Back To Module Home

Basic Building Blocks for Modern System Design

0% completed

Introduction to Building Blocks

Domain Name System

Load balancers

Cache

Databases

Key-value Store

Content Delivery Network (CDN)

Sequencer

Distributed Monitoring

System Design: Distributed Monitoring

Introduction to Distributed Monitoring

Prerequisites of a Monitoring System

Distributed Cache

- [ ]

Distributed Messaging Queue

- [ ]

Pub-sub

- [ ]

Rate Limiter

- [ ]

Blob Store

- [ ]

Distributed Search

- [ ]

Distributed Task Scheduler

- [ ]

Sharded Counters

- [ ]

Conclusion

- [ ]

Mark Module as Completed

Prerequisites of a Monitoring System

Learn about the metrics and alerting in a monitoring system.

We'll cover the following

Monitoring: metrics and alerting#

A good monitoring system needs to clearly define what to measure and in what units (metrics). The monitoring system also needs to define threshold values of all metrics and the ability to inform appropriate stakeholders (alerts) when values are out of acceptable ranges. Knowing the state of our infrastructure and systems ensures service stability. The support team can respond to issues more quickly and confidently if they have access to information on the health and performance of the deployments. Monitoring systems that collect measurements, show data, and send warnings when something appears wrong are helpful for the support team.

To further understand metrics, alerts, and their connection with monitoring, we'll go over their significance, their potential benefits, and the data we might want to keep track of.

Point to Ponder

Question

What are the conventional approaches to handle failures in IT infrastructure?

Show Answer

Metrics#

Metrics objectively define what we should measure and what units will be appropriate. Metric values provide an insight into the system at any point in time For example, a web server's ability to handle a certain amount of traffic per second or its ability to join a pool of web servers are examples of high-level data correlated with a component's specific purpose or activity. Another example can be measuring network performance in terms of throughput (megabits per second) and latency (round-trip time). We need to collect values of metrics with minimal performance penalty. We may use user-perceived latency or the amount of computational resources to measure this penalty.

Values that track how many physical resources our operating system uses can be a good starting point. If we have a monitoring system in place, we don't have to do much additional work to get data regarding processor load, CPU statistics like cache hits and misses, RAM usage by OS and processes, page faults, disc space, disc read and write latencies, swap space usage, and so on. Metrics provided by many web servers, database servers, and other software help us determine whether everything is running smoothly or not.

> **Note:** Connect to the following terminal to see details about the CPU utilization of processes that are currently active on the virtual machine.

Terminal 1

Terminal

Click to Connect...

We use the `top` command to view Linux processes. Running this command opens an interactive view of the running system containing a summary of the system and a list of processes or threads. The default view has the following:

- On the top, we can see how long the machine has been turned on, how many users

are logged in, and the average load on the machine for the past few minutes.

- On the next line, we can see the state (`running` , `sleeping` , or `stopped`) of tasks running on the machine.
- Next, we have the CPU consumption values.
- Lastly, we have an overview of physical memory how much of it is free, used, buffered, or available.

Now, let's take the following steps to see a change in the CPU usage:

1. Quit by entering `q` in the terminal.
2. Run `nohup ./script.sh &>/dev/null &`. This script has an infinite loop running in it, and running the command will execute the script in the background.
3. Run the `top` command to observe an increase in CPU usage.

Populate the metrics#

The metrics should be logically centralized for global monitoring and alerting purposes. Fetching metrics is crucial to the monitoring system. Metrics can either be pushed or pulled into a monitoring system, depending on the preference of the user.

Here, we confront a fundamental design challenge now: Do we utilize push or pull? Should the server proactively send the metrics' values out, or should it only expose an endpoint and wait reactively for an inquiry?

In pull strategy, each monitored server merely needs to store the metrics in memory and send them to an exposed endpoint. The exposed endpoint allows the monitoring application to fetch the metrics itself. Servers sending too much data or sending data too frequently can't overload the monitoring system. The monitoring system will pull data as per its own schedule.

In other situations, though, pushing may be beneficial, such as when a firewall prevents the monitoring system from accessing the server directly. The monitoring system has the

ability to adjust a global configuration about the data to be collected and the interval at which servers and switches should push the data.

Push and pull terminologies might be confusing. Whenever we discuss push or pull strategy, we'll consider it from the monitoring system's perspective. That is, either the system will pull the metrics values from the applications, or the metrics will be pushed to the monitoring system. To avoid confusion, we'll stick to the monitoring system's viewpoint.

Point to Ponder

Question

Logging is the act of keeping records of events in a software system. How does it help in monitoring?

Show Answer

> **Note:** At times, we use the word "metrics" when we should have used "metrics' values." However, we can figure out which of them is being referred to through the context they're being used in.

Persist the data#

Figuring out how to store the metrics from the servers that are being monitored is important. A centralized in-memory metrics repository may be all that's needed. However, for a large data center with millions of things to monitor, there will be an enormous amount of data to store, and a time-series database can help in this regard.

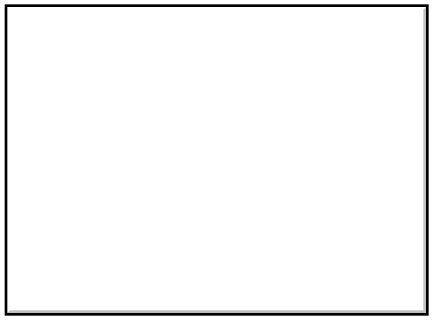
Time-series databases help maintain durability, which is an important factor. Without a historical view of events in a monitoring system, it isn't very useful. Samples having a value of time stamp are stored in chronological sequence. So, a whole metric's timeline can be shown in the form of a time series.

Application metrics#

We may need to add code or APIs to expose metrics we care about for other components, notably our own applications. We embed logging or monitoring code in our applications, called **code instrumentation**, to collect information of interest.

Looking at metrics as a whole can shed light on how our systems are performing and how healthy they are. Monitoring systems employ these inputs to generate a comprehensive view of our environment, automate responses to changes like commissioning more E2C instances if the applications' traffic increases, and warn humans when necessary. Metrics are system measurements that allow analyzing historical trends, correlations, and changes in the performance, consumption, or error rates.

Alerting#



Alerting is the part of a monitoring system that responds to changes in metric values and takes action. There are two components to an alert definition: a metrics-based condition or threshold, and an action to take when the values fall outside of the permitted range.

Introduction to Distributed Monitoring

Next

System Design: The Distributed Cache

Mark as Completed

Report an Issue