**Part 2: Solution Architecture Archetypes in Context**

We have found that most solutions and patterns in edge computing seem to fit three specific approaches, with slight variations and modifications. We thus refer to these approaches as pattern archetypes. The three chapters in this section will explore and delve into the three main archetypes. These chapters are meant to build successively on the patterns and concepts introduced in previous chapters, so [*Chapter 4*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_04.xhtml#_idTextAnchor073) builds on the concepts of *Chapter 3*, and [*Chapter 5*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_05.xhtml#_idTextAnchor091) on [*Chapter 4*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_04.xhtml#_idTextAnchor073). This part will show how the architectural components should be used together, the circumstances and situations that should be embraced or avoided, and how and why they are commonly paired.

This part has the following chapters:

* [*Chapter 3*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057)*, Core Edge Architecture*
* [*Chapter 4*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_04.xhtml#_idTextAnchor073)*, Network Edge Architecture*
* [*Chapter 5*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_05.xhtml#_idTextAnchor091)*, End-to-End Edge Architecture*

# 3

# Core Edge Architecture

This chapter covers the basic user edge architecture, which tends to focus on managing and enabling IoT-type sensors and smart devices. The subtle differences between this architecture and legacy IoT architectures are highlighted. The use of container technology is common in these edge architectures to accommodate the limited capacity of field-deployed edge devices, even if they don’t traditionally support common container engines or Linux distributions.

You will learn about the first of the three archetype patterns: the edge device hub pattern (depicted in Figure 3.6), how it improves on legacy IoT architectures, and some common variations and modifications you can make to it.

In this chapter, we will cover the following main topics:

* What is legacy IoT architecture?
* Device configuration
* Edge devices versus edge hub
* Containers
* Disconnected operations

By the end of this chapter, you should have a good idea of how and when to use this approach to migrate existing solutions or when to start with a clean slate. Along the way, you will learn ways in which this approach will bring value by saving time and money.

# Suggested pre-reading material

* Using containers to build applications (<https://www.docker.com/resources/what-container/>)
* Learn about WebAssembly (Wasm) (https://webassembly.org)
* Wasm system interface (<https://wasmbyexample.dev/examples/wasi-introduction/wasi-introduction.all.en-us.html>)
* WAIT (<https://dl.acm.org/doi/10.1145/3498361.3538922>)
* FIDO Alliance’s Device Onboard (FDO) specification (<https://fidoalliance.org/intro-to-fido-device-onboard/>)

# What is legacy IoT architecture?

In this section, we will cover legacy IoT architectures: their purpose, promise, and fundamental drawbacks. You will learn why IoT has provided value to business executives, where it was heading as a natural technological progression before being superseded by edge computing, and reasons why it may not have been adopted as widely as initially anticipated.

## A bit of history

Large commercial IoT networks became a viable solution for business with the convergence of cheap, low-power embedded processors and inexpensive, ubiquitous cellular data transmission. At that point, it became less expensive to transmit data from sensors than to have humans visit the sensors and record the data manually. However, the data was still tabulated and stored in central locations – a **data center** (**DC**) or, eventually, the cloud.

That approach and those needs formed the basis for initial IoT architectures, which connected devices over transmission networks directly to data processing and storage facilities. We’ll cover the initial variations that evolved as precursors to edge computing in a series of three diagrams.

In the first example, rudimentary IoT devices communicated over the internet using cellular data connections directly to an IoT hub for aggregation, processing, and storage. Data was not filtered, reformatted, or otherwise modified until it entered the IoT hub. It was the responsibility of the IoT hub to understand the native protocols and data formats used by the IoT devices and the device locations and to reformat the data while enriching the records with external data sources before sending the results off to other applications for reporting and storage (Figure 3.1):

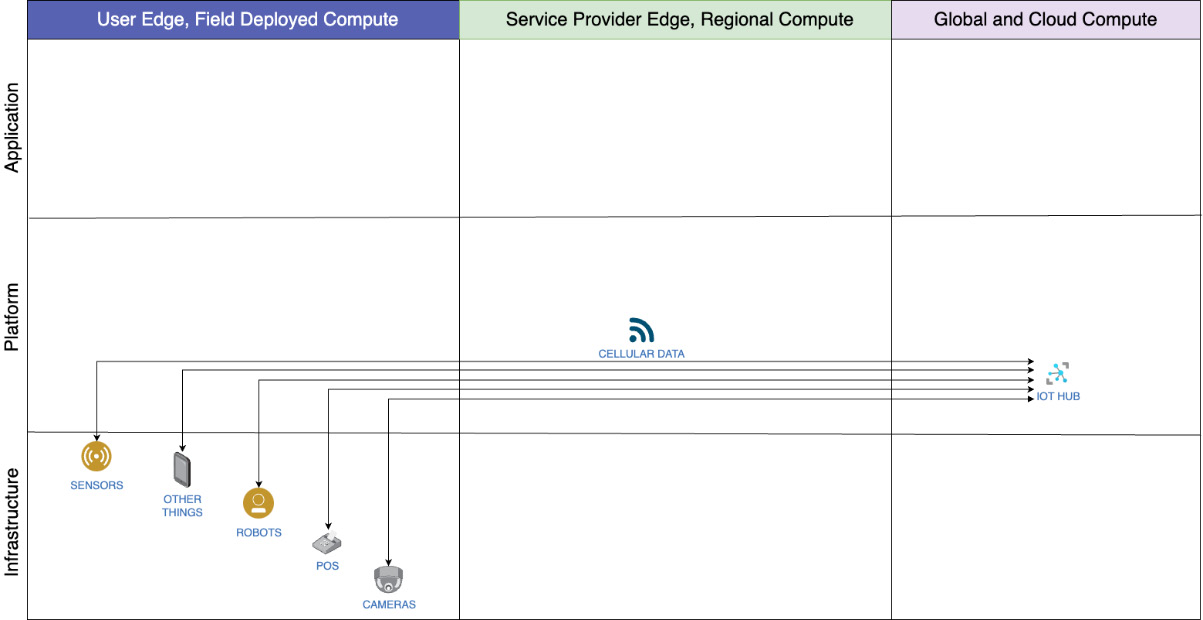


Figure 3.1 – IoT devices backhauling data to an IoT hub over cellular radio

This gave great flexibility in device placement, and typically a 15- to 25-mile transmission range. But it correspondingly required all of the devices to use the same technology and possibly the same mobile **service provider** (**SP**).

In the second example, IoT devices began using a variety of connections and transport types, including Wi-Fi, Bluetooth, and **NarrowBand-IoT** (**NB-IoT**). Additionally, some IoT devices began using **publish/subscribe** (**pub/sub**) brokered protocols such as **Message Queuing Telemetry Transport** (**MQTT**) instead of proprietary or industry-specific protocols. This necessitated the introduction of an IoT router that could support incoming signals and route them over the internet. At the same time, the routers could also package and/or reformat the data into a standard format. The IoT router thus allowed the IoT hub to gradually offload functionality edgeward (Figure 3.2):

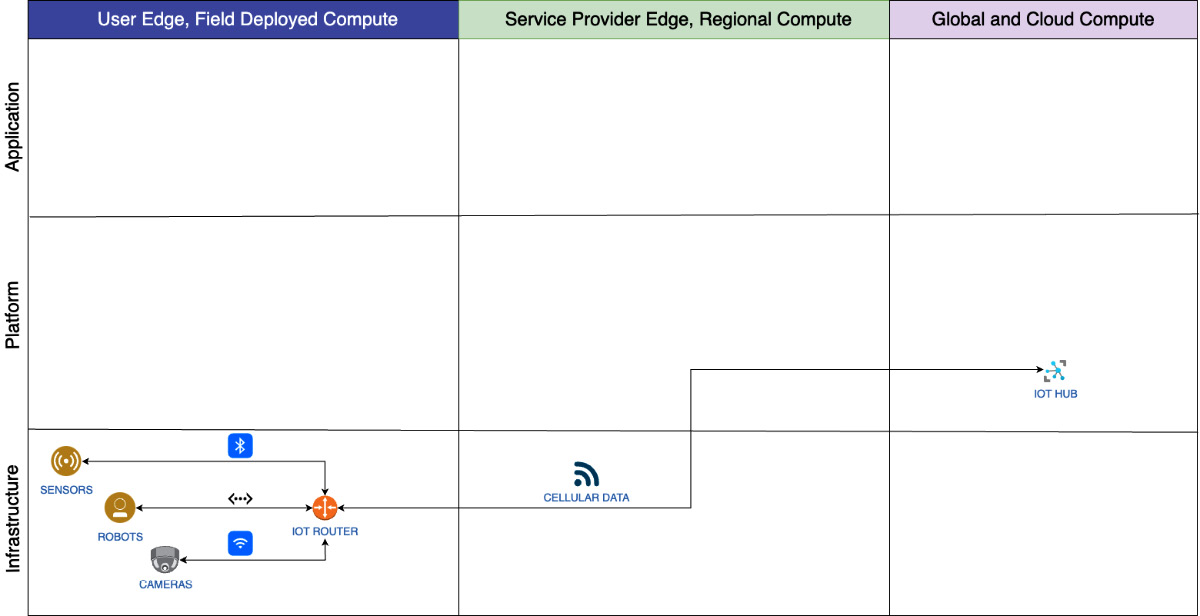


Figure 3.2 – IoT devices sending mixed-format messages to IoT routers, then to an IoT hub

A fundamental drawback to using an IoT router was that the transmission types were mostly shorter-range technologies with Bluetooth in the 6- to 25-foot range, Wi-Fi in the 10- to 150-foot range, and LoRa over 9 miles in optimum conditions. This allowed flexible placement within a defined area. The trade-off was the flexibility gained by supporting heterogeneous transmission options.

In the third example, smartphones provided the ability to perform the duties of both IoT router and IoT hub when powered by cloud provider-specific SDKs and when communicating northbound to proprietary interfaces in cloud services. This emerging trend reduced costs by using **commercial off-the-shelf** (**COTS**) technology and by leveraging existing cloud-native development skills (Figure 3.3):

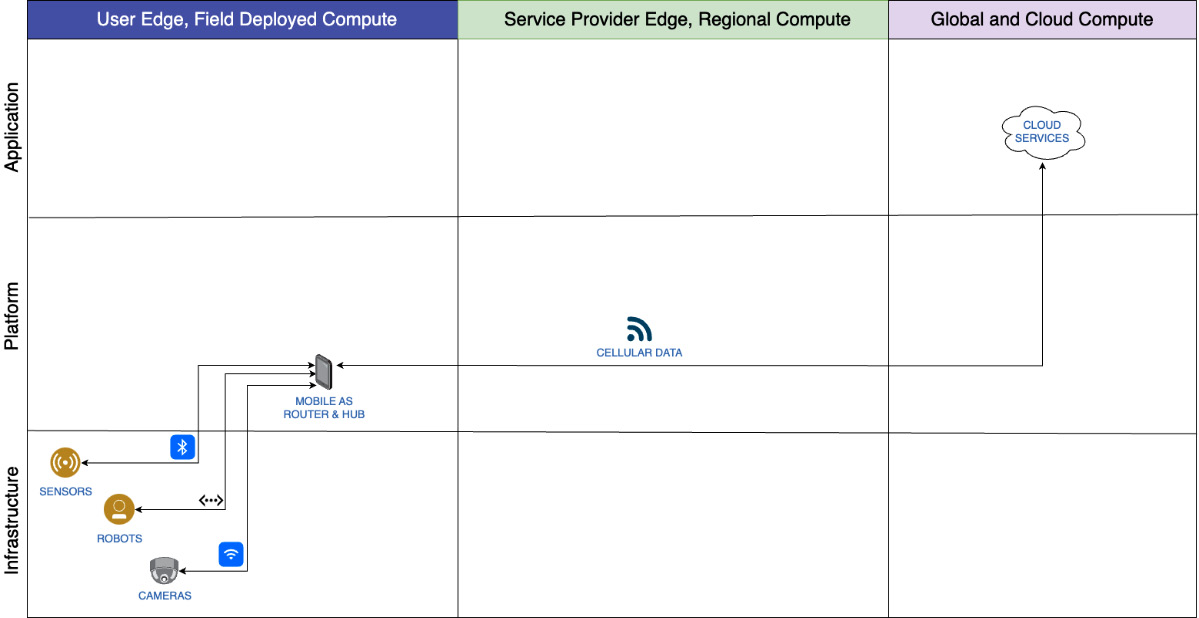


Figure 3.3 – IoT devices connected to mobile devices operating as local IoT hubs

As the preceding three diagrams depict, IoT solutions evolved to migrate supporting platforms southbound toward IoT devices as hardware became cheaper, new protocols and transmission options became supported, and cloud-native development practices emerged. However, IoT devices stubbornly remained single-purpose fixed-function, still needed to connect to a gateway of some sort, and still transmitted raw data northbound to the cloud (backhaul) where all processing, reporting, alerting, and storage were performed by cloud-based services.

## Purpose and promise

IoT devices are defined by their capabilities: they are standalone devices that contain functionality consisting of one or more sensors, actuators, or displays/gauges, can connect to the internet (usually wirelessly), and may be IP-addressable. This distinguishes them from peripherals that must be connected to a host (computer), use a device driver, and are only usable when connected. IoT devices can be a fixed function and not updatable by the end user, or they may run custom firmware or even an embedded **operating system** (**OS**) capable of loading and running applications.

Prior to IoT device availability, each device would need to be connected directly to a host machine on a local network using a wired connection that was then connected to an uplink for data backhaul. Embedding wireless connectivity into IoT devices allowed them to bypass the host requirement, and sometimes local network requirements, and connect directly to the internet.

This ability to connect directly to the internet simplified deployment architecture complexity and expense, and reduced maintenance costs by removing the need for local IT staff in some situations. Direct connectivity also allowed remote access to devices, eliminating the need for dispatching humans on-site to monitor and maintain the devices. Since on-site personnel were no longer needed, it decreased the cost of operating a fleet of devices, increased the ability to scale the size of fleets of devices, and also increased the quality and quantity of data being collected, the effectiveness of the devices, and the range of possible products that could be created based on these new skills.

## Fundamental drawbacks

While IoT devices brought value to the organizations that deployed them, those deployments were not without trade-offs. The drawbacks of this approach were the expense of the data connections and the configuration of the data links, limits to the data transmission speeds, and a lack of standards for the data being transmitted, thus requiring standardizing on a single vendor and/or product (thus requiring vendor lock-in) or maintaining a solution at the receiving end that would catch the data transmission and convert it into a standard format and schema (thus increasing the cost of maintenance).

# Device configuration

In this section, we will cover the device configuration use case as an introduction to the edge device hub archetype pattern. You will learn the role and purpose of the infrastructure, platforms, and applications (collectively referred to as architectural elements going forward and shown in Figure 3.4) that support device operation at the edge. We will briefly discuss the benefits and drawbacks of placement decisions.

## Rationale

The device configuration use case is meant to supplant legacy IoT architectures that connect fixed-function devices and embedded systems directly to a host in the cloud. Having a single platform for all devices puts the burden of device registration, protocol support, data collection, and analytics on one platform in one location. However, this approach required all data to be sent to the cloud if it wasn’t discarded, resulted in delayed analysis (thus delayed reaction times due to latency), and incurred data transfer and storage expenses. It could also force lock-in to a particular product suite, implementation library or SDK, vendor, or cloud. Some solutions had scalability issues, and costs increased linearly with scale. The advent of edge computing and the new architectures that it introduced mitigated every one of those issues. Let’s cover the individual elements with an eye toward what an edge-native approach would entail. The next subsections will explain the elements depicted in Figure 3.4 and their purpose:

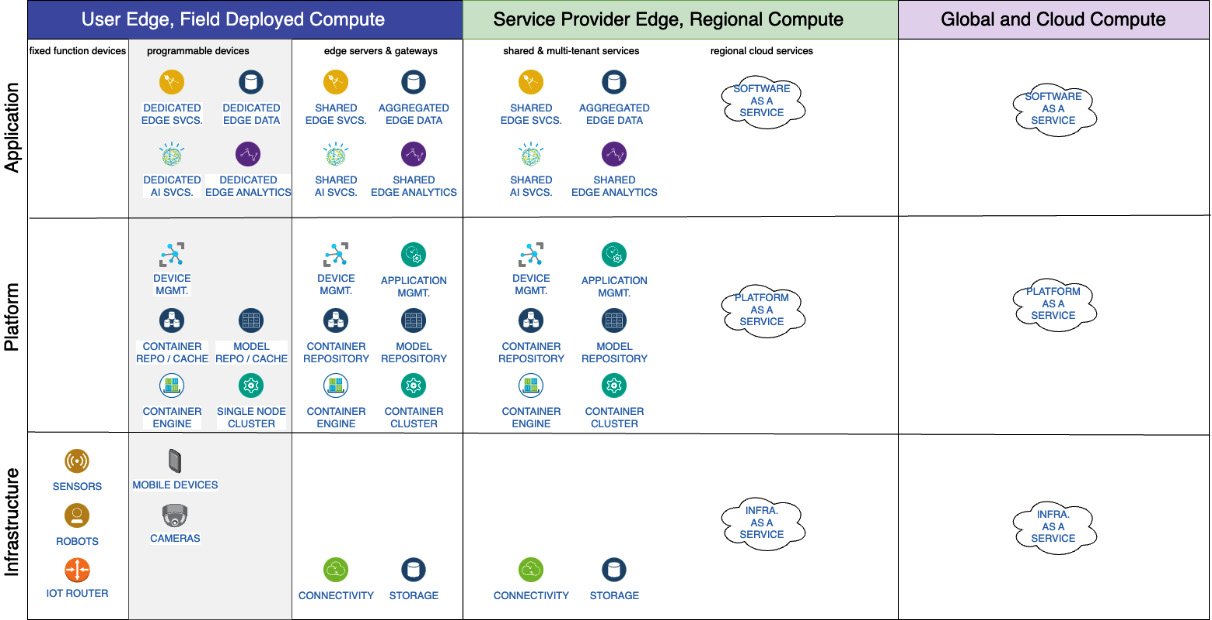


Figure 3.4 – Architectural elements for the edge device hub archetype pattern

## Architectural element categories

Let’s begin looking at the preceding diagram, starting with the columns moving from left to right. The major columns are for the user edge, the SP edge, and the cloud. The major rows are divided into common cloud layers of applications, platforms, and infrastructure. This grid of rows and columns allows you to observe elements by purpose and simultaneously compare and contrast groups of elements with those in similar locations or those with a similar purpose.

Starting at the top left, the sub-categories of the user edge are:

* **Fixed-function or constrained devices**, which contain single-purpose hardware that cannot run containers
* **Programmable devices**, which contain hardware that supports Linux host machines and devices capable of running container images
* **Edge servers and gateways**, which typically provide shared services, capabilities, and functionality to all hosts and devices in a facility or deployment locality

Next, in the top center, the sub-categories of the SP edge are:

* **Shared and multi-tenant services**, which provide shared services, capabilities, and functionality to multiple locations in a region. They may also provide cross-region failover, additional scalability for local solutions that the user edge may not have, and burst-to-region capabilities from the user edge.
* **Regional cloud services**, which may provide **Infrastructure-as-a-Service** (**IaaS**), **Platform-as-a-Service** (**PaaS**), and **Software-as-a-Service** (**SaaS**) solutions, especially to locations with regional or other geography-imposed requirements.

