Desenvolvimento de Aplicações com Arquitetura Baseada em Microservices

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1007] - Tópicos Avançados em SI 4
https://bit.ly/vcg-microservices

Licença do material

Resources

- There is no textbook required. However, the following are some books that may be recommended:
  - Building Microservices: Designing Fine-Grained Systems
  - Spring Microservices
  - Spring Boot: Acelere o desenvolvimento de microsserviços
  - Microservices for Java Developers A Hands-on Introduction to Frameworks and Containers
  - Migrating to Cloud-Native Application Architectures
  - Continuous Integration
  - Getting started guides from spring.io

Warm up

- Last lecture we discuss the importante of autoscaling when deploying large-scale microservices
- We also explored the concept of autoscaling and the different models of and approaches to autoscaling
  - the time-based, resource-based, queue length-based, and predictive ones
- We then reviewed the role of a life cycle manager in the context of microservices and reviewed its capabilities

TOP ACTION

Term	Count	Action
orders	243	
results	153	
default	24	
customer	24	
contact	24	

TOP PROTOCOLID

Term	Count	Action
resultsxml	174	
b2bxml	144	
scpxml	48	
crmxml	48	
employee	22	

Logging and Monitoring Microservices

AssertLab
Advanced Software and Systems Engineering Research Technologies

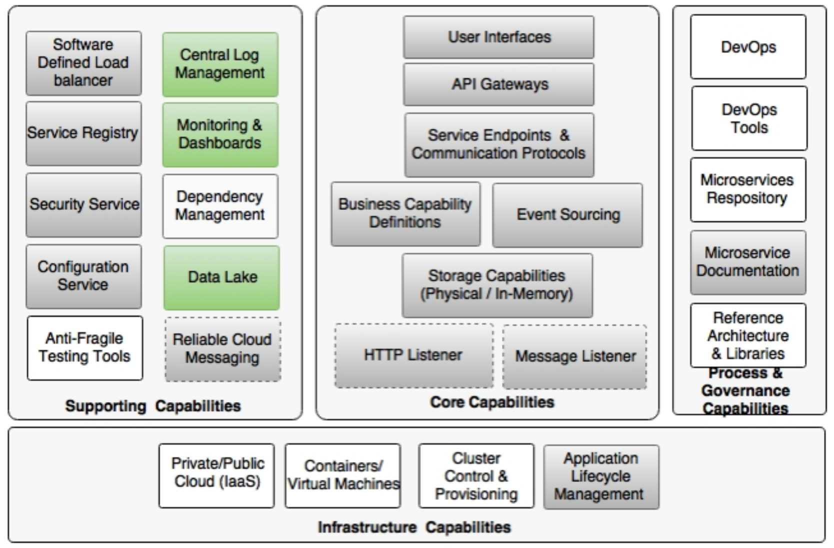
Context

- One of the biggest challenges due to the very distributed nature of Internet-scale microservices deployment is the logging and monitoring of individual microservices
- It is difficult to trace end-to-end transactions by correlating logs emitted by different microservices

