

Chapter 3: Data Processing at the Edge

- ❖ Data Acquisition and Processing: Data handling, python data handling, data storage and cloud connectivity, Data Aggregation, Data Time stamping and Synchronization, Data Security and Privacy.
- ❖ Edge analytics and machine learning at the edge: Introduction to Edge Analytics. Edge Machine Learning. Model Selection and Optimization. Collaborative Edge Learning.
- ❖ Resource management and task offloading strategies: Task Offloading, Edge-Cloud Collaboration, Dynamic Resource Provisioning.
- ❖ Edge caching and data synchronization: Introduction to Edge caching and data synchronization, Benefits of Edge Caching and Data Synchronization, Challenges in Edge Caching and Data Synchronization.
- ❖ Self-Learning Topics: Task Migration, Offline Operation, Bandwidth Optimization

Chapter 3: Data Processing at the Edge

- ❖ Data Acquisition and Processing: Data handling, python data handling, data storage and cloud connectivity, Data Aggregation, Data Time stamping and Synchronization, Data Security and Privacy.
- ❖ Edge analytics and machine learning at the edge: Introduction to Edge Analytics. Edge Machine Learning. Model Selection and Optimization. Collaborative Edge Learning.
- ❖ Resource management and task offloading strategies: Task Offloading, Edge-Cloud Collaboration, Dynamic Resource Provisioning.
- ❖ Edge caching and data synchronization: Introduction to Edge caching and data synchronization, Benefits of Edge Caching and Data Synchronization, Challenges in Edge Caching and Data Synchronization.
- ❖ Self-Learning Topics: Task Migration, Offline Operation, Bandwidth Optimization

Chapter 3: Data Processing at the Edge

- ❖ Data Acquisition and Processing: Data handling, python data handling, data storage and cloud connectivity, Data Aggregation, Data Time stamping and Synchronization, Data Security and Privacy.
- ❖ Edge analytics and machine learning at the edge: Introduction to Edge Analytics. Edge Machine Learning. Model Selection and Optimization. Collaborative Edge Learning.
- ❖ Resource management and task offloading strategies: Task Offloading, Edge-Cloud Collaboration, Dynamic Resource Provisioning.
- ❖ Edge caching and data synchronization: Introduction to Edge caching and data synchronization, Benefits of Edge Caching and Data Synchronization, Challenges in Edge Caching and Data Synchronization.
- ❖ Self-Learning Topics: Task Migration, Offline Operation, Bandwidth Optimization

Need of Data acquisition and processing in Edge computing

Edge computing was initially the primary technology to address 5G latency, but has recently been introduced into new areas such as IoT and the Internet.

The problem of cloud computing can also be solved with the help of modern information technology. The intelligent services provided by advanced computer technology meet key requirements such as flexible connection, real-time operation, data upgrade, software intelligence and security, and data basic processing. Robotic arms come in many different forms, but they all share common characteristics. The ability to select a course of action and precisely point-and-shoot in three dimensional (or two-dimensional) space. The traditional mechanical arm can only perform simple repetitive actions such as translation and grabbing, but when the mechanical arm is combined with the information system, it can complete complex operations such as cooking and mixing. At this time, it is necessary to identify and analyze the complex motion behavior of the manipulator. **However, the traditional data acquisition and processing methods cannot analyze the data quickly, which leads to low efficiency. Therefore, it is necessary to collect and process the data of manipulator behavior recognition based on edge calculation.**

How is data processed in edge computing?

Edge computing is a distributed information technology (IT) architecture in which client data is processed at the periphery of the network, as close to the originating source as possible.

Edge Data Management is the approach for managing and retrieving data for analysis and processing from the nearest source to reduce the latency, mitigate short delays and enable high throughput.

Data Acquisition and Processing:

Data acquisition (commonly abbreviated as DAQ or DAS) is the process of sampling signals that measure real-world physical phenomena and converting them into a digital form that can be manipulated by a computer and software.

Data Acquisition Methods :

There are four methods of acquiring data:

- ❑ collecting new data
- ❑ converting/transforming legacy data
- ❑ sharing/exchanging data
- ❑ purchasing data.

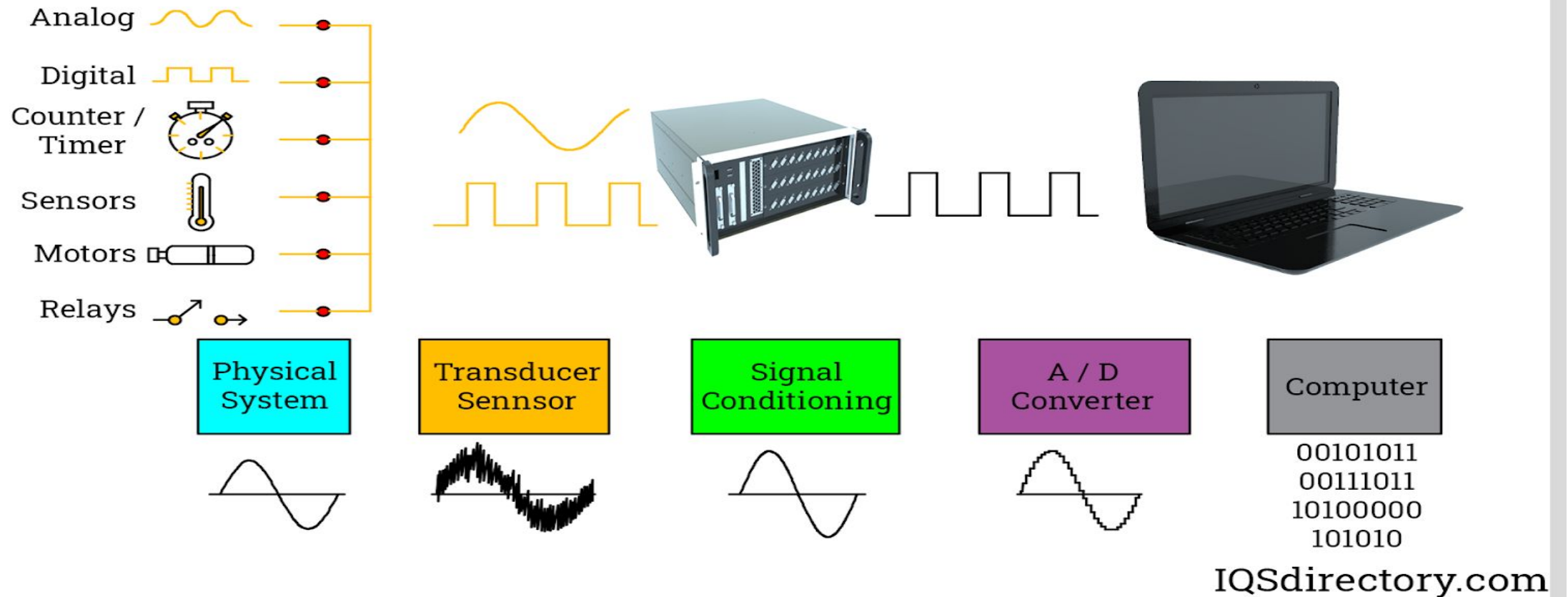
This includes automated collection (e.g., of sensor-derived data), the manual recording of empirical observations, and obtaining existing data from other sources.

What is the process of data acquisition?

A data acquisition system is a collection of software and hardware that allows one to measure or control the physical characteristics of something in the real world. A complete data acquisition system consists of DAQ hardware, sensors and actuators, signal conditioning hardware, and a computer running DAQ software.

- As an example, a data acquisition system can be used when testing the temperature of a heating coil used for heating an object to a specific temperature. The level of success of the heating coil is understood by measuring its temperature. **That simple task of measuring and recording temperature is called data acquisition and is achieved using a data acquisition system.**
- The reason for measuring and recording the electrical and physical phenomena using a data acquisition system is to enable further analysis. A data acquisition system uses software to perform its functions and it is capable of quickly processing and storing data in many ways. Data acquisition systems can capture data from an actual system and store the data in a simple format that is easily retrievable for further engineering or scientific review.
- Data acquisition systems are **either handheld, or they can be remotely operated**. Handheld data acquisition systems are used when there is a requirement for taking readings of a specimen which can be physically interacted with. When direct human interaction with an object is not possible or necessary, this is when remote data acquisition systems are used to take remote DAQ (data acquisition) measurements.
- **Whether it is cloud, fog, or edge computing, it is only a method or mode to realize the computing technology needed by the Internet of Things, intelligent manufacturing, etc. On the basis of a single system, a reserved sensor interface is used to complete the realization of the sensor acquisition function. As shown in Figure 2, by connecting the sensors that meet the requirements of the interface hardware with the edge device, the software is used to complete the preprocessing of the collected data, thereby forming the underlying data acquisition module. This can not only expand the scope of application of the sensor, but also ensure that the collected data is uploaded to the cloud in time, shortening the delay of data processing and feedback**

Data Acquisition System Components



A full data acquisition system consists of DAQ hardware, sensors, actuators, signal conditioning gear, and a computer running DAQ software

Transducer

A transducer is a device that converts energy from one form to another. Usually a transducer converts a signal in one form of energy to a signal in another. Transducers are often employed at the boundaries of automation, measurement, and control systems, where electrical signals are converted to and from other physical quantities (energy, force, torque, light, motion, position, etc.). The process of converting one form of energy to another is known as transduction.

Types:

❖ Mechanical transducer

Mechanical transducers, so-called as they convert physical quantities into mechanical outputs or vice versa.

❖ Electrical transducers however convert physical quantities into electrical outputs or signals. Examples of these are: a thermocouple that changes temperature differences into a small voltage;

❖ a linear variable differential transformer (LVDT), used to measure displacement (position) changes by means of electrical signals.

Sensors, actuators and transceivers

Transducers can be categorized by which direction information passes through them:

A sensor is a transducer that receives and responds to a signal or stimulus from a physical system. It produces a signal, which represents information about the system, which is used by some type of telemetry, information or control system.

An actuator is a device that is responsible for moving or controlling a mechanism or system. It is controlled by a signal from a control system or manual control. It is operated by a source of energy, which can be mechanical force, electrical current, hydraulic fluid pressure, or pneumatic pressure, and converts that energy into motion. An actuator is the mechanism by which a control system acts upon an environment. The control system can be simple (a fixed mechanical or electrical system), software-based (e.g. a printer driver, robot control system), a human, or any other input.

Bidirectional transducers can convert physical phenomena to electrical signals and electrical signals into physical phenomena. An example of an inherently bidirectional transducer is an antenna, which can convert radio waves (electromagnetic waves) into an electrical signal to be processed by a radio receiver, or translate an electrical signal from a transmitter into radio waves. Another example are voice coils, which are used in loudspeakers to translate an electrical audio signal into sound, and in dynamic microphones to translate sound waves into an audio signal.

Transceivers integrate simultaneous **bidirectional functionality**. The most ubiquitous example are likely radio transceivers (in aircraft called **transponders**), used in virtually every form of wireless (tele-)communications and network device connections. Another example are **ultrasound transceivers that are used for instance in medical ultrasound (echo) scans**.

Signal conditioning

In electronics and signal processing, signal conditioning is the manipulation of an analog signal in such a way that it meets the requirements of the next stage for further processing.

In an analog-to-digital converter (ADC) application, signal conditioning includes voltage or current limiting and anti-aliasing filtering.

In control engineering applications, it is common to have a sensing stage (which consists of a sensor), a signal conditioning stage (where usually amplification of the signal is done) and a processing stage (often carried out by an ADC and a micro-controller). Operational amplifiers (op-amps) are commonly employed to carry out the amplification of the signal in the signal conditioning stage. In some transducers, signal conditioning is integrated with the sensor, for example in Hall effect sensors.

In power electronics, before processing the input sensed signals by sensors like voltage sensor and current sensor, signal conditioning scales signals to level acceptable to the microprocessor.

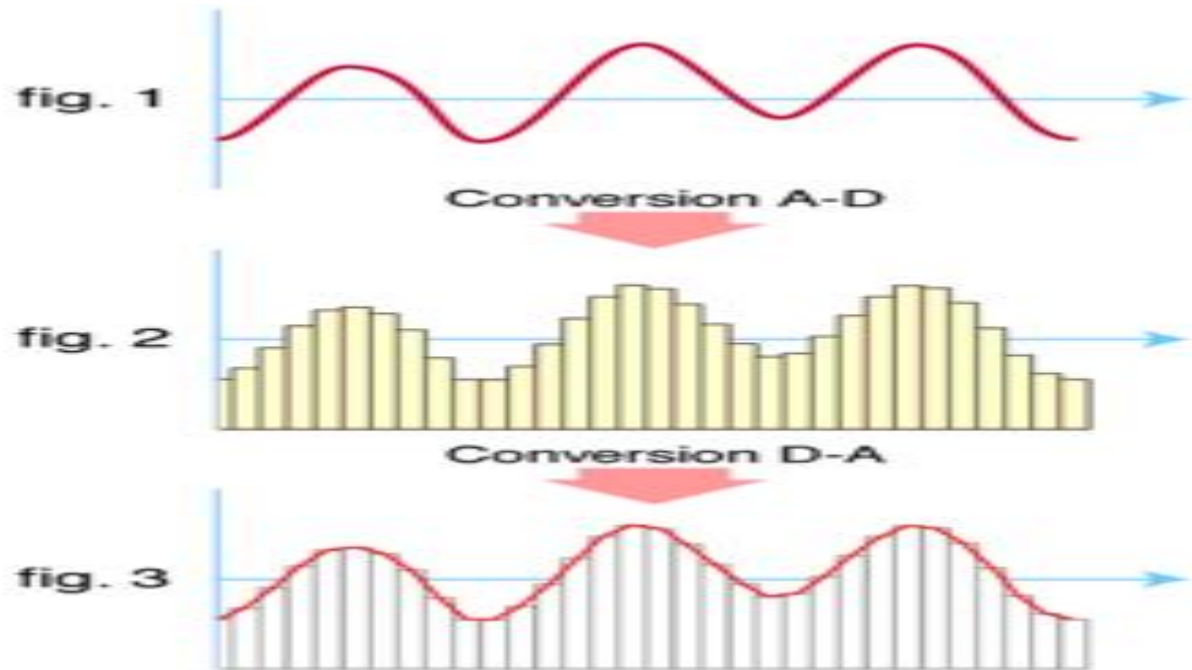
Sensor inputs can be accelerometer, thermocouple, thermistor, resistance thermometer, strain gauge or bridge, and LVDT or RVDT. Specialized inputs include encoder, counter or tachometer, timer or clock, relay or switch, and other specialized inputs. Outputs for signal conditioning equipment can be voltage, current, frequency, timer or counter, relay, resistance or potentiometer, and other specialized outputs.

Processes

Signal conditioning can include amplification, filtering, converting, range matching, isolation and any other processes required to make sensor output suitable for processing after conditioning.

Analog-to-digital converter (ADC, A/D, or A-to-D)

In electronics, an analog-to-digital converter (ADC, A/D, or A-to-D) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. An ADC may also provide an isolated measurement such as an electronic device that converts an analog input voltage or current to a digital number representing the magnitude of the voltage or current. Typically the digital output is a two's complement binary number that is proportional to the input, but there are other possibilities.



An ADC converts a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal. The conversion involves quantization of the input, so it necessarily introduces a small amount of quantization error. Furthermore, instead of continuously performing the conversion, an ADC does the conversion periodically, sampling the input, and limiting the allowable bandwidth of the input signal.

