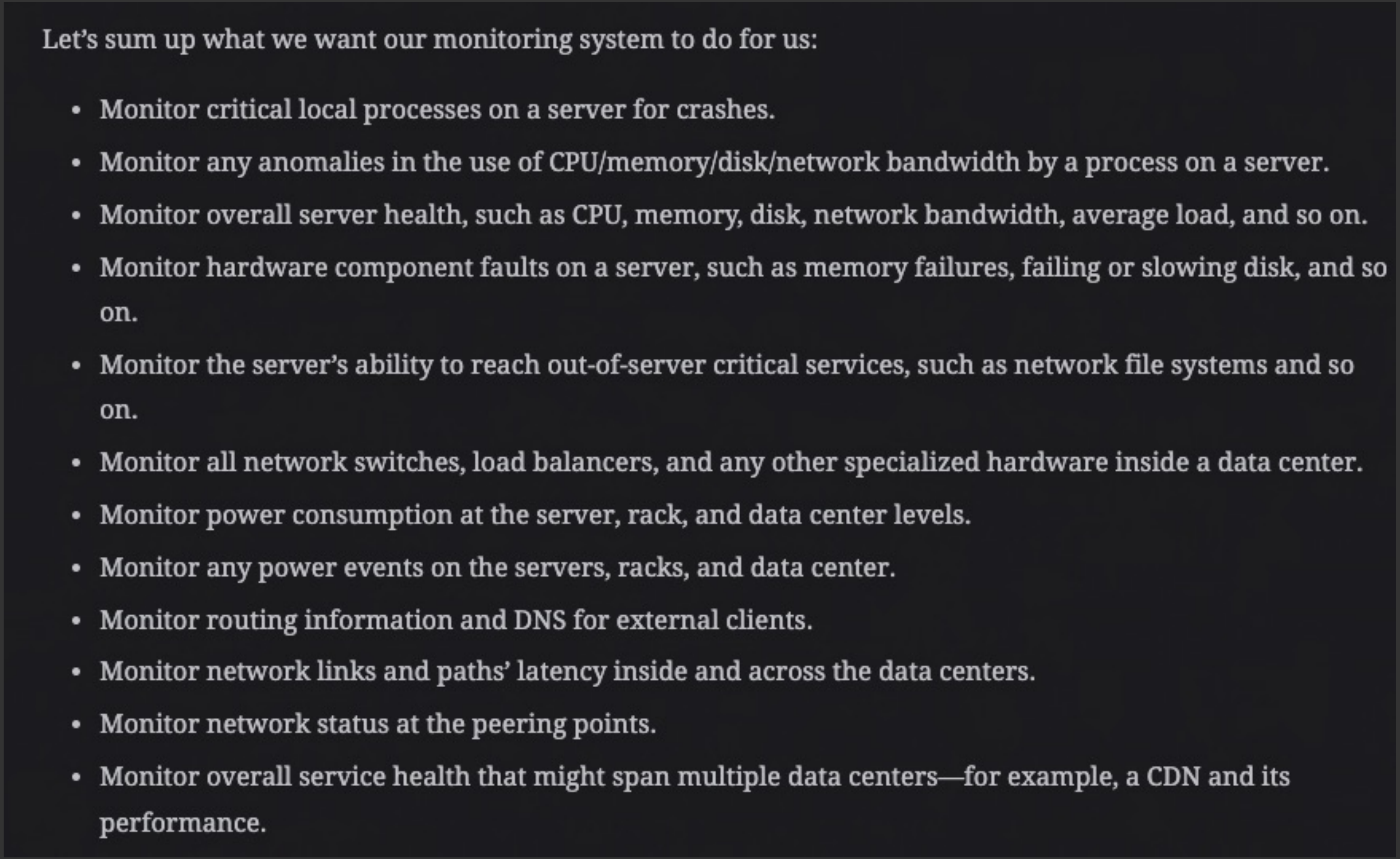
Monitor - Server-side Errors

Requirement

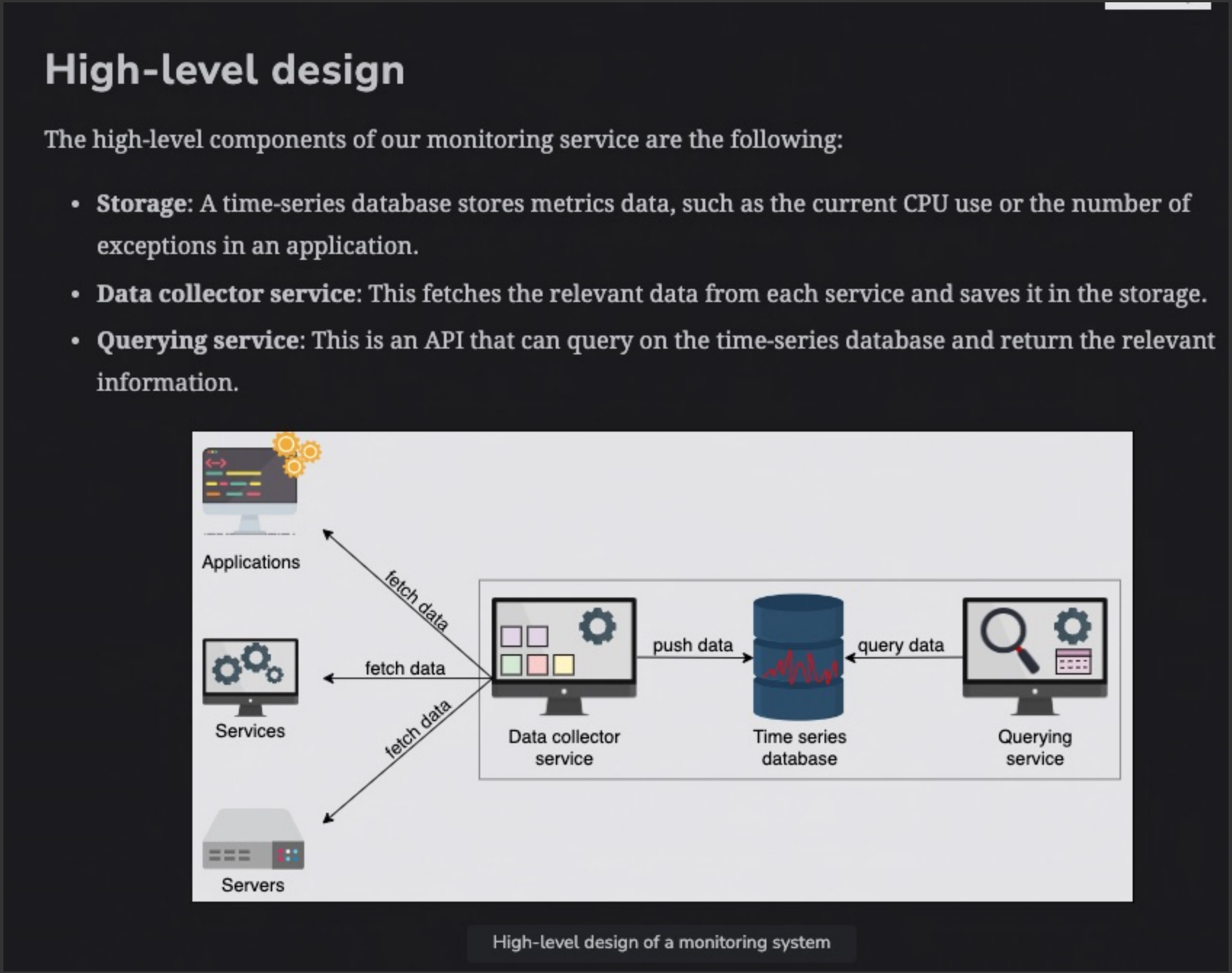


Let's sum up what we want our monitoring system to do for us:

- Monitor critical local processes on a server for crashes.
- Monitor any anomalies in the use of CPU/memory/disk/network bandwidth by a process on a server.
- Monitor overall server health, such as CPU, memory, disk, network bandwidth, average load, and so on.
- Monitor hardware component faults on a server, such as memory failures, failing or slowing disk, and so on.
- Monitor the server's ability to reach out-of-server critical services, such as network file systems and so on.
- Monitor all network switches, load balancers, and any other specialized hardware inside a data center.
- Monitor power consumption at the server, rack, and data center levels.
- Monitor any power events on the servers, racks, and data center.
- Monitor routing information and DNS for external clients.
- Monitor network links and paths' latency inside and across the data centers.
- Monitor network status at the peering points.
- Monitor overall service health that might span multiple data centers—for example, a CDN and its performance.

