WHITE PAPER

4 Steps to Select the Right Database for Your IoT Solution

Roshan Kumar, Redis

CONTENTS

Introduction

Choosing the right database platform(s) for IoT solutions is quite daunting. One challenge is location—IoT solutions can be distributed across geographical regions. As opposed to a centralized cloud-based approach, many solutions are adopting a combination of both fog computing at the edge and cloud computing. Critical applications that collect large volumes of data and rely on low latency are getting pushed towards the edge, yet there's a need to connect the edge servers with the cloud. Therefore, database platforms must offer the flexibility to process the data at the edge, and synchronize between the edge servers and the cloud.

Another challenge is the variety of features needed to support IoT use cases. The capabilities you want in your database could range from real-time data streaming; data filtering and aggregation; near-zero latency read operations; instant analytics; high availability; geo distribution; schema flexibility, etc. This article walks you through the four steps to choosing the right database platforms for your IoT Solutions:

1. Identify the data needs for your solution,
2. Break down your solution into independent services and list their data needs,
3. Group your services by their data requirements and select the right database(s),
4. Evaluate cost and resource efficiency

Step 1. Identify the data needs for your solution

A complete IoT solution is much more than simply connecting a device to the internet. IoT solutions depend on collection and processing of data from connected devices, making intelligent decisions such as triggering notifications or actions, computing real-time analytics, gleaning patterns from historical data, and so on.

To simplify our discussion, let's assume a sample generic IoT solution. In our solution, we have sensors and actuators that are installed across the enterprise. Thousands of sensors and actuators connect with an edge server. The IoT solution collects the data from all the sensors continuously, and makes real-time decisions to control the sensors and actuators, alert the system monitors of unusual activity, and provide a historical view of the analytics to the end users.

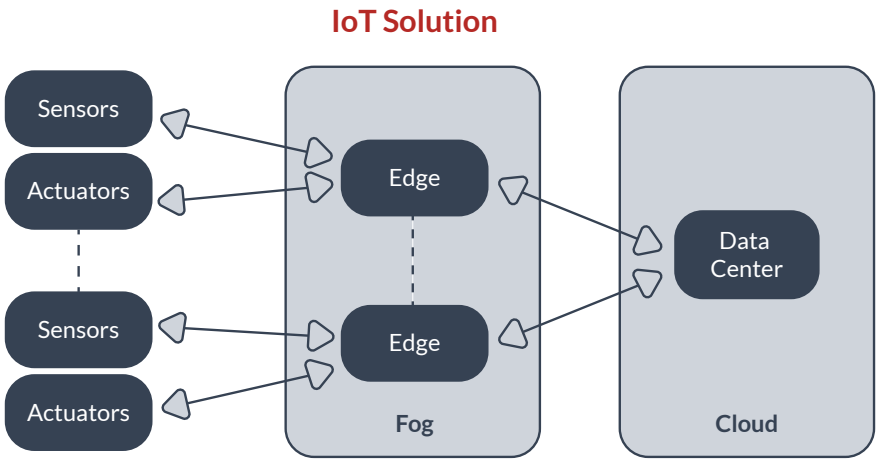

Figure 1. Setting the stage for a sample IoT solution

Before you decide on the services and the databases that go with them, it's necessary for you to have clarity on ***what*** you are doing with your data, and ***where***. These questions help you understand and prioritize your data needs:

- What is the topology of the edge servers? What's the system configuration of the edge server—how much RAM and storage is available?
- What data processing and decision making is delegated to the edge servers?
- Is the cloud solution deployed in one region, or is it dispersed across multiple regions?
- What is the system configuration of the cloud servers?
- What is the volume of data transferred from the device to the edge server, and from the edge server to the central server? What is the estimate for the peak volume?
- What different formats of the data arrive from the devices, and what are the normalization, translation and filtering techniques applied on the data?
- Does the IoT solution control the devices or actuators? If yes, what device features are controlled by the IoT solution? Do they require a real-time response?
- What is the expected turnaround time to respond to any critical event?
- What are the real-time analytics computed from the incoming data?
- What are the business insights derived from the historical data?
- What are the external services that depend on the data?

Step 2. Break down your solution into independent software services

The previous step focused on questions related to the "what" and "where" of your solution. Now you will answer the how. In this step you will design the software services or components that perform independent, specific tasks.