For the Internet of Things, break through in edge computing technology mean that many controls will be implemented through local devices without being handed over to the cloud, and the processing will be completed at the local edge computing layer. This will undoubtedly greatly improve processing efficiency and reduce the load on the cloud. Due to being closer to the user, it can also provide users with a faster response and solve their needs at the edge

Data handling

Data must be appropriately handled in edge to ensure it is available when needed. This implies that data collection, storage, and forwarding must be well planned. The challenge is where to store all that data. The challenge is where to store all that data.

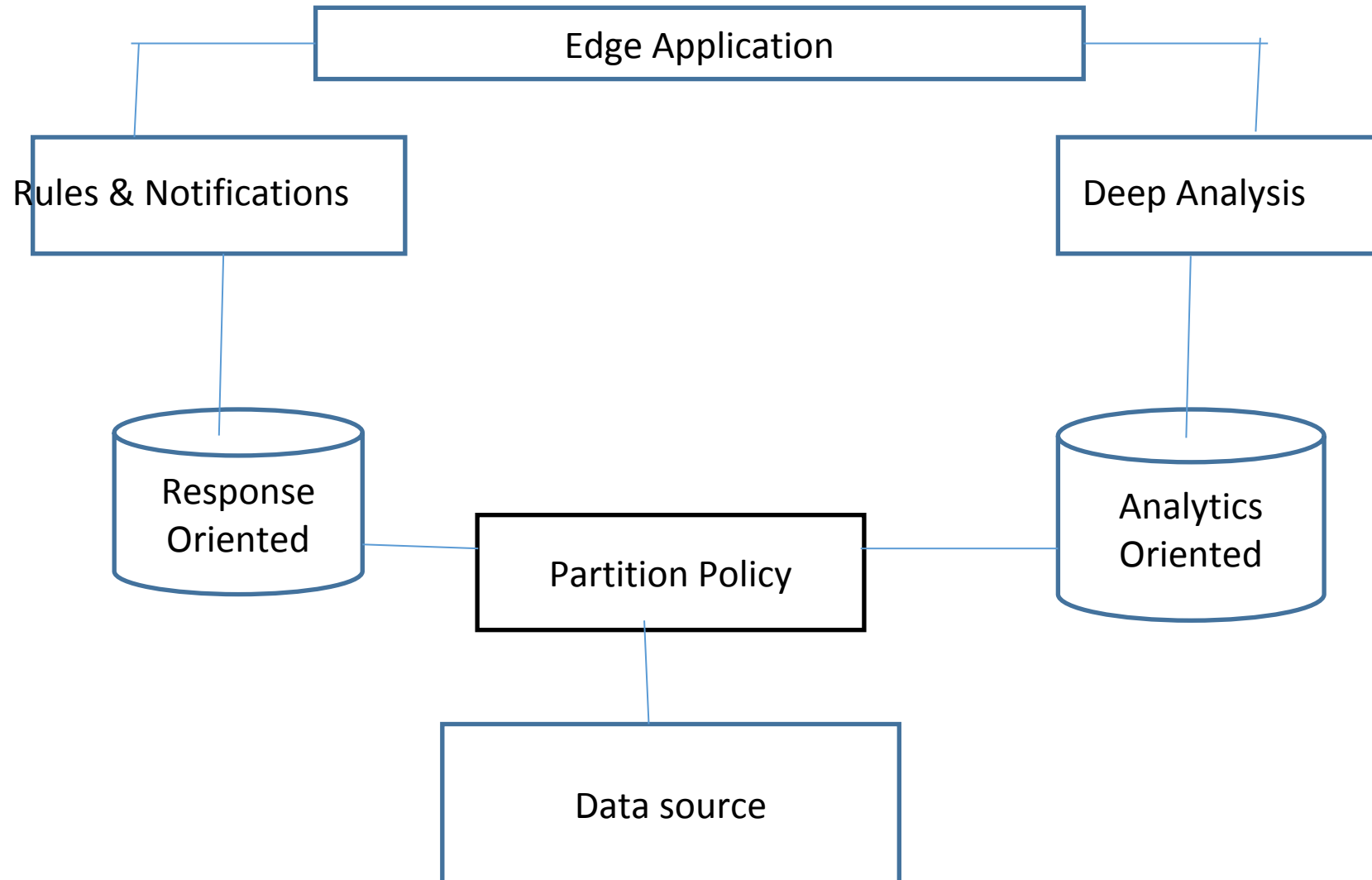
1. Data Partitioning
2. Store and Forward

Edge devices continue to get more powerful, with more storage, faster processing, and features that mimic a traditional server, such as running standard operating systems. That means the opportunity to partition data and data processing on these devices will continue to increase.

You need to allow access to the data in reliable ways, and with as little latency as possible. It's time to get better at partitioning across these systems.

Data Partitioning

<https://techbeacon.com/enterprise-it/data-management-edge-time-get-partitioning>



Data source in edge computing

Edge computing is a distributed computing framework that brings enterprise applications closer to data sources such as IoT devices or local edge servers. This proximity to data at its source can deliver strong business benefits, including faster insights, improved response times and better bandwidth availability.

So how do you partition the data?

Emerging issues to consider when deciding whether to partition data at the edge or the at central server include:

The usage patterns of the data at the edge, and the capabilities of the edge-based device or system

- Data security and governance requirements
- Performance and reliability
- Physical data storage on both ends

Central to the partitioning decision is looking at the core requirements of the system at the edge, including how you'll use the data. Moreover, you need to understand if data storage at the edge is even viable.

For example, say you're looking to store GPS and soil data for a mapping system hosted in a drone that's programmed to fly around a cornfield to do soil moisture assessments for the optimization of irrigation systems. You need to look at a few things:

- ❖ Can the drone's onboard storage, memory, and CPU store the data gathered during each flight?
- ❖ How much data will be stored, and how will it be shared with external systems, such as systems that exist on public clouds?
- ❖ Is the data needed to allow for responsive actions, such as going over the same part of the field twice, if data collection failed for some reason during a flight?

Data partitioning guidance

In many large-scale solutions, data is divided into partitions that can be managed and accessed separately. Partitioning can improve scalability, reduce contention, and optimize performance. It can also provide a mechanism for dividing data by usage pattern. For example, you can archive older data in cheaper data storage.

However, the partitioning strategy must be chosen carefully to maximize the benefits while minimizing adverse effects.

Why partition data?

- ✓ **Improve scalability.** When you scale up a single database system, it will eventually reach a physical hardware limit. If you divide data across multiple partitions, each hosted on a separate server, you can scale out the system almost indefinitely.
- ✓ **Improve performance.** Data access operations on each partition take place over a smaller volume of data. Correctly done, partitioning can make your system more efficient. Operations that affect more than one partition can run in parallel.
- ✓ **Improve security.** In some cases, you can separate sensitive and nonsensitive data into different partitions and apply different security controls to the sensitive data.
- ✓ **Provide operational flexibility.** Partitioning offers many opportunities for fine-tuning operations, maximizing administrative efficiency, and minimizing cost. For example, you can define different strategies for management, monitoring, backup and restore, and other administrative tasks based on the importance of the data in each partition.
- ✓ **Match the data store to the pattern of use.** Partitioning allows each partition to be deployed on a different type of data store, based on cost and the built-in features that data store offers. For example, large binary data can be stored in blob storage, while more structured data can be held in a document database. See [Choose the right data store](#).
- ✓ **Improve availability.** Separating data across multiple servers avoids a single point of failure. If one instance fails, only the data in that partition is unavailable. Operations on other partitions can continue. For managed PaaS data stores, this consideration is less relevant, because these services are designed with built-in redundancy.

Analytics-oriented edge data

This is a bit more complex. This pattern allows you to take immediate actions based upon the simple state of the data, such as the current internal temperature of a robotic welder.

However, you can also analyze the data at the edge. This allows you to derive more value from the data, by doing such things as considering a million records of temperature data from the robotic welder to determine patterns that may alert maintenance of issues that need to be addressed to avoid a failure that stops production.

This type of deep analytics normally occurs on centralized systems, such as in public clouds. However, in this case, it's more efficient to do the analytics on the edge.

As edge-based devices become more powerful, analytics-oriented edge data is becoming a more feasible option. Even deep analytics, normally reserved for cloud servers or traditional on-premises servers, work just fine at the edge device or system.

Response-oriented edge data

This is the main reason to place data at the edge. This category includes any applications where data needs to be stored and viewed immediately, and you also need to avoid both latency and connectivity issues.

One example is edge-based systems on jet airplanes. While some of the data, such as maintenance and performance data, is centrally stored, the data related to the working of the core avionics is stored onboard.

The result is little or no latency when that data needs to be accessed, such as when the avionics system needs to automatically correct problems with the engine (e.g., fuel mixture) in near real time.

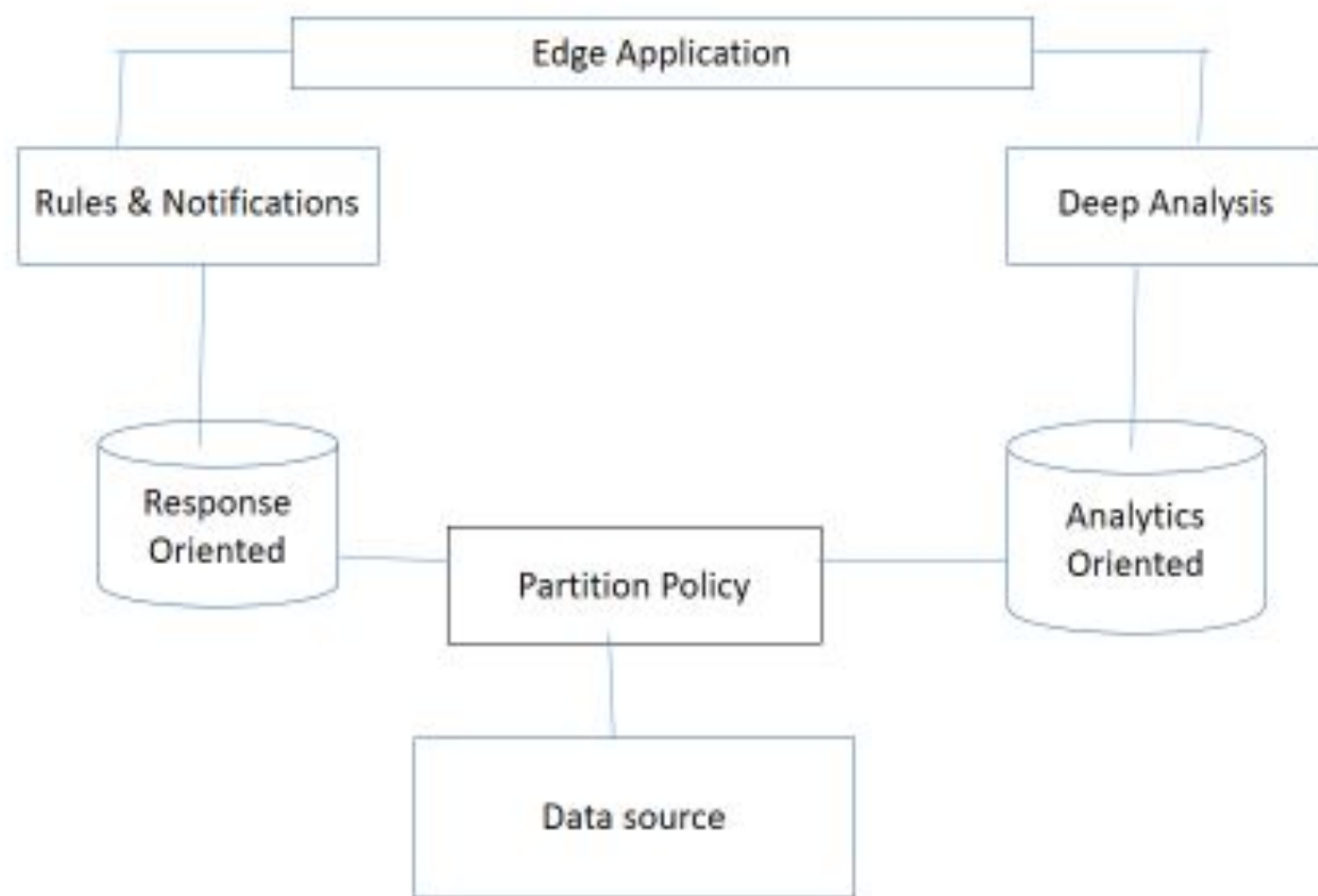
The core actions that result are typically procedural and simplistic problems related to simple data. These include: If this is out of whack, then do this. However, they can be more complex, such as: If this is out of whack, then try this, and if that fails, try this, and if that fails, try this. Those who have done basic programming understand these patterns.

The idea is to make the simple data instantaneously accessible, with the ability to react to the simple data. The goal is to make tactical transaction-oriented decisions and take actions based upon the state or values of the data in the edge-based device or system.

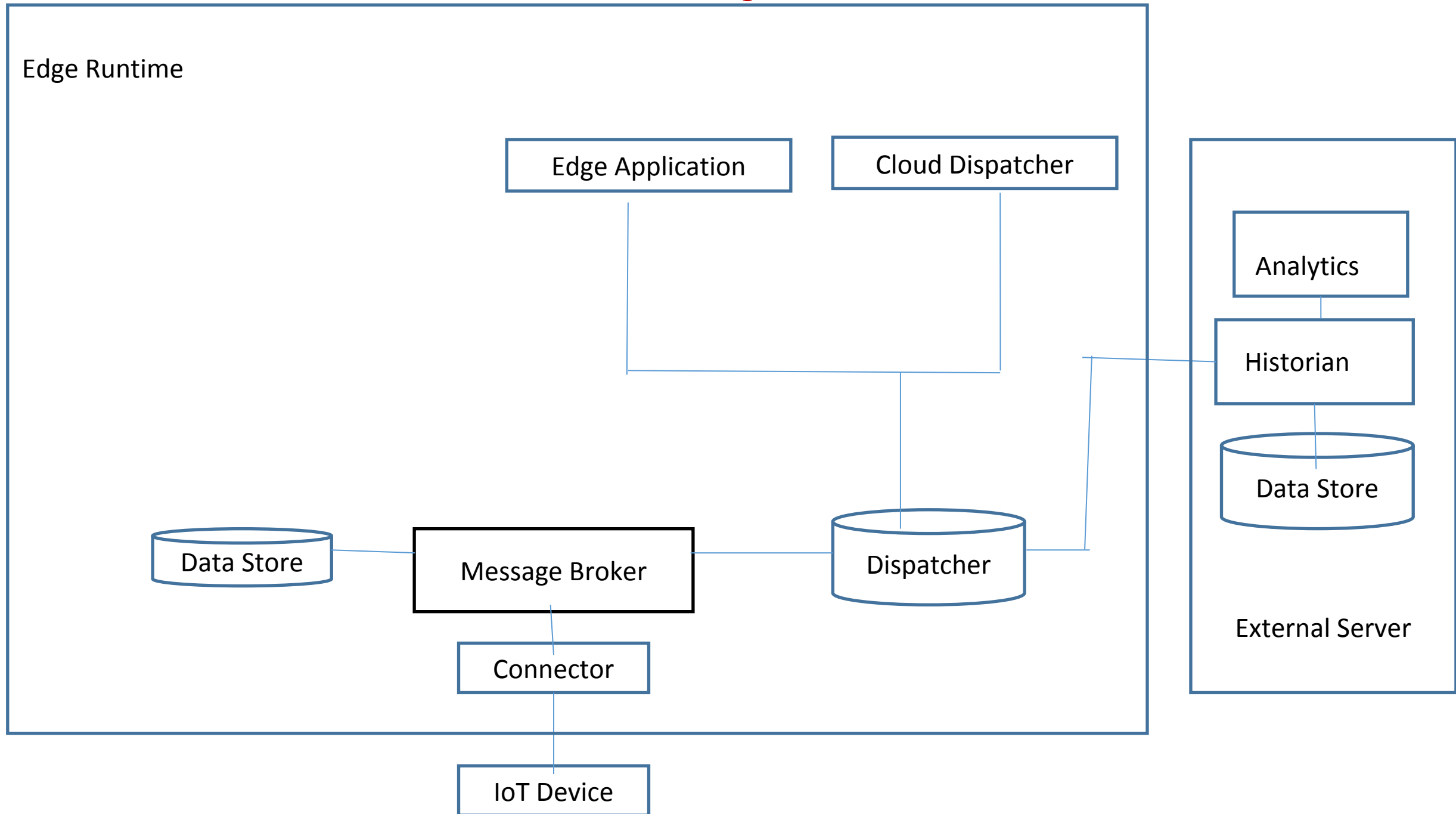
Partitioning rules and guidelines

- ❖ Consider the types of transformations in the mapping and the order in which transformations appear so that you do not get unexpected results. You can partition a mapping if the mapping task can maintain data consistency when it processes the partitioned data.
- ❖ For flat file partitioning, session performance is optimal with large source files. The load may be unbalanced if the amount of input data is small.
- ❖ When a Sequence Generator transformation is in a mapping with partitioning enabled, ensure that you set up caching in the Sequence Generator transformation. Otherwise, the sequence numbers the task generates for each partition are not consecutive.
- ❖ Sequence numbers generated by Normalizer and Sequence Generator transformations might not be sequential for a partitioned source, but they are unique.
- ❖ When a Sorter transformation is in a mapping with partitioning enabled, the task sorts data in each partition separately.
- ❖ A Sorter transformation must be placed before any Joiner transformation or Aggregator transformation that is configured to use sorted data.
- ❖ You cannot use in-out parameters for key range values.
- ❖ If your mapping has more than eight partitions, mapping task performance might degrade. You can configure the Buffer Block Size and DTM Buffer Size advanced properties in the mapping task to improve performance.

