a room in a building would define its properties like whether it is occupied, and its temperature as shown in a minimal example in listing 5.2.

Listing 5.2 A minimal example of how a room might be defined using the Digital Twin Definition Language.

```
{
  "@id": "dtmi:com:dtia:Room;1", #1
  "@type": "Interface", #2
  "displayName": "Room",
  "contents": [
    {
      "@type": "Property",
      "name": "occupancyStatus",
      "displayName": "Occupancy Status",
      "schema": "boolean"
    },
    {
      "@type": "Telemetry",
      "name": "temperature",
      "displayName": "Temperature",
      "schema": "double"
    }
  ],
  "@context": "dtmi:dtdl:context;3" #3
}
```

#1 The @id attribute defines a unique identifier for the entity.
#2 The @type attribute specifies what kind of thing this data represents.
#3 The @context attribute is a link to a vocabulary that defines what terms mean - the DTDL version 3 context file.

5.3.3 Choosing an ontology for your model

There are many ontologies defined for a variety of domains, for example Microsoft has defined a number of industry standard ontologies specifically for digital twins using the DTDL. When considering the ontology you will adopt in your digital twin's model, you can adopt one of several strategies:

1. Adopt an existing standard ontology if one exists for your domain. The benefit of this approach is interoperability with other systems that use the same standard, combined with proven concepts that are likely supported by existing tools. The main drawback here is that the

rigidity of defined standards may not fit your specific needs perfectly, limiting your ability to innovate.

2. Extending an existing ontology gives you the benefits of adopting an existing standard, but with the ability to extend it to support your specific needs where necessary. The main drawback of this approach is the need to manage your extensions to the standard coupled with reduced support from other tools and systems.
3. Creating a custom ontology will enable you to create concepts that align perfectly with your domain at the expense of not having a shared vocabulary with other systems which may limit your interoperability.

For the digital twin of my home, I evaluated the Digital Buildings Project from Google (https://github.com/google/digitalbuildings) and the ontology defined by Microsoft in DTDL for smart buildings, which is based on the RealEstateCore ontology (https://www.realestatecore.io/) —an ontology for property owners. I chose the RealEstateCore-based DTDL as it is better suited to my home whereas the Digital Buildings ontology is better suited to large, modern, commercial buildings. The major concepts and relationships in the DTDL ontology are shown in figure 5.5. Where the model does not support concepts or relationships that I would like to model (for example a relationship between an asset and documents), I will extend it.

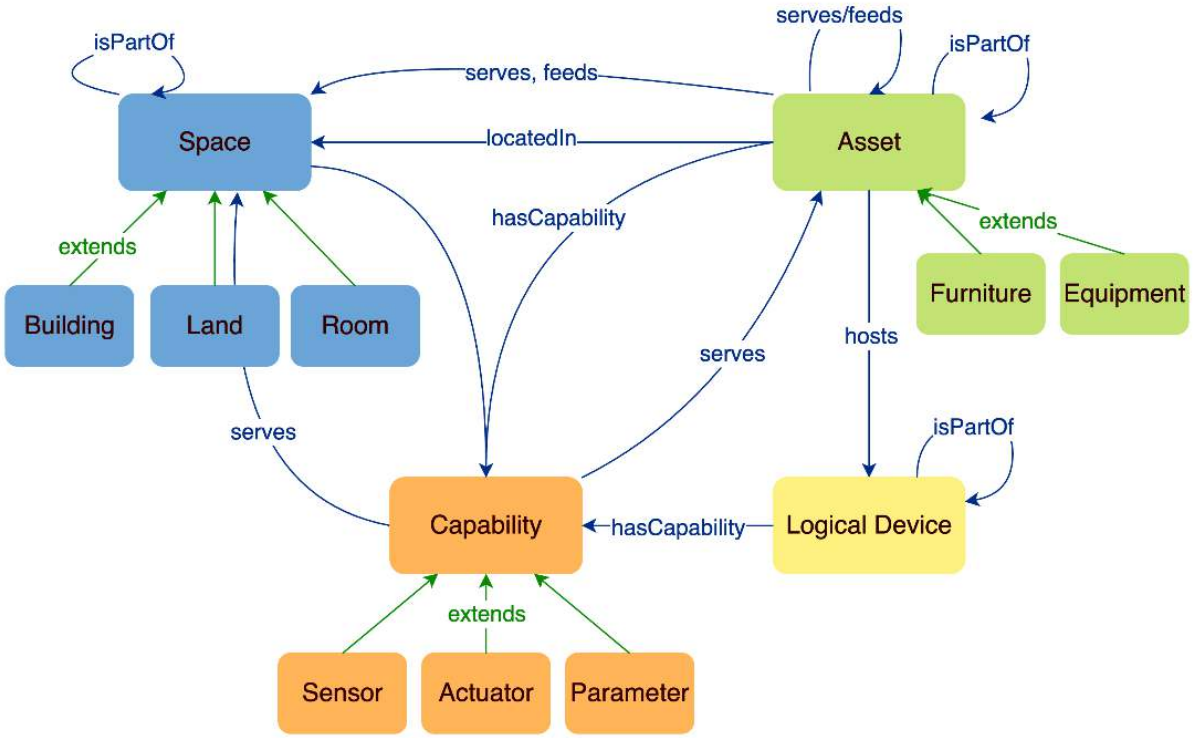
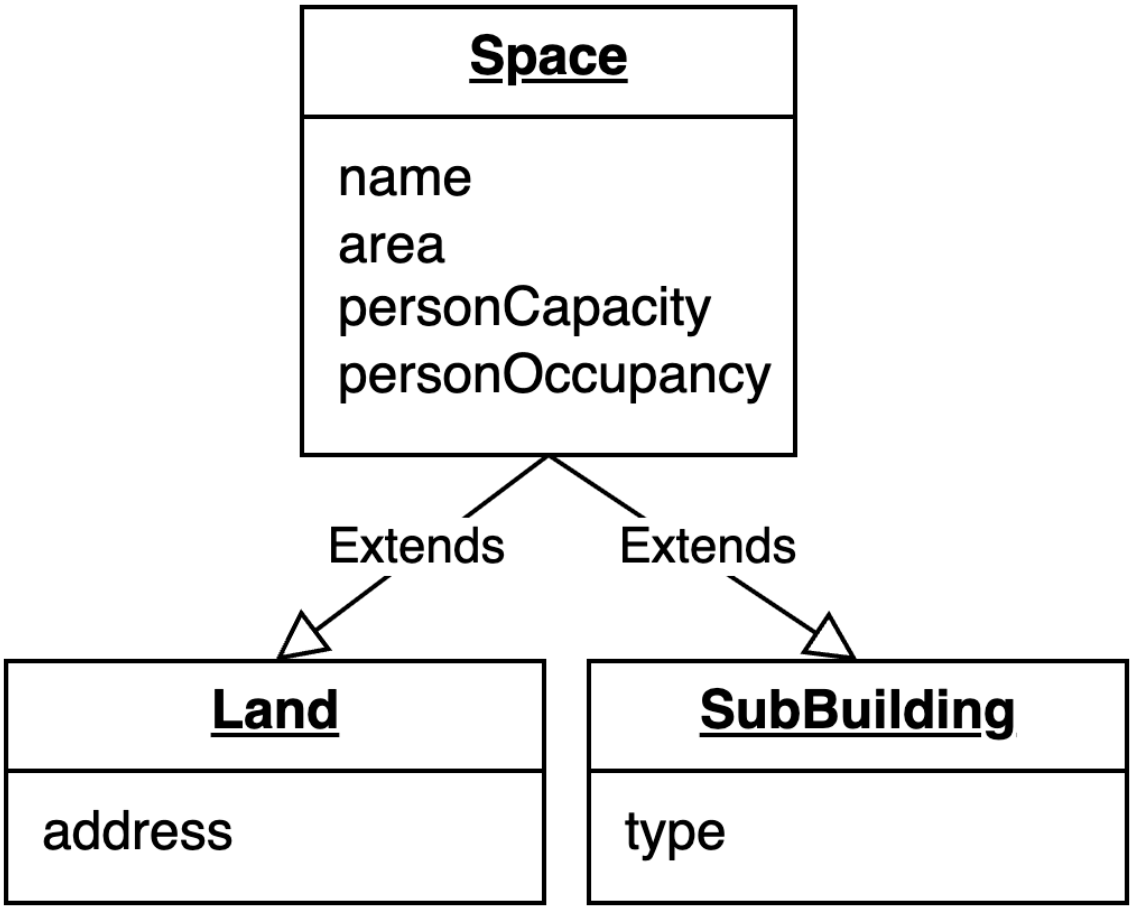


