Design Considerations for Implementing a Full-Stack Monitoring Solution

The following are some of the key design considerations while implementing full-stack monitoring:

Distributed Monitoring

Consider choosing the topology that is best-suited to the monitoring needs of the applications. For example, small teams with fewer applications to monitor can choose to implement monitoring tools in a single-server standalone topology, while large IT operations teams with many different monitoring applications can choose a distributed architecture in which the master nodes contain metadata of the monitoring tool, while monitoring and slave nodes are centralized and act as the data collectors and data nodes, then connected together as a centralized monitoring system. Monitoring tools clusters should contain a minimum recommended master nodes and data nodes for high availability. If a network outage occurs, monitoring systems may not be able to detect the underlying problems; however, with distributed slaves, if one slave is offline, a new slave node can be added to the cluster, which performs checks and writes the results back to the data nodes. Choose on-premises or SAAS-based monitoring tools depending on your workloads (cloud/on-premises). Pricing also varies and should be considered.

Availability

Site reliability engineering (SRE) teams focus on the reliability, uptime and availability of the applications and thus monitor continuously to identify any potential risks/outages. This mandates that the monitoring tools to be up and running all the time. Most monitoring systems are set up in the same network they monitor, and if a network outage occurs, the monitoring system can’t monitor for security threats. Thus, monitoring tools must be configured inside a high-availability cluster, with availability zones and geo-redundancy regions for availability, backup and disaster recovery.

Scalability

It is common to have multiple monitoring systems to monitor multiple locations or data centers with thousands of hosts. Data from each host is collected multiple times daily, which can result in huge load and performance hits. In a distributed monitoring system with slaves distributed, data checks can be performed locally in the data center, the data can be processed and slaves can then send the results to the master. Based on the load, the hardware CPU cores can be utilized to run multiple checks in different threads or slaves can be added to the cluster for scalability and redundancy.

Storage

Monitoring systems collect and use data from several workloads for various monitoring needs. This results in continuous monitoring, which generates volumes of data accumulation and can result in slow performance of the monitoring tools to analyze the data, demanding high storage needs to store the data and impacting faster troubleshooting. Monitoring systems use databases that are document-based or time series-based. Time series-based databases can store data with time as points, and modern monitoring systems offer plugins to collect data from various sources and provide a built-in schema of the metrics to be stored to the database. Consider customizing the plugins depending on the metrics needed to be stored, as well as aggregation of data based on the granularity and the necessary data storage timeframe. Data can be aggregated and stored in weekly or monthly tables in the database to minimize the overhead of maintenance cost of the monitoring tools. Configure retention and backup policies to retain the aggregate data for machine learning and anomaly pattern detections.

Performance

Performance of the monitoring tools should be tested, given the size and capacity of the master and slave monitoring nodes and the number of hosts to be monitored. Performing various checks in the hosts and then returning data to the server nodes involves lot of processing with high compute needs. Use queues or a transport mechanism to manage the data transfer from the clients to monitoring systems to avoid data loss due to network outage or latency. Most monitoring systems offer a built-in or support a queuing mechanism that performs retries to read/write operations in case of data loss. The slave agent nodes can improve the performance of the monitoring systems by performing data enrichment and data transformation locally, then forwarding the results to the master agents for better and faster reporting. Decide the time intervals you need to monitor the data based on the SLOs defined to optimize the performance of monitoring tool. Monitoring systems can use agent-based or agentless monitoring approaches: In an agent-based system, the data is collected using binaries installed in each host to be monitored, while an agentless monitoring system uses a pull-based approach to collect data from individual hosts. When using an agent-based approach, consider the memory footprint of the agent installation: Use optimized queries for search capabilities/indexes on the primary fields and aggregations in the monitoring system database to improve performance of the data retrieval, graphing and reporting.

Security

Configure security with permissions in the monitoring systems. The slave nodes/master systems should communicate securely with each other and access the endpoints and various ports of different processes for performing checks. Configure security at all levels of applications, different components, middleware software and firewalls to allow access to the ports and processes the monitoring systems need to access and monitor.

Limit the security access to the defined stakeholders as per the need and administrator permissions to configure the monitoring tools for policies, thresholds, integration with other tools, managing users, backup, retention and other factors.

Automation

Because monitoring systems use either agent-based or agentless monitoring modes to collect data, agent binaries or plugins must be installed in every host to be monitored, which can be a tedious and time-consuming task. While a few of the monitoring tools can perform auto-discovery of the hosts and the agent's binaries, custom configurations can be pushed to the hosts through configuration management tools and configured automatically to perform the checks and forward the instrumentation to the centralized monitoring system. The infrastructure scripts for the monitoring systems need to be version-controlled, and scheduled jobs and maintenance activities—such as cleanup of old data and running enabling batch scripts on the monitoring data to trigger remediate actions—can be deployed using configuration management and CI tools, reducing much of manual effort and maintenance costs. Configuring the monitoring of provisioning, de-provisioning and use of resources can be done through automation scripts in the monitoring tools.