Last, in the top-right column, global and cloud compute provide bare-metal, managed services, and **as-a-Service** (**aaS**) solutions.

The categories are sometimes overlapping, may sometimes have gaps, and will not be exhaustive. They are meant to be easy to understand, clearly indicate best practices, and illustrate an approach that will likely lead to successful software architecture outcomes. As such, it may not accommodate all edge and corner cases.

### User-edge elements

Let’s take a more detailed look at the architectural elements listed in each of the categories falling under user edge. We will discuss specific elements, options, and rules of thumb.

#### Constrained devices

Beginning with the fixed-function devices category, there are no applications or platforms listed since applications and platforms are software-based solutions, and this category is about hardware-imposed constraints. Moving to infrastructure, you’ll notice that there are a few items shown. That is because there are many types of devices in this category and those shown are only representative in nature.

The functional types of IoT devices are display, sensor, actuator, emitter, transmitter, receiver, controller, processor, and some combinations thereof. The devices may have their functionality hardwired or fixed, be updatable to some extent, and be programmable to some extent.

*RULE OF THUMB*

*It is important to note that if an IoT device can host the services supporting an application, it is de facto an edge computing device and considered programmable or dynamic, not fixed function, in nature. A microcontroller, for example, cannot do that.*

The best placement decisions for these IoT devices are as close to where they need to be used as possible. Considerations should include determining if the device in question should be placed in the location permanently, if it may need to move occasionally, if it needs to be relocatable on demand, if it has environmental requirements, and if it has a range of movement or is otherwise mobile or articulated.

#### Programmable devices

In this category, the applications, platforms, and infrastructure should be dedicated only to the needs of the device and possibly any southbound connected or managed fixed-function devices. This is because programmable devices are either dedicated to a specific task or could be otherwise constrained and are not suitable to also function as a server running shared services.

Therefore, our recommendation is to never run shared services on programmable devices unless that device’s express purpose is to be a server appliance. Examples of that exception – a programmable device that is also a server appliance – would be a hard drive designed to be a file and/or media server, and an edge appliance designed for and dedicated to managing a fleet of fixed-function devices. Likewise, services on the devices are likely to be stateless.

Beginning in the applications row, programmable devices will likely be required to run applications and services of all types … whatever the purpose of the device required. That means that it will be rare that the applications on the device will be limited to a single role or type. The diagram depicts applications related to services, data storage, **artificial intelligence** (**AI**), **machine learning** (**ML**), and analytics processing. It makes the most sense to run all required applications on the device itself unless the available storage and processing resources do not permit this.

*RULE OF THUMB*

*Keep in mind that not all applications need to be running at all times, so consider if you should schedule some to be run based on when they are needed. This will also determine if you need to use an application management solution that permits time-based, location-based, and mobility status or mode-based execution constraints.*

In the platforms row, you will find device management, caches and repositories, and container execution environments. Again, these are the most common platforms to be needed for the edge device hub pattern.

The device management element may be a gateway capable of protocol conversion that provides device protocol support, communication, and translation. It could provide device management support including firmware deployment and system telemetry. It could also enable data filtering, a rules engine for events and alerts and complex data transformations, as well as ML model support for data insights and analytics.

It is unlikely that you will have an application management solution running on the device itself, but you may have an application management agent performing some of those duties.

Regarding container and model repositories, there are three options to consider depending on your needs and requirements. First and simplest, you may have a local (or even embedded) repository that is populated by a deployment pipeline. This is only useful if you have a handful of deployment targets that are usually continually connected to the internet. Second, you could have a local repository that is automatically populated by an application or model management system when it retrieves assets from an authoritative remote repository. This is helpful for situations when the device loses connectivity if the management system is configured to check the local repository first or as an automated failover. Third, you may have a caching proxy that impersonates the remote repository. This approach would allow you to create a pyramid structure of caches and repositories for extreme scalability. Or, you could replicate assets horizontally between peered repositories locally or regionally.

For container execution environments, you would likely have Docker or Podman to run containers locally. Some approaches may call for standardizing all environments on the K8s API, in which case you should consider lightweight K8s options such as K3s, KubeEdge, and MicroShift. But even so, Kubernetes clusters have significant overhead requirements and incur a performance penalty over container engines. Note that there are also tools that bridge the gap between the two layers, such as Eclipse ioFog, which uses the Docker runtime locally but can integrate with a remote K8s cluster.

In your container environments, consider how you will connect to individual devices. Initiating outbound connections is preferable since allowing inbound connections to a device increases the potential attack surface. To do this, you would use an application management solution that employs autonomous agents to manage applications locally in these environments and to periodically poll outbound connections. This approach also ensures resilience because the management solution does not need an active network connection to function. However, mission-critical deployments must utilize **out-of-band management** (**OOBM**) with a dedicated network, not by sharing the same network used for operations.

In the infrastructure row, you’ll find the devices. These devices could be robots, RFID scanners, scales, or lights. Common locations for these devices include residences, offices, roads and infrastructure, commercial buildings, factories, in vessels, and on other craft.

#### Edge servers and gateways

In this category, servers host edge services, AI services, and analytics for multiple edge devices at that deployment location. Any data collected on the server is usually aggregated rather than stored separately to allow reporting on the facility or location as well as sub-units/floors/buildings/sections or individual devices or rooms.

Edge services placed on servers may be stateful, so consider using onboard or local, rather than remote, storage to increase resiliency and decrease latency.

In the platforms row, the device management solution covers the management of fixed-function as well as programmable devices and is the ideal location for both. The server class of edge devices would be expected to be sized to handle any potential growth in the number of deployed devices as well as any increase in running supporting services for those fleets.

Similarly, the server is the ideal location for field-deployed application management services. The server is expected to have both the capacity to run the application management platform and high bandwidth and throughput with low latency, which is ideal for the task. This may also allow the platform to continue running in situations where the local network is physically separate from the internet connection and in situations where the northbound connectivity is unreliable. However, if there are no local edge servers capable of running the application management platform, the SP edge may be the next best location for it.

For model and container repositories, an edge server located on-premises would be the best location for the purposes of latency, resiliency, and suitability. Since each deployment location or edge facility has unique attributes such as geography, purpose, and even governance/regulations, application configuration, ML models, and even, potentially, applications themselves may differ from those stored at other locations and thus require a separate repository instance.

In your container execution environments, you will likely require a multi-node Kubernetes cluster if your edge server capacity allows. This can be implemented on a single machine with each node in a separate **virtual machine** (**VM**), or on separate physical machines for optimum **high availability** (**HA**) and resiliency. This ability to run multiple copies of services on separate cluster nodes is another benefit of the edge server class of machines and ensures the availability of critical locally hosted services. Without this capability, services hosted locally on programmable devices would either lose functionality when a local service went down or would have to add complexity by implementing a solution whereby one device could fail over to a similar service on a nearby device (assuming that the services are fungible and not custom to each device).

In the infrastructure row, you will find connectivity and storage. The storage in this tier should be used primarily for data sharing between devices, for backups and **disaster recovery** (**DR**), and for archiving. Any specialty storage solutions, such as a data historian, would likely belong in the platforms row above and more likely at the SP edge rather than field deployed. Please note that we are using generalities here, and there are always exceptions, especially in cases of extreme scale or vertical specialties.

The connectivity shown here is assumed to be software-based (**software-defined networking**, or **SDN**). The connectivity provided could be in the form of an edge gateway device that features protocol conversion. In many commercial deployments such as a big-box retailer or distribution center, or industrial deployments such as large factories, the connectivity may include private LTE or 5G enabling solutions. In farms and outdoor settings, it may include mesh networks and LoRaWAN installations and support. At stadiums, it can include complete telephony deployments, although those particular locations may rightly be considered satellite locations in the SP edge.

Wrapping up this category, it is important that we mention that edge server deployments do not always have a fixed placement. The deployment locations may have mobility and could include vehicles, boats, and satellites. The **Mayflower Autonomous Ship** (**MAS**) is a good example of an unmanned vessel with a few constrained edge servers (see more about it at https://mas400.com/). The MAS is a floating sensor platform operated by constrained programmable edge servers running services in a container engine. The field-deployed edge elements are shown in Figure 3.5. For a more complete view, see slide 6 (Solution Architecture Details) from a presentation of the Open Horizon open source project at <https://wiki.lfedge.org/display/LE/2020+Fall+Kickoff+Virtual+Event+Series?preview=%2F29892870%2F29906467%2FMayflower+LF+Edge+2020-10-01+v2_compressed.pdf>:

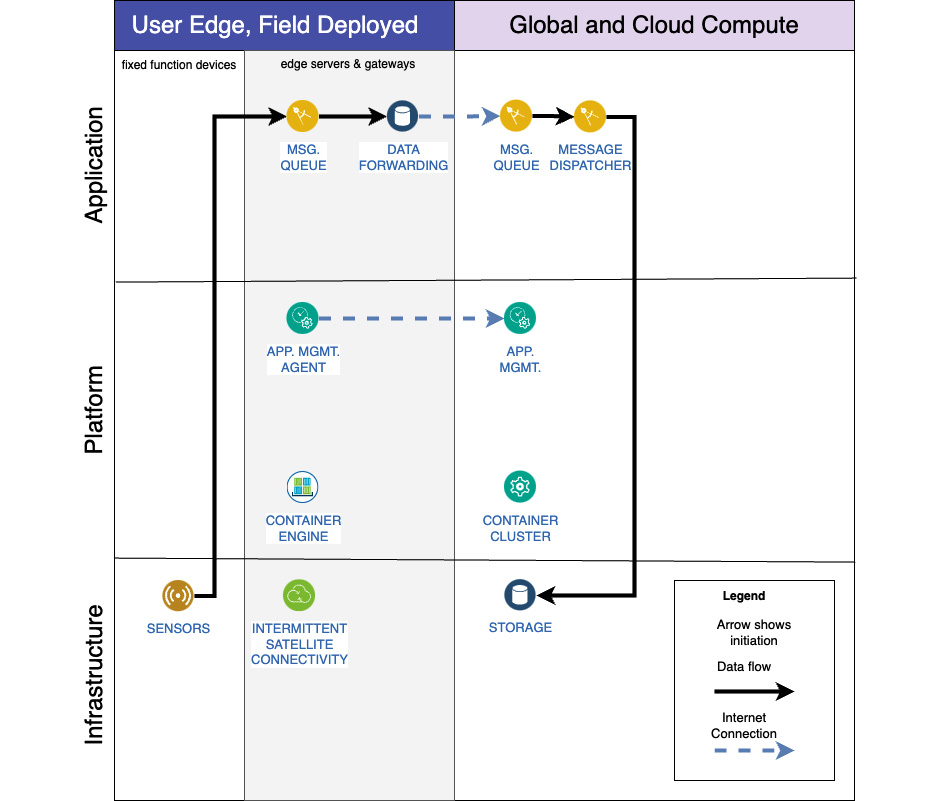


Figure 3.5 – MAS edge solution architecture partial view

An important point to note from the preceding diagram is that there is no benefit to using regional compute over cloud compute resources. Since the boat is sporadically connected to the internet using a low-throughput, high-latency connection, the ability of the cloud to scale and the availability of resources outweigh the potential benefit of additional connection latency differences between boat-to-region versus boat-to-cloud. Additionally, since the boat is traveling, there is no single region that would provide an advantage much of the time.

A second point is that the solution is highly secure. There are no open inbound ports on the vessel. All connections are initiated outbound, and none are kept open for a significant amount of time. This is possible because, other than the mission objectives, all commands are determined, planned, and executed locally on the vessel. And not only are all data connections secured and messages encrypted, but the application management solution also uses **perfect forward secrecy** (**PFS**), which ensures that each session uses new keys.

# Edge devices versus edge hub

In this section, we will introduce and cover the edge device hub archetype pattern. We will recommend our preferred order of placement and our reasons for that order, including benefits and drawbacks. We will briefly show how this pattern is applied to some real-world deployments.

*NOTE*

*An important note before we continue: in this chapter, we do not discuss the network underlay or overlay, topology decisions and optimizations, recommendations, or best practices. For the purpose of focusing on the edge devices and their supporting services, the network is presumed to exist. Later chapters in this book will touch on those matters.*

In Figure 3.6, we introduce the first archetype pattern. By now, you should be oriented to the style and layout of the chart since we have been building up to it with the previous diagrams:

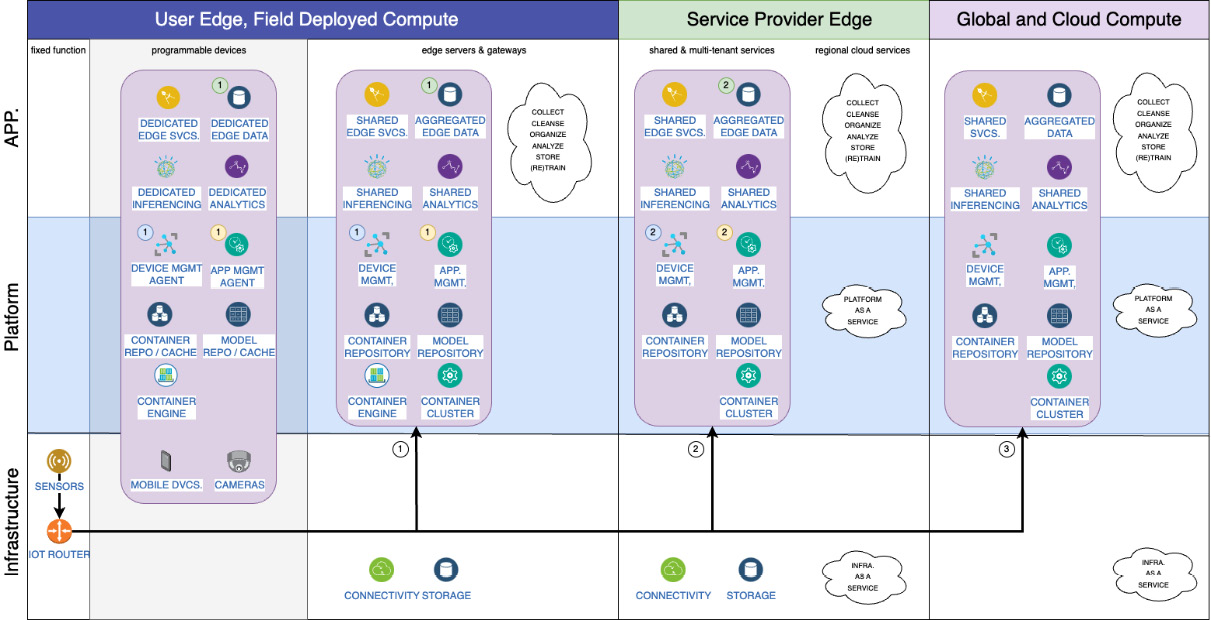


Figure 3.6 – Edge device hub archetype pattern

Let’s delve into the details of Figure 3.6 and describe the main points of interest while showing how to use it.

## Reviewing the pattern

This pattern and the recommendations we make are designed for best performance, resilience, and scalability. They may not always be the least expensive or simplest options to manage, but automation and autonomous solutions will mitigate most management issues and could ultimately make deployments easier to manage by fewer persons while providing more configuration flexibility.

In [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013), we discussed the various edges, cloud-out versus edge-in, and the concept of an edge cloud. Building on those discussions, let’s discuss the trade-offs between running all workloads on the far edge compared to using edge servers and regional compute as mini clouds with lower latencies than global compute using the edge device hub pattern.

If you have fixed-function devices on-premises, the best option for low-latency data transfer and the quickest surfacing of event notifications would be to perform as much inferencing and analytics in the field as possible. This will also lead to more resilience when you encounter upstream connectivity issues. However, you may not have enough computing capacity, power, or storage at that location. In that scenario, offloading to the SP edge becomes the next best option unless there are no geographically or topologically proximate regional computing facilities available. Offloading to the cloud would be the least favorable option overall unless timeliness is not critical and your budget is. Just beware of data egress charges from cloud providers, which could erase any potential savings.

For programmable devices, we recommend performing as much work on the device itself as practical, and then the rest on shared edge servers. In an environment with mixed fixed-function and programmable devices, you may want to weigh the mix of analytics workloads more heavily onto the edge servers while keeping most of the inferencing on the programmable devices for the best outcomes. Device and application management should be placed in the most central location to the devices, with only agents running on the programmable devices unless those devices are designed specifically for that purpose (such as a server appliance that also happens to be a programmable device). Another factor to consider is resiliency. If you must process captured data immediately, the constrained devices will need to perform more of the computation and then buffer the output.

For data storage, we recommend leaving data where it is originally stored/recorded/placed for as long as possible and then archiving locally, then regionally, and then globally, in that order of preference. The reason for this recommendation is based on the twin assumptions that most of this data may not be used and is ultimately too expensive to transfer given its inherent value. Additionally, retaining data close to the source minimizes data privacy and governance risks. In order to best utilize that data, we recommend using a solution that enables dynamic and ad hoc querying and data virtualization from any connected node to any arbitrary group of nodes. For more details on this topic, see [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110).

For application management, we recommend using a policy-based, no-code, autonomous solution that initiates connections from the edge devices to the server. This approach will allow flexible and scalable solutions while minimizing any potential attack surface. It also helps to ensure disconnected operations without human intervention. Solutions that use this approach include Open Horizon and RHEL Device Edge coupled with Ansible Pull.

For initial deployments on day 1, we recommend using a solution that supports the FDO specification. See the link in the Suggested pre-reading material section at the start of the chapter to learn more. There is also an FDO project at the Linux Foundation that provides a reference implementation. FDO support allows zero-touch provisioning and deployments of most devices while ensuring proper device ownership and attestation.

Last, for those interested in edge computing in contested environments, the preceding recommendations should also serve as a solid foundation for tactical edge deployments.

## Self-propelled inspection robot example

The following real-world site inspection example solution uses a self-propelled robot – Spot from Boston Dynamics. The solution uses the edge device hub pattern and spotlights the strengths of this approach. Note that the portion of the solution running on the edge server could also run in the SP edge or in the cloud, but it is more performant and resilient when run on-premises.

The robot in this example can travel on pre-programmed routes, including climbing stairs and using elevators. It is sophisticated enough to autonomously recognize obstacles and routes around them. It can also recognize humans and yield to them. It can typically complete an inspection mission in 30 minutes or less on a single charge. The route takes the robot through areas of the premises without connectivity or with severe signal degradation.