Building block we will use
The design of distributed monitoring will consist of the following building block:

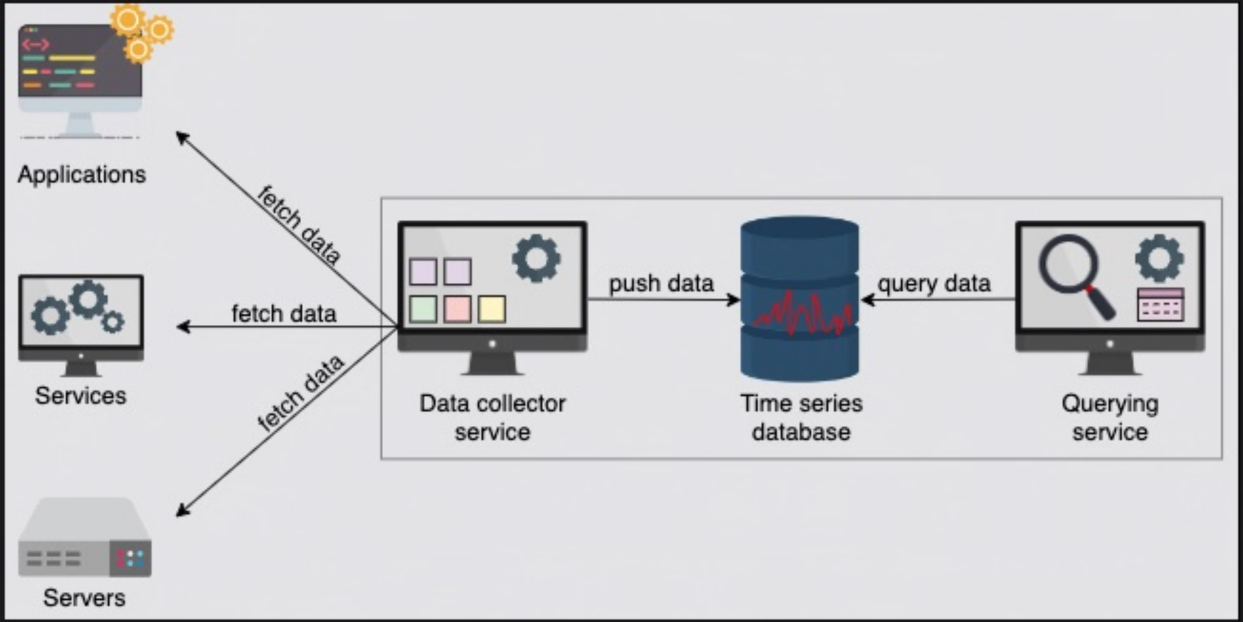
Blob storage: We’ll use blob storage to store our information about metri



High-level design

The high-level components of our monitoring service are the following:

- **Storage**: A time-series database stores metrics data, such as the current CPU use or the number of exceptions in an application.
- **Data collector service**: This fetches the relevant data from each service and saves it in the storage.
- **Querying service**: This is an API that can query on the time-series database and return the relevant information.



