**Part 3: Related Considerations and Concluding Thoughts**

The final six chapters in the book apply what was described in Part 1 and formalized in Part 2. These chapters give guidance on when and why to use an approach and when to avoid it. The experience condensed here will give architects the ability to innovate and safely modify the archetypes to fit their specific requirements. The chapters survey many different topics and consider how they apply to edge computing architectures, but these should be taken only as an initial overview, and a springboard to more in-depth research and thinking by the reader, not a comprehensive introduction to the topics.

This part has the following chapters:

* [*Chapter 6*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110)*, Data Has Weight and Inertia*
* [*Chapter 7*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_07.xhtml#_idTextAnchor131)*, Automate to Achieve Scale*
* [*Chapter 8*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_08.xhtml#_idTextAnchor152)*, Monitoring and Observability*
* *Chapter 9, Connect Judiciously but Thoughtlessly*
* [*Chapter 10*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_10.xhtml#_idTextAnchor190)*, Open Source Software Can Benefit You*
* [*Chapter 11*](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_11.xhtml#_idTextAnchor208)*, Recommendations and Best Practices*

# 6

# Data Has Weight and Inertia

Whether audio, visual, sensory, or telemetry, edge computing is all about data. Whether the data is processed or otherwise transformed at the source where it is generated, stored for later analysis, or used to build models in the future, data is the lifeblood of edge computing. Since data is so valuable, organizations need to store and analyze it before determining relevancy. However, it needs to be done both securely and while actively protecting it in order to meet regulatory compliance regulations. This chapter will discuss considerations about data in its various states. Decisions you make about data will affect where data is primarily stored (data at rest) and how it is stored. Thus, how it is accessed or transferred (data in motion) becomes a secondary consideration, and tracking how it is used and transformed (data in processing) is critical to having confidence in its trustworthiness.

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

* Data encryption
* Data storage and management
* Using data to build machine learning models
* Connectivity and the data plane

# Suggested pre-reading material

* Security and Privacy by Design or Default (SPbD) principles: <https://securecontrolsframework.com/domains-principles/>
* Data Confidence Fabric (Trust Fabric): <https://www.lfedge.org/projects/alvarium/>
* Edge Computing Security: It Starts With Solid Device Identity and Attestation (Medium, Robert Andres): <https://medium.com/the-edge-of-things/edge-computing-security-it-starts-with-solid-device-identity-and-attestation-78cace26ed92>
* Fully Homomorphic Encryption (FHE): <https://research.ibm.com/blog/fhe-cloud-security-hE4cloud>
* What is Artificial Intelligence (AI)?: [https://www.ibm.com/topics/artificial-intelligence](https://www.ibm.com/topics/artificial-intelligence%0D)
* What are Foundation Models?: <https://research.ibm.com/blog/what-are-foundation-models>

# Data encryption

In this section, we will discuss the driving factors for encrypting data, emerging techniques for protecting data while increasing its value, and how to have confidence that data has not been unintentionally or surreptitiously modified. You will learn about protecting data during processing without decrypting. By the end of the section, you should be aware of which techniques work well on the edge.

## Motivations for encrypting data

Data is encrypted so that unintended parties cannot read and ultimately use the data. Since the data does not belong to those third parties, nor do they have permission to use it, they should not be able to benefit from the usage of that data – also known as the concept of data ownership. It’s about the principle of ownership and the data owner being able to determine to what ends the data and derived value should be put – data sovereignty. Therefore, encryption of data is a method for the data owner to prevent, and to show that they are actively attempting to prevent, unauthorized access to their data and its derived value – to ensure data privacy.

It is similar in intent and purpose to a locked gate and fence around a field. The locked gate is not impervious or impenetrable. Unauthorized visitors could potentially bypass the gate or force it open, for example, but its presence is a clear indication that someone is attempting to prevent access to the field. Circumventing that gate is an act that acknowledges the presence of that barrier and is an overt expression of an intent to bypass that barrier in order to gain entry to the field. Likewise, the presence of encrypted data is an indication that persons without the key are not intended to access the data, and bypassing that encryption is an action showing the intent to use the data despite a lack of permission.

To ensure both that third parties cannot read the data and that they cannot eavesdrop on the data while it is transiting networks or being manipulated in working memory, the data must not only stay encrypted but also be processed in protected memory and be transferred over encrypted and isolated networks – a concept known as data integrity. Additionally, (Perfect) **Forward Secrecy** is a technique that provides additional protections to data in motion by using a new session key to encrypt information each time. This ensures that even if one session’s encryption between two parties is compromised, all other sessions between them are still protected. In a perfect world, methods such as those provided by Fully Homomorphic Encryption would be available to process that data and derive value from it without ever decrypting it.

When it comes to edge computing, and especially at the user edge, the expense of encrypting and decrypting data may be higher than at any other location due to less capable hardware driving higher data transmission latencies and slower processing. Additionally, decision-makers may be under the impression that there is no pressing need for their data to be encrypted. To help mitigate this misconception, and to ensure that potential vulnerabilities are minimized, it is important that **User Experience** (**UX**) and software development teams be trained in the concepts behind **Security and Privacy by Design** (**SPbD**) and Security and Privacy by Default.

But just encrypting data at rest and data in motion may not be sufficient.

## Protecting data without making it difficult to use

Protecting data at rest and in transit can be covered by encrypting the data itself, made more secure by also encrypting transmissions. But this still leaves a potential gap or vulnerability during processing, since the data needs to be decrypted before performing operations on it. But what if the data didn’t need to be decrypted to perform computations such as multiplication and addition?

The promise of **Fully Homomorphic Encryption** (**FHE**) is efficient algorithms that can produce encrypted results of calculations that only the owner could subsequently decrypt using the encrypted data as input. This seemingly impossible approach protects data while in processing since the data is never decrypted and the key used to originally encrypt it is not known by the FHE algorithms.

When first developed, FHE tools were needed for financial privacy. They were also thought of as allowing third parties to provide services to perform operations on sensitive and private data without violating confidentiality. However, a new case for FHE has emerged with the practice of training **Machine Learning** (**ML**) models on data. By using FHE, which supports floating point math operations, models can be trained on data that cannot be directly read, and that only the data owner could then subsequently decrypt and use.

A drawback to the use of FHE, especially on the edge, is the amount of processing required, and the amount of time it may take to process. This means that using FHE for near real-time calculations will likely require cloud-based resources. But if the use case isn’t as time-critical, edge computing may be sufficient for these techniques.

So, making data difficult, if not impossible, for an untrusted party to read and use is a great start. But how do you know whether the data has been manipulated by others?

## Ensuring that data modifications are noticeable

Protecting data is not just about preventing others from reading and using the data. It’s also about knowing where the data came from (provenance), whether it has been modified since it was generated, by whom, and whether the parties in the chain of custody can be trusted (lineage). One mechanism to collect and report on all that metadata is a **Data Confidence Fabric** (**DCF**).

DCFs can generate data confidence scoring values to indicate how likely it is that a piece of data came from the indicated origin, and assurance about which subsequent systems and applications it has traversed, thus giving a degree of certainty that the documented chain of ownership has been correctly captured. DCFs can also indicate the likelihood of whether and when data has been modified, based on a collected history of annotations written to immutable storage by trust insertion technologies and thus associating the history of the data record with the record itself.

Consequently, data with a high confidence score is likely to be trustworthy, and data with lower scores is either known to be untrustworthy or of unknown provenance and lineage. In any event, only data with high scores is unlikely to have been manipulated and is thus objectively trustworthy. The benefits of using a DCF include being able to trust data sourced from a third party, and knowing how data has traveled, along with any modifications along that journey.

Since data from sensors, actuators, and other IoT sources are typically generated at the edge, this is the primary location to begin implementing a DCF-based solution. To do so, base your planning on four considerations:

* Choose a trusted DCF solution (such as Project Alvarium, <https://alvarium.org/>) used by organizations in the same type of business, vertical, or industry.
* Ensure the IoT devices that you use can support a hardware-based root of trust or similar technology that definitively identifies the device. Build in a device attestation strategy, utilizing a chain of certificates to confirm both the integrity and trustworthiness of every device.
* Identify all locations where you need to integrate trust insertion technologies.
* Deploy trust insertion only in configurations with reliable, stable, and constant network connectivity to the DCF.

A positive consequence of using DCFs is that compliance with privacy standards becomes much easier due to the proof it provides of what has and has not been done to the data. In the next section, we will cover solutions to manage your data.

# Data storage and management

In this section, we cover data storage considerations, viable alternatives to data retention, and data catalogs/governance/policy enforcement in edge computing. You will learn about considerations to weigh when implementing data management solutions. By the end of the section, you should be able to articulate what data management processes you will follow and what tools you plan to use to enforce the processes and policies.

## Strategies for defining and enforcing data policies

Over the last several years, we’ve seen governments around the world propose and implement legislation aimed at protecting private data, beginning with the EU’s **General Data Privacy Regulation** (**GDPR**). This type of legislation has encouraged organizations to be more intentional about what data they collect, how they store, use, and share it, and for how long and where it is retained.

For organizations to comply with these privacy regulations, there are three foundational solution components that should be used. These include a data catalog, data policy sets comprised of rules, and data policy enforcement mechanisms to ensure that those rules are followed without exception. Let’s explore each one of these to understand the scope of responsibility of each of these data management solutions, and briefly discuss how edge computing may affect deployment considerations.

Data catalogs are solutions that collect descriptions of collected data, data types, sources, data purposes, collection timestamps, access rules, storage formats, and other metadata related to answering the questions “What data is being collected, from whom, and why”? There are at least three considerations that will affect what data catalog solutions you choose to employ and where they should be deployed:

* **Consideration 1**: These catalogs are typically accessed by data scientists and others doing research, analysis, and business intelligence reporting and summarizing, which means that they are typically centrally located
* **Consideration 2**: They may also be accessed by automated systems querying and enforcing data policies, which means that they should be located near data storage, especially if connectivity is not always available
* **Consideration 3**: In the case of distributed data storage strategies where the data is kept as close to the source as practical, this implies that a description of that data should accompany the storage or querying capability

Data policy sets and rules are a way of encoding an institution’s official procedures for data access, usage, retention, and movement in a machine-readable format so they can then be enforced. The policy sets and rules also serve as proof of intent that the organization knows data should be protected and is actively striving to do so.

These data policy rulesets, like the catalog, need to be centrally located and accessed but also need to be available in offline scenarios. For this reason, we recommend a central ruleset storage location with caching or distribution to edge locations. Or, alternatively, using a solution that generates rules as code and distributing the resulting application to edge locations.

To have a complete data governance solution, you not only need to know what the data is, its provenance and lineage, and how it should be used, but you also must enforce those rulesets. This requires that data reads, writes, and deletes must be mediated by logic that understands the complete context of the transaction: who is requesting it, for what purpose, in what geographic location, with what consuming application, and for what (data) purpose. That logic must understand the data, its metadata, the request context, and applicable data policy rulesets.

In light of the above discussion of data governance rules, the resulting policies constitute the maximum length of time that personal or private data may be kept. The minimums are more flexible and can be based on the cost of data retention, the amount of available storage, and other practical considerations. An exception to those bounds may be data that your organization is legally required to retain.

With the restrictions that privacy legislation requires come limitations on how the data can be used. However, there are some techniques available that would provide similar outcomes to using personal data without violating the data subject’s privacy. Those data usage options will be discussed next.

## Usage options ranging from real to synthetic data

Many years ago, architects and developers working at a top website were reviewing the web server infrastructure access logs to determine what statistics and site/section/page usage data to surface. Product managers would need to learn how visitors interacted with the site, how long they lingered on various pages and sections, and what pages they likely visited accidentally, or were navigational dead-ends. Technical staff would need to see which pages were generating errors for visitors, and which were too large or taking too long to load. Information architects and SEO staff would evaluate the URLs, seeing how the site was indexed and read by machines (site crawlers, indexing routines, and screen readers), and ensuring that all content was translated equally into all supported languages.

Every one of those persons needed insights from the log files, but some of the information contained therein was considered personal data and could reasonably be used to positively identify a specific person (which would violate their expectations of privacy). Except for the singular case of resolving technical support issues, none of the staff needed access to information classified as Restricted Data (IP address, location data, date and time of request, authentication, usernames, and even some referral information). So the challenge for the architects was determining the most efficient method of surfacing the required information without revealing the sensitive data. Here are some heuristics we developed to ensure that we didn’t inadvertently break any rules, violate user privacy, or even display or use information that we shouldn’t.

The first and most obvious option is not to store or otherwise persist any data if it is not going to be used or should not be used. Next would be retaining data that can be stored, but should not be viewable by all persons with access. Third would be masking any sensitive information when displaying records to those authorized to view portions of the data. Fourth would be preventing access to the data by unauthorized parties. And fifth would be removing the data when its retention period had expired or the data subject had requested its removal. Let’s examine each of these rules of thumb in turn:

* Keep only the data you need. The next topic to be covered in this chapter section will be data retention strategies for the edge. This topic will address how to categorize your data and then determine rules for each category.
* Restrict access to your data. This requires data policies and enforcement solutions to be implemented as discussed in the preceding Data storage and management section.
* Data access rules should be granular. The solutions that display data should have filtering solutions in place that know each viewer and what data they are or are not authorized to view. The presentation of sensitive data should then be blocked or masked when it should not be viewed in that context.
* Actively prevent some data access. This may mean having your data access solution return an error message when access should be prevented, or blocking access at a higher level in the application stack.
* Prune your data. Don’t assume that data is immutable, given that some records will need to be removed at a time short of the data retention limit. Always ensure that records marked for removal can either be removed in an acceptable amount of time or that they can be dereferenced and not otherwise retrieved.

However, there are times when a change in the way you use your data can remove the complexity the preceding rules of thumb require. If you were able to modify the data that you collect before using and storing it so that all restricted data is replaced with equivalent fictional data that would give statistically equivalent results in the aggregate, then many data access and restriction rules could potentially be relaxed. This would also result in your data management overhead being proportionately reduced. This type of replacement data is termed synthetic data.

Synthetic data can be used to replace usage data on the whole but not individual personal or account records. It may also have the potential to reduce the overall amount of data that needs to be retained. However, you will still need to have active data retention processes and governance in place for your data as a whole. Let’s look at those considerations now.

## Rules of thumb for retaining data, or not

A connected car expert recently said that when roughly half of the billion cars on the road today are comprised of connected cars, they will generate 30 zettabytes of data a year. How much of that data is analyzed, or even needed?

You can classify data in many ways, depending on your vocation, industry, or practices. In data science, data is classified based on what is known about it:

* **Context-based**: When aspects of its creation are known and affect the data
* **Content-based**: When the contents of the data affect the category it is assigned to
* **User-based**: When a human labeling the data determines its usage

Data comes in two forms: structured and unstructured. Structured data is organized to fit a data schema or model, and unstructured data is not. Management of data in storage systems categorizes data by its retention needs: short-term data, long-term data, and useless data. Compliance and security professionals also label (classify) data as one of the following five types: restricted, confidential, internal-only, private, and public.

Unlike data science classification, data on the regional and user edge falls into one of two types: system data and user data. System, or operational, data can be used to configure and operate architectural elements. System data is structured and persisted as needed. User, or generated, data is created or captured by edge devices. User data can be structured (temperature readings) or unstructured (acoustic monitoring). See Figure 6.1 for strategies on deciding how to manage edge data.

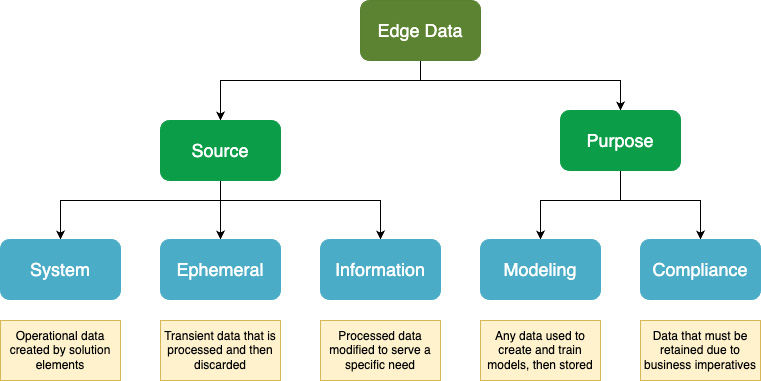


Figure 6.1 - Thinking about data retention at the edge by data origin and usage

Based on the preceding chart, if data is not used to configure or operate solutions, is not processed for a business intelligence purpose, and is not required to be retained, then you might not need to store the data after initial processing is complete.

Next, let’s discuss how edge data can be used for model generation and training, and strategies for model-based inferencing on the user and regional edges.

# Using data to build machine learning (ML) models

In this section, you will read about techniques for efficient (re)training, inferencing, deployment, and customizing ML models. We will also discuss what has prevented high levels of demand from being met, and what is being done to resolve that.

Before we dive into the topic, it’s appropriate to briefly review **Artificial Intelligence** (**AI**) and what distinguishes it from ML and **Deep Learning** (**DL**). IBM describes AI as “leverage[ing] computers and machines to mimic the problem-solving and decision-making capabilities of the human mind.” See “What is Artificial Intelligence (AI)?” in the Suggested pre-reading material section at the beginning of the chapter for a deeper explanation and some background history. ML is a branch of AI and a component of the field of data science that uses data and algorithms to imitate the way we believe human brains acquire knowledge. ML typically uses structured or labeled data and human intervention (supervised learning) to learn and make predictions (infer a result). DL consumes large amounts of unstructured data and may not need supervised learning to create useful models. Both ML and DL models typically cover a single domain of knowledge and are thus highly specific to one type of task.

## The promise of foundation models

Three factors that delay or prevent widespread adoption of ML-powered automation include long training times and the cost, steep learning curves and limited expert availability, and big resource requirements using expensive **Graphical Processing Unit** (**GPU**) hardware for inferencing using the models. Every advance made that addresses those factors has the potential to make these tools less expensive to operate and thus more profitable to use, enabling employees to be more productive by accomplishing more tasks with greater accuracy in less time.

Foundation models, unlike ML and DL models, are trained on very large sets of data, which are usually unlabeled. The resulting models can then be fine-tuned for many different (but related) purposes, unlike single-purpose ML and DL models. This can make foundation models more efficient and quicker to initially build, but also to repurpose for other tasks.

The promise is that instead of groups of humans curating large amounts of source data and then spending weeks or longer training ML or DL models just to solve a specific problem, an enterprise can use fewer people to train a single foundation model and then fine-tune it to solve many problems. To shorten the process even further, an organization could start with an existing foundation model from a trusted source and only add the needed fine-tuning for its specific purpose. Thus, leveraging foundation models requires fewer persons and can reduce the time to value for a business from weeks to days.

The trade-off with foundation models is that they are so large that they need to be run in a cloud environment or large data center. Conversely, some ML models can be optimized to run on single machines or even highly constrained devices. Therefore, ML models could be deployable to the user edge, while DL models might be deployable to the service provider edge.

Additionally, using federated learning techniques allows multiple data sources (such as distributed data, for example) to be used to train a single deep learning model. This horizontal federated learning approach works by having each data source trained on its local data using a single shared model, and then sending the results back to the original model source (usually in the cloud). This can be more efficient and faster than sending the data to the source for training.

## How small and efficient can we make models?

While there is plenty of demand for the usage of ML to perform repetitive tasks and surface insights as close to where the data originates as possible, there are several factors preventing this. First is the **availability** of GPUs and other accelerators. Second is the **cost** of purchasing and operating that hardware. And third is the lack of general-purpose tools that will **customize** and train the ML models as close to where they are needed as is practical. Fortunately, teams are actively working to solve all of these challenges.

At the time of writing this chapter, a new article was just posted showing how MIT, IBM, and researchers in China just created a new technique named EfficientViT for semantic segmentation (object detection) that improves performance over traditional approaches nine-fold. They anticipate that this innovation will allow near real-time object detection locally with improved accuracy on constrained edge devices. Solutions like this one will affect both cost and availability at the edge by more efficiently utilizing existing hardware.

Just this year, it has become common to see single-purpose ML models no larger than a few hundred **megabytes** (**MBs**) in size created to run on inexpensive microcontrollers, which are not typically thought of as capable of running these workloads. Factories are now using inexpensive consumer-grade mobile phones, instead of specialized hardware, to run visual inspection tasks using ML models less than a **gigabyte** (**GB**) in size and optimized for the onboard GPU.

## Customizing existing models for each deployment

When determining the optimum parameters to consider when evaluating a model, you should consider the factors of size (as it impacts both storage and deployment speed), (re)training cost, operating cost, and the inference speed of the model when matching to the desired hardware it will run on. Your KPIs should be used to assign the appropriate weights to each category. For example, if your primary concern is operating cost and the speed of inferencing isn’t critical, then it’s acceptable to weigh operating efficiency much higher than performance.

Just as important are the types of models that the GPU or other accelerators support. If there is any incompatibility, the inferencing environment will likely fall back to using the CPU, which will incur a significant performance penalty. Even slight differences in how a model was created, including quantizing methods, could affect the performance, so careful attention to model creation parameters should be prioritized. Given the heterogeneous nature of hardware on the edge, it is likely that there will be incompatibilities.

Thirdly, choose a model deployment solution that can deliver model assets to the consuming edge node. This will both guarantee confirmation of artifact delivery to the endpoint and also allow the deployment solution to communicate attributes of the edge node including GPU type, available storage space, and computing power to the delivery pipeline so that accurate placement decisions can be made. This type of model deployment solution is a good example of a general-purpose platform, which is our next topic.

## Using general-purpose platforms rather than single-purpose applications

When decision-makers set out to use their data to build ML models (especially for edge-based operations), they are usually being tactical and attempting to either solve a single problem or a class of related problems. Starting the evaluation process with constraints like these limits the potential range of solutions being considered to a set of competing single-purpose applications. In fact, considering an application to solve the particular problem at hand is the typical knee-jerk reaction. Instead, we’d like to propose an alternative course of action.

On a recent community call discussing the Open Retail Reference Architecture (ORRA), I heard an insight delivered in an off-hand manner that got my attention. Kristen Call, senior industry advisor at Intel Corporation, while discussing the value of edge-based platforms, said: “Don’t buy a purpose-built solution. Your data is the solution, build on it.” Her point was that it is easy to find a single tool that solves a particular problem, but that is the end of the value that it provides. Each application installed consumes finite resources on edge computing nodes (devices and clusters) and must be carefully considered. A question each architect should ask is: “Does this tool provide value for more than just this one solution”?

Solutions should not be siloed, and potential derived insights should not be lost. But if a store were running a general-purpose edge computing platform that enabled sensor fusion, it could export insights and generate business rules dynamically in a way that provides value to both the store and the shopper in a manner that no single application could.

For example, if on one trip to your grocery store, you were to purchase three items – steak, corn, and beer – the platform could store that information and note that those three items were purchased together by a shopper. One insight could be that they might be preparing to cook out. Another could be recommendations that in similar situations, other shoppers also purchased napkins, single-use tableware, condiments, tablecloths, and dinner candles. A third would be matching to current store specials and promoting those related items. If the shopper had the store application installed on their phone, it could push those recommendations to the shopper when they make a shopping list or when they enter the store or approach the checkout.

The point of the preceding scenario is that none of those would be possible if the purchase data were stored in a single application. Surfacing of the data insights would not be possible without a general-purpose platform. Using the insights to create solutions is a lot simpler with the tools a platform provides when compared to the cost and maintenance of integrating single-purpose solutions – if their interfaces even enable that. Ultimately, a platform that enables innovation, especially if it provides a low-code or no-code interface, provides a flexible foundation while at the same time promoting efficient use of edge computing resources on-premises.

Since we’ve now mooted that your data is the solution, let’s move on to discussing how to optimally access that data while minimizing its movement.

# Connectivity and the data plane

In this section, we cover issues related to data at rest and in motion, specifically from the perspective of edge computing, data virtualization and the edge, and transparent progressive failover starting at the edge and moving cloudward. You will learn about various options available to edge computing architectures that will assist in automating data management, placement, and migration capabilities.

## Optimizing data availability without connectivity

Part and parcel of manipulating and storing data is the movement of data, which necessarily includes connectivity. But what if that connectivity is not available, is intermittent, or is slow with low throughput and high latency? How can you ensure that all local data is available to all services and applications that can access it even while remote data is not?

One technique to consider is using a data virtualization solution that can access local data in any format and allow SQL-like querying of structured and unstructured data. While that data could be persisted in an object store, it does not have to be. There are several solutions available that implement data virtualization and querying functionality like this.