The robot can be outfitted with a sensor and/or manipulator pack and uses an attached “backpack” consisting of a low-power edge server. In this scenario, the robot is used for two tasks: checking on fire extinguishers to ensure proper placement, charge, access, and condition with a standard camera and local inferencing; and inspecting electrical panels with a thermal camera and visual inferencing to detect potential hot spots. If a potential issue is detected with a high degree of certainty, a work ticket will be opened upon restoration of network connectivity. See Figure 3.7 for a diagram of the solution:

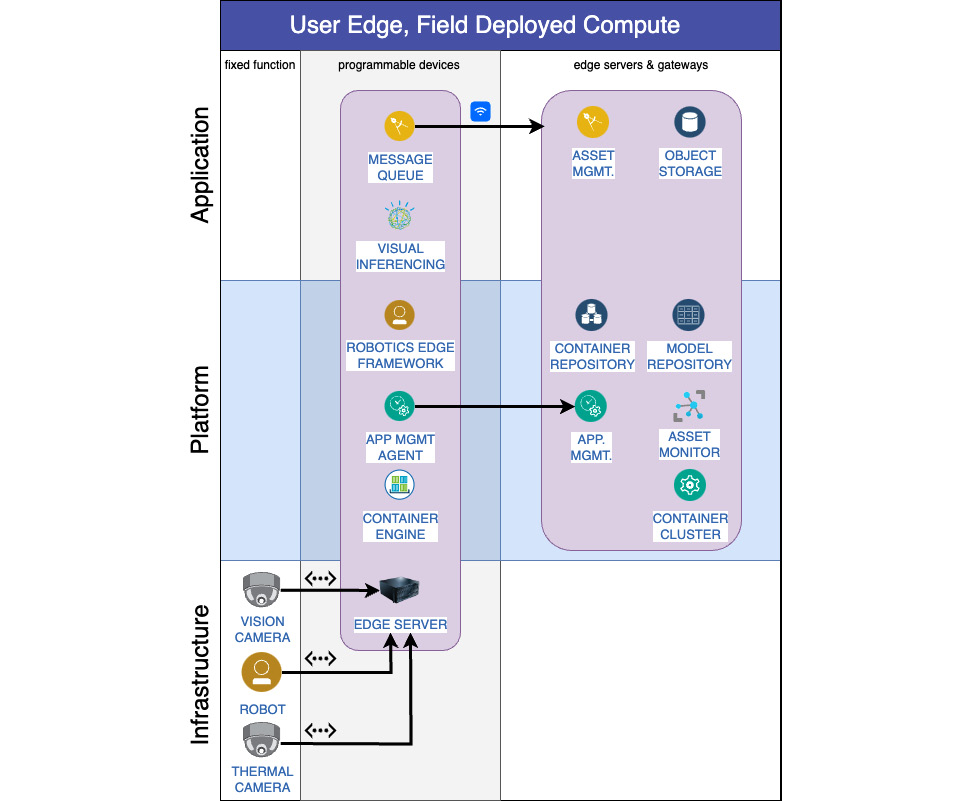


Figure 3.7 – Site inspection solution using edge device hub pattern

You’ve now seen the archetype pattern, discussed its component elements, and seen an example of a real implementation. Now, let’s discuss one of the reasons why edge-native services and applications work so well when deployed to most locations and hardware … containers.

# Containers

In this section, we will cover the importance of using containers when developing applications and services. You will learn about how this innovation propelled edge computing to wide adoption. We will briefly discuss serverless computing (AKA cloud functions), and we will touch on the promise of Wasm and the **Wasm System Interface** (**WASI**) for edge computing.

As mentioned in [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013)’s Cloud-out versus edge-in section, software development for edge computing benefitted greatly from the best practices developed for the cloud environment. A large aspect of that benefit was the use of **service-oriented** (**SO**) software architectures, followed by containerization.

Containerized applications use principles of isolation and abstraction to remove some host and operating system dependencies from the packaged applications that need to run in those environments. This also allows applications with conflicting dependencies to run on the same host without contention. This abstraction from the host enabled an efficient use of system resources superior to some native applications and to those running in VMs. But arguably the best consequence of this approach is the simplicity of placing, running, and removing containerized applications from host machines. A close second-best consequence is the relative ease of developing and deploying applications that support multiple architectures.

Building and supporting software for fixed-function IoT devices, also referred to as embedded systems, has a long list of known pain points. Chief among them, according to Mitch Maiman (<https://www.electronicdesign.com/technologies/embedded/article/21165541/intelligent-product-solutions-pain-points-for-embedded-software-design>) are limited device resources, unstable drivers, the frequency of required firmware updates, unstable development toolsets, incompatible processors within the same micro-architecture, and a shortage of talented developers. This translates into high operating expenses for the business and a manually intensive development and deployment cycle. With those issues in mind, it becomes easy to see the relative advantages of replacing fixed-function and constrained devices with programmable devices wherever possible.

One emerging solution that marries the ease of development, execution environment, and portability of containerized applications with the ability to run in constrained environments is the Wasm approach. While initially developed for web browsers, Wasm now has several runtimes available for embedded systems that facilitate edge-native software development best practices.

*NOTE*

*Alternatively, Rust programs can be compiled into native code and executed on a bare-metal device. This approach provides an alternative to***real-time operating systems***(***RTOSes***).*

In the near future, we anticipate software development processes allowing containerized application development to generate a single artifact that can be run in both container engines and by Wasm runtimes. Docker has taken the first step in that direction by enabling native Wasm execution within the Docker engine. As a side note, Wasm shows some promise in being a potential technology for powering serverless on the edge, or edge functions, to coin a term.

Serverless computing, or cloud functions, is an approach that uses the sandboxed approach of containers and has applied constraints and trade-offs in order to achieve quick application instantiation. This approach intended to remove the requirement for a persistent application and instead create an ephemeral resource that could be spun up when requested and then immediately removed when finished. As such, serverless applications were intended to be stateless.

When you have a large pool of computing resources available and an expectation of limitless scalability, serverless becomes an inexpensive and efficient way to use those resources. However, the quick instantiation times can strain the less powerful machines sometimes used for edge computing. Edge architectures, especially those not intended for edge servers, have already factored out the requirements of hyper-scalability. Therefore, there is not usually a need for serverless resources in any solutions except those used in edge and enterprise clouds, and those environments are built to handle the demands of serverless computing. As a result, there has been little to no innovation in exploring serverless computing on constrained edge devices.

# Disconnected operations

In this section, we will cover unique options and opportunities for not only planning for interrupted network connections but actively planning for those scenarios as an operating requirement. You will learn how some organizations anticipate and respond to those situations in three separate approaches.

In the Delving into edge-in architectures section of [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013), the first practice listed was Tolerate interruptions and unavailability of service dependencies and connectivity. In the edge device hub pattern, you should expect that each individual connection might be unavailable and plan for that eventuality. That may mean planning for devices to lose connectivity to their hub and for performance fluctuations. If that happens, do you plan for the devices to buffer enough untransmitted data, accept the potential data loss, or take some other approach? That may mean that the hub cannot provide the data in response to remote queries or cannot re-transmit the data northbound. It is also important to set expectations regarding the maximum length of time that an outage can occur before unrecoverable data loss happens. Bandwidth, throughput, and latency will vary at any time in the field. This expectation should underpin the architecture.

The Nexoedge open source project (<https://lfedge.org/projects/nexoedge/>) is designed for scenarios where partial network outages may occur to remote storage. By storing partial copies of the data in separate locations, you ensure that all data will still be available even if connectivity is interrupted in one of those remote locations. The drawbacks to this approach are that the data is not local, and thus retrieval will incur data transmission costs and latency, and that it is still vulnerable to a full network outage.

The AgriRegio project (<https://agriregio.peasec.de/>) uses an approach it calls an “offline-first principle” and designs systems and applications to function without any connectivity as its primary mode of operation. All usage, storage, and connections to remote sensors operate without an internet connection. The only functionality that is not present in those scenarios is installing or upgrading applications, and any optional northbound data transmission.

The Liquid Prep open source project (<https://github.com/Liquid-Prep>) creates an ad hoc network mesh using any capabilities present in each device to forward messages to a local hub. The mesh does not require any member to be connected to the internet, and each member can store a few days’ worth of data while disconnected. Once connectivity is restored to the hub, accumulated messages are passed along. If the outage lasts longer than the available storage, the **first in, first out** (**FIFO**) queue removes the oldest messages first.

In general, assume that connectivity will not always be present. And if possible, build applications and architectures that do not break when connectivity is removed. For more discussion on this topic, see [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110).

# Summary

In this chapter, we covered the edge device hub architecture pattern for managing and enabling IoT-type sensors and smart devices and discussed the impacts of placement trade-off decisions. We showed how containerized applications are key in these edge architectures and touched on the role serverless applications could play. You should also have an idea of how you zero-touch deploy edge applications on day 1.

In the next chapter, we will dive into the SP edge and discuss how edge computing affects network overlays and **communication SPs** (**CSPs**).

# 4

# Network Edge Architecture

**Communication service providers** (**CSPs**) and telcos are mostly interested in network edge solutions because they offer them a way to monetize their investments in 5G infrastructure. While CSPs have built their business on operating physical networks tied to specific geographies, telcos realize that they have access to the same public cloud infrastructure as every other business. That is significant because we will see in this chapter that software is being used to define networks, especially in the era of 5G and beyond.

In [Chapter 3](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057), we described the basic edge architecture. In this chapter, we will explore network edge architectures and learn how they address, among other features, security and reliability, latency considerations, and storage constraints. The new paradigm is to place compute power at the network edge, thus making it possible to intelligently manage workload placement. The main topics that will be covered are as follows:

* **Network functions virtualization** (**NFV**)
* **Software-defined networking** (**SDN**)
* Underlay and overlay networks
* Network traffic management
* **Multi-access edge computing** (**MEC**)
* Network edge architecture
* Sample architectures

We see that CSPs want to become **digital SPs** (**DSPs**) because hyperscalers are starting to offer software-based network services via cloud-native functionality. This is forcing CSPs to embrace cloud and virtualization as they think about offering new types of 5G and edge computing services.

# Definitions

From a **network function** (**NF**) to a DSP, a few definitions are in order before proceeding.

One or more communication connections between different computing devices comprise a network. An NF, as we know it, is the transmission of data between those physical devices while adhering to certain rules. For instance:

* Distribution of data to a pool of servers by a load balancer is an example of an NF
* Filtering of data and deciding which data is safe to consume by a firewall is another example of an NF

As the name suggests, a **virtual NF** (**VNF**) is a virtualized version where the NF is implemented using software.

Gartner (<https://www.gartner.com/en/information-technology/glossary/csp-communications-service-provider>) defines a CSP as someone who offers telecommunications services leveraging the landline or wireless network infrastructure. While CSP is a generic term, you will hear **internet SP** (**ISP**), which provides internet services, or **telecommunication SP** (**TSP**), which also offers cable and satellite services.

A DSP is the next step in the evolution of a CSP, whereby companies deliver digital content over the network. Examples of such online services are Apple Music, Spotify YouTube, and so on, and iTunes or Amazon Music are examples of digital stores where people can access digital content.

# NFV

Let’s take a short trip down memory lane. Media gateways, routers, **home location registers** (**HLRs**), **IP Multimedia Subsystem** (**IMS**), and so on were components that made up legacy telecom systems. Customized hardware, operating systems, and other software were required to offer telecom services to customers. These legacy network devices were very expensive and had high operating costs. Therefore, customers ended up with high **operating expenses** (**OPEX**) and **capital expenses** (**CAPEX**). That resulted in a lock-in with long-term contracts with the telcos for support and maintenance. Another byproduct of such environments was that scalability came at a premium and self-healing capability was minimal or non-existent.

IP-based technologies gradually replaced legacy networks, to the point where even legacy services such as voice (phone) communications are now delivered over IP (VoIP). This led to the ubiquitous wireless data networks of today, of which 5G is the latest incarnation. Telcos started prioritizing the adoption of cloud-native computing. That was a paradigm shift by the **network equipment providers** (**NEPs**) to offer VNFs. These are virtualized versions of legacy **physical NFs** (**PNF**s) designed to run in **virtual machines** (**VMs**) or containers on commodity hardware. Such virtualization using software leads to agile networks, provides scalability, self-healing, and closed-loop assurance, and comes with significant OPEX and CAPEX savings.

NFV, although very similar and often used interchangeably with VNF, refers to the running of SDN functions on commodity hardware. This network virtualization initiative has been led by some of the world’s biggest telecom operators.

*CARRIER NETWORK*

*A carrier network is a rather complex system that connects various network devices to transmit data from one location to another. One of the main challenges for network vendors is maintaining their***service-level agreements***(***SLAs***). Carrier networks are being transformed by NFV because it helps vendors maintain their SLA guarantees.*

The NFV specification is managed by the **European Telecommunications Standards Institute** (**ETSI**). A standard diagram showing the VNF and NFV layers is depicted in Figure 4.1 (Source: <https://www.etsi.org/technologies/nfv>):

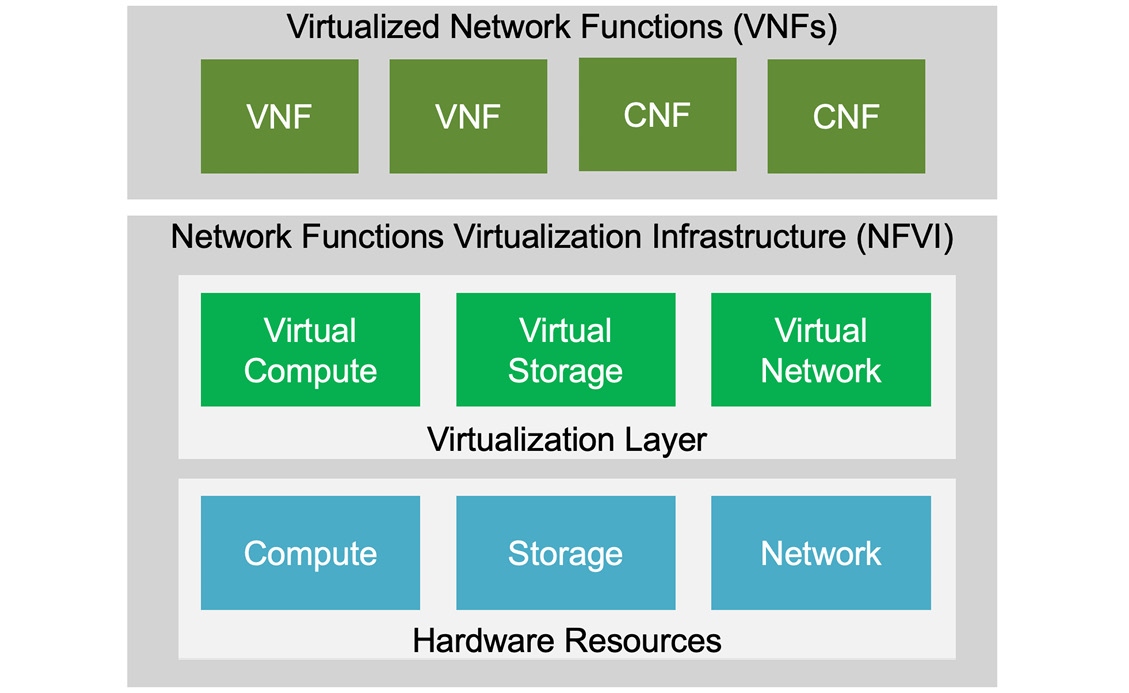


Figure 4.1 – NFV and VNF layers

*NOTE*

*ETSI is an independent non-profit standards organization in the field of information and communications.*

There are three layers in this network edge platform architecture, starting from the bottom: hardware resources, virtualization layer, and virtualized functions. They have to do with decoupling. The NFV decouples NFs from the hardware, which could be proprietary, and runs them as software. Other functions such as firewalls, traffic control, and virtual routing are run as VNFs.

The VNFs are deployed as VMs running Linux on commercial hardware. The traditional network appliances that might be running on proprietary hardware are also known as PNFs. You also see boxes named **CNF**. These are cloud-native NFs, which are containerized versions of VNFs.

Two other important areas in an NFV architecture are not described here because they are not that relevant to edge computing – namely, **management and orchestration** (commonly referred to as **MANO**), and **operations and business support systems** (**OSS/BSS**).

Source: [https://www.ibm.com/cloud/architecture/architectures/network-automation](https://www.ibm.com/cloud/architecture/architectures/network-automation%20)

## NFV considerations

At a high level, the different classes of NFV workloads as put forth by ETSI are:

* **Data-plane workloads**: Related to packet handling
* **Control-plane workloads**: Related to session management, routing, or authentication
* **Signal processing workloads**: Related to digital processing
* **Storage workloads**: Related to disk storage

Organizations have strong controls over their physical network, but when considering an NFV solution, architects must pay attention to their security posture. At a minimum, look at these three areas:

* **Data encryption**: Encrypt data in storage, when transmitted, and when in use
* **Security key management**: Securely store and manage keys associated with data encryption
* **Access control**: Establish access controls to all aspects of the virtual network

We have talked about cloud-native and virtualization of NFs but not in the same breath. Since it was introduced in 2012, applying NFV principles to core network infrastructure has led to more agile and cost-efficient network deployments. However, these VNFs have traditionally been implemented using VMs. The question quite often asked by hyperscalers is, Can it all run in containers? In Figure 4.1, we even depicted **cloud-native network functions** (**CNFs**), and while the move to containerize VNFs has started, certain telecom applications require very high-performing networks in terms of throughput, and latency and telcos are taking their time to create containerized versions of their services. Another subtle but important reason is that VMs have been proven to be more secure in networking topologies when compared to containers because of the isolation provided by hypervisors at a system level. One advantage of VMs is that they provide better integration with hardware. This is critical for data acquisition and processing, for example. Containers abstract the underlying hardware, which makes them more portable but less integrated. To understand the difference between VM and container security, check out this URL: <https://www.techtarget.com/searchsecurity/tip/Container-vs-VM-security-Which-is-better>.

# SDN

As the term implies, software is used to provide NFs. It is done by using **application programming interfaces** (**APIs**) to communicate with network hardware and direct network traffic over a virtual overlay network. SDN aims to bring the benefits of cloud computing to network deployment and the management of networks by delivering them as code.

The devices, such as a router or switch, in a traditional network are only aware of the status of the network device next to it. SDN, on the other hand, can manage all the devices because of the centralization of network control.

Simply put, companies are using SDN because it’s a way to efficiently control network traffic and can be scaled as needed. SDN separates the control plane (that is, routing and packet forwarding functions) from the data plane (that is, underlying infrastructure). The brain of the SDN network is the centralized SDN controller, which offers a secure network since network administrators can set access policies from a central location across the entire network. The AI technology built into SDN controllers can detect periods of high network utilization and when they occur. Based on that, the SDN controller can request more processing be completed at the edge to alleviate network bottlenecks.

