Monitoring & Logging

Richard Thomas, Larene Le Gassick & Guangdong Bai

What is Logging?

email: contact@larene.dev

What is Logging?

- Record what happens in a program
  - and in what order
- Diagnostic logging — low-level program-related logs
  - function calls, errors, health check, …
- Audit logging — business logic logs
  - user login
  - password change
  - order placed
- Transaction logs (database specific)

Why Logging?

- Debug problems and analyse root cause
- Know if your program is running correctly in production
- Find out what your users are doing
  - For reporting, revenue, metrics
- Audit security and compliance, and respond to incidents
  - Recall accountability learnt in security quality attribute
- Troubleshooting for users (e.g. support desk)
- Find out related actions between services
  - Log correlation

Logging in Practice

- Console (stdout) — you've done this
  - `print("some string")` in Java or Python
  - `Console.Writeline("some string")` in .NET
  - `console.log("some string")` in JavaScript
- File
  - "202604201120.log"
  - Use java.logging.util, Log4j2, SLF4J, or other libraries …
- Logging server
  - Similar to logging to a file, but over HTTP
    - or some other transfer protocol

Handlers & Formatters

- Handlers
  - Console, Stream, Socket, Memory
  - File (single file or rotating files)
  - Write your own
- Formatters
  - SimpleFormatter: “human readable”
  - XMLFormatter: XML
  - Write your own

Logging: java.util.logging

```
static final Logger LOGGER =
       Logger.getLogger(LogExample.class.getPackage().getName());

static final Logger LOGGER =
       Logger.getLogger(this.getClass().getPackage().getName());
```

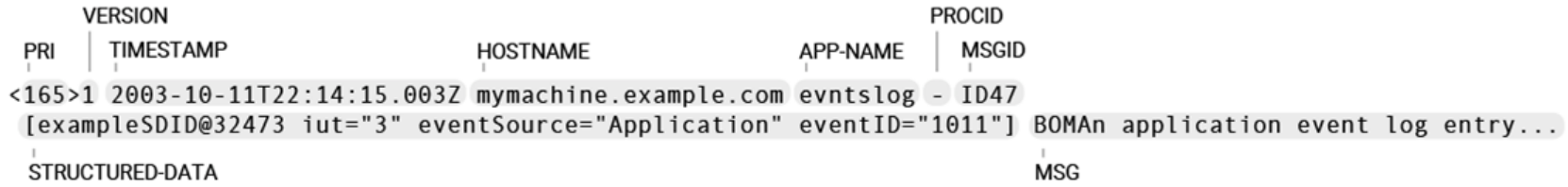
```
LOGGER.log(Level.WARNING, "My {0} has turned {1}",
           new Object[]{bodyPart, colour});
```

Simple Logging

- Log message, in its simplest form, is just a string
- Add a date and time at the start of that string
- Add a "log level" between the date and the message
  - e.g. debug, info, warning, error

```
2021-07-29 14:54:55.1623|INFO|New report created by user 4253
2021-07-29 14:56:19.2814|INFO|Report updated by user 257
2021-07-29 14:59:20.4211|INFO|User 478 deleted report
```

- Standards have been developed, such as "Syslog"