Another technique would be to ensure that all functionality is available without an internet connection. In [Chapter 3](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057)’s Disconnected operation section, we touched on the AgriRegio Projekt’s “offline-first principle.” Let’s examine that in more detail and look at the impact on and considerations for data collection and transfer as a best practice approach. See Figure 6.2 for an overview of the farm deployment.

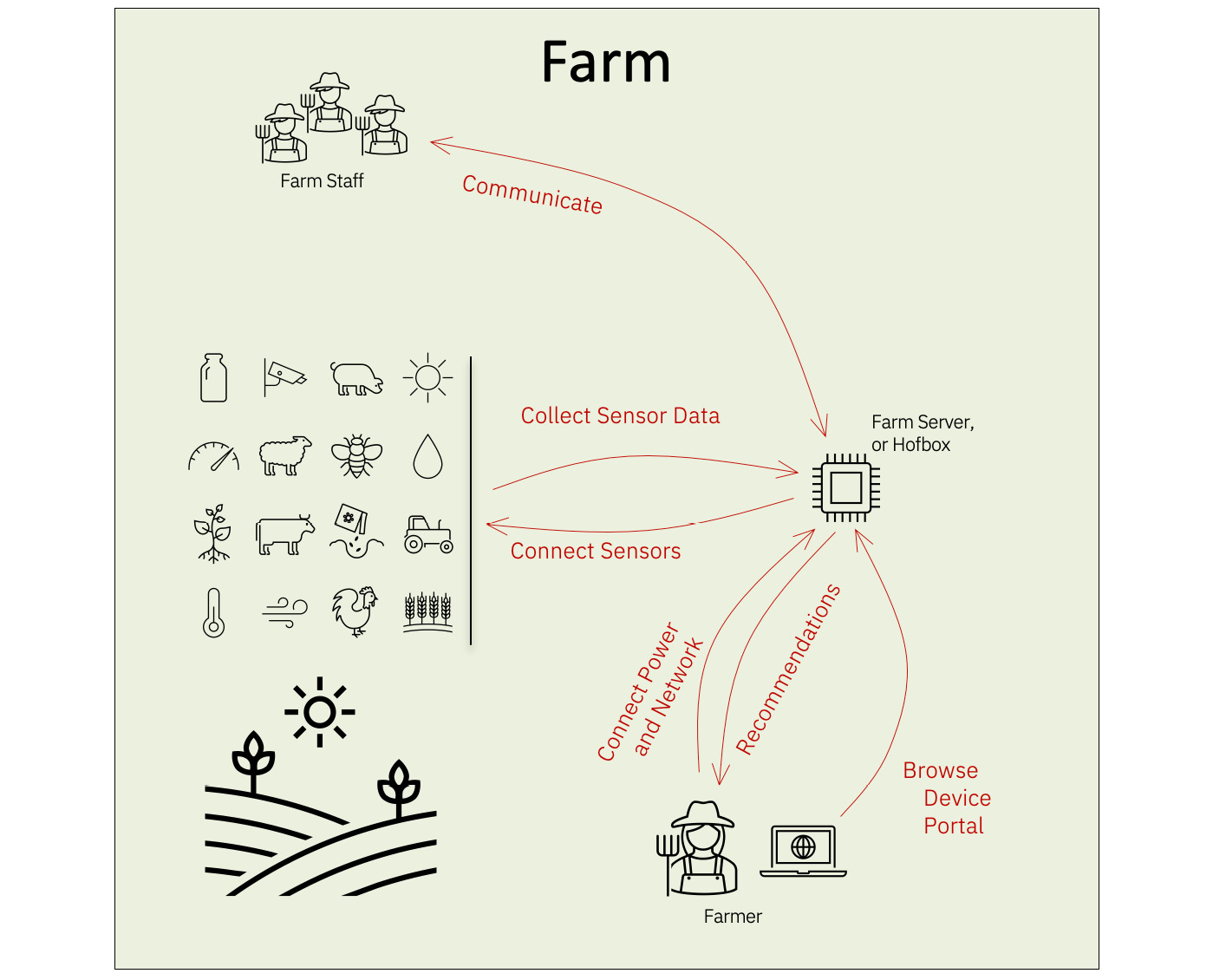


Figure 6.2 - How the farm box (Hofbox) manages data without the internet

On the typical small farm under consideration (less than five acres on average), there will be sensors deployed to measure moisture, sunlight, and water depth in a stream, capture images, and so on. These sensors will be placed semi-permanently (moved fewer than 13 times a year), permanently, or temporarily. They collect data measurements at rates ranging from once a minute to once a day. These sensors should be able to store or buffer that data for a limited amount of time – usually, not more than a week’s worth of data before potential data loss occurs.

The data that is captured is the property of the farm owner and should be controlled by the owner. The farm owner’s dwelling would contain an edge gateway server named the Hofbox that would be the central storage, processing, and reporting location for all collected data. We anticipate that data will be transferred from the sensors to the Hofbox in one of three manners: direct transmission from sensors to server by Wi-Fi or LoRa, indirect transmission by network mesh (Wi-Fi, LoRa, Bluetooth), or manual transfer by connecting a mobile device to the sensor with USB to retrieve the data and then subsequently connecting the mobile to the server with Wi-Fi. Of note in the preceding scenarios is that no internet connection of any kind is needed to collect and transmit the data.

The data management is provided by containerized applications running on the farm server, which may include object recognition and other visual analytics aided by ML models. An internet connection would only be needed to transmit collected data to a regional hub if the farmer opted in, to install new applications, and to update existing applications and re-trained models. That internet connection would only need to be active during the transmission period.

This approach is possible when following edge-native architectural principles, as discussed in [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013). For a more detailed discussion of thriving in situations with ambiguous connectivity, see [Chapter 9](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_09.xhtml#_idTextAnchor172).

## Aggregating data versus keeping it distributed

When IoT devices first began to be connected to the internet, cloud- and data center-based central aggregation and processing of the data was best practice. This was because the resulting solution could be highly available and scalable, and could be relatively inexpensive to operate. At the time, there was no capability for any reasonable processing to happen outside of the cloud or regional data centers, and cloud-native development was still in its infancy.

Now, almost a decade since that time, two advancements have completely altered the solution landscape. The first change is the ability to generate insights from data streams in stages, at the source and then, optionally, regionally before consuming the results and/or aggregating in the cloud (if needed). The second change is the ability to virtualize and thus query the data where it is generated without transferring to, or aggregating the data at, a central location. This ability to extract value and insights directly from the data source simplifies solutions, delivers results faster, and reduces expenses dramatically.

This edge computing data virtualization revolution began with the GaianDB open source project. Since then, it has seen several iterations and subsequent expansions and improvements. Other organizations have combined that approach with additive innovations such as ad hoc P2P network underlays and embedded device support.

We are now at the point where IoT routers and IoT hubs (see [Chapter 3](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057) for a detailed discussion on these elements) are now largely obsolete as data on fixed-function devices can be injected directly into a data virtualization solution at the source and immediately queried from any edge or cloud node. This is a drastic simplification over the former process of legacy protocol support, data ingestion into an IoT platform, data transformation, message queueing, transport, and finally, importing into a database to be queried. And yet, data virtualization has yet to be widely adopted, and there are few enterprise-grade solutions available that stretch from the cloud to the edge. We expect a sea change in this regard within the next few years as architects recommend, and CXOs embrace, this approach to upgrade, modernize, and automate existing solutions.

One other reason not to keep data distributed in the past was the need to aggregate data in a single location for training machine learning models. Using federated learning techniques to incrementally and independently train models has now eliminated another objection to keeping data distributed.

## Migrating data files automatically

As stated earlier in the chapter, when looking for solutions, prefer a general-purpose platform or tool over a single-purpose application (off-the-shelf or homegrown) since it will provide more flexibility with less maintenance. And if that general solution adheres to open or industry standards, so much the better. We’ve covered querying and streaming data records, both locally and remotely. Now it’s time to turn our attention to data files.

A typical challenge when accessing files locally on edge devices is determining when and how to archive the files, where to put them when archiving so that it does not take too long to retrieve them nor incur excessive or unbudgeted costs in doing so, and how to access them later without knowing deterministically where to look for them. Let’s examine those issues one at a time.

Automatically archiving files on a schedule when they pass a certain age threshold since creation, or when the filesystem volume passes a pre-specified percentage full, is a technique typically used. This approach can work when files are added in a predictable manner, provided that the process doesn’t fail silently. However, it is not adequate when new files are added between scheduled times. Most tools also do not intercept attempted writes and force a cleanup/archiving routine while pausing the write until the cleanup operation is complete. Conventional methods completely fail if the file is larger than the target storage volume.

Determining where to put the files is an issue when you have multiple potential destinations, such as hot, warm, and cold storage options, regional storage options, and multiple cloud vendors with variable pricing.

Knowing where files are currently located would require following a specified formula, using a global lookup mechanism, or trying each location sequentially while falling back when not found. Each one of these options can incur a latency penalty and require complex configuration and maintenance.

Thus, the best practice approach would be to use a general-purpose edge-native unified storage solution that stores files locally, regionally, and globally (example: Nexoedge <https://lfedge.org/projects/nexoedge/>). This solution would need to function as a proxy on the edge nodes and allow local file operations to act on the files no matter where they are currently stored. The solution would need to move files in the background as they are accessed more or less frequently, and as space, connectivity, and costs allow.

# Summary

In this chapter, we covered the states of data at rest, data in motion, and data in processing. You have learned about how, when, and why to encrypt data, including the ability to perform operations on the data without decrypting it using fully homomorphic encryption. You were introduced to the idea of data trustworthiness using a data confidence fabric, which can be implemented on the least powerful machines on the user edge where data is born.

You learned about data storage and management approaches, including data governance, policies, and enforcement, as well as ways to simplify management and lower costs with synthetic data. You were also introduced to a way of thinking about data retention on the edge that will lead to simplified decisions.

We also covered how to optimize machine learning models for the edge. This could include using foundation models to simplify and speed up training. And we touched on model federation. We discussed the benefit of general-purpose platforms over single-purpose applications.

Then, we talked about how connectivity affects data collection and usage. We gave an example of how to use edge-native architecture principles to ensure constant access to data by decision-makers. We showed how data does not need to be aggregated northward but can be queried as needed while keeping it distributed. We also touched on ways to migrate data and files automatically.

In the next chapter, we’ll discuss how to efficiently put these lessons into practice by automating (almost) everything.

# 7

# Automate to Achieve Scale

Deploying applications and models to hundreds and thousands of edge devices and edge clusters/nodes, then monitoring and managing all these things, is a challenge and not for the faint of heart. With the proliferation of these heterogeneous edge devices and the services required in an edge solution, it is almost impossible to manually deploy applications on these devices and manage them at scale, especially when it comes to networking components. The more prudent approach would be to introduce automation. In fact, it is an absolute necessity.

From monitoring the devices to updating the applications and models running on those devices, the goal of automation is to allow enterprises to build capabilities that can act and react with the solution components in a more secure and trusted manner with minimal human interaction. The other facet of automation is the ability to scale. An automated process should work for hundreds and thousands of devices whether it is to configure things or to roll things back. Like the digital journey or the AI journey, automation in the deployment of edge computing components is now trending more toward automation facilitated by open source communities such as LF Edge.

This chapter dives into areas where automation can help speed up deployment tasks and help **information technology** (**IT**) and **operations team** (**OT**) staff. The following topics are covered in this chapter:

* Automating service delivery
* Attaining scalability with automation
* Remote operational security
* Automation with **artificial intelligence** (**AI**)

Whether it is to manage edge devices more efficiently, apply configurations more consistently across varied and heterogeneous infrastructure, or distribute updates or patches, the goal of automation is to be able to do all of this automatically and flawlessly.

Most of the topics discussed thus far have had to do with the architecture, configuration, and setup of an edge computing topology or solution, which fall into the category of day 1 tasks. This chapter on automation addresses not only day 1 operations but also day 2 tasks, which have more to do with maintaining a system. Solution architects must always keep day 2 operations in mind when architecting a solution because a well-architected solution ensures a healthy system, which is best achieved with deployment automation.

# Automating service delivery

The goal of service delivery is to reduce the number of steps required to deliver a service. This is true in IT as well as in edge computing. Whether it is a simple application or a **machine learning** (**ML**) model to be deployed on edge devices quickly and securely, automating those steps is what teams strive for. Doing so helps eliminate configuration errors and scale such solutions.

Physical installation of devices is not something that the OT can automate, but the configuration of the devices and deployment of applications that run on those devices can be automated. If a large bank that has over 10,000 ATMs across the country wants to apply an update to its ATMs, it cannot afford to send personnel to every ATM. Rather, the update is done remotely either through command-line scripts or a DevOps toolchain. Another good example is damage detection sensors on wind turbine blades monitored by edge nodes located on top of the wind turbines.

## DevOps

Collaboration between **development** (**Dev**) and **operations** (**Ops**) has given us a set of processes known as **DevOps**, which allows for a pipeline to develop, test, deploy, and monitor code in a continuous manner. This is done with a set of open source tools, such as Ansible and Eclipse IDE, that work together in the form of a toolchain.

*EDGEOPS*

*Not all DevOps features are applicable to edge computing, especially continuous delivery, because as we know, some devices operate in disconnected mode. EdgeOps is a new paradigm that addresses edge computing-specific aspects, which can vary and be less predictable compared to cloud environments due to the heterogeneous nature of the edge hardware. This article from the Eclipse Foundation explains the EdgeOps project:*[*https://www.eclipse.org/community/eclipse\_newsletter/2021/february/2.php.*](https://www.eclipse.org/community/eclipse_newsletter/2021/february/2.php)

## Infrastructure as code

**Infrastructure as code** (**IaC**) uses the DevOps methodology to describe the layout of IT infrastructure and provides the ability to provision and manage resources. These resources include compute, storage, and network. They can be in the cloud, in a data center, or even at the edge.

There are many automation frameworks that enterprises can choose from, including Ansible, Chef, Puppet, and Terraform. There are pros and cons to each of these frameworks.

Enterprises either end up using what their developers are comfortable with or rely on the recommendation of their cloud provider. Ansible and Terraform are probably the two top automation frameworks (see the reviews on [https://www.techrepublic.com](https://www.techrepublic.com/)). While both help with automating provisioning and deployment tasks, and both are open source and cloud agnostic, there are subtle differences. See Table 7.1 for the differences:

|  |  |
| --- | --- |
| **Terraform** | **Ansible** |
| An IaC software tool that is used to automate building, provisioning, and managing IT resources | A suite of command-line software tools that are used as IaC to automate application deployment and configuration management tasks in IT |
| Written in the Go language | Written in the Python language |
| Uses a declarative approach | Uses both declarative and procedural approach |

Table 7.1: Differences between Terraform and Ansible

[Chapter 9](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_09.xhtml#_idTextAnchor172) provides more detail about the declarative and procedural styles of coding.

*GITOPS*

*GitOps extends IaC by including Git as the single source of truth where the infrastructure and application deployment manifest files are stored. All changes are initiated from Git.*

## Extending automation to the edge

The challenge has been automating the proverbial last mile when it comes to edge computing. The reason for that is the small footprint of far edge devices. Deploying infrastructure to and at the edge is best accomplished using a DevOps toolchain. DevOps toolchain diagrams are usually shown as a sideways figure 8 loop and encompass IT and OT tasks working seamlessly.

A DevOps pipeline, on the other hand, is a set of agreed-upon processes and related tools that allow developers to do their development tasks and collaborate with operations to deploy the coded applications. Figure 7.1 shows a pipeline with tools that help in building, testing, and deploying an edge server.

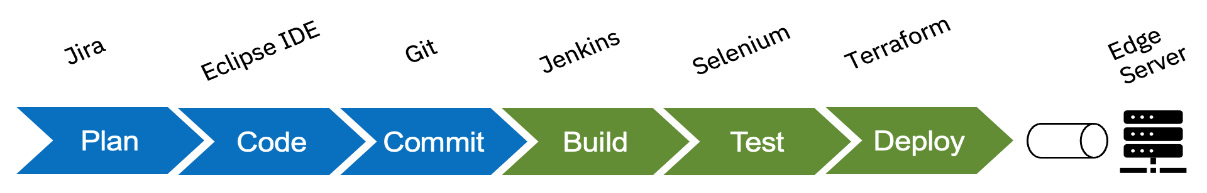


Figure 7.1 - DevOps pipeline to deploy edge infrastructure

With this diagram, we simply wanted to get the point across that the steps to configure an edge server/cluster are rather straightforward. The tools shown are example open source tools.

When it comes to deploying applications to edge servers/clusters and far-edge devices, a similar set of tools could be used but there would be two pipelines. The first pipeline would deploy the application to the edge server or cluster, and a second pipeline would be used to deploy container-based applications on the far edge devices. See Figure 7.2.

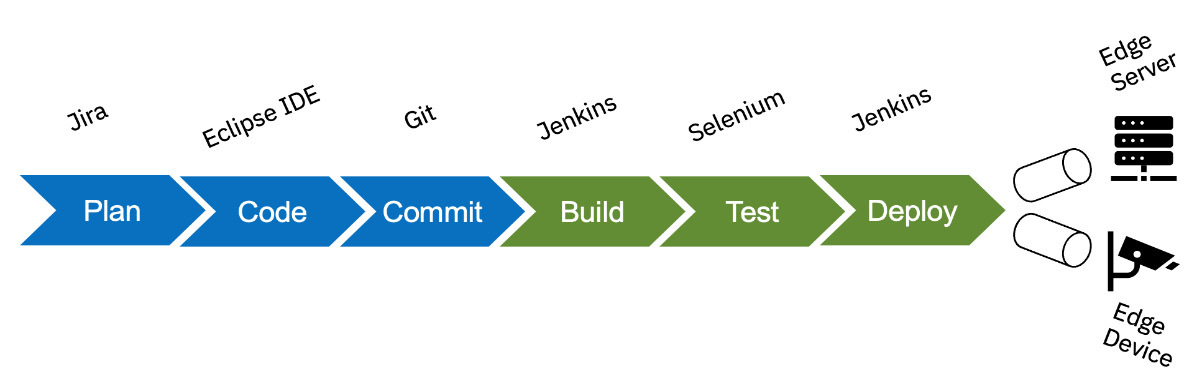


Figure 7.2 - DevOps pipeline to deploy an edge application

We talked about deploying edge infrastructure and applications on edge devices. What about developing edge applications? Let’s take a brief look at Open Horizon. [Chapter 10](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_10.xhtml#_idTextAnchor190) describes everything you need to know about Open Horizon.

Open Horizon, which is a project from the Linux Foundation (<https://www.lfedge.org/projects/openhorizon/>), is an open source platform for the development and autonomous management of software applications that are deployed on edge devices or nodes. Information about DevOps support in Open Horizon can be found at <https://github.com/open-horizon/devops>.

## Developing edge applications

The fun thing about edge computing is that anyone can create a simple application and deploy it to an edge device such as Raspberry Pi, a robot, a drone, or a Jetson Nano camera. Obviously, creating and maintaining applications that perform visual analytics or data inferencing requires expertise in several different fields of software development. The following code snippet is a very simple Hello World service that outputs **Hello from Packt** every five seconds. Such a simple edge service is typically used to test and confirm that an edge device is working properly:

#!/bin/sh

# Simple edge service

while true; do

    echo "HZN\_DEVICE\_ID says: Hello from Packt!"

    sleep 5

done

The service also outputs the device ID, which in a real-world use case is an easy way to identify the device. If you notice closely, it is an Open Horizon environment variable, namely **HZN\_DEVICE\_ID**. You could choose not to have it and make the program even more simple. If you want to get all the supporting code that deploys the service, you can find it on the Open Horizon GitHub repository (<https://github.com/open-horizon/examples/blob/master/edge/services/helloworld/CreateService.md>).

# Scalability with automation

We often hear the term **automation at scale**, which really means using data analytics and ML tools to help with automation across an organization. From an edge computing perspective, it would translate to having automation as we move from right to left—from the cloud to the network to the enterprise all the way to the far edge devices.

Now imagine, if that trivial program shown previously was to be deployed on hundreds of cameras in a large automotive manufacturing plant, the IT department would use a configuration and management software tool to do that. The biggest benefit of automation is being able to scale and, very importantly, roll back if every device is not updated. No matter the number of devices, one or more DevOps pipelines would help with that task. Once tested, the same deployment task can be repeated many times without the concern of introducing any human errors.

## Prepping an edge device

We have talked a lot about edge devices and their variances in form, size, and use. But what does it take to onboard an edge device such as a camera or a robot? The edge device/node owner takes on that responsibility and performs five tasks, as shown in Figure 7.3.

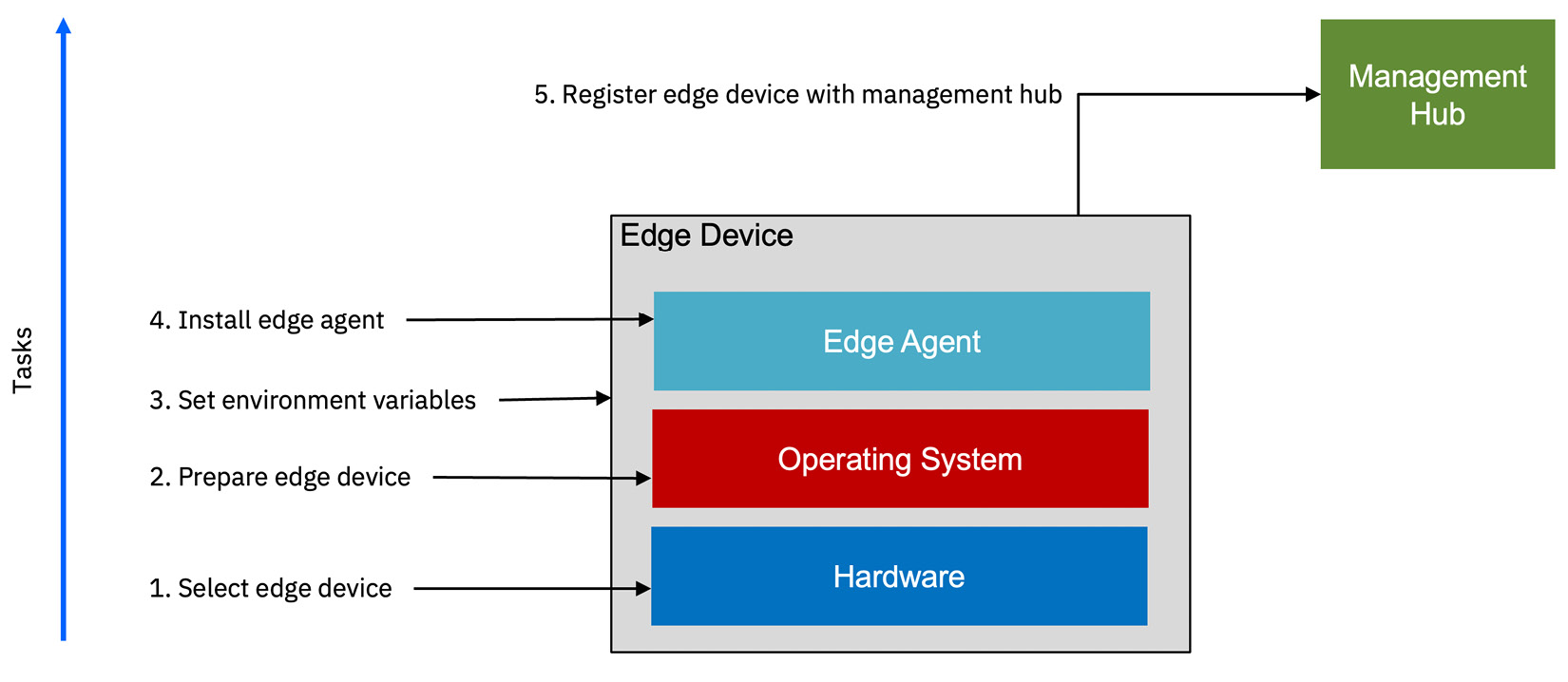


Figure 7.3 - Steps to prep an edge device

The high-level tasks are as follows:

1. Select the appropriate device, making sure the type of hardware is supported by the proposed edge solution.
2. Prep the device by installing the minimal operating system.
3. Set various environment variables.
4. Install an agent so that it can be registered with an edge management hub.
5. Register the edge device with an edge management hub in order to run edge services.

Once the edge device is registered, the management hub can monitor it and the device can also be added as an endpoint in a DevOps toolchain so that applications can be deployed to it.

The tasks shown previously must be done for each edge device that needs to be part of the edge solution. Zero-touch provisioning solutions make it possible to automate the initial imaging and configuration of new nodes. The zero-touch paradigm is described in [Chapter 9](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_09.xhtml#_idTextAnchor172). From a scalability perspective, and even from the perspective of IoT devices that do not have compute and storage, there is another option, which is to prep an edge cluster that is connected to such IoT devices. The steps are very similar.