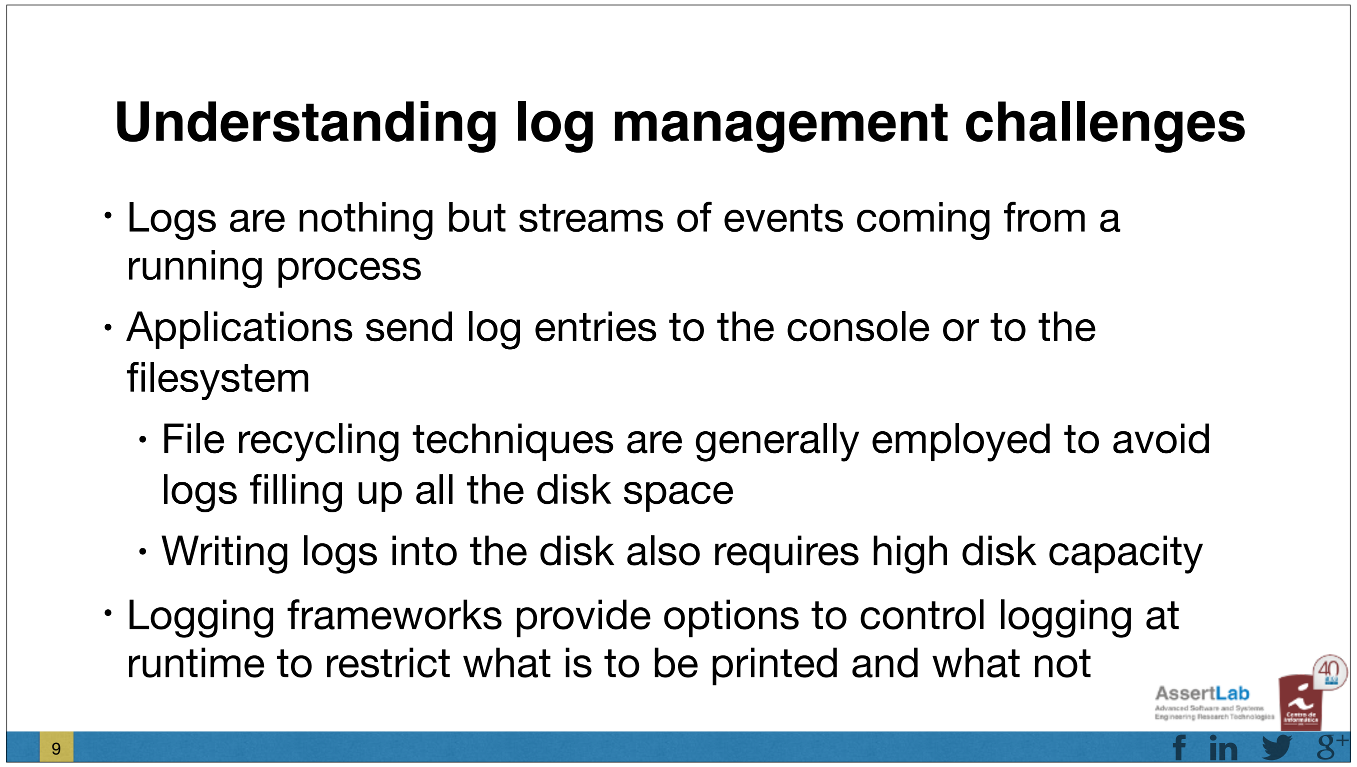
Topics

- The different options, tools, and technologies for log management
- The use of Spring Cloud Sleuth in tracing microservices
- The different tools for end-to-end monitoring of microservices
- The use of Spring Cloud Hystrix and Turbine for circuit monitoring
- The use of data lakes in enabling business data analysis

Reviewing the microservice capability model


Software Defined Load balancer
Central Log Management
Service Registry
Monitoring & Dashboards
Security Service
Dependency Management
Configuration Service
Data Lake
Anti-Fragile Testing Tools
Reliable Cloud Messaging
Supporting Capabilities
User Interfaces
API Gateways
Service Endpoints & Communication Protocols
Business Capability Definitions
Event Sourcing
Storage Capabilities (Physical / In-Memory)
HTTP Listener
Message Listener
Core Capabilities
DevOps
DevOps Tools
Microservices Respository
Microservice Documentation
Reference Architecture & Libraries
Process & Governance Capabilities
Private/Public Cloud (IaaS)
Containers/ Virtual Machines
Cluster Control & Provisioning
Application Lifecycle Management
Infrastructure Capabilities

This lecture will cover the Application Lifecycle Management capability in the microservices capability model discussed in lecture 4, Applying Microservices Concepts