Figure 5.5 The main concepts and relationships in the DTDL implementation of the RealEstateCore ontology provided by Microsoft (https://github.com/Azure/opendigitaltwins-building). © Microsoft Corporation, licensed under MIT license. https://opensource.org/license/mit.

MODELLING EFFICIENTLY WITH INHERITANCE

To build scalable digital twins, DTDL uses a core concept from object-oriented programming called inheritance.

As shown in the diagram, a generic *Space* interface holds shared properties such as name and capacity. More specific interfaces extend from this generalized space, inheriting all the properties of a space, whilst adding additional properties that only apply to the more specific type of space —such as an address for a piece of land.



5.4 Knowledge graphs

The primary tool for implementing an ontology and building a robust, interconnected model of reality for a digital twin is the *knowledge graph* which is a structured representation of information that models real-world entities like (nodes) and the relationships (edges) between them. These relationships carry semantic meaning, making the data machine-readable and enabling automated reasoning about complex connections.

5.4.1 Graph theory

Knowledge graphs are based on graph theory, which models systems as connected points (nodes) and lines (edges). This abstraction reveals that complex problems are often about the underlying relational structure rather than granular physical details.

For example, figure 5.6 shows how the seven bridges of the city of Königsberg (now Kaliningrad) can be represented as an abstract network. The physical layout on the left becomes the graph structure on the right—nodes connected by edges. In a knowledge graph for a digital twin, these connections carry additional semantics, for example, a 'hasLocation' edge between a sensor node and a room node doesn’t just show they’re connected, but explicitly defines their spatial relationship.

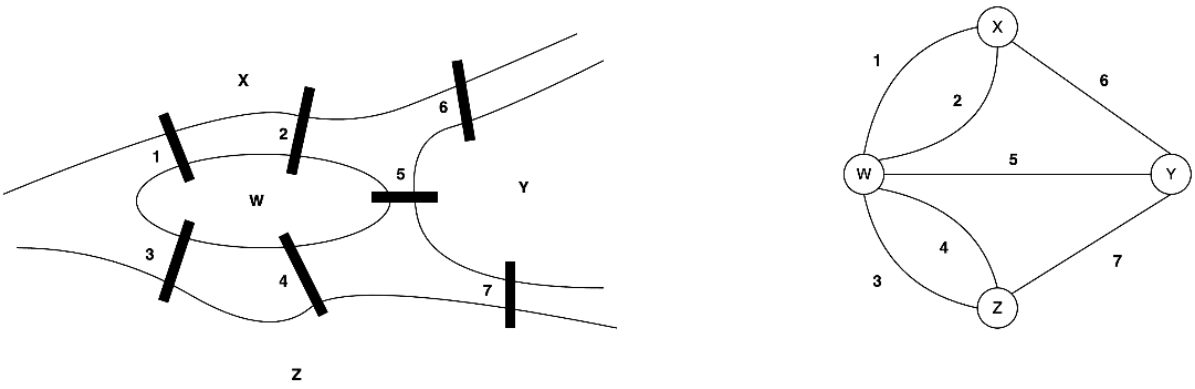


Figure 5.6 Euler's '7 Bridges of Königsberg' problem that was the origin of graph theory. He proved it was not possible to cross each of the 7 bridges connecting the two islands in the city (shown to the left) once and only once. The abstract representation of the seven bridges, two islands, and the shore either side of the river as nodes connected by edges is shown to the right.

