Observability
(Logging, Metrics, Monitoring & Alerting)

Tejas Parikh (t.parikh@northeastern.edu)
CSYE 6225
Northeastern University

Logging

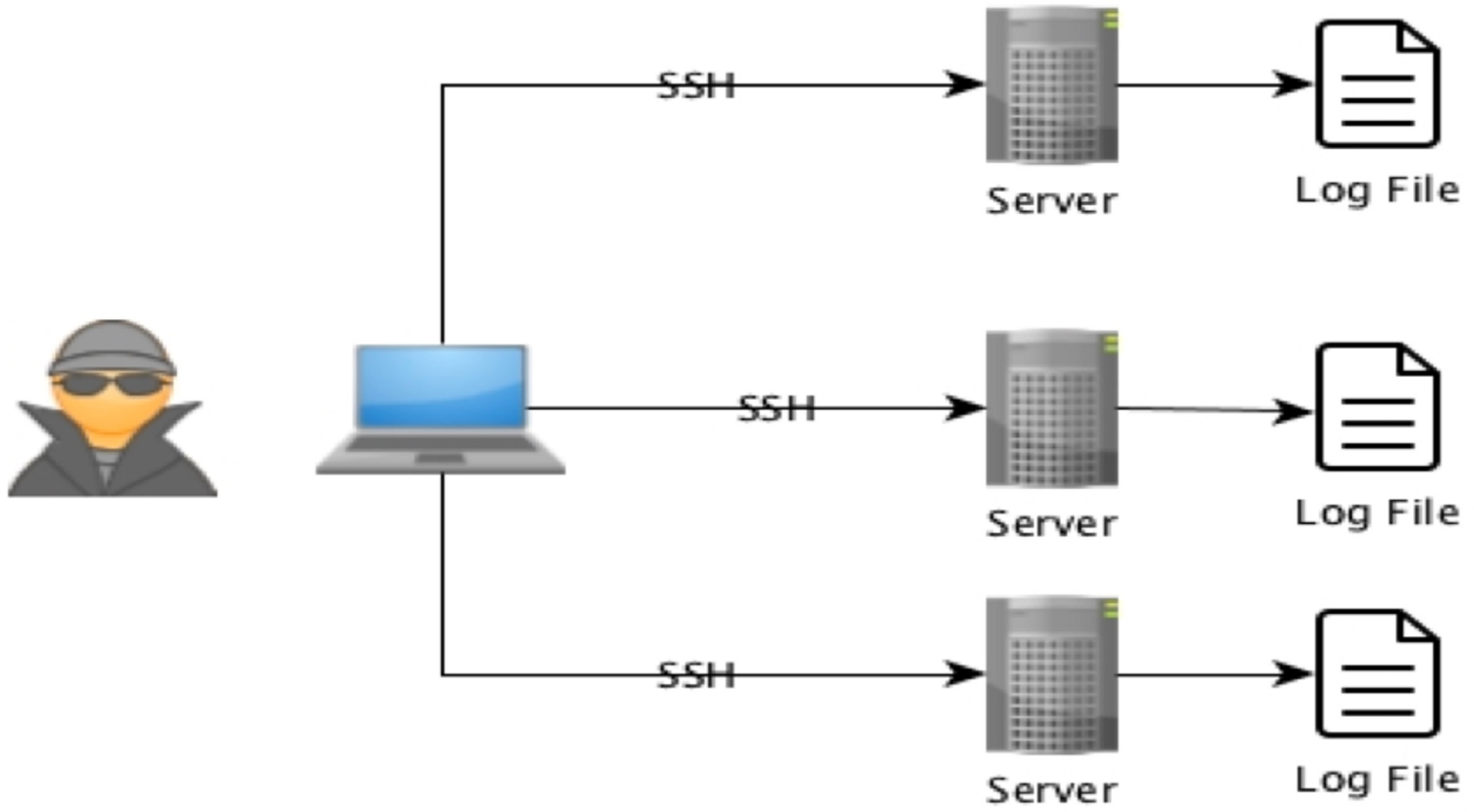
Understanding the Concept of Logging

- Application logs are one of the most import piece of analytical data in cloud-based service infrastructure.
- Logs are critical to successful debugging, troubleshooting, monitoring, and error reporting in an application.