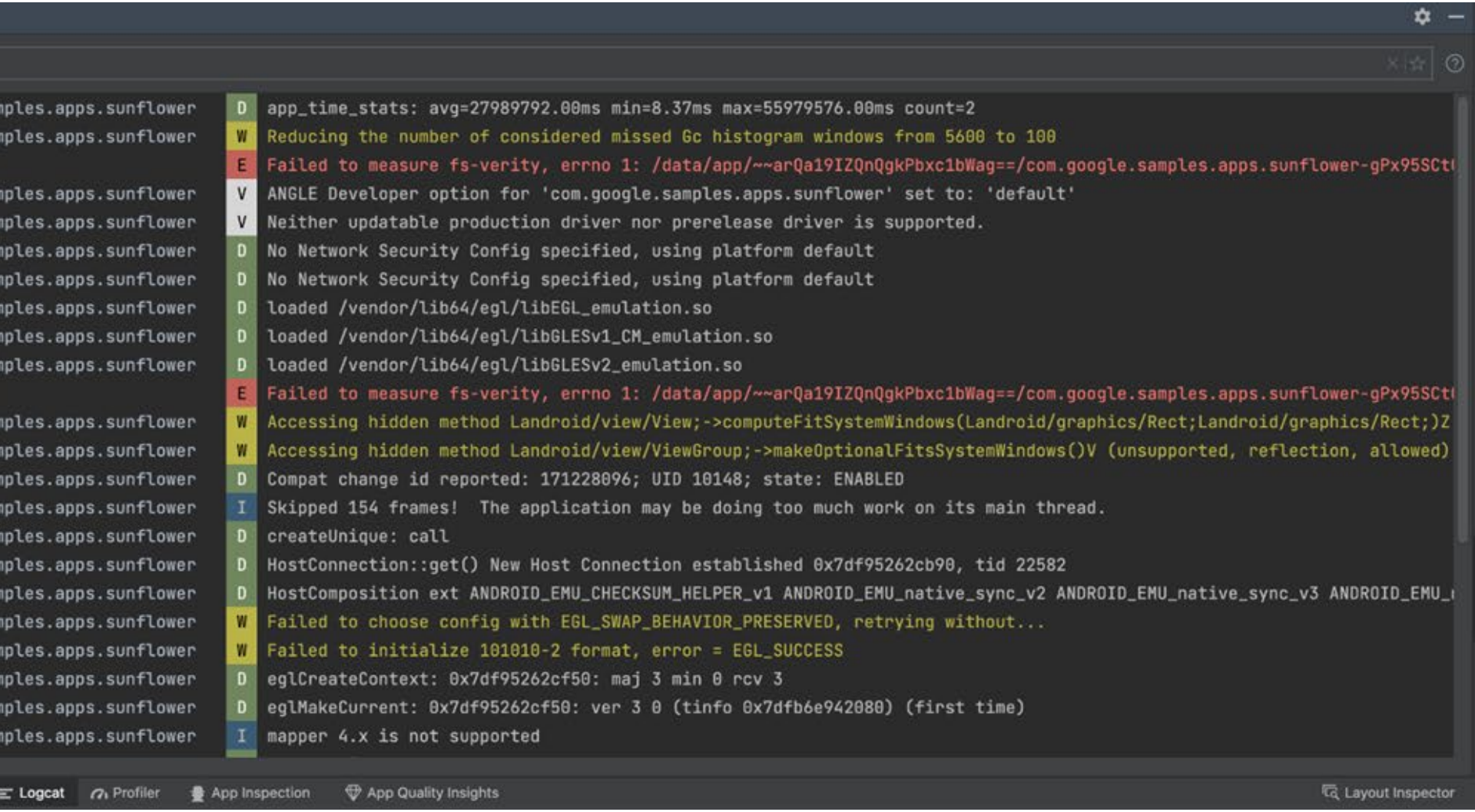


Log Levels

- Library specific, but generally:
  - DEBUG: Detailed internal information
  - INFO: General system events
  - WARN: Unexpected behaviour, non-critical
  - ERROR: System errors needing attention
  - FATAL: System crash or critical failure

- j.u.l levels: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST
  - Level is cut off for recording messages

Android Logcat



Performance

```
LOGGER.warning("my " + bodyPart + " has turned " + colour);
```

What are the performance characteristics?

- When logging is enabled?
- When logging is not enabled?
  - early culling based on levels

```
LOGGER.log(Level.WARNING, "My {0} has turned {1}",
           new Object[]{bodyPart, colour});
```

Simple Logging Issues

- Large systems — How to search and filter?
  - grep or find
  - regex
- How do we create reports?
  - Lot of manual work
- No consistent pattern, very fragile

Structured Logging

- From (strings)