When you break down the sample IoT solution described earlier into independent services, you may get the design shown in Figure 2. The IoT solution itself is distributed geographically, where some of the components are deployed on the edge network, and others at a centralized location.

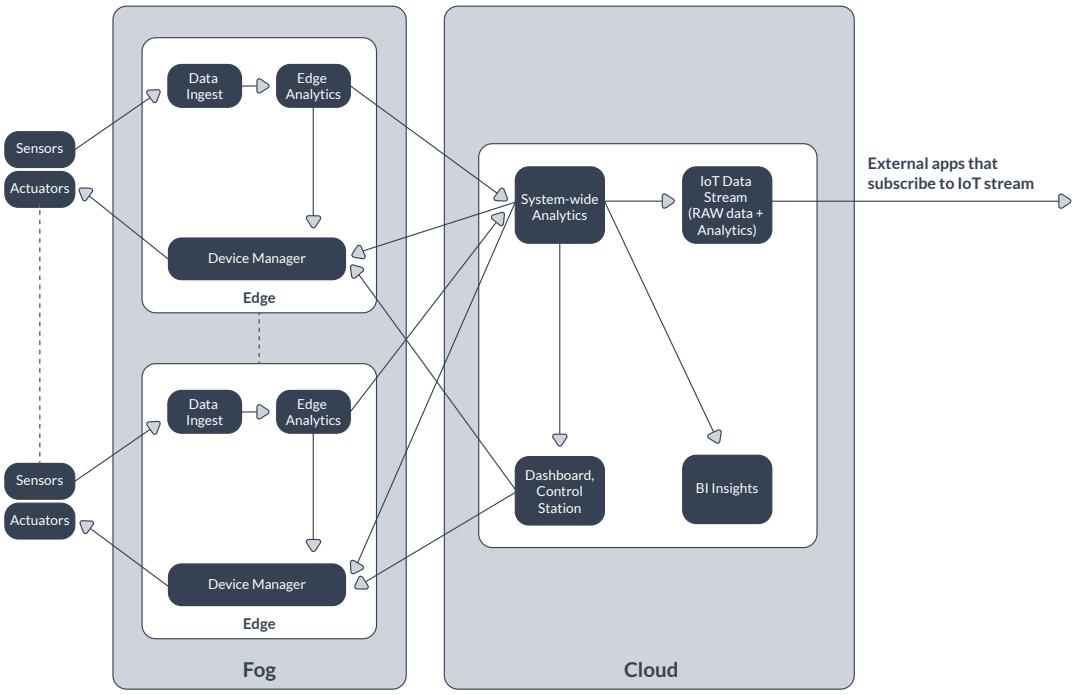

Figure 2. Software services of our example IoT Solution

Let's now break up the architecture into services and analyze their responsibilities and data needs:

	Service	Role	Database Needs
1	Data Ingest	This is the first stop to collect and store all the logs and messages from the devices.	The data ingest server must support high-speed write operations, as occasionally the data may arrive in bursts. In other words, it must ensure that the data is captured and not lost under any unusual circumstances.
2	Edge Analytics	The edge analytics server performs the data translation, classification, aggregation, filtering and functions on the incoming data. It's responsible for all real-time decision making at the edge that feeds into the device manager.	The database must support high-speed read and writes with sub millisecond latency while providing tools and commands with which to perform complex analytical computations on the data.
3	Device Manager	The device manager is responsible for communicating messages to the end points or the devices.	The messages sent to the devices are control signals that need to be accessed and delivered with minimum latency.
4	System-wide Analytics	This service collects data from all the edge servers and performs data transformation and analytics operations at the central server.	The database must provide commands to perform analytical computations on the data, and store the data as long as required by the analytics engine.
5	Command and Control Dashboard	This service delivers the visual representation of the current state of the IoT ecosystem.	The database must maintain data that is both current and accurate. It must deliver data in read operations with sub-millisecond latency.
6	Business Intelligence	This component delivers reports, queries and inferences from historical data.	The database must store data for a long period of time in a cost effective way, while providing tools to query and analyze the data.
7	IoT Data Stream Outlet	This outlet collects the IoT data, normalizes it to a common format and pushes it to the subscribers.	This database must have the ability to perform data transformation operations efficiently and support publish and subscribe capabilities.