Old Ways to Look at Logs

- A lot of times these logs are written to flat file on the local disk. Whenever an issue is reported a developer or sysadmin connects to the system via SSH and looks at the log file to find out what happened. The tools available in situation like these are grep or a text editor with search functionality.
- This becomes cumbersome when the application is hosted on multiple servers, as you now have to scan thru 100’s of log file spread across 10’s of servers without any good tools. There is no way to look at all the logs from various servers aggregated based on criteria such as timestamp, log source, etc.
- A common approach to this problem is to use centralized logging where logs from various sources can be aggregated in a centralized location.

Old Ways to Look at Logs



AWS CloudWatch Features

- CloudWatch Logs lets you monitor and troubleshoot your systems and applications using your existing system, application, and custom log files.
- With CloudWatch Logs, you can monitor your logs, in near real-time, for specific phrases, values or patterns (metrics).
- For example, you could set an alarm on the number of errors that occur in your system logs or view graphs of web request latencies from your application logs.
- You can view the original log data to see the source of the problem if needed.
- Log data can be stored and accessed for as long as you need using highly durable, low-cost storage so you don’t have to worry about filling up hard drives.

Logging

Best Practices for Logging

When to Log

- Logs are not generated automatically by your application.
- The application developer i.e. you must deliberately create them to provide insight.
- First and foremost, log data that will help you troubleshoot an issue without a debugger.
- Debugging an application is not the only reason to log though.
- Log data when important events occur such as errors, configuration changes, component startup or shutdown and more.
- You may also want to simply monitor your application to make sure it is working and undergoing activity of some sort.
- **Don’t muddle the logs by logging everything you can.**

What to Log

- You can probably think of lot of information you want to put in a log file and the possibilities are technically endless.
- From a high level, the best logs messages tell you exactly what happened, and when, where and how.

Exceptions and Errors

- Both caught and uncaught exceptions should be logged.
- If a certain exception happens too often it is good to know.
- You should log any information you have about the error (type, message, stack trace, input parameters, and what the application was doing when the error occurred).

Warnings

- Sometimes issue arise that aren’t quite errors and don’t stop action from continuing but should be brought to somebody’s attention at a later time. These issues indicate potential issues in the application.

Informational Messages

- To investigate problems reported by users of your service, it's helpful to store all input and output of the service.
- Investigating customer issues otherwise can be very difficult.
- If you services call another service, it may be helpful to log input and output of that service as well. This will help identifying issues at the right level (your service vs. their service).

Debug Messages

- Debug log messages aid you when fixing defect and/or following the flow of the code execution.
- You can log entry and exit of methods, input and output values and anything else that will help your team identify issues in your services.
- **Debug logging should NOT be enabled in production.**

Logging

How to Log

Use The Right Logging Levels

It is important to ensure those log messages are appropriate in content and severity.

Log Level	Description
ALL	All log levels including custom levels.
TRACE	Designates finer-grained informational events than the DEBUG.
DEBUG	Designates fine-grained informational events that are most useful to debug an application.
INFO	Designates informational messages that highlight the progress of the application at coarse-grained level.
WARN	Designates potentially harmful situations.
ERROR	Designates error events that might still allow the application to continue running.
FATAL	Designates very severe error events that will presumably lead the application to abort.
OFF	The highest possible rank and is intended to turn off logging.

Log Level Inheritance

- A log request of level p in a logger with level q, is enabled if p >= q.
- For the standard levels, we have ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

Production Log Level

By default the message priority should be no lower than INFO. That is, by default DEBUG message should NOT be seen in the logs.