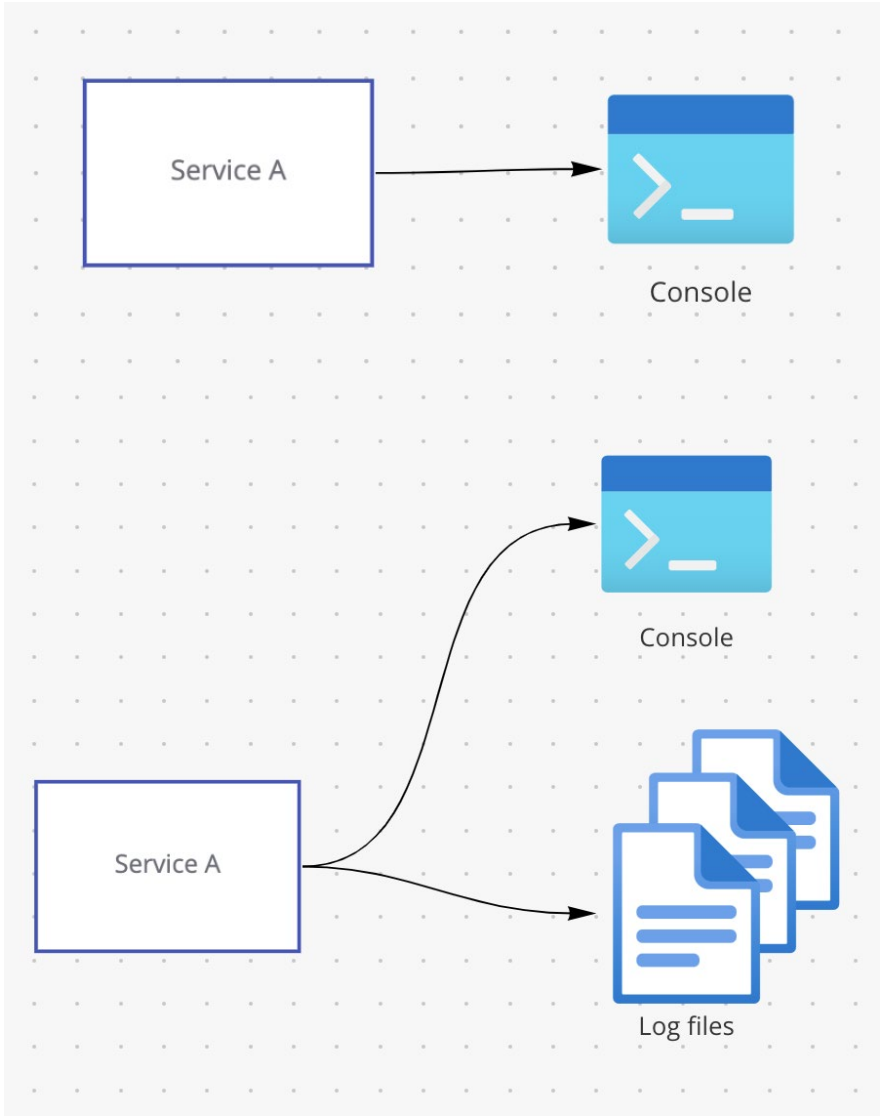
```
2021-07-29 14:54:55.1623|INFO|New report created by user 4253
2021-07-29 14:56:19.2814|INFO|Report updated by user 257
2021-07-29 14:59:20.4211|INFO|User 478 deleted report
```

- To (objects)

```
{
    "TimeStamp": "2021-07-29 14:52:55.1623",
    "Level": "Info",
    "Message": "New report created",
    "UserId": "4253",
    "ReportId": "4567",
    "Action": "Report_Creation"
}
```