Without going into the details, it suffices to know that there is one component, the SDN controller, that is central to SDN architecture. See Figure 4.2, which also depicts three layers – the application layer, the control layer, and the data layer:

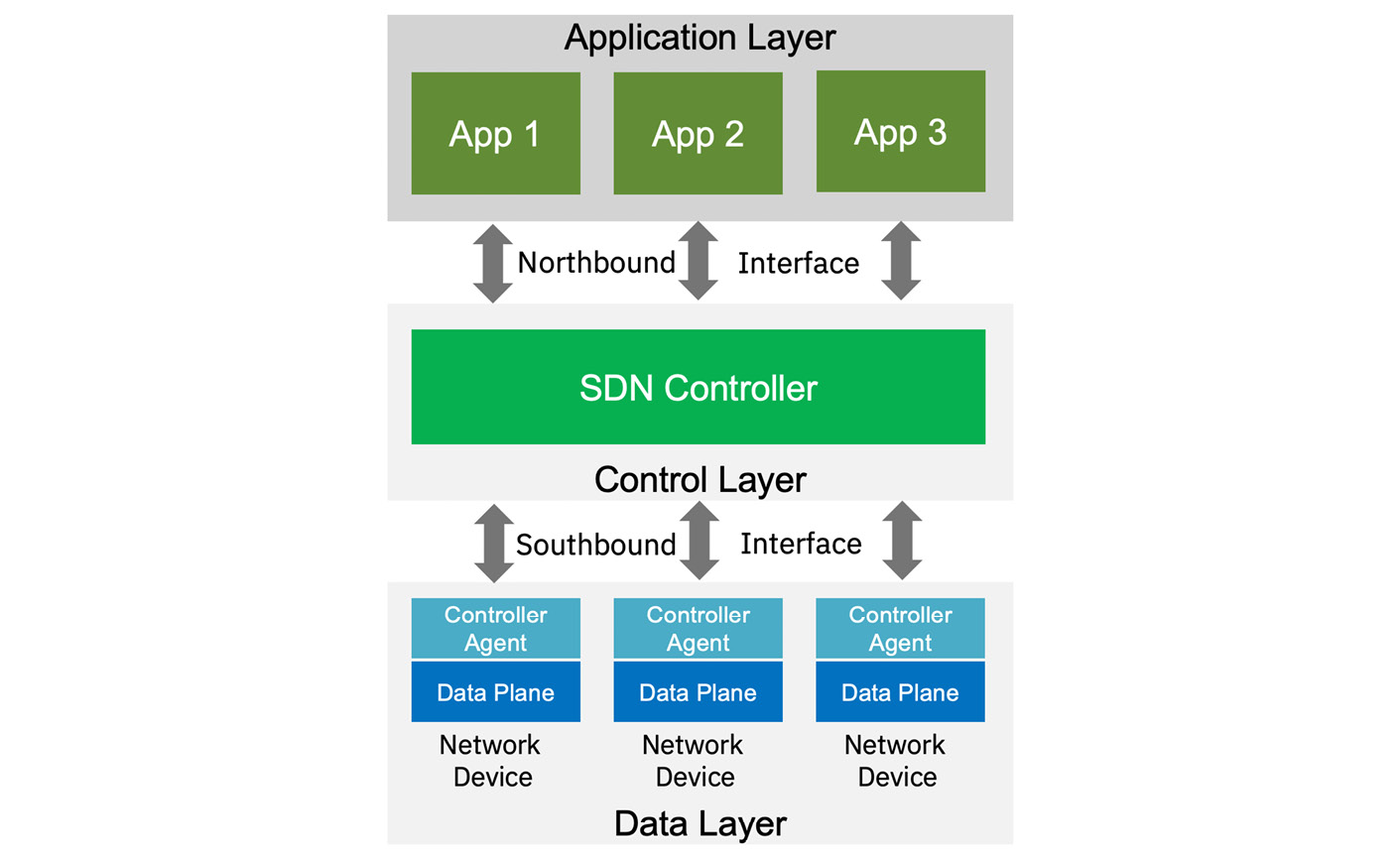


Figure 4.2 – SDN topology layers

The **data layer** is also known as the **infrastructure layer**. The northbound interface is typically a REST API or a web interface that lets the user or application layer communicate with the SDN controller, while the southbound interface allows the SDN controller or the control layer to interact with the network devices.

Because the SDN controller is centralized, it could be a potential **single point of failure** (**SPOF**). A solution architect has to be aware of that when designing an SDN solution topology.

## VNF, NFV, SDN, and edge computing

A VNF is not directly related to edge computing, but it helps in supporting emerging use cases such as AR/VR and image processing at the regional/network edge. Introducing VNFs at the edge of the network and near the end users is beneficial because it reduces end-to-end latency, accelerates **time to response** (**TTR**), and, most importantly, mitigates unnecessary utilization of the core network.

We saw how SDN, by way of the SDN controller, helps with cost-efficient networking and direct network management. That in turn could help improve efficiency and reduce latency in the realm of edge computing. SDN complements NFV because NFV moves services to a virtual layer and SDN helps control data packet routing through centralized management functions.

# Underlay and overlay networks

After all the talk about NFV and SDN, it behooves us to briefly describe underlay and overlay networks. This concept seems to follow the software engineering principle – solve any problem by introducing an extra level of indirection.

Very simply put, an **underlay network** is the underlying physical infrastructure of the network. An **overlay network** is a virtual logical network constructed on top of an underlay network using virtualization (see Figure 4.3):

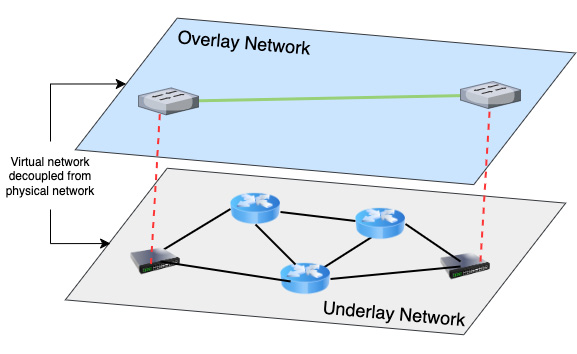


Figure 4.3 – Overlay and underlay networks

Routers, switches, firewalls, and servers are devices found in an underlay network, which are interconnected via routing protocols. In an overlay network, which is software-based, data is transmitted via virtual links. [Chapter 9](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_09.xhtml#_idTextAnchor172) compares the security aspects of underlay and overlay networks. Suffice it to say that overlay networks provide segmentation and isolation, which helps with security. They also simplify the management of network devices by providing more granularity to apply policies.

From an edge computing perspective, the overlay network has relevance because it provides a path to create logical networks that could be leveraged by applications and, especially, edge devices. This way, the overlay network can connect to thousands of devices more quickly since interaction with the physical network is not required and network administrators are not constrained by the physical network.

# Network traffic management

Speaking of networks, the question every telco wants to know is if they are meeting their SLA commitments. That is where network traffic reporting and management come into play. It is a way to determine and manage the health of a network by collecting real-time data from all network elements such as routers, switches, and so on, and endpoint devices such as laptops, mobile phones, and more. By monitoring, intercepting, and inspecting network traffic, telcos can direct traffic to an optimum resource based on certain **quality of service** (**QoS**) policies. This helps network administrators to alleviate congestion, reduce latency, and minimize packet loss.

While all these are tasks performed by network administrators, today’s networks are too complex to be managed manually. It is made possible by software known as **network management software**, also known as **network management systems**. The network management system uses some standard protocols to automatically collect information from various network devices. This is useful for tasks such as updating software or performance monitoring. Some of the network management protocols used are listed in Table 4.1:

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| SNMP | **Simple Network Management Protocol** (**SNMP**) is an open application layer protocol used to monitor the network and detect any network faults. It collects information about managed devices on IP networks for analysis. Examples of devices are routers, switches, cable modems, servers, and more. |
| ICMP | **Internet Control Message Protocol** (**ICMP**) is a TCP/IP network layer protocol. It is used by network devices, usually routers, to send operational information. It determines if data is reaching the intended destination at the right time. |
| Streaming network telemetry | Streaming network telemetry is a relatively new protocol that uses a push mechanism to send **key performance indicators** (**KPIs**) from network devices to the network management system. It sends it at a higher rate with lesser impact on the network devices compared to SNMP. |

Table 4.1 – Protocols used in network management

Given the components and the protocols used, we could visualize a high-level network management architecture, as shown in Figure 4.4:

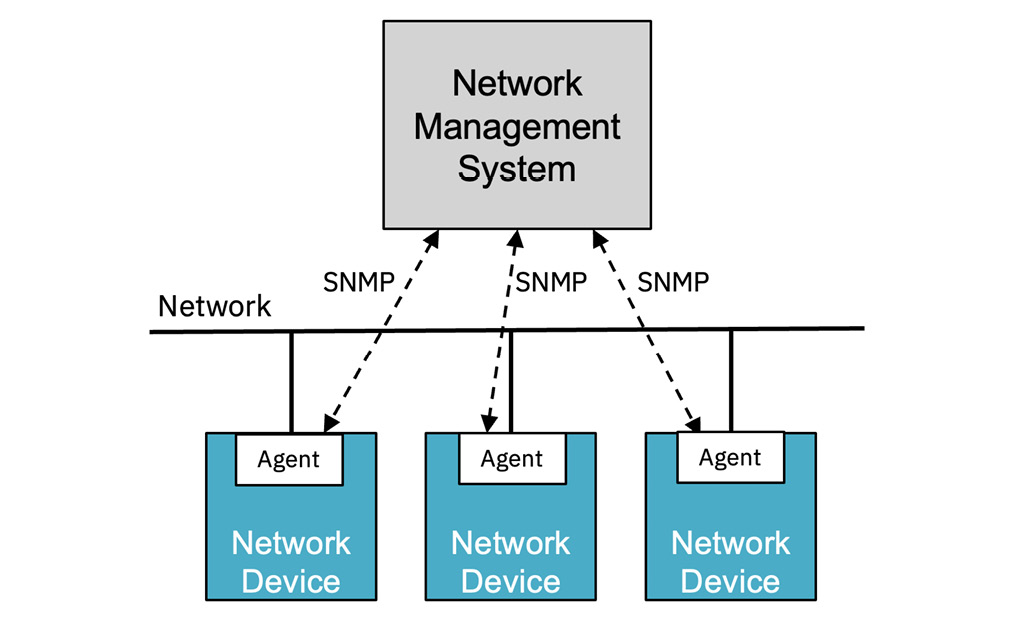


Figure 4.4 – Network management architecture

Network management systems are meant to be the eyes and ears of network administrators. They use AI/ML features to determine patterns in the network traffic flow to assist network administrators with many repetitive but vital tasks:

* Automating network maintenance and software updates
* Monitoring network performance and accelerating troubleshooting
* Identifying security threats

With respect to edge computing, incorporating AI/ML technology into network automation and management helps with the enforcement of network and edge placement policies because they would be based on past and existing network patterns. It can also help with the identification and classification of a multitude of devices on the network. For example, the system can determine if a separate network slice makes more sense to handle network traffic congestion and data loads.

# MEC

MEC is a network architecture that brings cloud capabilities to the edge of the network. It was formerly known as **mobile edge computing**. It is where mobile networks and the internet meet and hand off network traffic.

What started out as a concept is now also a standards framework developed by ETSI. They decided to expand the aperture by replacing “mobile” with “multi-access” because it was no longer just about mobile phones but had to do with the plethora of connected devices. MEC is about making edge devices, including IoT-type devices, smarter by running applications on them and making sense of the data that they generate.

This is where many of the previously described concepts converge. Figure 4.5 shows MEC positioned within a large edge architecture. As with everything else in the realm of edge computing, MEC is meant to reduce latency and ensure a highly efficient network service delivery in remote places. MEC-related tasks are meant to be performed in real time or near real time:

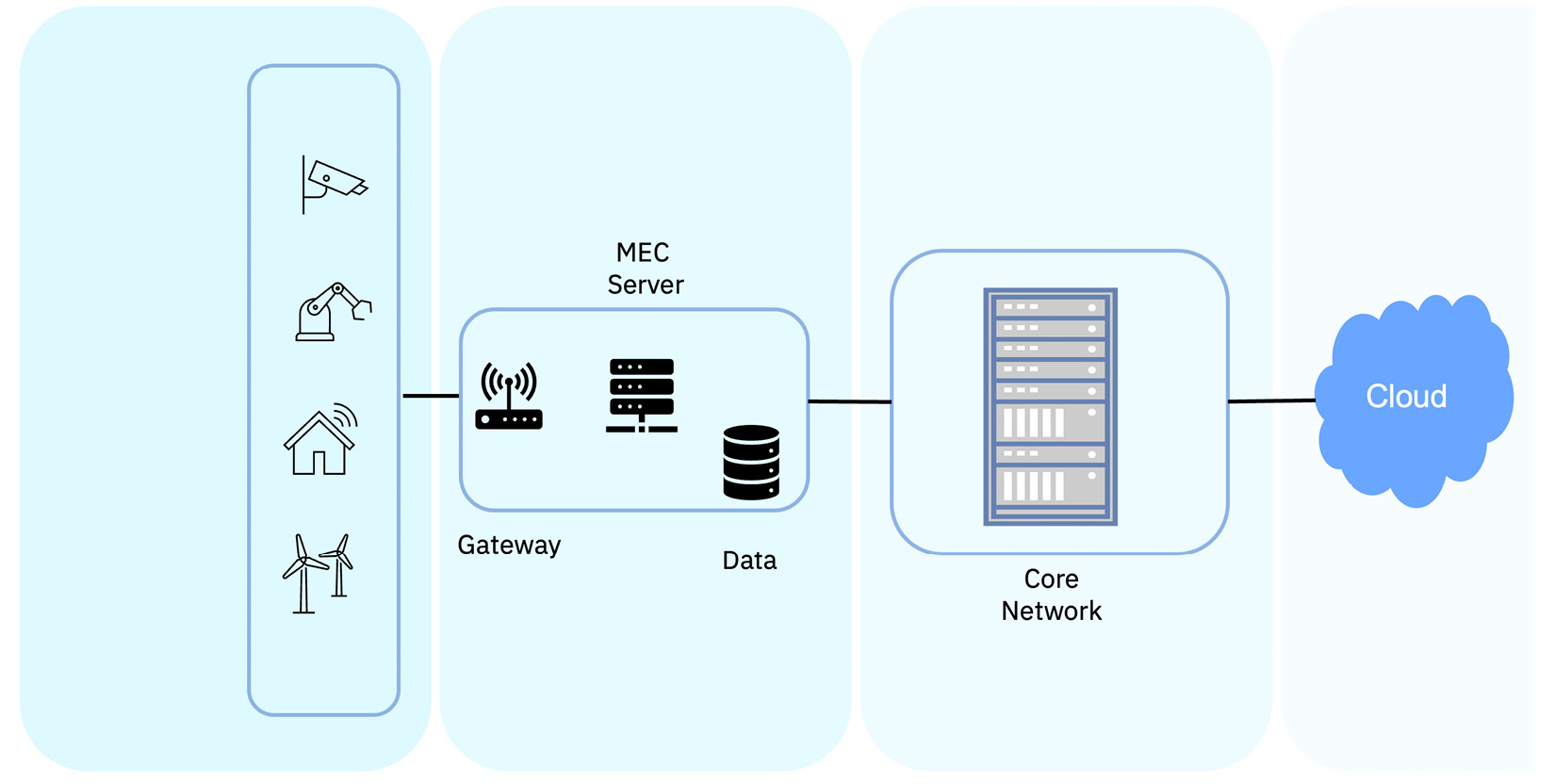


Figure 4.5 – MEC within the larger edge architecture

In the past, mobile applications running in cellular networks were placed in centrally located and serviced large **data centers** (**DCs**). Now, SPs have found good use cases for MEC because they can move workloads and services, which are now decoupled from physical devices, out of the core network to the edge of the network. An example is running workloads at cellular base stations, which are a lot closer to the user. Deploying applications on MEC units and the placement of MEC closer to the edge increases throughput and reduces latency.

MEC plays a key role in many edge solutions because, depending on the use case, it can be deployed in any of these three locations:

* Near the tower or base station that is closest to the end user
* In a central office or regional DC that is relatively the farthest point from the devices or **user equipment** (**UE**)
* Somewhere in between, which can act like a hub location

We will see later in the chapter example scenarios on how mobile operators are introducing 5G services and leveraging the same cloud-native infrastructure to run both MEC and **virtualized radio access networks** (**vRANs**) on the same commodity hardware.

# Network edge architecture

The network core, as depicted in Figure 4.4, is the infrastructure that runs and supports all the devices in an enterprise’s internal network. It is one of two major components in a wireless telecommunication network. The network edge is a collection of servers and devices that connect the company’s internal network to the internet. It can be on-premises or in the cloud.

[Chapter 3](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057) described the differences between legacy IoT-type architecture and cloud-native architecture. Telcos have a lot of legacy infrastructure and make extensive use of VMs on proprietary hardware in the compute layer. Part of their modernization journey includes the use of container technology running on **commercial-off-the-shelf** (**COTS**) hardware. As they adopt newer technologies, telcos and CSPs see the need to support some existing legacy systems as well as inject new technology. This blend of old and new technologies shows up in network edge architectures. Let’s look at some real-world scenarios.

## RAN

A RAN is the other major component in a wireless telecommunications system. It connects UE or electronic devices to the network via a radio link. We talked about the virtualization of NFs. The RAN can also be virtualized, and we see that happening more with the advent of 5G technology. A vRAN uses the same virtualization principles whereby the RAN NFs are virtualized and deployed on a containerized cloud platform. Figure 4.6 shows the virtualization levels in a RAN:

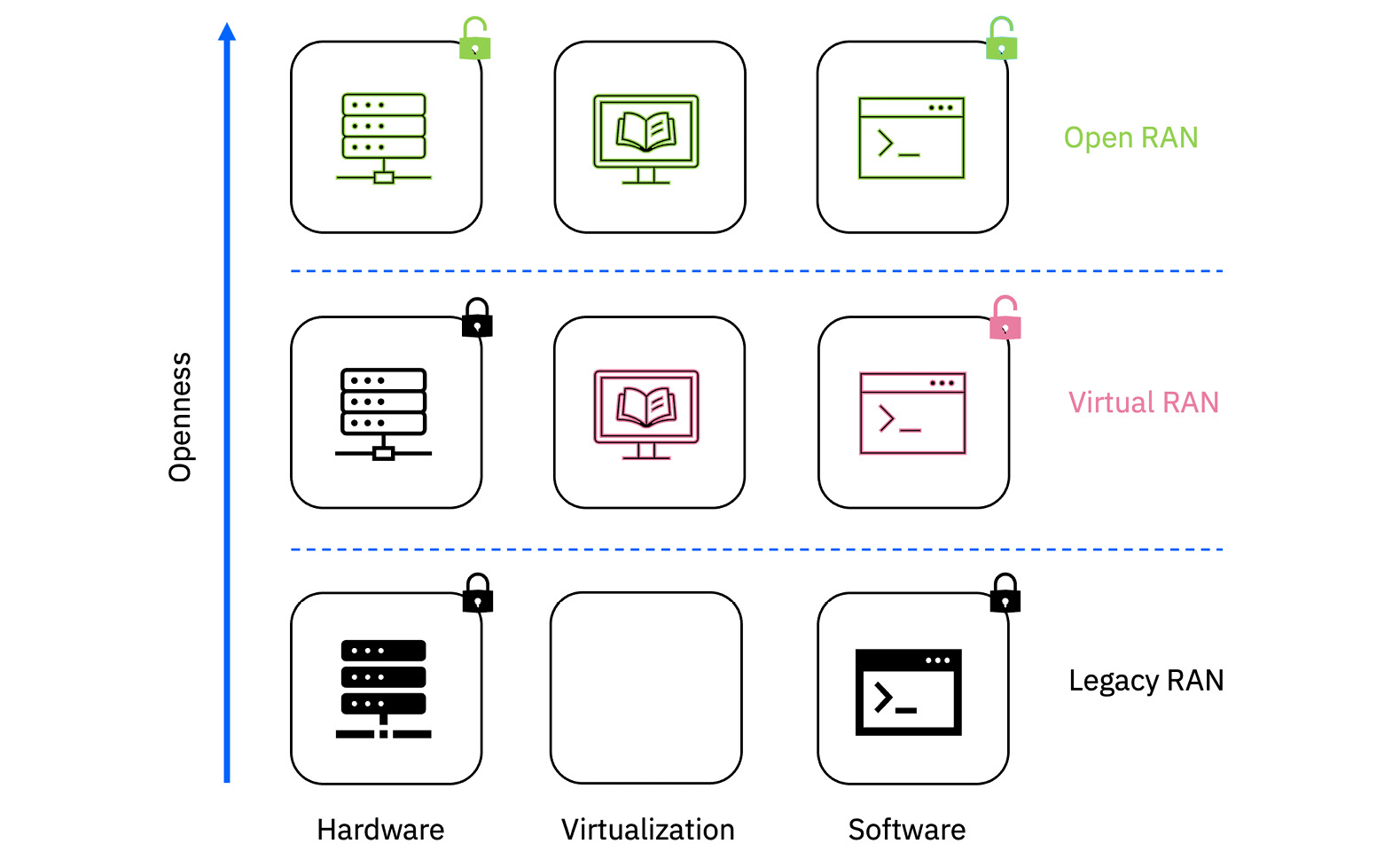


Figure 4.6 – Virtualization in a RAN

In the telcos’ network modernization journey, virtualizing RAN functions is key because as with NFV, it makes it easier to use commodity hardware, and because everything is done through software, it is easier to manage and make enhancements. This is in fact the ongoing goal of 5G network transformation. We assume readers are familiar with 5G mobile technology, which at a high level has three main components – RAN, transport network, and core network. See Figure 4.7. We will leave the vRAN versus Open RAN debate to the networking experts:



Figure 4.7 – 5G network components

Suffice it to say that disaggregation is the goal, coupled with cloud-native and container-based RAN solutions. Disaggregating the hardware from the software gives more freedom to the network operators on how and when to deploy certain technologies and features. It also provides flexibility to their maintenance cycles. It simplifies network operations, lowers costs, and provides greater efficiency when using open architecture.