Data Partitioning



Storage and Forward



Store and Forward

1. Data growth over time. Edge-based devices and systems are typically limited and difficult to upgrade, whereas cloud-based storage can be increased at the push of a button.
2. Data access across partitions during processing. Will you run processes that span both the edge-based computer and central cloud-based servers? If so, what access patterns will you employ?
3. Data storage systems. Will you leverage a traditional database on both sides, such as MySQL? Or will you leverage a purpose-built database, such as those built for embedded systems? There once were few options, but many edge-based systems leverage traditional operating systems and thus can host traditional databases.

Security issues with data gathered

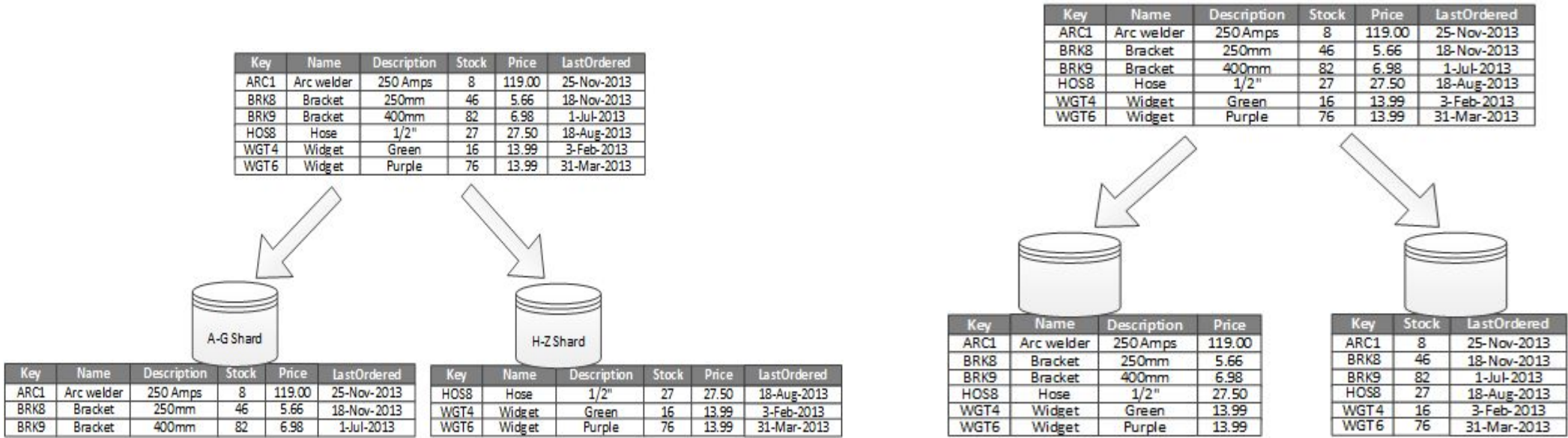
While most people don't think of security as a primary driver for edge/central data partitioning, for most applications it's going to be at the forefront. Sometimes this is due to the sensitive nature of the data being gathered at the edge. Other times, if the data is changed or corrupted in any part of the partition, bad things can happen. Consider the jet engine example above.

Edge-based security technology is still emerging, and thus much of the security you'll have to implement will be tactical in nature. You can count on encryption and key management to be core to data security on the edge device, and in some cases multi-factor authentication will play a role.

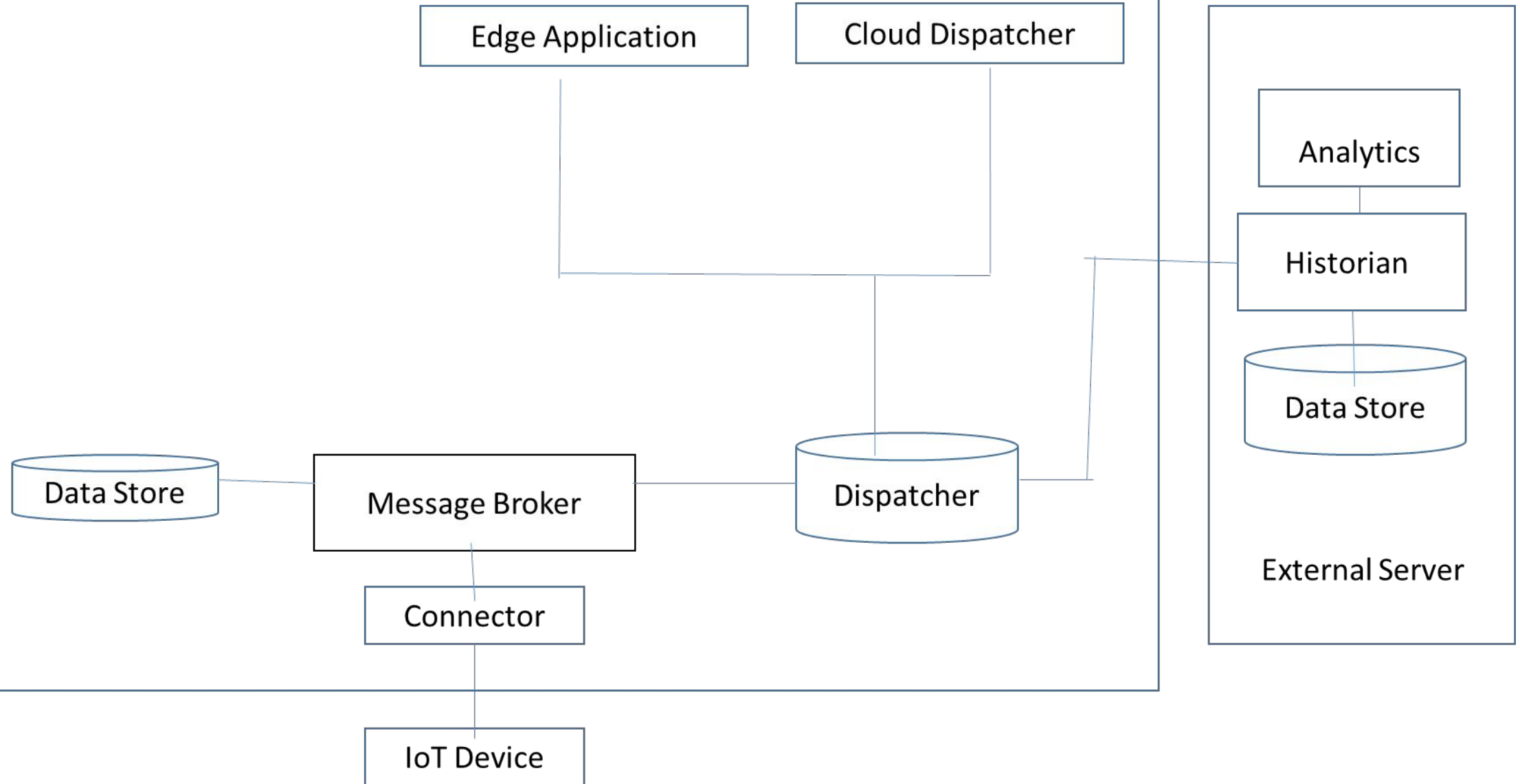
Designing partitions <https://learn.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning>

There are three typical strategies for partitioning data:

- ❖ Horizontal partitioning (often called sharding). In this strategy, each partition is a separate data store, but all partitions have the same schema. Each partition is known as a shard and holds a specific subset of the data, such as all the orders for a specific set of customers.
- ❖ Vertical partitioning. In this strategy, each partition holds a subset of the fields for items in the data store. The fields are divided according to their pattern of use. For example, frequently accessed fields might be placed in one vertical partition and less frequently accessed fields in another.
- ❖ Functional partitioning. In this strategy, data is aggregated according to how it is used by each bounded context in the system. For example, an e-commerce system might store invoice data in one partition and product inventory data in another.



Edge Runtime



<https://drive.google.com/drive/my-drive>

<https://drive.google.com/drive/folders/1UTssrcDrGzeBGPfZrBHmJy71YP22Cw4S>

Data Storage and cloud connectivity

Local data storage

- ❖ File storage handling
- ❖ JSON files
- ❖ SQLite DB

Remote data storage

- Rest API
- MongoDB
- Time-series database

Sending data to cloud


- Azure IoT SDK



PRATYUSH HPCS

(India's First Multi Petaflop HPC Supercomputer)

COMPUTE



Cray XC40 System	
System Configuration	
Accelerator Node (1%)	
Number of Nodes	16
Accelerator	Intel KNL 7210, self-hosted mode
Memory Per Node (GB)	96 GB
Total Peak Performance	42.56 TFLOPS
Additional Nodes	
External Login Nodes	5
Utility Servers	8
Utility Racks	1



Local data storage

- ❖ **File storage handling:** Local storage stores files permanently on physical server equipment on your premises, either in flash memory or hard disk drives where application is running

Python File Handling

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short. [Python](#) treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let's start with the reading and writing files.

- ❖ JSON files
- ❖ SQLite DB

Advantages of File Handling

- **Versatility:** File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.
- **Flexibility:** File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).
- **User-friendly:** Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.
- **Cross-platform:** Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

Disadvantages of File Handling

- ❖ **Error-prone:** File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).
- ❖ **Security risks:** File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.
- ❖ **Complexity:** File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.
- ❖ **Performance:** File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

Working of open() Function in Python

Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function open() but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

f = open(filename, mode)

Where the following mode is supported:

r: open an existing file for a read operation.

w: open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.

a: open an existing file for append operation. It won't override existing data.

r+: To read and write data into the file. The previous data in the file will be overridden.

w+: To write and read data. It will override existing data.

a+: To append and read data from the file. It won't override existing data.

“t”: Text –Default value

“b” – Binary (for example, Image)

```
import os
def create_file(filename):
    try:
        with open(filename, 'w') as f:
            f.write('Hello, world!\n')
        print("File " + filename + " created successfully.")
    except IOError:
        print("Error: could not create file " + filename)

def read_file(filename):
    try:
        with open(filename, 'r') as f:
            contents = f.read()
            print(contents)
    except IOError:
        print("Error: could not read file " + filename)

def append_file(filename, text):
    try:
        with open(filename, 'a') as f:
            f.write(text)
        print("Text appended to file " + filename + " successfully.")
    except IOError:
        print("Error: could not append to file " + filename)
```

```
def rename_file(filename, new_filename):  
    try:  
        os.rename(filename, new_filename)  
        print("File " + filename + " renamed to " + new_filename + " successfully.")  
    except IOError:  
        print("Error: could not rename file " + filename)
```

```
def delete_file(filename):  
    try:  
        os.remove(filename)  
        print("File " + filename + " deleted successfully.")  
    except IOError:  
        print("Error: could not delete file " + filename)
```

```
if __name__ == '__main__':  
    filename = "example.txt"  
    new_filename = "new_example.txt"
```

```
create_file(filename)  
read_file(filename)  
append_file(filename, "This is some additional text.\n")  
read_file(filename)  
rename_file(filename, new_filename)  
read_file(new_filename)  
delete_file(new_filename)
```


JavaScript JSON

JSON or JavaScript Object Notation is a format for structuring data. What is it used for? Like XML, it is one of the way of formatting the data. Such format of data is used by web applications to communicate with each other.

Why JSON?

The fact that whenever we declare a variable and assign a value to it, it's not the variable that holds the value but rather the variable just holds an address in the memory where the initialized value is stored. Further explaining, take for example:

`let age=21;`

when we use age, it gets replaced with 21, but that does not mean that age contains 21, rather what it means is that the variable age contains the address of the memory location where 21 is stored.

Characteristics of JSON

- ❖ It is Human-readable and writable.
- ❖ It is light weight text based data interchange format which means, it is simpler to read and write when compared to XML.
- ❖ It is widely used as data storage and communication format on the web.
- ❖ Though it is derived from a subset of JavaScript, yet it is Language independent. Thus, the code for generating and parsing JSON data can be written in any other programming language.

Writing JSON to a file in Python

Serializing JSON refers to the transformation of data into a series of bytes (hence serial) to be stored or transmitted across a network. To handle the data flow in a file, the JSON library in Python uses `dump()` or `dumps()` function to convert the Python objects into their respective JSON object, so it makes it easy to write data to files. See the following table given below.

PYTHON OBJECT	JSON OBJECT
Dict	object
list, tuple	array
str	string
int, long, float	numbers
True	true
False	false
None	null

Method 1: Writing JSON to a file in Python using `json.dumps()`

The JSON package in Python has a function called `json.dumps()` that helps in converting a dictionary to a JSON object. It takes two parameters:

`dictionary` – the name of a dictionary which should be converted to a JSON object.