Understanding log management challenges

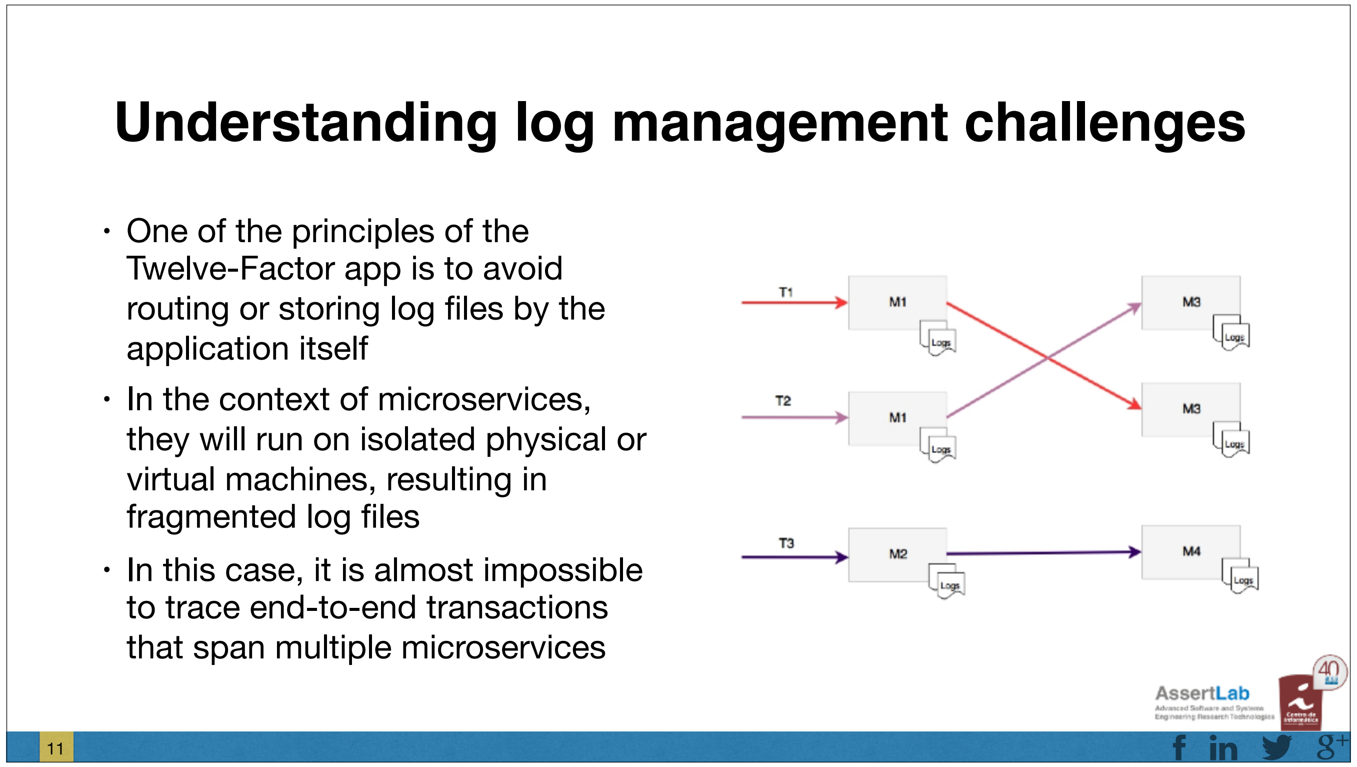
- Logs are nothing but streams of events coming from a running process
- Applications send log entries to the console or to the filesystem
  - File recycling techniques are generally employed to avoid logs filling up all the disk space
  - Writing logs into the disk also requires high disk capacity
- Logging frameworks provide options to control logging at runtime to restrict what is to be printed and what not

For traditional JEE applications, a number of frameworks and libraries are available to log. Java Logging (JUL) is an option off the shelf from Java itself. Log4j, Logback, and SLF4J are some of the other popular logging frameworks available. These frameworks support both UDP as well as TCP protocols for logging.

Understanding log management challenges

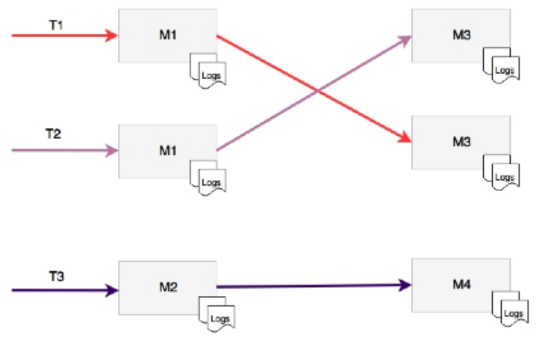
- When moved from traditional to cloud deployment, applications are no longer locked to a particular, predefined machine
- The machines used for deployment can change from time to time
- Logs written to the disk are lost once the container is stopped and restarted

Moreover, containers such as Docker are ephemeral.
This essentially means that one cannot rely on the persistent state of the disk



Understanding log management challenges

- One of the principles of the Twelve-Factor app is to avoid routing or storing log files by the application itself
- In the context of microservices, they will run on isolated physical or virtual machines, resulting in fragmented log files
- In this case, it is almost impossible to trace end-to-end transactions that span multiple microservices