## Prepping an edge cluster

A solution architect must be aware of when using edge clusters makes sense. **Edge clusters** are usually container-based clusters and are used either in colocation scenarios or in situations when more scalability and computing capability are needed to support many edge devices. The tasks are shown in Figure 7.4.

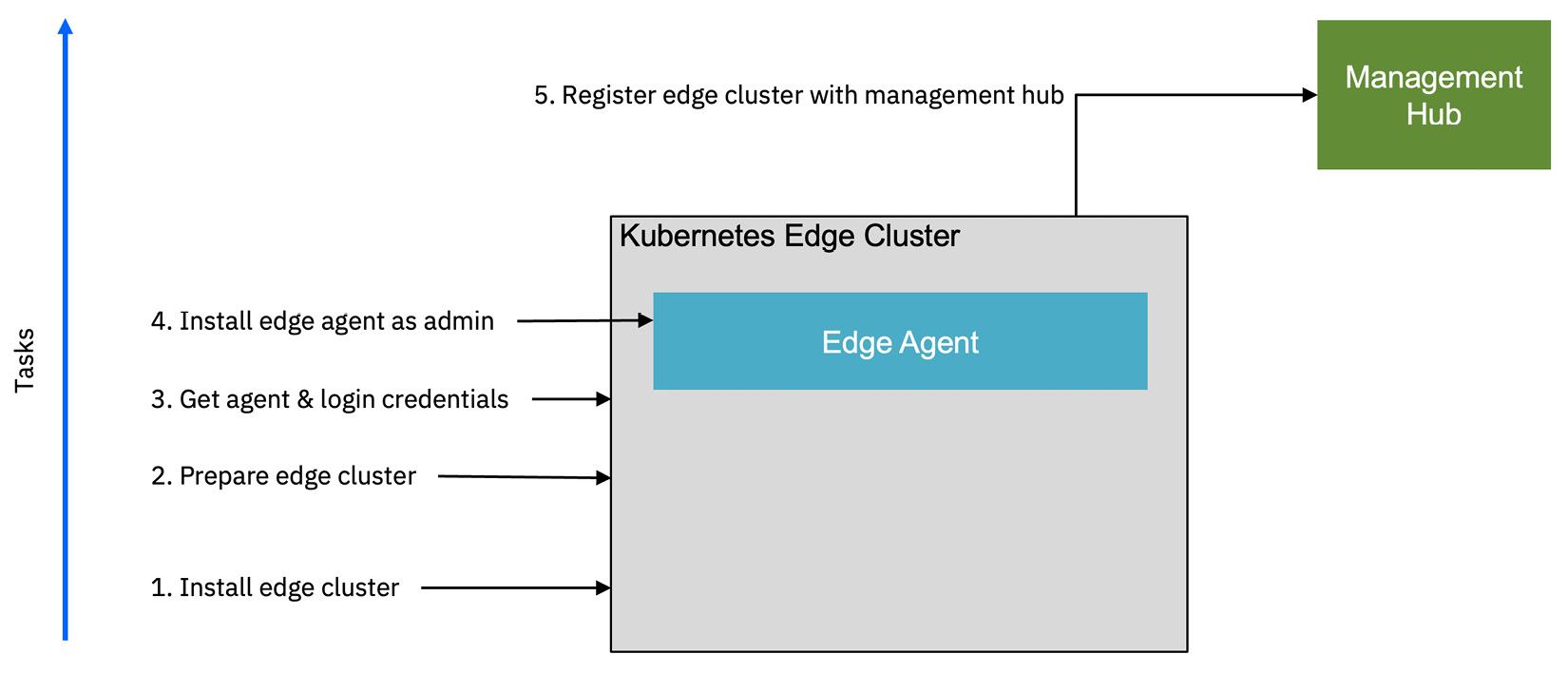


Figure 7.4 - Steps to prep an edge cluster

The OT gets involved when setting up an edge cluster. As before, after the edge cluster is registered, the management hub can monitor it and it can be added as an endpoint in a DevOps toolchain so that applications can be deployed to it.

# Operational security

As your IT team/OT manages day 1 provisioning and deployment tasks based on the architecture and plans defined during day 0, let’s address operational security issues from both a planning and process perspective, with emphasis on aspects that may be unique to edge computing. In this section, we’ll discuss limiting access in all layers of the architecture, how that applies to automation, and an example based on the concept of the Tactical Edge. As you read through this section, consider creating a custom checklist containing the items and considerations that apply to your specific architecture deployment scenario and then review it with the staff that will be working on the deployment.

## Limiting physical access

When it comes to regional deployments at the service provider edge, most access to equipment is managed by the facility staff and infrastructure. But when it comes to field deployments at the user edge, there can be little to no consistency or standardization. Some servers may be in racks, mounted in a room, or out in the open. Other edge devices may be wall mounted or sitting on a shelf or desk. Ensure that devices are in a locked room or otherwise out of sight and arrange for local staff to inspect the devices on a regular, scheduled basis. If practical, see whether automated visual monitoring of high-value hardware can be added to your architecture as a requirement.

Likewise, consider hardware specifications that include cases that limit physical access to device inputs, such as connectors, plugs, and slots, and that allow physical connections to be locked in place. If an unauthorized person cannot plug an external device into your hardware after it has been deployed, that is one less way to exploit your solutions. Lastly, ensure the hardware cannot be booted from an externally mounted storage device, and that you use trusted boot technology.

## Limiting connectivity

When deploying devices in the field, it is important to consider the ways in which a device can communicate, enumerating the allowed methods from that list, and then disabling all other methods. It may not be enough to disable functionality in the software, so consider physically altering the hardware to prevent the use of that function. Likewise, see whether you can determine the allowed hosts that the device may communicate with, and ensure that no other host is allowed to communicate.

If practical, disable all inbound communications and only allow the device to initiate outbound communications. The rule of thumb is the same as the standard zero-trust approach: deny all, then allow some as required. Any automation tools and configurations you use should implement this pattern. Ensure that your architectures and resulting requirements communicate this clearly and firmly, and that tests verify that it has been implemented for all frequencies, transports, protocols, and ports. Consider using device management solutions that track and automate the collection and control of this information and functionality, and only trust solutions that provide source code to allow independent verification.

## Trusted hardware and provisioning

Additionally, solution architects should consider the use of **Trusted Platform Module**(**TPM**) and **Trusted Execution Environment** (**TEE**) technologies, when possible, to aid in ensuring the identity and ownership of the hardware device.

These technologies can also be used to establish the ownership of devices and automate the transfer of ownership. The **Fast IDentity Online** (**FIDO**) Alliance created the **FIDO Device Onboard** (**FDO**) specification in 2019 to standardize this hands-free onboarding process. Intel Corporation then created an open source project named FDO to provide tools and an SDK and implementation example so that organizations could easily create solutions that complied with the specification. FDO works with TPM and TEE technologies.

The process begins when device manufacturers create an ownership voucher before shipping the device. When the user activates the device, the other half of the authentication process is completed, making the device operationally secure. See Figure 7.5 for an overview of the ownership transfer process following the FDO specification.

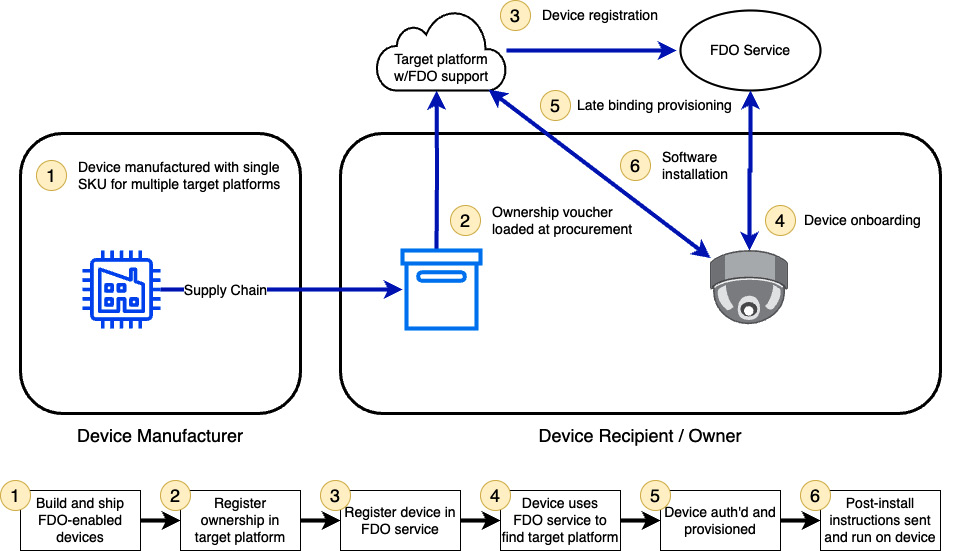


Figure 7.5 - Provisioning a device with FDO

Open Horizon’s support for the FDO project can be found at <https://github.com/open-horizon/FDO-support/blob/main/README.md>. There is a GitHub repository (<https://github.com/open-horizon/FDO-support>) where the corresponding integration software between FDO and Open Horizon can be found.

## Trusted data

Here we will list various solution components that will enable operational data security and show how they can be combined.

In the Data encryption section of [Chapter 6](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_06.xhtml#_idTextAnchor110), we discussed the concept of a **Data Confidence Fabric (DCF)** and how it can generate data confidence scores to give you a level of confidence that you understand the source of the data, how it may have been manipulated or altered, and how likely it is that the data has or has not been subject to undetected tampering.

Likewise, in the same section, we covered a technique that allows calculations to be performed on encrypted data without first decrypting it, and the results delivered are encrypted. That technique, **fully homomorphic encryption** (**FHE**), should be considered in any architecture where the data being collected is numeric and sensitive, confidential, or regulated.

In [Chapter 3](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057), we briefly mentioned the concept of perfect forward secrecy, where two parties communicating over an encrypted protocol (such as TLS 1.3) can then further encrypt messages using asymmetric public key encryption so the receiver can decrypt the message using their private key and the sender’s public key, with the session being encrypted using a key unique to that session. The strength of this approach is that in the off chance the session can be intercepted and decrypted by a third party, the key used to decrypt that session cannot also be used to decrypt messages between those parties sent in any other session.

## Trusted compute

We’ve talked about trusted data, trusted hardware, limiting connectivity, and other security-related features that businesses like to see before embarking on an edge solution. It all adds up to this paradigm of **trusted edge computing** (**TEC**), which strives for the next level of security of running secure and isolated containers, secure storage, and even secure memory. It also addresses business application protection mechanisms running on edge infrastructures. The solution architect must assure enterprises that their data, their program logic, and, ultimately, their intellectual property are protected. That should ideally involve remote attestation of business logic running on edge devices.

Running an application in a container provides some resource isolation, but it does not provide the sort of security isolation of identity or network isolation that you get when using VMs. Hence, you have to take advantage of the different layers of a container platform. Figure 7.6 shows the various layers, which put together provide the required security isolation and trust.

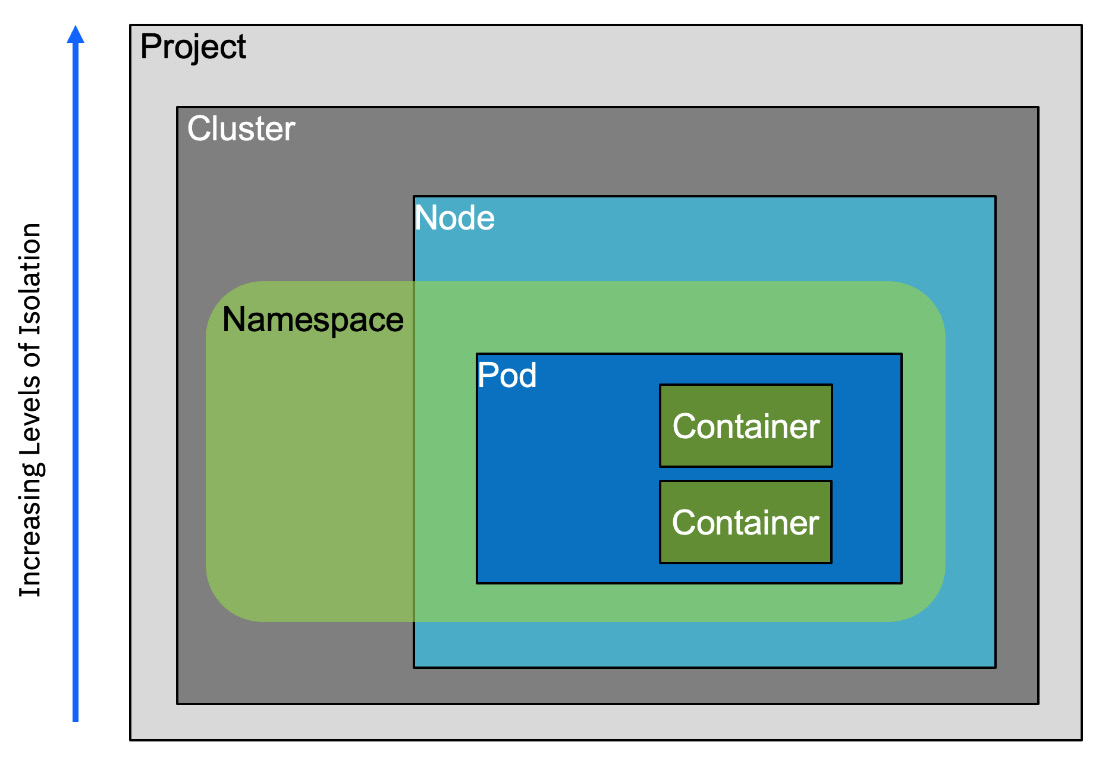


Figure 7.6 - Isolation provided by Kubernetes layers

As an example, Kubernetes has different layers that are nested. If you start from the smallest unit, the container on the inside, and move out, the level of isolation and security progressively increases. Container technology is one option. There could be other alternatives but the reason we focus on containers is that many edge applications are by their very nature more suited to run in containers.

## Tactical Edge

**Tactical Edge** is a term that describes secure edge computing in contested environments where different entities across different domains are vying for similar resources. It combines the best practices of operational security in edge computing working in tandem with a secure edge platform. Further, its default operating assumptions are geared toward both offline-first principles as well as **denied, disrupted, and intermittent connectivity with limited bandwidth** (**DDIL**). Let’s briefly describe what system components go into a Tactical Edge platform architecture, define their operational characteristics, and then discuss ways to automate the provisioning and operation of Tactical Edge environments.

Tactical Edge consists of hardware and software working together as a unified solution to support mission-critical workloads in field-deployed environments outside the core network and cloud infrastructure. It should do the following:

* Operate in disrupted, disconnected, denied, degraded, intermittent, or limited communication environments
* Function in degraded or otherwise extreme environmental conditions and operating environments
* Be self-sustaining as long as possible
* Be built as ruggedized units and maintained with **commercial off-the-shelf**(**COTS**), open source, and custom-built technologies
* Be deployable at scale, usable, and accessible in the field by anyone, in adverse conditions, and with minimal training

Attributes and elements of the Tactical Edge include the following:

* Trusted infrastructure, computers, devices, data, models, and identities
* Small-footprint devices, including both compute and wearables
* Runtime protection with zero trust
* Secure, redundant communication capabilities
* Resilient operation, yet breakable if removed from a specified geofence

*NOTE*

*Geofence is a technology that uses location services to enforce or activate a virtual boundary.*

* Deployable and replaceable in minutes without IT or OT staff, including zero touch

*NOTE*

*Zero-touch deployment is the ability to remotely provision edge devices without having to do it manually. The zero-touch paradigm is discussed in*[Chapter 9](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_09.xhtml#_idTextAnchor172)*.*

* Capable of running on-demand analytics and sensor fusion use cases adapted to local capacity
* Capable of bursting northbound for extra capacity

A Tactical Edge platform that meets the requirements listed previously is built on compute hardware that employs zero trust from the silicon on up, with a hardware-based root of trust for platform attestation, workload signing, and confidence-scoring injection into a data confidence fabric. Running on that hardware should be a Kubernetes distribution, integrated to provide hardware-based workload isolation and protection from competing workloads. Together, the two provide data protection at rest, in motion, and while processing. Additionally, a secure, pull-based workload and ML model life cycle management solution should be employed to handle placement and deployment tasks. Also, on top of that, a secure, redundant network overlay should be deployed and run.

*NOTE*

From the silicon on up*means visualizing the stack or architecture of a computer from the physical layer (motherboard, processing units, and memory) up to the operating system and/or virtualization layer to the running applications. It’s not enough to secure the applications if the operating system is not secure. And it’s not enough to secure the operating system if the hardware has vulnerabilities. By securing the complete system, beginning with the hardware, you ensure you’ve presented the smallest potential attack surface possible.*

Such an environment fits scenarios where the operating climate is harsh and stressful with restricted communication networks. See Figure 7.7. Some examples are ground operations on a battlefield, a fleet of merchant ships or aircraft carriers facing challenging weather at sea, and disaster response teams in disaster areas, where solutions must be resilient to disruption and sometimes survival is based on distributed decision-making.

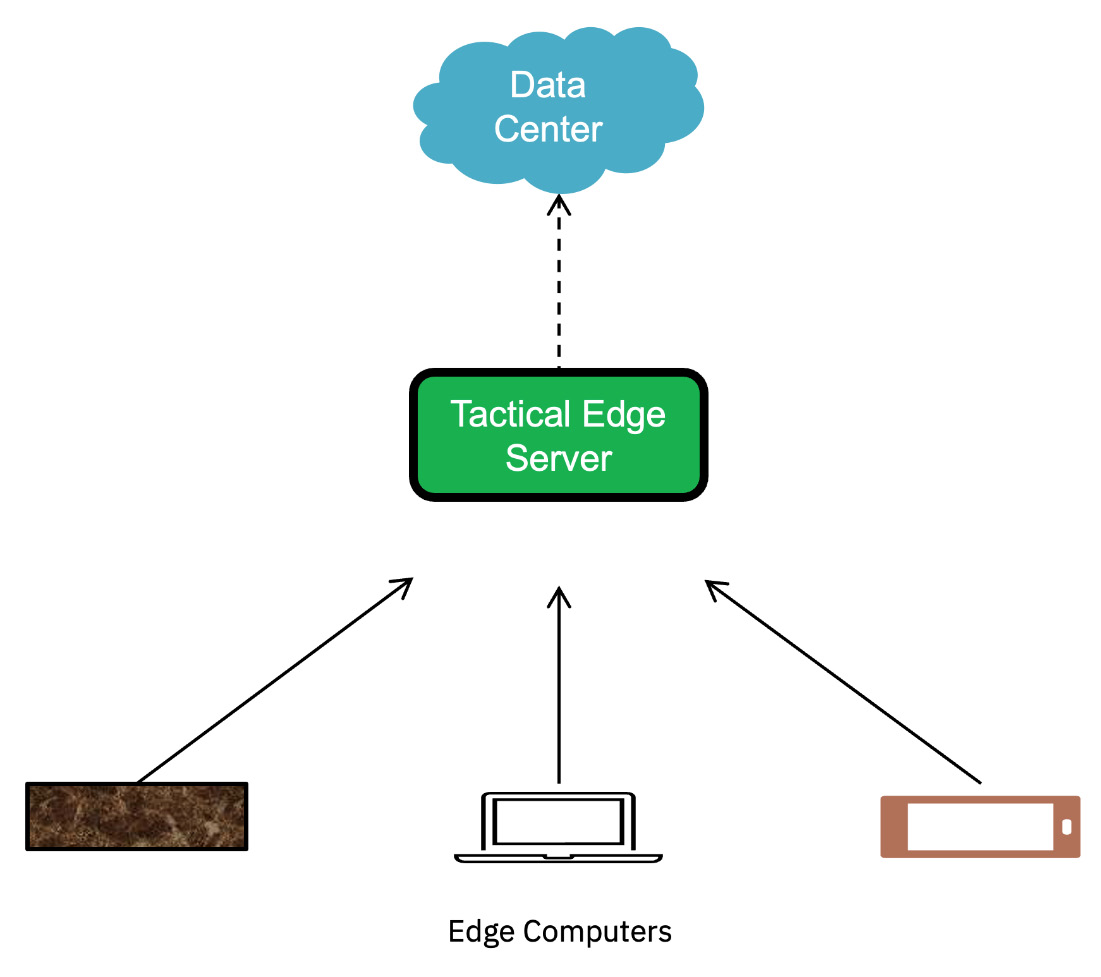


Figure 7.7 - Tactical Edge components

By using secure hardware compute and devices that support FDO technology for provisioning, hardware can complete all day 1 tasks over local networks or be pre-provisioned elsewhere and then deployed locally for scenarios and locations with no local connectivity. This approach will meet all of the preceding Tactical Edge considerations: zero-touch and zero-trust operation by minimally trained people in the field using COTS hardware.

In the next section, we will discuss scenarios based on how the use of AI can optimize automation.

# Automation with AI

We discussed deploying AI applications and ML models at the edge in [Chapter 5](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_05.xhtml#_idTextAnchor091), which is becoming a common scenario because enterprises are adamant about reducing the time for decision-making and minimizing data movement. In [Chapter 4](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_04.xhtml#_idTextAnchor073), we touched upon using AI/ML applications to determine network traffic patterns and using automation to perform network maintenance and monitor network performance. This latter discussion, albeit brief, is more in keeping with the automation theme.

Using AI techniques to automate facets of the edge computing paradigm will allow for automation at scale. With so much data being generated by edge devices, enterprises are finding ways to not only infer and analyze that data but also create a corpus that can be used to learn from, build, and train new models. We now see the rise of such corpus models as **Large Language Models** (**LLMs**).

## LLMs and generative AI

LLMs are massive amounts of data gathered from numerous existing sources that can be used to answer natural language queries, perform summarizations, or create classifications. These ML models that use neural networks are trained on massive amounts of data and can predict the next word in a natural language sentence. LLMs are a subset of **generative AI** (**GenAI**).

These models and techniques have given rise to a new form of AI, namely GenAI. As the name suggests, this category of AI models can generate new and previously unseen content, including text, music, and images. From an edge computing perspective, the best use case of GenAI is image processing.

LLMs and GenAI stem from **foundation models** (**FMs**), which are any model trained on very large data from the web using ML. They contain what are called tunable parameters in a mathematical sense, which ends up being their unit of size. When a model is listed as 3B, it means the model contains 3 billion tunable parameters. Although the terms LLM and FM are often used interchangeably, they are different because FMs can be more than just linguistic models. Figure 7.8 depicts which of the model types could be more relevant in the domain of edge computing.