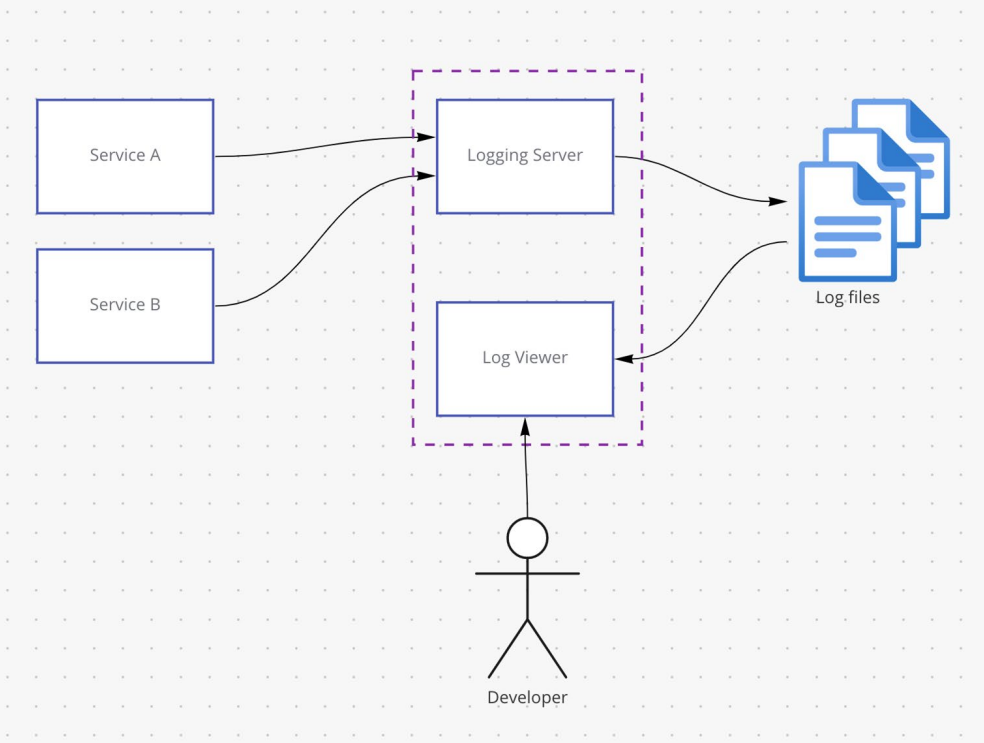
Logging Architectures

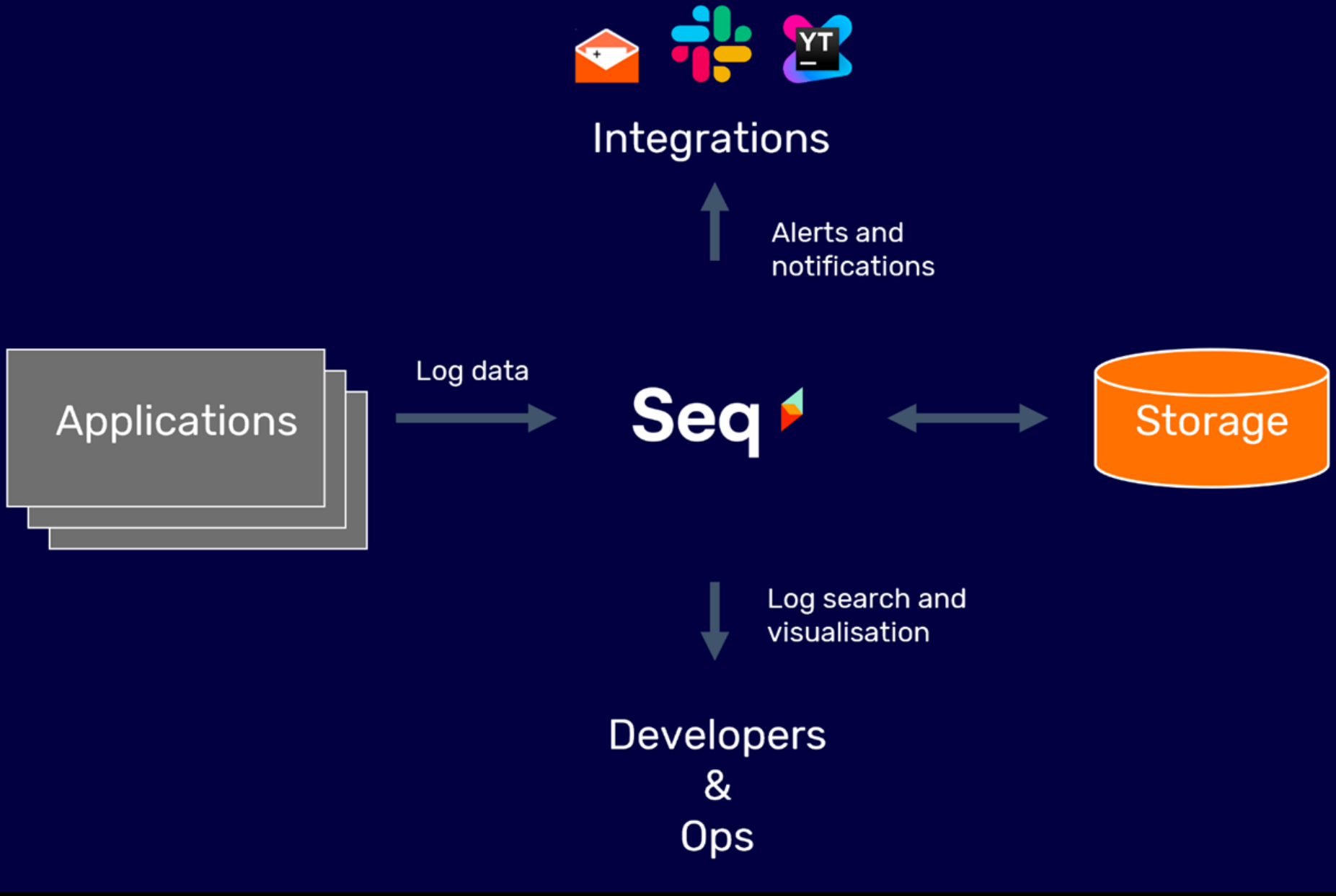
- **Goal**: Transform scattered log entries into centralised, searchable, actionable insights
- Service A has a **logging library** installed, providing functions to make structured logging easier
- Service A also has a **log "appender" or "sink"** installed – helpers responsible for sending the logs where they're meant to go (e.g. console, or file, or server)