`indent` – defines the number of units for indentation

After converting the dictionary to a JSON object, simply write it to a file using the “write” function.

```
import json

# Data to be written
dictionary = {
    "name": "sathiyajith",
    "rollno": 56,
    "cgpa": 8.6,
    "phonenum": "9976770500"
}

# Serializing json
json_object = json.dumps(dictionary, indent=4)

# Writing to sample.json
with open("sample.json", "w") as outfile:
    outfile.write(json_object)
```

Output:

```
{
    "name": "sathiyajith",
    "rollno": 56,
    "cgpa": 8.6,
    "phonenum": "9976770500"
}
```

fference between File System and DBMS

Basics	File System	DBMS
Structure	The file system is a way of arranging the files in a storage medium within a computer.	DBMS is software for managing the database.
Data Redundancy	Redundant data can be present in a file system.	In DBMS there is no redundant data.
Backup and Recovery	It doesn't provide Inbuilt mechanism for backup and recovery of data if it is lost.	It provides in house tools for backup and recovery of data even if it is lost.
Query processing	There is no efficient query processing in the file system.	Efficient query processing is there in DBMS.
Consistency	There is less data consistency in the file system.	There is more data consistency because of the process of normalization.
Complexity	It is less complex as compared to DBMS.	It has more complexity in handling as compared to the file system.
Security Constraints	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file systems.
Cost	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.
Data Independence	There is no data independence.	In DBMS data independence exists, mainly of two types: 1) Logical Data Independence. 2)Physical Data Independence.
User Access	Only one user can access data at a time.	Multiple users can access data at a time.
Meaning	The users are not required to write procedures.	The user has to write procedures for managing databases
Sharing	Data is distributed in many files. So, it is not easy to share data.	Due to centralized nature data sharing is easy
Data Abstraction	It give details of storage and representation of data	It hides the internal details of Database
Integrity Constraints	Integrity Constraints are difficult to implement	Integrity constraints are easy to implement
Attributes	To access data in a file , user requires attributes such as file name, file location.	No such attributes are required.
Example	Cobol, C++	Oracle, SQL Server

The main difference between a file system and a DBMS (Database Management System) is the way they organize and manage data.

File systems are used to manage files and directories, and provide basic operations for creating, deleting, renaming, and accessing files. They typically store data in a hierarchical structure, where files are organized in directories and subdirectories. File systems are simple and efficient, but they lack the ability to manage complex data relationships and ensure data consistency.

On the other hand,

DBMS is a software system designed to manage large amounts of structured data, and provide advanced operations for storing, retrieving, and manipulating data. DBMS provides a centralized and organized way of storing data, which can be accessed and modified by multiple users or applications. DBMS offers advanced features like data validation, indexing, transactions, concurrency control, and backup and recovery mechanisms. DBMS ensures data consistency, accuracy, and integrity by enforcing data constraints, such as primary keys, foreign keys, and data types.

SQLite – Introduction

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a popular choice as an embedded database for local/client storage in application software such as web browsers. It is also used in many other applications that need a lightweight, embedded database.

SQLite is ACID-compliant and implements most of the SQL standards, using a dynamically and weakly typed SQL syntax that does not guarantee domain integrity. To use SQLite in a C/C++ program, you can use the sqlite3 API, which provides a lightweight, simple, self-contained, high-reliability, full-featured, and SQL database engine. The API is implemented as a library of C functions that can be called from your program. One of the main benefits of using SQLite is that it is very easy to get started with. To create a new database in SQLite, you simply need to create a new file on your file system and connect to it using the sqlite3 API.

```
import sqlite3
```

```
conn = sqlite3.connect('AA_db.sqlite')
```

```
cur = conn.cursor()
```

```
cur.execute('CREATE TABLE experiments (name VARCHAR, description VARCHAR)')
```

```
conn.commit()
```

```
conn.close()
```

Why SQLite?

There are several reasons why you might choose to use SQLite in your project:

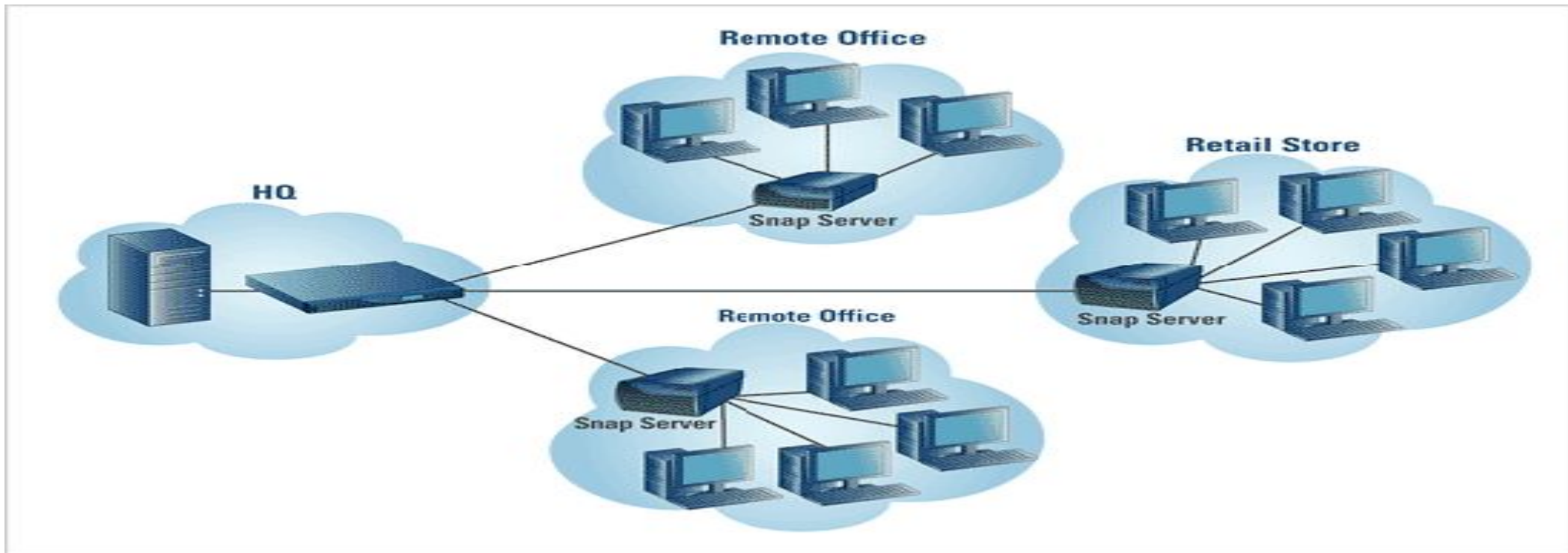
- ❖ **Ease of use:** SQLite is very easy to get started with, as it requires no setup or configuration. You can simply include the library in your project and start using it.
- ❖ **Embeddability:** SQLite is designed to be embedded into other applications. It is a self-contained, serverless database engine, which means you can include it in your application without the need for a separate database server.
- ❖ **Lightweight:** SQLite is a very lightweight database engine, with a small library size (typically less than 1MB). This makes it well-suited for use in applications where the database is embedded directly into the application binary, such as mobile apps.
- ❖ **Serverless:** As mentioned earlier, SQLite is a serverless database engine, which means there is no need to set up and maintain a separate database server process. This makes it easy to deploy and manage, as there are no additional dependencies to worry about.
- ❖ **Cross-platform:** SQLite is available on many platforms, including Linux, macOS, and Windows, making it a good choice for cross-platform development.
- ❖ **Standalone:** SQLite stores all of the data in a single file on the filesystem, which makes it easy to copy or backup the database.
- ❖ **High reliability:** SQLite has been widely tested and used in production systems for many years, and has a reputation for being a reliable and robust database engine.

Remote storage

Remote storage, alternatively called cloud storage, is a description of storage accessed over a network (remotely). For example, a networked computer may utilize remote storage to hold video files that take up a lot of disk space. Remote storage may also be used as a place for offsite backup

How to store data remotely?

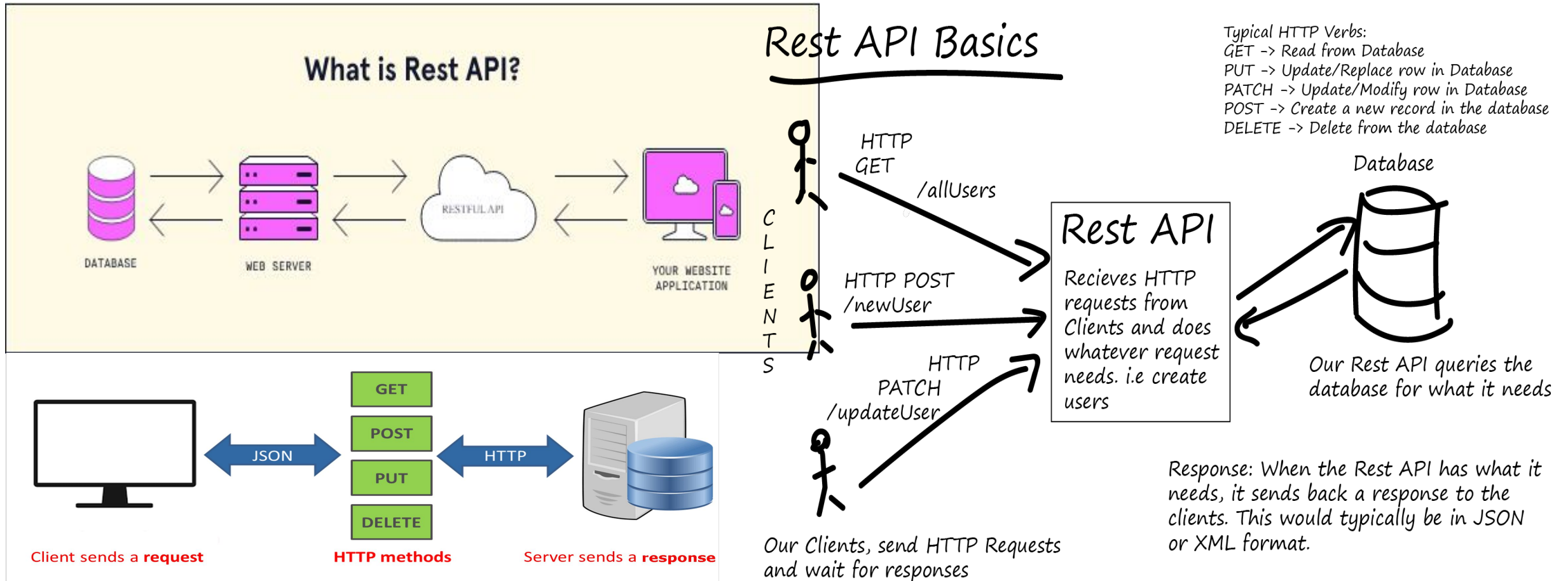
Cloud Storage uses remote servers to save data, such as files, business data, videos, or images. Users upload data to servers via an internet connection, where it is saved on a virtual machine on a physical server



What is a REST API?

An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST API is an API that conforms to the design principles of the REST, or representational state transfer architectural style. For this reason, REST APIs are sometimes referred to RESTful APIs.

Representational State Transfer (REST) is a software architecture that imposes conditions on how an API should work. REST was initially created as a guideline to manage communication on a complex network like the internet.



The Rest API will have HTTP GET, POST and PATCH endpoints. The root path will return a html template. The Rest API supports CRUD operations on sqlite database. Additionally, there is a login endpoint which creates and returns a Jwt token, and a protected endpoint which requires the said Jwt token.

REST and Python: Building APIs

REST API design is a huge topic with many layers. As with most things in technology, there’s a wide range of opinions on the best approach to building APIs. In this section, you’ll look at some recommended steps to follow as you build an API.

Identify Resources

The first step you’ll take as you build a REST API is to identify the resources the API will manage. It’s common to describe these resources as plural nouns, like customers, events, or transactions. As you identify different resources in your web service, you’ll build out a list of nouns that describe the different data users can manage in the API.

Identify Resources

The first step you’ll take as you build a REST API is to identify the resources the API will manage. It’s common to describe these resources as plural nouns, like customers, events, or transactions. As you identify different resources in your web service, you’ll build out a list of nouns that describe the different data users can manage in the API.

endpoints:

	HTTP Method	Description
000/api/users	GET	Get the list of all users in the database

Endpoints

Once you’ve identified the resources in your web service, you’ll want to use these to define the API endpoints. Here are some example endpoints for a transactions resource you might find in an API for a payment processing service:

HTTP method	API endpoint	Description
GET	/transactions	Get a list of transactions.
GET	/transactions/<transaction_id>	Get a single transaction.
POST	/transactions	Create a new transaction.
PUT	/transactions/<transaction_id>	Update a transaction.
PATCH	/transactions/<transaction_id>	Partially update a transaction.
DELETE	/transactions/<transaction_id>	Delete a transaction.

These six endpoints cover all the operations that you’ll need to create, read, update, and delete transactions in the web service. Each resource in your web service would have a similar list of endpoints based on what actions a user can perform with the API.

HTTP method	API endpoint	Description
GET	/events/<event_id>/guests	Get a list of guests.
GET	/events/<event_id>/guests/<guest_id>	Get a single guest.
POST	/events/<event_id>/guests	Create a new guest.
PUT	/events/<event_id>/guests/<guest_id>	Update a guest.
PATCH	/events/<event_id>/guests/<guest_id>	Partially update a guest.
DELETE	/events/<event_id>/guests/<guest_id>	Delete a guest.

With these endpoints, you can manage guests for a specific event in the system.

This isn't the only way to define an endpoint for nested resources. Some people prefer to use query strings to access a nested resource. A query string allows you to send additional parameters with your HTTP request. In the following endpoint, you append a query string to get guests for a specific event_id:

GET /guests?event_id=23

This endpoint will filter out any guests that don't reference the given event_id. As with many things in API design, you need to decide which method fits your web service best.

Pick Your Data Interchange Format

Two popular options for formatting web service data are XML and JSON. Traditionally, XML was very popular with SOAP APIs, but JSON is more popular with REST APIs. To compare the two, take a look at an example book resource formatted as XML and JSON.

```
{
  "title": "Python Basics",
  "page_count": 635,
  "pub_date": "2021-03-16",
  "authors": [
    {"name": "David Amos"},
    {"name": "Joanna Jablonski"},
    {"name": "Dan Bader"},
    {"name": "Fletcher Heisler"}
  ],
  "isbn13": "978-1775093329",
  "genre": "Education"
}
```

JSON stores data in key-value pairs similar to a Python dictionary. Like XML, JSON supports nesting data to any level, so you can model complex data.

Design Success Responses

Once you've picked a data format, the next step is to decide how you'll respond to HTTP requests. All responses from your REST API should have a similar format and include the proper HTTP status code.

The API returns a response that contains a list of cars. You know that the response was successful because of the 200 OK status code. The response also has a Content-Type header set to application/json. This tells the user to parse the response as JSON.

It's important to always set the correct Content-Type header on your response. If you send JSON, then set Content-Type to application/json. If XML, then set it to application/xml. This header tells the user how they should parse the data.

You also want to include an appropriate status code in your response. For any successful GET request, you should return 200 OK. This tells the user that their request was processed as expected.

Design Error Responses

There's always a chance that requests to your REST API could fail. It's a good idea to define what an error response will look like. These responses should include a description of what error occurred along with the appropriate status code.