*PARAMETER*

*A parameter is a configuration variable inside the model that can be tuned to optimize its performance.*

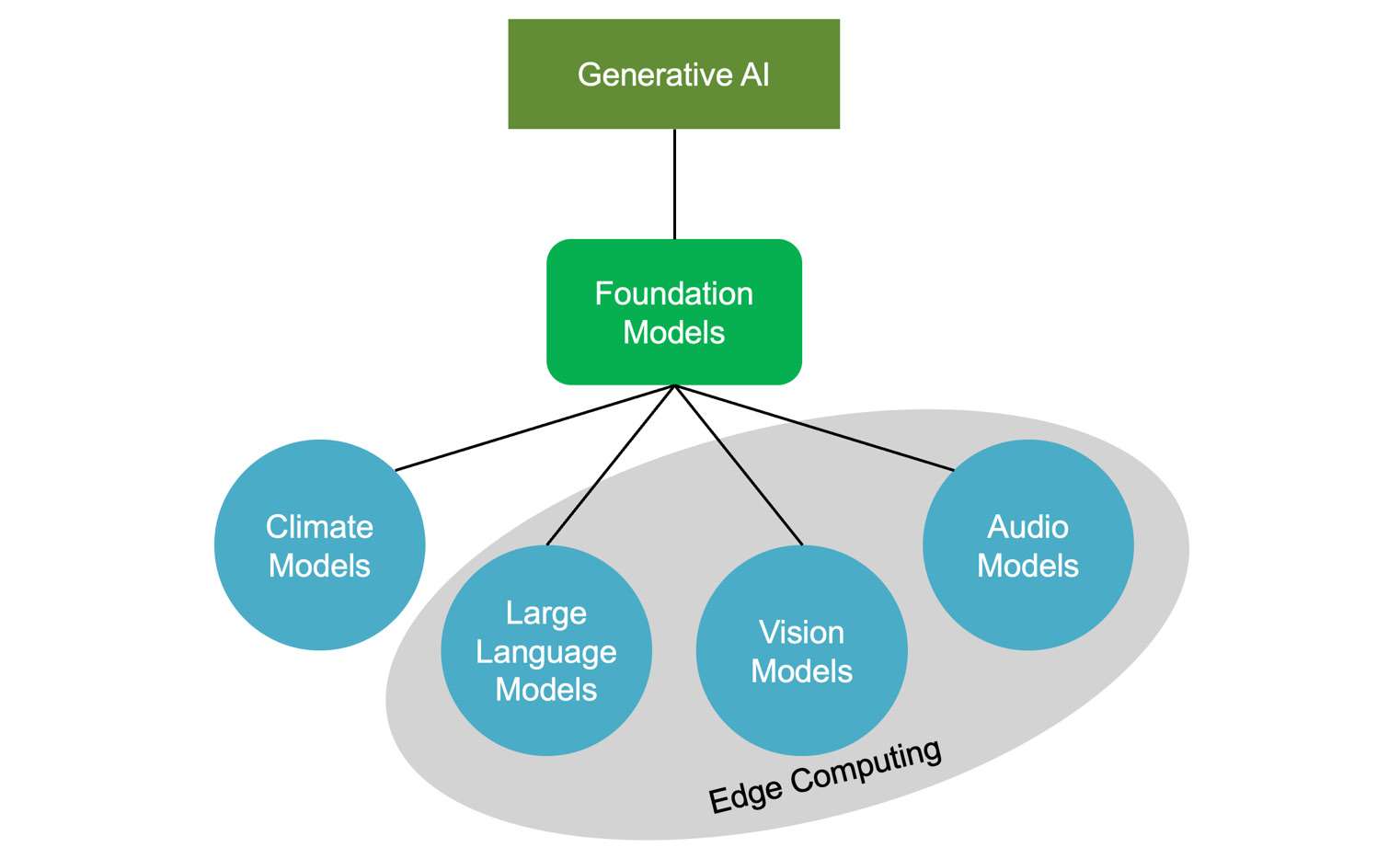


Figure 7.8 - FMs and LLMs in Edge Computing

We have seen drones used in industrial settings where they are deployed to assess and inspect an area or equipment. Imagine a drone sent to inspect a large concrete bridge to look for cracks or other structural variations. In the traditional world of AI, it would be limited to a small set of historical data, specifically images, to ascertain whether a linear marking is a crack or not. A human would have to determine whether it was indeed a crack after further analysis.

But now the drone can use the power of GenAI and compare what it is seeing to data contained in these large FMs to determine on its own whether something is a crack or a fault in the concrete structure. Using data from those large models improves the performance of visual analysis. Industry-specific vision models can be applied in different scenarios to get quicker and better results.

## Using AI in automation

The challenge lies in using all these AI domains to automate tasks and help OT and IT with managing and monitoring edge devices, clusters, and network components. Thanks to FMs, it is possible to automate the drone scenario of detecting faults or defects in industrial scenarios. That would possibly require less human intervention and mitigate errors in such scenarios.

Another example of where GenAI could play a major role in edge solutions is in responding to questions from customers as in the case of directional kiosks. This would take advantage of the question-and-answer capability of LLMs. Based on what the customer is enquiring about, the system could anticipate the customer’s next question and suggest things. Similarly, **quick service restaurants** (**QSRs**) could look at replacing order takers at drive-throughs with AI systems backed by GenAI, specifically the natural language processing capability. Taking the customer’s order, recommending add-ons, and speeding up the entire process is another example of automation with AI.

Automating with AI is a bit more obvious when dealing with the network because we can use historical data to determine network traffic patterns and use AI to automate network maintenance and help it perform better. As an example, this would really help in determining how to slice a 5G network, which can be a complex task. When it comes to **multi-access edge computing** (**MEC**), IaC can be an essential tool to facilitate their deployments. Furthermore, with the load that MEC could experience, it is important to continually monitor the demand and optimize all its services. Demand monitoring and auto-scaling could be achieved with automation, specifically automation using AI.

IEEE published a survey about automated application deployment on MEC that be found here (<https://ieeexplore.ieee.org/document/10225499>).

Section II describes MEC and provides a reference architecture. Section III describes the various IaC tools and compares them, including Ansible and Terraform.

# Summary

The chapter started with a discussion of service delivery automation, which is a rather broad term for automating human tasks in a business and is relevant to edge computing. Whether it is DevOps or IaC, achieving scalability through automation ends up being one of the major benefits. We saw that from monitoring edge devices to updating the applications and models running on those devices, there is an opportunity to apply automation.

The latter half of the chapter addressed the security aspect by introducing the onboarding of edge devices with security features before shipping them. While it seemed like we digressed, it was important to talk about security in the context of automation. Automation allows enterprises to build capabilities that can react with minimal human interaction in a more secure and trusted way.

The final section dealt with the most interesting and relevant topic, namely AI. It seems AI is omnipresent in edge computing from AI applications deployed on edge devices to the use of AI in the deployment of devices and their operation. Enterprises can now look ahead to the use of FMs in edge computing.

While we have briefly mentioned monitoring in this chapter, [Chapter 8](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_08.xhtml#_idTextAnchor152) takes a deeper look into the domain of observability.

# s8

# Monitoring and Observability

As Peter Drucker once said, “You can’t improve what you don’t measure.” Edge devices are often deployed in remote environments with minimal to no presence of IT operations teams, making monitoring and troubleshooting rather challenging. No matter how remote, enterprises need to maintain and protect their infrastructure, and that requires remote monitoring to manage their distributed assets.

The irony here is that edge computing solutions are typically used to monitor other systems, be it in stores, warehouses, or shop floors, or during manufacturing processes. So, who is monitoring the edge computing system, which is usually a combined hardware and software solution topology?

In an edge computing solution, there are many components to be monitored, from the application to the infrastructure. Components include edge hubs, servers and hosts, systems, network components, edge applications, and AI/ML models

In this chapter, monitoring and observability from an edge computing perspective are discussed to ensure that a deployed edge solution is performing as designed. Of all the components we focus on, the network and connectivity are critical due to their unique characteristics. Any monitoring solution must provide enterprises with the opportunity to measure the performance of the edge solution and, if needed, make improvements. Thus, these related topics are addressed:

* Monitoring and observability
* Measuring to improve
* What to measure

# Monitoring and observability

In the previous chapter, we talked about the need for automation to update applications with minimal human interaction. Monitoring, similarly, is key to knowing that edge devices are working properly and that the applications and models deployed on those devices are running. Note that there are two aspects: the physical hardware and the deployed software. The hardware in this case includes not only the various edge devices but also the edge hubs/servers.

Although sometimes used interchangeably, there are subtle differences between monitoring and observability. Based on our findings, in Table 8.1, we list some of the differences between the two:

|  |  |
| --- | --- |
| **Monitoring** | **Observability** |
| Reactive in nature, collecting logs | Proactively interpreting collected data |
| Is the system working? | What is the system doing? |
| Collection of metrics, events, logs | Deals with traces |
| What is the state of the system? | Why is the system in the current state? |
| Indicates when something is wrong | Tells you why something is wrong |
| Enables observability | Requires more than monitoring |

Table 8.1 – Differences between monitoring and observability

While monitoring is reactive and collects logs, observability is proactively interpreting the data that has been collected. A **log** is an information about an event in a human-readable format. The measurement of that event, in the unit related to that event, is a **metric**. The relationship between two or more events is known as **tracing**. These three types of captured event records form the basis of observability, as shown in Figure 8.1. Observability is in some sense a superset of monitoring:

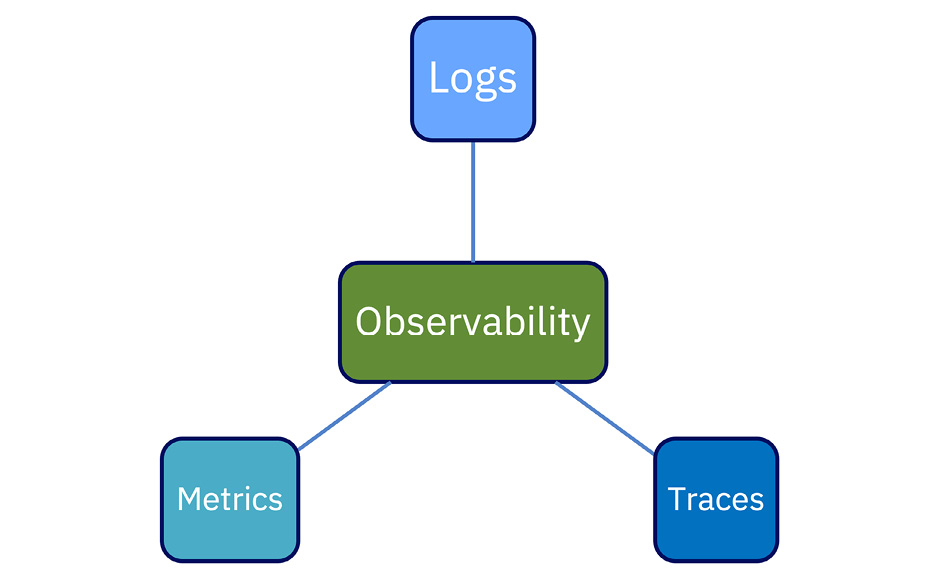


Figure 8.1 – The three components of observability

Observability is not unique to IT practice. It is actually an engineering term used in control systems. These could be applied to systems on the shop floor, in automobiles, or in any complex system that requires diagnostics to determine when something has gone wrong. From an edge computing perspective, observability allows enterprises to proactively detect, troubleshoot, and resolve issues, optimize performance, and ensure the reliable operation of edge devices. A solution architect must anticipate that customers will also enquire about network observability because they would want to ensure that the network is operating optimally.

## How monitoring works

It is important to constantly assess the health of a system. That is done by collecting data and writing those data records to logs. That is what monitoring is all about—measuring the health of applications, watching a system’s performance, creating alerts, and detecting failures. However, the value of monitoring really shows when developers and operations staff work together and determine the metrics to gather and what to log. There has to be a balance between logging too much and not logging enough.

## How observability works

With the metrics and logs collected, the next task is to analyze what has been collected and correlate data from several sources. When a problem occurs, observability would indicate why the problem happened. A good observability platform would not only detect the failure but even suggest remediation. An intelligent platform would go one step further and remediate the issue in order to maintain the promised **service-level objective** (**SLO**).

*ADDITIONAL INFORMATION*

*An SLO is a specific metric within a***service-level agreement***(***SLA***), such as 99.99% uptime.*

Of late. we see **artificial intelligence** (**AI**) and **machine learning** (**ML**) being incorporated into observability platforms to detect anomalies and surface predictive insights. This is especially true of network observability.

## How network observability works

Monitoring edge devices and edge applications and observing how they perform is the obvious part of edge computing. But, remember, the network is crucial in an edge solution. The **network operations** (**NetOps**) folks are constantly monitoring the network. Given that today’s networks are so complex due to the combination of hardware and software components and that they generate so much data, total network observability by humans is near impossible. It is left to intelligent observability software platforms to maintain complete situational awareness of the network.

The three main components of network observability are shown in Figure 8.2 – telemetry data from various sources in the network, a data platform that can ingest the telemetry data for analysis, and the ability to take corrective action:

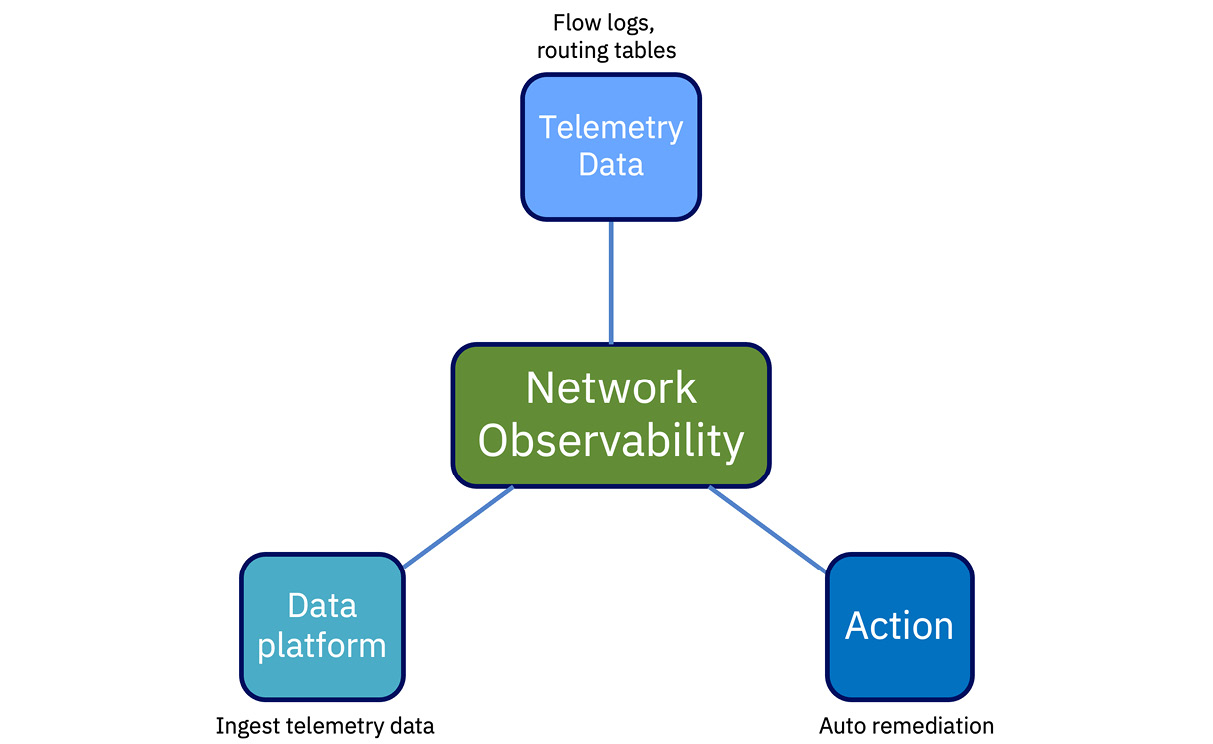


Figure 8.2 – The three components of network observability

Watching how the network is functioning, predicting trends in network traffic, and (above all) ensuring service assurance is what network observability is all about. And now, with new technologies such as **network function virtualization** (**NFV**), 5G, network slicing, and the dynamic nature of networks, using AI is essential when it comes to observability.

There are many monitoring and alerting tools that enterprises can use. Some of the common open source software tools are listed here in alphabetical order. This is by no means an exhaustive list:

* Fluentd (<https://github.com/fluent/fluentd>)
* Grafana (<https://grafana.com/grafana/>)
* Graphite (<https://github.com/graphite-project>)
* Nagios (<https://www.nagios.org/>)
* Prometheus (<https://prometheus.io/>)
* SkyWalking (<https://skywalking.apache.org/>)
* Zabbix (<https://www.zabbix.com/>)

# Measuring to improve

Performance optimization, threat detection and mitigation, and ensuring reliable operation of any system, including edge solutions, is the ultimate goal of monitoring and observability. All the collection of metrics and logging and setting up alerts will not amount to much if the gathered information is not used to do **root cause analysis** (**RCA**) and fix what is wrong or improve a system’s performance. In that context, dashboards play an important role in the observability domain.

Dashboards should help with visualizing curated data, providing context, and offering a holistic view over time rather than at just a point in time. By displaying graphs and trend lines along with the actual metrics, dashboards help humans see the history of the data and the impact of an alarm when it occurred, and, most importantly, they add context and reason. Note that alarms should only be set for things that are important and they should be actionable.

## Network observability example

We are starting to see how 5G is transforming sporting events for spectators by providing immersive experiences. Notwithstanding the costs, 5G connectivity allows fans to watch the event from multiple angles on different handheld devices and even indulge in **augmented reality** (**AR**). While the business outcome is a wonderful experience for the fans and spectators, the **communications service provider** (**CSP**) having to offer these services has to work very hard in the networking realm.

From the organizer’s perspective, the solution architect would have to work closely with the network engineers from the CSP and make many architectural decisions. This would involve making infrastructure-related decisions regarding equipment in and around the stadium, such as how many 5G towers, **multi-access edge computing** (**MEC**) nodes, and private 5G cores there should be, and more. Then, there would be network-related decisions to be made regarding the CSP, such as the amount of bandwidth to provide, how many network slices to allocate to the event, how to synchronize various camera feeds, etc. Figure 8.3 attempts to capture some of the network-slicing aspects during a large stadium event. Another option could be to use private 5G within the stadium:

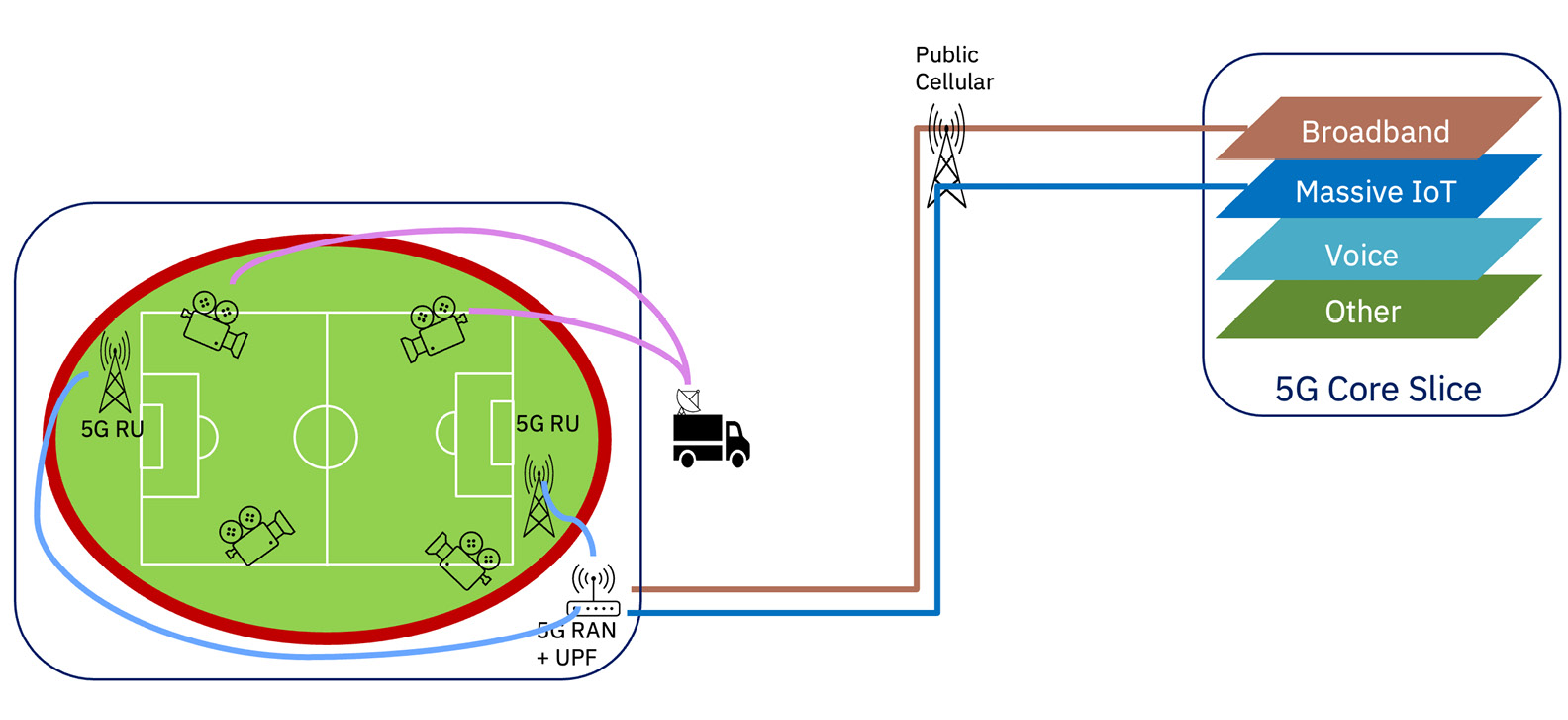


Figure 8.3 – CSP offering network services at a stadium event

Not all of the above-mentioned architectural decisions are static, one-time decisions. The network team would have to observe the network traffic and be ready to scale. As bandwidth demand increases, the system would have to dedicate more network slices to the event. These types of actions, based on traffic demand, metrics, and thresholds, are best done with AI-enhanced software because they can be tuned to anticipate and predict trends. The system would measure response times and the number of subscribers, seeing how spectators are interacting with the services, noting weather conditions, and taking corrective actions, all in real time so that the CSP can maintain the promised SLAs. The goal of monitoring and observability systems is to provide insights into resource utilization and data feeds and prevent any sort of hiccups. All this is done to ensure spectators at the event have a fantastic experience.

# What to measure

There are certain network-related metrics that help NetOps teams maintain the network uptime and then there are some infrastructure elements that need to be closely observed.

## Real user monitoring

In the stadium scenario, we mentioned the NetOps team would monitor how spectators were interacting with the services. That would require **real-time user monitoring** (**RUM**) data as opposed to synthetic or historical data. Getting a real-time view of what users are experiencing online is critical because the NetOps team needs to analyze events as they are happening and proactively look into slow connections or fix any misconfigurations.

Monitoring software can use RUM data to make DNS routing and other network traffic-steering decisions. There is also intelligent traffic steering, which is usually implemented using the overlay network.

*ADDITIONAL INFORMATION*

**Synthetic data***is different from real-world data. It is annotated data created using computer simulations and algorithms. It is artificial data from the digital world, but it reflects the real world. Synthetic data is used for training AI models, which requires large amounts of data.*

**Synthetic monitoring***is something that development teams use to test the features and performance of applications by simulating user actions. It is meant to discover any potential errors and have them fixed before an end user encounters them.*

*Synthetic monitoring can be used to improve network performance.*

## Network performance management

Whether it is a physical network or a software-defined network, monitoring its performance is what NetOps teams do. Performance data is collected using polling and other network streaming telemetry, so organizations can analyze and use graphing software to view it.

Some of the factors that affect a network’s performance are as follows:

* **Number of devices**: From an edge computing perspective, any network should be able to handle hundreds and thousands of devices.
* **Bandwidth**: This is the amount of data transmitted over a certain amount of time. New technologies such as 5G have vastly improved network bandwidth.
* **Throughput**: This depends on bandwidth and is a measure of the number of messages successfully sent through the network over a certain amount of time.
* **Network latency**: Given the remoteness of devices, network latency is the biggest concern. Network latency is basically the measure of how long it takes for a message to be transmitted across the network from one endpoint to the next. Remember, latency is affected by the number of devices on the network and the type of devices.

Most **network performance management** (**NPM**) software measures these basic metrics and a lot more, and they can present the information in simple dashboards. Network infrastructure and the type of edge applications used in the network affect network performance.

## Anomaly detection

Any value or event that is out of the ordinary or an outlier in a set of data is considered an anomaly. Anomaly detection is an important tool in network troubleshooting and intrusion detection systems. It is usually done by analyzing traffic flow patterns and network packets to uncover any hidden threats or intrusions. Network monitoring has to be done 24/7, and that is better done by an automated system. When an anomaly is detected, it should be mitigated or autocorrected or the NetOps team should be immediately notified. This is critical in maintaining network security.

## Capacity

Observing how the system performs over a period of time helps predict resource demands and even identify capacity gaps. That is true in network monitoring as well as for other infrastructure. Is the network able to handle the deluge of data from the edge devices? If the edge hub/server cannot keep up with the amount of data being generated by the edge devices, the monitoring and system should proactively adjust compute and even storage resources to maintain optimal edge solution performance.

# Business outcomes

Thus far we talked about the various technology-related measurements and metrics. Those become important because they directly or indirectly affect business outcomes. The common business drivers that enterprises focus on are as follows:

* **Customer experience**: Maintaining an optimally operating system improves customer experience. If the system is not performing well, it eventually affects the customer and adversely affects customer satisfaction.
* **Regulatory compliance**: Whether it is via traceability, audit logging, or security-related metrics, observability helps with meeting regulatory requirements. Through constant observance, the system must help maintain security and privacy standards.
* **Risk mitigation**: Continuous and automated observability can aid in identifying potential risks early and taking proactive mitigation measures. This would reduce the impacts of such risks, which could be expensive.

# Improving edge solution

In this section, we describe some of the monitoring challenges caused by the unique nature of edge topologies. To overcome these challenges and improve the performance of such edge solutions, the solution architect must be aware of how the designed solution should operate and how it is operating. There are the infrastructure components in the solution and there are edge applications running on those components. Different personas are interested in knowing how the two are performing - the Ops team and the developers.

