**Monitoring** **Microservices** **application**

A Production-Ready Service Is Properly Monitored.

* Its key metrics are identified and monitored at the host, infrastructure, and microservice levels.
* It has appropriate logging that accurately reflects the past states of the microservice.
* Its dashboards are easy to interpret and contain all key metrics.
* Its alerts are actionable and are defined by signal-providing thresholds.
* There is a dedicated on-call rotation responsible for monitoring and responding to any incidents and outages.
* There is a clear, well-defined, and standardized on-call procedure in place for handling incidents and outages.

The properties which tells us about the overall status and health of Microservices are called key metrics. There are two types of metrics:

* **Host and infrastructure metrics**: Host and infrastructure metrics are those that pertain to the status of the infrastructure and the servers on which the microservice is running. The host and infrastructure metrics that should be monitored for each microservice are the CPU utilized by the microservice on each host, the RAM utilized by the microservice on each host, the available threads, the microservice’s open file descriptors (FD), and the number of database connections that the microservice has to any databases it uses. Monitoring these key metrics should be done in such a way that the status of each metric is accompanied by information about the infrastructure and the microservice. This means that monitoring should be granular enough that developers can know the status of the keys metrics for their microservice on any particular host and across all of the hosts that it runs on. Below are the key metrics:

*CPU*

*RAM*

*Threads*

*Database connections*

* **Microservice metrics**: Microservice metrics are metrics that are unique to the individual microservice. we also must monitor the availability of the service, the service-level agreement (SLA) of the service, latency (of both the service as a whole and its API endpoints), success of API endpoints, responses and average response times of API endpoints, the services (clients) from which API requests originate (along with which endpoints they send requests to), errors and exceptions (both handled and unhandled), and the health and status of dependencies. Below are the key metrics:

*Availability*

*Latency*

*Endpoint success*

*Endpoint responses*

*Endpoint response times*

*Clients*

*Errors and exceptions*

*Dependencies*

Once the key metrics for a microservice have been identified, the next step is to capture the metrics emitted by your service. Capture them, and then log them, graph them, and alert on them. We’ll cover each of these steps in the following sections.

Monitoring a production-ready microservice has four components.

* **Logging: Determining precisely what to log is specific to each microservice. The best guidance on determining what needs to be logged is, somewhat unfortunately, necessarily vague: log whatever information is essential to describing the state of the service at a given time. Luckily, we can narrow down which information is necessary by restricting our logging to whatever can be contained in the code of the service. Some microservice-level key metrics and information, like hashed user IDs and request and response details can and should be located in the microservice’s logs. Logging needs to be scalable it needs to be available, and it needs to be easily accessible and searchable.**
* **Dashboards: Every microservice must have at least one dashboard where all key metrics (such as hardware utilization, database connections, availability, latency, responses, and the status of API endpoints) are collected and displayed. A dashboard is a graphical display that is updated in real time to reflect all the most important information about a microservice. Dashboards should be easily accessible, centralized, and standardized across the microservice ecosystem.** **Well-designed dashboards also give developers an easy, visual way to detect anomalies and determine alerting thresholds.**
* **Alerting: The detection of failures, as well as the detection of changes within key metrics that could lead to a failure, is accomplished through alerting. To ensure this, all key metrics—host-level metrics, infrastructure metrics, and microservice-specific metrics—should be alerted on, with alerts set at various thresholds. Effective and actionable alerting is essential to preserving the availability of a microservice and preventing downtime.**

**All alerts should be useful: they should be defined by good, signal-providing thresholds. Three types of thresholds should be set for each key metric, and have both upper and lower bounds: normal, warning, and critical.**

**Once an alert has been triggered, it needs to be handled quickly and effectively. The root cause of the triggered alert should be mitigated and resolved.**

**Current Monitoring of Cradle Application**

**Cradle Monolithic application using Geneos Tool for monitoring. Geneos has several features to provide the current information about the application and infrastructure.**

**Current Monitoring System (on both servers):**

* **Checking host server availability. It configured to hit the ../basal/ URL to check if the server is available or not.**
* **Geneos is configured to check the if application server(JBOSS) is running. it uses process id filter to check this process.**
* **Geneous check if the application is up and running. it uses curl command to hit the URL and based on the response code it decides.**
* **Application monitoring using logs. It check specific word in webapp.log and service.log file to check if the application auto restated. It is configured to look for work [started \*....].**
* **Performance monitoring using gc.log file filtering. Configured to filter the specific word [fullgc] in the gc.log file if the occurrence is more than 15 it alerts.**
* **Database monitoring is configured by hitting some query and received response.**

**Features**:

*Alerting*

*Provide host server status*

*Provide application server status*

*Provide application status using curl*

*log monitoring/filtering*

*Database monitoring*

**Lacking**:

*Graphical interface*

*latency issue*

*CPU and Memory utilization*

**Tools for Microservices based architecture**

**AppDynamics**

**Nagios**