**DataDog**

Metrics are numerical values that can track anything about your environment over time, from latency to error rated to user signups.

Visualizing Metrics

Alerts are simply setting a threshold in a monitor. When that threshold is breached, a notification is sent to the designated recipient.

A trace is used to track the time spent by an application processing a request along with the execution path taken.

**Datadog 101: Site Reliability Engineer**

[Datadog Agent documentation](https://docs.datadoghq.com/agent/)

To verify that everything is running the way it should, run the following command: datadog-agent status

To see all of the resolved configuration values of the running Agent : datadog-agent config

Command to restart the Agent: systemctl restart datadog-agent

Documentation for all of them in the [Integrations documentation](https://docs.datadoghq.com/integrations). [Agent documentation](https://docs.datadoghq.com/agent/faq/how-datadog-agent-determines-the-hostname/?tab=agentv6v7). See the [Writing a Custom Agent Check documentation](https://docs.datadoghq.com/developers/custom_checks/write_agent_check/) for details.

On Linux systems, these files are located in/etc/datadog-agent by default. The file that configures every aspect of the Datadog Agent is datadog.yaml.

docker-compose up -d

DD\_API\_KEY: This is required by the Agent to submit metrics and events to Datadog. It’s set in the host environment, and you can see it by running env |grep DD\_API\_KEY in the terminal.

To start the application stack, run this command: docker-compose up -d

docker-compose exec datadog agent status

docker-compose exec datadog agent config

docker-compose ps

The problem is that the discounts service doesn't write logs in a standard format. It's up to the application developer to emit logs that Datadog knows how to parse. There are ways around this that you'll learn about in the "Logs" and "APM" labs. The [Python integration documentation](https://docs.datadoghq.com/logs/log_collection/python/?tab=jsonlogformatter) also provides a solution.

Two important concepts of APM are *traces* and *spans*:

* A trace tracks the time spent by an application processing a request and the status of this request. Each trace consists of one or more spans.
* A span represents a logical unit of work in a distributed system for a given time period. Multiple spans construct a trace.

By default, the Datadog Agent collects traces via tcp on port 8126. Datadog tracing libraries need to know how to reach the Agent, and the types of information to include with traces. Agent has relied on **Agent container** environment variables and labels on service containers to tag metrics and logs correctly. To send correctly tagged APM traces to the Agent, applications will use **service container** environment variables.

Application tracing is enabled by using Datadog's APM libraries in application code. learn more about ddtrace for Python [in the Datadog Python APM Client documentation](https://ddtrace.readthedocs.io/en/stable/index.html). NPM is built on [eBPF](https://ebpf.io/), which enables detailed visibility into network flows at the Linux kernel level. Consequently, NPM is powerful and efficient with extremely low overhead.

Configuring NPM on a host, Kubernetes, or ECS is similar, and covered is in the [NPM Installation documentation](https://docs.datadoghq.com/network_monitoring/performance/setup/?tab=docker). read the [Network Page documentation](https://docs.datadoghq.com/network_performance_monitoring/network_page/). Retransmits represent detected failed TCP packets that are retransmitted to ensure delivery, measured in the count of retransmits from the source. Latency is the time between a TCP frame being sent and acknowledged. Learn more about the syntax of log search queries, see the [Logs Search Syntax docs](https://docs.datadoghq.com/logs/search_syntax/).

Parsing rules can be very powerful for creating a custom pipeline to parse semi-structured log lines into well-structured, taggable objects, in the same way that Datadog automatically parses JSON log lines. [Pipelines docs](https://docs.datadoghq.com/logs/log_configuration/pipelines/?tab=source) For more information about using Grok and other parsers, see the [Parsing Docs](https://docs.datadoghq.com/logs/log_configuration/parsing/?tab=matchers).

Datadog’s Service Level Objectives (SLOs) track metrics and monitor trends over longer periods of time. Learn more about the special syntax features of the notification text area in the [Notifications docs](https://docs.datadoghq.com/monitors/notify/). Learn more about [Watchdog in the docs](https://docs.datadoghq.com/watchdog/#overview). SLOs can help you prioritize your development time, read the [SLO docs](https://docs.datadoghq.com/monitors/service_level_objectives/).

A dashboard is Datadog’s tool for visually tracking, analyzing, and displaying key performance metrics, which enable you to monitor the health of your infrastructure. Dashboards are very flexible, able to display almost any Datadog visualization and even logs, rich notes, iframes, and more in the form of Widgets. the [Dashboards docs](https://docs.datadoghq.com/dashboards/). To learn more, check out the following:

* [Notebooks docs](https://docs.datadoghq.com/notebooks/)
* [Datadog 201: Becoming a Power User](https://learn.datadoghq.com/courses/dd-201) course
* [Incident Management docs](https://docs.datadoghq.com/monitors/incident_management)
* [Introduction to Incident Management](https://learn.datadoghq.com/courses/intro-to-incident-management) course
* [API Reference docs](https://docs.datadoghq.com/api/latest/)
* [Datadog API: Automation and Infrastructure as Code](https://learn.datadoghq.com/courses/dd-api-automation-iac) course

**Relic**

Monitoring is the act of collecting regular data about systems so performance can be viewed and tracked.

Is our service online and available?

Is our service functioning correctly?

Is our service performing well?

MTTD is the amount of time, on average, between the start of an issue and when teams become aware of it. This does not include time spent troubleshooting or fixing the issue.

MTTR is the average amount of time between when an issue is detected, and when systems are fixed and operating normally again. Ideally this includes both time spent fixing the issue, and implementing proactive steps to prevent it from happening again.

Observability is the practice of instrumenting systems to gather actionable data depicting not only when and where an issue occurred, but—more importantly—why it occurred.

Monitoring is a verb. It’s symptom oriented. It can identify when something is wrong, and where. Observability is a noun. It’s a type of approach that lets you ask why something is wrong. It provides the flexibility to dig into “unknown unknowns” on the fly.

observability requires gathering a lot of performance data, called telemetry data. There are four distinct types of telemetry data, summarized by the acronym “MELT”: Metrics, Events, Logs, and Traces.

Observability platforms ingest data from our services with agents. Agents are integrated into a service through a process called instrumentation.

An agent is typically a small application, typically a file or few lines of code, that automatically collects data and sends it somewhere to be analyzed, such as an observability platform.

Alerts notify stakeholders when telemetry data surpasses a predefined threshold, indicating a need for attention.

The role of a synthetic monitor is to run into an issue before one of your users does. Think of a synthetic monitor like a program, or a bot, that attempts the same action(s) over and over again.