## Monitoring challenges at the edge

It might seem obvious, but the operations team must review the current solution and take stock of the current state of all the components involved, especially the network components. Then, they must understand what the desired state is and get insights across all the components. In most cases, this should all be done in real time so that changes can be made without any interruption of service.

We mentioned the uniqueness of edge solutions. Its uniqueness lies in its ability to operate in disconnected mode. Hence, the challenge then becomes monitoring edge devices when they’re operating in disconnected or offline mode. **Disconnected**, or **offline**, means that the devices are operating but do not have connections to the enterprise or the cloud for a certain period of time. That means end-to-end metrics won’t be collected but operating logs will continue to be written locally. The system should be designed so that when full connectivity is restored, all local data will be uploaded and synced up automatically and the collection of metrics will resume. See a typical deployment in Figure 8.4:

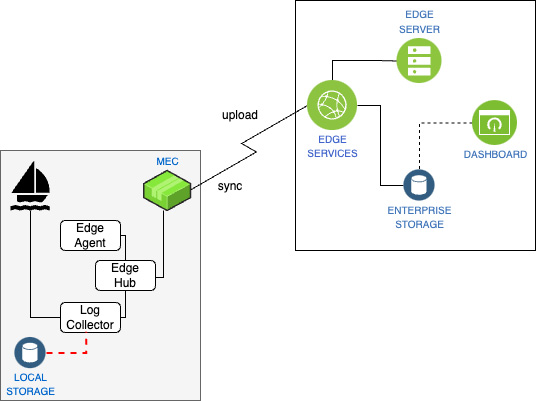


Figure 8.4 – Edge monitoring in disconnected mode

In general, the question is what is the right way to measure and report on autonomous edge devices such as ships, space-borne devices, and even autonomous vehicles when they are disconnected or offline? The recommendation is to plan for offline operations and continue with local logging and monitoring. When the opportunity arises for them to be reconnected, immediately reconnect them and upload and sync all the data with the enterprise system.

## Configuration changes at the edge

Observability is not always about how the system is performing or why it is not performing. It is possible that the system is not performing the way it is supposed to because there is a configuration change, either planned or unplanned. If a device is physically moved, is inoperable, or was configured incorrectly, then a connectivity component pointing to a nonexistent endpoint could break things in the edge application. Hence, configuration changes should be part of the observability solution, and observability dashboards should help correlate configuration changes with possible system errors, even if the cause is an edge device being inadvertently tampered with. The challenge for any edge computing operations team is to determine why a remote edge device is not online.

# Edge application monitoring

By design, we have focused on infrastructure monitoring because, in our opinion, infrastructure plays a major role in IoT/edge solutions. That doesn’t minimize the importance of applications and AI models running on edge devices. While these models and applications seem to have fewer variations, they nevertheless should be monitored.

Similar to NPM, there is **application performance monitoring** (**APM**): a process for monitoring the performance of applications. APM is meant to help IT professionals ensure that the deployed applications are performing reliably as designed. In edge computing, real-time APM is critical because it helps you identify issues before they have a major impact on the designed solution. The common metrics that APM tracks are CPU usage, memory usage, disk usage, response times, application uptime, and error rate.

Given that edge computing applications are typically cloud-native and container-based, there are other metrics that are more relevant, such as node availability, container start times, input/output data rates, number of instances, cloud costs, and APIs called. As was mentioned earlier, monitoring is critical because, either directly or indirectly, it affects business outcomes.

# Personas

There are certain personas that are closely involved and interested in the monitoring and observability of IT solutions. They are **site reliability engineers** (**SREs**), IT administrators, and NetOps and DevOps teams. Since edge solutions often use AI models to analyze data, we think AI and data scientists are the other personas that should keep an eye on those dashboards.

As a solution architect, one ought to take a holistic view and ensure that the monitoring dashboards are (a) customizable and (b) accessible to every team in the enterprise that has a stake in the solution. Based on our experience, in Table 8.2, we list some of the tasks that each of the personas would be responsible for. The business user persona is also listed because, ultimately, this is the persona benefitting from these monitoring and observability efforts. Enterprises may have additional roles or decide to combine a couple of them:

|  |  |
| --- | --- |
| **Persona** | **Tasks** |
| SRE | Monitor the availability of the system |
| Monitor alerts and critical incidents |
| Identify the root causes of issues that arise |
| IT admin | Track resource usage and optimize capacity |
| Respond to outages |
| Manage SLOs |
| DevOps | Monitor application performance |
| Obtain release and pipeline feedback |
| NetOps | Monitor network performance |
| Watch for and respond to network outages |
| Monitor network anomalies |
| Data engineer | Monitor the performance of models |
| Watch for model drift and model retraining |
| Business user | Track the overall health of the system |
| Monitor business-defined SLAs |

Table 8.2 – Observability personas and tasks

Additionally, a solution architect should actively look for how the information in the dashboards is consumed. If an automation step can be added, especially through judicious application of AI, propose that. Anytime you can shorten the **observe, orient, decide, act** (**OODA**) loop, you reduce your enterprise’s time to act, thus increasing responsiveness and decreasing costs. Ask yourself this: why are we collecting this data, how will we use the data, and what possible responses can we choose based on the data? Those are your automation opportunities.

# Summary

If monitoring actively collects data by way of logs and metrics, observability applies intelligence to make sense of the collected data and provide actionable insights. The chapter listed the differences between the two and pointed to the fact that monitoring and observability complement each other to help solve problems in the IT world. The same is true in edge computing. The end goal should be to dynamically and continuously improve the overall efficiency of the edge computing infrastructure. If you find an area in your processes where observability is not currently implemented, consider addressing that gap at your next opportunity.

What to measure and measuring to improve were discussed in the latter part of the chapter. This highlighted the opportunity gap: enterprises should know the current state and desired state of the edge solution. Only by continuously monitoring the system can enterprises achieve the desired state, and that drives operational maturity. Lastly, application performance monitoring was discussed.

In [Chapter 9](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_09.xhtml#_idTextAnchor172), the connectivity aspects of an edge solution will be described, which is a natural extension to maintaining the desired state.

# 9

# Connect Judiciously but Thoughtlessly

There are three common and potentially overlapping scenarios when connecting applications and services to each other and to fixed infrastructure services: connecting within a secure trusted network behind a **DMZ** (**Demilitarized Zone, a perimeter network**), connecting over an untrusted network, and connecting over a transitory or unstable network, whether trusted or not. Each organization involved in providing or enabling that access will have its own processes for provisioning and managing that connectivity. However, all of them serve a purpose orthogonal to the needs of an application architecture.

This chapter discusses a point of view for connectivity that begins and ends with the needs of the application and those that maintain it to optimize the speed and agility of developers and **SRE** (**Site Reliability Engineering**, IT Operations using software engineering practices) professionals.

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

* Declarative versus imperative configuration
* Zero Trust or as close as you can get
* Overlay, underlay, and shared responsibilities

By the end of this chapter, you will understand the benefits of application-directed networking and the potential value it can bring to an enterprise by bridging operational silos and abstracting proprietary cloud vendor-offering interfaces and network underlays.

# Suggested pre-reading material

If you would like to learn more about the topics we will discuss, the following resources are recommended:

* Application-centric infrastructure: <https://venturebeat.com/business/what-is-application-centric-infrastructure/>
* Application-centric networking: <https://www.ibm.com/blog/application-centric-networking-for-the-modern-enterprise-era/>
* Zero Trust explained: [https://www.youtube.com/watch?v=yn6CPQ9RioA](https://www.youtube.com/watch?v=yn6CPQ9RioA%20)
* NIST Special Publication 800-207, Zero Trust Architecture: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf>

# Declarative versus imperative configuration

In this section, you will learn about approaches to, and tools for, configuring your edge-connected networks that will increase deployment velocity while removing barriers and bottlenecks. We will give an example from the emerging field of application-centric, or application-directed, networking. By the end, you will be able to explain the benefits of this paradigm and consider when to use it for your distributed edge application architectures.

## Comparing the two approaches

Cloud Foundry and other platforms introduced, and Kubernetes popularized, the idea of marrying declarative configurations describing the desired outcome state to the concept of eventual consistency, thus bringing a system gradually into alignment from the current state toward the eventual goals. The declarative paradigm is a way of stating (with configuration files or a higher-order language) the outcome that you would like to achieve without needing to specify how to do so. This gives the implementation of the solution the flexibility to determine whether the target system or environment already matches that stated outcome, or whether changes need to be made to bring it into correspondence with that end goal.

The imperative approach, by contrast, specifies specific operations to be performed in the specified order without an understanding of the current system state or the administrator’s eventual goals. This produces brittle solutions that can easily fail when they hardcode assumptions about an operating environment and its participants. These imperative configurations can also lack validation checks and enforcement of full life cycle coverage unless those are also specifically anticipated and covered. This leads to boilerplate code/configuration inflation and a push-and-forget mentality that assumes the task is complete once the tool has run without throwing errors. Every subsequent modification to that configuration or code requires a human to modify it, test the updates, and then push the resulting assets into production.

In edge computing, when you are dealing with so many different architectures, operating systems, and hardware devices, using an imperative approach to configuration introduces risk by specifying precise, static steps to take that may very well be environment- and microarchitecture-specific. This is why edge-native best practices work best with tools and solutions that use a declarative approach to configuration and allow the tool to dynamically determine the best way to achieve the desired outcome. It abstracts away the complexity and differences from one application host and environment to another. See Table 9.1 for the authors’ high-level summary directly comparing the approaches.

|  |  |
| --- | --- |
| **Declarative Programming** | **Imperative Programming** |
| Describes the desired result with no direction on how to achieve it | Describes how to accomplish the desired result |
| Specifies what is to be done | Specifies how it is to be done |
| Variables are usually immutable | Variables could be mutable |
| Code optimized by the system based on constraints created by the programmer | Code optimization is the responsibility of the programmer |
| Details are largely hidden or minimized from the developer’s view | Developers have a lot of control, which is important in low-level programming |

Table 9.1 – Declarative programming contrasted with the imperative approach

With these two approaches in mind, let’s discuss the barriers and bottlenecks faced by modern enterprises when it comes to placing and connecting applications to services, other applications, and infrastructure.

## What slows down application deployment on the edge?

In an enterprise, CloudOps or NetOps (the terms, and team responsibilities, are sometimes interchangeable in this context) is responsible for providing secure, performant connectivity for applications wherever they are located to the resources they are connecting to. DevOps is responsible for the **continuous integration and continuous deployment** (**CI/CD**) pipelines, which place the workloads and related assets at their destination and connect them to their resources and dependencies. It is incumbent on DevOps to communicate with CloudOps about the workload connectivity requirements, and then wait for CloudOps to provision the requested connectivity with the appropriate access control restrictions in place.

This back-and-forth can take weeks to complete and will need to be performed for all environments: development, testing, and production. Additionally, the process will need to be repeated for any modifications to existing deployed workloads, and for workload removal.

If a system or tool is misconfigured or otherwise does not function properly in production, it requires collaboration between the CloudOps and DevOps teams to analyze the situation, perform a **root cause analysis** (**RCA**), and then request the appropriate changes to fix the issue(s). However, it is quite possible that existing observability and monitoring tools do not provide enough information to determine the root cause without back-and-forth experimentation and testing. This leads to a very long time being spent with all teams addressing discovered problems.

One other issue is that this **slow-motion communication and configuration change process** prevents the ability of either team to allow the solution to adapt itself rapidly and dynamically to changing environmental factors, whether cost- or performance-based. In other words, it prevents **dynamic freedom of movement**.

In summary, two or more different teams are responsible for different types of tasks:

* Deploying applications efficiently without error
* (Dynamically) connecting applications securely with good performance and within budget

The tools at their disposal are only designed and optimized for those specific tasks and views and do not translate well to other domains and those domains’ terminology, leading to potential communication issues when troubleshooting. And the teams are not incentivized to help other teams meet their KPIs, nor to provide methods and processes to make the other team’s tasks quicker and easier to accomplish. See Figure 9.1 for a comparison of the two approaches and how the latter application-centric approach allows responsibilities to be shifted left.

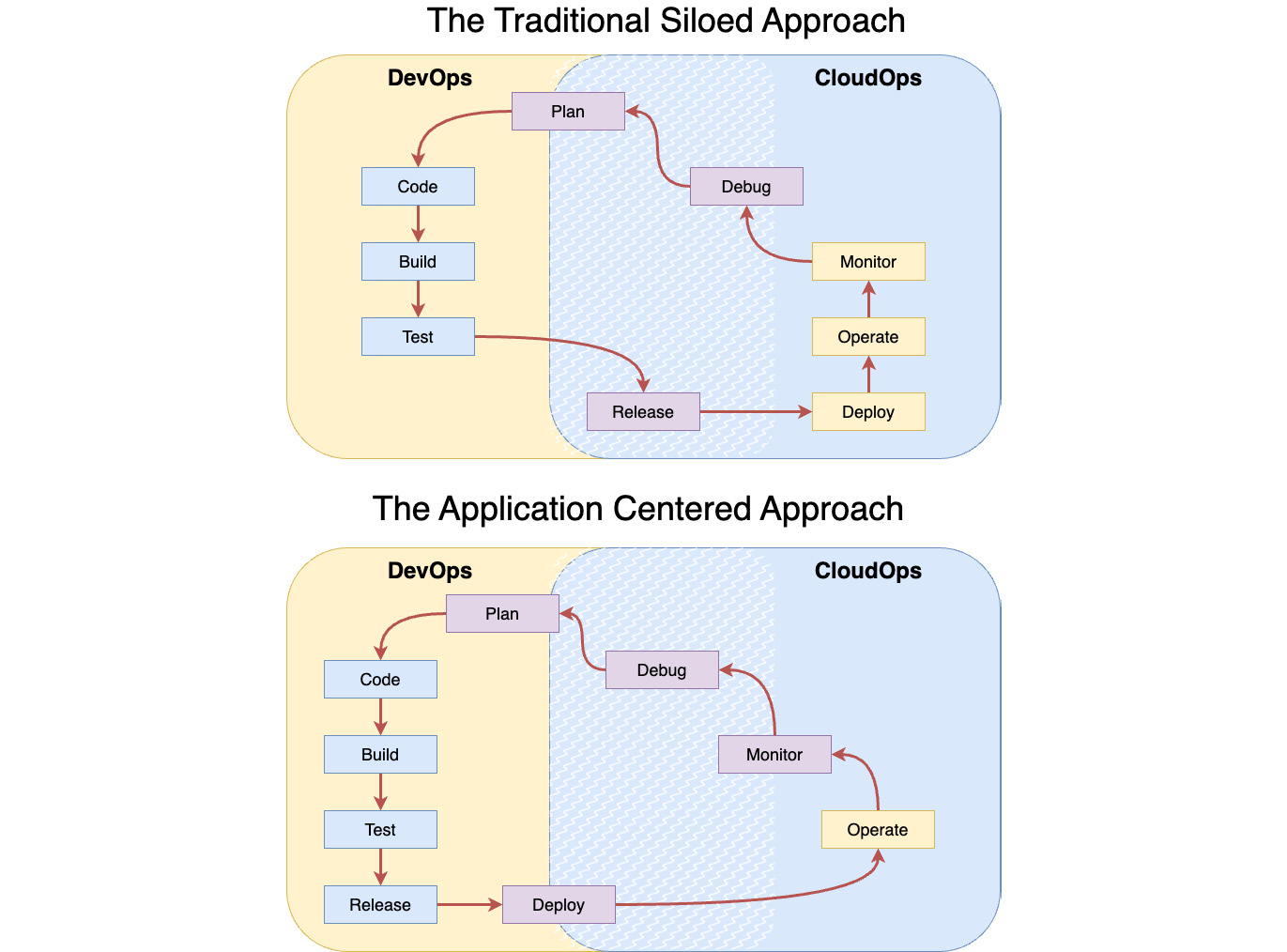


Figure 9.1 – Application life cycle management responsibilities compared

If a type of solution could provide DevOps with a self-service function when it comes to network connectivity, as shown in the lower diagram of Figure 9.1, while still allowing CloudOps to have granular control over the when/where/what policies and rules of that connectivity, that would go a long way toward improving the status quo. Additionally, it would need to provide enhanced tools and dashboards to expose the state of the workloads and networks and allow “what if” scenarios to be proposed and tested. It would need to allow CloudOps to delegate some execution authority to the DevOps team to be able to make isolated changes to network overlays (and sometimes underlays) specific to the workloads being remediated without potentially disrupting any other activities on the network underlays. Lastly, it should provide the capability to implement dynamic automation so the configuration, and resulting workload deployments, could react to changing conditions by moving and scaling seamlessly.

Let’s break the problem down into specific approaches that will help you better solve the aforementioned issues and proposed solution requirements.

## Solutioning edge-connected networks and applications

Eight best practice approaches are listed here that address those issues for workloads deployed to the edge in a hybrid, multi-cloud world, ideally using declarative configuration:

* Any solution you choose should be abstracted from any physical hardware or vendor-specific APIs. By using a tool, interface, or API that is vendor- and solution-agnostic, you avoid the trap of specifying your desired outcome(s) as a series of “do this and then that” imperative steps that must be followed in an inflexible sequence. This frees you to instead list the outcomes to be achieved, and in order if required. It also allows you to use vendor-neutral language or terms to describe those outcomes so that you do not accidentally use vendor-specific meanings, overloaded words, or vague terminology. It also gives you flexibility and freedom of movement by ensuring that you are not locked into a specific vendor, platform, or paradigm.
* Your solutions should rely on auto-discovery of network underlays and related infrastructure, namespaces, policies, ACLs, applications, services, and platforms. Few individuals in a company know how to connect to multiple cloud vendor platforms, query the information, and then consolidate all of that in an easy-to-understand format. By using a solution that can provide that capability, you increase your CloudOps team’s capacity to respond to requests and allow the team members to focus on higher-value tasks. Furthermore, there’s the issue of providing access (and maintaining that access) to individuals who need the information. Instead, organizations are bottlenecked by requesting the information from those who have it, assuming they receive the information they need in the first response. Instead, by having a single solution that is provided with read-only access to all cloud resources, you can then provide access to that solution to any person in the organization who needs the results. A self-service platform could provide many, if not all, of these capabilities. However, adding the auto-discovery of resources to this mix reduces initial setup and ongoing maintenance times from days to hours or even minutes.
* Your solutions should be able to dynamically reconfigure network infrastructure within the bounds of CloudOps-controlled and -specified guardrails. If a DevOps team member needs a reasonable change made to enable their workload to function, and it does not interfere with existing and planned network usage, and the changes are capable of being automated, then that type of change is safe to perform. If that work can further be delegated to a tool that will validate the configuration parameters and implement the change without error, then it is safer to have the tool perform the update than it would be for a human.
* Any solution you choose should be able to dynamically provision and modify an application-connected network overlay. This is another task that would typically be performed by a CloudOps team. However, it also requires detailed, error-free communication and advanced planning between teams to work out what should happen, to schedule times when all parties are available to perform the work without impacting the production environment or causing unreasonable downtime, and then testing time afterward to ensure it was correctly performed and has the desired outcome. If your solution can automate all of those steps safely and without error, then it speeds up rollouts while also reducing the potential for errors and the expense of all of the workers it would normally require. A solution that can do this more than pays for itself.
* Your solutions should be application-aware and application-centric. This capability ensures that applications, services, infrastructure, and related resources exist and are where they are supposed to be. It also determines how they are configured (including access controls), what the software versions are, and what resources they are accessing (both physical and virtual), and then surfaces that information so that it can be consumed by other tools and solutions. Without this capability, humans would have to perform all those actions flawlessly every single time.
* Your preferred solutions should be able to operate autonomously. There are several reasons for this requirement. With the appropriate configuration, a tool is more than capable of achieving the outcome, so why have a person manually follow the steps? More importantly, the steps will need to be performed on remote systems, and by ensuring that only your tool has access to the target systems, you reduce the potential attack surface and do not need to update which persons need access to which systems as your organization experiences normal employee turnover. And with no human in the loop, you can ensure that the tasks can be performed at any time.
* Your solutions should be integrated with real-time observability and monitoring solutions. When you use declarative configuration in conjunction with automated and autonomous solutions, you ensure that tasks are performed consistently and correctly every time. But if you are also able to add information about the operating environment, application and network performance, and all potential resources, you level up your capabilities. This means that your deployments can respond to changing conditions and situations to match the desired operating targets more closely. It will also give you the ability to consider and test “what if” scenarios to iteratively improve and optimize application performance.
* Any solutions you choose should enable zero-touch, hands-free operation. This is the last, critical step in removing **human-in-the-loop** (**HITL**) interactions to prevent errors and reduce operating expenses. This does not mean that humans cannot or will not be able to monitor the results and adjust as needed. In fact, the law of unintended consequences ensures that a human should be monitoring for undesirable situations that would not normally be foreseeable. But the point of this is removing persons from the rote, repetitive tasks and allowing them to instead perform high-value work that is not well-suited for automation.

The aforementioned best practice approaches can be implemented piecemeal by connecting and integrating component solutions (for example, Skupper, <https://skupper.io/index.html> or OpenZiti, https://openziti.io/), in a cohesive solution or platform (for example, IBM Hybrid Cloud Mesh, <https://www.ibm.com/products/hybrid-cloud-mesh>, or Nephio, <https://github.com/nephio-project>), or as part of a platform engineering solution (for example, Azure Radius platform engineering, <https://github.com/radius-project/radius>).

With the preceding best practices in mind, let’s discuss compatible approaches to security in edge-native application development, deployment, and operations.

# Zero Trust or as close as you can get

In this section, we will discuss common and emerging practices to secure the application’s use of the network, and what really matters. We will review major aspects of a Zero Trust architecture in the context of edge computing. By the end, you will be able to describe the approaches, what potential benefits they bring, and when they might be useful to your edge architectures.

Let’s begin with the basics and discuss handling secrets. It can be surprising how often this discipline is overlooked or ignored, yet proper implementation is critical to prevent unauthorized access.

## Managing secrets on the edge

Software development teams should not only be trained in how to securely connect applications to remote services but they should also be provided with edge-native solutions to enable them to properly manage the credentials, API keys, certificates, and other secrets. While some tools such as GitHub provide built-in secrets management capabilities, solutions that span the architecture should have a dedicated secrets management solution, such as Hashicorp Vault, EnvKey, or OpenBao.

While it is important to store secrets in a central location, it is also critical that administrators are able to do the following:

* Distribute secrets only to the applications that need them
* Send secrets only when they are needed
* Configure the secrets to be readable only by the intended applications
* Deploy, update/renew, and rotate certificates on demand

Do not embed secrets, credentials, or keys inside applications, containers, or even environment variables. Passing secrets as arguments on the command line is also not secure. Use a secrets management solution that stores the information securely and separately and only binds the secrets to the application at runtime.

Now that this topic has been addressed, let’s step back and ensure familiarity with terms and principles for security in edge architectures, beginning with Zero Trust.

## Zero Trust architectures in edge computing

The **Zero Trust Architecture** (**ZTA**) for remote, cloud-based assets published by NIST in SP 800-207 is shown in Figure 9.2. In the middle are the three core components:

* Policy Engine
* Policy Administrator
* Policy Enforcement Point

On the outside are the data sources providing input and policy rules to the core components.

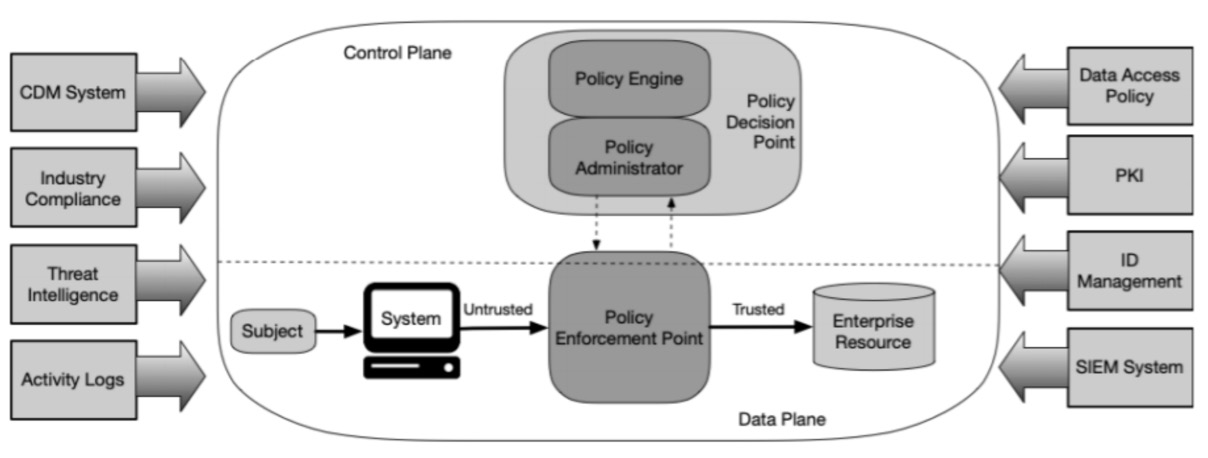


Figure 9.2 – ZTA from NIST

(Image Source: NIST SP 800-207:<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf>.)