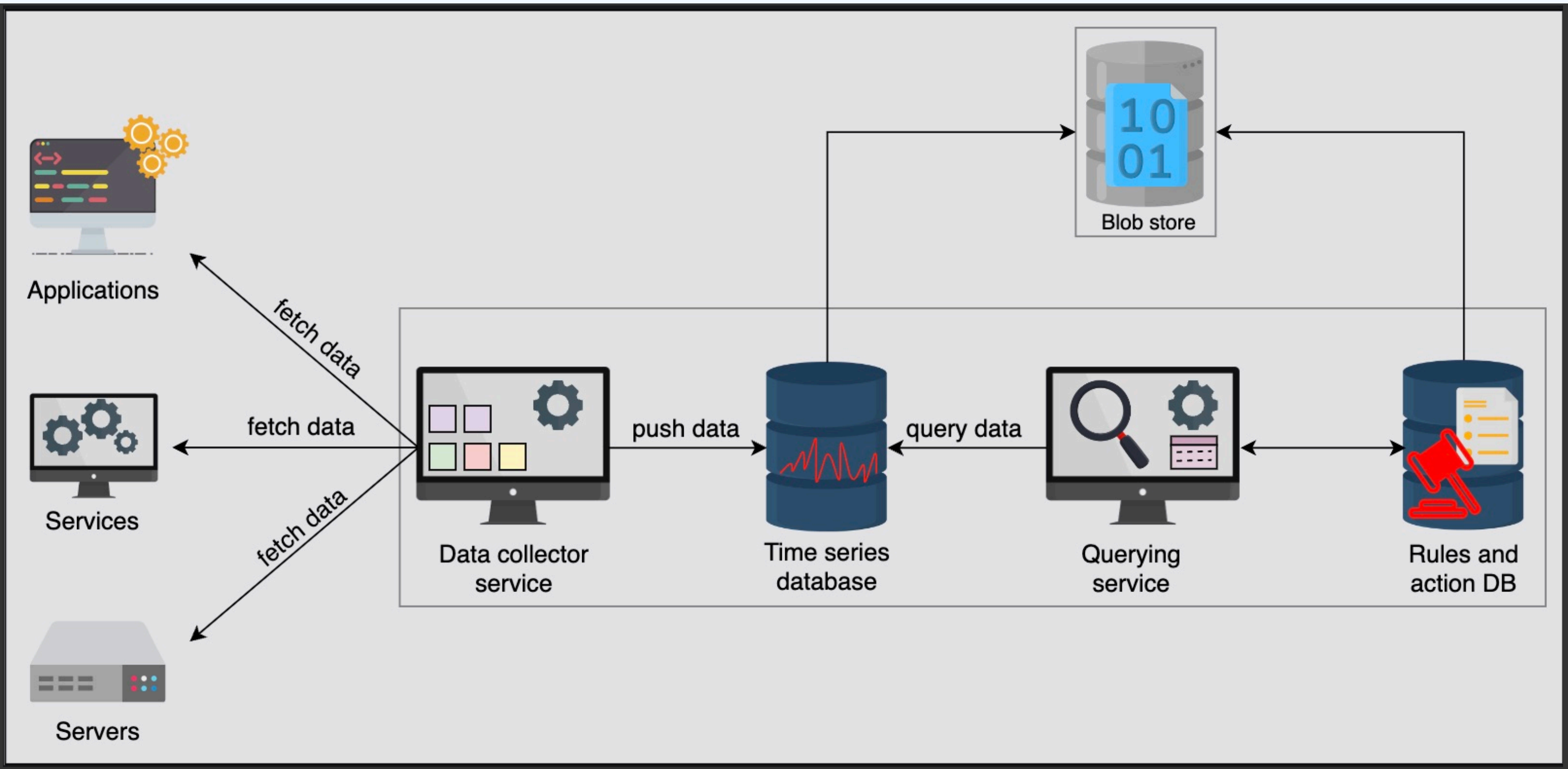
High-level design of a monitoring system

Detailed design of Monitoring System

Storage .

We'll use time-series databases to save the data locally on the server where our monitoring service is running.

We need to store metrics and know which action to perform if a metric has reached a particular value. For example, if CPU usage exceeds 90%, we generate an alert to the end user so the alert receiver can do take the necessary steps, such as allocate more resources to scale.



Data Collector

We need a monitoring system to update us about our several data centers. We can stay updated if the information about our processes reaches us, which is possible through logging. We'll choose a pull strategy. Then, we'll extract our relevant metrics from the logs of the application. As discussed in our logging design, we used a distributed messaging queue.