The additional semantics encoded by a knowledge graph are defined by an ontology—a formal specification of the concepts, relationships, and rules that exist within a particular domain.

This semantic foundation transforms a collection of data points into a coherent model that mirrors how the physical world actually works. Instead of writing custom code to handle every relationship, you define the domain once in an ontology, and the knowledge graph automatically provides rich, queryable context for all your data.

NOTE

The term *knowledge graph* was first popularized by Google around 2012 following it's acquisition of Freebase, itself an online collection of shared knowledge. Today the panel of facts related to a search term that you get when performing a Google search is retrieved from its knowledge graph which as of 2020 reportedly held 500 billion facts about 5 billion entities.

5.4.2 Building a knowledge graph as a labeled property graph

Labeled property graphs are a popular way to represent a knowledge graph where both nodes and edges are *labeled*, assigned a label—the same as a class in an ontology—and both nodes and edges can have multiple key value properties attached. A simple example of a knowledge graph implemented as a labeled property graph based on RealEstateCore is shown in figure 5.7. This example shows how you might model a thermostat located in a room on the second floor of a hotel. You can imagine for a hotel with several thousand rooms spread over many levels how much more complex this model becomes.

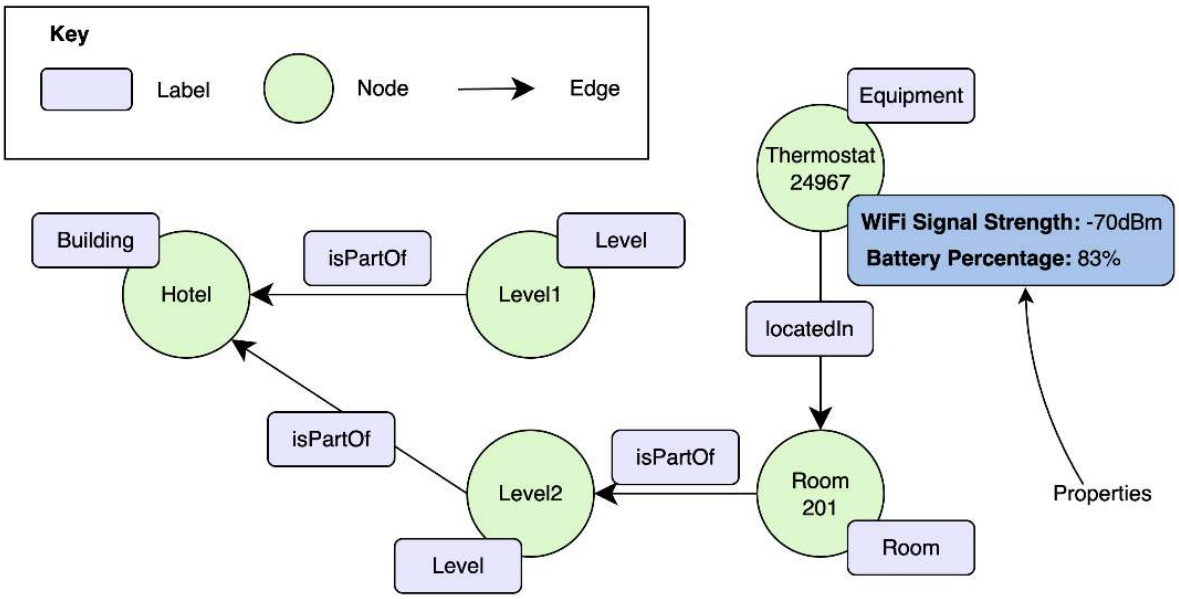


Figure 5.7 An example of a knowledge graph represented as a labeled property graph showing labels on both nodes and edges, and key, value pair properties against a node. This fragment represents a thermostat located in a room on the second floor of a hotel.

OTHER WAYS TO REPRESENT A KNOWLEDGE GRAPH

Resource Description Framework (RDF) is a popular format to represent a knowledge graph. RDF uses subject-predicate-object triples to create a web of linked data, where relationships are expressed as URIs rather than labeled edges as shown below for the graph in figure 5.7 in Turtle format. You can visualize this graph by heading to https://www.ldf.fi/service/rdf-grapher and pasting the RDF below into the box provided and clicking 'Visualize'.

```
@prefix hotel: <http://dtia.com/hotel/>.
@prefix rec: <http://www.realestatecore.io/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

hotel:Hotel rdf:type rec:Building.

hotel:Level1 rdf:type rec:Level ;
             rec:isPartOf hotel:Hotel.

hotel:Level2 rdf:type rec:Level ;
             rec:isPartOf hotel:Hotel.

hotel:Room201 rdf:type rec:Room ;
              rec:isPartOf hotel:Level2.

hotel:Thermostat24967 rdf:type rec:Equipment ;
                      rec:locatedIn hotel:Room201 ;
                      rec:wifiSignalStrength "-70dBm" ;
                      rec:batteryPercentage "83%".
```

The *Web Ontology Language (OWL)* is built on top of RDF and provides a richer vocabulary and constructs to define ontologies such as the *Semantic Sensor Network (SSN)* ontology (https://www.w3.org/TR/vocab-ssn/) for describing sensors and their observations. The DHT22 temperature and humidity sensor that is used in appendix B is defined in the SSN ontology here https://www.w3.org/TR/vocab-ssn/#dht22-description.

Next Generation Service Interfaces NGSI-LD is a graph-based context information model and API based on RDF, OWL, and JSON-LD that provides a uniform representation of entities and attributes as graph-based data. It is the basis of the Garnet Framework (https://garnet-framework.tech/), an open-source framework for building digital twins that use knowledge graphs.

5.4.3 Graph traversals

When modeling the real world in a knowledge graph, traversal is how we discover insights and answer complex questions. *Traversing* a graph involves identifying a starting point—a specific node—and then following the relationships to other nodes recursively until you find the information you are looking for, often using efficient graph traversal algorithms such as breadth-first, or depth-first search. Taking the example of the small subset of the knowledge graph representing a hotel shown in figure 5.6, I can build and traverse the graph using the declarative, open query language Cypher. Listing 5.3 shows how I use Cypher to first build the graph in a Memgraph server (running locally in a Docker container), and then run a traversal against it to find the battery level of all equipment on level two of the hotel.