## CSPs and hyperscalers

We have all read about how CSPs are looking for ways to take advantage of new technologies and grow their revenues beyond just connectivity because a lot of it is software-based. From containerization to virtualization, hyperscalers are looking to expand into new areas such as networking. Hence, we see a new partnership of sorts between these two entities. We say that because in this partnership between telcos and hyperscalers, each is lacking a piece of the solution stack. While telcos own the physical network and location, the hyperscalers provide the cloud platform and virtualization techniques.

But hyperscalers recognized the opportunity and have packaged their cloud platforms and tools into managed, turn-key edge cloud infrastructure. With CSPs owning much of the infrastructure real estate, it makes sense for them to partner with hyperscalers. And the beauty of such a partnership is that it can have many-to-many relationships, as shown in Figure 4.8:

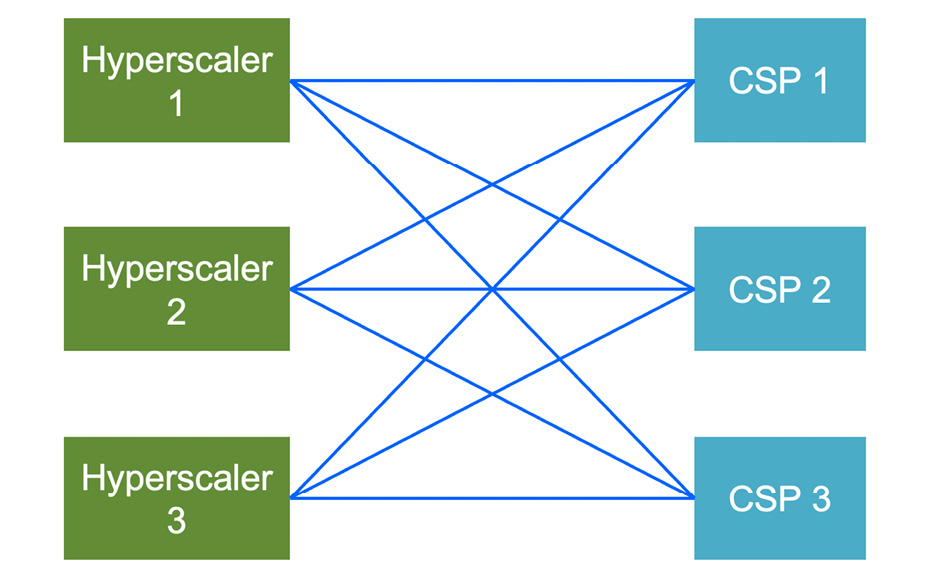


Figure 4.8 – Hyperscaler-to-CSP relationship

While hyperscalers dominate this paradigm, at some point, this partnership could swing in the CSP’s favor as they attempt to capture a share of the hyperscaler’s revenue by positioning compute and storage resources in various locations in their network. Depending on the industry and the strength of the hyperscaler in a particular geography, a CSP can choose to work with a hyperscaler and with their help offer new telco services to customers in that region. Similarly, a hyperscaler could offer its cloud platform and services to any CSP willing to work with it. It boils down to the vertical opportunity and technical breadth of each player in a region.

# Sample architectures

Now, let’s look at some sample architectures where the previously described components, technologies, and principles come together. The major components are the network hardware, the virtualization layer, the applications, and the enterprise cloud. Figure 4.9 is not meant to muddy the waters more but to provide yet another perspective on the various edge realms:

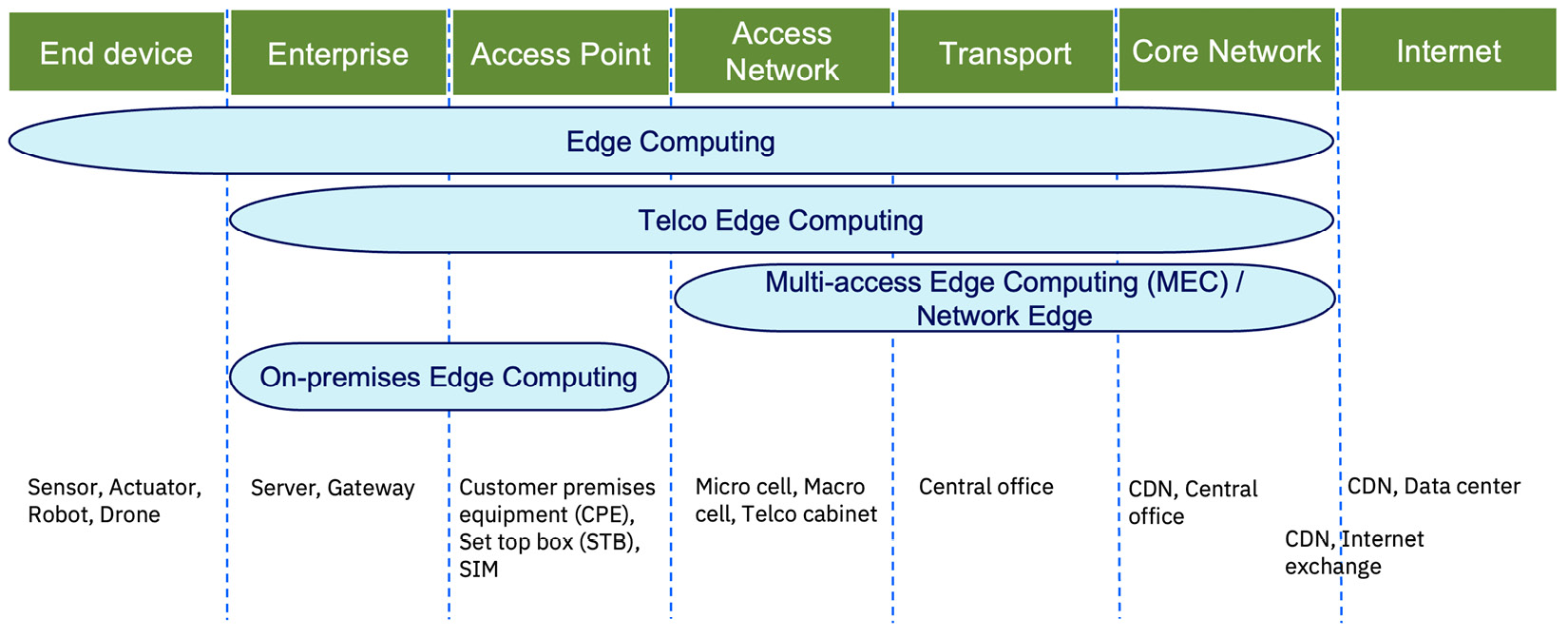


Figure 4.9 – Various edge realms

Note that an **access point** (**AP**) is a term used to describe a network device that allows wireless devices to connect to wired networks. Transport is defined as the network infrastructure providing connectivity to various customer services. It typically connects the RAN and the core network. In the case of 5G, it provides the network slicing function.

*NOTE*

*In 5G RAN architecture, it is possible to split the two functional units: the***distributed unit***(***DU***), responsible for real-time functions, and the***centralized unit***(***CU***), responsible for non-real-time functions.*

Source: [https://www.rcrwireless.com/20200708/fundamentals/open-ran-101-ru-du-cu-reader-forum](https://www.rcrwireless.com/20200708/fundamentals/open-ran-101-ru-du-cu-reader-forum%20)

## Manufacturing scenario

Industrial manufacturing still relies heavily on **programmable logic controllers** (**PLCs**) and **human-machine interfaces** (**HMIs**). A PLC is a ruggedized industrial computer that is adapted and programmed for the control of manufacturing processes in assembly lines, industrial robots, and other machines. These systems and devices are the workhorses of industrial manufacturing and work really well. However, they depend on proprietary operating systems and are not only expensive to acquire but cost a lot to maintain and upgrade. We should mention that there is now open source PLC software such as PLC4X from Apache Foundation (<https://plc4x.apache.org/>), 4diac from Eclipse (<https://eclipse.dev/4diac/>), and others.

These days, robot controllers that are software-controlled have taken their place and have the ability to control many industrial robots. If you visit a car or an airplane manufacturing company, you will notice many such robots working seamlessly churning out products. There could be many industrial robots in an assembly line in a manufacturing unit. With many such assembly lines in the company, the density of such edge-type devices increases. Programming, management, and control of these devices is critical.

Keeping things functioning smoothly in such an environment requires deploying workloads onto these devices consistently and in a timely manner. One option from the network edge perspective is to employ RAN functional splitting, where the **radio unit** (**RU**) is placed on-premises, and the other RAN components of DU, the **centralized unit** (**CU**), and the network core are deployed in the public cloud:

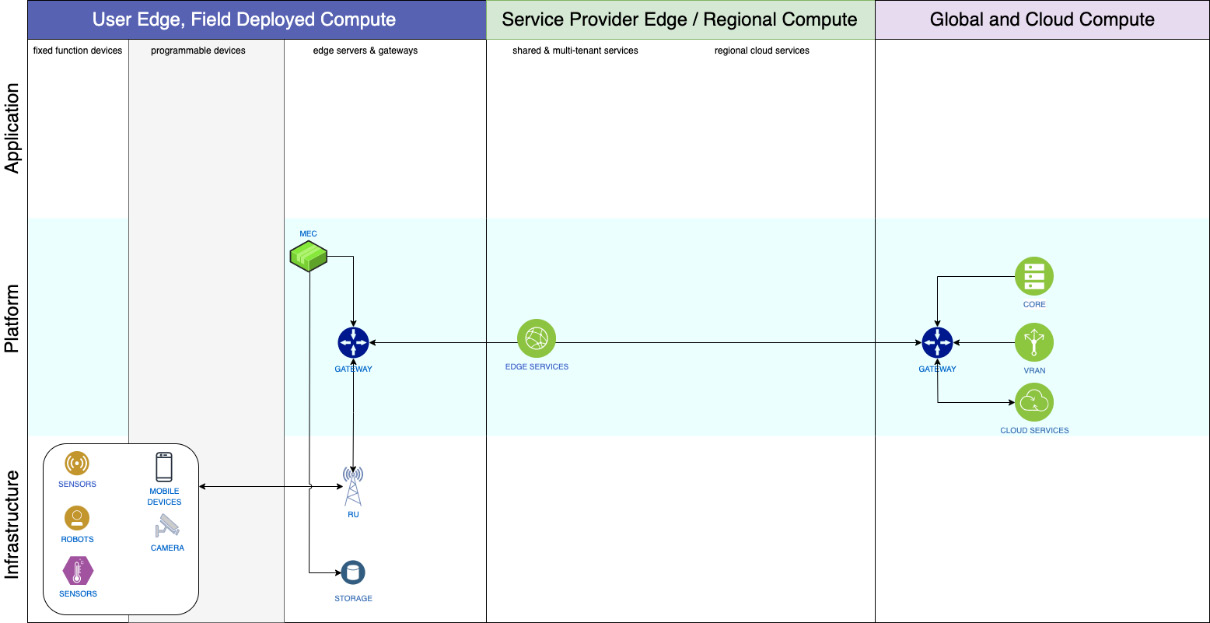


Figure 4.10 – High-level architecture showing MEC and RU on-premises

Additionally, a MEC unit is placed on-premises where the edge application is deployed. A database could also be deployed on-premises to store the data generated by the devices for analysis, as shown in Figure 4.10.

In this solution, the focus is on coverage and device density, rather than low latency and high bandwidth. This architecture is favored by enterprises that prefer a low RAN footprint on-premises.

## Healthcare scenario

Hospitals typically operate in an environment where every second and minute is precious, hence the need for quick decisions, which translates into low-latency networks. We all remember getting X-rays, and somebody would physically deliver them to a doctor or a specialist for diagnosis. Whether they are X-rays, CT scans, or ultrasounds, these days images are taken and transmitted as electronic files. Often, these large image files are sent to another location in the hospital for diagnosis, and that requires greater bandwidth in the network.

Deviating a bit from the previous solution, one network option is to place all three RAN components – the RU, DU, and the CU – on-premises and keep the network core in the public cloud.

In addition, the CSP could choose to dedicate a 5G network slice for the hospital. The solution architect also has the option to place an MEC unit in or near the hospital premises, which would not only help with faster data access but also maintain the data’s security posture. See Figure 4.11:

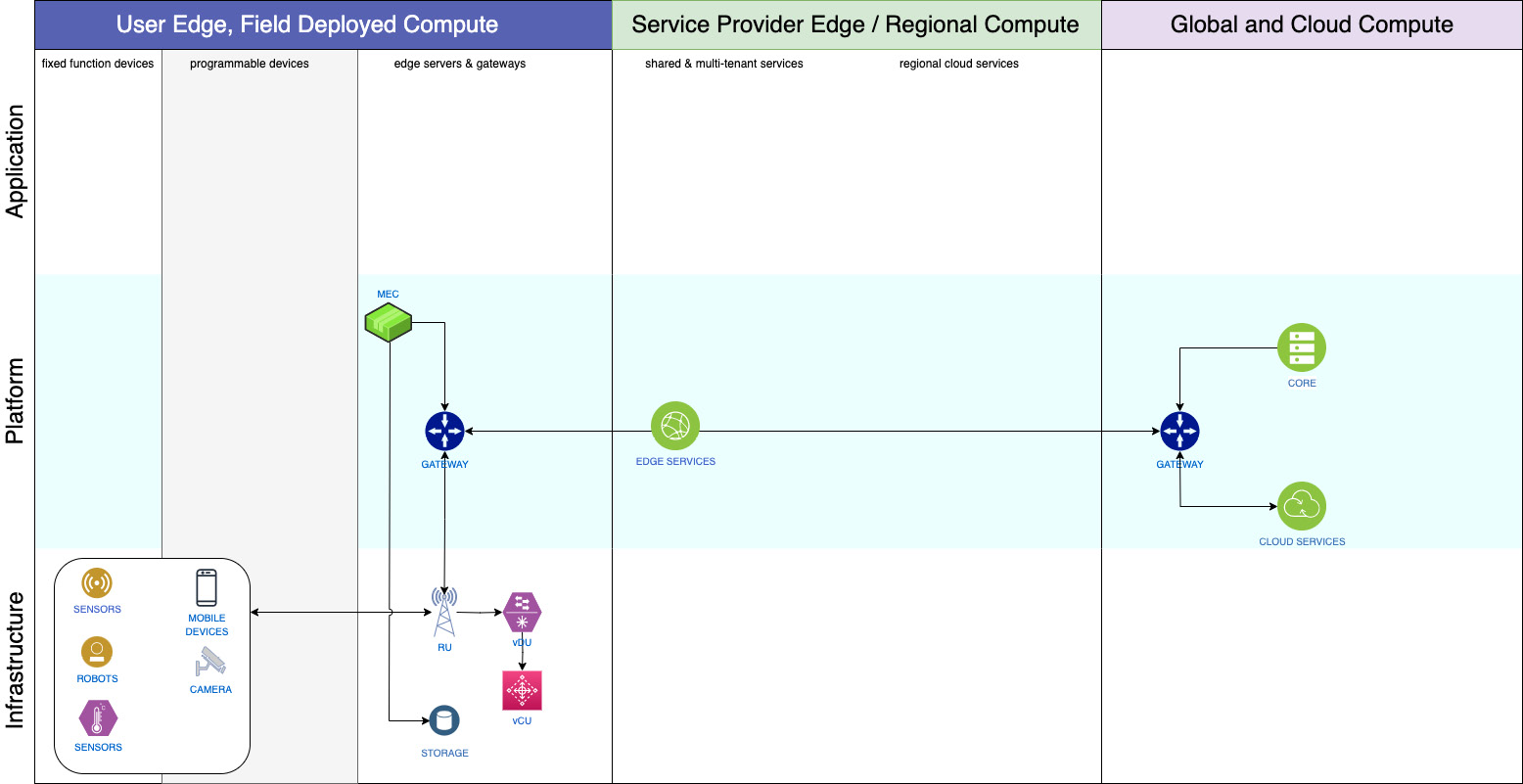


Figure 4.11 – High-level architecture showing MEC and RAN on-premises

It makes sense to deploy the AI/ML application, which is required to help with the identification and analysis of those medical images, on that MEC platform in the hospital. After the image is analyzed, it is often stored within the hospital premises or in a nearby secure repository to comply with regulatory requirements such as the **Health Insurance Portability and Accountability Act** (**HIPAA**) in the US. The solution architect has to think not only about network design but also about other aspects such as storage and data sovereignty.

