Back To Course Home

Grokking Modern System Design Interview for Engineers & Managers
0% completed

Key-value Store

Content Delivery Network (CDN)

Sequencer

Distributed Monitoring

Monitor Server-side Errors

Design of a Monitoring System

Detailed Design of a Monitoring System

Visualize Data in a Monitoring System

Monitor Client-side Errors

Distributed Cache

Distributed Messaging Queue

Pub-sub

Rate Limiter

Blob Store

Distributed Search

Distributed Logging

Distributed Task Scheduler

Sharded Counters

Concluding the Building Blocks Discussion

Design YouTube

Design Quora

Design Google Maps

Design a Proximity Service / Yelp

Design Uber

Design Twitter

Design Newsfeed System

Design Instagram

Design a URL Shortening Service / TinyURL

Design a Web Crawler

Design WhatsApp

Design Typeahead Suggestion

Design a Collaborative Document Editing Service / Google Docs

Spectacular Failures

Concluding Remarks

Course Certificate

Mark Course as Completed

Detailed Design of a Monitoring System

Learn the details of designing a monitoring system and understand its pros and cons.

> **We'll cover the following**

We'll discuss the core components of our monitoring system, identify the shortcomings of our design, and improve the design to fulfill our requirements.

Storage#

We'll use time-series databases to save the data locally on the server where our monitoring service is running. Then, we'll integrate it with a separate storage node. We'll use blob storage to store our metrics.

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. For this purpose, we need another storage area that will contain the rules and actions. Let's call it a rules databse. Upon any violation of the rules, we can take appropriate action.
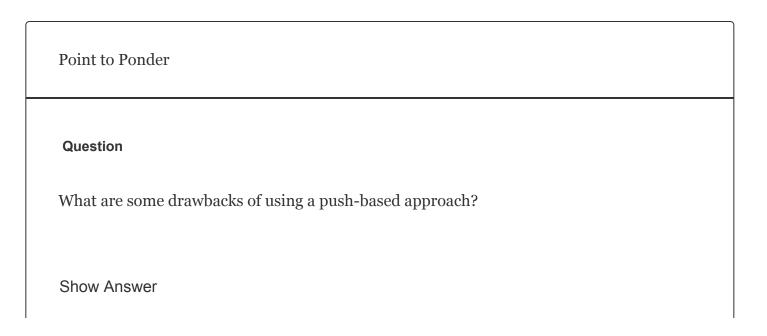
Here, we have identified two more components in our design—that is, a rules and action database and a storage node (a blob store).

Adding blob storage and a rules and action database

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'llextract our relevant metrics from the logs of the application. As discussed in our logging design, we used a distributed messaging queue. The message in the queue has the service name, ID, and a short description of the log. This will help us identify the metric and its information for a specific service. Exposing the relevant metrics to the data collector is necessary for monitoring any service so that our data collector can get the metrics from the service and store them into the time-series database.

A real-world example of a monitoring system based on the pull-based approach is DigitalOcean. It monitors millions of machines that are dispersed globally.

Point to Ponder

Question

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

Show Answer

Service discoverer#

The data collector is responsible for fetching metrics from the services it monitors. This way, the monitoring system doesn't need to keep a track of services. Instead, it can find them using discoverer service. We'll save the relative information of the services we have to monitor. 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.

Let's add our newly identified component to our existing design.

Adding the service discoverer

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.

Let's add the components discussed above, which completes our design of the monitoring system.

Detailed design of monitoring system

Our all-in-one monitoring service works for actively tracking systems and services. It collects and stores data, and it supports searches, graphs, and alerts.

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.

Let's think of a way to overcome the problems with our monitoring service.

Improving our design#

We want to improve our design so that our system can scale better and decide what data to keep and what to delete. Let's see how the push-based approach works. In a push-based approach, the application pushes its data to the monitoring system.

Push-based monitoring system

We used a pull-based strategy to avoid network congestion. This also allows the applications to be free of the aspect that they have to send the relevant monitoring data of to the system. Instead, the monitoring system fetches or pulls the data itself. To cater to scaling needs, we need to apply a push-based approach too. We'll use a hybrid approach by combining our pull-based strategy with the push-based strategy.

We'll keep using a pull-based strategy for several servers within a data center. We'll also assign several monitoring servers for hundreds or thousands of servers within a data center—let's say one server monitoring 5,000 servers. We'll call them secondary monitoring servers.

Now, we'll apply the push-based strategy. The secondary monitoring systems will push their data to a primary data center server. The primary data center server will push its data to a global monitoring service responsible for checking all the data centers spread globally.

We'll use blob storage to store our excessive data, apply elastic search, and view our relevant stats using a visualizer. As our servers or data centers increase, we'll add more monitoring systems. The design for this is given below.

> **Note**: Using a hierarchy of systems for scaling is a common design pattern in system design. By increasing nodes on a level or introducing additional levels in the hierarchy, we get the ability to scale according to our current needs.

Monitoring systems pull the data from various servers and then push the data to the data center monitoring system

1 of 2

Points to Ponder

Question 1

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

Show Answer

1 of 2

Humans need to consume enormous amounts of data, and even after different kinds of data summarizations, the data can still be huge. Next, we'll tackle how to present enormous data to human administrators.

Back

Design of a Monitoring System

Next

Visualize Data in a Monitoring System

Mark as Completed

Report an Issue