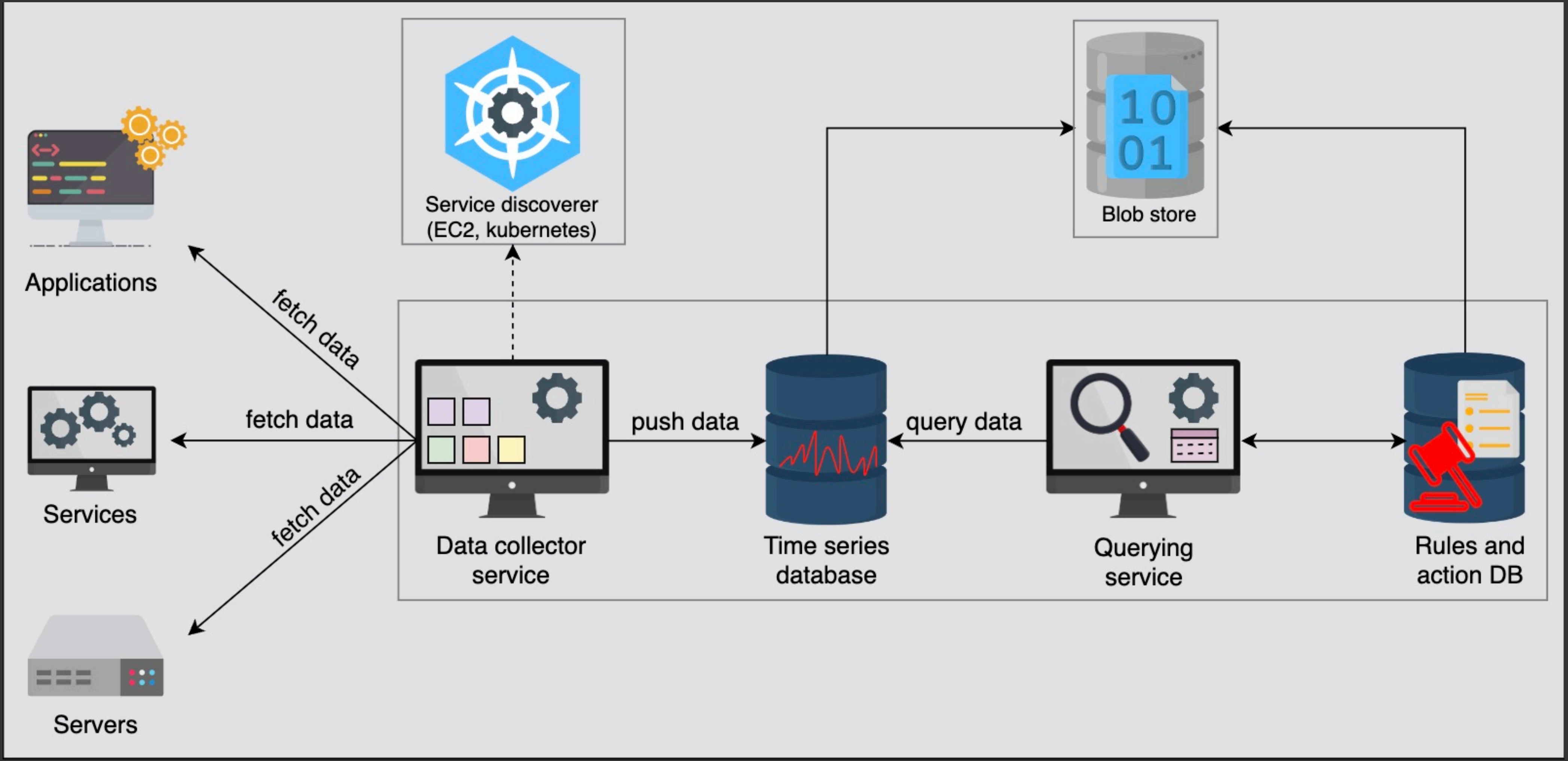
What are some drawbacks of using a push-based approach?

The push-based monitoring tool collects metric data from the applications and servers and sends it to a central collection platform. Each microservice sends its metrics to the monitoring system, resulting in a heavy traffic load on the infrastructure. This is how monitoring might become a bottleneck for business operations. The monitoring can be near real-time. However, if appropriate care isn't taken, it can overwhelm the infrastructure with continual push requests from all services, resulting in network floods. We must also install daemons on each of these targets to send metrics to the monitoring server, which requires additional work.

→ In real world example monitoring system based on pull-based approach is digital Ocean.

Service discovery

We'll use a service discovery solution and integrate with several platforms and tools, including EC2, Kubernetes, and Consul. This will allow us to discover which services we have to monitor. Similar dynamic discovery can be used for newly commissioned hardware.