ZTA uses zero-trust principles when it comes to enterprise infrastructure and workflows, meaning no implicit trust is granted to any assets, devices, or user accounts, no matter where they are located or who owns them.

A common framework for implementing these zero trust principles in hybrid cloud and edge computing is the secure access service edge framework. Let’s delve into that next.

## Secure access service edge

There are unique security challenges with edge and distributed computing applications. The **secure access service edge** (**SASE**) framework addresses those challenges by combining Zero Trust security with wide-area networking. Hyperscalers offer SASE as a service applicable directly to edge devices, thus allowing devices and systems to securely connect to applications and services anywhere.

Instead of the traditional security perimeter around a data center, SASE offers policy-based security services at the edge. This is made possible by using a **software-defined wide area network** (**SD-WAN**) capability in the overlay network. Using SD-WAN lets enterprises scale their security posture. Figure 9.3 shows the convergence of network and security services in a cloud-based SASE architecture:

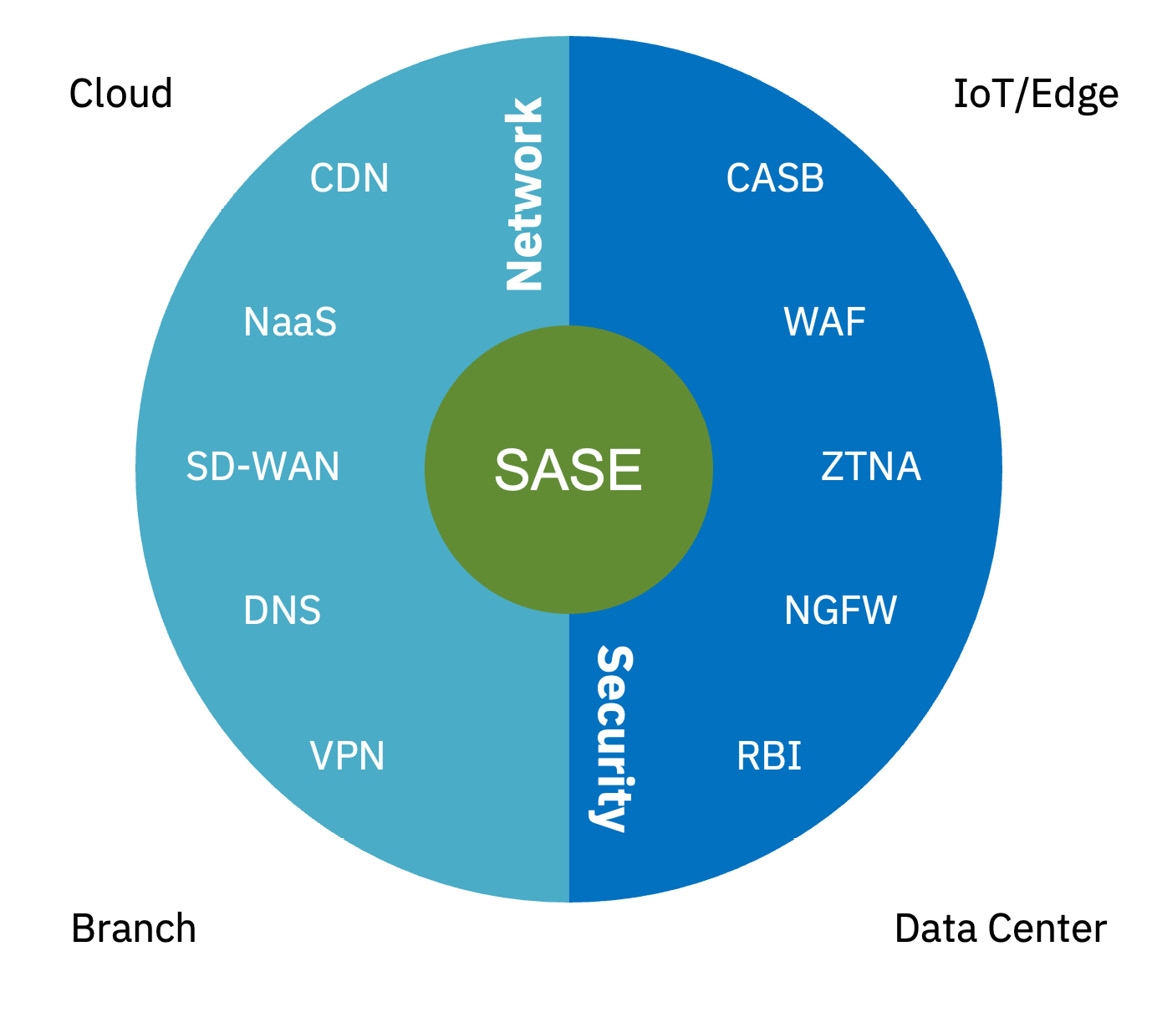


Figure 9.3 – The SASE architecture

The following are the descriptions of the technologies mentioned in Figure 9.3:

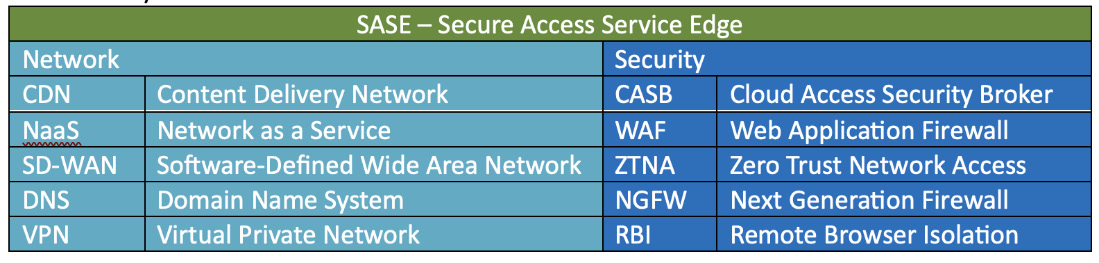


Figure 9.4 – Glossary of terms

The policy-based approach that SASE provides is yet another example of the use of the declarative approach to application architecture configuration. Please note that it is largely focused on the network overlay. This is a good time to look at the network overlay and discuss how edge computing and application-centered networking should focus on this abstraction. Let’s discuss how we should approach the overlay and underlay differently and the reasons for those differences.

# Overlay, underlay, and shared responsibilities

Should edge computing architectures care about the underlying physical infrastructure, or should architectures just assume that the underlay exists, has standard capabilities, meets industry norms regarding **service-level agreements** (**SLAs**), and is reasonably well-maintained, and then abstract away any details and differences?

Enterprises can continue to use network segmentation, which is an architectural approach to isolate the internal network from the rest of the internet. In so doing, it not only improves security and access control but also helps with performance by creating access policies that are enforced via firewalls. With newer technologies now, there are other options.

In this section, we cover different approaches to edge-friendly network overlay implementations. Along the way, we discuss how the overlay can assist with network-level application isolation and why that is important. By the end, solution architects should be thinking about approaches to connect applications and services to each other and to cloud infrastructure.

## The network underlay

In the Industrial edge scenario section in [Chapter 5](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_05.xhtml#_idTextAnchor091), we discussed the network underlay as a choice between public 5G and private 5G. In the Retail edge scenario section, we further discussed using network slicing for specific locations and use cases. Both of those are good examples of possible network underlay approaches. In commercial and residential deployments, various combinations of Ethernet and Wi-Fi are typically used for **local area networks** (**LANs**) in conjunction with repeaters as needed to extend coverage and prevent dead spots. In agriculture and other outdoor deployments, they may use a combination of LoRaWAN, Wi-Fi, and Bluetooth, or all three in a shared mesh.

However, these approaches potentially expose machines, hosts, and applications to each other and could allow traffic from one to interfere with traffic from others, in addition to traffic from other connected networks and the internet. To isolate one network or subnet from another, and to limit what traffic can pass between them, firewalls are typically used. However, this introduces configuration and maintenance overhead and can potentially block communications, which prevents distributed applications from functioning unless exceptions are introduced in firewall rules (which provides another potential attack vector). But what if machines did not open inbound ports for requests, and what if applications were not directly reachable by external sources?

## The network overlay

[Chapter 4](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_04.xhtml#_idTextAnchor073) briefly described both the network underlay and the network overlay (which is the virtual network layer on top of the physical network infrastructure). The virtual aspect of an overlay allows the network to connect to thousands of edge devices quickly without needing to interact directly with physical network components. Devices on overlay networks are interconnected via logical links, making up the overlay topology. It is important to note that overlay network topology will vary based on the underlay network architecture.

By extending Figure 4.3, we see that edge devices can communicate directly with the networking components in the overlay, rather than overload the physical components in the underlay (see Figure 9.5). Another advantage of an overlay network is that because it is software-defined, it scales well.

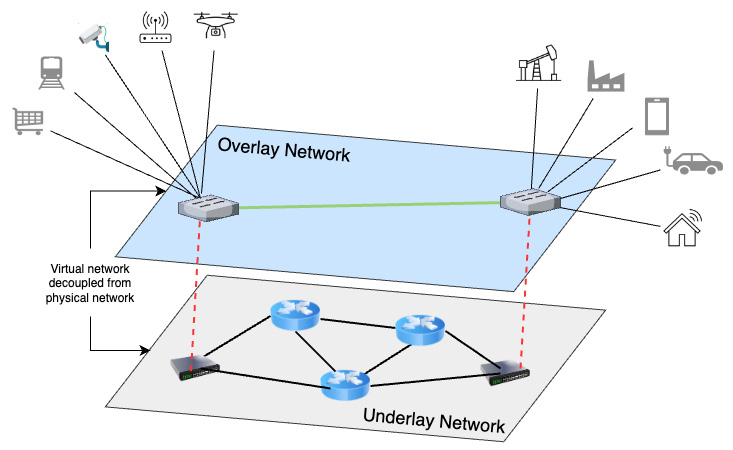


Figure 9.5 – Edge devices connecting to network overlay

With the distinctions between the network overlay and underlay explained, let’s revisit Zero Trust in light of each.

## Zero Trust Network Access

**Zero Trust Network Access** (**ZTNA**) is an emerging approach that authenticates users and devices, authorizes their secure access to each resource independently (other users, devices, services, applications, and infrastructure), and validates that access on a periodic and continuous basis. Like most Zero Trust approaches, it uses the principle of least privilege and defaults to denying all access before authorizing the context of the specific request. See Figure 9.6, which shows how ZTNA relates to the various actions and participants:

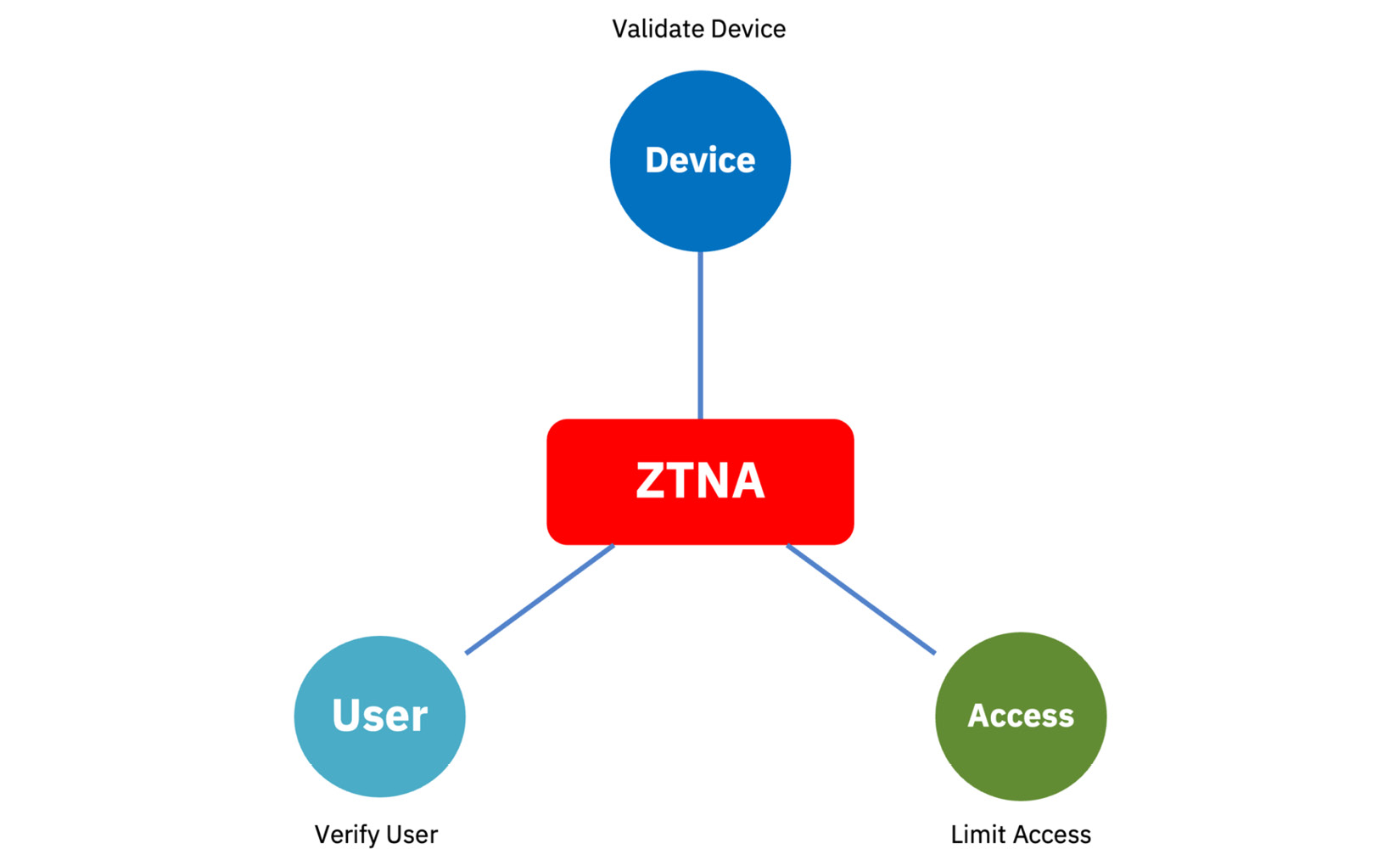


Figure 9.6 – ZTNA in edge computing and how it relates to system components

The ZTNA approach is well-suited to an edge computing solution because it provides a single, general-purpose solution for the wide range of heterogeneous edge devices that cannot be trusted but are generating and sending data to edge hubs. Since these edge devices cannot be locked in a secure location, it makes sense to enable access control policies on the edge devices and set up security policies at a more “central” location. A goal is to remove the implicit trust of those edge devices. While network teams bear responsibility for configuring elements in a zero-trust network, they should work closely with security teams to develop the overall ZTA. Incidentally, ZTNA 2.0 offers continuous trust verification of the devices and protects all data.

While there are many benefits to using ZTNA, it is not a complete security solution for edge networking. Solution architects must take a holistic approach and recommend SASE, which helps bring security services to WAN topologies such as those in edge computing.

After ensuring that all component services and users are connected and communicating with each other, it is critical to also confirm that only the intended participants in the conversation can read the messages being sent.

## End-to-end encryption

When we talk about connecting, the next most obvious question turns to security. Enterprises always want to know whether the connection is secure. That is because a secure connection guarantees that the data flowing through that connection is encrypted, thus protected from promiscuous viewing by others on the same network segment. The connection can span many network hops, which are points or segments in the network along with data is passed.

**End-to-end encryption** (**E2EE**) means that the data is encrypted at the source and remains encrypted in transit until it reaches the intended destination. To put this in an edge computing context, the data generated by the edge device is immediately encrypted and transmitted over a secure network to the edge hub or even to the cloud, where, depending on the need and the application, it could be decrypted and acted upon or stored in the encrypted state. This requires an encryption algorithm at the source and a corresponding decryption facility at the destination before the data can be used. See Figure 9.7, which illustrates a hypothetical flow in an end-to-end physical solution:

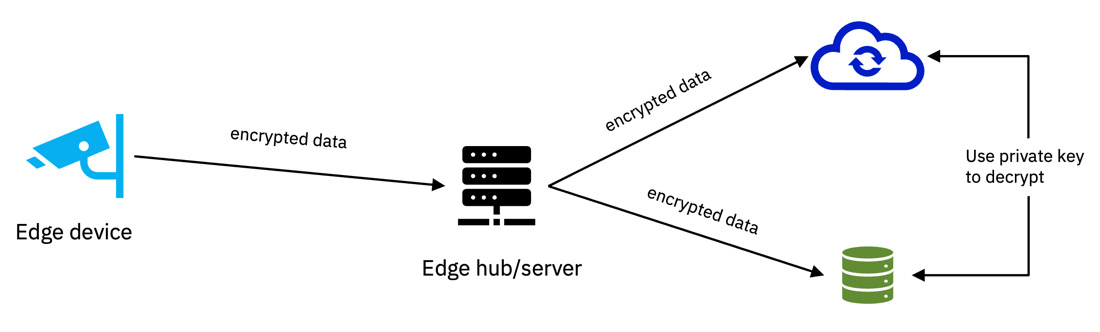


Figure 9.7 – E2EE in edge computing

Remember, E2EE ensures that only the sender and the receiver can see the data in its unencrypted form. There is no entity in the middle, such as the cloud provider or network operator, that can see the data using any kind of key. While encryption comes with a minor cost and inconvenience, sending unencrypted data over an unsecured network can be far more dangerous and expensive. Solution architects have to weigh up those factors while determining when and where encryption should be enforced, especially E2EE.

*NOTE*

*Gartner defines encryption in the context of network-based communications as “*the process of systematically encoding a bit stream before transmission so that an unauthorized party cannot decipher it*” (*[*https://www.gartner.com/en/information-technology/glossary/encryption*](https://www.gartner.com/en/information-technology/glossary/encryption)*).*

To connect the concepts of known and authorized participants with the goal of ensuring that only those parties can read the messages being sent, let’s add the concept of application isolation in a distributed solution.

## Application-centric networking

Dynamically creating and managing network connections between applications, microservices, and infrastructure is a new capability provided by modern software networking solutions, which some have described as **application-centric networking** (**ACN**). Furthermore, the most comprehensive ACN solutions support endpoints from hyperscalers and cloud-based SaaS offerings, as well as enterprise, regional, and edge clouds, thus providing **hybrid**, **multi-cloud** application-centric network overlay connectivity in an application-centric model.

ACN first and foremost requires understanding the application’s connectivity requirements in the form of the following:

* What it needs to connect to – local or remote services and infrastructure
* How it needs to connect – protocols and ports
* Prioritized performance targets – latency limits and response times

Then, it needs to understand the current network overlay and underlay topology and utilization and to be able to construct a model of that state. It also should understand the available deployment endpoint locations: providers, namespaces, cost, performance, and access methods. Lastly, it should have the ability to both observe the performance aspects of all of those in near real time and to deploy infrastructure in the form of gateways at each endpoint location and waypoint gateways at network interconnect points.

With the aforementioned capabilities, especially if they are integrated and can operate autonomously, a distributed application can react to changing network conditions and dynamically redeploy services from one endpoint location to another with no downtime. This is the promise of ACN for edge-based applications accessing remote resources in hybrid, multi-cloud endpoints.

# Summary

In this chapter, we discussed how an application-centric approach to connectivity could increase application deployment velocity by orders of magnitude, thus enabling freedom of movement for applications, services, and software-based infrastructure. Furthermore, if this approach is implemented with a tool using declarative configuration and autonomous implementation, frictionless and reactive application component movement is possible in minutes instead of hours or days.

We covered security frameworks and paradigms that have recently emerged to handle zero-trust scenarios. These standards ensure that all connected components and systems are proactively identified and authorized for granular access.

We also covered the benefits of decoupling edge applications from the network underlay through an overlay abstraction. This allows application isolation from other network users and systems. Consider using approaches provided by application-centric multi-cloud networking connectivity or platform engineering solutions. These new and emerging practices will give your edge architectures the flexibility needed to rapidly adapt to changing environments while maintaining a secure posture.

In the next chapter, we’ll discuss how enterprises can implement the newest software solutions in open source projects. We’ll show how companies can contribute to these projects to ensure their needs and requirements are met without giving up intellectual property. We’ll also cover how open source licenses can be abused, and how to protect your company from that risk.

# 10

# Open Source Software Can Benefit You

Modern enterprise applications are typically 76% based on open source software (<https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html>). This chapter discusses strategies to ensure that these dependencies don’t become your Achilles heel. Several recent well-publicized events have shown how companies that do not actively and continuously embrace open source principles can cause issues for those who depend on the software they maintain. In this chapter, we will discuss how mature open source software projects with open governance can attract more resources to your product development efforts and act as a force multiplier.

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

* Open source and edge computing – the benefits and trade-offs
* SBOMs are your friend – securing the software supply chain with a Software Bill of Materials
* What makes you so special? Contribute without giving up intellectual property
* Let the cat out of the bag – successfully open source your code and documentation

By the end of this chapter, you will have learned how you can consume open source software while avoiding potential pitfalls. You will also read how enterprises have successfully contributed software to open source projects without giving up their intellectual property rights.

# Suggested pre-reading material

* Why do enterprises use and contribute to open source software by Dan Whiting (<https://www.linuxfoundation.org/blog/blog/why-do-enterprises-use-and-contribute-to-open-source-software>)
* Creating an open source program office (<https://ospo-alliance.org/ggi/introduction/>)
* OSPO 101 training (<https://github.com/todogroup/ospo-career-path/tree/main/OSPO-101>)
* Open Source Initiative approved licenses (<https://opensource.org/licenses/>)
* SPDX license identifiers (<https://fossa.com/blog/understanding-using-spdx-license-identifiers-license-expressions/>)
* Developer Certificate of Origin (<https://developercertificate.org/>)

# Open source and edge computing

In this section, we’ll go over the benefits and trade-offs for companies relying on **open source software** (**OSS**) projects to supplement, support, and standardize their software product development process. You’ll learn about how to build support in your company to use and contribute to open source foundations, and you’ll become an advocate for this decades-old approach to collaborative endeavor.

## Edge computing and OSS are intertwined

Edge computing, as it is now implemented, first began in earnest in late 2015 as an outgrowth of cloud computing (see <https://dzone.com/articles/a-brief-history-of-edge>) and so is much younger than open source software development. However, it began to hit critical mass in late 2018, just as the Linux Foundation began organizing its hundreds of open source projects into themed umbrella groups around natural affinities (meaning, categories naturally organized around a shared industry, technology, or market). Thus, LF Edge began in January 2019 with two mature projects and several new code contributions, and the Eclipse Foundation followed suit with its Edge Native Working Group. This meant that by the time that edge computing began to enter the public consciousness, a vibrant group of open source communities already existed as a nucleus to form standards around and to house blueprints for end-to-end solutions.

Edge computing, therefore, became the first modern open source native field in computing. The first software solutions were all developed as open source projects, and then commercially supported solutions downstream from those were created. Thus, the roots of edge computing are firmly grounded in OSS, and any changes or innovations in OSS will have a correspondingly large effect on edge computing as a whole.

Let’s take a look at how this heritage of open source affects your software development velocity and overall effort and costs.

## Do you really need to create that component?

The first OSS projects contributed to LF Edge revolved around utility-type component solutions that solved recurring problems and replaced large chunks of boilerplate code. They also implemented an edge-native approach to problem solving, unlike the **IoT** (**Internet of Things**) solutions from the previous 5–10 years that were all in the process of being converted to **SaaS**-based products. This points to the first major benefit of relying on OSS – you don’t write and maintain code that is not your company’s core competency and is not part of your product’s value proposition. This adheres to the **don’t repeat yourself** (**DRY**) principle and the “don’t re-invent the wheel” idiom.

*SOMETHING TO THINK ABOUT*

Only create software components that implement your product’s core value proposition*.*

If a problem has already been solved, the solution has several adopters already using it in production, and it largely meets your needs, then how does it benefit you to invest the time and effort in attempting to solve the same problem yourself in isolation?

