**Part 1:Overview of Edge Computing as a Problem Space**

To build a shared understanding, we lay the groundwork and survey the territory in the first two chapters. Since the concepts and terminology in edge computing can be overloading, it’s important to explain exactly what is meant when we elucidate our concepts, problems, and solutions. The first two chapters in the book aim to provide clarity and a common foundation that will be built on in Parts 2 and 3. This part begins with how to think and talk about edge computing grounded in context. It then delves into the various components, describes their purposes, and shows how they relate to others and where they best fit.

This part has the following chapters:

* [*Chapter 1*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013)*, Our View of Edge Computing*
* [*Chapter 2*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_02.xhtml#_idTextAnchor036)*, Edge Architectural Components*

# 1

# Our View of Edge Computing

One of the first challenges when discussing edge computing between IT professionals is establishing a shared vernacular. In the authors’ experience, professionals in this space differ in how they describe the goals, methods, available tools, and deployment targets/operating environments. We’ve also found that, due to experiences and even generational distinctions, some fundamental assumptions may be at play. By agreeing on a definition of terms at the outset, we avoid misunderstandings and talking past each other.

In this chapter, we will start by describing various edge computing scenarios from an infrastructure point of view, moving from cloud to far edge based on our experiences, research, and available de facto standards. Along the way, we compare and contrast different points of view that will affect architectural choices you can make, such as edge computing versus distributed computing and the network edge versus the enterprise edge.

We will rely on conventions covered in the Suggested pre-reading material section, such as State of the Edge annual reports and LF Edge whitepapers. By the end of the chapter, you should have the shared vocabulary and a wider perspective needed to engage in fruitful conversations on edge computing with software architects and other IT professionals.

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

* Speaking like an edge native
* Which edge? Categorizing edges
* Your computer or mine? Tactics for service deployment
* Cloud-out versus edge-in
* Introducing archetype patterns

# Suggested pre-reading material

* State of the Edge Report 2023 (The Linux Foundation)

(<https://stateoftheedge.com/reports/state-of-the-edge-report-2023/>)

* From DevOps to EdgeOps: A Vision for Edge Computing (Eclipse Foundation) (<https://outreach.eclipse.foundation/edge-computing-edgeops-white-paper>)
* Sharpening the Edge – Part 1 (LF Edge) (<https://lfedge.org/wp-content/uploads/sites/24/2023/12/LFEdge_Akraino_Whitepaper2_v1_PrePrint.pdf>)
* Sharpening the Edge – Part 2 (LF Edge) (<https://lfedge.org/wp-content/uploads/sites/24/2023/12/LFEdgeTaxonomyWhitepaper_062222.pdf>)
* Software-as-a-Service (SaaS) overview (<https://www.salesforce.com/saas/>)
* Defining software deployment as days (<https://dzone.com/articles/defining-day-2-operations>)

# Speaking like an edge native

In this section, you will learn to articulate fundamental differences between the edge and the cloud. This impacts the available infrastructure, platforms, services, and application deployments. Additionally, you will be able to explain concisely what the edge is and what the field of edge computing does to both **line-of-business** (**LOB**) executives and other non-IT professionals. This includes being able to explain the value it can provide, as well as why this field is emerging at this point in time.

## What is the edge?

The edge in edge computing is commonly used to describe the location where computing takes place. The name itself is meant to evoke a spot in a corner or by the wayside, and not in a central area, and actually refers to the very end of a communications network (the edge of the internet; see Figure 1.1). Thus, edge computing happens outside a cloud computing facility, and many times outside the four walls of a traditional **data center** (**DC**).

Edge computing describes computing capabilities situated at degrees of distance from a centralized location, usually the cloud or a corporate DC. The placement of the equipment is chosen in order to improve the performance, security, and operating cost of the applications and services that will run in that environment. In exchange, some factors may be de-emphasized, such as resilience, availability, and throughput. Edge computing can reduce latency and bandwidth constraints of services by not transferring collected data to the cloud or a DC for processing and thus not needing to remotely retrieve subsequently generated information. Most recently, the edge has also become a frequent deployment target for control logic supporting industrial automation and **machine learning** (**ML**) models used in visual analytics tasks.

By shortening the distance between devices and the computational resources that serve them, the edge brings new value to existing use cases and can introduce new classes of applications. This results in distributing workloads and ML assets southbound along the path between today’s centralized DCs and the increasingly large number of deployed edge computing devices and clusters in the field, on both the **service provider** (**SP**) and user sides of the last mile network – the portion of the SP network that reaches user premises.

*NOTE*

*When the terms “southbound” and “northbound” are used when discussing an application architecture, they refer to points of the compass in reference to the relative location of your current point of view. So, “northbound” would refer to services, tiers, and locations that are more physically proximate to the cloud, or, in the case of*Figure 1.1*, locations to the right-hand side. Likewise, “southbound” refers to locations closer to the user edge, or locations on the left-hand side.*

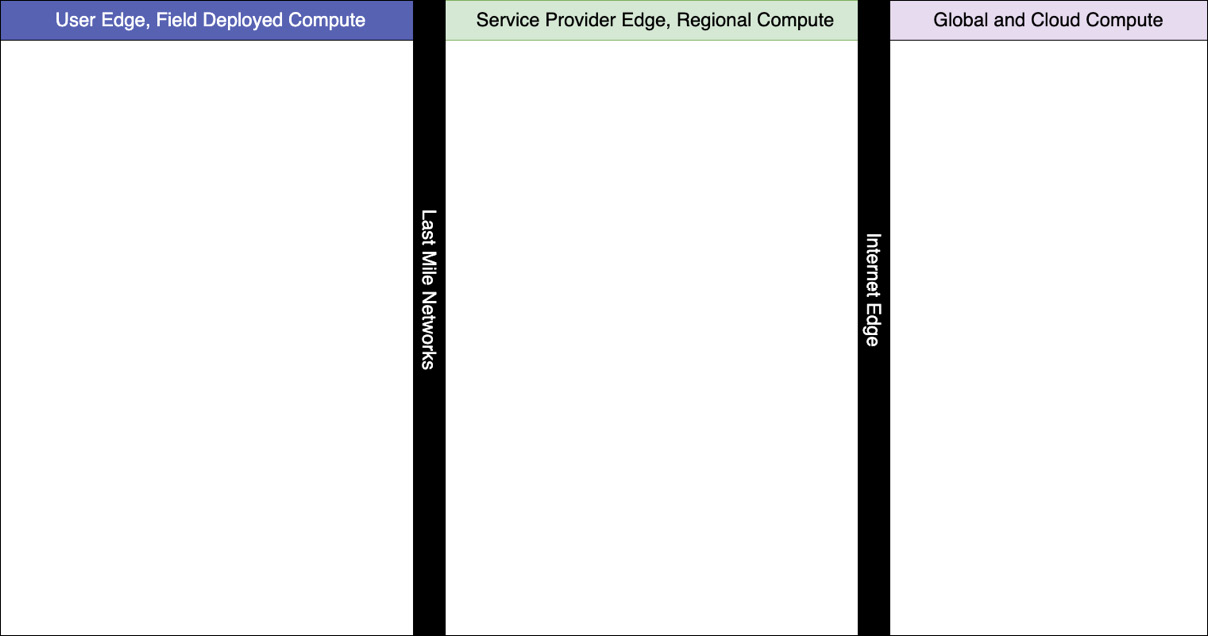


Figure 1.1 – A representation of the edge as distinct from the cloud

Notice in the preceding diagram, which we will be using as a starting point for many charts used throughout the book, how all computing resources located to the left of the thick, black line labeled **Internet Edge** would be considered the edge, and all computing resources to the right of the line would be considered the cloud. Despite those designations, you might hear about “clouds” located in the SP edge referred to as “regional clouds” or “enterprise clouds” and in the user edge as “edge clouds.” Future iterations of the chart will collapse the **Internet Edge** and **Last Mile Networks** lines.

Edge computing, therefore, is described by ways in which its operating environments are different than the physical security, hardware homogeneity, and service scalability of cloud computing. Edge compute nodes (and by nodes, we include both standalone devices and compute clusters) can be solitary, and many times are not rack-mounted. Edge nodes may not have reliable or consistent power, network connectivity, or even air filtering, climate control, and controlled physical access. Multiplicities of edge nodes may not be the same version or brand of hardware, with differing specifications and capacities, and thus edge computing nodes are described as heterogeneous. Edge nodes may use smaller or fewer processors, slower and/or more power-efficient processors, and fewer specialty or co-processors to accelerate specific types of tasks or workloads. Last, edge nodes may have a permanent or fixed placement or might have mobility or otherwise be portable.

Moving on to the contents of edge nodes … the type of chip being used, the micro-architecture, could be mounted in a constrained device or an off-the-shelf, commodity unit, but it typically runs a Linux distribution or similar enterprise- or consumer-class operating system. We do not speak of embedded systems or fixed-function devices of the IoT class as being used for edge computing functions, although they certainly are used to send data northbound to edge computing systems.

*EDGE MICRO-ARCHITECTURES*

*Typical chip micro-architectures supported by most edge computing solutions include:*

*-***x86\_64 or amd64**

*-***arm32 or arm6, arm7**

*-***arm64 or armhf, including Apple M1 and M2**

*-***ppc64le**

*-***risc-v**

*-***s390x***(typically running LinuxONE)*

When writing and packaging applications for the edge, we no longer write an application in a high-level language such as Python, NodeJS, or even Golang, and package it up for a package delivery system such as **pip**, **npm**, and others. Instead, we typically containerize the application to make it self-contained along with all its dependencies so that it doesn’t need to be installed. A container image is downloaded from a registry and run in a container engine such as Docker or Podman. There are also common techniques available to support multi-arch containers that will build and run on all the common micro-architectures listed previously, which is the approach we recommend using. See the following article for more information: <https://developers.redhat.com/articles/2021/08/26/introduction-nodejs-reference-architecture-part-5-building-good-containers>.

NOTE: Containers are not the only edge-native approach for isolating workloads. Enterprises may use **virtual machines** (**VMs**), serverless functions, or even WebAssembly (Wasm) depending on the code-base purpose, or execution environment. Regardless of the chosen approach, proper automation should be employed to ensure isolation is maintained.