## Campus network scenario

In this scenario, the three RAN components – the RU, DU, and CU, and maybe even the network core – are all deployed on-premises. The advantage of such an architecture is that the customer has full control of the platform and it is secure. See Figure 4.12:

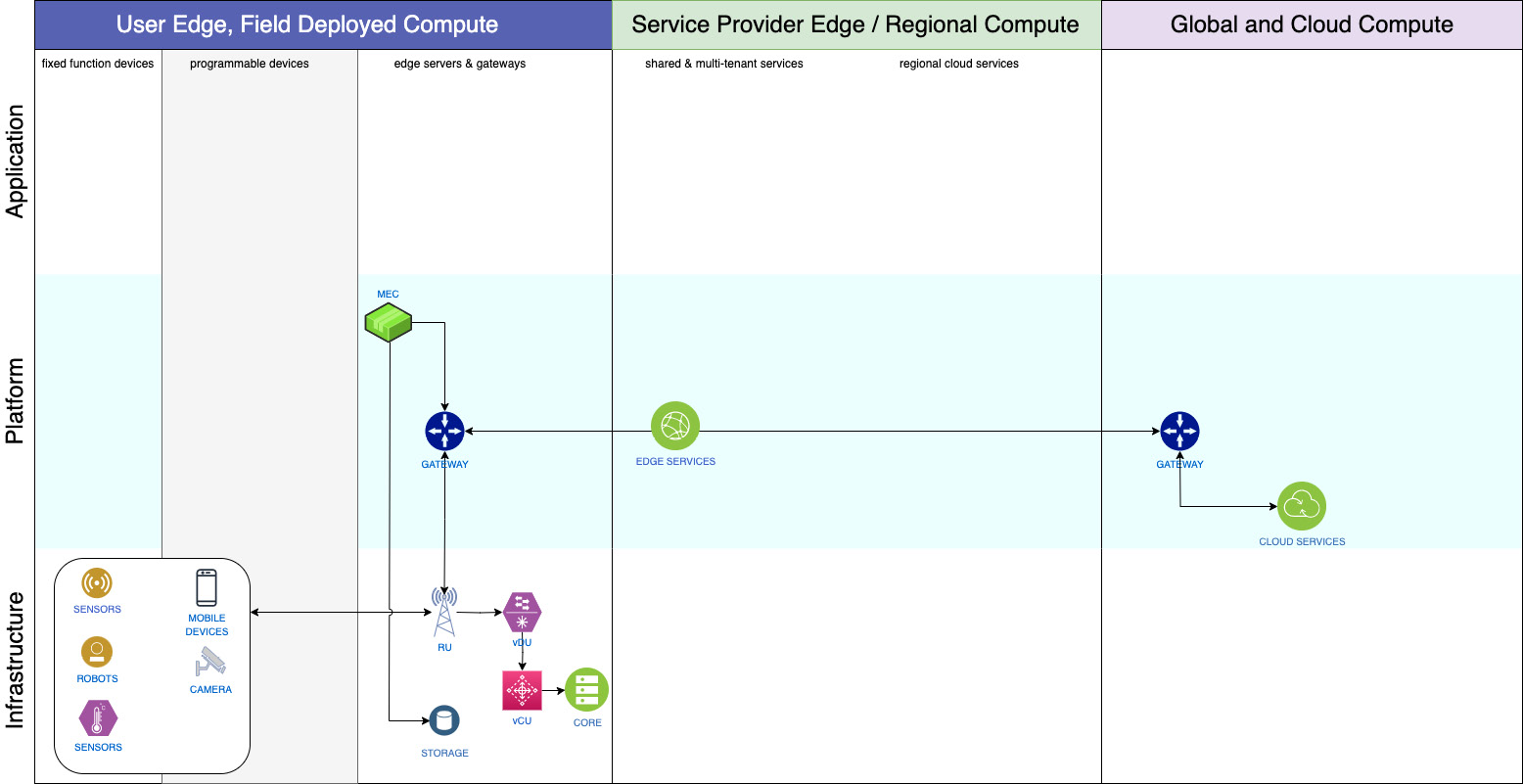


Figure 4.12 – High-level architecture showing MEC, RAN, and network core on-premises

A solution architect can envision such a setup in a stadium where the goal is to create a private campus network for the sole purpose of providing an immersive experience to the spectators. This is done by delivering an AR service in the stadium for the spectators and providing additional data and statistics during the game.

The primary purpose is to distribute video content taken from multiple angles during a live sporting event via multiple 5G-enabled cameras. The captured video is sent through a dedicated 5G network slice to a MEC site for AR processing. Player names, profiles, and statistics are added to the video stream as an AR overlay, and the enriched video is transmitted back to spectators in the stadium in real time.

Such an environment is facilitated by possibly setting up a private 5G network on-premises to support these common edge requirements of coverage at scale, low latency, high bandwidth, and very high device density.

# Summary

In this chapter, we discussed the different aspects of network edge architecture. It showed the importance of CSPs needing to adopt cloud and virtualization to offer new services that drive growth and improve customer experience. CSPs are attempting to become DSPs.

We talked about the virtualization of NFs and how it is helping telcos take advantage of cloud-native technologies. You also learned about SDN. We saw the prevalence of network management systems and how they help network administrators.

Finally, we closed the chapter with a description of MEC, which has gained popularity with the advent of 5G technology. Three common edge use cases were described that showed variations in the location of the network components.

In the next chapter, we will look at end-to-end architectures.

# 5

# End-to-End Edge Architecture

Often, we come across solution architectures that encompass every aspect of edge computing, from the enterprise all the way to the far-edge device. Chapters 3 and 4 laid the groundwork to discuss such end-to-end edge architectures, which include compute nodes of different sizes and capabilities, analytics done at various points, storage options of all the generated data, and, finally, the network. A solution architect has to take into account those macro components plus factor in types of devices and their form factors, the applications that get deployed on the edge devices, management and monitoring options, and security aspects.

You come across such end-to-end edge architectures in many industries, from automotive manufacturing to healthcare to logistics to large retail. What makes such architectures interesting is the kind of data that is generated by the edge devices and the scale of the entire solution.

In this chapter, we will explore some specific industry edge architectures and learn how they integrate existing networks with 5G technology. We will take a look at the following topics:

* IT and OT convergence
* AI and Edge Computing
* Industrial Edge Scenario
* Manufacturing scenario
* Retail edge scenario
* Retail store scenario
* Edge reference architecture
* Edge and distributed cloud computing

# IT and OT convergence

IT and OT have always been at odds, and with the push to digital transformation, the roles have overlapped even more, causing more friction. OT is traditionally focused on controlling and monitoring processes in the enterprise, such as in the warehouse or factory shop floor, where they deal with a heterogeneous environment consisting of different types of equipment and devices. IT, on the other hand, deals with a more homogeneous environment of systems and the management of the same.

It is often said IT is about the business, while OT is the business. Security, organizational siloes, and cultural barriers notwithstanding, IT and OT need to work together for the good of the company, especially in this era of digital transformation. For a business to drive efficiency, make productivity gains, and eventually increase profitability, there needs to be IT and OT convergence. In an edge computing paradigm, IT and OT should be the yin and yang of a business. See Figure 5.1:

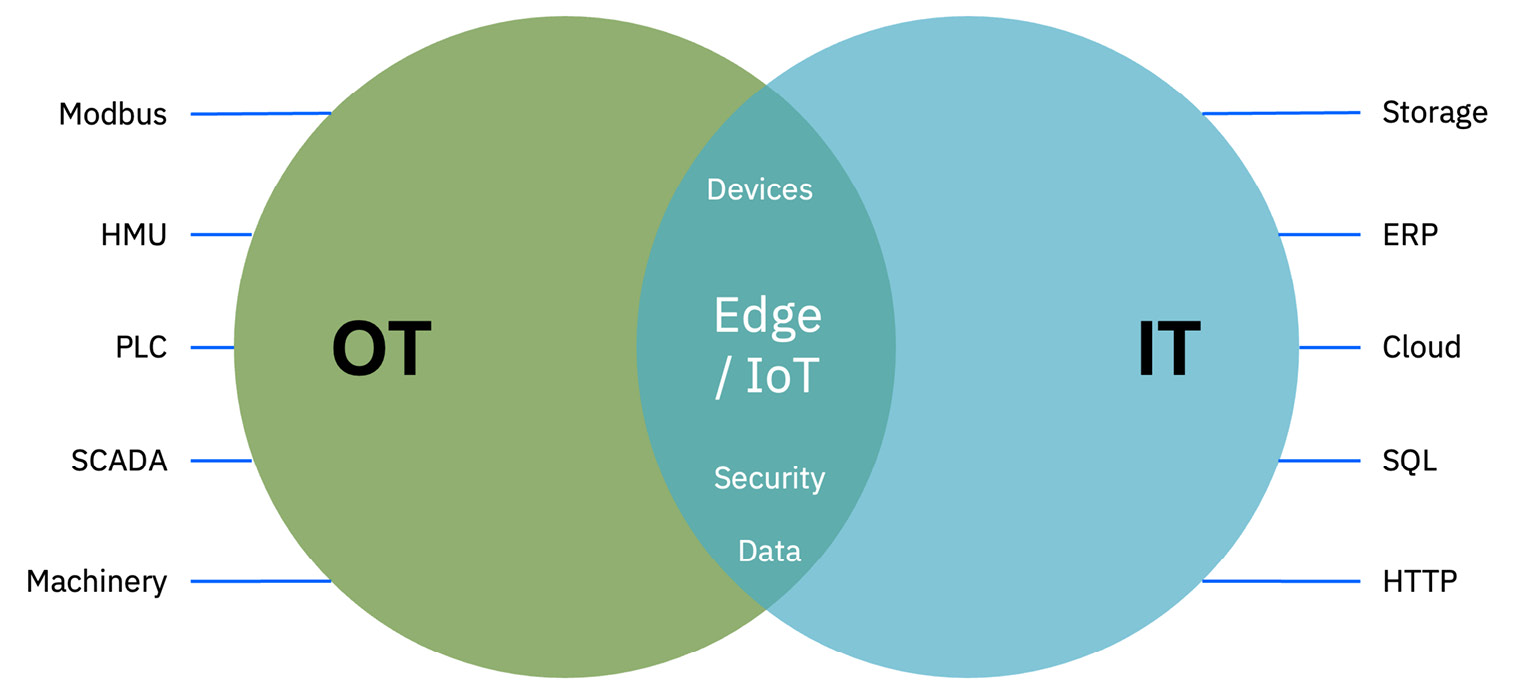


Figure 5.1 – IT and OT convergence facilitated by edge computing

OT brings its vast experience with **supervisory control and data acquisition** (**SCADA**) systems, **programmable logic controllers** (**PLCs**), **hydro-mechanical units** (**HMUs**), and **manufacturing execution systems** (**MES**). IT understands software and IT-related platforms such as the cloud, databases, **enterprise resource planning** (**ERP**) tools, and so on. But there are some common components such as edge devices, data, and, above all, security that both teams have to deal with, and that’s where we see them needing to work together.

As mentioned, edge and IoT are bringing IT and OT closer. In fact, we can now make the case that there is a lot more convergence and cooperation because applications such as video analytics, health monitoring, logistics tracking, and digital farming deployed on a global network of devices and systems have broken down the perceived walls. From managing these devices and sensors to deploying applications on them, everyone responsible for maintaining security in these environments has forced this convergence of IT and OT. In this digital era, converged IT-OT infrastructure will need to connect with various production-floor machines or warehouse systems or to devices that extend beyond the walls of the traditional **data center** (**DC**) and deliver analytics, monitoring statistics, and other client-side services to customers in real time. Edge computing provides a common or unified platform for teams to both work in and bring their expertise to solve business problems and demonstrate value within the enterprise and to their external clients.

# AI and edge computing

This is yet another type of convergence, that of AI and edge computing. Certain applications, such as autonomous vehicles on the road, healthcare monitoring, and industrial robots in an assembly line, require immediate responses because they do real-time analysis and are faced with making quick decisions. This is where deploying AI algorithms at the edge comes in because it brings intelligent decision-making to the edge and reduces the need to transfer data to central servers.

We talked about deploying AI models to the edge, but the training and retraining of those models are done on the enterprise edge or the regional edge and typically not done at the far edge. Even the deployment and management of these AI models across a large number of edge devices has its own challenges of scale and consistency. Not all devices are created equal, and neither are the AI models. Solution architects must be cognizant of the form factor of the edge devices, the constraints they have, and the size and requirements of the AI models.

The convergence of AI and edge computing is undoubtedly a significant step forward in IoT and real-time decision-making. However, several challenges must be addressed to realize its full potential. The deployment and management of AI models across potentially thousands of edge devices present scalability, consistency, and maintainability challenges. Figure 5.2 shows the flow of data from the far-edge devices, after inferencing, sent to storage in the layers to the right, and the AI model building and training being done where there is more compute and storage then deployed onto the edge devices:

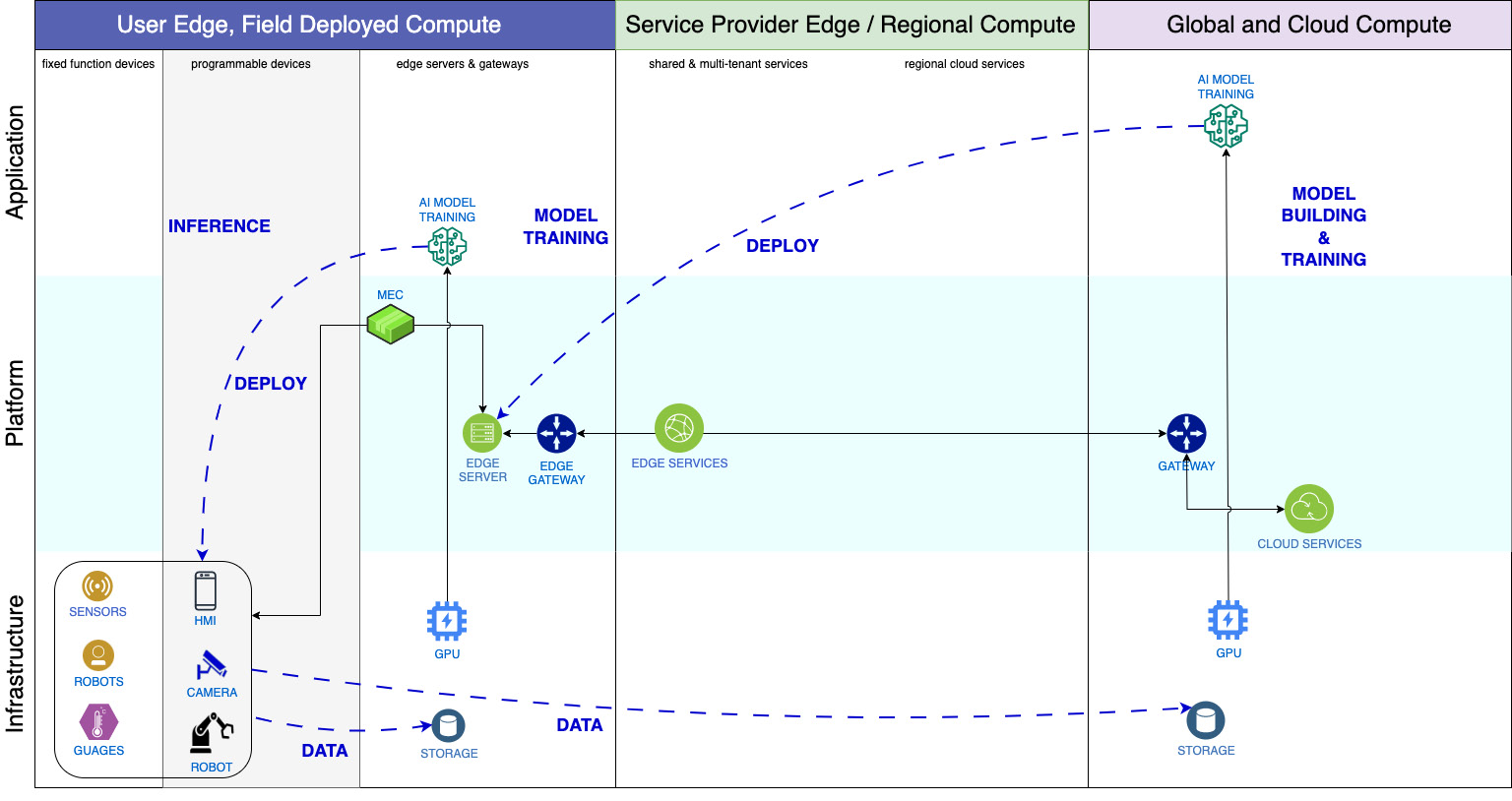


Figure 5.2 – AI model life cycle in edge computing

Inferencing at the edge in real-time or edge analytics in near real time is possible thanks to AI at the edge – specifically, AI applications deployed on the edge devices. The following chapter, [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110), talks about dealing with all the data that is generated or collected at the edge. Suffice it to say that a lot of data is required to train AI models, which means all that data has to be stored somewhere, and, as shown in Figure 5.2, some of it can be stored in the enterprise edge or it can all be sent for storage in the enterprise cloud.

We would be amiss to not mention the hardware advancements that have played a key role in bringing AI to the edge. Smaller yet more powerful processors are now available that are capable of running complex AI models at a much lower cost. Simultaneously, ML algorithms are more optimized now, and the processing power required for modeling and training fits in the constrained edge devices. Also, model training and pruning techniques are more efficient now, which makes it possible to run these AI algorithms on edge devices.

Solution architects must be mindful of the size of ML models that can be deployed on the devices. Not only do the models have to be small in size, but they have to be so designed that they perform a limited set of AI tasks efficiently. [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110) talks about using data to build ML models, and [Chapter 8](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_08.xhtml#_idTextAnchor152) discusses the deployment operations of these AI models at the edge.