Logging Architectures

- Services A and B have the same **logging library** installed, as well as a **log sink/appender** sending log messages to a **specific log server**
- **Log server** receives logs from the services, and stores the logs
- Developer or internal staff view the logs in a **log viewer**
  - Log servers often have search and analysis tools




Integrations
Alerts and notifications
Applications
Log data
Seq
Storage
Log search and visualisation
Developers & Ops

General Overview of Logging Architecture

- **Log Producers**: Generate logs (apps, services, containers)
- **Log Shippers**: Agents that collect and forward logs (e.g. Fluentd, Filebeat)
- **Log Aggregators**: Buffers and routes logs, enabling scaling and decoupling (e.g., Kafka, Redis)
- **Log Storage**: Log persistence for querying and analysis (e.g. Elasticsearch, S3, Loki)
- **Log Analysis Tools**: Dashboards, alerts, and visualisation (e.g. Kibana, Grafana, Splunk)

Popular Logging Libraries

Logging libraries are dependencies added to project

Many available, minor variations on the appenders or sinks

- Log4j2 (Java)
- java.util.logging (Java)
- SLF4J (Java)
- Serilog (.NET)
- Microsoft.Extensions.Logging (.NET)
- Bunyan (Node JS)
- logging (Python)
- many more…

Popular Log Server and Analysis Tools

Self-managed logging server, or SaaS or PaaS product (check prices!)

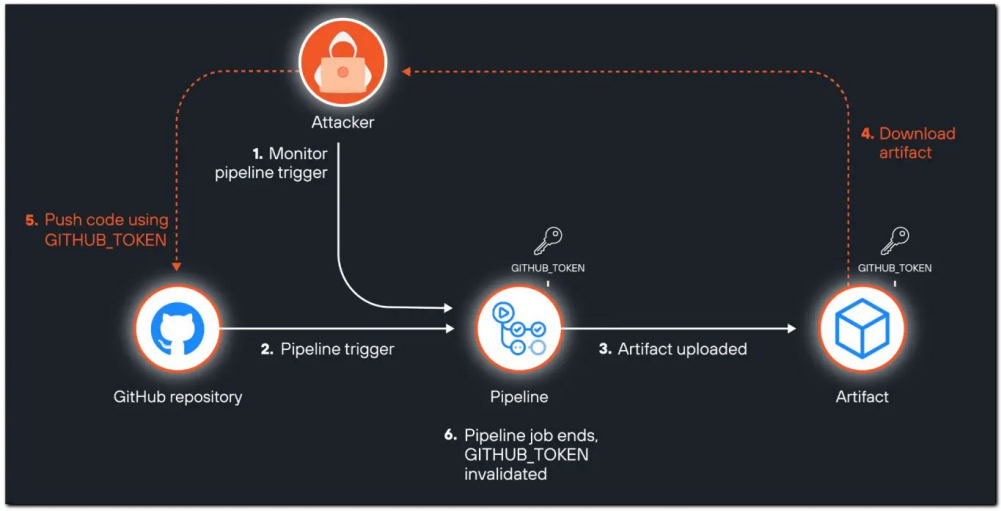
- **AWS Cloudwatch**
- Splunk
- Datadog
- Logstash+Elasticsearch+Kibana+Beats — Elastic/ELK stack
- Azure Monitoring
- Seq (self-managed only)
- many more…