## Are the edge and the cloud extremes of the same thing?

In the previous paragraphs, we compared attributes of edge computing nodes largely by contrasting them with cloud computing hardware, connectivity, and facilities. Indeed, edge computing is largely distinguished from cloud computing by pointing out the differences and trade-offs, as depicted by the arrows at the bottom of the LF Edge diagram shown next:

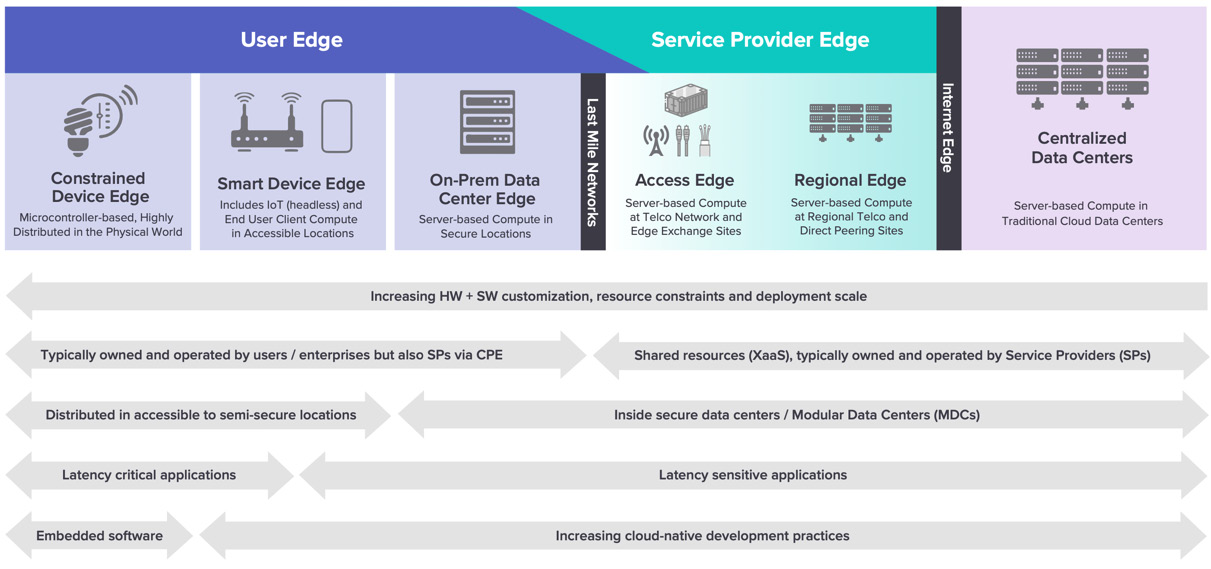


Figure 1.2 – The edge continuum with trade-offs shown at the bottom with gray arrows

Image Source: Linux Foundation Whitepaper

Jeff Ready, former CEO of Scale Computing, has a pithy way of contrasting the edge with the cloud:

“The edge is really the inverse of the data center. A data center deployment is likely 1 or a small number of physical locations, hundreds of servers at those locations. The edge, on the other hand, is hundreds or thousands of locations, with often 1-3 servers at each one. You can’t manage them the same way. You’re still deploying thousands of servers, but with many, many locations you obviously can’t have a team of IT pros at each one like you would in a datacenter. You need automated deployment, automated management, automated error recovery to make the edge work.”

(<https://blocksandfiles.com/2023/02/02/dell-vxrail-edge/>)

Edge computing environments are very different from, and in most cases filling requirements that are the direct opposite of, cloud computing. As Eclipse’s Mike Milinkovich has said: “If you care about the physical location of your devices, then you are doing edge computing.” However, edge computing has been established on a foundation of software development processes informed by cloud-native development best practices. In short, edge computing would not be possible if it weren’t for the cloud.

## How does edge computing bring value, and why now?

Edge computing reuses applicable cloud computing programming best practices, which give it a standard approach to software development that is fast, flexible, and works across multiple architectures. This methodology provides small teams with minimal cross-architecture experience in a way to create cross-platform distributed applications comprised of multiple loosely coupled services. This is what powers edge computing (and cheap computing).

Edge computing came about at a time when inexpensive but powerful compute became plentiful and custom fabrication tools more widely available. The Raspberry Pi single-board computer introduced ARM-based processors to hobbyists around the world at affordable prices while also spawning a large ecosystem of software utilities and hardware add-on boards. Since these systems could run many common Linux variants, they also formed the basis for **proofs of concept** (**POCs**) that could be easily turned into commercially viable solutions.

We’re now beginning to see a similar wave of innovation with RISC-V-based systems that will further enable low-powered and efficient solutions that could even be embedded into standard hardware components. This would bring us to a point where computers with a Linux operating system that are capable of running containerized workloads could be powering every household appliance and consumer device or component. For example, Intensivate is running containers on the RISC-V ISA-compatible SoC that controls SSD drives.

By virtue of having inexpensive but powerful compute available and placed adjacent to where data is being generated, and being able to program that compute using existing tools and methods, you can simultaneously reduce the cost of computation while decreasing response times and reducing latency. Complex analytics no longer require offloading to the cloud, but ultimately, the available trade-offs largely depend on which edge you choose for workload placement.

# Which edge? Categorizing edges

In this section, we cover the names and characteristics of various edge categories (or edges) that are commonly used, including which terms have been deprecated or have fallen out of the vernacular. By the end, you should be able to list the edges and describe the benefits and drawbacks of each, as shown in exhaustive detail next (Figure 1.3):

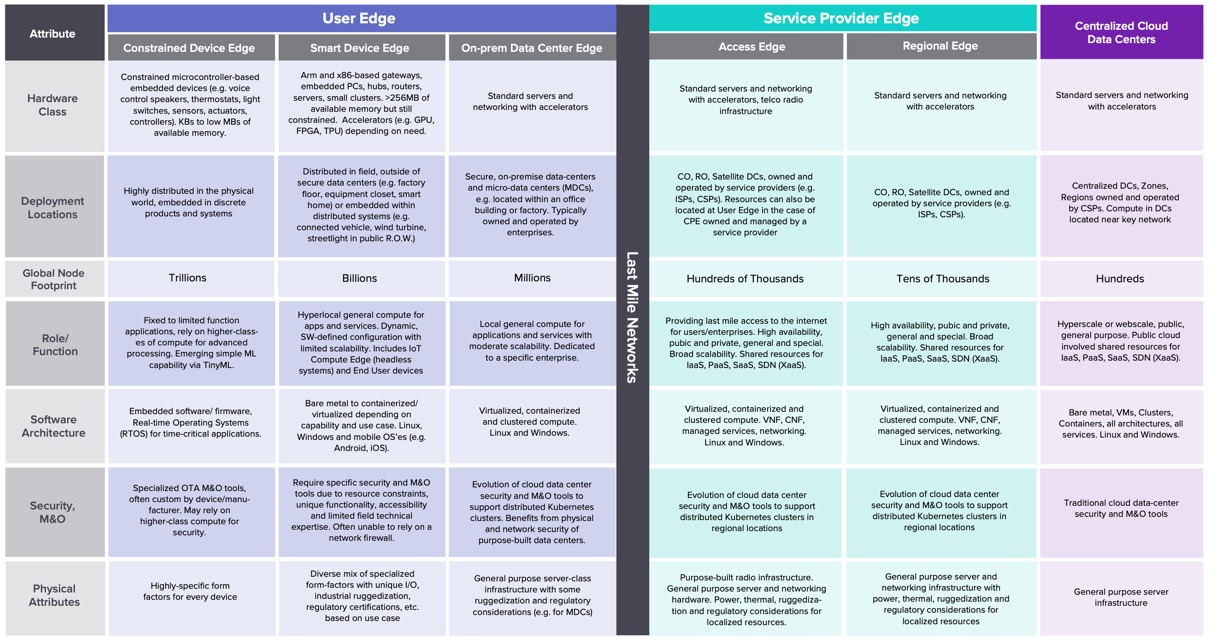


Figure 1.3 – Detailed benefits and drawbacks of each edge

Image Source: Linux Foundation Whitepaper

## The user edge – field-deployed compute

A fundamental confusion that commonly comes up in conversations revolves around where people think the edge is. One idea in people’s minds is that “the edge” may be in houses, commercial offices, factories, vehicles, and utility shacks on the side of the road or at the base of a cell tower. These types of locations are typically thought of as the **far edge** because they are farthest away from a DC or cloud on a network and typically beyond the **last mile** of an SP network – hence, they are at the edge of a network or the farthest possible location on the network from a peering point or exchange.

But, the far edge is not the only location where edge computing takes place. The Linux Foundation’s LF Edge organization refers to all edge computing locations falling after the last mile as belonging to the **user edge**, which follows a nomenclature categorizing types of computing by the owner of that compute. The fundamental assumption is that infrastructure at these locations is typically not shared beyond a single organization, business, or person. The Eclipse Foundation’s Edge Native Working Group terms it **Field Deployed** while seeing the last mile and its associated infrastructure collectively as **Access-Transport**, as shown in Figure 1.4:

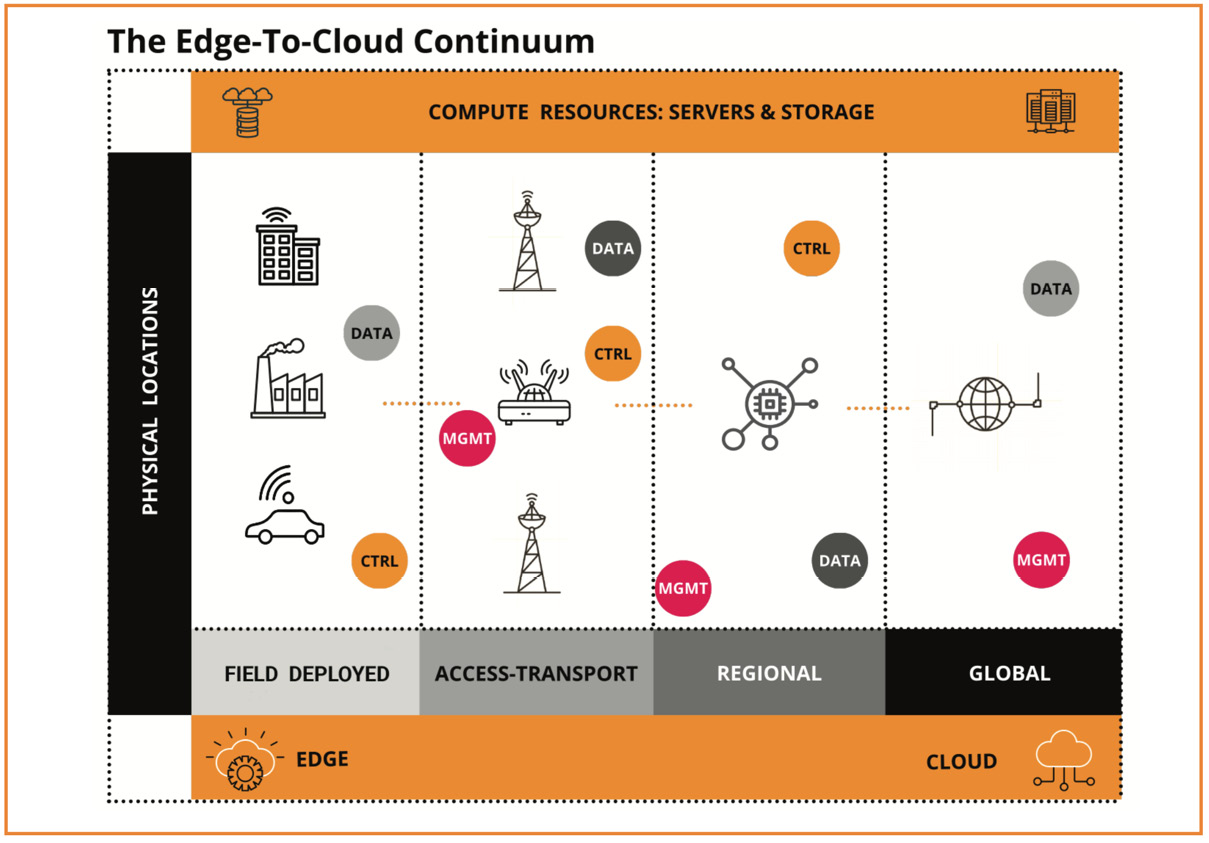


Figure 1.4 – Eclipse Foundation Edge Native Working Group terms for edge

Image Source: Eclipse Foundation Whitepaper

## The SP edge – regional compute

On the other side of that last mile network connection, but beyond the edge of the internet’s network infrastructure, is compute typically referred to as **telco infrastructure** belonging to **communication SPs** (**CSPs**). Satellite locations would be termed **central offices** (**COs**), and larger hubs would be **regional offices** (**ROs**). CSPs themselves are now comprised of phone companies (telcos and mobile operators or **mobile virtual network operators** (**MVNOs**)), **content distribution networks** (**CDNs**), and newer edge providers. They operate and offer programmable compute consisting of both software-enabled communications equipment and a limited amount of traditional racks of compute. LF Edge terms this collective category as the **SP edge**. The Edge Native Working Group calls this category simply **regional** and avoids associating it exclusively with SPs.

As the field of IoT computing was maturing, but before edge computing had gained widespread adoption, Cisco and others began using the term **fog computing** to refer to a method of distributing key cloud-like computing and storage infrastructure and services outside of the cloud and closer to devices. The term never entered widespread usage and was soon supplanted by the more general term **edge computing** to cover all computing outside of the cloud on programmable devices.

And lastly, you have traditional DCs. These are locations where physical access is controlled, racks of homogenous compute hardware are provided and remotely manageable, and both **Platform-as-a-Service** (**PaaS**) and **Infrastructure-as-a-Service** (**IaaS**) are offered. These DCs are typically owned and used by a single party, but mostly only differ from the cloud or global compute by size and scale. Large DCs such as these are rarely referred to as locations where edge computing happens; however, they are technically part of the edge as long as they reside outside of the internet’s network.

# Your computer or mine? Tactics for service deployment

In this section, we review different scenarios where more than one application or service can be deployed and running concurrently. We will give an example of each and discuss the purpose of that approach. By the end, you should recognize when each tactic is required.

## Edge computing doesn’t require dedicated resources

In Sharpening the Edge, the author refers to **edge computing** as primarily using your own systems, while **cloud computing** involves sharing systems and infrastructure with others. This simplification is largely correct at the macro scale, although edge computing can also involve sharing: by running multiple applications on a single device at the same time for multiple users, or sequentially at regularly scheduled intervals (day versus night, weekdays versus weekends, business open versus business closed). In the case of CSP-hosted edge infrastructure, it could even include hosting applications from multiple tenants each in an isolated environment on shared edge devices or clusters like cloud providers would. Let’s take a deeper look at examples of each type of sharing in turn.

## Single device, running multiple applications simultaneously

In a New England-based chip factory in late 2022, as reported by IBM in a blinded case study, IT staff deployed cameras containing a CPU and GPU capable of running Linux and containers. On those cameras, they placed containerized applications and ML models to detect if persons were wearing protective equipment, if the equipment was being worn properly, and if they maintained a safe distance from hazardous locations.

This involved multiple containers for object detection, object recognition, and object placement, as well as for sending messages and relevant screen captures extracted from video (with individual identities blurred out to preserve privacy) when infractions were detected above a specified level of certainty. The messages were sent over the local Wi-Fi network to an application on the shift manager’s mobile phone within seconds of detection whereby the receiving application alerted the manager to investigate further if the provided still image appeared to warrant it.

Before edge computing devices made this possible, a solution would have been built to stream video to the cloud where inferencing would have been performed. By the time the manager would have been informed, minutes would have passed, and the individuals would likely no longer be in the area. Edge computing removed the expense of transporting video feeds to the cloud, reduced the resulting inferencing latency, eliminated any cloud computing costs, and ultimately ensured that the manager was notified up to 3 or more minutes sooner.

Before the cloud, the factory would have sent **closed-circuit television** (**CCTV**) feeds to a monitoring location where one or more persons would have viewed a bank of screens looking for issues on low-resolution displays, and called a manager if they spotted any issues. This approach would have been even more expensive and slow, and thus only likely to have been used to prevent major losses or accidents, or recorded and reviewed by investigators at a later time to determine potential causes of an accident.

## Single device, alternating applications by schedule or purpose

In a grocery store, low-resolution video cameras stream feeds to constrained edge nodes attached to cameras containing low-power CPUs and limited RAM. These compute devices can run limited inferencing in a single container using tiny ML models at a few frames per second.

With those capabilities, they are used during store hours for spill detection or traffic counting when pointed at an aisle, dwell time when pointed at an end cap, and shelf restocking when pointed at a row. After the store is closed, those applications are replaced by a security application that looks for the presence of persons when the location should be unoccupied.

Edge computing makes these capabilities possible at an operating cost of pennies per day and without needing more connectivity than a local network connection.

## Hosted edge infrastructure – applications on shared compute

An edge SP that began by providing only network peering with internet backbones and cloud providers has now begun offering bare-metal servers with connectivity to one or more providers of customer choice and large amounts of low-cost bandwidth. They do not provide infrastructure or platform services, making their offering ideal for customers who need always-on, reliable, inexpensive connectivity while at the same time providing low latencies due to physical proximity to customer facilities.

Hosted edge nodes are ideal for supplementing customer edge workloads temporarily while also avoiding the vendor lock-in of proprietary cloud solutions. The drawback to using hosted nodes is that the customer will need to provide any infrastructure and platform services and support. But it makes for an excellent extension or overflow to existing customer DCs or for situations when a company outgrows its existing locations but has not yet secured new facilities.

It also works well for temporary events, especially when they take place in a specific or limited geographical area. To that end, there is even a start-up that will deliver a self-contained edge DC in a container to your location for as long as you need it.

# Cloud-out versus edge-in

Another common way of describing an architecture is based on its foundation and assumptions, and in what direction and manner it grows as its scope and responsibilities increase. If it starts out based in the cloud, using cloud-native best practices, and then later adds on capabilities that allow it to run in some fashion on the edge, that approach is described as **cloud-out**. On the other hand, if it starts out on the edge using edge-native development best practices, and then bursts to (hybrid) cloud infrastructure, that would be termed **edge-in**. Let’s look at each in turn, go over an example, and discuss their relative strengths and weaknesses.

## Looking deeper at cloud-out architectures

Cloud-out architectures begin with the cloud; that is, with global compute infrastructure using cloud-native development best practices. While [Chapter 11](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_11.xhtml#_idTextAnchor208) will discuss these practices in depth, let’s list an abbreviated version to frame the discussion:

* Package independent, autonomous services in lightweight containers or other abstractions
* Develop in the most appropriate language and/or framework for the purpose
* Compose with loosely coupled microservices
* Provide API-centric interfaces supporting common protocols and conventions
* Deliver a clean **separation of concerns** (**SoC**), especially between stateless and stateful services
* Abstract from underlying dependencies, both hardware and operating system
* Deploy on elastic infrastructure to automate scale-up and scale-out
* Rely on independent, automated application lifecycle management
* Implement **configuration as code** (**CaC**) maintained within an agile process
* Define resource allocation through declarative policies

Source: https://thenewstack.io/cloud-native/10-key-attributes-of-cloud-native-applications/

Looking at the preceding summary, a good example of the cloud-out approach to application development would be building a product using the SaaS model. SaaS is a way of building and delivering software so that it does not need to be installed (it is hosted by someone, usually in the cloud), and it does not need to be purchased (it is paid by subscription). Access can be provided in a browser or over the internet via APIs.

SaaS solutions demonstrate the cloud-out approach because they are typically hosted in the cloud, implemented with microservices, abstracted from dependencies, and deployed on elastic infrastructure. Individual SaaS offerings may also follow many other cloud-native best practices, but compliance with those principles is not obvious without access to the source code.

SaaS solutions clearly do not demonstrate the edge-in approach because they are not typically built to handle dependency unavailability (hence built with highly available architectural principles), service portability, target system constraints (since they do not need to be installed or remotely deployed), and dependence on orchestration. Additionally, they presume an always-on network connection with low latency and high throughput.

So the pros of the SaaS architecture approach are that the application and its constituent services do not need to be built for wide micro-architecture compatibility and thus can be narrowly tailored to the deployment target’s specific hardware requirements. It can rely on the hosting facility provider to maintain and support the infrastructure, platform, and connectivity as well as the facilities themselves. And within reason, they can scale horizontally up to the provider’s available capacity.

The cons of this approach are that the application and services may be narrowly tailored to the environment and are thus not inherently portable, and the risk of vendor lock-in is considerable. This may also render the application brittle when exposed to new or unanticipated conditions, thus requiring more ongoing maintenance than if it had been built using edge-native principles. Finally, a SaaS service will clearly be unavailable in the absence of a network connection.

## Delving into edge-in architectures

Edge-in architectures start with field-deployed compute and adhere to the edge-native programming model. [Chapter 11](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_11.xhtml#_idTextAnchor208) will cover this topic in greater detail, so here’s a quick summary:

* Tolerate interruptions and unavailability of service dependencies and connectivity
* Design for service operational portability between system tiers
* Avoid explicit dependence on container orchestration features
* Employ external application configuration and secrets per instance, including support for runtime updates
* Anticipate system-imposed constraints
* Ensure services are self-contained
* Follow data privacy regimes based on target requirements
* Leverage platform-provided services when possible

Source: https://www.ibm.com/docs/en/eam/4.5?topic=clusters-edge-native-development-best-practices

Based on the preceding points, a good example of an application following this approach would be one or more multi-arch containerized gardening applications running on a Raspberry Pi, which is connected to moisture, light, humidity, and temperature sensors. The application can retrieve data from the sensors, persist it locally, and make it available through a web application displaying a page with the data or making it available through a REST API as a data payload.

This architecture follows edge-in principles due to its ability to collect data from a sensor while it is connected, to provide the collected data while it maintains a network connection, and to resume operation when rebooted or after an interruption of power. The applications are portable and can be run on multiple architectures, providing that sensors are present on that machine. They do not require the orchestration that Kubernetes provides. They do not require services residing on another machine; the data stays resident on the machine itself unless manually moved.