Timestamp in Log Events

- The correct time is critical to understanding the proper sequence of events.
- Timestamps are critical for debugging, analytics, and deriving transactions.
- All log events should have the timestamp in GMT/UTC time zone to avoid confusion when looking at logs from servers hosted in various time zones.
- Logging framework will automatically add a timestamp field to the log message.

Use developer-friendly formats

- Write logs in developer-friendly formats like JavaScript Object Notation (JSON), which is also readable by humans.
- You also avoid the hassle of dealing with new line characters in messages such as exception stack trace, which can end up as multiple log messages instead of one.

Avoid Multi-Line Events If Log Format Is Not JSON

- Multi-line events generate a lot of segments, which can affect how data is indexed.
- Consider breaking multi-line events into separate events or use JSON format so that message stays as 1 event.

Log Locally To Files

If you log to a local file, it provides a local buffer and you aren't blocked if the network goes down.

Log Message Encoding

- Log in TEXT format only.
- Binary information cannot be processed by logging stack and cannot be searched and thus will be of no help.
- Log messages should be in ENGLISH only.

Override toString() and hashCode() method

- Always override toString() to give a good representation of the object.
- Too often you get output that looks like SomeFancyObject@15ba6d5, which is useless.

Guarded Logging

Always check if logging for particular log level (such as DEBUG) is enabled before assembling log message. This will save CPU cycles and improve performance.

```
// When logging is set to ERROR following message does not get logged but log4j ends up spending
   time to create the log message
   log.debug(“fancy” + “log”+ “message”);


//Optimal way to write debug log events
   if(log.isDebugEnabled()) {
   log.debug(“fancy” + “log”+ “message”);
   }
```

What Not to Log

Do not log sensitive data such as passwords, credit card information, social security number, or any other Personally Identifiable Information.

Java Logging Framework Libraries

- Apache log4j
- Apache Extras for Apache log4j
- Apache Commons Lang
- Json Smart
- log4j-jsonevent-layout

Where Does System.out & System.err Log Messages Go?

- System.out and System.err are both redirected to CATALINA_BASE/logs/catalina.out when using Tomcat's startup scripts (bin/startup.sh/.bat or bin/catalina.sh/.bat).
- Any code that writes to System.out or System.err will end up writing to that file.
- If your webapp uses System.out and/or System.err a lot, you can suppress this via the 'swallowOutput' attribute in your configuration element and send to different log files (configured elsewhere: see the documentation for configuring logging).

Metrics

Measure Anything, Measure Everything

- Application metrics usually provide the most important data point yet they are the hardest to collect.

Metrics can answer questions like

- How slow is this query?
- How many times is this page accessed?
- What does my JVM memory usage look like during normal operations?

Data Source Types

- **Counters** - A counter is a numeric value that gets incremented when some event, such as a client connecting to a server, occurs. When reported to monitoring systems, a counter will typically be converted to a rate of change by comparing two samples. Counter values should be monotonically increasing.
- **Gauge** - Gauges store an arbitrary value. Simply put, it takes any number and gets flushed to the backend.
- **Timers** - Timers collect numbers. E.g. Response time for for an API. They are not necessarily limited to store values of time.

Popular Client Libraries

- Statsd
- Netflix Servo

Popular Backend Datastores

- Graphite
- InfluxDB
- OpenTSDB
- Prometheus

Benefits of Metrics Data

- Metrics data helps with analysis of long term trend.
- Compare performance of application across different versions.
- Data can be used for Alerting.
- Data can be helpful in Debugging.

Alerting

Need for Alerting

- Automated alerts are essential to monitoring.
- They allow you to spot problems anywhere in your infrastructure, so that you can rapidly identify their causes and minimize service degradation and disruption.

Level of Alerting

- **Low Severity** - for record only.
- **Moderate Severity** - Usually via email notification. Requires human intervention but not right away.
- **High Severity** - The most urgent alerts that require human intervention right away. These are the alerts that can impact customer facing SLAs.

Additional Resources

See Lecture Page