Logging Best Practice

- Do not log passwords!
- Do not log personally identifiable information (PII)!
  - Unless absolutely necessary
    - e.g. User ID instead of Email Address

Logging Data Leak Incidents

2018: Twitter urged hundreds of millions of users to reset their passwords after a developer accidentally logged plain text user credentials in their internal systems



2024: GitHub Actions artifacts generated during CI/CD workflows could potentially leak GITHUB_TOKEN of project developers

https://www.developer-tech.com/news/unit-42-researchers-critical-github-actions-vulnerability/

Logging Best Practice

- Do not log passwords!
- Do not log personally identifiable information (PII)!
  - Unless absolutely necessary
    - e.g. User ID instead of Email Address
- Don’t log everything, the more noise, the more likely you’ll miss something important
- Don’t expect to store logs forever, most have retention policies of 7 days or a month
- Test your logging!

More Info

- https://docs.oracle.com/en/java/javase/17/docs/api/java.logging/java/util/logging/package-summary.html
- https://www.slf4j.org/

What is Monitoring?

email: contact@larene.dev

Monitoring

- Observe dynamic behaviour of system
  - Alert when events happen
    - e.g. node fails
  - Dashboard to view system status
- Particularly beneficial for distributed systems
  - Multiple systems to monitor
- System architecture
  - Devices hosting system containers

Host

- Device to monitor
  - Component of system
- Any infrastructure delivering system behaviour
  - Servers, gateways, routers, …

Item

- Data to be monitored
  - e.g. CPU load, memory utilisation, network load, queue length, …
- Consider polling interval
  - Frequently – may stress device
  - Infrequently – may delay notice
- Record data
  - How much?
  - How long?