The application architecture does not demonstrate a cloud-out approach because it may not be abstracted from the underlying hardware dependencies (the sensors), and it does not use elastic infrastructure to scale.

So, the benefits of this approach are that the solution is resilient to adverse conditions and can function fine without an internet or network condition, being completely self-contained. It also runs on inexpensive hardware and would cost pennies in electricity to operate monthly. And when the hardware eventually fails, it is cheap to replace.

Wrapping up, you’ve seen the differences between cloud-out and edge-in architectural principles, compared how they work in the real world and read about the pros and cons. Now, let’s discuss the larger patterns that most applications follow on the edge … the archetypes.

# Introducing archetype patterns

In this section, we introduce you to the concept of archetypes. Along the way, we cover the days involved in the software life cycle and discuss deployment methods. By the end, you will be ready to learn about and use archetype patterns.

## What is an archetype?

An archetype is the original model or form of something that embodies all primary qualities of that item, whether that item is an abstract concept or a physical object. In the case of architectural patterns, our assertion is that most application architectures can be derived from an original pattern archetype (source) or a slight variation based on local or business considerations. Therefore, in this book, we attempt to tease out archetypes and discuss them at length based on our belief that identifying and mastering them will give you the foundational skills needed to tackle most, if not all, edge application scenarios. By following these patterns judiciously, you will be able to create solutions that are not only portable but also future-proof.

In the archetype-pattern diagrams to be found in this book, we will follow certain conventions:

* Show all elements of equal size and shape so as to imply that all components are of equal importance
* Refer to a component by its function or role instead of a product name, which emphasizes replaceability and a vendor-neutral approach
* Indicate placement by edge category as denoted in columns ranging from the cloud on the right to the field-deployed far edge on the left
* Draw rectangles around edge nodes to indicate placement of components, applications, and services
* Connect edge devices, nodes, systems, platforms, and infrastructure with arrows indicating the direction of data flow

See an example diagram at <https://wiki.edgexfoundry.org/display/FA/Open+Retail+Reference+Architecture?preview=/55705782/81625502/Open%20Retail%20Reference%20Architecture%20Diagram.png>.

Once you settle on an architecture pattern on day 0, based on the best fit to the requirements and business needs, the next consideration you need to address is how to deploy that architecture … both on day 1 and maintaining it on days 2-N. Let’s delve into the idea of days 0-N, which will also be used in [Chapter 7](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_07.xhtml#_idTextAnchor131) when we discuss applying automation to these deployments.

## The days of software creation

When you are involved in creating and launching a software product or an application, it is helpful to think about distinct blocks of work that are typically worked on at the same time. This categorization applies whether you work in project management, software architecture, or operations (DevOps or SRE). Those blocks or categories are denoted as “days” followed by a number to indicate where they fit in the sequence of events:

* Day 0 is when initial planning and preparation happen. These types of tasks involve designing, which can include traditional UX design, as well as application and information architecture. The final product requirements should be specified, agreed to, and documented by the end of this timeframe. The goal of these tasks is to prepare for software development work to begin.
* Day 1 denotes programming, provisioning, and configuration of environments and pipelines. That would also include dependency installation, finalizing automation, and all unit and **end-to-end** (**E2E**) testing. At the end of this period, the application should be documented, approved by all relevant parties, and ready for launch into a production environment.
* Day 2 marks the support period where time is spent working on issues, optimization, A/B testing of new features, blue/green testing of differing versions, and so on. This time should also be used to build up a support database, any FAQs, and possibly training support chatbots or other automated response mechanisms.
* Day N (end) would then be for activities related to the retirement of an application and its provisioned environment, assets, and supporting infrastructure. By the end of this period, all traces of the application should be removed except for any items of historical value or required to be kept intact for legal retention purposes:

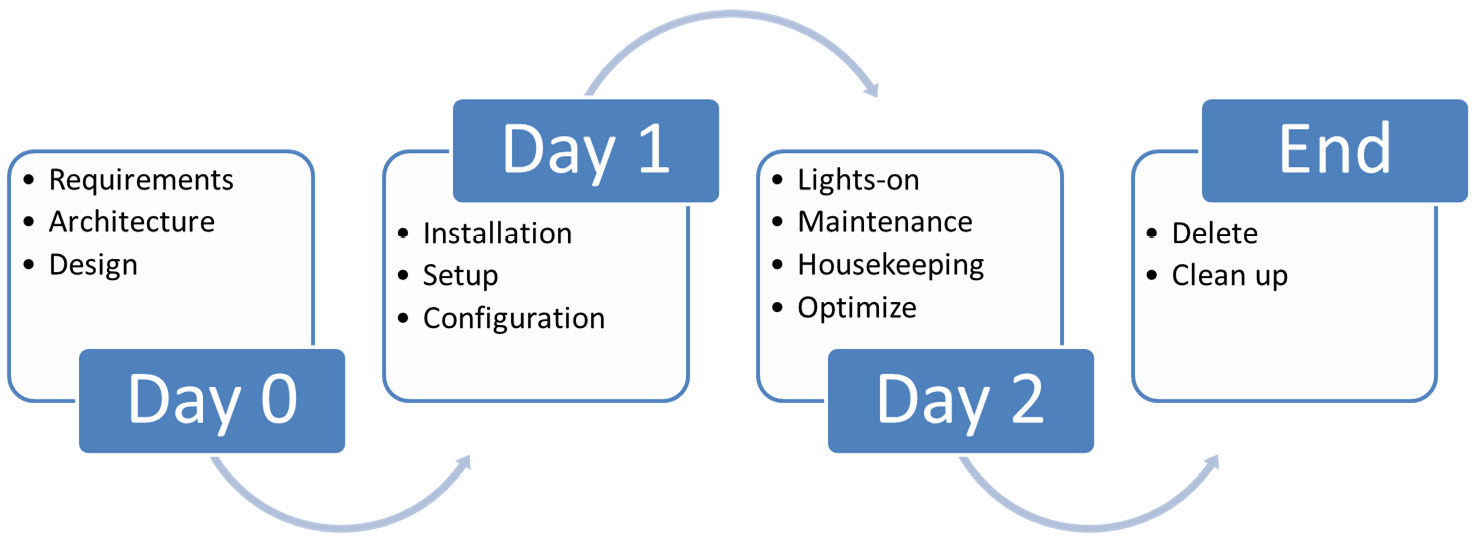


Figure 1.5 – Depiction of days 0-N categories with major activities shown in each

Image Source:: <https://dzone.com/articles/defining-day-2-operations>

## Deploying archetype patterns

When it comes to deploying solutions that live entirely in the cloud, you can rely on IaaS and provider-specific deployment tools and configurations. This is made easier due to being able to integrate with **identity and access management** (**IAM**) solutions and because both managed and DIY components can be well integrated. Even when your solutions span multiple clouds, there are well-known open source deployment tools designed to abstract away cloud provider-specific details and allow you to focus on your specific tasks.

But when it comes to field-deployed solutions at the far edge, potential connectivity issues and heterogeneous systems, combined with a lack of infrastructure services, will require a different approach. We’ll go into greater detail in Chapters 9 and 11, but using deployment and application life-cycle management tools that can operate autonomously and pulling configuration from a central control plane rather than pushing to the edge tend to resolve most of those deployment issues, including scalability and security.

# Summary

In this chapter, you learned about what capabilities the edge and the cloud have in common, and what distinguishes one from the other when it comes to the tiers of compute: infrastructure, platforms, services, and applications.

We covered the names and characteristics of edge categories and described the benefits and drawbacks of each. We also discussed various ways that edge nodes can be shared between tasks. We described how an architecture can be scaled based on its scope and responsibilities with cloud-out and edge-in paradigms. Also, we introduced you to the concept of archetypes and the “days” involved in the software development life cycle.

Now, you should be ready to learn about the basic components and building blocks that go into archetype patterns. Just as importantly, we’ll also discuss how to approach solutions from the right perspective in order to build a future-proof application, follow best practices, and use long-term thinking.

# 2

# Edge Architectural Components

Edge computing architectures, although relatively new, have their origins in IoT architectures. There are a lot more devices in play now, some with compute and storage. These devices are key to edge computing architectures. The different sizes, form factors, the compute, and storage capacity of these edge devices make for many variations in solution architectures.

These solution architectures are unique because there are limitations at each layer, from device to compute to storage. Architects designing them often must think about the limitations, especially when it comes to the far-edge aspect. One must keep in mind the intrinsic benefits of edge computing such as low latency, high performance, less power consumption, high bandwidth, and multiple dispersed locations. Edge computing has given rise to a new paradigm of application architecture specifically designed to run in the distributed edge domain, which we call **edge-native applications**.

This chapter describes the four major roles of the components in an edge architecture. We then talk about the common functional and **non-functional requirements** (**NFRs**) and discuss the software and hardware components that commonly go into the creation of edge architectures. It concludes with a discussion of device architectures, data transmission protocols, and architectural decisions. The main topics are as follows:

* Edge components
* Functional requirements
* **Non-functional requirements** (**NFRs**)
* Use cases and patterns
* Architectural decisions

# Edge components

There are four major roles for the edge components in an enterprise’s edge computing architecture: edge devices, the edge gateway, or server in the enterprise edge (part of the user edge’s field deployed compute), the micro data center in the Service Provider Edge’s Regional Compute, and the enterprise cloud. The edge server not only acts as a gateway to connect all edge devices in a secure manner but also allows for the management of all those devices. See Figure 2.1:

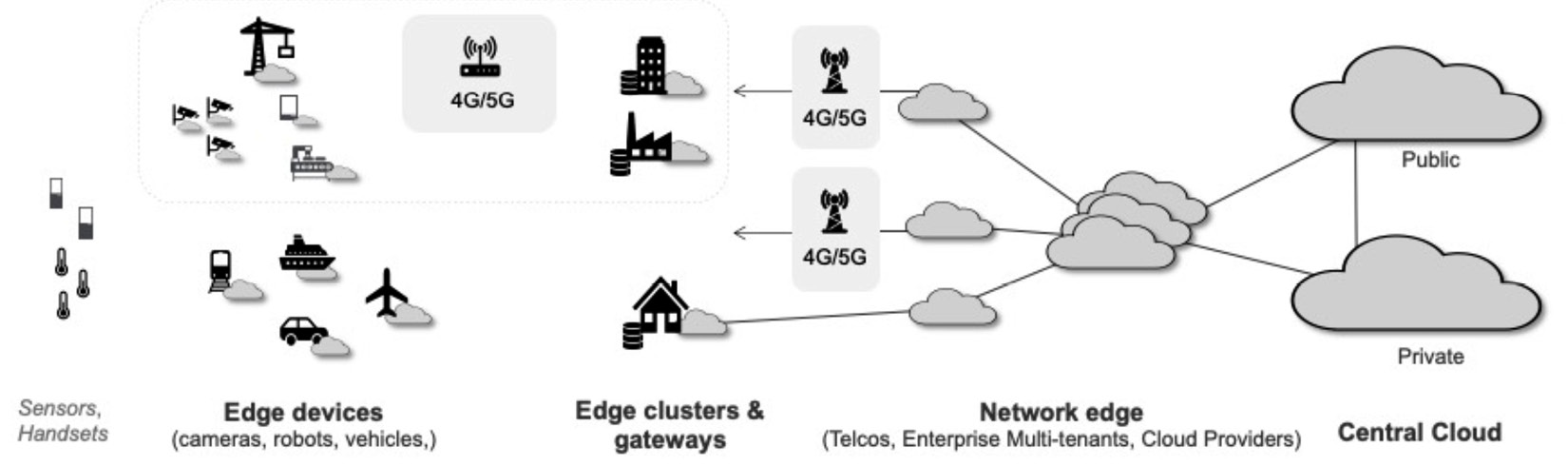


Figure 2.1 – Common representation of the edges in edge computing

The enterprise cloud is shown on the far right. This could be a public, private, or hybrid cloud, which is the domain of hyperscalers. To the left of it is the realm of regional compute, where the telcos or **communications service providers** (**CSPs**) operate. Next to it is the user edge (in this case, the user could be an enterprise) where the edge clusters and gateways are deployed. On the far left, IoT and edge devices are shown, including sensors, gauges, cameras, robots, and the like.