NOTE

Memgraph (https://memgraph.com/) is an in-memory graph database designed for real-time analytics and transactional workloads on highly connected data.

Before running the code, start Memgraph in a Docker container with the following command:

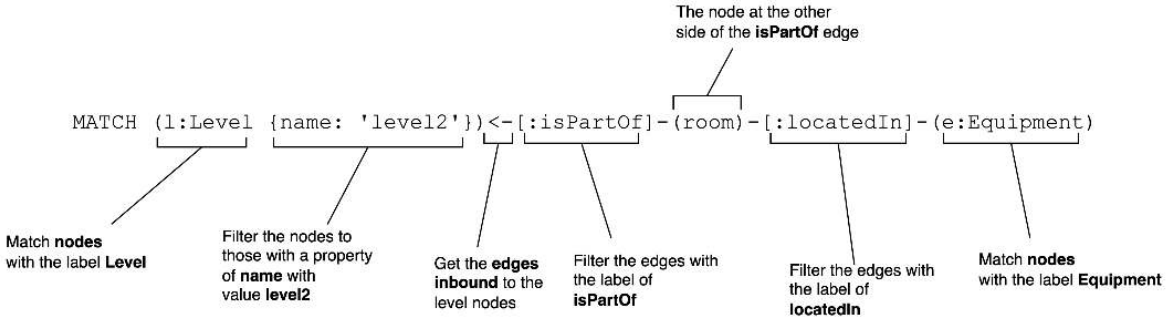
```
docker run -p 7687:7687 -p 7444:7444 --name memgraph memgraph/memgrap
h-mage
```

Listing 5.3 Create knowledge graph shown in figure 5.7 as a labeled property graph and run a traversal over it.

```
import mgclient

conn = mgclient.connect(host='127.0.0.1', port=7687) #1
c = conn.cursor()

c.execute("MATCH (n) DETACH DELETE n") #2
c.execute("""
CREATE (hotel:Building {name: 'Hotel'})
CREATE (l1:Level {name: 'Level 1'})
CREATE (l2:Level {name: 'Level 2'})
CREATE (room:Room {name: 'Room 209'})
CREATE (thermo:Equipment {name: 'Thermostat 24967', WiFiSignalStrengt
h: -70,
➥ BatteryPercentage: 83})
CREATE (l1)-[:isPartOf]->(hotel)
CREATE (l2)-[:isPartOf]->(hotel)
CREATE (room)-[:isPartOf]->(l2)
CREATE (thermo)-[:locatedIn]->(room)
""")

c.execute( #3
  """
   MATCH (l:Level {name: 'level2'})<-[:isPartOf]-(room)-[:locatedIn]-
(
   ➥e:Equipment)
   RETURN l.name, e.name, e.BatteryPercentage
  """
)

for row in c.fetchall():
    print(f"{row[0]} has {row[1]} (Battery: {row[2]}%)")
conn.close()
```

#1 Connect to the graph database running locally in Docker.
#2 Clear and create the knowledge graph.
#3 Traverse the graph to get the battery level of all Equipment on level two of the hotel.

The traversal shown in listing 5.3 can be thought of in plain language as *get me all equipment in rooms that are located on a level named level2 in the hotel*, with the full anatomy of the Cypher statement shown below.



This traversal is illustrated in graphically in figure 5.8. The key to efficient data retrieval from a graph as shown here, is to identify the starting node.

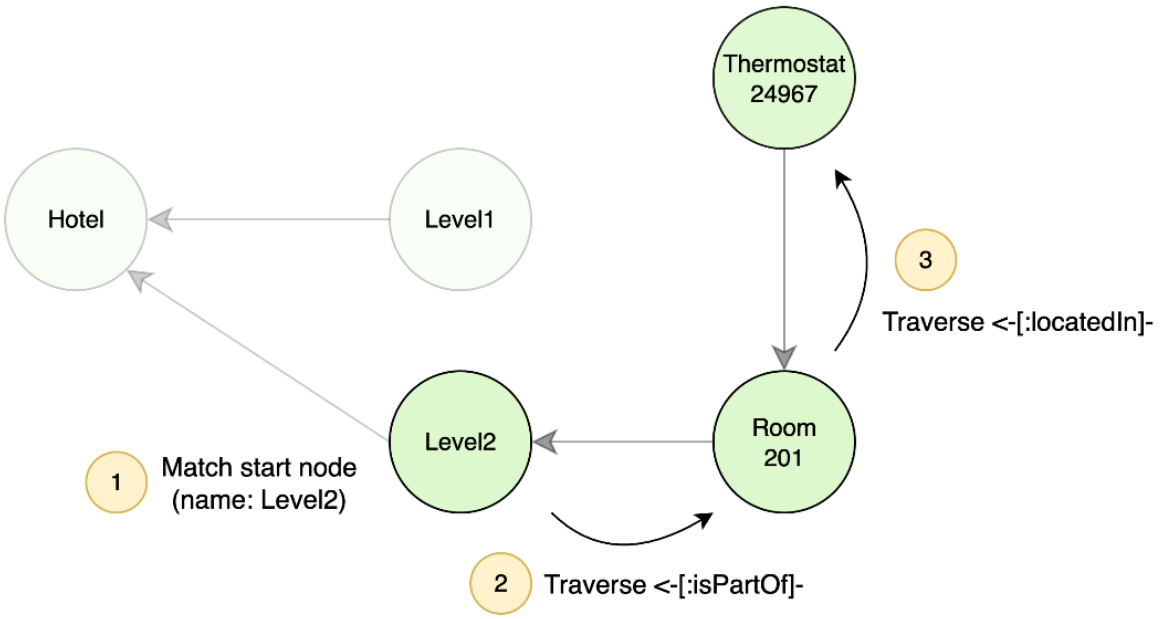


Figure 5.8 An example traversal that identifies a starting node, and follows edges to gather information.

For an excellent deep dive into graph databases and how they can be used to build such knowledge graphs, the book *Graph*

Databases in Action is recommended.