Table 1. Breakdown the services and identify their data needs

Step 3: Group your services by their data needs and select the right databases

Now that you have broken down your solution into software services and identified their data needs, the next step will be to choose the right database(s). Before you decide on the database, let's plot a time-speed graph for the data of each service. The time axis tells you how long the data stays in the database unchanged, and the speed axis denotes the data read/write speed required by the service.

When we plot the graph for our IoT example and attach the services to the plot, the graph looks like the one below. For example, the data is constantly coming in and going out of the Data Ingest Server. Therefore, the data stays in the database for a very short period of time. At the same time, the data may arrive in high volume and velocity. Therefore, we need a high-speed database with low latency to hold the data for the ingest service. The analytics engine reads the data stream constantly, makes real-time decisions, and makes relevant modifications to the database on an ongoing basis. The business intelligence service, on the other hand relies on historical data.

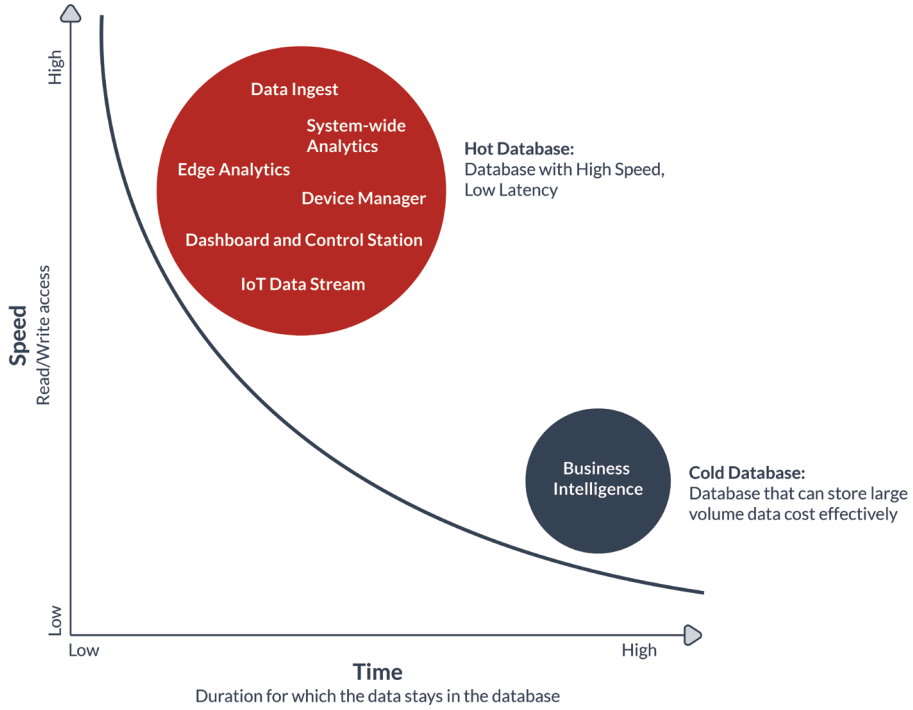

Figure 3. Time-Speed Curve

When you are done with the time-speed graph, the next step is to group the services that have similar data access characteristics. The grouping of services based on their data need helps you limit the number of databases in your environment, reducing the operational overhead. This process also helps you eliminate the databases that won't fit your requirements.