AssertLab

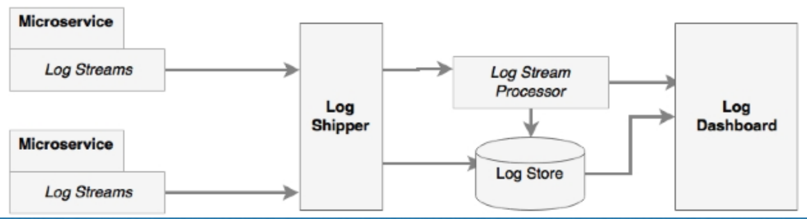
As shown in the diagram, each microservice emits logs to a local filesystem. In this case, microservice M1 calls M3. These services write their logs to their own local filesystems. This makes it harder to correlate and understand the end-to-end transaction flow. Also, as shown in the diagram, there are two instances of M1 and two instances of M2 running on two different machines. In this case, log aggregation at the service level is hard to achieve

A centralized logging solution

- The new logging solution, in addition to addressing the preceding challenges, is also expected to support the following capabilities:
  - The ability to collect all log messages and run analytics on top of the log messages
  - The ability to correlate and track transactions end to end
  - The ability to keep log information for longer time periods for trending and forecasting
  - The ability to eliminate dependency on the local disk system
  - The ability to aggregate log information coming from multiple sources such as network devices, operating system, microservices, and so on

A centralized logging solution

- The solution to these problems is to centrally store and analyze all log messages, irrespective of the source of log
- The fundamental principle employed in the new logging solution is to detach log storage and processing from service execution environments
- In the centralized logging solution, log messages will be shipped from the execution environment to a central big data store
  - Log analysis and processing will be handled using big data solutions



Big data solutions are better suited to storing and processing large numbers of log messages more effectively than storing and processing them in microservice execution environments.
The benefit of this centralized approach is that there is no local I/O or blocking disk writes. It also does not use the local machine's disk space. This architecture is fundamentally similar to the lambda architecture for big data processing.

The selection of logging solutions

- Cloud services
  - Loggly (AWS CloudTrail can be integrated with Loggly for log analysis)
  - Papertrial, Logsene, Sumo Logic, Google Cloud Logging, and Logentries are examples of other cloud-based logging solutions
  - Latency is one of the key factors to be considered when selecting cloud logging as a service

The selection of logging solutions

- Off-the-shelf solutions
  - Graylog uses Elasticsearch for log storage and MongoDB as a metadata store
  - Splunk uses the log file shipping approach, compared to log streaming used by other solutions to collect logs

Best-of-breed integration

- Log shippers
  - Logstash acts as a broker that provides a mechanism to accept streaming data from different sources and sync them to different destinations
  - Fluentd is another tool that is very similar to Logstash, as is Logspout, but the latter is more appropriate in a Docker container-based environment

Best-of-breed integration

- Log stream processors
  - A typical architecture used for stream processing is a combination of Flume and Kafka together with either Storm or Spark Streaming
- Log storage
  - Real-time log messages are typically stored in Elasticsearch, which allows clients to query based on text-based indexes
  - Apart from Elasticsearch, HDFS is also commonly used to store archived log messages
  - MongoDB or Cassandra is used to store summary data, such as monthly aggregated transaction counts
  - Offline log processing can be done using Hadoop's MapReduce programs

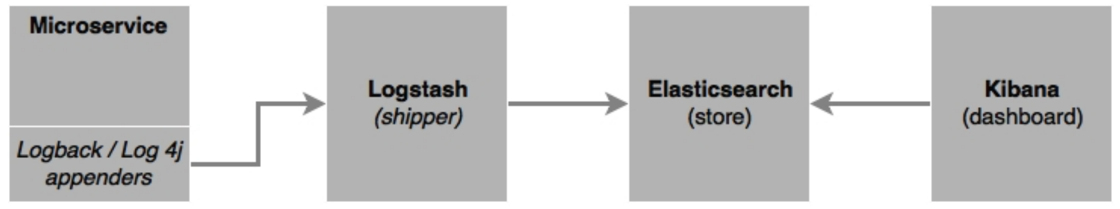
Stream-processing technologies are optionally used to process log streams on the fly. For example, if a 404 error is continuously occurring as a response to a particular service call, it means there is something wrong with the service. Such situations have to be handled as soon as possible. Stream processors are pretty handy in such cases as they are capable of reacting to certain streams of events that a traditional reactive analysis can't