* The first instinct of a solution architect should always be to look for an existing component to reuse. Therefore, the first benefit of using open-source software solutions is that it provides a rich vein of re-usable solutions and components.
* The second major benefit of relying on OSS is being able to create solutions quickly through judicious re-use of existing components. Since re-using code is quicker than writing it from scratch, your project should take less time to write and debug, resulting in better code quality. This assumes that the code you re-use already has unit tests and complete code coverage, supports your coding standards, and its integration requires minimal effort.
* A third benefit of relying on OSS is that your code will inherit support for de facto and/or actual standards and conventions. The pattern that has emerged over the last few decades of software development is for a developer or tool to implement an approach, which is then for refined and improved, and then competing approaches emerge and either gain support from users of the previous approach or fail to gain significant traction, thus reinforcing support for the existing approach. Over time, the existing approach becomes the de facto standard, and it may even have the unique distinctives of the solution codified into an actual standard. We’d argue that any existing open source project that is being used in production by several products or companies has become a de facto standard.
* A fourth benefit of relying on OSS is that your product will be easier to maintain and have a shallower learning curve for new developers being onboarded the more you reuse existing OSS components. In many situations, developers may be familiar with existing standards and the libraries or components that implement them. Further, there will be an existing body of work (e.g., tutorials, videos, example code, IDE hinting, and autocomplete) available that demonstrates and documents how to use those components, and they will likely have been refined to follow familiar conventions and paradigms, which allow them to converge on the familiar (meaning, this process is likely to produce a solution that will look familiar to architects and developers when they examine the details for the first time, therefore making it intuitive to understand and thus easier to maintain).
* A fifth benefit of developing software and features as open source is that the burden, risk, and benefits of non-differentiating code and functionality can be shared between partners and even between competitors. This resource sharing lowers costs and, thus, also frees up resources to be better used elsewhere.
* Finally, a sixth benefit of relying on OSS is that support for those components can be outsourced, thus creating a smaller support burden for your product organization. This may take the form of referencing existing (external) documentation for those component features, contracting with an organization that provides commercial support for those components, or integrating existing external documentation into your product documentation if the source code and artifacts feature a license that allows this reuse.

Given the benefits outlined here, most organizations will experience more than a net positive experience if they leverage existing OSS components in their software development process. This leads to the next question – how can software architects encourage a culture of open source reuse within an enterprise?

## Creating and supporting an open source program office (OSPO)

Later in this chapter, we’ll cover consuming OSS without introducing new vulnerabilities, as well as contributing to OSS projects while retaining control of your intellectual property. In the meantime, let’s go over the benefits of having a dedicated group in your organization tasked with the responsibility for promoting, influencing, and advising on the best approaches to consume from, and contribute to, OSS.

Your company or organization’s interaction with OSS will be a net positive if, at minimum, there is a defined open source strategy. This strategy should address what consumption is allowed (and why), what contributions are allowed, how these align with the business and product strategy, and how to calculate and achieve a **return on investment** (**ROI**). An OSPO should manage and implement that strategy, either directly or indirectly, through delegation to groups in each business unit.

The TODO Group defines the role of an OSPO as follows:

“An open source program office is designed to be the center of the universe for a company’s open source operations and structure, helping to bring all the needed components together. This can include setting code use, distribution, selection, auditing and other policies, as well as training developers, ensuring legal compliance and promoting and building community engagement. The office can also provide advocacy and communications about all things open source inside and outside the company.” (<https://todogroup.org/zh-cn/resources/guides/how-to-create-an-open-source-program-office/>)

Let’s look at an example of how this is implemented in a large enterprise today. Inside IBM’s **CIO** (**Chief Information Officer**) office, the Open Technology team functions as an OSPO. They are responsible for managing annual developer training (which is required for all open source contributions), tracking developer contributions, and maintaining company-level agreements with open source foundations. The team gives talks both externally and within the company and has both developer advocates and project leaders active in many open source projects. They identify and track consumption of open source software across the company, facilitate collection and reporting on the software supply chain, including SBOM creation and usage, track and report on vulnerabilities, and ensure license compliance through regular code scans. They also maintain a short list of open source projects of strategic importance to the company and direct company-level investment in them.

Company divisions and units can also create their own open source or open technology strategies and align with a company-level strategy. Finally, larger umbrella organizations within foundations may have corresponding steering committees within a company to collaborate and coordinate on a unified, cross-unit approach to working within, and supporting, projects based on open-source software and technologies (<https://blog.opensource.org/the-five-stages-of-the-open-source-program-office/>).

As you can see in the preceding example, overall guidance is provided from a central group implementing the corporate open source strategy, while divisions and units have the flexibility to determine the specifics and align those with their business objectives. This approach enables standardized company-wide safe consumption of open source solutions. With a central team providing tools such as code scanning, coupled with individual units providing dependency graphs in the form of a software bill of materials, vulnerabilities can be quickly identified across an enterprise. Let’s explore how this is possible in the next section.

# A software bill of materials is your friend

In this section, we will explore OSS supply chain issues – how to consume or depend on OSS without introducing new vulnerabilities using a **Software Bill of Materials** (**SBOM**), how to identify mature and stable OSS projects, how to nurture and assist projects you rely on, and responding to projects that abandon their initial commitments.

## Using SBOMs to track software dependencies

A software architect can be called on to provide specific recommendations for OSS solutions in an architecture or implementation. The risks in doing so are that the proper solution (or one of its immediate dependencies) for a given set of requirements might be from a project that is not mature or stable (see the following subsection for more about that topic). Therefore, it is incumbent on you to perform due diligence, either recommending against using those projects or only with strong caveats when no other good options are available, so that the ultimate decision-makers can make an accurate and informed determination.

Here are some tasks you should perform in order to analyze a set of dependencies for potential areas of vulnerability. This list should be considered the minimum due diligence, not an exhaustive set of steps. Think about how you might expand the following list below to cover your circumstances or unique situation:

1. The first step in performing an analysis of an OSS project’s release is to obtain the SBOM files for both the source code and the released assets. It is critical to use both because code will rely on different dependencies at build time compared to runtime. The project may make SBOMs available for their source code in GitHub repositories. See Figure 10.1 for an example of where to look. Likewise, an SBOM can be easily generated from a container using Docker’s CLI.

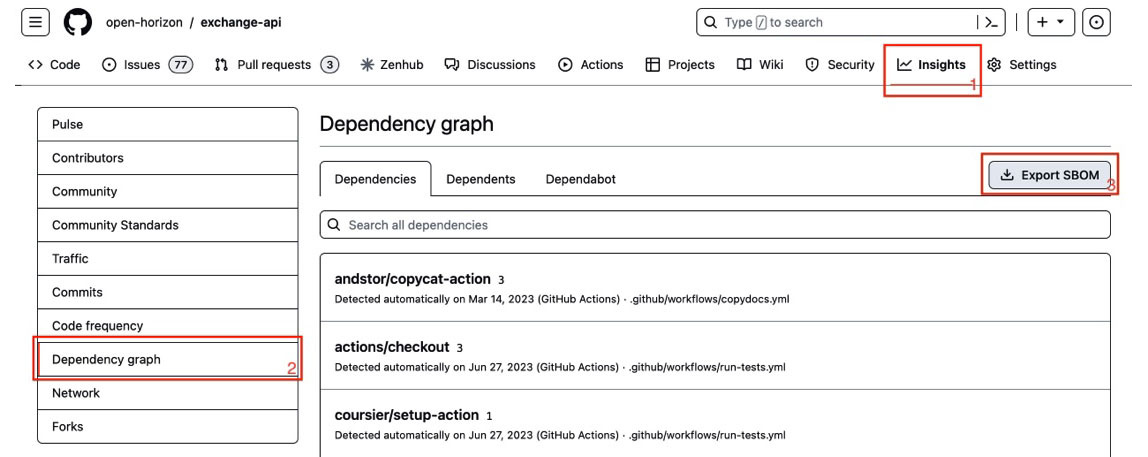


Figure 10.1 – A GitHub repository screenshot showing the Export SBOM function

1. The second step would be to review the SBOM contents, looking for critical, unaddressed vulnerabilities more than 60 days old. If you find any, it could mean that the project code is not maintained per the recommended standards from the **OpenSSF** (**Open Source Security Foundation**) if the associated **CVE** (**Common Vulnerabilities and Exposures notice**) is legitimately filed and properly describes a critical vulnerability that directly affects how code is used in this situation.
2. The third step would be to look at the licenses of the immediate dependencies to ensure that they meet your needs and requirements. This should include defining which licenses are compatible with the parent project, ensuring that only compatible licenses are used by dependencies and in contributions. A recommended resource to consult, if you have any questions about a software license, is the **Open Source Initiative’s** (**OSI’s**) list of OSI Approved Licenses at <https://opensource.org/licenses/>.
3. Finally, a fourth step would be to manually review the list of projects and repositories to see whether any of them have appeared in the news recently or on a trusted technology site, regarding any upcoming changes in license, unaddressed breaches or vulnerabilities, or community controversy or instability. If any of those names trigger a red flag or give you pause, consider investigating it more fully.

*NOTE*

*At the time of writing, the authors were not aware of a tool that could automate all steps of the scan process outlined previously, but keep in mind that new tools may have been created or released since then.*

### Let others share your load

As you consider all of the work that needs to be done managing source code contributions and dependencies, keep in mind that organizations exist that will assist you with these tasks. Open source foundations such as the Eclipse Foundation, the Apache Foundation, and the Linux Foundation perform some or most of the preceding steps on behalf of developers, or may provide the tools needed to do so. For example, the Eclipse Foundation’s **Intellectual Property** (**IP**) team will perform dependency license checks when accepting and onboarding a new project, and before accepting sizable contributions to existing projects. Also, these tools and services provided by the foundations are one element in a more comprehensive approach to managing cyber risk (<https://www.eclipse.org/org/workinggroups/eclipse-cyber-risk-concept.php>).

Now that we’ve reviewed code quality and risk management, let’s look more closely at how to gauge an OSS project’s overall maturity and how that relates to code quality.

## The characteristics of a mature OSS project

Another aspect of OSS solutions that a solution architect should evaluate before making recommendations is a project’s maturity and stability. The risks of using software from a source that is not yet mature would be that the project could suddenly change direction by removing critical features, adding unneeded features that cause performance issues, stopping the maintenance of documentation, or ceasing development work entirely.

If a component of your solution were to become unreliable in this manner or otherwise unavailable, how quickly could you replace it, and at what cost? Let’s see some questions you should ask about a project that may require deeper investigation, before potentially adopting it for your solution or product:

* How is a community defined, and how large and active is it? Some projects that offer their software under an open source license believe that allowing anyone to submit or comment on an issue constitutes community engagement. Others feel that providing a forum for software users to communicate (e.g., chat rooms, discussion channels, and documentation comments) is sufficient. However, allowing the public to communicate with you is not the same thing as a group of **empowered stakeholders** giving input into a product roadmap or contributing code, designs, artwork, and documentation freely to a project. Just talking about a project is not the same thing as contributing to it. We would argue that a community should be comprised of persons who not only use a project but also meaningfully contribute to it, with their contribution acknowledged.
* How many adopters are there, and how long have they been active (are they still using it)? An indicator of not only the health of a project but also its value is the number of persons or organizations who actively use the solution. If those users are tracked and are willing to be publicly identified as users, then that is a strong signal of the intrinsic value the solution brings.
* How many companies participate in the community? If a company is not only an adopter but also actively contributes in some way, that indicates that the company finds strategic value in the solution and finds that using the software brings more benefits than developing something similar in-house. It also indicates that those running the project welcome the participation of others. If companies use project software but do not contribute back, it’s possible that they are prevented from doing so, not necessarily that they are unwilling to.
* How often and how regular are code releases? Is there a published feature roadmap? Can the community submit feature requests? If a project has regularly scheduled releases every six months or less, this is a good indicator of a stable project with a history of discipline and the ability to follow a schedule. If those releases are being documented with an artifact developed by the community, such as an architecture decision record or a feature request, so much the better. Also, if you can see a history of detailed release notes, their development and release processes are likely sufficiently mature.
* Is the project leadership composition vendor-neutral? Are there regular elections from the community, or are leaders appointed by a company or small cadre of “insiders”? A key indicator of potential trouble in a community is when it is known for having a single leader or small group of persons who control all the decisions. While having strong opinions on how a task should be performed, or what a specific outcome should be, is not an issue here, see whether there is a track record of community requests being consistently overruled or ignored. If so, this could become an issue.
* What is the quality and coverage of the documentation? Documentation is typically the least-enjoyed task for volunteers on a project, so if the documentation is thorough, up-to-date, clearly written, and complete, then the project is likely healthy and mature.
* Is there a good amount of unit tests available? What is the code coverage like? Can you fork the repo and run an end-to-end test without it failing? Failing tests can be a sign of code rot and insufficient attention or a lack of resources. However, this should not be a disqualifying consideration if you are willing to work to remediate this deficiency.

In summary, an immature project may lack a (complete) governance structure, with leadership positions filled with experienced staff, designated or identified replacement staff for critical roles, active community members from several organizations, and consistent contributions from people of varying abilities. Open governance, where meetings are held publicly and recordings/minutes are published, and where a community elects its project leadership and holds them accountable, prevents unstable communities and “benevolent dictators” from controlling a project.

Try to avoid using any software from an immature or unstable project, or even better, get involved and work to remedy any shortcomings. Here are some thoughts on how to do so effectively.

## How to nurture and assist projects you rely on

More than just avoiding using software from unsustainable projects, your organization (and likely you) should identify which dependencies are strategic to your product or solution, and then actively contribute to the project that maintains those dependencies to ensure their ongoing success. This is especially true for emerging fields such as edge computing.

The most important step in supporting a project is to get formally involved in a community. This may mean reaching out to the leadership of a project, and the parent foundation if there is one, to let them know that your organization finds value in their project and your intention to begin using it in production (becoming a project adopter). As you interact in community events and meetings, find out what they need – attend meetings, read their issues and product roadmaps, email or message team or group leaders, and ask them about the challenges they face. Eventually, see how you can dedicate resources and persons to a project and earn your way into leadership positions by contributing code (becoming a committer), squashing bugs, and eventually, reviewing commits (becoming a maintainer).

Once your enterprise is established in a community, it’s time to use your resources to improve project organization by amplifying what the community is already working on, and by helping out where it requires the most assistance. Research and find out how the community is funded, what its budget is, and how well it stays within that budget. See what other contributions organizations have made. Ask what initiatives the community wanted to tackle but couldn’t, and work with it to address the initiatives that will bring mutual benefit.

Another area where you can bring value is by spreading the message of how a project has brought you and others value. Discuss publicly how the project has helped you – publish articles, videos, refer others to the project, and volunteer to teach others. Be a developer advocate. See whether you have connections in your social network that you could introduce to project leadership, especially in areas or verticals that they would like to expand their reach in.

One of the most neglected areas in OSS projects is test coverage. Offer to assist with testing code, testing steps in documentation, and submitting fixes and patches in the requested format. This will go a long way toward improving code quality and catching bugs before they are included in a release.

Finally, personally offer to join the leadership of the project, whether actively chairing a group or advising from a board position. These simple but effective steps don’t all cost money, but they are welcome in OSS projects and can make a big difference in the health and stability of a project. They also act as a counter to pressures within a project to stray from the original mission statement, or ignore foundational promises, as we’ll discuss next.

## Responses to projects that stray from their mission

Over the last few years, and increasing in frequency over the last year, companies that started out monetizing a commercial offering downstream from an open source project that they founded have migrated the license of the previously open source code to a more restrictive source-available license. This action provoked emotional responses in those consuming the code, ranging from shock and surprise to feelings of betrayal. Yet the founders did not do anything illegal, and they were trying to preserve a business that they had created from the ground up. Conversely, they could not have built their projects without the wealth of open source dependencies that they rely on.

What responsibilities did the founders have toward those who consumed the open source code, whether an actual community or not? And what backup plans should the consuming parties have had in place for this eventuality? What follows is purely the opinion of the authors and does not constitute legal advice. Always consult with legal counsel before acting or publishing potentially defamatory statements.

First, talk to the founders rather than gossiping or spreading hearsay. It’s possible that you misunderstood the situation or didn’t hear the whole story. Reality can be more complicated than most of us imagine. Also, it’s possible that the situation is resolvable. In many cases, it can be done in a manner not originally anticipated.

Second, determine whether the situation really needs addressing, or whether it can be simply ignored. If it cannot be ignored, then consider what actions can be taken in response, up to and including creating a fork of the project or establishing a competing project or alternative. Keep in mind that most forks or competition typically fail to reach their objectives. Reach out to others who may also be affected, and see whether you can collaborate on an alternative. If you cannot, then that likely means that creating an alternative is not a good option.

Frederic Desbiens, program manager for IoT and edge at the Eclipse Foundation, sums up the situation regarding companies that migrate the licenses of their projects away from open source in this way:

“As such, they are breaking the dynamics of the open source ecosystem to their own benefit. Can you imagine the OSS ecosystem ten years from now if the licenses for the Linux kernel, GCC, or even bash were changed in the same way? There is a lesson to be learned here. Organizations valuing a healthy open source ecosystem should contribute to projects where contributions are accepted without copyright assignation. For example, the Eclipse Contributor Agreement leaves copyrights with the individual or organization making the contribution. This makes license changes such as the one you describe much harder since they would require consent by all copyright holders.”

After discussing the SBOM, the dependencies that arise from it, and how to evaluate those, let’s now turn to the topic of when and whether to open source your own solutions, or components of it. We’ll also discuss aspects of contributing code to external OSS projects.

What makes you so special? (Contributing without giving up intellectual property rights)In this section, we’ll go over strategies for contributing to open source software projects. The focus will be on protecting your intellectual property and copyrights while enabling sharing and collaboration. What follows are the opinions of the authors and does not constitute legal advice. Always consult with legal counsel before contributing to open source.

## Common objections

When the authors consult with teams and companies about an open source strategy for their edge computing solutions, one of the first objections that companies invariably raise is a concern about “giving away” their intellectual property or competitive advantages. The concern appears to be that placing their code in the open gives their competition insight into the “secret sauce” that gives their products a competitive advantage, and that others will now be able to easily replicate their success.

It is the authors’ opinion that any innovation worth protection should have a patent disclosure filed in advance of code being contributed or committed. One of the key factors identifying an invention worthy of this protection is that the idea is easily discoverable, although innovative. By contributing code, you help fulfill the discoverability requirement. Thus, contributing code to an open source project may strengthen your company’s case, not weaken it.

## Recommendations for contributing code

Additionally, many successful companies have built thriving businesses around open source projects. Well-governed OSS foundations have led the way by providing all of the recommendations in the following list. These recommendations cover contributing code, documentation, machine learning models, and other assets to open source projects while minimizing potential risk to profitability and competitive advantage:

* Patent protections and corresponding open source licenses should be harmonized. You should communicate with your patent attorney when you file your disclosure(s) and ensure that they know that you may create and contribute open source artifacts that implement your invention(s), choosing an OSS license that allows usage to that effect.
* All contributed files should contain an appropriate copyright notice. This copyright may not be legally removed, although others may append their own copyright lines as they amend or add to the files.
* Different files may be covered by different licenses. For example, computer code may be covered by an Apache 2.0 license, while the documentation could be licensed under CC BY 4.0 International. Ultimately, you should use a **Software Package Data Exchange** (**SPDX**) license identifier or expression in all file headers.
* All contributions should be signed with a **Developer Certificate of Origin** (**DCO**) when committed. This ensures that the contributor claims that they created the code and that they submit it under the specified license.
* Every organization should have a **Contributor License Agreement** (**CLA**) with the organization they contribute code to. The CLA describes the terms of contributions and protects OSS projects from potential future legal issues. These could include claims that a contribution was not authorized for a specific purpose, or that the contributor was not authorized to make the contribution.
* Machine learning models are a special case, as you should be concerned not only with the license of the resulting model but also the provenance of the data used to train it. Training data should also be scrutinized for potential bias so that the resulting model does not unintentionally weigh or exclude relevant factors, thus invalidating the results.

The preceding guidelines are an overview of the issues, concerns, and recommendations around protecting your property while collaborating with others in the context of an OSS project. If your organization has an OSPO, they will likely have a more complete set of guidelines and processes for you to follow. Also, if you don’t have an OSPO in your organization or enterprise, we hope you now see the need for one to be created. Finally, consider working with an established OSS foundation if it meets your needs and goals.

Now that we’ve discussed how to protect your ideas and inventions while collaborating, let’s jump into the specifics of how to actually make the contributions.

# Let the cat out of the bag (Successfully open source your code and documentation)

In this section, we’ll discuss different methods and approaches to open sourcing your materials, from the bare minimum to standard and formal approaches. By the end, you should have a good idea of your options and the benefits of each one.

Each of the options listed in this section share some common assumptions. Chief among those assumptions is that each repository should explicitly call out (meaning, draw attention to or otherwise direct the reader to) the main or default license. Do not make code available publicly without specifying a software license. Other assumptions are that each repository lists the maintainers, contributors, security reporting process, ways to contribute, and adopters. Finally, each repository should contain a **README** file (meaning, a file literally named “README” that contains information that should be consulted first) and a pointer to any other documentation.

## Five options for open sourcing

The first and simplest option is to place source code in a public repository. This allows an organization to make the code available without the overhead of creating a formal project or concerns about building a community. In some situations, this option allows a company to “test the waters” and see what happens as a tentative first step toward a more robust and formal approach. However, ultimately, this is not a sustainable approach longer-term.

A second option is to find an existing project compatible with the aims of your code and either contribute it wholesale to that project, or incubate your code as a sub-project under the watchful eyes and mature supervision of a parent project and community. This option can be a shortcut or quick way to build an existing community, test the resolve of the contributing organization toward building a sustainable project eventually, and safely create the internal structures and processes for your eventual project while the intended leadership learns the ropes.

A third option would be to find an open source foundation that your company can work with to assist with the legal, structural, marketing, and process issues of creating a new project from scratch. They can assist you with all of the work needed to form the project and get it started, including setting up meetings, mailing lists, presentation materials, trademarks, logos, press materials, and publicity. They may also have existing services, including repositories, registries, CI/CD tooling, and website hosting.

A fourth option would be to forgo the formality of creating a project and community while associating the code with your company, by hosting it in a company-affiliated repository. This gives people the confidence in knowing who backs the code and that you may likely have a vested interest in the code’s longevity, ongoing maintenance, and support. However, there is also a degree of uncertainty, since there’s no guarantee that the company won’t eventually change the license or halt funding and development of the code, thus effectively abandoning it.

Out of all the options listed here, the second and third are approaches used by the modern open source movement and foundations, and they most closely match the open governance model. Consider the benefits and drawbacks that we have outlined here when determining which of the options may work best for you.

Next, let’s briefly touch on the topic of what makes a good candidate for open source.

## What to open source

The preceding sections lead straight to this question – what artifacts, solutions, or application source code should be released as public open source? Given the large number of existing OSS projects, software architects shouldn’t advocate for releasing something new unless it is of potential value to others. Here’s the authors’ take on what may be of value and interest to others, listed in order from most valuable to least:

* **Most importantly, working and supported code that solves a problem**: This code should build successfully in all stated environments, provide test coverage, and at a minimum, instructions on how to use it.
* **Code snippets, utilities, and examples**: These may not build on their own but are designed to demonstrate a concept or approach to a solution. They are most helpful when accompanied by robust comments or documentation.
* **Pseudo-code, explanation, and documentation**: These are not technically source code but are artifacts that directly or indirectly teach concepts and the usage of code.
* **Raw data, lists and tables, processed and annotated data, and information**: These can be used to provide input, training, and functions as a source of transparent data publishing.

Conversely, what is not likely to provide value when released as open source are the following:

* **Code that is under active development and does not function as-is**: If it does not work, the code should either be in a private repo or not merged into the main branch. The expectation is that public code made available in a main branch should be capable of being built and run.
* **Broken projects that used to work but do not anymore**: Code rot happens when it is not actively maintained. Projects in this state should be archived or deleted if nobody can be recruited to maintain them. An exception to this would be for code of historical value, especially when the runtime environment is no longer available, but this should be clearly stated in documentation.

In summary, consider releasing open source functional code that offers a solution and runs independently, not pieces or parts of a solution or incomplete code that requires other unavailable pieces to run. Code should build and run in all supported environments and have automated passing tests with each release, which are ideally regular. Abandoned or untouched or no longer supported code should be archived. If you don’t want to receive and answer emails about code, archive it.

# Summary

In this chapter, you read about four key topics related to using open source software in the emerging field of edge computing. The first section talked about why open source is so important to edge computing and hybrid cloud development. We also discussed the potential benefits that using OSS brings to solutions, including increased development speed and agility, support for standards, shallower learning curves, and easier outsourcing of component development.

The second section discussed the software supply chain and how to manage it more easily with an SBOM. This gives you the ability to track dependencies, and it exposes additional information that could lead an architect to identifying potential risks in the projects supporting those dependencies.