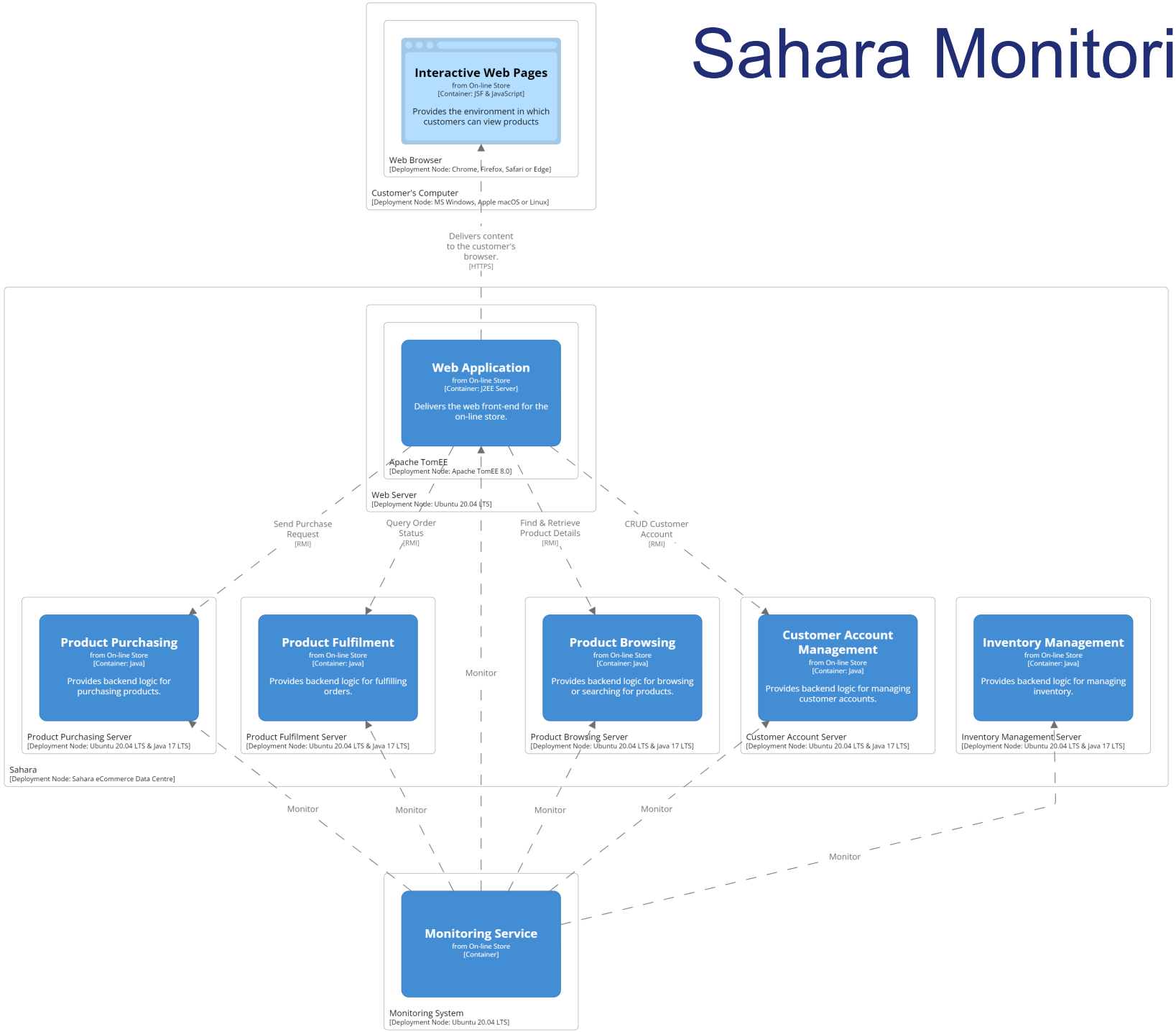
Trigger

- Condition requiring action
  - e.g. CPU load too high, queue too long, …
- Severity
  - *Info* through to *Urgent*
    - Info: CPU Load
    - Warning: CPU Load > 80% for > 5 minutes
    - High: CPU Load > 90% for > 5 minutes
    - Urgent: CPU Load > 95% for > 5 minutes

Action

- What to do when trigger occurs
  - Alert maintenance team
  - Execute script
  - …
- Severity
  - Do different things based on severity