Best-of-breed integration

- Dashboards
  - The most commonly used dashboard for log analysis is Kibana on top of an Elasticsearch data store
  - Graphite and Grafana are also used to display log analysis reports

A custom logging implementation

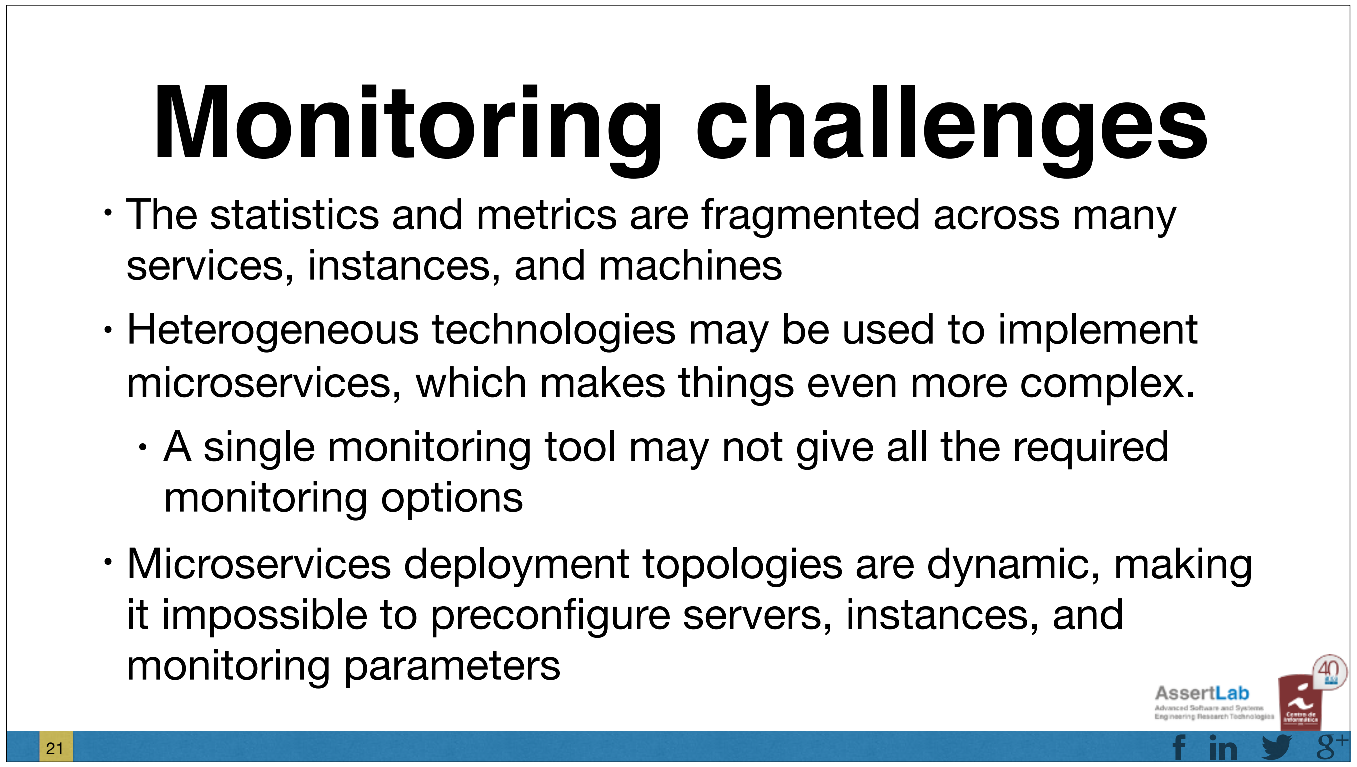
- The most commonly used architecture for custom log management is a combination of Logstash, Elasticsearch, and Kibana, also known as the ELK stack



Monitoring microservices

- Microservices are truly distributed systems with a fluid deployment topology
- To add more complication, these services dynamically change their topologies
- It is important for operations teams to understand the runtime deployment topology and also the behavior of the systems
  - This demands more than a centralized logging can offer

In general application, monitoring is more a collection of metrics, aggregation, and their validation against certain baseline values. If there is a service-level breach, then monitoring tools generate alerts and send them to administrators.
One of the main objectives of microservice monitoring is to understand the behavior of the system from a user experience point of view. This will ensure that the end-to-end behavior is consistent and is in line with what is expected by the users.