Cloud intelligence deployed locally on IoT edge devices

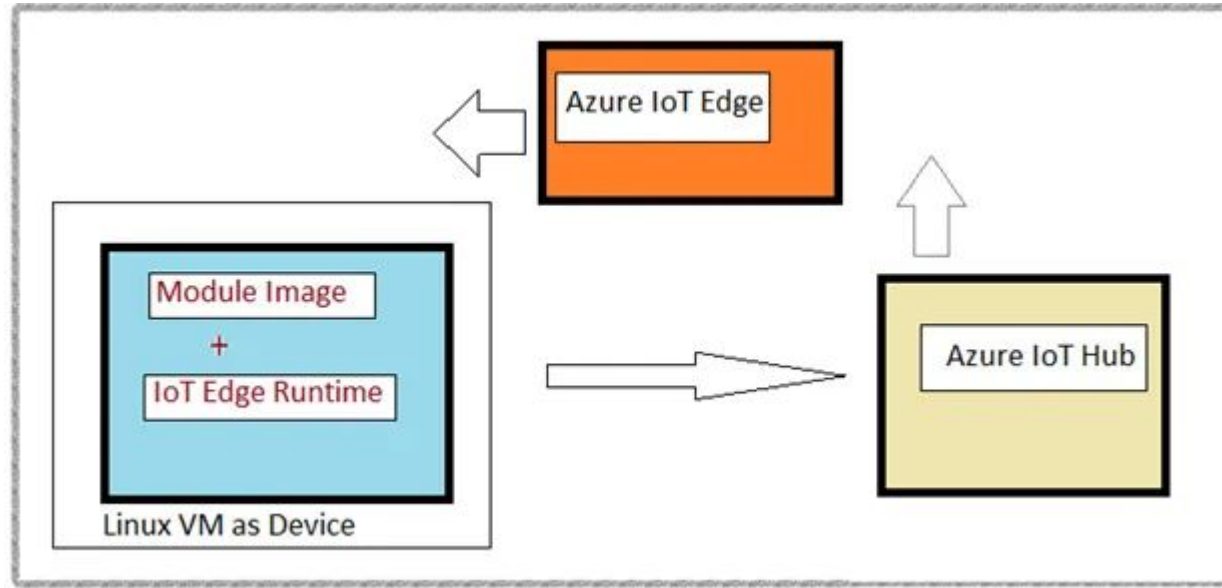
Deploy Azure IoT Edge on premises to break up data silos and consolidate operational data at scale in the Azure Cloud. Remotely and securely deploy and manage cloud-native workloads—such as AI, Azure services, or your own business logic—to run directly on your IoT devices. Optimize cloud spend and enable your devices to react faster to local changes and operate reliably even in extended offline periods.

Using Azure IoT Edge to exchange real time data from devices to IoT hub

1. Azure IoT Edge

Azure IoT Edge is an Internet of Things (IoT) service that builds on top of IoT Hub. This service is meant for customers who want to analyze data on devices, a.k.a. “at the edge”, instead of in the cloud. By moving parts of your workload to the edge, your devices can spend less time sending messages to the cloud and react more quickly to changes in status.

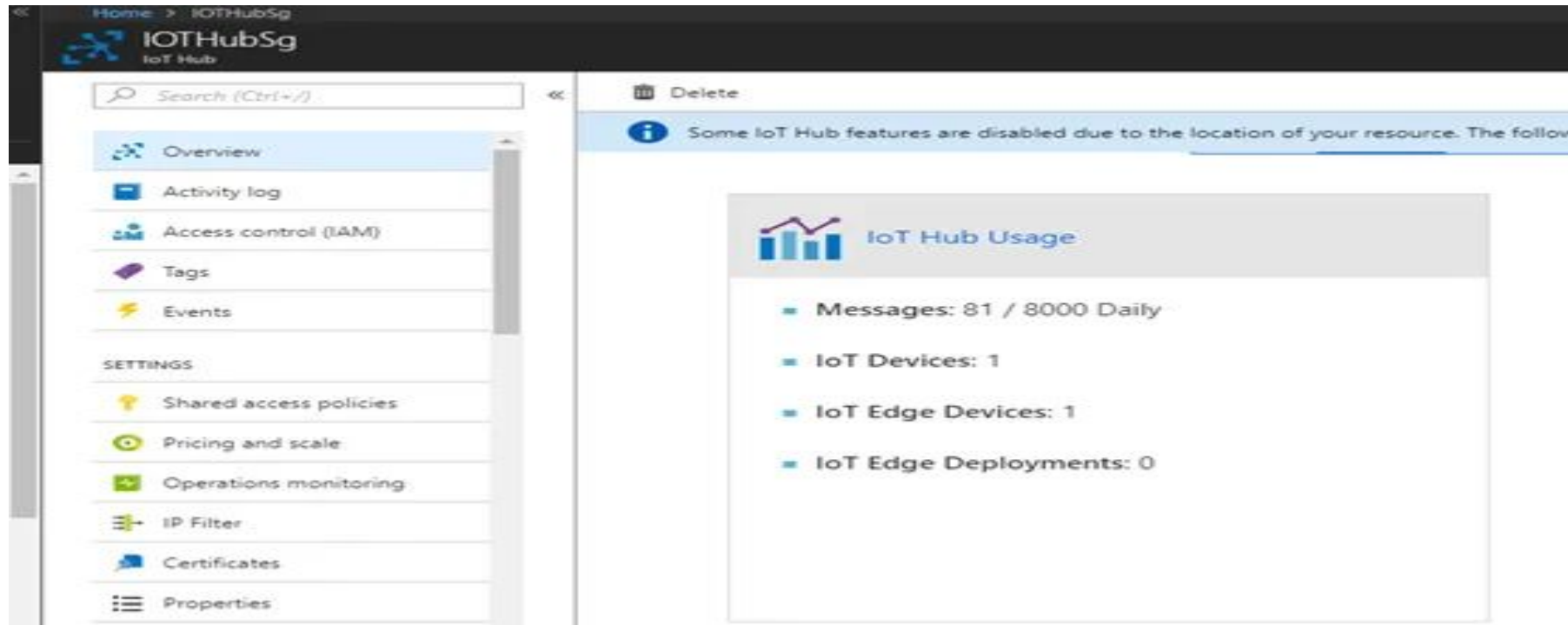
In this documentation we have turned our Linux virtual machine into an IoT Edge device. We have also created a custom module in .NET console app(.NET Core) and then we have deployed the module from the Azure portal to our device. The module that we deploy will send messages to the IoT Hub.



2. Creating Azure IoT Hub in portal

IoT hub can be created from Azure portal as well as through Azure CLI. In this project we have used azure portal and manually created it. If you are having an active azure subscription, then login to the portal and follow the steps below.

1. Sign in to the Azure portal.
2. Select Create a resource > Internet of Things > IoT Hub.
3. Enter the Subscription, Resource group, Region and Name
4. Once the above part is done you will be prompted to select the Pricing and scale tier



<https://sanajitghosh.medium.com/using-azure-iot-edge-to-exchange-real-time-data-from-devices-to-iot-hub-7895e2e2436b>

3. Creating a device identity and registering with IoT hub

To make sure that devices gets an identity so that it can communicate with the IoT hub, we need to first connect it.

To do this we need to declare this to be an IoT Edge device from the very beginning

In the Azure portal, navigate to your IoT hub.

1. Select IoT Edge then select Add IoT Edge Device.
2. Give your simulated device a unique device ID. Here we have given “azureuser” as unique device name
3. Select Save to add your device.
4. Select your new device from the list of devices.
5. Copy the value for Connection string — primary key and save it. You’ll use this value to configure the IoT Edge runtime in the next content.

4. Creating the device with IoT Edge Runtime

Our next step is to create a Linux virtual machine and install the IoT Edge runtime on it. Since we are not having any physical sensors or hardware that sends data on real time basis, so our plan will be to prepare a device that can actually create such scenario.

For doing so we need to at first spin up a Ubuntu Linux Virtual machine from azure portal.

4.1 Installing the IoT Edge Runtime

The IoT Edge runtime will be now deployed in IoT Edge device. IoT Edge device has three components. The IoT Edge security daemon starts each time an Edge device boots and bootstraps the device by starting the IoT Edge agent. The IoT Edge agent facilitates deployment and monitoring of modules on the IoT Edge device, including the IoT Edge hub. The IoT Edge hub manages communications between modules on the IoT Edge device, and between the device and IoT Hub.

Open the Linux terminal with root privileges and start using the following commands

4.2 Registering your device to use the software repository

```
> sudo -s /*root directory*/  
> curl https://packages.microsoft.com/config/ubuntu/16.04/prod.list./microsoft-prod.list  
> sudo cp ./microsoft-prod.list /etc/apt/sources.list.d/  
> curl https://packages.microsoft.com/keys/microsoft.asc | gpg -- dearmor > microsoft.gpg  
> sudo cp ./microsoft.gpg /etc/apt/trusted.gpg.d/
```

4.3 Installing the Moby engine

```
>sudo apt-get update  
>sudo apt-get install moby-engine  
>sudo apt-get install moby-cli
```

Moby engine is an open source container runtime that powers Docker, as the engine for Azure IoT Edge. This design decision enables developers to package and deploy standard Docker containers as modules on Azure IoT Edge.

4.4 Installing the IoT Edge daemon

The IoT Edge daemon installs as a system service so that the IoT Edge runtime starts every time your device boots. The installation also includes a version of hsm lib that allows the security daemon to interact with the device's hardware security.

Install it by the following command.

```
>sudo apt-get update
```

```
>sudo apt-get install iotedge
```

5. Creating the app (.NET core console app)

The device in IoT Edge requires a module, which is an image file of an application containerized in Docker and published in Docker hub. Microsoft has already provided a sample temperature Docker file in their documentation.

We will now create a module i.e a .NET core console app from scratch and then containerize it with Docker. For accessing the codes, you can follow up my GitHub handle <https://github.com/sanajitghosh/consoleappcore.git>

The code structure of the app is given in the below snapshot where we have also created an exe file from the .NET core app. You can follow the link on how to generate an exe file from a core app

<https://blog.jongallant.com/2017/09/dotnet-core-console-app-create-exe-instead-of-dll/>

6. Containerizing the application with Docker

In this part we will containerize the application with Docker. For more information about Docker and containerization please visit the link <https://docs.docker.com/engine/examples/dotnetcore/#view-the-web-page-running-from-a-container>

At first we need to create a Docker file in the Linux VM. Once the Docker file is ready you need to build and expose the ports of the container using the following commands.

7. Running the application image in Docker container

Let's see how the Docker image looks like and let's run the application to see whether the output is in the form of the desired string value

To run the Docker image and to see whether the container running use the following commands

```
>docker run <image id>
```

```
>docker ps
```

8. Pushing image to Docker hub

After we have successfully run the application in the container our next task will be to push the image in the Docker hub so that it can be accessible from anywhere. You must be noticing the repository `sg7890/iotimage` is the module for the IoT edge device. So basically we have successfully published our Docker image to Docker hub. We can easily build, run and ship the image from the Docker hub anywhere and at any time without much consideration on the platform and environment.

Below are the commands to push the Docker image to Docker hub

```
>docker tag <image id> yourhubusername/repositoryname
```

```
>docker push yourhubusername/repositoryname
```

9. Setting up the module in IoT Edge

In the previous part we have created an image module of the application and pushed it to Docker hub. Now we are going to add this module to IoT Edge.

At first we need to set up the module from Device Details. Here we have already created it and it is running. Follow these steps to set the module:

1. In the Azure portal, navigate to your IoT hub.
2. Go to IoT Edge and select your IoT Edge device.
3. Select Set Modules.
4. In the Deployment Modules section of the page, click Add then select IoT Edge Module.

10. Connecting the device with Connection string

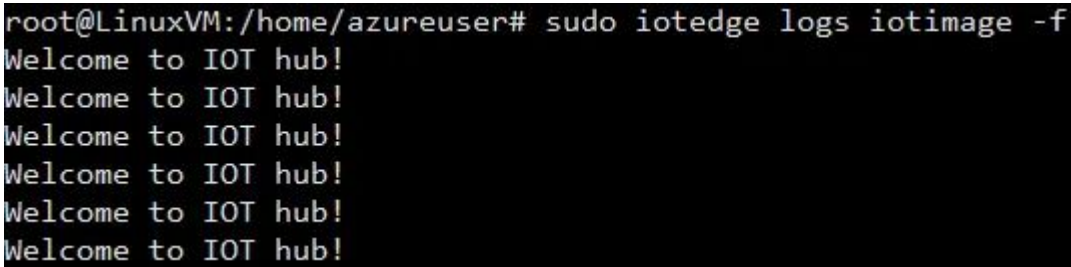
After completing the task in context 9, fill up the module details with name and image URL. Once the module is set and saved, navigate to the IoT Edge>Device Details and copy the Connection string (Primary Key)

11. View Generated data

In this Documentation, we have created a new IoT Edge device and installed the IoT Edge runtime on it. Then, we used the Azure portal to push an IoT Edge module to run on the device without having to make changes to the device itself.

Open the command prompt on the computer and run your device again. Confirm that the module deployed from the cloud is running on your IoT Edge device:

>sudo iotedge logs iotimage -f

A terminal window with a black background and white text. The prompt is 'root@LinuxVM:/home/azureuser#'. The command entered is 'sudo iotedge logs iotimage -f'. The output shows six lines of 'Welcome to IOT hub!' messages.

```
root@LinuxVM:/home/azureuser# sudo iotedge logs iotimage -f
Welcome to IOT hub!
Welcome to IOT hub!
Welcome to IOT hub!
Welcome to IOT hub!
Welcome to IOT hub!
Welcome to IOT hub!
```

Finally, we can see from the above command that our module i.e., iotimage (created from .Net application) have successfully send messages to the IoT hub via IoT Edge device.

❖ Edge analytics and machine learning at the edge: Introduction to Edge Analytics. Edge Machine Learning. Model Selection and Optimization. Collaborative Edge Learning.

Edge machine learning refers to the process of running machine learning (ML) models on an edge device to collect, process, and recognize patterns within collections of raw data.

Edge analytics is an approach to data collection and analysis in which an automated analytical computation is performed on data at a sensor, network switch or other device instead of waiting for the data to be sent back to a centralized data store.

Edge analytics has gained attention as the [internet of things \(IoT\)](#) model of connected devices has become more prevalent. In many organizations, streaming data from manufacturing machines, industrial equipment, pipelines and other remote devices connected to the IoT creates a massive glut of operational data, which can be difficult -- and expensive -- to manage. By running the data through an analytics algorithm as it's created, at the edge of a corporate network, companies can set parameters on what information is worth sending to a cloud or on-premises data store for later use

Analyzing data as it's generated can also decrease latency in the decision-making process on connected devices. For example, if sensor data from a manufacturing system points to the likely failure of a specific part, business rules built into the analytics algorithm interpreting the data at the network edge can automatically shut down the machine and send an alert to plant managers so the part can be replaced. That can save time compared to transmitting the data to a central location for processing and analysis, potentially enabling organizations to reduce or avoid unplanned equipment downtime.

Another primary benefit of edge analytics is scalability. Pushing analytics algorithms to sensors and network devices alleviates the processing strain on [enterprise data management and analytics systems](#), even as the number of connected devices being deployed by organizations -- and the amount of data being generated and collected -- increases.

What is Tiny Machine Learning?

Tiny Machine Learning is a set of machine learning libraries that can run on 32-bit microcontrollers and occupy very little memory. Increasingly, 32-bit microcontrollers are so inexpensive that embedding them into a range of sensors, meters or as front-end sentinels for more complex devices is commonplace.

As a result, this technology targets [very low power IoT applications](#) and can be used to determine when to power up or power down more complex, power hungry devices. Companies and divisions investing in this technology include ARM, Google and Tensorflow, along with several other organizations and their experts in the machine learning and embedded computing spaces.

So, how does this technology interact with the edge? Tiny Machine Learning (TinyML) will drive the need for modern edge data management in more complex devices and gateways as organizations seek to make [complex AI applications](#) work on edge-computing devices. Though TinyML does not necessarily require modern edge data management, when deployed across a grid of smart meters, sensors or as a front-end sentinel for more complex devices the requirement will manifest itself.