# Industrial edge scenario

This scenario was briefly discussed in [Chapter 4](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_04.xhtml#_idTextAnchor073). Many enterprises continue to use 4G or Wi-Fi connectivity within their premises, but many devices rely on wired connectivity. For example, industrial systems continue to use PLCs because they are ruggedized, reliable, and cost-effective. PLCs and **human-machine interfaces** (**HMIs**) are being combined with edge computing technologies to provide better control and analytics. Nowadays, edge controllers, as shown in Figure 5.3, are providing PLC functions and more by way of monitoring applications and optimizing energy usage:

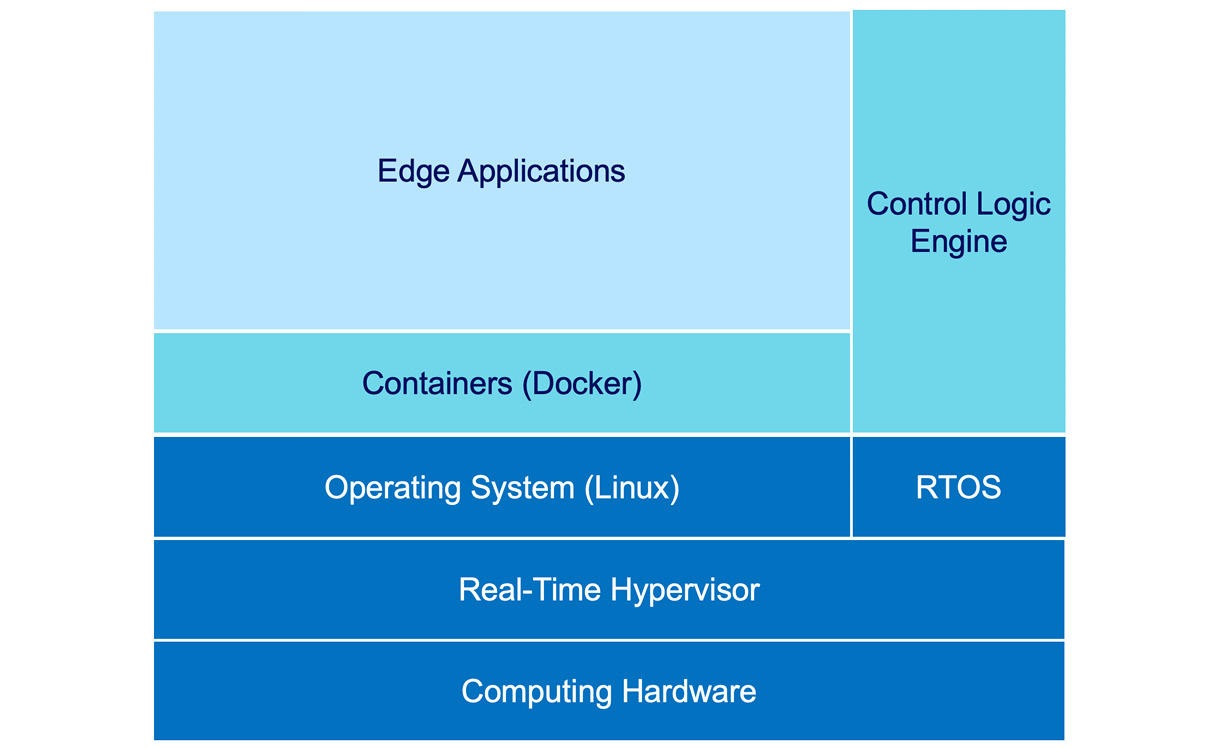


Figure 5.3 – Example edge controller augmenting PLC system

Figure 5.3 shows the components of an edge controller. The traditional control logic engine continues to run on a **real-time operating system** (**RTOS**) because they are very efficient and provide predictable latencies, meaning function calls are guaranteed to return within a specific timeframe. Both RTOS and other OSes, such as Linux, are run on a virtualization layer on purpose-built hardware. Containerized edge applications make up the new software technology that applies to industrial systems to help enterprises offer new services. These controllers allow companies to deploy newer applications and get better insights on the shop floor while continuing to operate on existing underlying automation systems.

Edge controllers work in conjunction with existing PLC systems to optimize assembly-line performance and energy usage and monitor machinery health. From an edge computing perspective, the edge controller is used to gather data and provide additional analytics and visualization, even video, as and when needed.

When it comes to using edge controllers, the subtlety of OT and IT convergence should not be lost on the solution architect here. OT teams, because of their knowledge and experience, will want to continue to have control and decision-making over the hardware used in these edge controllers while they defer software-level decisions to IT.

Source: [https://www.controleng.com/articles/edge-control-evolution/](https://www.controleng.com/articles/edge-control-evolution/%20)

# Manufacturing scenario

In an effort to make the manufacturing process more efficient, we see assembly-line robots, warehouse robots, acoustic calibrators, and industrial cameras inspecting flaws on the manufacturing line becoming more commonplace in the realm of industrial automation. From an edge computing perspective, these are all edge devices that run applications specific to the tasks they perform. We will look at a scenario that uses AI to detect anomalies or flaws in robotic welding, with the ultimate goal of preventing assembly-line stoppage. Typically, such quality checks are done manually by the **quality control** (**QC**) team, which adds time delays and could be costly.

The four groups of components in this scenario are:

* The devices, including robotic welding components and ruggedized cameras on the shop floor
* The edge-related platform components in the enterprise
* 5G networking components and software
* Services in the enterprise cloud

Enterprises have different options when it comes to networking. They can work with **communications service providers** (**CSPs**), partner with telcos, or get all the services from a hyperscaler. Before going any further into the architecture, we want to point out that there are two deployment options when it comes to rolling out 5G – public and private. Table 5.1 lists the characteristics of public 5G and private 5G without getting into the minutia:

|  |  |  |
| --- | --- | --- |
| **Category** | **Public 5G** | **Private 5G** |
| Consumer of network | Public 5G wireless networks offer the same level of service and security to all customers, businesses, and consumers that are on the network, other than slicing. We will refer to the public network as being owned by a **mobile network operator** (**MNO**), also referred to as a CSP. In the case of the public 5G network, the 5G spectrum is owned by the MNO/CSP, and the management of the network and service is also their responsibility. | Private 5G wireless networks are operated privately by a group, a company, or government agencies. They create a more secure wireless experience and are protected at busy times. Some examples where private 5G networks are deployed are manufacturing facilities, retail locations, shipping ports, airports, hospitals, defense bases, and so on that require secure, predictable performance. |
| Security and isolation | The security risk is higher and comes from the public sharing the network. Also, when the network is busy, it could impact all users equally. | These networks are the most secure because there is connectivity to the public network and the enterprise has full control of it. |
| Size of the network | These networks are typically built to serve millions to hundreds of millions of subscribers, hence the scale is many times larger than private networks. The design and deployment of these networks are much different than with private 5G networks. | The networks are typically small and deployed in one or more locations of an enterprise, and can be owned and operated by the enterprise, or can be deployed and managed by **systems integrators** (**SIs**) and **managed SPs** (**MSPs**). |
| Integration | It is a distributed architecture when deploying public 5G networks on a hybrid cloud platform. | Private 5G networks can be isolated from the public 5G network, or some enterprises may integrate with the public network of MNO/CSP for mobility and roaming outside the enterprise location/campus. |

Table 5.1 – Characteristics of public 5G and private 5G

In this particular instance, we have chosen to describe the option where the auto manufacturing company has decided to work with a hyperscaler for all its 5G networking needs. They have decided to set up a private 5G environment on-premises where along with the **centralized unit** (**CU**), the **distributed unit** (**DU**) and **radio unit** (**RU**) are the 5G core. Hence, there are no components in the SP column and no tasks, per se, for the CSP. See Figure 5.4, which depicts the topology of the proposed solution:

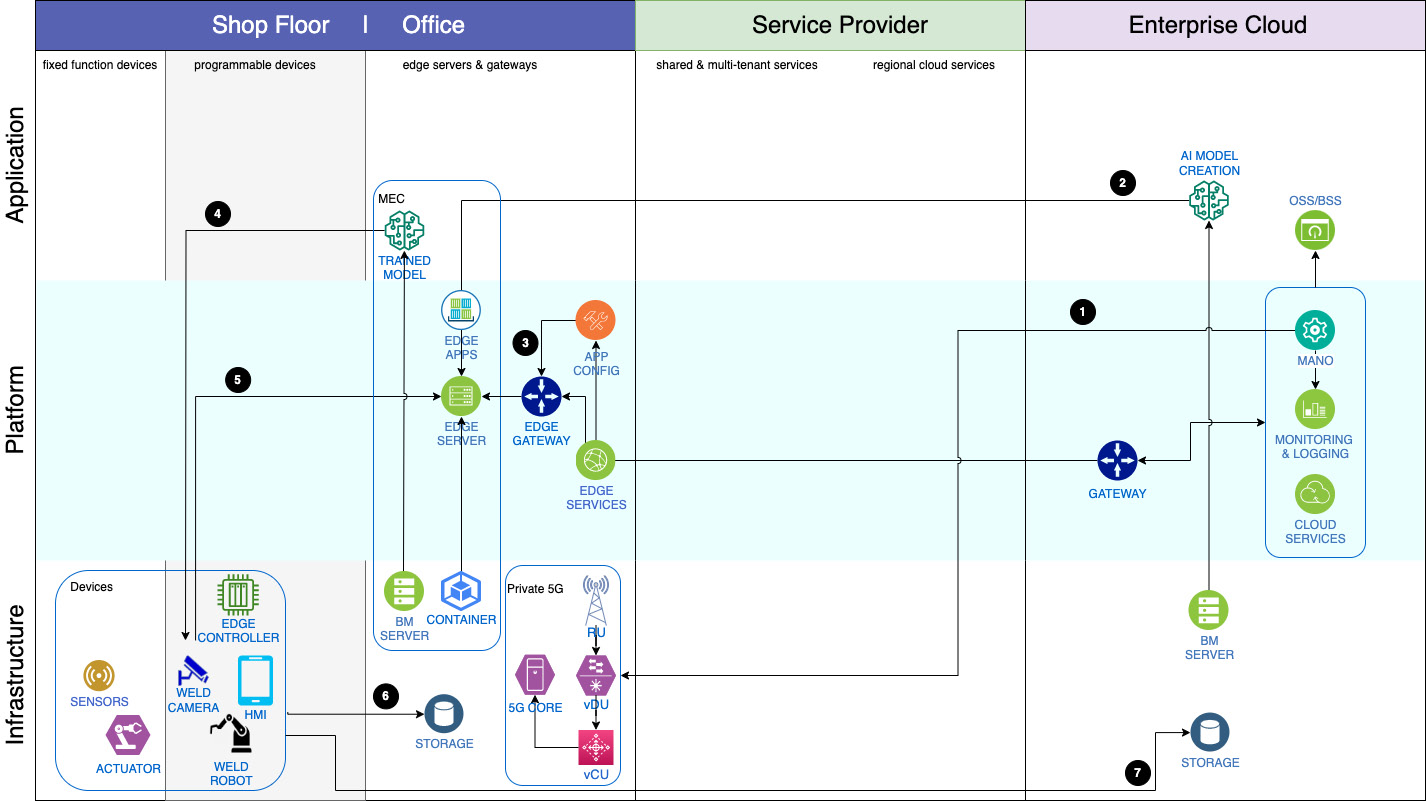


Figure 5.4 – Edge solution architecture in a manufacturing scenario

The architecture flow is explained in Table 5.2. The numbers correspond to those depicted in Figure 5.4.

|  |  |
| --- | --- |
| **No.** | **Description** |
| 1 | The network management layer is multi-tenanted and hosted by the hyperscaler in the enterprise cloud.  **Management and network operations** (**MANO**) plus tasks related to **operations support systems** (**OSS**) and **business support systems** (**BSS**) are all handled by the hyperscaler from the enterprise cloud, which is the “managed from” location.  As mentioned earlier, the private 5G on-premises network provides a secure network for use by all devices in the enterprise. |
| 2 | Initial ML model creation and training require a lot of data and computing power, which is available in the enterprise cloud.  This rather large model is deployed on the enterprise edge. |
| 3 | Depending on the size and specifications of the app, edge-related applications are deployed on the edge server and/or on the programmable devices. |
| 4 | A task-specific model, which in this scenario is trained to detect welding flaws, is deployed on the camera, which is watching the output of the robotic welding process. As mentioned, the camera’s task is to look for flaws in the welding, such as overlap, undercut, or porous weld. |
| 5 | Visual data from step 4 is streamed to the application module controlling the robotic welding machine, which could be a **neural network** (**NN**)-based inference engine. If a flaw is detected, corrective action is immediately sent to the robotic weld machine, avoiding an assembly-line stoppage and thus reducing material wastage and costs. |
| 6 | Data collected from the vision system is stored on-premises for quick analysis of the current line so that tweaks can be made in the app and/or the model. |
| 7 | All data is sent to the enterprise cloud to maintain history and auditability. As described in step 1, this data is used to train/retrain the model before it is redeployed.  Other cloud services, such as monitoring and logging, are also provided by the enterprise cloud. |

Table 5.2 – Flow of the edge architecture for a manufacturing scenario

The scenario just described dealt with finding welding flaws. The same can be applied to other tasks in a manufacturing process, such as painting and inspecting. The applications deployed on edge devices such as cameras or sensors would differ based on the task being performed at that manufacturing station. Next, we look at a scenario in the retail space.

# Retail edge scenario

Large retail stores in rural locations have a different challenge. Network connectivity in such locations isn’t very good, so it behooves them to work with telcos to deploy a private 5G network in those areas, which will allow them to have high-speed connectivity. In such a scenario, there would be 5G connectivity to the store and even inside the store. Such a remote location scenario is also a perfect setup where the disconnected edge solution could come into play when there are connectivity glitches.

There are many edge use cases in the retail industry thanks to all those cameras that we see in stores and even some that are hidden. Some of the obvious ones we get to see in our daily lives when we shop are security-related: theft prevention at **point-of-sale** (**POS**) systems, tracking customer movement, product restocking by way of smart shelves, and so on. Among the new ones are what they call “the store of the future” where you get to customize and try the product before buying or immersive shopping, especially in the realm of beauty and apparel using AR/VR technology.

All this is made possible by a couple of things – data and connectivity. Retailers have realized they can get more – and immediate value – from their own store data by analyzing it in real time and acting on it. Imagine sending a discount coupon to the customer while they pick up two or more of the same item. We know that new technologies such as immersive shopping and real-time visual transactions require high-performance computing and low-latency connectivity. Fast connectivity such as 5G gives retailers the bandwidth they need to offer these new technologies. Imagine being able to hold your mobile phone up and the camera shows the store display, highlighting the different merchandise on sale.

# Retail store scenario

Whether it is a grocery store or retail merchandise store, edge-/IoT-related solutions are everywhere. Temperature sensors on display freezers, cameras on self-checkout registers, and high-definition shelf monitoring camera systems are some of the devices you see in stores. The four groups of components we described in the previous scenario are also applicable here.

We mentioned different networking options available to enterprises. In this scenario, we describe the option where a grocery store chain is working with a telco to bring 5G networking to its stores. Specifically, the grocery chain has opted for a 5G network slice. Before going any further into the architecture, here’s a brief on network slicing.

## Network slicing

Network slicing is the ability to create multiple logical “slices” of a single physical core within the 5G network infrastructure using **software-defined networking** (**SDN**). These slices are optimized for specific use cases, types of services, or sets of users. See Figure 5.5:

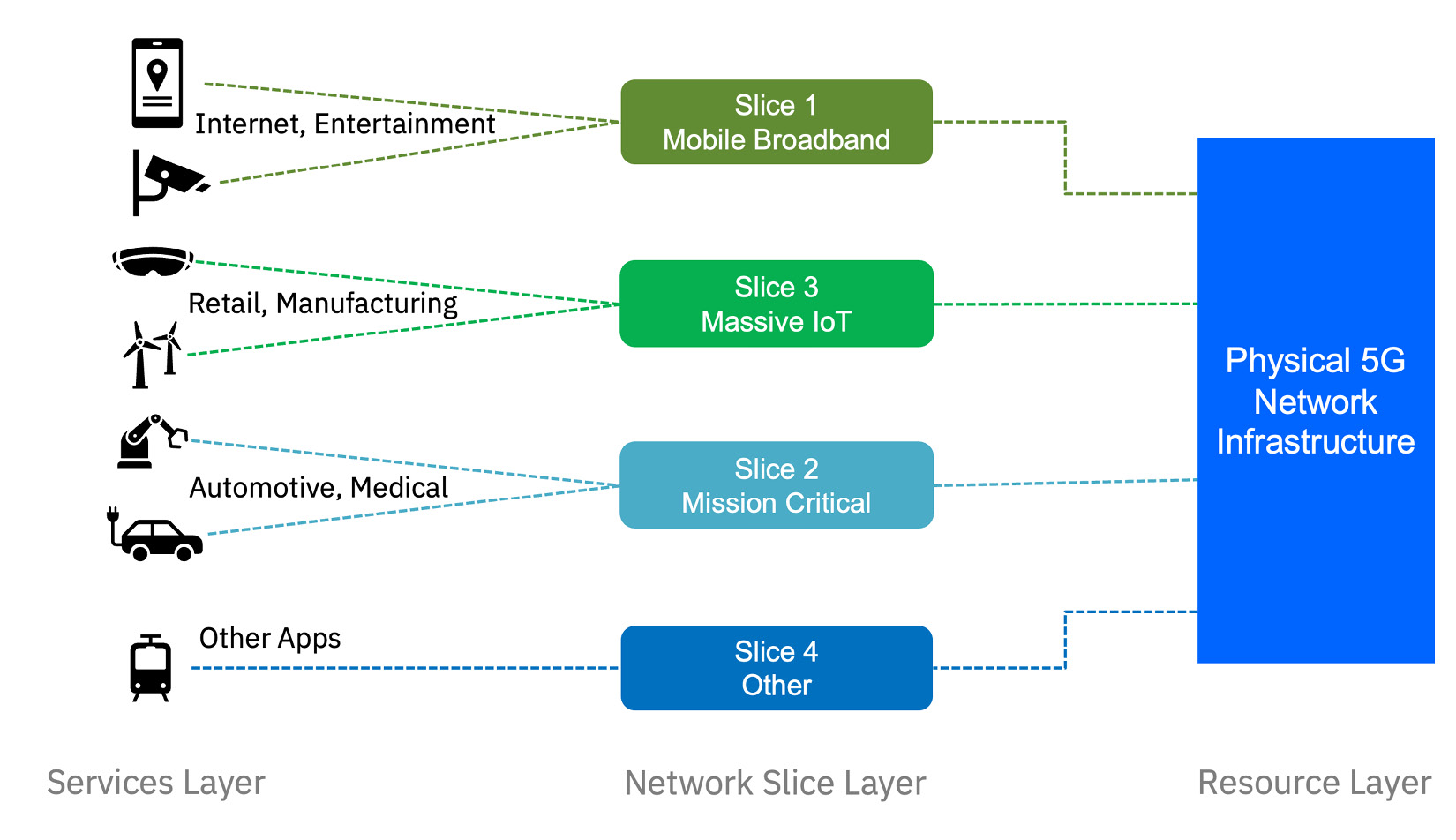


Figure 5.5 – 5G network slicing

SPs also have the option to slice the network in different ways depending on the demands of their customers. Table 5.2 lists the different slicing types:

|  |  |
| --- | --- |
| **Slicing Type** | **Description** |
| Vertical  slicing | This type is industry-focused and designed to serve verticals, such as production optimization, manufacturing, retail, healthcare, and so on. Applications cover public safety, IoT, IPTV, and so on. |
| Horizontal slicing | This type is more focused on characteristics the slice offers that can support a wide variety of use cases. The slices are created based on characteristics such as bandwidth for downloads and uploads, latency sensitivity, capacity/density sensitivity, symmetric traffic types, and so on. |
| Static slicing | These are fixed slices that do not change their specifications and are created to cater to specific use cases and capabilities such as M2M and IoT. |
| Dynamic slicing | Allows SPs to deliver slices in real time and on demand. With this type of network slicing, SPs can quickly provide unique deployments and performance thresholds for individual 5G use cases as required by enterprises or consumers. |

Table 5.2 – Network slicing types

## Example scenario

In our scenario, a telco will be providing a dedicated slice for all grocery stores in the chain and will be responsible for the management and orchestration of the network connectivity to the stores. It is also possible that the grocery chain might use the regional cloud services of the SP to store data from the stores in the region and use that data to train models. See Figure 5.6, which depicts the topology of a solution in a grocery store:

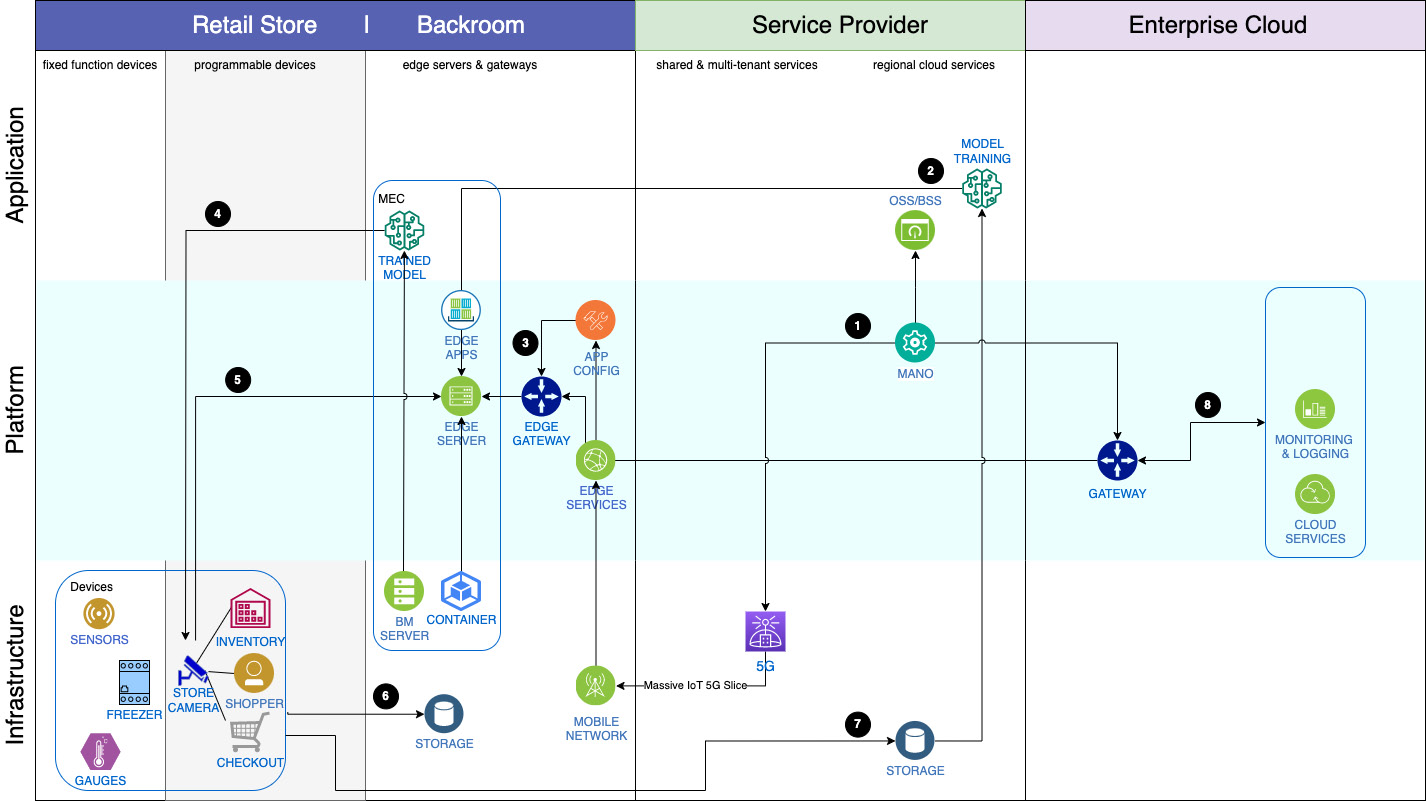


Figure 5.6 – Edge solution architecture in a retail scenario

The architecture flow is explained in Table 5.3. The numbers correspond to those depicted in Figure 5.6:

|  |  |
| --- | --- |
| **No.** | **Description** |
| 1 | The network management layer is multi-tenanted and hosted by the SP in the regional cloud that is closer to the customer.  MANO plus tasks related to OSS and BSS are handled by the SP from the regional cloud, which is the “managed from” location.  A 5G network slice is provided as a secure network for use by all devices in the grocery stores. |
| 2 | ML model creation and training requiring a lot of data and computing power are available in the regional network cloud.  This model is deployed on the enterprise edge. |
| 3 | Depending on the size and specifications of the app, edge-related applications are deployed on the edge server and/or the programmable devices. |
| 4 | Task-specific and device-relevant models, which in this scenario are visual inspection models, are deployed onto the cameras that are mounted in various locations in the store. As mentioned, these could be security-related, for theft prevention at POS systems, tracking customer movement, or even for product restocking. |
| 5 | Visual data from these cameras is streamed to the application modules that can quickly make inferences and take immediate action. All this is to stop merchandise theft, provide a better customer experience, or provide a personalized shopping touch to in-store shoppers. |
| 6 | Data collected from the vision system is stored on-premises for quick analysis of current shopping trends in the store so that the models can be tweaked in real time and made available in the apps. |
| 7 | Most of the data is also sent to the regional cloud for maintaining history and auditability and can then be sent to the enterprise cloud to get a corporate view across all stores. |
| 8 | Other cloud services, such as monitoring and logging, are provided by the enterprise cloud. |

Table 5.3 – Flow of the edge architecture for a retail scenario

We described an edge solution within a grocery store. One can envision replicating this in department stores or home goods stores because the components in play are very similar: inventory shelves, POS systems, backroom, and so on. In the next scenario, we widen the aperture and include some aspects outside the store.

# Edge reference architecture

Thus far, most of the discussion has been about edge and 5G components. In this section, an edge scenario is described from an architectural perspective. Cloud and network components, which are “non-edge” components, such as cloud region, **availability zones** (**AZs**), load balancers, and other cloud services, are discussed. These components are required to support most edge computing solutions no matter the scenario. Figure 5.7 shows the cloud components on the right and most of the edge-related components on the left, which would be on-premises:

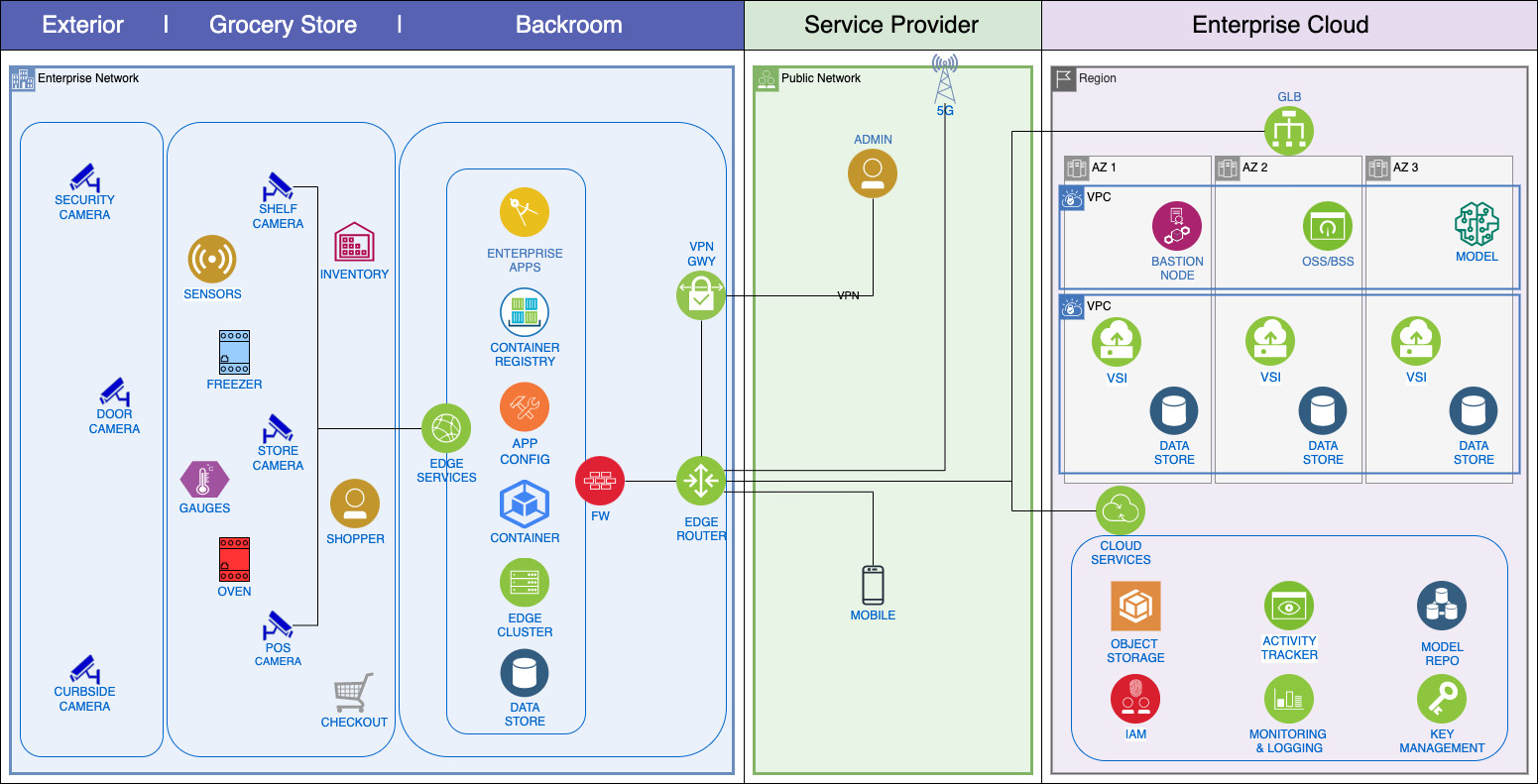


Figure 5.7 – End-to-end edge reference architecture

In this large grocery store scenario, we make an assumption that a CSP will provide the needed 5G network and a hyperscaler will provide all the other IT-related services. Figure 5.7 shows three distinct areas going from right to left – the cloud, the public network (which is the internet), and the enterprise network.

## The cloud

Depending on where the customer (in this example, the grocery store chain) is located, the nearest cloud region is typically chosen to get the best response times when accessing the required cloud services. Every hyperscaler has DCs in multiple regions around the world. Often, these are named US-West or US-East, Europe-North, or Asia-Central.

In our example, if this were a grocery store chain in France, it is quite possible the nearest cloud location could be in Frankfurt in the EU-DE region. Cloud regions have zones often referred to as AZs. Depending on the cloud provider, there are one or more AZs in a region. If available, a production deployment would choose three AZs to support **high availability** (**HA**), which would be our recommendation. A global load balancer distributes workloads across the AZs.

The customer may choose to deploy **bare-metal** (**BM**) servers in the cloud, as they would in their DCs, or opt for a **virtual private cloud** (**VPC**), which is more cost-effective. Software and the data required to run are hosted on **virtual server instances** (**VSIs**) or **virtual private servers** (**VPSs**).

Finally, there is a vast array of cloud services to choose from. The most common ones that are used are security- and observability-related, such as **identity and access management** (**IAM**), key management, logging, monitoring, container registry, and so on. And last but not least, there is storage – and plenty of it – in the cloud from object store to block store to file store. Depending on the application’s needs, the solution architect would determine the type of data store to use.

## The network

Networking was covered in detail in the two earlier scenarios. As already mentioned, in this scenario, we simply assume that a CSP will be providing 5G network connectivity to the business.

A secure **virtual private network** (**VPN**) can be used to connect from the cloud to the enterprise network, or the other option is a high-speed direct connection such as Direct Link or Direct Connect. In either case, the network traffic is encrypted, and it does not traverse the public internet. Access from the internet, if needed, is possible via a secure VPN gateway.

Another route into the store’s network would be via the internet for users of the store’s mobile application. Known as **buy online, pick up in-store** (**BOPIS**), this access method has become very popular now because many users browse and shop for merchandise using the mobile app, place an order online, and then pick up their order in the store or at the curbside.

## The edge

As before, it helps to segregate the backroom from the store and the exterior when it comes to the on-premises enterprise network because the IT hardware components are usually in a secured corner in the backroom and we see far-edge and IoT devices dispersed throughout the store, such as freezers, ovens, POS systems, cameras and more. You can even find cameras outside the store in the parking lot or curbside. We alluded to the fact that many of these high-end cameras have some computing power where visual inferencing applications can run.

The edge devices are managed by the edge cluster/node, which is in the backroom. Other related applications such as ML apps or other analytics apps can also be deployed on these edge nodes. Often, these are containerized applications. Depending on how much data the customer wants stored locally, you will also find one or more data stores, albeit in a smaller form factor. There is a firewall protecting these components, and the only access to this on-premises environment is via an edge router.

As you can see, many components make up an edge solution. And depending on the industry and the use case, more variations in edge devices, compute, network, or storage are possible that solution architects must be mindful of.

# Edge and distributed cloud computing

In this chapter, detailed industrial and retail architectures have been discussed, but all the solutions depict a single location. Enterprises can duplicate these solutions in their other branches, stores, or locations. This is where we see a new trend that merges edge computing and distributed cloud computing. Chapter 2 briefly mentioned distributed cloud as one of the new cloud deployment models. We describe it and a corresponding topology in the next scenario.

## Distributed cloud computing

In distributed computing, public cloud services are made available in different private physical locations, also known as **satellite locations** or **outposts**. These locations are outside the hyperscaler’s facilities and could be on-premises, in another cloud, or even in co-location centers. Distributed cloud computing is not a distributed system.

A **distributed system** is a collection of autonomous computers that work together to create an impression of a single computer. They share resources and operate concurrently but can fail independently.

## The scenario

Figure 5.8 shows a managed-from-the-cloud scenario on the right and two remote locations on the left. Plant location A in the top left is deployed in a hyperscaler center, and plant location B in the bottom left is shown as deployed on-premises. This is important to note because the remote locations can both be on-premises or in the cloud. These cloud locations can be a different hyperscaler than the one managed from the cloud:

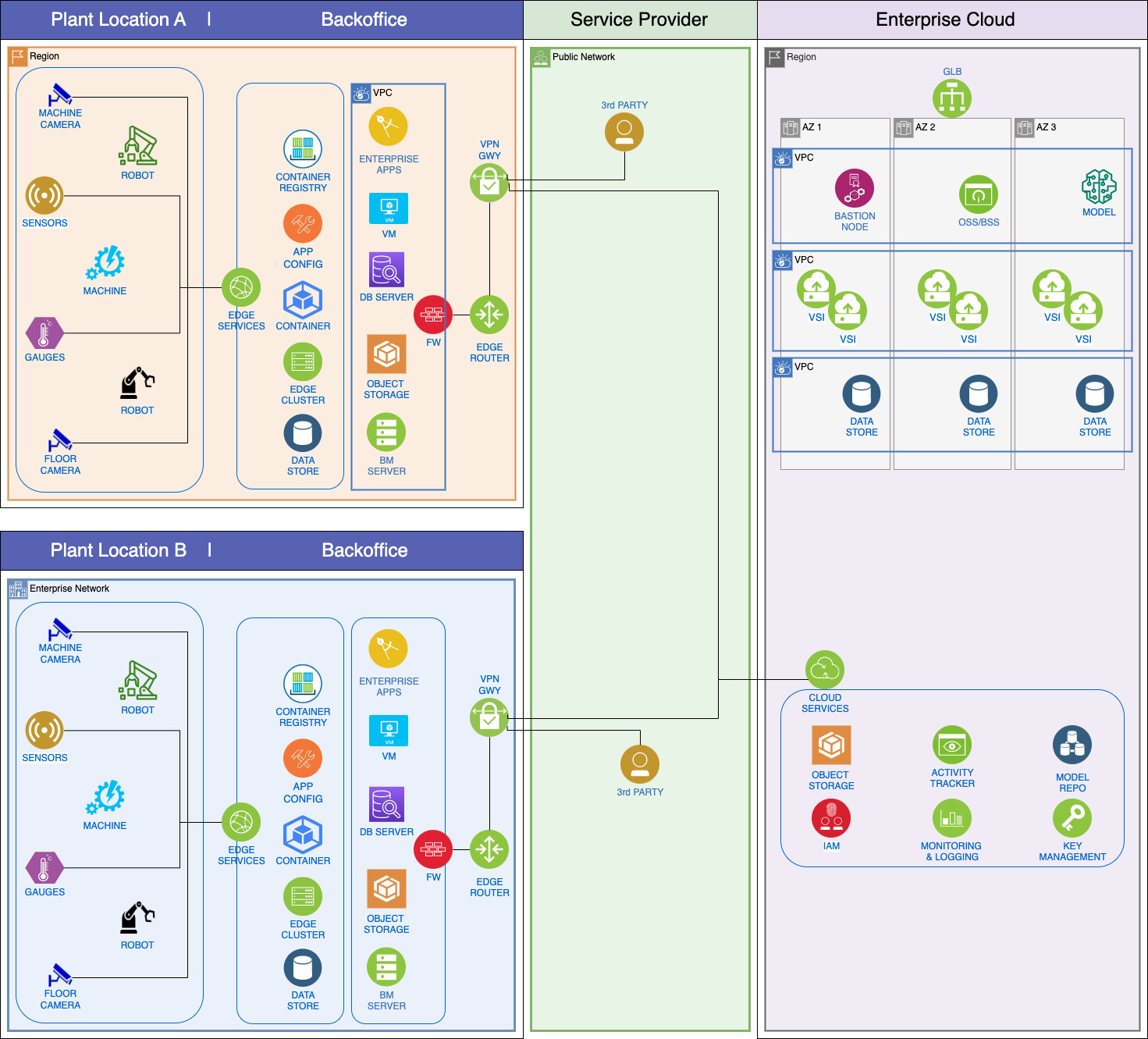


Figure 5.8 – Distributed cloud and edge architecture

In the distributed paradigm, the same cloud services can be extended out to the edge, meaning the same services are available to both plant location A and plant location B. Imagine an auto manufacturer having multiple locations where different parts are manufactured. They all must adhere to certain specifications while producing different parts that must be assembled together to produce a vehicle. In such a scenario, having access to the same cloud services that control the workflow in the various sub-assemblies would not only maintain consistency across production lines but also provide a single pane of management for the corporate manufacturing department. Plants also have the option to run enterprise applications specific to each location’s requirements.

Now, combine that with edge computing practice, wherein each remote location hosts the edge node or clusters from which edge devices can be managed and edge applications can be deployed onto the devices. These devices, as shown in Figure 5.8, can be fixed or mobile robots, assembly-line cameras, and other industrial automation devices. We see this trend continuing to evolve as industries find broader use cases.

# Summary

The chapter started out with a discussion of IT and OT convergence, which is beneficial to any business in this digital age. Then, we looked at the biggest driver of edge computing – namely, AI; more specifically, AI applications and models that can now be deployed on edge devices. Moving intelligent inferencing and analysis closer to where data is generated provides businesses with not only speed and efficiency but also addresses security and privacy concerns.

In the latter part of the chapter, two end-to-end edge scenarios were described – a manufacturing scenario and a retail scenario. Such scenarios are applicable in almost every industry, but each solution has its own nuances that solution architects should know. For example, if it is a regulated industry scenario, one must worry about data compliance requirements and data security. In a delivery scenario, whether it is delivery by land, air, or sea, one must know and account for disconnected operations. Lastly, we introduced distributed cloud computing and showed how it complements edge computing.

It is imperative that solution architects address data security in every facet of these edge solutions, be it data at rest, data in transit, or data in use. That topic is covered in detail in [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110).