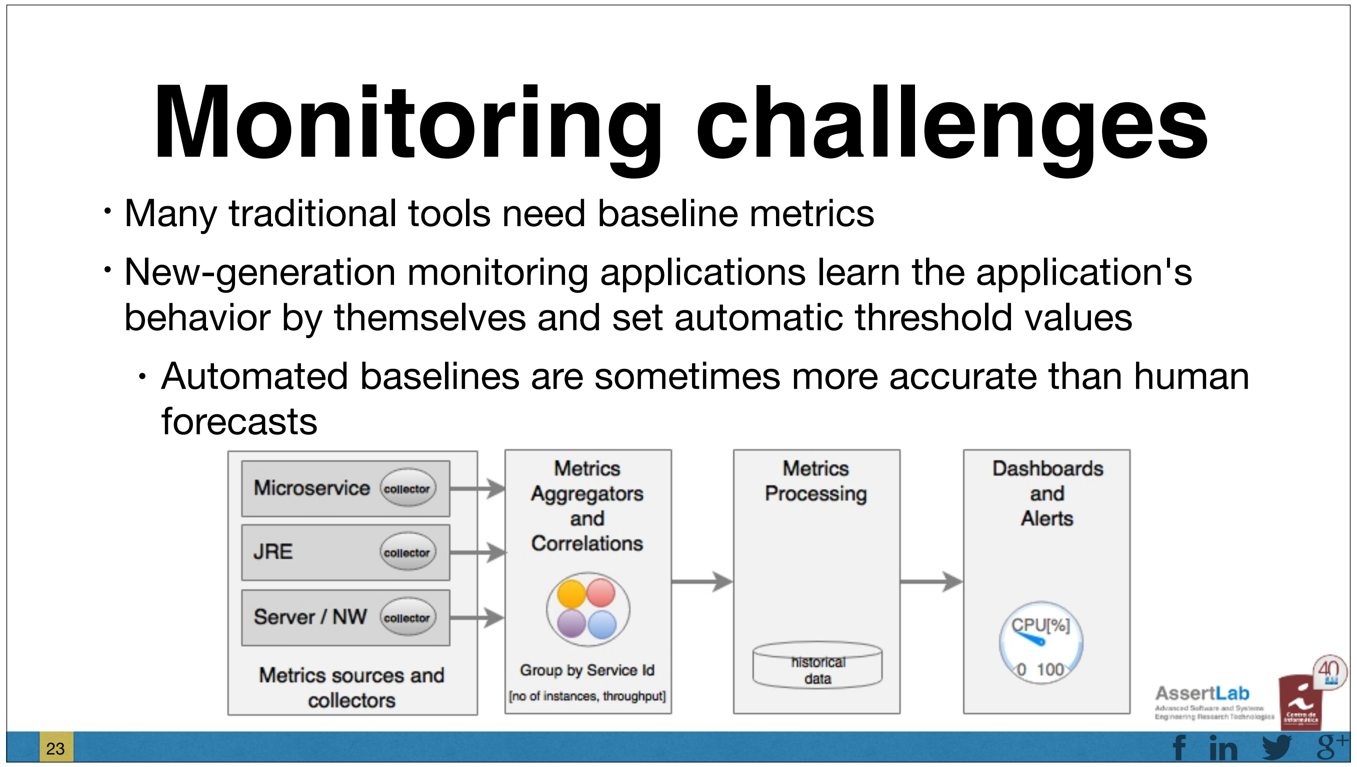
Monitoring challenges

- The statistics and metrics are fragmented across many services, instances, and machines
- Heterogeneous technologies may be used to implement microservices, which makes things even more complex.
  - A single monitoring tool may not give all the required monitoring options
- Microservices deployment topologies are dynamic, making it impossible to preconfigure servers, instances, and monitoring parameters

Similar to the fragmented logging issue, the key challenge in monitoring microservices is that there are many moving parts in a microservice ecosystem.