5.4.4 Building a knowledge graph of my home

I have built a knowledge graph, as a labeled property graph, that represents my home and gives me a way to link together the different data sources that make up the digital representation of the house. This includes the structure of the house and the relationships between rooms, appliances, sensors, and documents such as manuals, service records and photographs. As far as possible I have followed the RealEstateCore ontology as defined in DTDL although I have had to extend it in parts where the ontology did not define the relationships I need. Figure 5.9 shows a visualization of my complete home knowledge graph as shown in Memgraph Lab, and the detail of a small fragment showing node and edge types. Details of how to load and visualize this graph are provided in GitHub.

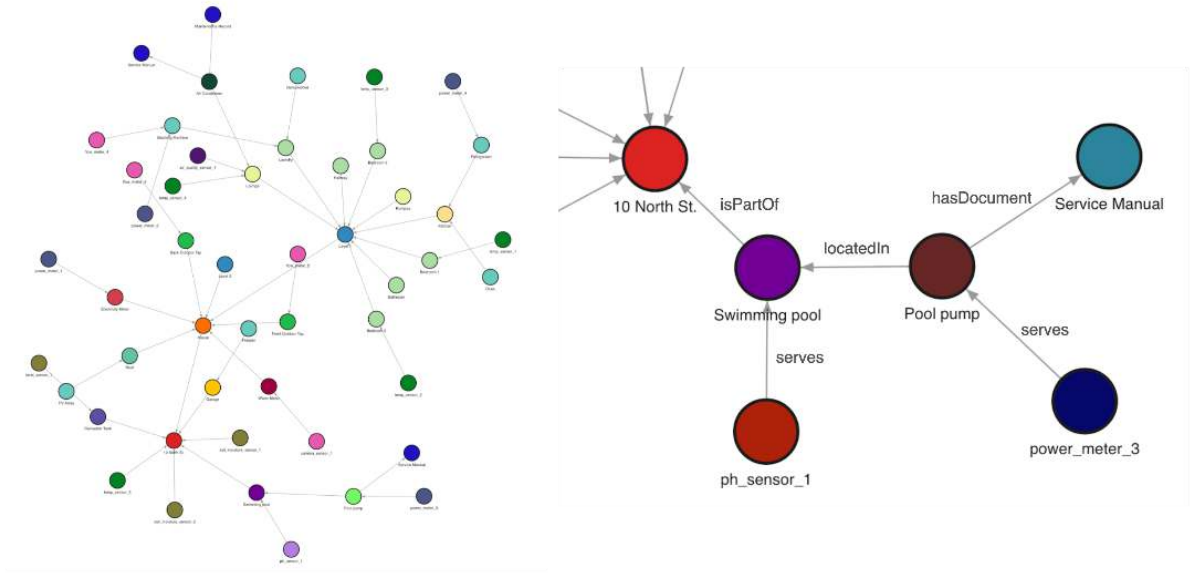


Figure 5.9 The complete knowledge graph of my home shown at left, with the detail of the relationships between the swimming pool, pump, related sensors and documents shown at right.

One of the benefits of modelling my home as a knowledge graph in this way is that I continue to build the system, discovering new entities that I want to model, and different relationship types, I can easily add them to the graph without complex schema changes.

NOTE

Willow (https://willowinc.com/) uses this same approach of a knowledge graph based on DTDL to create a digital twin of the iconic One Manhattan West skyscraper in New York city.

5.4.5 Choosing a knowledge graph service

As the complexity of your digital twin grows, you may no longer want to manage your own graph database infrastructure to host your digital twin's knowledge graph. At this point you can make use of managed services within public cloud providers to host it for you. Both AWS and Microsoft offer fully managed services for the developing knowledge graphs for digital twins. I trialed building my home graph in both Azure Digital Twins and AWS IoT TwinMaker, but ultimately decided on Memgraph for it's deployment flexibility—it runs seamlessly as a Docker container both locally for development and on AWS for production, enabling rapid experimentation without infrastructure constraints. This portability allows me, and you the reader, to quickly prototype and iterate on different graph models making it ideal for the exploratory nature of home digital twin development. As the database becomes larger and more complex and I look to a managed service, I can migrate my data to one of these cloud services, whilst maintaining the same graph traversal semantics I use today.

5.5 Standards and interoperability

Standards matter when you need to integrate with external systems, share models with partners, or ensure your digital twin can evolve with changing technology. Without them, the interoperability of the components of your digital twin, from the sensor communication level, all the way up to running simulations is more challenging.

5.5.1 A framework for standards implementation in your digital twin

Standards in digital twin development aren't a one-size-fits-all solution but exist at different levels, and a successful implementation hinges on prioritizing the right ones for your specific journey. At the foundational level, you'll find communication protocol standards (for example, MQTT and OPC-UA) that govern how data is transmitted between physical and digital assets. Moving up the stack are data model and schema standards (for example, IFC for construction and ISO 15926 for the process industry) which ensure data from disparate sources is semantically consistent and interoperable. Finally, at the highest level are the formal ISO standards (for example, ISO 30173 (https://www.iso.org/standard/81442.html), ISO 30194 (https://www.iso.org/standard/53314.html)) that provide a high-level framework and conceptual model for digital twins themselves. Understanding this tiered approach is important because it allows you to identify and implement the most critical standards first, laying a solid foundation before tackling the more abstract, higher-level frameworks. A well-considered strategy for standards implementation ensures not only technical compatibility but also long-term scalability and future-proofing of your digital twin.

5.5.2 Practical standards assessment

Navigating the world of any standards requires a careful assessment of their practical value and associated costs. Adopting a standard isn't just a technical decision but a strategic investment that can bring significant benefits but also comes with hidden costs, particularly related to compliance and audits. This section provides a practical framework for evaluating whether a standard is worth adopting based on its ability to solve a real-world problem or its long-term relevance in the industry.