Querying service

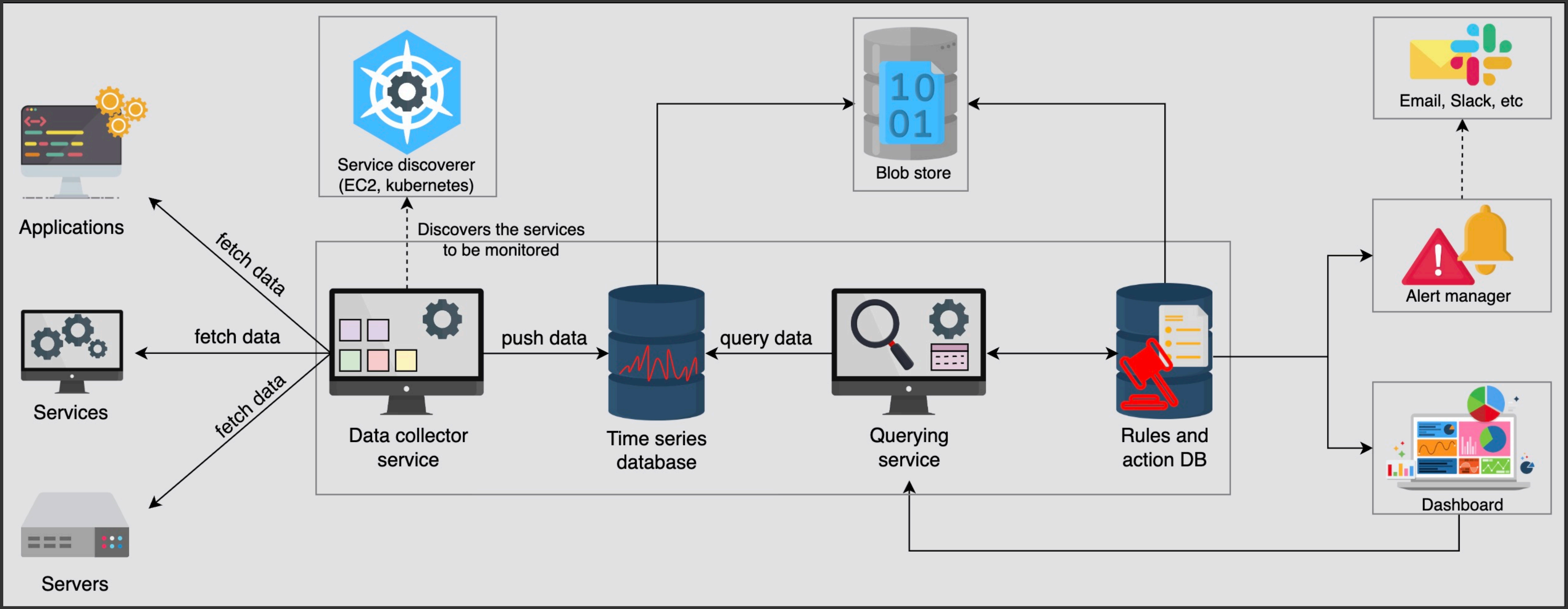
We want a service to access the database and fetch the relevant query results. We need this because we want to view the errors like values of a particular node's memory usage, or send an alert if a metric exceeds the set limit. Let's add the two components we need along with querying.

Alert Manager

The alert manager is responsible for sending alerts upon violations of set rules. It can send alerts as an email, a Slack message, and so on.

Dashboard

We can set dashboards by using the collected metrics to display the required information—for example, the number of requests in the current week.



Pros

The design of our monitoring service ensures the smooth working of the operations and keeps an eye on signs of impending problems.

Our design avoids overloading the network traffic by fetching the data itself.

The monitoring service provides higher availability.

Cons

The system seems scalable, but managing more servers to monitor can be a problem. For example, we have a dedicated server responsible for running the monitoring service. It can be a single point of failure (SPOF). To cater to SPOF, we can have a failover server for our monitoring system. Then, we also need to maintain consistency between actual and failover servers. However, such a design will also hit a scalability ceiling as the number of servers further increase.

Monitoring collects an enormous amount of data 24/7, and keeping it forever might not be feasible. We need a policy and mechanisms to delete unwanted data periodically to efficiently utilize the resources.

What happens if a local or global monitoring system is down?

Hide Answer
We can store the data locally and wait for the system to be up and running again. But there’s a limit for the local data storage. So, either we delete previous data or we don’t store new data. To make a decision, relevant policies need to be created.