Sahara Monitoring



Sahara Monitoring

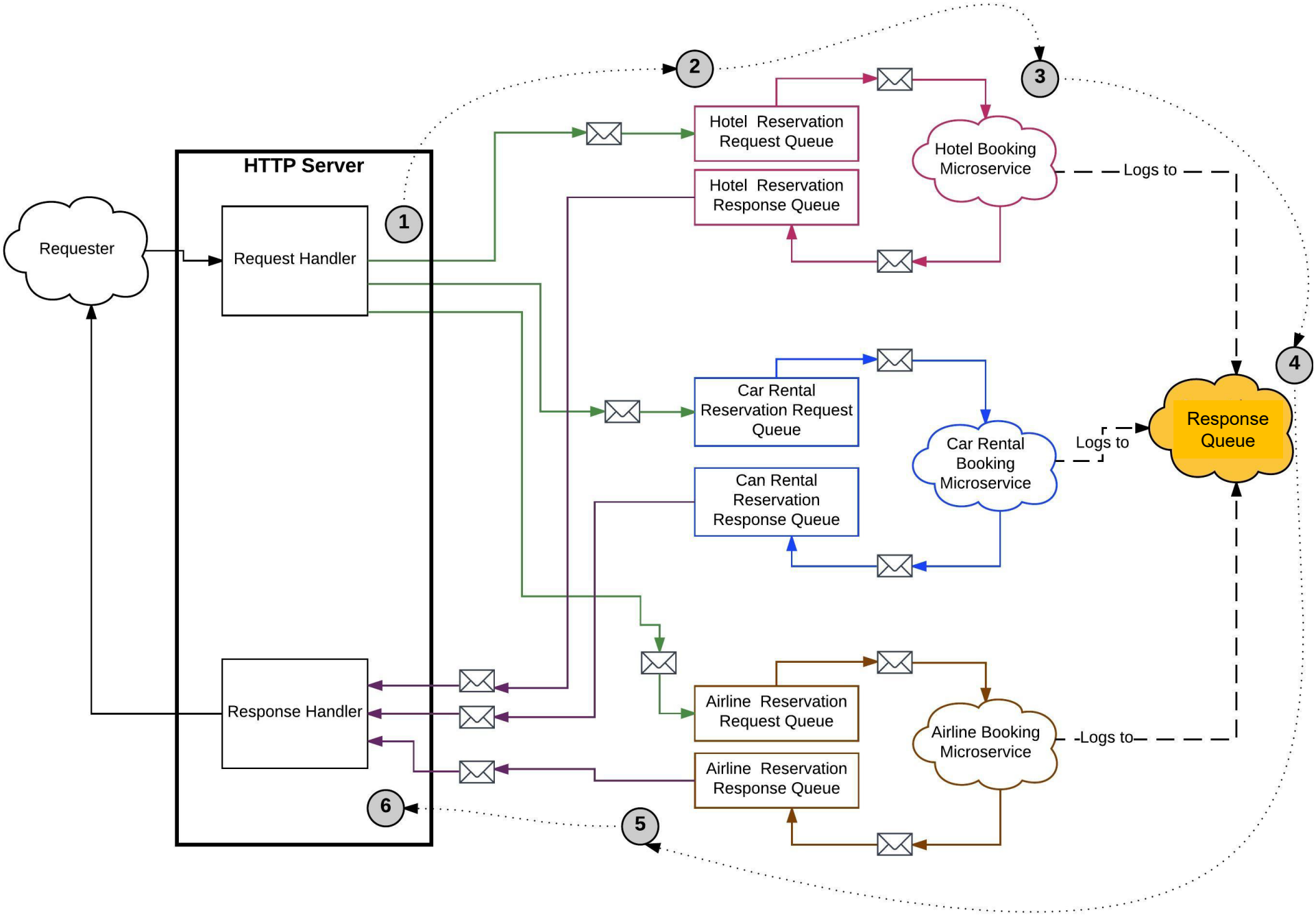
- Items
  - Purchasing: CPU load, network connections
  - Browsing: DB response time, dropped connections
  - Web App: JSF thread count, memory usage
- Triggers
  - Memory Usage > 80%
- Actions
  - Memory Usage > 80%
    - Dashboard: Set web server icon orange
  - Memory Usage > 85% for > 5 minutes
    - Dashboard: Set web server icon bright red
    - …

Many Tools

- Zabbix
  - Open source
  - https://www.zabbix.com/
- DataDog
  - https://www.datadoghq.com/
- …

Correlation IDs

- Track sequence of activities in distributed system
  - Deal with asynchronous messaging
- Allows recognition of messages related to each other
  - Can deliver result, now that everything is done
    - e.g. Email customer that all ordered items have shipped
- Essentially a transaction ID
  - Yes, synchronous transactions are a myth
  - But, we need to track events that deliver external outcomes



https://www.rapid7.com/blog/post/2016/12/23/the-value-of-correlation-ids/

Correlation IDs

- Generate when initial request is received
  - Needs to be unique for system
- Pass as part of message
  - Often part of message header
    - e.g. HTTP header, X-Correlation-ID
- Pass to all services
  - Allows tracking of request processing

Correlation IDs & Logs

- Record correlation ID in all log entries
- Allows tracking of activity across distributed services
- Particularly important for microservices
  - Services don’t know how other services contribute to behaviour delivery

Unique Correlation IDs

- UUID
  - Simple
  - IDs are large
  - Can be traced to generating computer
  - Might duplicate if generated close together
    - less than 7 seconds
  - Might duplicate across computers

Unique Correlation IDs

- Application IDs
  - Generated by application
  - Logic needs to manage uniqueness across requesters
    - e.g. customer ID + browser time
- CUID
  - Collision resistant IDs
  - Designed for horizontal scaling
  - Monotonically increasing IDs
    - allows binary search
    - generate < 10,000 per millisecond
    - process clocks must be ***synchronised***

More Info

- Correlation ID Pattern
  - *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*
    - Gregor Hohpe, Bobby Woolf
  - https://www.informit.com/articles/article.aspx?p=1398616
- CUID
  - http://usecuid.org/