THE TRUE COST OF STANDARDS ADOPTION

The cost of adopting a standard goes beyond initial implementation. It includes:

- *Implementation costs* cover the direct costs of technology, software, and labor needed to integrate a new standard into your existing digital twin ecosystem.
- *Compliance and audit costs* are often the most overlooked cost. Many formal standards, especially those from ISO, require regular audits to maintain compliance. These audits can be costly, both in terms of fees and the internal labor hours spent on preparation, documentation, and the audit process itself.
- *Training and expertise* ensures your team has the necessary skills to work with a new standard and requires investment in training and can also mean hiring new, specialized personnel.
- *Ongoing maintenance* costs relate to maintaining compliance with updated and evolving standards and keeping your systems aligned with the latest protocols.

5.5.3 A practical assessment framework

A pragmatic approach to standards adoption involves two key questions. Answering them will help you prioritize and decide

which standards are truly important for your digital twin journey.

1. *Does it solve a problem?* This is the most critical question. A standard should be adopted because it provides a clear solution to a specific challenge you are facing. For example:
    - *Interoperability* - if your digital twin needs to seamlessly integrate with other systems, such as a Building Information Modeling (BIM) platform, adopting a standard like IFC (Industry Foundation Classes) is essential. It provides a common language for data exchange, solving the problem of data silos.
    - *Data integrity* - if you need to ensure the accuracy and reliability of data from various sensors and devices, a communication protocol standard like *OPC-UA (Open Platform Communications Unified Architecture)* can solve this by providing a robust, secure, and standardized way to transfer data.
2. *Is it a widely adopted and relevant standard?* Sometimes, a standard's value isn't just about solving a current problem but about its future relevance and industry-wide acceptance. A standard that is widely adopted is more likely to be supported by a larger ecosystem of tools, vendors, and experts. When considering adoption and relevance, consdier:
    - *Industry consensus* - is the standard backed by a reputable consortium or a leading industry body? Look for standards that have broad support, as this indicates they are likely to remain relevant.
    - *Future-proofing* - adopting a well-established and evolving standard (like an ISO standard for digital twins) can future-proof your system, making it easier to integrate with future technologies and ensuring its long-term viability.

5.5.4 Real world example: the asset administration shell (AAS)

The Asset Administration Shell (AAS), developed by Plattform Industrie 4.0 and the Industrial Digital Twin Association (IDTA), is a standardized digital representation of a physical asset, crucial for Industry 4.0 digital twins.

The AAS acts as a digital "envelope" for an asset, containing all information and functionality relevant to its entire lifecycle (from design through disposal).

Its main components are:

- Submodels: Modular, standardized blocks that represent a specific aspect or function of the asset. For example:
  - A 'Digital Nameplate' for static data (serial number).
  - A 'Technical Data' submodel for specifications.
  - A 'Maintenance' submodel for service records.
- Standardized submodels ensure data is universally understandable.
- Identification and Metadata: Each AAS and its submodels are assigned a unique identifier (IRDI or URI) for global location and referencing. Metadata provides a high-level description to facilitate discovery.
- Security and Access Control: Includes mechanisms to define which stakeholders (for example, technician, supplier, customer) are authorized to view or modify specific parts of the AAS.

TRY IT OUT: BROWSE THE ASSET ADMINISTRATION SHELL

You can try out an online AAS browser that contains entries for a range of equipment at https://v3.admin-shell-io.com/.

Figure 5.10 shows an example of a Siemens pressure gauge showing it’s nameplate sub model and identifier.

Figure 5.10 An example of an AAS record for a Siemens pressure gauge, available from the IDTA AAS server at https://v3.admin-shell-io.com/. Licensed under the Apache License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.

Because this pressure gauge carries its own "digital passport". the digital twin can automatically ingest its calibration data without a human typing it in.

5.6 Serving the model of reality

The digital twin’s value lies in its ability to provide a usable, integrated model of reality to clients. This model, built from disparate data and unified by a knowledge graph, must be made accessible and interactive. Our challenge is to present this complex system, which combines data from various stores with a rich relationship network, in a way that is intuitive for users while accurately reflecting the real-world system it represents.

5.6.1 Context-aware data retrieval

To achieve this, we must move beyond traditional data retrieval methods. Standard database queries are limited, requiring explicit identifiers like a specific sensor or room ID. This approach falls short when users need to ask questions that are inherently relational and context-dependent, such as: "What is the temperature in all rooms on the second floor?" or "show me the energy consumption of all HVAC equipment serving the conference rooms".

These queries require a deep understanding of spatial, functional, and hierarchical relationships—information that is often invisible to conventional data stores. This is where the knowledge graph becomes essential. It acts as a semantic layer that stores and maps these relationships, from how components relate to specific spaces to how spaces are nested within buildings. This powerful semantic foundation allows our API to translate complex, human-intuitive questions into precise data retrieval operations across all of our distributed data stores.

The knowledge graph functions as an intelligent query router that first identifies relevant entities based on physical relationships, then orchestrates data retrieval from appropriate storage systems. When a user requests sensor information for a specific room, the query component follows this workflow shown in figure 5.11:

1. *Semantic resolution* - query the knowledge graph to identify the specified room, understanding that 'located within' may include direct placement or through spatial hierarchies.
2. *Entity discovery* - traverse relationships to find related entities—sensors, equipment, documents, or images, all connected to the same physical space.
3. *Data store routing*- use the discovered entity identifiers to query the appropriate time-series databases, document stores, or real-time data streams.

4. *Contextual assembly* - combine the retrieved data with the spatial and functional context from the knowledge graph to provide meaningful, structured responses.