At a macro level, there are four aspects to an edge computing solution and you will see them reflected in the architecture diagrams that follow in this chapter. They are as follows:

* **Edge devices**: While some IoT devices such as sensors, gauges, and actuators cannot run any software, many edge devices have some computing power and some storage. That lets them store some data and run some simple analytics. Depending on their form factor, certain edge devices have enough compute, memory ranging from 128 MB to 256 MB, and almost 1 GB of storage, which is enough to analyze the data and perform real-time inferencing without needing to send the data to a backend server or the cloud. That is what edge-native computing is all about. If they are not using ARM architecture, the devices could be x86 class CPUs equipped with one or two cores.

Note that now it is possible to deploy commodity AI accelerators connected to a USB port to supplement their inferencing and analytics capabilities.

* **Edge servers**: The other components in the user edge space are edge servers or **edge nodes**. There is a one-to-many relationship between an edge server and edge devices. Edge servers or gateways are constantly in touch with edge devices by way of agents running on the devices and are used to deploy applications onto those devices. These are typically **commercial off-the-shelf** (**COTS**) computers that could be located in a distributed facility such as a factory floor, store backroom, warehouse, or remote office. These could be ruggedized or placed in a protective enclosure. The small-sized machines have 8 cores, the medium-sized machines have 16 cores, and anything with more compute capacity would constitute a large machine. The memory in these machines starts at 16 GB RAM and they could have hundreds of gigabytes of storage. If inferencing at the device is insufficient, then data from the far-edge devices is sent to the edge server or even to the cloud for further analysis and deeper insights.
* **Regional compute or service provider edge**: This edge is sometimes also referred to as the network edge or micro data center. CSPs are taking advantage of newer networking technologies to create these regional clouds or local clouds that provide software-based infrastructure services for devices to communicate with at the far edge of the network. The major selling point of the telcos is that data from the edge devices does not have to be sent to the cloud but can reside in this regional cloud, thereby reducing the distance and time that data must travel. For the end user, it means decreased latency, better bandwidth, and more security. This is especially true with the use of 5G.
* **Enterprise cloud**: This is the centralized cloud that could be a public or private cloud or an on-premises data center. As is common to clouds, enterprises get unlimited compute and storage along with management capabilities, plus access to a growing portfolio of other cloud services. From an edge computing perspective, this is home to four facets: storage of most of the device data, device management at a global level, AI model building and training, and enterprise-level analytics.

Now that we have seen the major components that one finds in edge computing solutions, the next thing is to dive into some of the functional requirements that enterprises ask for.

# Functional requirements

In [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013), we talked about cloud-out and edge-in paradigms. Cloud-out is where computing is taken out of the data center and brought to the far edges of the network. Conversely, the movement of the generated data from the source or the edge to a location with more computational resources for analysis is the edge-in part. Those facets drive functional requirements in an edge computing solution. We will discuss the common functional requirements of an edge computing solution in the subsequent sections.

## Sensing

This is where we still deal with traditional sensors (e.g., IoT/accelerometers, thermometers, or actuators) as architectural components that acquire data and help create a signal. When combining technologies such as edge and **artificial intelligence** (**AI**), you introduce new ways of designing and deploying technology, thus improving and automating situational awareness with sense-making systems. These are deployed in stores, shop floors, industrial equipment, mines, and even in vehicles.

They are often referred to as systems that have situational awareness. These sense-making systems fuse human-like thinking with sensing technologies so they can take actionable insights to augment humans in their work or help humans. Additionally, such systems can support **augmented reality** (**AR**) deployments by providing data that will be “visible” to humans working nearby. That is the core idea of these edge computing patterns. Businesses that want to create edge/IoT solutions should have a good understanding of their assets, especially their people, because that helps them in making better decisions while incorporating this newly enabled sensing technology.

## Inferencing

Inferencing, by definition, is the act of reasoning from factual knowledge. Inferencing at the edge means providing actionable intelligence using AI-powered techniques based on the data gathered by different types of devices, such as sensors, cameras, microphones, and so on. One of the outcomes of real-time inferencing at the source of the data is a better security posture. This is because data does not have to travel too far, which reduces the attack area.

## Analytics

Edge analytics is the ability of **machine learning** (**ML**) models deployed on edge devices outside of the cloud or data center so that deeper analysis of the collected data can be done. The creation, training, and retraining of models are typically done on the user edge or the service provider edge.

Depending on the industry scenario involving imagery, the requirement aspects can expand to include image classification, object detection, and anomaly detection. While visual edge analytics is most obvious and common, we should point out that there are many other types of data produced by IoT-type devices that get analyzed – temperature sensors on freezers and ovens, shock and vibration analyzers on wheels and other moving parts, noise detectors to isolate certain types of noise, flow meters for liquids, pressure gauges in industrial settings, and speech and tone analysis. Collision avoidance systems in many new automobiles use multiple proximity sensors in lieu of cameras, while others use LiDAR technology.

The lifeblood of the three functional requirements mentioned so far is data. So, it behooves us to talk about data as one of the key requirements. [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110) does take an in-depth view of data.

*REMINDERS*

**LiDAR***stands for***light detection and ranging***. It is a technology that uses laser light to measure the distance between objects. It is used in autonomous vehicles, astronomy, and other applications.*

## Data

Whether it is audio, visual, telemetry, or sensor data, it is all about the generation, collection, movement, and analysis of data in an edge computing solution. Ensuring the data doesn’t have to travel far allows it to be analyzed more quickly. And, in case the data must be sent to an edge server, it takes less time compared to a traditional centralized cloud environment.

In any IT solution architecture, more so in edge computing architecture, the questions that solution architects are faced with are the following:

* Where should the data be sent to be stored?
* Where should the data be sent to be analyzed?

The solution architect must also determine whether it is critical to analyze the data at the source in real time and whether the data calls for more in-depth analysis in the future. Whether or not to store the data generated by the devices depends largely on business and jurisdictional compliance requirements.

Another view is to take a t-shirt-sized approach. Remember the footprint of the deployed applications ranges from being large to medium to small as you move away from the cloud – that is, going from right to left in Figure 2.1. While real-time inferencing is done by the far-edge devices, some of the deeper rules-based or even neural network-based analytics take place in the middle layers, while ML model building, training, and retraining, which require a lot more computing power, end up being performed in the layers to the right.

*REMINDERS*

**Inference***is coming to a conclusion based on evidence and reasoning.*

**Analysis***is the process of methodically breaking something down to gain a better understanding of it.*

Some of the common functional requirements have been discussed. There could be more, depending on the industrial use cases.

In the next section, we look at the non-functional requirements.

# Non-functional requirements

Edge computing architectures must satisfy several NFRs. While low latency and high bandwidth are two of the most common and obvious requirements, enterprises often list security as their most important requirement. The other NFRs are related to service management and operations.

## Security

When discussing inferencing, we mentioned that one of the outcomes of real-time inferencing at the source of the data is a better security posture since data doesn’t have to travel far, thus reducing the attack area. That said, any and all data in transit must be sent using a secure protocol. Onboarded devices could use keys, security certificates, or both. As in all IT solutions, components in an edge solution more often than not must meet regulatory, compliance, and local security standards because they deal with data. For a discussion of this topic in depth, see [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110), where we cover security aspects of data in motion and data at rest.

## Service management and operations

Given that an edge computing solution can have many edge devices and that these devices could be installed in remote or hard-to-reach locations, there are specific NFRs related to service management and operations that are described here:

* **Availability**: Availability is key in an edge computing architecture. The edge devices must always be available to gather or create data, and the solution must continue to operate even when there is no connectivity – that is, in a disconnected mode. It is important to eliminate all **single points of failure** (**SPOFs**) from the solution.
* **Reliability**: This is the other side of the availability coin. In certain edge devices such as moving vehicles, safety-related computations must always return correct results in a predictable timeframe, even if this decreases the overall availability of the system. A device can be available but may or may not be reliable, whereas a reliable device will have high availability.
* **Latency**: Because edge computing is so tightly tied to the network, you commonly hear about latency and bandwidth. Getting data from the devices and acting upon that data expeditiously is critical. Reduced latency is one of the primary requirements in any edge solution.
* **Resiliency**: The ability of a system to manage itself is important. This requirement refers to internal failures as opposed to external ones. When conditions change, are the agreements between the devices and the edge hub immediately terminated or refreshed? Make sure that agents and services are always in a well-defined state.
* **Scalability**: Scalability, when it comes to the number of devices deployed and the management of those devices, is usually an assumed requirement, and this can range from tens to thousands to millions of devices. Systems must be designed to handle the onboarding and deployment of any number of edge devices and edge servers. Figure 2.2 shows the scaling going from right to left:

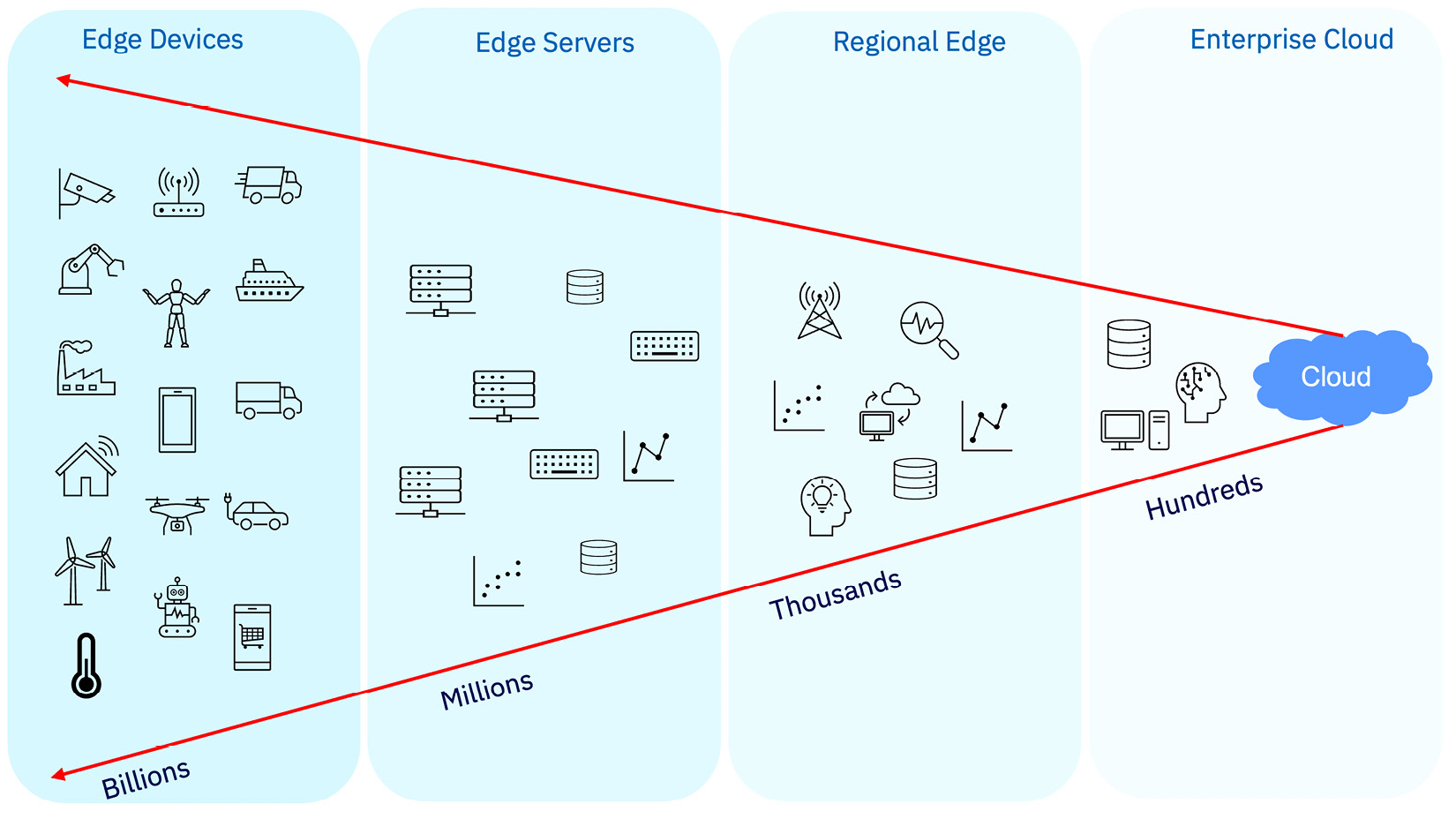


Figure 2.2 – Edge computing at scale

* **Maintainability**: Given that there might be thousands of devices, often in remote locations, maintaining them on a regular basis becomes an important requirement. Having a planned maintenance schedule with specific maintenance scenarios addresses this requirement. But what about unplanned maintenance? One might argue that unplanned maintenance is exactly what edge solutions are designed to prevent and handle.

We’ve talked about requirements, both functional and non-functional. Remember, there may be more, depending on the use case. Let’s look at some of the common edge use cases.

# Edge use cases and patterns

Edge use cases depend a lot on the industry that is targeted. So do the related applications that get deployed on the devices that generate or collect data. While the software stack might be similar, the form factor of the devices in play will determine the size and complexity of the applications. The applications will depend on the use cases relevant to a particular industry. For example, a camera might be trained differently when acting as a security camera than one looking for welding defects on a manufacturing shop floor. The devices are similar, in that they are both cameras, but the application or workload running on them is different depending on their function.

Edge workloads are often used to describe any edge-hosted service. Any software application that has utility when running in an edge node can be described as an edge workload. Such functionality is usually delivered as microservices running in Docker containers or related container technology, but it can have other forms as well. Examples of edge services include vibration analysis, visual recognition, acoustic insights, speech recognition, and so on. Solution architects must decide on what workloads run where based on device constraints and also the data posture.

While not exhaustive, Table 2.1 captures some of the common edge use cases (see <https://www.redhat.com/en/blog/edge-automation-seven-industry-use-cases-and-examples> for reference):

|  |  |
| --- | --- |
| **Category** | **Brief description** |
| Manufacturing operations | Automation of factory operations and quality improvement analysis, enabled by data collection from sensors, connected metrology systems, and AI. Includes root cause analysis for identification and correction of process issues. |
| Manufacturing quality inspection | Using visual data  Visual inspection for manufacturing plants, industrial equipment products, and transportation. Helps identify potential issues in the use and production of equipment through visual inspection of products. |
| Using acoustic data  Manufacturing quality inspection (AI acoustics) on the plant process floor using acoustic data for quality – for example, sounds of welds during robotic processes or sounds of motors in engine rooms indicating damage. |
| Asset management/ supply chain | Remote tracking, monitoring, and predictive maintenance of heavy equipment and operator network equipment, and automation of field service maintenance. Also includes remote monitoring and tracking of moveable items such as freight, food, and animals. |
| Public safety and emergency response | Connected systems to monitor, alert, and coordinate responses to public safety incidents and environmental issues. Includes in-car camera systems and body-worn cameras used by public safety officials. |
| Transportation infrastructure management | Use of sensors to enable real-time monitoring, adjustment, and maintenance of transport infrastructure, road traffic, and airport facilities. Includes the automation of public transportation systems. |
| Predictive maintenance and asset reliability | Using high-volume sensors, IoT, weather, and fleet data to predict reliability or optimize assets based on resource health insights from operational data and analytics. Leverages device telemetry to create a digital twin model of physical equipment. |
| Connected vehicles**\*** | Connected vehicles that enable coordination with infrastructure (V2I) and other cars (V2V), deliver content through infotainment systems, and feed data to insurers. Also includes fleet management for coordination of fleets of vehicles and tracking of vehicles for security. |
| Smart retail | Use of real-time sensors, displays, and connectivity in and around retail stores. Enables personalized digital signage, context-sensitive in-store marketing, insights into buying behaviors at the point of sale, and monitoring and adjustment of inventories. |
| Smart home | Connected home systems to monitor and manage security, such as perimeter breaches, as well as other home functions such as energy usage, lighting, heating, and locks. Does not include smart utility meters, home entertainment, media devices, or gaming consoles. |
| Financial crime prevention | Leverage applications for preventing fraud and security lapses; Edge provides the ability to run crime checks across different regulatory boundaries – for example, international banks. |
| Secure healthcare systems | By deploying secure edge apps across clinic and hospital locations, medical records and other sensitive patient data are encrypted when transmitted or stored and is often anonymized when accessed by other systems and non-essential personnel. |
| Building automation | By leveraging actuators, sensors, and control systems in buildings to monitor and control various aspects such as HVAC, energy, lighting, and security, edge computing enhances building automation systems. While the term “smart home” covers things inside the home, **building automation system** (**BAS**) deals with the services around and about the building. |

Table 2.1 – Edge use cases

**\* Software Defined Vehicle (SDV) builds on the connected car concept. It is a vehicle in which all the features and functions are controlled and driven by software. Whether it is the vehicle controls or infotainment system, the driver and passengers interact directly with the in-vehicle software platform** (<https://www.ibm.com/blog/the-software-defined-vehicle-the-architecture-behind-the-next-evolution-of-the-automotive-industry/>).

The use cases are dependent on the industry and the edge devices depend on those industry use cases. In the following section, we list some of the device specifications and the data transmission protocols used by the devices.

## Edge device specifications and protocols

[Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013) talked about the typical chip configurations used in edge computing solutions. They are listed again here for completeness:

* x86\_64 or AMD64
* ARM32 or ARM6, ARM7
* ARM64 or armhf, including Apple M1 and M2
* ppc64le
* RISC-V
* S390x (typically running LinuxONE)

Edge device types include gauges, sensors, actuators, cameras, robots, and other IoT core devices. These days, drones and automobiles have become the ultimate device types at the edge.

Some of these devices do not have any compute or storage capacity. In those cases, the devices gather or generate data that is transmitted to the nearest edge server or gateway acting as a hub (some refer to this sector as the first mile). That data, depending on its type, can be sent using any of the different protocols that are available. Table 2.2 lists some of the common data transfer protocols we see. The protocols are listed in alphabetical order and do not distinguish between wired and software protocols. We have also included some of the legacy protocols such as BACnet and Modbus:

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| BACnet | BACnet is a communications protocol designed to allow communication of building automation and control systems. |
| Bluetooth LE | Bluetooth Low Energy is a wireless personal area network technology used by many modern-day devices. |
| HTTP/HTTPS | Hypertext Transfer Protocol/Secure are communications protocols that are stateless. They are the foundation of the internet. |
| Kafka | Apache Kafka is an event streaming platform used to collect, process, store, and integrate data at scale. |
| LoRa | LoRa (Long Range) is a physical proprietary radio communication technique. |
| Modbus | Modbus is a data communications protocol for use with programmable logic controllers. |
| MQTT | Originally an initialism for MQ Telemetry Transport, this is a lightweight publish/subscribe machine-to-machine messaging connectivity protocol.  Now there is Sparkplug from Eclipse Foundation that provides MQTT protocol, ideal for industrial automation. |
| ONVIF | ONVIF is a standard for how IP products within video surveillance and other physical security areas can communicate with each other. |
| OPC UA | OPC Unified Architecture is a cross-platform, open source standard for data exchange from sensors to cloud applications. |
| RTSP | Real-Time Streaming Protocol is a stateful protocol used for video contribution. |
| Streams over HTTP | One of many HTTP-based adaptive protocols. |
| WebRTC | WebRTC is a combination of standards, protocols, and JavaScript and HTML5 APIs that enables real-time communications. |
| Z-Wave | Z-Wave is a wireless communications protocol used primarily for residential and commercial building automation. |
| ZigBee | ZigBee is a wireless technology that uses the packet-based radio protocol intended for low-cost, battery-operated devices in industrial settings. |

Table 2.2 – Data transmission protocols

The block architecture diagram shown in Figure 2.3 depicts the data flow between various edge-related components. Representative latency times between the different layers are shown in milliseconds at the top:

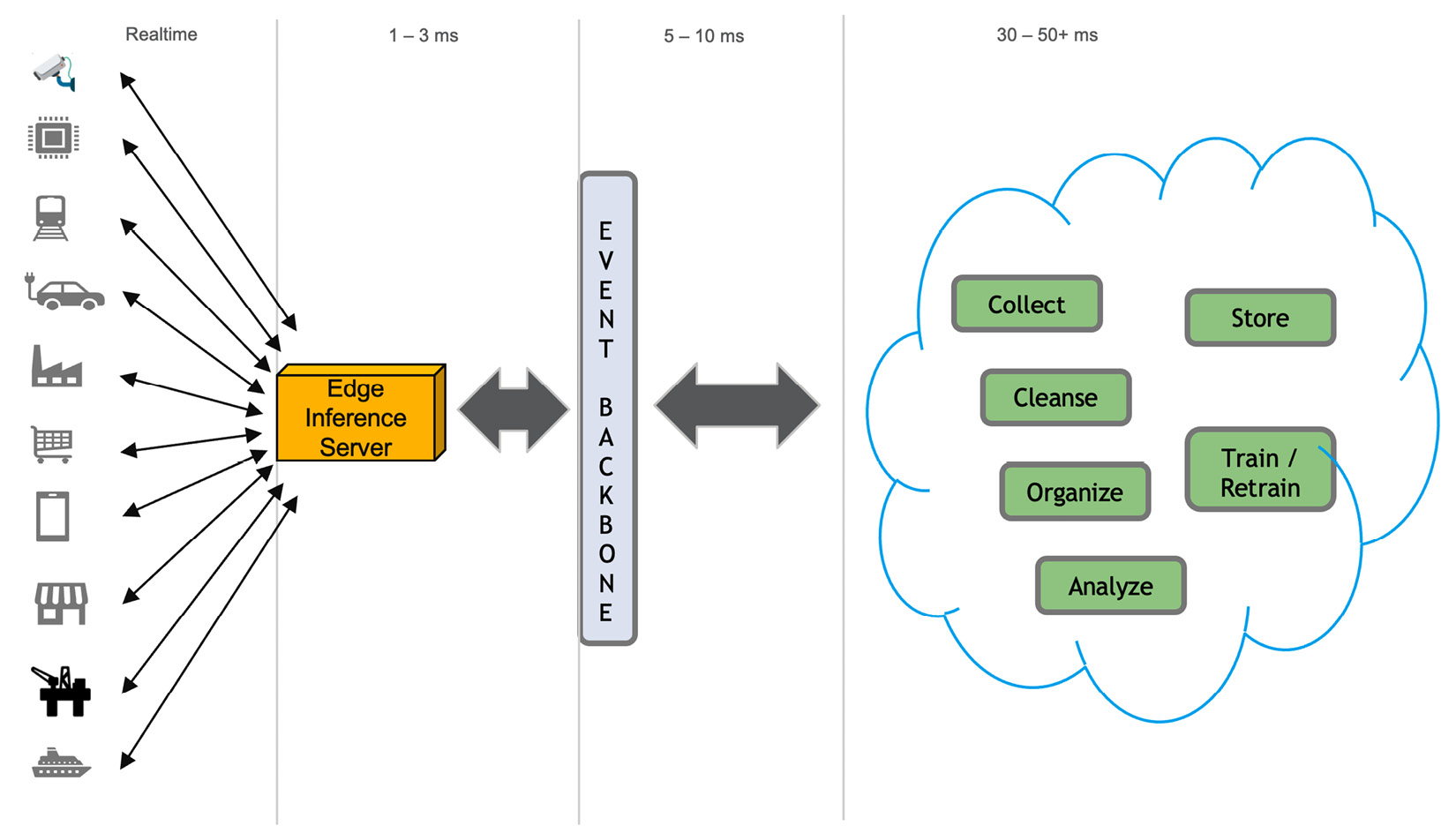


Figure 2.3 – A representative data flow in edge computing

The software stack in an edge solution architecture, as mentioned earlier, will vary depending on the use case, and edge use cases vary based on industry scenarios. No matter the industry, an edge computing topology involves a combination of hardware devices and software products. At the far edge, if possible, devices would run a containerized inference model in edge-native mode. These could be vision models, audio classification models, or sensory processing models. If not, the data would be sent to an inference edge server. Non-visual data, such as telemetry data and events, is typically sent to the communication layer known as the event backbone, which then would be routed to the appropriate destination. Software capable of training or retraining AI/ML models, including messaging, data, and AI-related middleware, could be aggregated, cleansed, and analyzed or stored in the next layer with more compute and storage.

As noted in [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013), containers are not the only edge-native approach for isolating workloads, including inference models. Enterprises may use **virtual machines** (**VMs**), serverless functions, or even **web assembly** (**Wasm**), depending on the code base purpose or execution environment. Regardless of the chosen approach, proper automation should be employed to ensure isolation is maintained.

# Architectural decisions

Solution architects are familiar with **architectural decisions** (**ADs**) when designing IT architectures. They are meant to capture key design issues and provide the rationale for selecting one of many alternatives in a solution. These decisions concern the design of a software system as a whole or one or more of its core components or connectors. They are made after considering various alternatives within a given aspect or domain.

Similarly, there are many facets that solution architects must decide on when designing edge solutions. It could be choosing the most appropriate transfer protocol to get data from a particular device, the type and location of storage, the use of private 5G versus public 5G, or the size and model of the edge server/cluster. These and other aspects must be considered before settling on a solution architecture.

# Grouping edge ADs

Components in an edge architecture are broadly grouped into four domains that map to deployment areas. This is where solution architects have to make architectural design decisions. Figure 2.4 shows the four component domains:

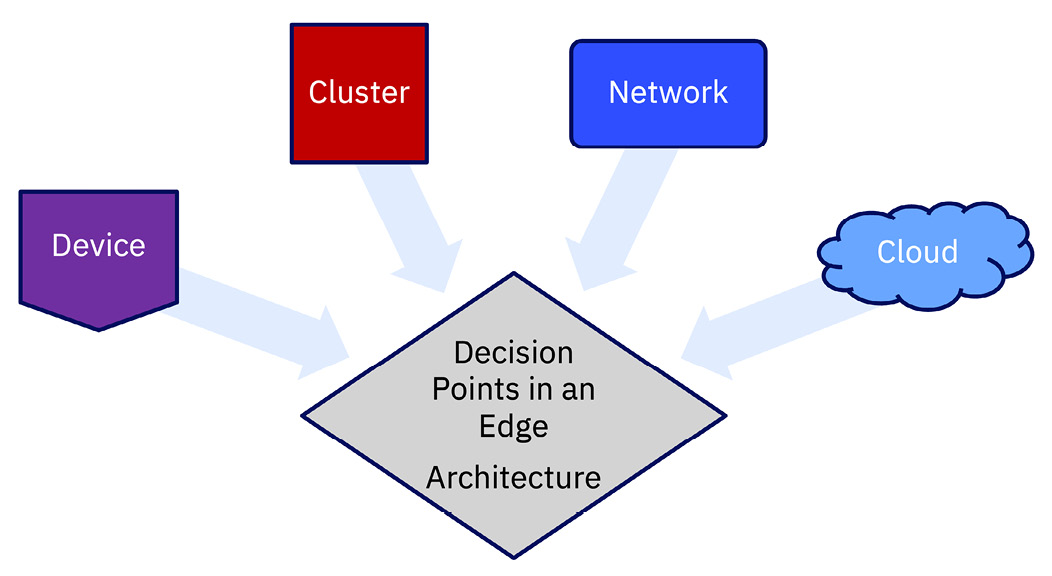


Figure 2.4 – The four decision points affecting edge architecture

Each of these domains has its own nuances and is equally important. And there are sub-domains within them, as shown in Table 2.3:

|  |  |
| --- | --- |
| **Component area** | **Description and sub-domains** |
| Cloud and data | Cloud deployment model.  The nature of the business and compliance requirements will dictate whether to use a private, public, hybrid, or distributed cloud. Additionally, data storage decisions will also determine data store type and size. |
| Network | The most important component.  In choosing the most optimal network design, architects have to weigh cost and availability and the nature of the business. There is 4G, Wi-Fi, 5G public, or 5G private. Now there are even different types of **RAN** (**radio access network**) to choose from.  That said, at times there won’t be any network connectivity, so one has to plan for disconnected operations, which is covered in later chapters. |
| Edge server/cluster | Role and location of edge server or cluster.  This determines not only the size but also the number of edge clusters/servers to deploy in order to support various devices. |
| Edge device | Device type and form factor.  This determines the type of applications that can be deployed on a given device and the data transfer protocol to use. |

Table 2.3 – Edge computing components

Solution architects should pay special attention to networking because of the recent advances in network technology, especially with the advent of 5G, which introduced, among other features, **software-defined networking** (**SDN**). While it is not explicitly called out, there is a cost dimension to all these ADs that could affect the final architecture.

As mentioned earlier, each of these four domains has sub-domains and decision points that can overwhelm solution architects who design end-to-end edge solutions. The crux of any AD is to justify the choice of a particular component. We have listed some of the more common ADs in the following section with the knowledge that not every decision point would be relevant to every solution architecture. Again, we go from right to left – that is, from the cloud to the devices.

# Cloud

The standard cloud deployment models are well-known – public cloud, private cloud, and hybrid cloud. We would be remiss to not mention the new distributed deployment model, which is described in [Chapter 5](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_05.xhtml#_idTextAnchor091). The cloud deployment model choice depends on the industry for which the edge solution is designed. As an example, the banking industry has stringent requirements on where data can be stored and even how it is stored. With such requirements in mind, the only option then is to use a private cloud.

|  |  |  |
| --- | --- | --- |
| **AD #** | **Requirement** | **Factors affecting decision** |
| C001 | Cloud deployment model | A private, public, or hybrid cloud model decision must be made based on bandwidth, latency, and cost. Always consider the optimal use between a physical data center, an on-premises cloud, a public cloud, or an acceptable combination. |
| Data storage | No matter which cloud model is chosen, the next decision point in edge computing is about data. Architects must consider where to store different types of data. Enterprises might agree to store generic data in the cloud, but business-sensitive data may need to be stored on-premises and be encrypted.  The other key aspect is data sovereignty, which is covered in [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110). Sometimes, it is mandated by regulations, and in other cases, it is desirable because of the social or political environment. |
| C002 | Cloud resource usage | The cloud offers unlimited resources when it comes to compute, data, and storage. Hence, it is more suited to build, train, and retrain ML models in the cloud rather than at the edge. |
| Data storage | Since there is so much data generated, the question of the use and storage of data and other cloud resources must be considered. It affects the deployment of ML applications and also the training or retraining of the ML models. |

Table 2.4 – Cloud-related ADs

# Network

Enterprises have the choice between private networks that are highly secure, single-tenant networks, and multi-tenant carrier networks offered by telcos. With 5G technology, we have seen a rise in private wireless networks. In choosing the type of network, solution architects must take into account the use case requirements, location, and cost. To add to the complexity, there are two types of networks – overlay and underlay.

The overlay network is a virtual network that operates on top of a physical infrastructure, the underlay. This abstraction model has opened new avenues of efficient network traffic routing. These new avenues affect both north-south and east-west traffic flows.