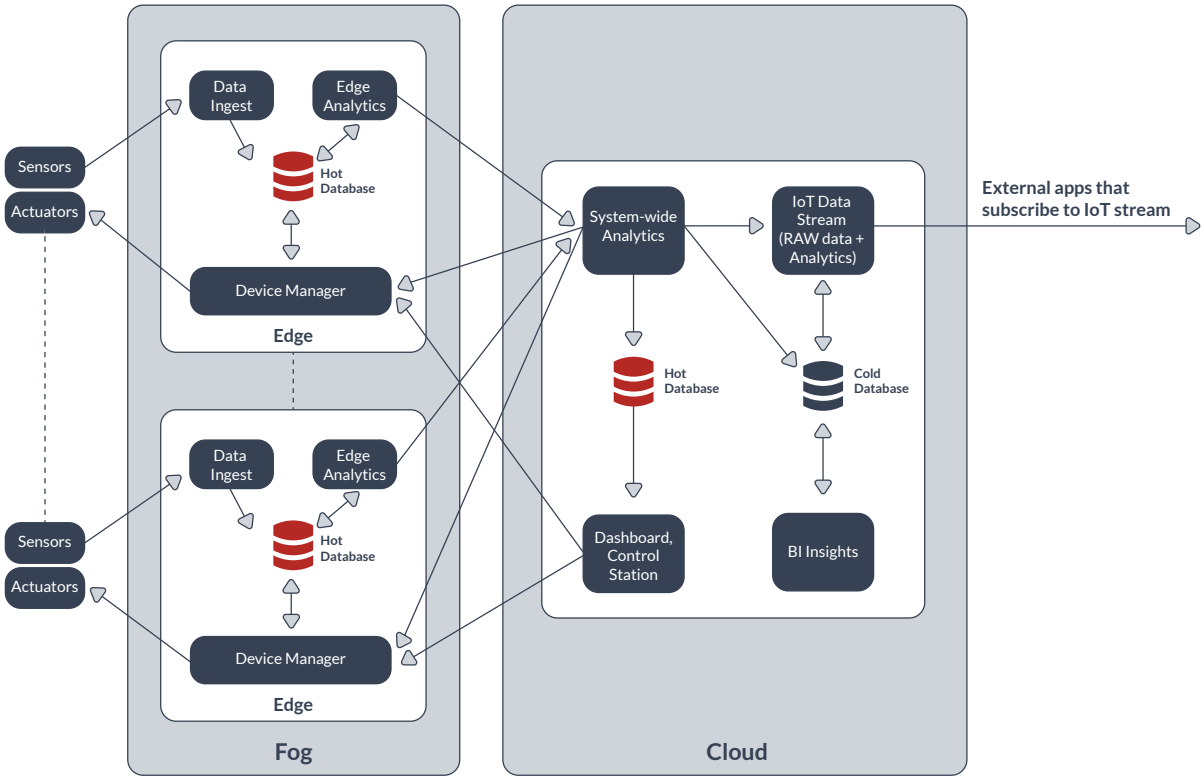

Figure 4. Software services with the databases

In our example solution, we group our services into two main databases: hot databases and cold databases. The final architecture, including the databases, will look like Figure 4. The databases, especially those that hold the hot data, are deployed close to the devices in order to minimize network latency. The database choices for hot and cold data:

Hot database: As the cost of RAM gets more affordable, an in-memory database is often a good choice. In-memory databases deliver data read and write capabilities with the least latency. When choosing a hot database, you should consider the following capabilities to help you narrow down your database options:

1. **Flexibility with data formats:** If you need to support a variety of devices, then you will be forced to support many data and communication formats. Look for a database that can support many different data formats.

2. **Querying capabilities:** Your solution, especially the analytics engine, will query the database to compute and deduce more information. You should verify all the features related to querying, indexing and data organization (for example, how to organize time-series data).

3. **Messaging and queueing:** You need a strong messaging and queueing solution to connect all the services that are spread across geographical regions. A database that supports streams will help you with messaging and queuing along with data transformation, analysis and filtering—all in real-time. With a publish/subscribe model (often called pub/sub), you could decouple producers and consumers, and build a highly flexible distributed IoT solution.

4. **Data spillover and scaling:** For many solutions (IoT solutions in particular) the data grows very fast. When you design the capacity for the hot storage, you should consider the eviction and spillover options. In other words, you should have a plan for what you would do if your RAM was full. Check whether the high-speed databases offer "tiered memory access," wherein the flash memory or a disk can be used as a RAM extender.

5. **High availability and disaster recovery:** Minimizing the downtime is essential for most businesses. The popular in-memory databases offer high availability through clustering, in-memory replication and automatic failure detection, as well as disaster recovery support through persistence and replication across data centers, zones and regions.

6. **Geo Distribution:** If your IoT network is geographically distributed, and if your architecture requires some of your services to stay close to your devices, then you may need a database that supports geo distribution.

7. **Binary safe:** IoT solutions often store binary data (videos from CCTV, for example). Does your solution require storing and accessing binary data? If yes, then you need to choose a database that's binary safe.