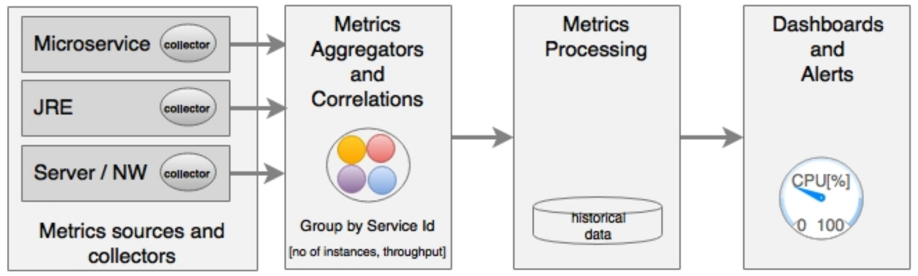
Monitoring challenges

- Many of the traditional monitoring tools are good to monitor monolithic applications
- Many of the traditional monitoring systems are agent-based preinstall agents on the target machines or application instances
- Challenges:
  - If the agents require deep integration with the services or operating systems, then this will be hard to manage in a dynamic environment
  - If these tools impose overheads when monitoring or instrumenting the application, it may lead to performance issues



Monitoring challenges

- Many traditional tools need baseline metrics
- New-generation monitoring applications learn the application's behavior by themselves and set automatic threshold values
  - Automated baselines are sometimes more accurate than human forecasts



Such systems work with preset rules, such as if the CPU utilization goes above 60% and remains at this level for 2 minutes, then an alert should be sent to the administrator. It is extremely hard to preconfigure these values in large, Internet-scale deployments.