How can a monitoring system reliably work if it uses the same infrastructure in a data center that it was supposed to monitor? Consider this given that a failure of a network in a data center can knock out the monitoring components.

Hide Answer
The actual deployment of a monitoring system needs special care. We might have an internal, monitoring-specific network to isolate it from the common network. We should use a separate instance of blob stores and other services.

It also helps to have external components to the monitoring, where external might mean an independent service provider’s infrastructure. However, designing such a system is complex and is more expensive.

How to Visualize Data

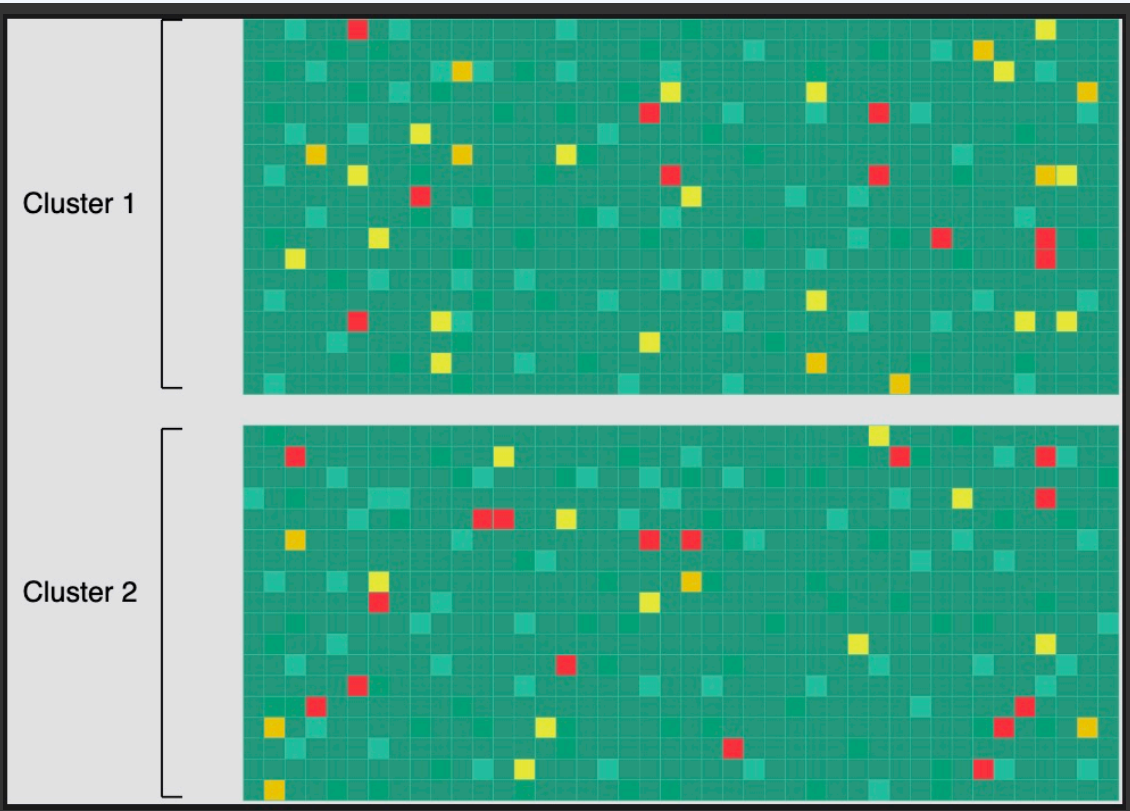
Large data centers have millions of servers, and visualizing the health data for all of them is challenging. An important aspect of monitoring a fleet of servers is to know which ones are alive and which ones are offline. A modern data center can house many thousands of servers in a building. We can use a heat map to display information about thousands of servers compactly in a data center.

A heat map is a data visualization technique that shows the magnitude of a phenomenon in two dimensions by using colors.

Summary
Monitoring systems are critical in distributed systems because they help in analyzing the system and alerting the stakeholders if a problem occurs.

We can make a monitoring system scalable using a hybrid of the push and pull methods.

Heat maps are a powerful tool for visualization and help us learn about the health of thousands of servers in a compact space.


Cluster 1
Cluster 2