When describing modern edge data management, the definition is largely focused on platforms with 64-bit processors (MPUs) with megabytes of on-chip memory and gigabytes of flash or other secondary storage. These 64-bit systems power smartphones, surveillance cameras and IoT gateways, many of which are already made up of billions of devices.

For a bit of background, these devices originally used 32-bit microcontrollers (MCUs). But as device operations have become networked as well as more sophisticated and able to leverage richer underlying compute resources, 32-bit MCUs have ceded to 64-bit MPUs. As a result, TinyML can be found in trillions of devices with orders of magnitude more than that of 64-bit-based IoT systems. The question is, how does TinyML impact mobile and IoT platforms as it relates to edge data management?

TinyML drives more edge data management

Handling metadata as well as monitoring and managing those downstream 32-bit devices will be performed as a shared task across complex devices and gateways on the edge and in the cloud. In turn, [edge data management](#) requires local on-device persistent data management as well as manipulation and analysis of data shared across the edge. Let's review a few real-world examples to make this more concrete.

It's important to understand that within a single system, there might be various levels of technology residing together, such as 64-bit MPUs and 32-bit MCUs. For example, consider a smart car: though the main CPU is the 64-bit MPU, there are several 32-bit and 16-bit MCUs managing everything, from a tire pressure monitoring system to the power management of each battery in a hybrid or fully electric vehicle. In an autonomous vehicle, there might even be several MPUs supported by several MCUs. If this is sounding far-fetched, keep in mind that most mid-range Ford models have dozens of MCUs in each car. If you have MCUs analyzing specific functions, the next logical step is to use MPUs capable of more complex analysis and downstream device management, both of which come into play way before we get to the level of a Tesla.

Sounds reasonable, but let's bring this a bit closer to home. A great example I've seen recently was provided by Peter Warden, Google's evangelist for TensorFlow Lite and TinyML, at a recent ARM conference. In a recent [test of a speech recognition](#) algorithm using deep learning techniques and occupying only 19 KB with a 32-bit ARM MCU, a device was able to run on extremely low power and recognize certain command words, such as wake up.

Providing a slightly larger — but still small — machine learning routine for machine vision, a device can perform gesture recognition. In and of itself, this isn't terribly useful. However, if you were to combine it with local logic for powering the rest of the system and accompanying MPU, as well as using robust speech recognition and machine vision, you can see how these MPU-based functions would take full advantage of downstream TinyML and support MCU capabilities.

Of course, in taking in all of this information, whether through speech or gesture recognition, these IoT and mobile devices must be able to quickly and efficiently analyze, determine and perform the required action. This is where data management comes in.

The role of data management in supporting TinyML on IoT devices

Data management support for MPU-based mobile or IoT devices to the MCU-based device running TinyML can take several forms. [Initial training of the TinyML algorithms](#) are done in a back-end laboratory but, once deployed, local persistent data is stored and used to further tune the algorithm for the specific voice patterns of the actual end-users.

What this means is that edge data management is necessary to support local machine learning inference. Further, edge data management would facilitate updates to deployed inference routines as well as any other code involved in device management. Essentially, edge data management can look across the probability of false positives for improving wake-up command and accuracy by running more extensive deep learning routines across both speech and gesture recognition. In addition, it can perform predictive maintenance, including pattern baseline data on q specific device stored in the MPU.

What is machine learning at the edge?

- Machine learning is a subset of artificial intelligence (AI) in which the AI is able to perform perceptive tasks within a fraction of the time it would take a human.
- Edge computing refers to the act of bringing computing services physically closer to either the user or the source of the data. These computing services exist on what we call edge devices, a computer that allows for raw data to be collected and processed in real-time, resulting in faster, more reliable analysis.

Machine learning at the edge brings the capability of running machine learning models locally on edge devices, such as the Internet of Things (IoT).

As customer expectations rise, so does the demand for fast, secure processing power.

Every interaction between company and customer is now a mix of hybrid technologies and touchpoints that require easy access to devices, data, and applications that power new experiences and create a positive end-to-end user experience.

Traditionally, this processing takes place by transporting datasets to distant clouds via networks that can have trouble operating at full capacity due to the long journey that the data must take to travel between destinations. This can potentially result in issues ranging from latency to security breaches.

With edge computing, you can place artificial intelligence/machine learning (AI/ML)-powered applications physically closer to data sources like sensors, cameras, and mobile devices to gather insights faster, identify patterns, then initiate actions without relying on traditional cloud networks.

How to create an edge strategy

Edge computing is an important part of an open hybrid cloud vision that allows you to achieve a consistent application and operations experience—across your entire architecture through a common, horizontal platform.

While a hybrid cloud strategy allows organizations to run the same workloads in their own datacenters and on public cloud infrastructure (like [Amazon Web Services](#), [Microsoft Azure](#), or [Google Cloud](#)), an edge strategy extends even further, allowing cloud environments to reach locations that are too remote to maintain continuous connectivity with the datacenter. Because edge computing sites often have limited or no IT staffing, a reliable edge computing solution is one that can be managed using the same tools and processes as the centralized infrastructure, yet can operate independently in a disconnected mode.

In general, comprehensive edge computing solutions need to be able to:

- ❖ **Run a consistent deployment model from the core to the edge.**
- ❖ **Offer flexible architectural options to best meet connectivity and data management needs.**
- ❖ **Automate and manage infrastructure deployments and updates from your core data center to your remote edge sites.**
- ❖ **Provision, update, and maintain software applications across your infrastructure, at scale.**
- ❖ **Continue operations at remote edge sites, even when internet connectivity is not reliable.**
- ❖ **Include a robust software platform that can scale in and out.**
- ❖ **Secure data and infrastructure in security-challenged edge environments.**

How to create a machine learning strategy?

There's no single way to build and operationalize ML models, but there is a consistent need to gather and prepare datasets, develop models into intelligent applications, and derive revenue from those applications. Operationalizing these applications with integrated ML capabilities – known as MLOps – and keeping them up to date requires collaboration amongst data scientists, developers, ML engineers, IT operations, and various DevOps technologies.

By applying DevOps and GitOps principles, organizations automate and simplify the iterative process of integrating ML models into software development processes, production rollout, monitoring, retraining, and redeployment for continued prediction accuracy.

This process can essentially be broken down into 4 steps:

- ❖ **Train:** ML models are trained on Jupyter notebooks on Red Hat OpenShift.
- ❖ **Automate:** Red Hat OpenShift Pipelines is an event-driven, continuous integration capability that helps package ML models as container images by:
 - Saving** the models ready for deployment in a model store.
 - Converting** the saved models to container images with Red Hat OpenShift build.
 - Testing** the containerized model images to ensure they remain functional.
 - Storing** the containerized model images in a private, global container image registry like Red Hat Quay, where the images are analyzed to identify potential issues, mitigating security risks and geo replication.
- ❖ **Deploy:** Declarative configuration managed by Red Hat OpenShift GitOps automates the deployment of ML models at scale, anywhere.
- ❖ **Monitor:** Models are monitored for reliability, speed, scale, etc. with the tooling from one of our ecosystem partners, and updated with retraining and redeployment, as needed.

Resource management and task offloading strategies: Task Offloading, Edge-Cloud Collaboration, Dynamic Resource Provisioning.

Offloading Tasks in the Edge–Cloud Environment: Important Factors

From the application perspective, there are several factors that affect the decision about offloading tasks and the overall service time for IoT applications. For example, latency-sensitive applications could lead to a significant delay in computation and communication, which should be considered in efficiently managing Edge–Cloud resources. The following are some of the main factors.

- Application characteristics: This is when there are some tasks that are working jointly, such as direct acyclic graph (DAG). In this case, if the resource manager offloads the tasks in different locations, the communication time will increase [54,55]. Therefore, considering the impact of task dependency could improve the application's QoS.
- Application tasks variation: In general, any IoT application consists of several tasks, which vary in their functionality; thus, they will have different demands of the CPU or the amount of transferred data. The place where the task is offloaded will therefore significantly affect the service time.
- Types of computational resources: Edge–Cloud resources are heterogeneous, having either different hardware capabilities or different hardware architecture, e.g., graphic processing units (GPUs) and field programmable gate arrays (FPGAs). Therefore, the resource manager needs to assign the tasks to the most appropriate hardware to get the best performance out of the resources.
- Users' mobility: Since some IoT applications require mobility support, when the task is offloaded to a local edge node, the IoT device may move to another area covered by a different edge node. This could lead to a significant degradation in service time [56].

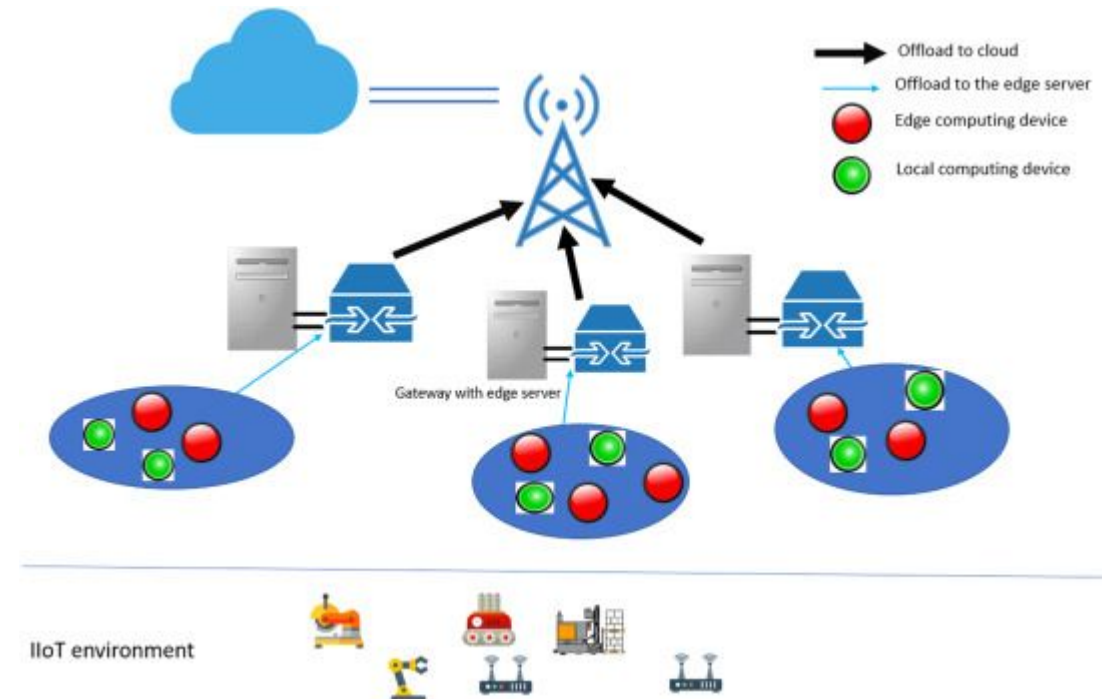
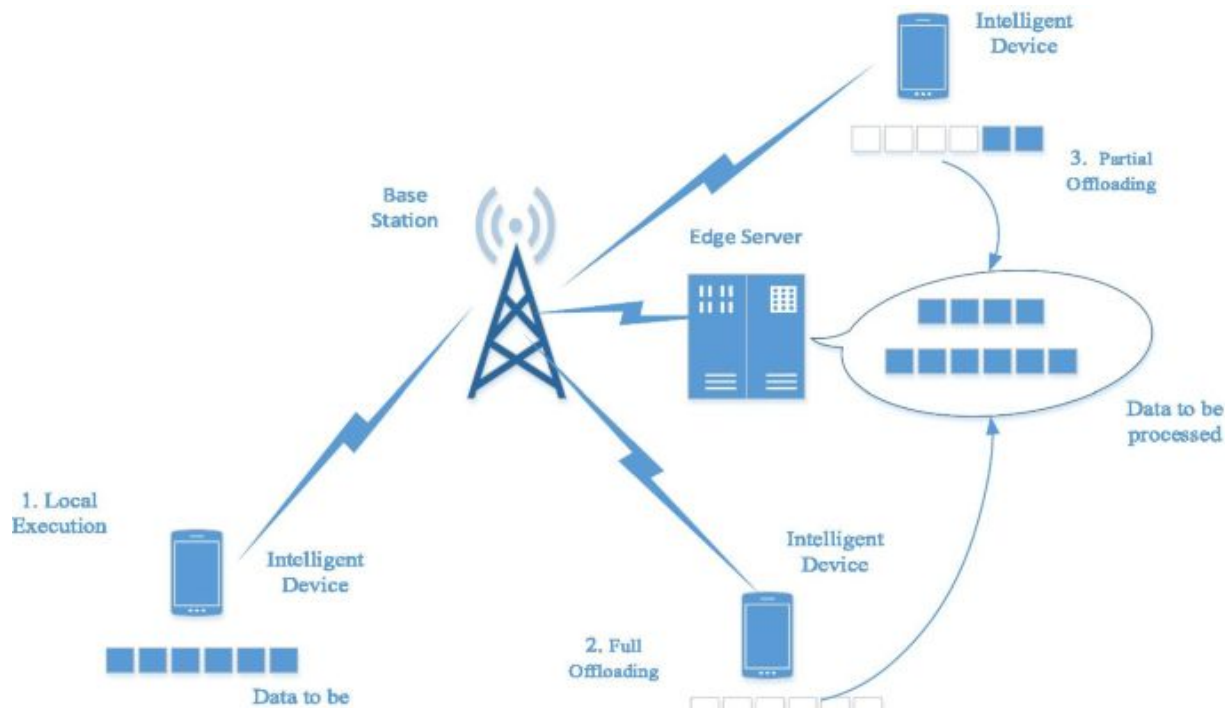
In this paper, we particularly focus on the impact of task variations in terms of computational and communication demands of IoT latency-sensitive applications, as well as the offloading of tasks to heterogeneous computational resources.

<https://www.techscience.com/iasc/v30n1/43965/html>
!

Resource management and task offloading strategies: Task Offloading, Edge-Cloud Collaboration, Dynamic Resource Provisioning.

What is task offloading in edge and cloud computing?

Cooperative task offloading is a technique used in edge-cloud networks to improve the performance of distributed systems. In the distributed approach, tasks are split among devices in the network, such as edge devices and cloud servers, to optimize resource usage and reduce the workload on individual devices



Task offloading refers to the user equipment processing some computationally intensive applications and uploading the data processing these applications to the edge server through wireless transmission under the condition of weighing continuous or other indicators.

<https://www.techscience.com/iasc/v30n1/43965/htm>

Offloading Tasks in the Edge–Cloud Environment: Important Factors

From the application perspective, there are several factors that affect the decision about offloading tasks and the overall service time for IoT applications. For example, latency-sensitive applications could lead to a significant delay in computation and communication, which should be considered in efficiently managing Edge–Cloud resources. The following are some of the main factors.

- **Application characteristics:** This is when there are some tasks that are working jointly, such as direct acyclic graph (DAG). In this case, if the resource manager offloads the tasks in different locations, the communication time will increase. Therefore, considering the impact of task dependency could improve the application's QoS.
- **Application tasks variation:** In general, any IoT application consists of several tasks, which vary in their functionality; thus, they will have different demands of the CPU or the amount of transferred data. The place where the task is offloaded will therefore significantly affect the service time.
- **Types of computational resources:** Edge–Cloud resources are heterogeneous, having either different hardware capabilities or different hardware architecture, e.g., graphic processing units (GPUs) and field programmable gate arrays (FPGAs). Therefore, the resource manager needs to assign the tasks to the most appropriate hardware to get the best performance out of the resources.
- **Users' mobility:** Since some IoT applications require mobility support, when the task is offloaded to a local edge node, the IoT device may move to another area covered by a different edge node. This could lead to a significant degradation in service time .