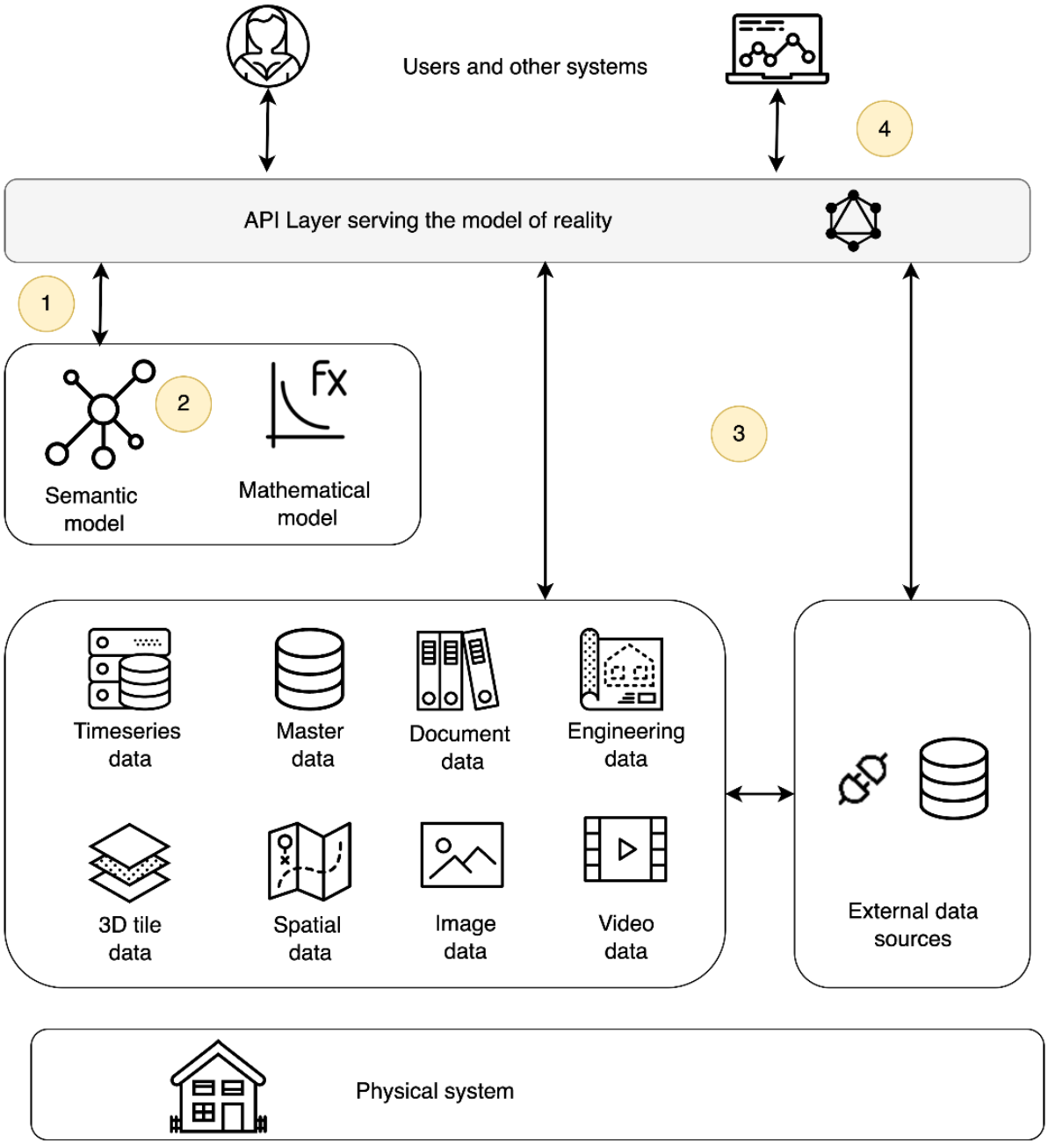


Figure 5.11 The knowledge graph provides a semantic model to the API layer that it uses to understand what data stores to query, both internal and external to the digital twin, constructing a model to return to the client.

5.6.2 Building an API layer for my home digital twin

The API layer forms the core of any digital twin system, translating complex internal models into accessible, actionable information. In a home digital twin implementation, this layer faces unique challenges: it must handle intricate relationships between spaces, devices, and sensors while delivering data in formats that both technical systems and end users can readily consume.

DATA ACCESS AND QUERY EFFICIENCY: REST, ODATA, AND GRAPHQL

As we have seen, digital twins operate on knowledge graphs —interconnected webs of relationships linking physical spaces, smart devices, sensor networks, and continuous measurement streams. The protocol we choose to expose this data affects both system performance and developer experience. Table 5.1 compares 3 popular API protocols for retrieving data over HTTP that can be adopted to serve data from a digital twin.

Table 5.1 A comparison of API protocols that can be used to serve data from a digital twin.

Protocol	Architecture	Query mechanism	Efficiency for complex data
REST	Multiple resource-specific endpoints, for example `/rooms`, `/sensors`	Fixed server responses; rigid data shape	**Low**. Requires multiple sequential requests
ODATA	Standardized protocol built on REST principles	Flexible querying via URL parameters (`$filter`, `$select`, `$expand`)	**Moderate**. Allows clients to request related data in a single request using `$expand`, improving efficiency over REST
GraphQL	Single endpoint (`$graphql`)	Client-defined query language specifying exact data needs	**High**. Addresses over-fetching; retrieves all required data across relationships in a single round trip

MULTI-STEP DATA FETCHING

Given the complexity of data relationships in a digital twin's model of reality, the way in which your API retrieves data is important to consider, both in terms of performance and usability of your interface.

Consider the complex query: "What is the current temperature in all rooms on the second floor?"

With a RESTful API the client must make multiple sequential requests:

1. Send `GET /floors/2/rooms` to retrieve all rooms on the second floor.

2. For each room, send `GET /rooms/{id}/sensors` to find sensors in that room.
3. For each sensor, `GET /sensors/{id}/measurements/latest` to get current reading.

> ***NOTE***
>
> I could hide these sequential requests to find rooms, sensors, and measurements behind a single HTTP endpoint and call it a "REST API" but that does not strictly align to the definition of REST.

Using an ODATA API, the client can use the powerful `$expand` feature to traverse relationships in a single request:

```
GET /rooms?$filter=floor eq '2'&$expand=sensors($expand=latestMeasure
ment)
```

This improves the efficiency by reducing requests to one, but the server determines the ultimate depth and shape of the response based on the expansion logic.