*REMINDER*

*North-south network traffic is the data flow to and from the cloud or data center. East-west network traffic is the data flow between deployed applications, amongst the components of applications, and to other cloud services.*

Figure 2.5 shows, at a very high level, network traffic flow patterns. Prior to virtualization becoming mainstream, most network traffic would be in the north-south direction. As more applications moved to the cloud, combined with the proliferation of edge data centers and edge locations, the amount of east-west network traffic within the facility and intra-facility has outpaced north-south network traffic:

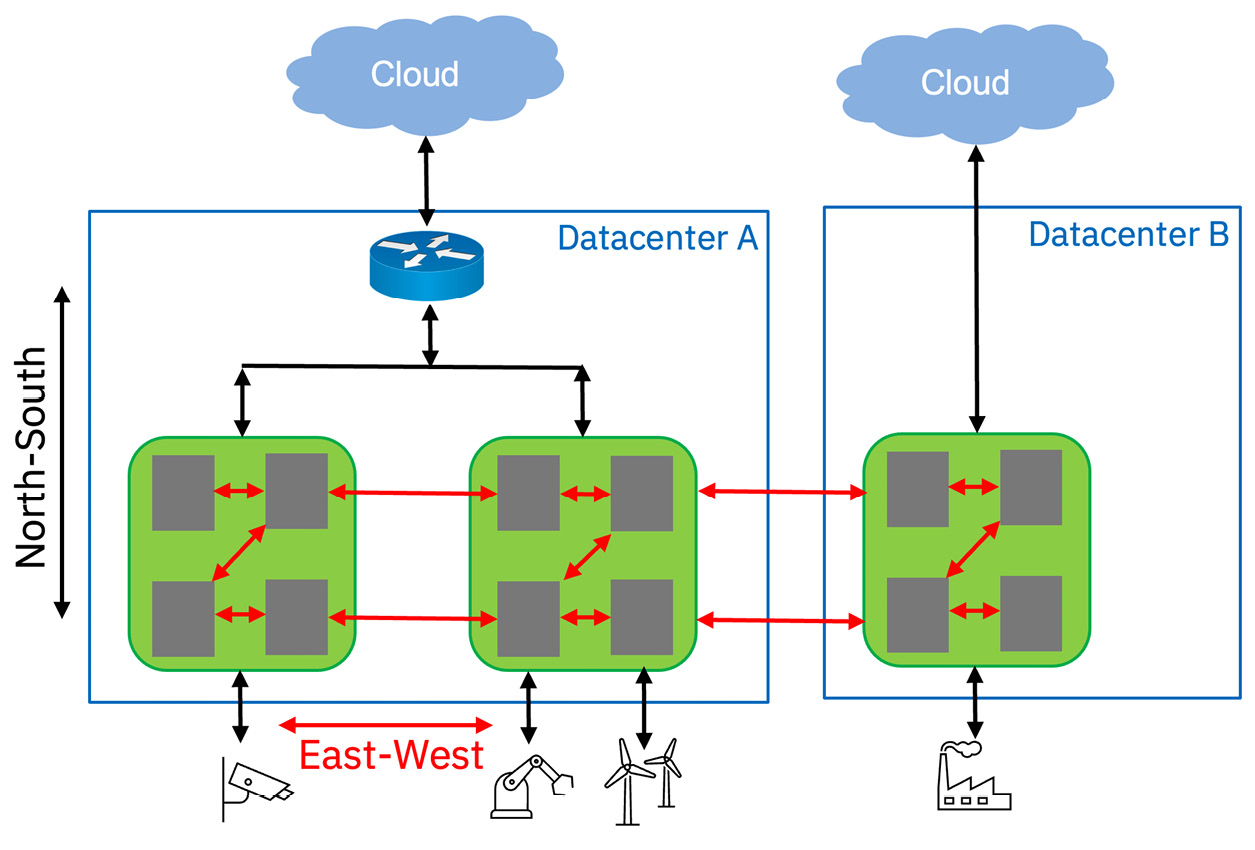


Figure 2.5 – Network traffic flow

Solution and network architects must pay close attention to southbound access interfaces of the edge computing domain because they impact the network access points and also affect the topology of the edge computing platform. Because data flow is multi-dimensional, two aspects should be considered when designing the southbound traffic management: security and futurizing the sensor network.

|  |  |  |
| --- | --- | --- |
| **AD #** | **Requirement** | **Factors affecting decision** |
| N001 | Access termination technology | Coverage area, terrain, device age, and support for legacy interfaces affect this requirement.  Whether it is tens of meters or many miles, the size of coverage area will determine whether to use 4G or 5G, a public or a private network, or even Wi-Fi.  The terrain or facility construction also affects the choices. If the distance is less than 50 meters and the 4G or 5G spectrum is not available for some reason, then Wi-Fi might be the only access protocol option.  While accounting for constrained computing form factor, low power, disconnected operations, and other networking challenges, architects must be aware that many legacy devices do not support the new cellular protocols. Wi-Fi is more prevalent, and sometimes, even that is not widely deployed. This leads to a closer look at gateway devices. |
| N002 | Edge gateway protocol | Establishing a unified stream of highly heterogeneous interfaces supporting the edge devices/sensors is critical in edge computing architectures. REST, AMQP, XMPP, MQTT, and CoAP are some very device-specific protocols out there. Which one would be most efficient in a particular scenario is something solution architects must decide. |
| Data transformation | Different data types generated at the edge call for different protocols and aggregation requiring data to be transformed before it can be consumed by the edge computing platform. Again, something the solution architect must decide is whether and where that should be done. |
| N003 | Edge network domain | To design an edge solution that is efficient and cost-effective requires the application domain to adopt two things – an API-based control plane, **software-defined network** (**SDN**) or a “flat” overlay network. Additionally, this large-scale distributed application architecture must be secured while operating on constrained resources.  Edge solutions can use many microservices for different functions, from visual analytics to heat sensing. An overlay network could be used for communication between these microservices.  Which deployment domain is best suited is a question that architects must answer – Kubernetes, **function-as-a-service** (**FaaS**), or maybe even serverless? |

Table 2.5 – Network-related ADs

# Server/cluster

As noted throughout the book, not all edge devices are created equal, especially when it comes to compute and storage. When IoT-type devices such as sensors, gauges, and actuators that do not have any compute or storage are part of an edge computing solution, an edge server or edge cluster is inserted and takes on the role of aggregating the data. In some cases, it is not viable to run a large ML application on the edge device, in which case that application would have to be deployed on the nearest edge server/cluster.

|  |  |  |
| --- | --- | --- |
| **AD #** | **Requirement** | **Factors affecting decision** |
| S001 | Deployment of edge servers/clusters | The use case will determine whether an edge server/cluster is required, and if so, based on the application to be hosted and the data generated, the server specification will be determined.  For example, in a retail scenario, cameras, inventory scanners, point-of-sale (POS) systems, and so on deployed in a large department store would have to be managed and monitored. Depending on customer traffic, sales volume, and store inventory, the solution architect would have to decide on the number and size of edge servers to configure. |
| S002 | Edge hub location | Deciding where to place the edge hub – on-premises or in the cloud – is another key decision point. Again, it might depend on the industry use case. Typically, one edge hub is enough, but depending on the deployment topology, multiple edge hubs may be needed.  For example, in healthcare and banking, industry regulations and/or government data sovereignty requirements will influence the edge hub location decision. |

Table 2.6 – Compute-related ADs

# Device

We have alluded to the fact that there are many different types of edge devices – audio devices, indoor and outdoor video cameras, light and motion sensors, heat sensors, vibration monitors, pressure gauges, and other telemetry devices. An edge solution topology could have a few of these edge devices or hundreds of them. In Table 2.7, we only focus on edge devices that are equipped with some computing power and storage capacity. From a data perspective, edge solutions should support one or more of the most common data transfer protocols that were listed in Table 2.2.

|  |  |  |
| --- | --- | --- |
| **AD #** | **Requirement** | **Factors affecting decision** |
| D001 | Edge device registration | To be part of an edge solution, all edge devices that have some amount of compute and storage capacity must be registered with an edge hub as edge nodes. This helps in management and monitoring. The new MQTT Sparkplug protocol includes handling the registration of devices, thus streamlining the process. |
| D002 | Edge device data | From streaming video to telemetry data, the big question is: can the data generated by the devices be used as is, or does it need to be aggregated or cleansed? Data transformation was mentioned in one of the edge network ADs.  With AI and ML applications deployed on devices, this decision about the handling of data from these devices becomes critical. |
| D003 | Store edge data | In one of the edge cluster ADs, it was mentioned that the decision to store or not store data depends on industry and governmental regulations and auditability reasons.  Often, all data from edge devices may not be that useful and probably does not need to be stored. If there is a choice, the solution architect might decide not to store that data because not only does transmitting all the data take time and money, but also, even data storage comes at a price. |
| Data security | Data security is described in the AD on security. Suffice it to say that data should be encrypted when stored and when transmitted. |

Table 2.7 – Edge device-related ADs

Lastly, we must mention security, specifically edge network security, as an important AD. Security is undoubtedly a key NFR but here, we talk about addressing edge network security. While solution architects should be aware of **zero trust network access** (**ZTNA**) and **secure access service edge** (**SASE**), they must take a holistic approach toward security covering all aspects of the solution, from the cloud to the network to the devices and the applications that run on those devices. ZTNA, SASE and Edge security in general, is covered in great detail in [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110).

|  |  |  |
| --- | --- | --- |
| **AD #** | **Requirement** | **Factors affecting decision** |
| K001 | Edge security | The architect must weigh the cost and challenges of deploying a distributed edge hub topology versus a centralized topology. In the former, a local hub controls a relatively small set of devices – for example, in a store. A centralized topology is where a large edge hub controls a multitude of devices in many remote locations. This is more cost-effective but a riskier design from a security perspective.  There is a security feature called **perfect forward secrecy** (**PFS**) that protects far-edge devices when transmitting data. But data encryption technology should be used so that all data is protected – data in transit and data at rest. |

Table 2.8 – Security-related AD

Justifying the decisions made when designing any architecture is important for any solution architect. Many ADs were described in this section, but we are certain there will be more decisions to be made. The important thing is to document all the alternatives, the decision made, and the reason behind it.

# Summary

In this chapter, we discussed the main components of an edge architecture. You learned that what makes edge architecture rather unique is the inclusion and heterogeneity of the plethora of edge devices. The typical functional requirements are few, but they are relevant and can be applied to various use cases across a whole lot of industries. We also discussed the different NFRs and their scope.

Lastly, there are many design decisions that solution architects have to make that could be overwhelming. Some of the common ADs were provided more as a starter set. We alluded to the complexity of architecting an edge solution because there are software and hardware components to account for, especially in edge-native designs. This should get architects thinking of the different aspects and domains. In the next three chapters, we will dive into the various architectures from basic to complex and provide recommended practices.