Cold database: The historical data for IoT solutions may grow to multiple terabytes, even exceeding a petabyte in some cases. The popular choices to store historical data include storage solutions on commodity hardware. The queries typically follow the map-reduce pattern. Often, the historical data is also indexed in a search engine for pattern matching and data aggregation. If you are storing the data in the cloud, check with your cloud provider to determine which data storage solution is the most cost effective in your region.

Step 4: Evaluate cost and resource efficiency

Classifying the databases into two types—hot and cold—doesn't always mean you should have only two databases, but doing so can help you narrow down your database choices. For most IoT use cases, one high-speed database could satisfy all the requirements for your hot database. If you need a specialized second hot database, then you should make that decision only after a thorough evaluation. In terms of cold storage, the options may range from relational databases to data lakes. A common mistake architects make is creating a polyglot architecture with a specialized database for each service. This not only increases the complexity of the application stack, but also the operational overhead and cost.

The total cost of owning a database is a calculation with many factors. The cost of the database itself is just a small portion of the total cost. Most of the expense associated with owning and operating a database comes from the cost of computing, networking and storage resources, as well as the operational overhead. Another important factor to consider is the cost of data loss, and the possibility that it may occur.

1. Database license cost: The database cost depends on the vendor's pricing model. The cost could be a function of the number of CPUs; the number of shards in the cluster; the database size; throughput (maximum number of operations per second); time period (annual, monthly, hourly, etc.); features for high availability and recovery; availability in multiple regions of the cloud; etc. If you are using a database that's available as open-source software and depending on the type of license you use, the database cost may even be nil.

2. Cost of computational resources: You have limited choices to expand the computational capacity of the edge servers in the fog environment, so you try to fit the database that gives you the best performance for your hardware configuration. You may be required to invest more in RAM to get better performance of your hot database(s).

In terms of cloud computing resources, your cost mainly depends on how efficient your database is in utilizing the resources. For example, a lightweight, thread-safe database may perform a million read/write operations per second with just two commodity servers, whereas some traditional databases may require tens if not hundreds of servers to perform the same number of operations per second.

In addition to database efficiency, your cost of hardware investment is a function of throughput, number of CPUs, RAM, flash storage, network cards, etc. Database architecture for high availability also plays a role. For example, a quorum-based failover architecture would require only one copy of the secondary server, while a non-quorum-based architecture will need two copies of data to avoid split-brain.

3. Data-loss cost: There are a few things you need to consider when calculating the cost of data loss:

a. What loss does your business incur if you lose the data in your database?
b. What's the possibility that the loss may occur?
c. What's the cost of restoring the data?
d. Does your vendor's SLA cover the cost?

Having proper insurance against data loss is extremely important, especially for commercial IoT solutions. The answer

to the first question ("What loss does your business incur if you lose the data in your database?") is completely specific to your enterprise—only you can establish the cost of business loss.

The rest of the questions depend on your database choice and architecture. A proper architecture will ensure that the data loss is less likely to happen. If you are purchasing the database from a vendor, verify your support agreement and SLAs. Should the database belong to an open-source community, evaluate how active the community is, and what kind of community or commercial support you receive for your database.

4. Operational overhead: This is a function of how many person-hours or human power you require to operate your databases. Automation is the mantra for success. A database that offers automation controls for deployment, provisioning, failover, scaling, data partitioning, backup, recovery, monitoring and alerts, will help you operate efficiently.

To evaluate long-term operational overhead, you should evaluate the cost of cloud vendor lock-in. If your database is bound to a cloud-vendor, your attachment to your vendor will grow as your data grows. A cloud-agnostic database will enable you to choose the most cost effective cloud provider without incurring the additional cost of data migration in the future.

Conclusion

When it comes to choosing the right database for your next generation IoT solution, it's quite easy to get lost in the plethora of databases available today. However, if you break your solution into component services and understand their database needs, you can effectively narrow down your database choices. Most IoT solutions can depend on a hot database for real-time data collection, processing, messaging and analytics, and a cold database to store historical data and gather business intelligence. This will make the architecture simple, lean and robust.

As a final note, Redis, the open source, in-memory database (www.redis.com), is a popular choice for a hot database in IoT solutions, widely used for data ingest, real-time analytics, messaging, caching and many other use cases.

700 E El Camino Real, Suite 250
Mountain View, CA 94040
(415) 930-9666
redis.com