Using a GrapQL API, the client sends one precise query that dictates the exact shape and fields of the required response.

```
query SecondFloorTemps {
  rooms(filter: {floor: "2"}) {
    name
    sensor {
      latestMeasurement {
        temperature
      }
    }
  }
}
```

In this approach the client gets exactly the temperatures needed, traversing the relationships between rooms, sensor, and latestMeasurement in a single, highly efficient round trip.

For digital twins built on interconnected data, GraphQL offers the advantage of allowing clients to efficiently query the entire graph, minimizing latency and maximizing bandwidth usage.

5.6.3 The power of a GraphQL schema

The power of using GraphQL lies in defining a schema that represents the specific domain which in this case is my home. This schema acts as a contract, outlining all the data that can be queried and how different data points relate to each other. For my home digital twin, I've based my schema on the RealEstateCore ontology. This provides several key advantages:

- *Standardization* - by adopting an existing, well-defined ontology, I am not creating a custom, isolated data model. RealEstateCore provides a standardized vocabulary for concepts like buildings, floors, rooms, and sensors.
- *Interoperability* - this standardization ensures my digital twin can interoperate with other systems that also use the RealEstateCore ontology as they can speak the same language as my digital twin, facilitating data exchange and integration.
- *Context and meaning* - the ontology provides the semantic foundation that turns raw data into meaningful information. A simple temperature reading of 25° becomes a data point with context—it's the temperature of the 'living room' sensor, which is a 'sensor' located within a 'space' that is a 'room', as defined by the ontology.

NOTE

A GraphQL schema is itself a model of the physical system that encodes the ontology. The schema's type system enforces this ontology at run time.

By using GraphQL with an established ontology like RealEstateCore, I can build a flexible, powerful API that accurately models the reality of my home, ensuring it is both useful for my own applications and compatible with the broader smart home ecosystem.

TRY IT OUT: BUILD A GRAPHQL API

Listing 5.4 shows an example of how you can create a simple GraphQL API that orchestrates two calls: a call to a geocoding API to determine the co-ordinates of a given city, and once it has these, it calls the Open Meteo API to get the forecast temperature and rainfall for the next seven days, exposing a single query to clients to get the forecast for a given city.

Listing 5.4 An example of a simple GraphQL API that orchestrates calls to two RESTful APIs to return weather data for the requested city.

```
import graphene, requests
from flask import Flask, request, jsonify

class WeatherData(graphene.ObjectType): #1
  time = graphene.List(graphene.String)
  rain = graphene.List(graphene.Float)
  surface_temperature = graphene.List(graphene.Float)

class Query(graphene.ObjectType): #2
  weather = graphene.Field(WeatherData, city=graphene.String(required
=True))

  def resolve_weather(self, info, city):
    geo_data = requests.get( #3
      f"https://geocoding-api.open-meteo.com/v1/"
      f"search?name={city}&count=1"
    ).json()

    if not geo_data.get("results"):
      raise Exception(f"City '{city}' not found")

    lat, lon = (geo_data["results"][0][k] for k in ["latitude", "long
itude"])

    weather_data = requests.get( #4
      f"https://api.open-meteo.com/v1/"
      f"forecast?latitude={lat}&longitude={lon}"
      f"&hourly=rain,surface_temperature&models=ecmwf_ifs025"
      f"&forecast_days=7"
    ).json()

    return WeatherData(**weather_data["hourly"])

schema = graphene.Schema(query=Query)
app = Flask(__name__)

@app.route("/graphql", methods=["POST"])
def graphql_server():
  try:
    result = schema.execute(request.get_json().get("query"))
    return jsonify(result.data)
```

```
  except Exception as e:
    return jsonify({"error": str(e)}), 400


if __name__ == "__main__":
  app.run(debug=True)
```

#1 Define the schema of the response.
#2 Define a single Query on the API.
#3 The Open Meteo API only works with coordinates, not city names, so we need to geocode the input city.
#4 Once we have the coordinates of the city, we can call the Open Meteo API.

Once you are running the simple GraphQL server shown in listing 5.4 you can retrieve the forecast temperature and rainfall for the next 7 days for any city with a query as follows for Perth:

```
curl -X POST http://127.0.0.1:5000/graphql \
 -H "Content-Type: application/json" \
 -d '{
    "query": "{ weather(city: \"Perth\") { time rain surfaceTemperatu
re } }"
 }'
```

Listing 5.5 shows a fragment of the GraphQL schema of the first version of home digital twin API (the source code of the API is available in the GitHub repository). The API provides a query that you can use to look at a space over a period of time, and returns all information related to that space, for that time period. Following this approach, I do not need to know what specific sensors are measuring what parts of physical space (as I did in my example in section 5.1)—the API orchestrates retrieving relationships from the knowledge graph, querying source systems, and constructing a response.

Listing 5.5 A fragment of the schema of the GraphQL API for my home digital twin showing one query type.

```
type Query { #1
  spaces(space: String!, start_date: String!,
    end_date: String!): [Space!]!
}

type Space { #2
  name: String!
  sensors: [Sensor!]!
  documents: [Document!]!
  images: [Image!]!
  measurements: [MeasurementGroup!]!
}

type Sensor {
  id: String!
  space: String!
}

type Document {
  id: String!
  url: String!
}

type Image {
  id: String!
  url: String!
}

type MeasurementGroup {
  name: String!
  unit: String
  values: [Measurement!]!
}

type Measurement {
  sensor_id: String!
  timestamp: String!
  value: Float!
}
```

#1 Queries define how data is retrieved from the GraphQL API.
#2 The GraphQL schema defines a set of types that define format of the

data returned by the API, based on the RealEstateCore ontology.

5.7 Summary

- A model of the reality of the physical world is at the heart of a digital twin.
- The wide variety of data that a digital twin acquires and stores needs to be turned into information, which is the basis of higher level understanding of the physical system it represents.
- Contextualization is the process of linking related data together to create information and knowledge.
- Ontologies provide formal semantic definitions of models of reality.
- Knowledge graphs provide a powerful mechanism for modeling the real world based on an ontology.
- There are a range of efforts underway by various bodies to formalize standards related to digital twins as models of reality.
- The model of reality maintained within a digital twin must be served to consumers via an API layer that mediates between the semantic model, and raw data stores.