Testing the Monitoring tools

Test your monitoring tools as you would any other application. Create test cases to test the configurations used for various checks; test whether right data has been reported from the right host with appropriate labels and tags; check the endpoints and the policies configured; test the latency/data transfer between the agents and master nodes; test whether the data correlation provided, if any, are giving the accurate information with time-bound; test to ensure the dashboards, alerts and visualization graphs are reporting proper data; as well as any machine learning and remediation scripts functional testing. Configure test environments to install monitoring tools in the staging environments to test the behavior, performance and the accuracy of the monitoring data before moving to production environments.

Choosing Metrics to Set Up

Metrics provides insights into the system behavior and choosing metrics that are not relevant can add more overhead and poor monitoring. Choose the metrics that provides the health of the overall platform such as throughput, latency and error metrics that can give insights of erroneous results from several sources, success metrics and performance metrics. Configure metrics for individual resources that measure utilization, saturation, errors, availability, performance such as response times, disk/CPU/memory usage, packets inflow/outflow, etc. Choose metrics to configure based on the events logs, such as scaling up/down hosts, containers up/stopped, build or deployment failures, profiling and stack traces, which can provide data correlation, and configure the metrics to enable drill-down from the work-level metrics to resource-level metrics to the event-level metrics, with required granularity and tagged to the scope of the resource. For an efficient full-stack monitoring, configure all the types of metrics to enable faster troubleshooting.

Defining SLOs

SRE teams focus more on the uptime, reliability and availability of an application and many factors can impact the reliability. Most commonly, application design can impact the performance of the application. The service-level objectives (SLOs) have to be designed with the desired objective and setting up target for the objective, with a clear definition of how the objective is measured and the conditions under which they are valid. Configure SLOs related to performance and availability that can be measured within the SLA with defined error budgets, and standardize the service-level indicators that can be created as templates for reuse across multiple projects.

Designing Alerts

Monitoring systems should know what to monitor, how to alert and what channels to use to communicate the alerts. Alerts can be defined on various criteria such as monitoring the potential issues/events and alert them before they occur (host, disk, memory, CPU metrics); monitoring the expected system behavior and key user actions for green status and alerting if the system behavior deviates monitoring the critical resources and components of the application and its impacts that affect various other parts of the applications and alert if the resources fail. Alerting also could be based on black-box monitoring, in which the alerts are configured based on externally visible factors and real users' impact, or based on white-box monitoring. Check historic values and remediation actions in the past to determine the threshold values and the priority levels, where high-priority alerts can be sent as pages to on-call engineers and low/medium priority alerts are sent as secondary notifications. The alerts should be designed to provide contextual information such as links to appropriate graphs/logs/attachments that can contain deep information about the events occurred along with correlations. Identify the right team to receive the alerts and the communication channel that best suits the engineers in on-call rotations and responders with escalation plans and mitigation plans. Configure the frequency and the priority level of the alert based on the use cases. Striking balance between low-alerting and overalerting is important and alerting needs to be continuously improved and fine-tuned to avoid false-positive alerts.

Visualization and Dashboards

Metrics are collected and stored as time-series data, wherein data can be best represented as a graph with time on the x-axis and one can notice the value changes over time. Configure graphs and dashboards to provide a holistic view of different components that can be visualized over various time scales to understand long time trends as well as recent changes that might be affecting the systems currently. Configure generalized dashboards to visualize the current health metrics as on-screen displays and focused dashboards to drill down with contextual information that can be used for troubleshooting and specific use cases such as capacity planning.

Incident Remediation

Designing a well-defined incident remediation process involves integrating monitoring system alerts to create an auto-ticket in the IT service management (ITSM) tools and posting which incident/problem management workflow must be triggered. Identify the potential use cases for auto-remediation, in which a reliable source can identify the potential problem and get the same responses without any manual input for decision-making. For example, a workflow can be designed to integrate with knowledge base to check for known issues and from known components or sources to validate the issue and run auto-remediation actions and notifications bound with SLAs. Extending monitoring systems for auto-ticketing, auto-investigation, auto-remediation and auto-notification through multiple communication channels to the on-call engineers will accelerate full-stack monitoring.

Machine Learning Analytics

With volumes of data being generated and stored in monitoring systems, consider machine learning techniques such as predictive machine learning, in which algorithms learn how system behaves normally, provides a resulting model of normal behavior and predicts a range of observation points that lie outside the range that are marked as anomaly in the data, which can be used for capacity planning use cases. Design test cases for testing the machine learning algorithms to avoid any false-positives that incorrectly identify anomalies or result in malicious events going unnoticed as normal behavior.

Understanding the needs of monitoring in your environment and then choosing the right tools, and implementing them with various design considerations in place, with continuous monitoring and optimizing further, can accelerate SRE (IT operations) teams to achieve full-stack monitoring.