1 Application Characteristics (Computational and Communication)

The computation tasks of IoT applications can be characterized by their needs for computational resources (i.e., CPU and RAM), as well as for communication (e.g., uploading and downloading data). IoT offloaded tasks usually vary in the degree of reliance on such resources between light and heavy. Low computation and communication demands include healthcare applications , whereas high computation and communication demands are involved in online video gaming . As depicted in fig. some tasks require more computation time owing to the intensive processing, and others require more network time owing to the transfer of a massive amount of data.asks involved in IoT latency-sensitive applications that require higher computational demands should for preference be processed in the Cloud, since the edge resources are limited, but this also depends on the demands of the transferred data. Conversely, tasks that require moving a large amount of data need to be processed in the Edge to avoid the long latency in the network backhaul.

Basically, the task completion time consists of three essential components, namely computation time, network time, and the site where the task is scheduled (e.g., which server type). The server can be located on the local edge, at other nearby edge nodes close to IoT devices, or in the Cloud.

The computation time of the IoT task depends on the number of instructions, measured as million instructions per second (MIPS), that need to be executed and the processing speed of the hosted resources, e.g., virtual machine (VM). The number of instructions represents the computational volume of an IoT task. As mentioned earlier, IoT tasks can range from a small number of code instructions to a high number, depending on the application. This factor, along with network conditions, determines where the tasks should be offloaded. For example, it is not logical to offload the tasks with vast amounts of data to the Cloud when edge resources are available, because this will increase the overall service time. However, these two factors must be considered together; thus, we need to understand and investigate the impact of each independently. The network time of an IoT task depends on the amount of data to be uploaded and downloaded, as well as the transmission latency between the sender and the receiver. In our case, the sender will be the IoT devices, and the receiver could be (local edge, other collaborative edges or the Cloud). Moreover, for each task, the amount of transferred data can vary based on the IoT application.

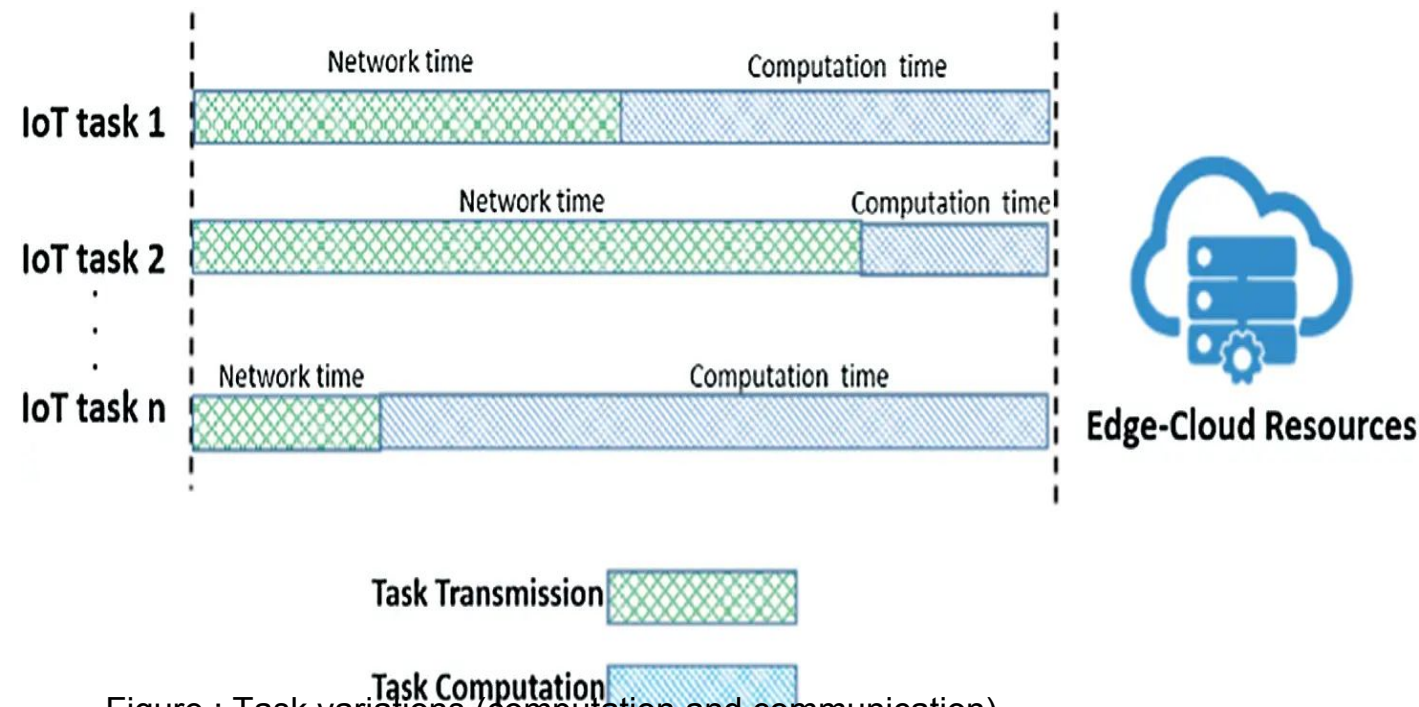
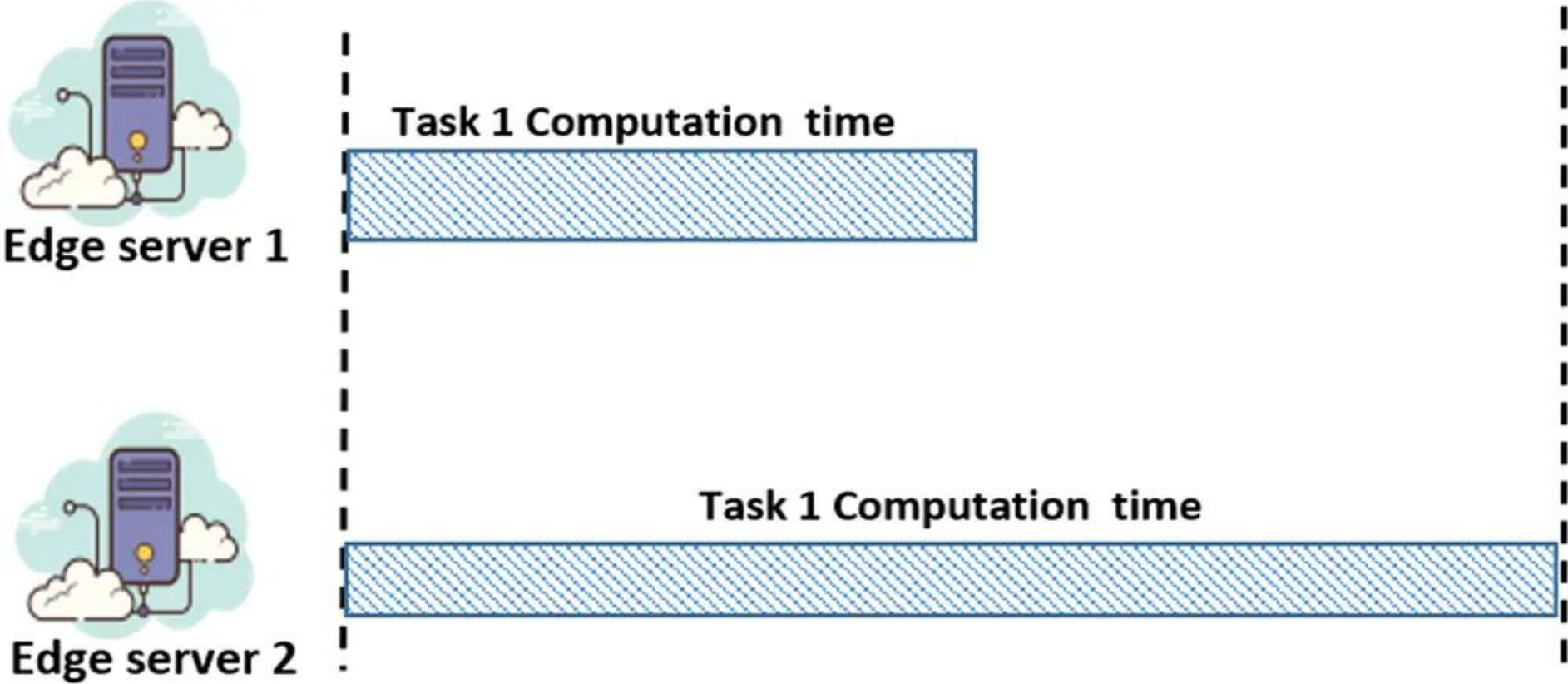


Figure : Task variations (computation and communication)

2 Computational Resource Heterogeneity

In terms of computational resources, both IoT devices and edge servers are heterogeneous. Consequently, for latency-sensitive applications, the computational resource needed to execute the task must be estimated in order to minimize the overall service time. Because of resource heterogeneity, this means there are some servers that are better than others in terms of capabilities, notably in handling the offloading tasks faster (fig). The quantity of required computational resources varies for each task. Thus, heavy tasks require a powerful machine to process the jobs more quickly. Given that a performance method is needed that measures the end-to-end IoT service effectiveness, considering both the computational and communication demands of offloading tasks, the following questions need to be answered:

- How do different applications' characteristics, including computation and communication demands, impact on the overall service time?
- How do different computational resources (e.g., different VM capabilities) impact on the overall service time?



Open Challenges

As far as offloading tasks in the Edge–Cloud environment is concerned, there are several open challenges remaining, some of which are discussed in this section.

Task dependency: There is a lack of studies on the issue of offloading tasks because they do not consider the dependency of the tasks. To be more precise, allocating tasks (that are dependent on other tasks' results) to different resources in the Edge–Cloud environment could lead to poor QoS for IoT applications. Thus, research is required into the application components and how they interact with one another.

Taking this factor into account may help to enhance both overall system performance and to yield application QoS.

Applications require a high degree of mobility: Offloading tasks of applications that require mobility support, such as self-driving cars, crewless aircraft vehicles, and mobile devices, is an open challenge. For example, processing users' tasks of an application, while moving from a covered area to another covered area, could lead to high network latency or process failure [6]. Although several researchers are attempting to tackle this issue, it remains a challenge.

Workload prediction: IoT tasks are dynamically changing and each task's procedure may have different execution times. Also, IoT devices are mobile, and the number of devices may increase in some areas; thus, the workload will be increased for the connected edge node. In turn, the amount of IoT workload will change dynamically over the Edge–Cloud system, which could lead to service performance degradation. Therefore, there is a need for workload prediction modeling, which could help to yield application QoS and maintain the performance of the Edge–Cloud system.

What is dynamic resource provisioning in cloud computing?

Secured Dynamic Provisioning of resources is implemented during the migration process from physical to virtual to a cloud model. At each point, security measures are taken to ensure that the data is transferred without interruption.

Cloud Provisioning Challenges

Like any other technology, cloud provisioning also presents several challenges, including:

Complex management and monitoring: Organizations may need several provisioning tools to customize their cloud resources. Many also deploy workloads on more than one cloud platform, making viewing everything on a central console more challenging.

Resource and service dependencies: Cloud applications and workloads often tap into basic infrastructure resources, such as computing, networking, and storage. But public cloud service providers offer higher-level ancillary services like serverless functions and machine learning (ML) and big data capabilities. Such services may carry dependencies that can lead to unexpected overuse and surprise costs.

Policy enforcement: User cloud provisioning helps streamline requests and manage resources but requires strict rules to make sure unnecessary resources are not provided. That is time-consuming since different users require varying levels of access and frequency. Setting up rules to know who can provide which resources, for how long, and with what budgetary controls can be difficult.

Cloud Provisioning Benefits

Cloud provisioning has several benefits that are not available with traditional provisioning approaches, such as:

Scalability: The traditional information technology (IT) provisioning model requires organizations to make large investments in their [on-premises](#) infrastructure. That needs extensive preparation and forecasting of infrastructure needs since on-premises infrastructures are often set up to last for many years. The cloud provisioning model, meanwhile, lets companies simply scale up and down their cloud resources depending on their short-term usage requirements.

Speed: Organizations' developers can quickly spin up several workloads on-demand, so the companies no longer require IT administrators to provide and manage computing resources.

Cost savings: While traditional on-premises technology requires large upfront investments, many cloud service providers let their customers pay for only what they consume. But the attractive economics of cloud services presents challenges, too, which may require organizations to develop a cloud management strategy.

Tools for Cloud Provisioning:

Google Cloud Deployment Manager

IBM Cloud Orchestrator

AWS CloudFormation

Microsoft Azure Resource Manager

3 Cloud Provisioning Types

Advanced Cloud Provisioning

Also known as “post-sales cloud provisioning,” customers get the resources upon contract or service sign up. They sign formal contracts with the cloud service provider. The provider then prepares and delivers the agreed-upon resources or services. The customers are charged a flat fee or billed every month.

Dynamic Cloud Provisioning

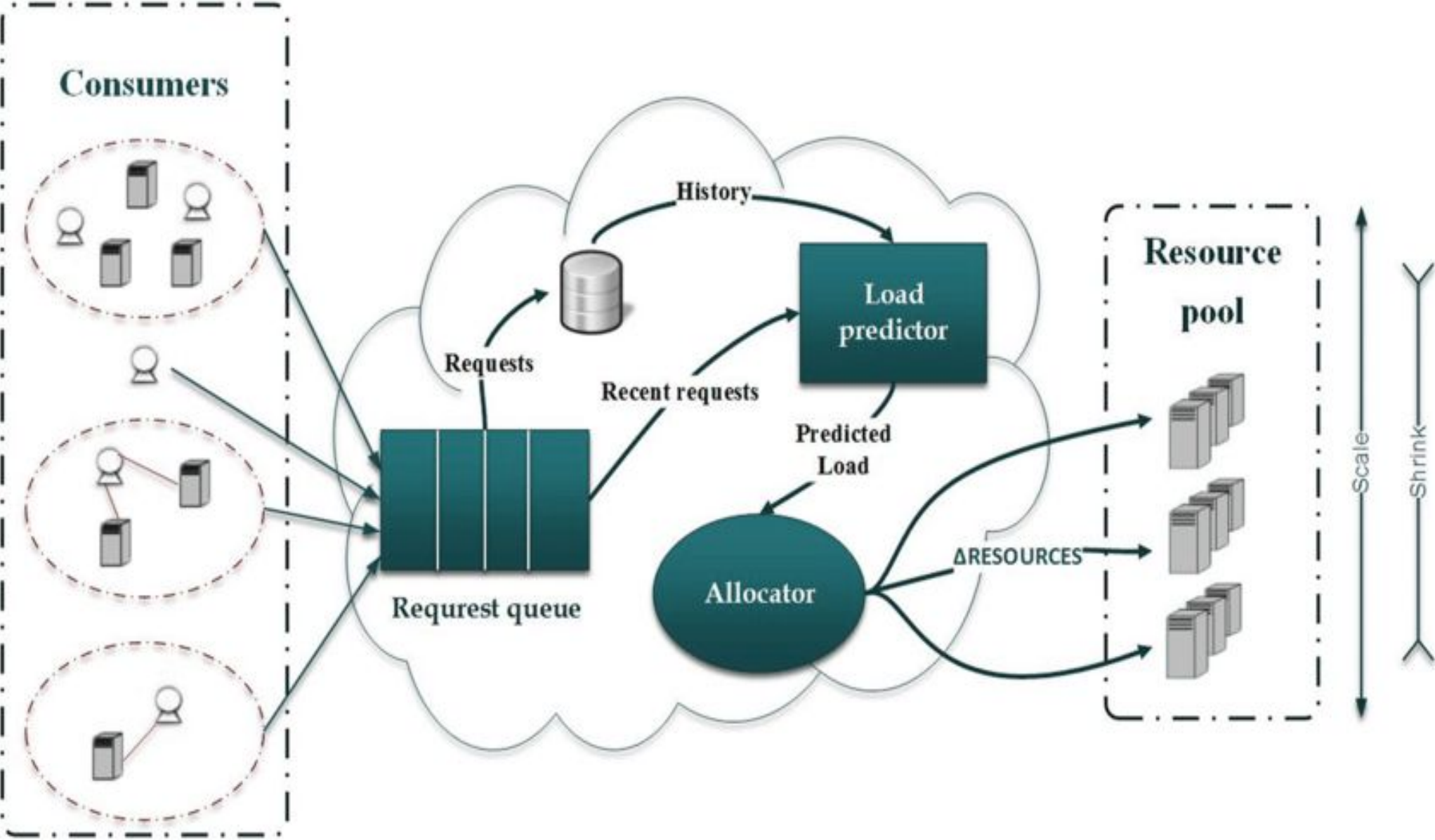
Also referred to as “on-demand cloud provisioning,” customers are provided with resources on runtime. In this delivery model, cloud resources are deployed to match customers’ fluctuating demands. Deployments can scale up to accommodate spikes in usage and down when demands decrease. Customers are billed on a pay-per-use basis. When this model is used to create a hybrid cloud environment, it is sometimes called “cloud bursting.”