The third section then covered reasons to open source company-developed solutions or components. You learned about ways to contribute safely without adding risk. Finally, the fourth section actually delved into how you can open source artifacts, and when you shouldn’t.

If you’re reading the book chronologically, you should by now have a holistic view of edge computing architectures and archetypes, system components, best practices, and software contributions and collaboration. That leaves a few remaining topics that function as a sort of connective tissue between those areas – how you can architect solutions to respond to worst-case scenarios, and how to make application architectures that become more resilient when unplanned situations present themselves. We’ll cover these in the final chapter.

# 11

# Recommendations and Best Practices

Once you’ve understood and internalized the concepts, terms, and patterns of edge computing, this chapter delves into the pitfalls that we’ve experienced and shows you how to avoid making the same mistakes. The three main areas we’ll cover involve practices that have been developed specifically for the edge-native environment, how to think about designing solutions that don’t easily break but actually improve when they encounter adverse circumstances, and what to do when things go awry.

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

* Edge-native best practices as an outgrowth of cloud native
* Make antifragile applications
* When things go wrong

By the end of this chapter, you will have gained a new perspective on how to architect long-lived and resilient solutions for edge environments. You will understand how to put these recommendations and guidelines into practice. You should also be able to guide developers and line-of-business executives appropriately by explaining how these principles will improve the solution and thus make it easier and less expensive to maintain.

# Suggested pre-reading material

* Antifragile: Things That Gain from Disorder by Nassim Nicholas Taleb
* Edge-native development best practices, IBM (<https://www.ibm.com/docs/en/eam/4.5?topic=clusters-edge-native-development-best-practices>)
* Why Software is Eating The World by Marc Andreessen (<https://genius.com/Marc-andreessen-why-software-is-eating-the-world-annotated>)
* The power of proactive repair by Addy Osmani (<https://www.linkedin.com/feed/update/urn:li:activity:7131581009191456768/>)

# Edge-native best practices as an outgrowth of cloud native

Cloud-native development best practices were initially formed from the convergence of adoption of the agile development process, learning how to effectively utilize shared infrastructure, and service-oriented architecture (as well as software eating the world). However, not all best practices from the cloud apply to the edge. Specifically, partially and completely disconnected operations, heterogeneous hardware and software environments, and the potential mobility of edge devices forced edge pioneers to take a hard look at what was learned, discard the lessons that no longer applied, and adopt new lessons, culminating in edge-native development best practices.

In [Chapter 1](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_01.xhtml#_idTextAnchor013), we discussed the edge-native programming model, in the Cloud out versus edge in section. In this section, we’ll dive into three areas where those fundamental assumptions affect how we architect solutions. We’ll discuss how an inversion-of-a-central-control-plane approach can be a more secure method for deploying applications to the edge, learn about the concept of dynamic runtime dependency resolution, and weigh the relative advantages of application deployment models.

## Pulling can be more secure than pushing

The cloud-native approach to managing distributed resources was created in an era of cheap and ubiquitous, homogenous cloud computing where resources seemed to scale almost infinitely and cost pennies to operate. The remote systems they were designed to control were heterogeneous in nature and typically resource constrained. This approach to centralized control is exemplified in the smart server/thin client pattern and more recently borne out in cloud-based IoT platform solutions and even web-based and SaaS solutions. For a deeper dive into this topic, see the Legacy IoT architecture section in [Chapter 3](https://learning.oreilly.com/library/view/edge-computing-patterns/9781805124061/B21141_03.xhtml#_idTextAnchor057).

Centralized control operation can present both a security and a management challenge that distributed or decentralized control does not have. Here is a list of key differences and distinctions between the two:

* First, a centralized “directory” of all of the devices must be maintained, including storing credentials for each endpoint, updating them when their addresses change, and periodically rotating credentials. A process must be created and maintained for adding new endpoints and removing retired or re-imaged endpoints.
* Second, each endpoint would have to be listening for an inbound request from the central control solution, which requires an open port and a network route between the control plane and the endpoint. This adds network management overhead and complexity. Also, the open port on each endpoint opens up a potential attack vector for malicious actors.
* Third, if one exists between the control plane and the endpoint, it likely requires configuring those ports and connections on a firewall. As illustrated in Figures 5.7 and 5.8, that firewall likely exists and will need ports opened.

Collectively, this is a lot of maintenance overhead to create and manage information about each endpoint in at least three separate locations: on the endpoint itself, in the control plane solution, and a firewall, with potential custom network routing rules as well. Likely, the creation and maintenance of that information will involve more than one team and is not an automated or automatable process.

This centralized approach to control has been what IT staff have been traditionally used to. Still, we are fast approaching the point where edge-based solutions are less expensive to operate and are automatable. This is causing enterprises to consider re-patriating workloads and subsequently distributing them out to the edge where and when it makes sense.

The newer edge-native approach has been driven by the increasing ubiquity and lower costs of powerful and energy-efficient programmable compute resources available in the field and outside the four walls of traditional data centers. This compute, when coupled with cloud-native development tools and best practices, has enabled new paradigms of distributed control, especially when combined with autonomous solutions and potentially augmented with ML-based intelligence.

Distributed or decentralized control spreads the responsibility out toward the systems that need to take action, thus allowing secure and automated approaches to onboarding and managing edge endpoints.

To contrast this with the centralized approach of edge endpoint management, endpoints in a distributed, pull-based design only need to be authenticated with the control plane and authorized to use it, which means that no remote systems need to initiate connections to the edge node, nor do they need a login account. You can enable an edge node’s access to a centralized control plane by an exchange of keys, thus eliminating the need for the creation of a centralized directory of devices and IP addresses or hostnames.

*NOTE*

*Even with the decentralized, pull-based approach, you may want to consider adding an out-of-band management channel to provide remote access for troubleshooting purposes.*

In the pull-based approach, the endpoints do not need an open inbound port, since all communication can be initiated by the endpoint, and thus will be indistinguishable from “normal” network requests, which are already likely to be configured and allowed on both the device and any intervening firewalls. Thus, no new configurations would need to be created and maintained in this arrangement.

To summarize, pulling eliminates the need to maintain a central directory of endpoints and credentials, removes the security risk of having open inbound ports, and enables automation of the configuration and deployment process by not requiring new network routing rules.

Now that you’ve seen how the edge-native, distributed approach to endpoint management simplifies and secures the deployments, let’s take a look at how it can also simplify application dependency management as well.

## Application dependency resolution approaches

When an edge computing node receives information about a workload, that configuration information would include data about the service to be executed as well as its dependencies. In order to properly initialize and execute the services, they must be started in the proper sequence successfully, thus requiring a dependency management strategy involving both when to resolve those dependencies and how to resolve them.

Managing these dependencies can get incredibly complicated and be a maintenance nightmare (unless you have an automated dependency resolution function), and you need dynamic runtime dependency resolution so that killing or upgrading one dependency doesn’t cause you to need to restart the whole application (even in loosely coupled and/or stateless applications).

Your goal should be to minimize dependencies that affect service availability to the maximum extent possible. Services should therefore be able to work in a disconnected or offline mode, which would allow them to accept requests even if other services they depend on are unreachable.

Managing the service software life cycle of containerized applications with dependencies is difficult, but managing them remotely without the ability to log on to the host is even more difficult. Let’s begin by defining the stages of the life cycle of edge-native applications, shown in Table 11.1:

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Publishing/deployment | This is the act of transporting a container image binary or application package from a repository to the endpoint destination securely. This stage includes validating the application once it reaches the destination to ensure the artifacts match the expected characteristics. It should also include persisting the artifacts locally on the destination so that they remain available through host restarts. |
| Initialization/execution | This stage optionally includes deploying secrets and configuration, then starting the application. There may be separate parameters for the first execution compared to subsequent starts. |
| Operation (updating, monitoring, restarting, rollback, and inspecting) | The operation stage covers updating an application per configuration when a newer version becomes available, inspecting the running state, restarting if it terminates unexpectedly or the host restarts, and rolling back the application to a prior version per configuration. |
| Stopping/removal | This stage covers halting and optionally removing running services. It may include removing resource dependencies that are no longer in use or needed. |

Table 11.1 – Edge-native application life cycle stages

With an understanding of the stages outlined in the previous table, let’s explore how some tools infer service dependencies while others explicitly specify them and their relationships.

### Implicit versus explicit dependency definitions

Some or all of the stages in the service software life cycle as depicted in the preceding table are typically managed directly on the host with tools such as Docker Compose, or in Kubernetes clusters with tools such as Helm charts. An important consideration to keep in mind is that these tools do not allow the definition of explicit service dependencies and that services are run at the execution stage in the order in which they are defined. In other words, they utilize an **implicit dependency approach**, meaning that the order in which services appear in the configuration may imply dependent relationships, but likely only indicate a preference in the order in which they should be initialized.

Other tools, such as Open Horizon, enable an agent running on the host or in a cluster to independently manage the service software life cycle. Open Horizon defines explicit service dependency relationships in service definition files. This allows each service’s dependencies to be initialized and executed before the parent service regardless of the order in which they are defined. It uses an **explicit dependency approach**. An advantage of this approach is that service upgrades can be more granular and that the whole application does not necessarily need to be restarted if one service or dependency is updated or restarted.

### Stateful versus stateless service considerations

A **stateful service** is one in which some or all of the required information used in a request is kept or otherwise persisted between successive invocations. A **stateless service**, on the other hand, sends all required information to service that request in that invocation.

When deploying applications that may have more than one dependency on a single stateful service, ensure that your tools do not duplicate the stateful service. For example, Open Horizon allows you to specify a dependency as a **singleton** to prevent multiple instantiations when multiple services share a single, stateful service dependency.

On the other hand, you may wish to run multiple instantiations of a single service if you need to upgrade one dependency pair independently of another, or for performance reasons if your environment is not resource constrained and you are load balancing between them in a high-availability architecture.

## Deployment models for distributed edge applications compared

In the previous sub-section, we discussed application dependency management approaches, which assumed that the containerized application dependencies were being deployed to, and executed on, a single host. Now, in this sub-section, we’ll investigate ways to deploy applications and their dependencies when they collectively span multiple hosts.

**Tightly coupled microservices** used in edge applications are ones in which resources are shared, interfaces are specific or dependent, or requests are synchronous. In these situations, the application should be treated as a monolith and both tested and released in a single deployment. This approach goes against the spirit of edge-native application development, but there are specific scenarios that may require it, usually involving stateful and transactional applications.

**Loosely coupled microservices** tend to be called asynchronously, encapsulate all dependencies internally, and do not introduce breaking changes outside of a major version update. These characteristics should allow individual services to be updated without scheduling application downtime, and without requiring an update to any other services.

As part of the deployment process, applications should have their performance characteristics compared to a previous baseline to ensure that they do not negatively impact their target deployment host environment or exceed available resources. For example, if a service previously utilized 30 MB of storage and 2 GB RAM, those requirements may have been added as deployment properties for the service and as availability constraints on the edge node. If the update requires more RAM, ensure that any policy properties and constraints have also been updated and that any assumptions about target device resource availability are still valid.

In all of the preceding discussions, the danger being avoided with the solution designs is creating applications that easily fail when operating, maintaining, and migrating. But is there an approach to creating edge architectures that can circumvent or mitigate the tendency toward frailty? More than being reactive and implementing self-healing, can we be proactive and prevent potential breakage in the first place? Let’s explore this in the next section.

# Making antifragile applications

In this section, let’s explore how to design solutions that improve when placed under stress. We’ll analyze potential areas of weakness. By the end of the section, you should be able to list some properties of an antifragile solution architecture. The goal of this section is to give you a new perspective and a new way of thinking about creating lasting and invulnerable applications, rather than delivering specific recipes for you to follow.

## Defining the terms

What do we mean when we use the term **antifragile**? In Nassim Nicholas Taleb’s book Antifragile, he describes systems that not only endure and survive adversity but improve under pressure and attacks. While not written specifically about designing resilient application architectures, the principles covered in his book apply generally to most systems, including those used in software, and that’s how we’ll be applying the lessons discovered and explained by Taleb.

Designing architectures for situational adversity and uncertainty that thrive under pressure ensures that your infrastructure and system components are strong from the start. And the reverse is also true: designing for nominal circumstances ensures that your solution will likely buckle when placed under pressure.

Building an approach that remains strong when introduced to adverse circumstances implies an awareness of where the stress will come from and how it may affect your solution and its components. This requires a holistic and detailed knowledge of the solution and potential deployment environments and creative thinking about how it could be used. As you think through these issues, we recommend creating a list of potential areas of vulnerability and sorting them by most concerning to least so that you can spend the most time working on the areas that need the most attention.

## What are your current areas of weakness or vulnerability?

As you create your solution architecture, we recommend being transparent about the trade-offs you encounter. Detail the points where improvements could have been made and why they weren’t. Work with the development and product organizations to ensure you all understand the strengths and weaknesses. You may be able to turn what you felt was a point of fragility into a product feature or selling point. Here are some specific areas of potential weakness to ponder when creating a solution design:

* **Connectivity**: As covered earlier, DDIL environments describe the range of potential connectivity effects, and an offline-first posture can remediate that. Think about the potential sources of connectivity loss or degradation and see how that might be mitigated or used as a feature.
* **Physical access**: This is a potential area of vulnerability, but also the primary method of access for provisioning devices and ongoing hardware maintenance. Moving to a system of zero-touch provisioning, onboarding, and life cycle management will partially offset the lack of access to the hardware. Strategies to quarantine potentially compromised devices, and to flag the data they produce as suspect, would help. How might people potentially abuse having physical access and can that be turned from a weakness into a strength?
* **Environmental factors**: Air quality, temperature extremes or volatility, and vibrations all contribute to decreased operating conditions and potentially rapid equipment failure. Are there negative operating modalities that have positive aspects you could leverage?
* **Peak demand**: Systems are typically provisioned for an anticipated range of demand or usage. What are the worst-case scenarios regarding short-term impacts? And what if a product that uses your system proves wildly successful and you need to scale up quickly for the long term?
* **Hardware or software dependencies**: Is your solution tied to a specific type of hardware, a software library from a specific vendor, or a proprietary connector or protocol? If you can’t abstract away from or around that dependency, can you prevent it from becoming a **single point of failure** (**SPoF**)?

Once you’ve probed and documented areas of potential weakness in your solution architecture, it’s time to think about what situations and features make an architecture antifragile. Dwell on the following properties and consider how they might apply to your situation. Use them as a test to determine whether your architecture is either fragile or antifragile.

## Properties of antifragile architectures

Taleb stated that antifragility is measurable using a straightforward test of asymmetry. Let’s think about how this would apply to a solution that you design. When subject to random events, does your solution as a whole deliver more positive benefits or more negative results? If, in the balance, the results are more positive, then your approach is likely antifragile.

Let’s think about scoring antifragility in the context of **machine learning** (**ML**) models. Object recognition models are typically trained with large labeled datasets of imagery, and can then be used to score the likelihood that a previously unseen image is an object that it recognizes. These models are known for failing to recognize objects they were trained on when shown the target object from a different perspective, with a new background, or in a different color or lighting arrangement. Since it does not deliver a net benefit when exposed to novel or random inputs, it is quite fragile.

Another property of antifragile architecture is that it responds well in unpredictable environments and unanticipated situations. Another way to think about this property is how well your components, infrastructure, or solution respond to edge conditions and corner cases. If-then logic, most algorithms, and some types of ML work best with known inputs, situations, and sequences. Therefore, attempting to predict and capture most scenarios in advance and responding appropriately would lead to a antifragile implementation.

A counter-intuitive property of an antifragile solution design is that the system becomes inefficient or weaker overall if volatility is suppressed, minimized, or otherwise filtered. For example, polling for messages at a scheduled interval is a fragile design since it generates load on a server by repeatedly checking for messages in situations when there are no messages for an extended time period, and likewise causes a backup when there is a surge in messages over a short time period because it is not able to check for messages more frequently.

However, polling at a variable rate in combination with a back-off algorithm allows the system to handle surges whenever they occur, quickly drain a queue or backlog, and then return to a slower interval as the number of messages declines. This would be an antifragile approach.

Some system types and patterns are fragile by their very nature, including command and control and other complex systems. The main cause of this fragility is the high number of interconnections and interdependencies. These not only mask the causes of certain responses but also prevent simple debugging. Therefore, the best way to reduce fragility in these types of systems is to reduce the complexity of the solutions employed within them.

## An ounce of prevention...

While the end goal of antifragility is to prevent situations that could introduce negative results and a lack of flexibility, it is also important to backstop that with functions or components that will also correct any errors or misconfigurations if they can be easily anticipated. This attempt at self-healing is also a valid secondary approach to antifragility in edge architecture. It can also function as a substitute for first-level technical support by implementing checks for, and corrections to, frequently encountered issues on day two (after the solution has launched and is in production).

Overall, being antifragile means delivering a way to remove disorder from a system, or otherwise increasing order. But no matter how well designed a system is, eventually major issues can and will happen. Let’s take a look at how to respond in the final section of this chapter.

# When things go wrong

In the previous section, we wrote about patterns that benefit from adverse events and unpredictability. But it is just as important to realize which patterns we should avoid, since they are fragile, prone to failure, inflexible, brittle, and temporal. In most of this book, the authors have sought to warn away from certain behaviors, structures, approaches, and implementations for these reasons. In this final section of the book, we will attempt to bring it all together into a series of steps to take to first avoid, and second to recover, from potential disasters and failures.

## What to avoid

It can be beneficial to re-state the obvious at times. More experienced architects will just assume these fundamental axioms, and thus run the risk of not passing them on to more junior staff. Let’s review the basics:

* **Always have a plan B**. That means that you should assume that code (or networks and systems) will fail at the most inconvenient moments and bake in functionality that uses the circumstances to your advantage.
* **Do not intentionally introduce errors**. You may have heard the oft-used phrase in agile programming Fail fast, fail early. Be careful with that thinking because it has unintended consequences. “Iteration while failing forward” is a better concept because it ensures that you catch small errors quickly and fix them as you progress. One of the funniest and saddest errors we ever saw was a log message stating The code should never execute this function.
* **Always test under load**. There will always be one or two scenarios you cannot anticipate until you thoroughly stress-test a solution. That holds true for edge computing architectures especially when you deploy to thousands or tens of thousands of heterogeneous edge devices across scores of disparate environments.
* **Do not make unvalidated assumptions**. All inputs, operating conditions, system requirements, bills of material, and outputs should be validated. If possible, the reasons for each should be documented to prevent the Chesterton’s fence condition where nobody knows the reason why a feature or function exists and thus does not know whether it can be safely removed.
* **Requirements should be written with the end user’s input**. It is imperative to work with your customer when performing acceptance testing of your solution. If you do not, you run the risk of either creating what was not requested or missing fundamental features.

## Anti-patterns

What patterns are bad for your architecture? Identifying processes or patterns that are not helpful or are counterproductive is just as important as knowing which patterns to adopt.

By definition, an anti-pattern is an approach to solving a software engineering challenge that appears to be useful, but whose outcomes are ineffective or cause more problems in the long run when adopted. They are considered to be bad programming practices and should be avoided.

### What situations expose anti-patterns?

The following points communicate the root causes that the authors have seen developers repeat. Therefore, we must assume that the lessons learned are non-intuitive or otherwise not obvious:

* **Attempting to scale an existing solution**: Your operations team might have a well-defined manual process for provisioning, deploying, and managing an edge resource. It may have been working well in an existing fleet comprising tens of devices. But that process might not scale well, and even become a bottleneck, when deploying to a larger fleet of hundreds or thousands of devices.
* **Applying a solution from one class to all classes**: All edge devices are not the same, so when it comes to deploying them, solution architects cannot treat them as a single monolithic group. You should consider categorizing them according to their purpose, product or vertical affinity, or innate properties before creating a solution for each group or class. Once all solutions have been considered, then propose any optimizations.
* **Using the same methods in all operating contexts**: In earlier chapters, we discussed the disconnected edge. Your edge solutions should not assume that all devices will be always connected to the network. Your designs should take into consideration that the edge devices will have to continue to operate in a disconnected mode. The duration of the disconnection is something that must be worked out between the architect and the customer.

## How to recover, gracefully or not

There are times when technology goes wrong and it is hard to recover gracefully. At one point or another, we all have relied on the navigation system in our cars and ended up at an incorrect destination. The only recourse in that situation is to backtrack or contact a human who can help you. Similarly, in an edge computing solution, if an animal or nature somehow wreaks havoc on a field-deployed device, the only recourse might be to send a human to rectify that problem. While uncommon, one has to be mindful of such a worst-case scenario.

* **Follow the plan**

The first step should be to follow your formal **disaster recovery** (**DR**) plans. More importantly, ensure that they are tested and you know that they work. Next, ensure that you test them on a scheduled basis by conducting drills. In the case of architectures utilizing high-availability or standby scenarios, ensure you alternate testing so that all permutations are eventually exercised. Also, ensure that the backup structure is kept intact, even when employees leave the organization.

We often see a decal on fire extinguisher cabinets that reads In case of emergency break glass. What we don’t know is whether the fire extinguisher will work when needed. In the same vein, when an edge hub goes down, will the newly disconnected edge devices continue to generate data? When the hub is replaced or brought back online, will everything sync up correctly? That’s why having the best DR design in an architecture is not enough. It needs to be tested by simulating real-world situations. Only then can one be sure to recover from a real disaster.

* **What if there isn’t a plan?**

If there isn’t a plan, we have two options:

* + Revert to the last known working state, even if that ultimately means rebuilding from scratch. It’s always faster the second (or third) time. Then, re-apply subsequent changes while continually monitoring. In this way, you can iteratively restore functionality and return to the current state. Hopefully, you are following product requirements to ensure proper functionality. If those do not exist, document and create those requirements as you go.
  + If reverting and rebuilding from there does not work, triage and build from scratch, beginning with the most important function first. Try to use off-the-shelf solutions as much as possible.
* **Prevent issues from recurring**

Here are some hard-won lessons we’ve learned from edge computing deployments that we’ve seen, heard about, or participated in. We hope that you will not repeat these same errors.

Always have a plan B, even if not thoroughly thought through. Make sure you know alternatives for infrastructure, providers, platforms, processes, solutions, optimizations, and personnel. We used to joke about not knowing what we’d do if one of our leaders got hit by a bus, but that points out a very real issue. For example, each leader should have an identified backup person or alternate, even if it’s only temporarily while that leader is off sick or on vacation. This helps ensure business continuity by ensuring that everyone knows “who has the D” (decision-making powers in a situation or circumstance) and also that no business-critical knowledge is solely vested in a single individual but instead everyone is cross-trained. Likewise, ensure that you’ve considered a backup or alternate provider, solution, device, or system component.

Given that, systems should be designed to prevent outages in the future at a minimum by having hot or warm backups or alternative processes and systems, especially if the backups operate in a different modality than the primary systems they are designed to backstop. One way to prevent future hiccups is to visualize existing data with the right tools to better identify possible weak links. While edge computing solutions are used in predictive analytics use cases, the same rigor needs to be applied to the edge solution architecture itself. That could be achieved by autonomic computing or a variant of self-analysis.

Don’t expect a solution component to function in a substantially different manner, even when placed in a new context. This applies both to optimizing performance as well as attempting to mitigate poor results. When it comes to investment advice and picking stocks, we have all heard the phrase past performance is not indicative of future results. On the contrary, you can tell a lot from the past performance of machines and IT systems. You might not be able to predict exactly how physical components in an edge solution will work or interact with each other in a future use case or solution, but you can get a good understanding of how most of the software system components will operate in the near future based on their past performance characteristics.

But the ultimate points here are to observe operating characteristics, collect and analyze data, draw inferences about operating behavior, perform small tests to confirm your assumptions about the behavior, document and disseminate your learnings, and implement those as best practices throughout your systems. Also, on a scheduled, periodic basis, re-confirm your assumptions to ensure that the operating environments have not changed sufficiently to invalidate your practices. The salient point is that your systems will require constant, ongoing care. They should be thought of as living entities in the sense that they will slowly change and develop new quirks over time.

# Summary

In this chapter, we detailed errors and mistakes that you might commonly encounter both in designing architectures and in working with existing deployed solutions. Some of the remedies we’ve reviewed may seem like common sense to you, but not to others.

We went over practices that were developed specifically for the edge-native environment: pulling versus pushing and their security implications, discussing application dependency management and resolution approaches, and deployment models. The next section covered ways to think about designing antifragile solutions that improve when placed under stress instead of breaking when exposed to random events. In the last section, we went over our options when systems fail.

As a result, you should have a list of considerations that will help you to architect long-lived and resilient solutions for edge environments. We hope that you will not only put these recommendations and guidelines into practice but also pass along what you’ve learned to both developers and management. And when you do, please mention where you read about them.