**Monitoring vs. Observability**  
  
**Monitoring:**

Monitoring involves collecting and analyzing predefined sets of metrics, logs, and traces to ensure that systems and applications are functioning as expected. It is generally focused on identifying issues that have already been defined and for which alerts have been configured.  
Purpose: Monitoring is primarily used to track the health of systems, detect known issues, and ensure that performance meets predefined standards. It answers the question, "Is my system working as expected?"  
Tools: Common monitoring tools include Prometheus, Nagios, Datadog, and CloudWatch, which focus on specific metrics like CPU usage, memory, and response times.  
  
**Observability:**

Observability is a broader concept that refers to the ability to understand the internal state of a system based on the data it produces. It goes beyond predefined metrics to provide insights into the underlying causes of issues, even those that are unexpected or unknown.  
Purpose: Observability enables teams to explore and investigate system behavior in real-time, identifying the root cause of issues without relying solely on predefined alerts. It answers the question, "Why is my system behaving this way?"  
Tools: Observability tools include those that offer deep integration of metrics, logs, and traces, such as Grafana, Elastic Stack, New Relic, and Honeycomb.  
  
**Key Concepts: Metrics, Logs, Traces**  
  
**Metrics**  
Metrics are quantitative data points that measure the performance of systems and applications. They are typically numerical values that are aggregated over time, such as CPU utilization, request rates, or error counts.  
Purpose: Metrics are used to monitor trends and patterns in system performance, helping teams to detect anomalies, forecast capacity, and optimize resources.  
Example: Monitoring CPU usage over time to detect potential bottlenecks.  
  
**Logs**  
Logs are time-stamped records of events generated by applications, systems, and infrastructure components. They provide detailed information about the operation of these entities, including errors, warnings, and informational messages.  
Purpose: Logs are essential for debugging and troubleshooting, as they offer a detailed view of what happened in the system at a particular time.  
Example: An application log entry showing a failed login attempt, including the timestamp, user ID, and error code.  
  
**Traces**  
Traces represent the journey of a request or transaction as it flows through various components of a distributed system. They capture the end-to-end latency and performance characteristics of the system.  
Purpose: Traces are used to understand the flow of requests across services, identify bottlenecks, and diagnose issues in complex, distributed systems.  
Example: A trace showing the path of an HTTP request through multiple microservices, highlighting latency at each service.  
  
In summary, while monitoring provides essential insights into system performance through predefined metrics and alerts, observability offers a more comprehensive view, enabling teams to understand and diagnose complex issues by exploring metrics, logs, and traces.