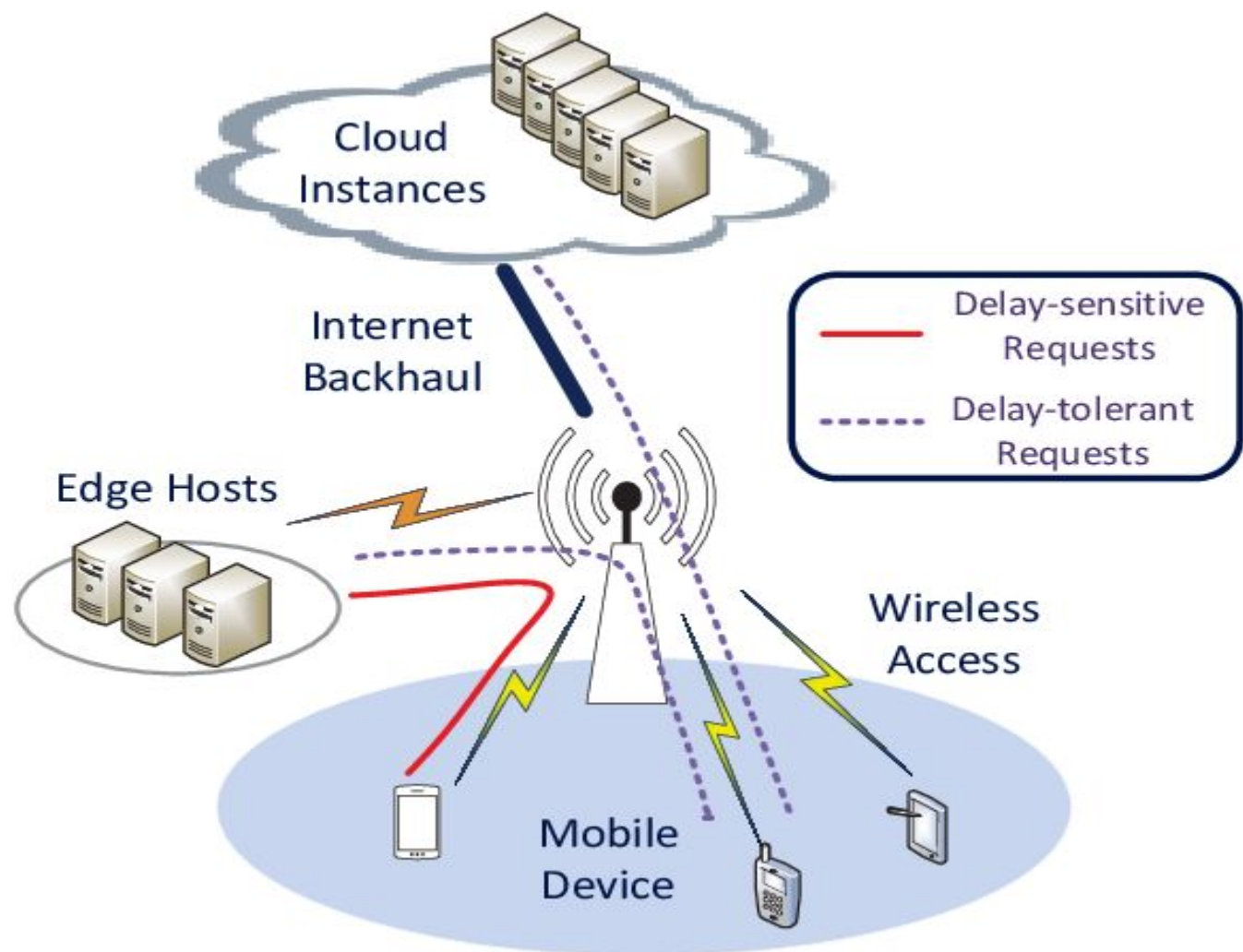
User Cloud Provisioning

In this delivery model, customers add a cloud device themselves. Also known as “cloud self-service,” clients buy resources from the cloud service provider through a web interface or portal. The model usually involves creating a user account and paying for resources with a credit card. The resources are quickly spun up and made available for use within hours, if not minutes. An example of this includes an employee purchasing cloud-based productivity applications via Microsoft 365 or G Suite.

Dynamic provisioning or On-demand provisioning: With dynamic provisioning, the provider adds resources as needed and subtracts them as they are no longer required. It follows a pay-per-use model, i.e. the clients are billed only for the exact resources they use. Consumers must pay for each use of the resources that the cloud service provider allots to them as needed and when necessary. The pay-as-you-go model is another name for this. “Dynamic provisioning” techniques allow VMs to be moved on-the-fly to new computing nodes within the cloud, in situations where demand by applications may change or vary. This is a suitable choice for programs with erratic and shifting demands or workloads. For instance, a customer might want to use a web server with a configurable quantity of CPU, memory, and storage. In this scenario, the client can utilize the resources as required and only pay for what is really used. The client is in charge of ensuring that the resources are not oversubscribed; otherwise, fees can skyrocket.

<https://www.geeksforgeeks.org/resource-allocation-methods-in-cloud-computing/>





Edge caching and data synchronization: Introduction to Edge caching and data synchronization, Benefits of Edge Caching and Data Synchronization, Challenges in Edge Caching and Data Synchronization.

Slow websites kill conversions. You've probably experienced this when trying to purchase something online, or streaming videos from a service that keeps buffering. The slower the page takes to load, the more anxious you get. This is why every technical team and business owner should take an interest in caching responses and enhancing performance. There are lots of ways you can speed up your site, app, or platform, but one of the best ways is to cache at the edge; this technique is called edge caching. Edge caching is a technology that improves the performance of applications and accelerates the delivery of data and content to end users. By moving content delivery to the edge of your network, you can make your platform more performant by speeding up the process of delivering content from the global network.

What is edge computing?

Edge computing is an evolution of cloud-computing patterns. In the cloud-based approach, computations are all done in a centralized location. Your application might have users in Germany, Australia, and Africa, but your centralized server(s) are in North America. This can cause significant latency as requests and data need to do a round-trip across the world from a user's device to the centralized/origin server and back. Edge computing breaks the traditional idea that processing and data storage must be done in a centralized location. With edge computing, processing can be performed at the edge of the network where it's closer to the end users, which means that your data won't have to travel across the network. Edge computing has several advantages, but the most notable is that it improves user experience by reducing latency for applications, services, and IoT devices. In addition to providing faster services for end users, it also helps reduce cloud costs by reducing the amount of traffic being sent to centralized data centers (which can be located anywhere in the world).

How does edge computing relate to edge caching?

Caching has been a popular performance-boosting technique well before edge computing was first introduced. Before edge computing, caches were stored on a server or device located within the network, such as a web server or a content delivery network (CDN). With the rise of edge computing, the edge has moved beyond being merely a point of access to the network to being used as a place for data processing and storage; in the context of edge caching, it is increasingly used as a location to store cache. Edge caching allows businesses to cache frequently accessed data or content on a server or device that is located closer to the end-users, such as an edge server or a device at the network edge. Edge caching is a great way to improve performance, but that doesn't mean every platform should use edge caching. Before you rally your team to deploy edge caching, let's examine the benefits, drawbacks, and use cases of this strategy.

Benefits of edge caching

More people have access to the internet than ever before, it has become an essential part of modern life. The world of today is flooded with a constant stream of data and media. Keeping up with this growing demand for content has become an increasingly important concern for business owners and technical teams. The trend toward edge-based solutions, such as edge caching, is a response to the need for companies to maintain a competitive presence in the market. Storing content and data in a cache located at the edge can provide many benefits, including:

Speed: The closer your content is to the end user, the faster it will load. Edge caching can reduce latency by several orders of magnitude — up to milliseconds or even microseconds on mobile networks!

Reliability: Edge caching can provide high availability by serving cached content when failures are occurring at the origin or along transit paths. This ensures that your platform will be available to your users, even when other parts of the network are experiencing problems.

Security: Edge caches are often located in private networks, which enhances the security of sensitive data like personally identifiable information. Eliminating the need to pass data through a public network is especially useful for industries with regulatory compliance concerns like healthcare and financial services.

Reduced costs: By reducing the load on your origin servers and network traffic, you can reduce the amount of workload on your cloud server. This could mean significant savings on your cloud bill.

Limitations of edge caching

As with anything, edge caching is not without its limitations.

Some drawbacks to consider are:
Limited capacity: The amount of data and content that can be cached at the edge is limited by the capacity of edge servers and devices. Although technological advances have increased capacity in recent years, you may still find yourself limited at this point in time.

Inconsistency: If the data or content you are caching frequently changes, you may find that your cache becomes obsolete quickly. This can be a downside to caching in general and highlights why it is important to carefully select which data or content to cache. In addition, implementing cache-invalidation techniques and expiration times can help mitigate this problem.

Complexity: Edge caching can add a layer of complication to your infrastructure and tech stack, especially if you have to consider invalidation techniques and expiration dates. Depending on the size of your team, and other priorities you might have, building and maintaining an architecture that enables edge caching may be too large for your organization at this point in time.

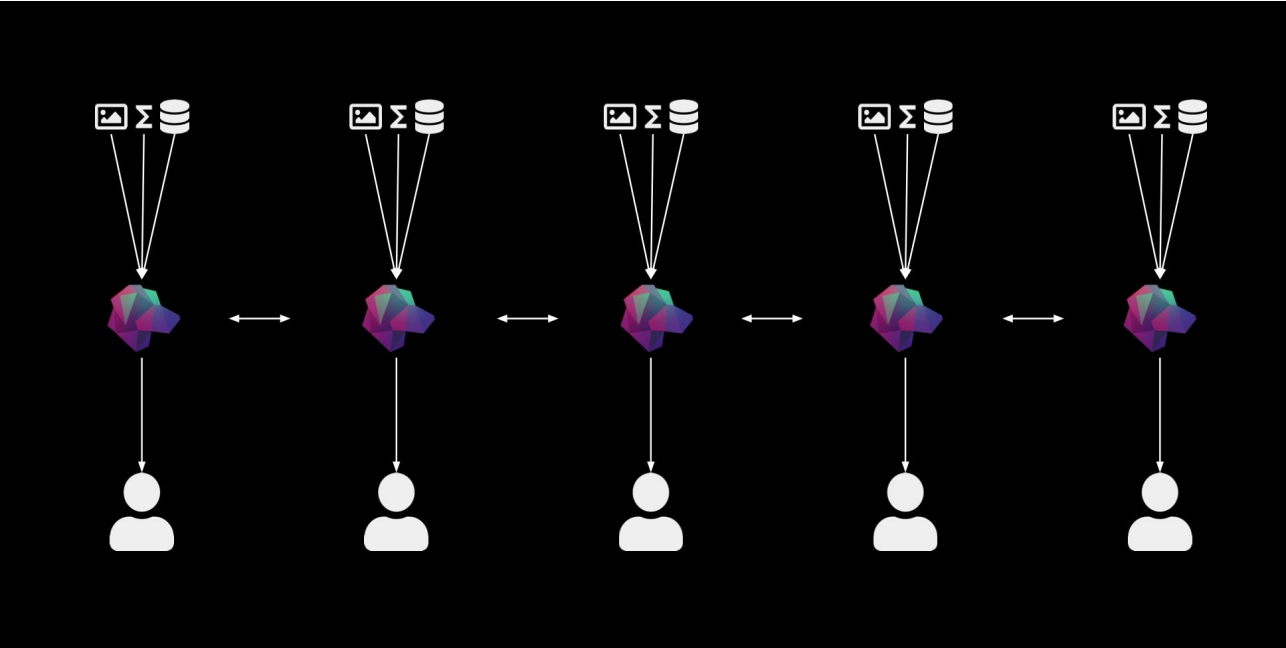
Common use cases for edge caching

Now that you're familiar with the advantages and disadvantages of edge caching, it's time to evaluate your platform against some common use cases for edge caching.

Gaming platform: Video games are resource-intensive, requiring huge amounts of graphics and audio files. Caching these assets at the edge can drastically improve gameplay performance. In addition to improving performance, edge caching can also lower costs by reducing the amount of traffic that has to traverse the public internet before going back into the private network.

Video streaming: Streaming services can encounter high levels of network congestion which can lead to poor performance and user experience. In order to alleviate this problem, video files can be cached at the edge of the network. This will allow devices to stream video without sending a request to a centralized server; improving performance and increasing the platform's availability.

IoT devices: IoT devices are becoming more powerful, complex, and interconnected, with more capabilities than ever before. This means that there is an increasing amount of data being sent from one device to another each day. This also means that there are many more points where network congestion can occur. Edge caching can help to reduce bandwidth usage by moving critical data to the device itself. IoT devices often require real-time interactions, edge caching dramatically improves performance as IoT devices can access the cached data almost instantaneously. Additionally, database queries and API responses can be cached on an edge device, allowing dynamic processing to be significantly faster as it's done close to the IoT device which significantly reduces latency. Moving cache to the edge is becoming increasingly attractive as more and more powerful IoT devices hit the market.



Usefull Links

<https://www.electricaltechnology.org/2021/11/transducer.html>

<https://techbeacon.com/enterprise-it/data-management-edge-time-get-partitioning>

<https://learn.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning>

<https://sanajitghosh.medium.com/using-azure-iot-edge-to-exchange-real-time-data-from-devices-to-iot-hub-7895e2e2436b>

<https://www.techscience.com/iasc/v30n1/43965/html>

!

<https://www.geeksforgeeks.org/resource-allocation-methods-in-cloud-computing/>

<https://uwaterloo.ca/broadband-communications-research-lab/sites/ca.broadband-communications-research-lab/files/uploads/files/cerpd.pdf>