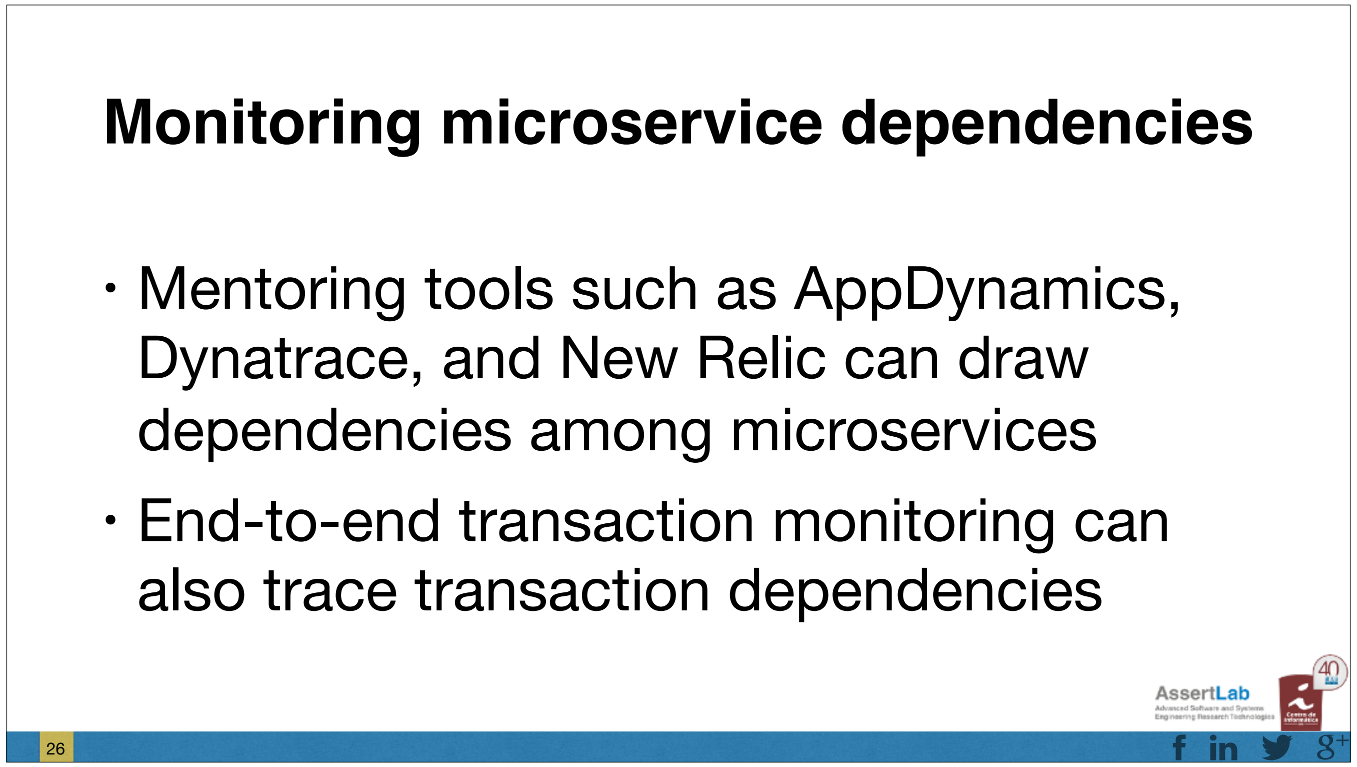
Monitoring tools

- AppDynamics, Dynatrace, and New Relic are top commercial vendors in the Application Performance Management (APM) space
- Cloud vendors come with their own monitoring tools, but in many cases, these monitoring tools alone may not be sufficient for large-scale microservice monitoring
- Some of the data collecting libraries, such as Zabbix, statd, collectd, jmxtrans, and so on operate at a lower level in collecting runtime statistics, metrics, gauges, and counters
  - Typically, this information is fed into data collectors and processors such as Riemann, Datadog, and Librato, or dashboards such as Graphite

The selection of monitoring tools really depends upon the ecosystem that needs to be monitored. In most cases, more than one tool is required to monitor the overall microservice ecosystem.

Monitoring tools

- Spring Boot Actuator is one of the good vehicles to collect microservices metrics, gauges, and counters
  - Netflix Servo, a metric collector similar to Actuator, and the QBit and Dropwizard metrics also fall in the same category of metric collectors
  - All these metrics collectors need an aggregator and dashboard to facilitate full-sized monitoring
- Monitoring through logging is popular but a less effective approach in microservices monitoring
- Sensu is a popular choice for microservice monitoring from the open source community
- Circonus is classified more as a DevOps monitoring tool but can also do microservices monitoring
- Prometheus provides a time series database and visualization GUI useful in building custom monitoring tools



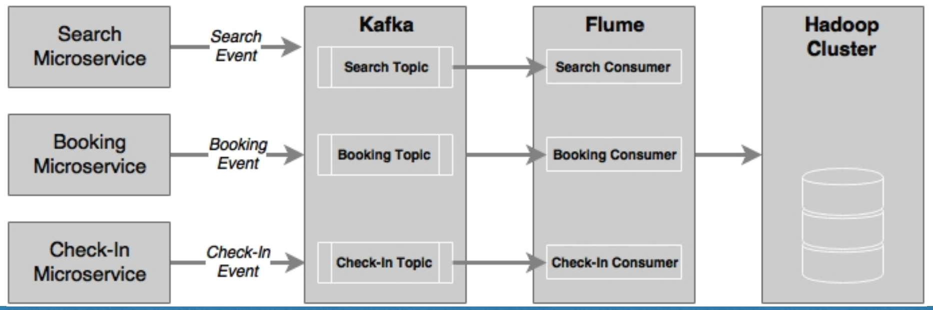
Monitoring microservice dependencies

- Mentoring tools such as AppDynamics, Dynatrace, and New Relic can draw dependencies among microservices
- End-to-end transaction monitoring can also trace transaction dependencies

When there are a large number of microservices with dependencies, it is important to have a monitoring tool that can show the dependencies among microservices. It is not a scalable approach to statically configure and manage these dependencies.

Data analysis using data lakes

- Fragmented data poses challenges in data analytics
- A data lake or data hub is an ideal solution to handling such scenarios
  - An event-sourced architecture pattern is generally used to share the state and state changes as events with an external data store



AssertLab

When there is a state change, microservices publish the state change as events. Interested parties may subscribe to these events and process them based on their requirements. A central event store may also subscribe to these events and store them in a big data store for further analysis

Microservice management

- Microservice management is another important challenge we need to tackle when dealing with large-scale microservice deployments
- In the next lecture we will explore how containers can help in simplifying microservice management

Homework 8.1

- How to implement a custom centralized logging using Elasticsearch, Logstash, and Kibana (ELK)?
- In order to understand distributed tracing, try to upgraded BrownField microservices using Spring Cloud Sleuth.

Homework 8.2

- How to enhance The BrownField microservices with Spring Cloud Hystrix and Turbine to monitor latencies and failures in inter-service communications?
  - Use the circuit breaker pattern to fall back to another service in case of failures