Monitoring & Logging

Logging Revision

- Record what happens in a program
    - and in what order

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});
```

Logging Levels: java.util.logging

- Levels: `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST`
  - in order
  - also `OFF` and `ALL`
- Level is cut off for recording messages

Logging Hierarchy: java.util.logging

- Hierarchical loggers – by package level
  - `com.csse6400.LogExample`
  - `csse6400` is parent of `csse6400.LogExample`
- Messages get propagated up the tree by default
- Intermediate level loggers must be created before use
  - root created automatically

```
Logger childLogger =
       Logger.getLogger(LogExample.class.getPackage().getName());
Logger parentLogger =
       Logger.getLogger(LogExample.class.getPackage());

childLogger.log(Level.WARNING, "Parent and child are notified");
```

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});
```

Logging: java.util.logging

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

Properties: java.util.logging

- `logging.properties` file
  - Need to tell JVM to use it
  - Default location is application root directory
- Specifies
  - Handlers
  - Formatters
  - Levels
- Easy adjustment without changing code

SLF4J

- Many logging frameworks, which share common features
  - Log message, where is it from, when is it from, importance level, an exception with stack trace, ...
  - Even advanced features (rotating log files, database storage, alerts by email, ...) are shared across multiple packages
- Brings all these together under a uniform interface
  - Allows you to use a single interface for your logging, but switch logging systems, either before or after compiling
  - Avoids every library having its own logging facility, or even its own logging
- Efficient, if using logback
  - Compared to j.u.l

SLF4J

- Uses all of the logging levels from Log4j except `FATAL`
- Supports bindings to NOP (do nothing), Simple (console), j.u.l, Log4J, logback, Java Commons Logging (Apache)
- Richer configuration
- Supports Markers (from logback)
  - used to filter log messages in config
- Based on a very simple application of the Façade Pattern
  - Uses an adapter pattern internally
  - Simple Logging Façade for Java (slf4j)

SLF4J

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

LOGGER.warn("My " + bodyPart + " has turned " + colour);

LOGGER.warn("My {} has turned {}", bodyPart, colour);

LOGGER.info("Coordinate is: x={}, y={}, z={}",
           position.getX(), position.getY(),
           position.getZ());
```

More Info

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

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

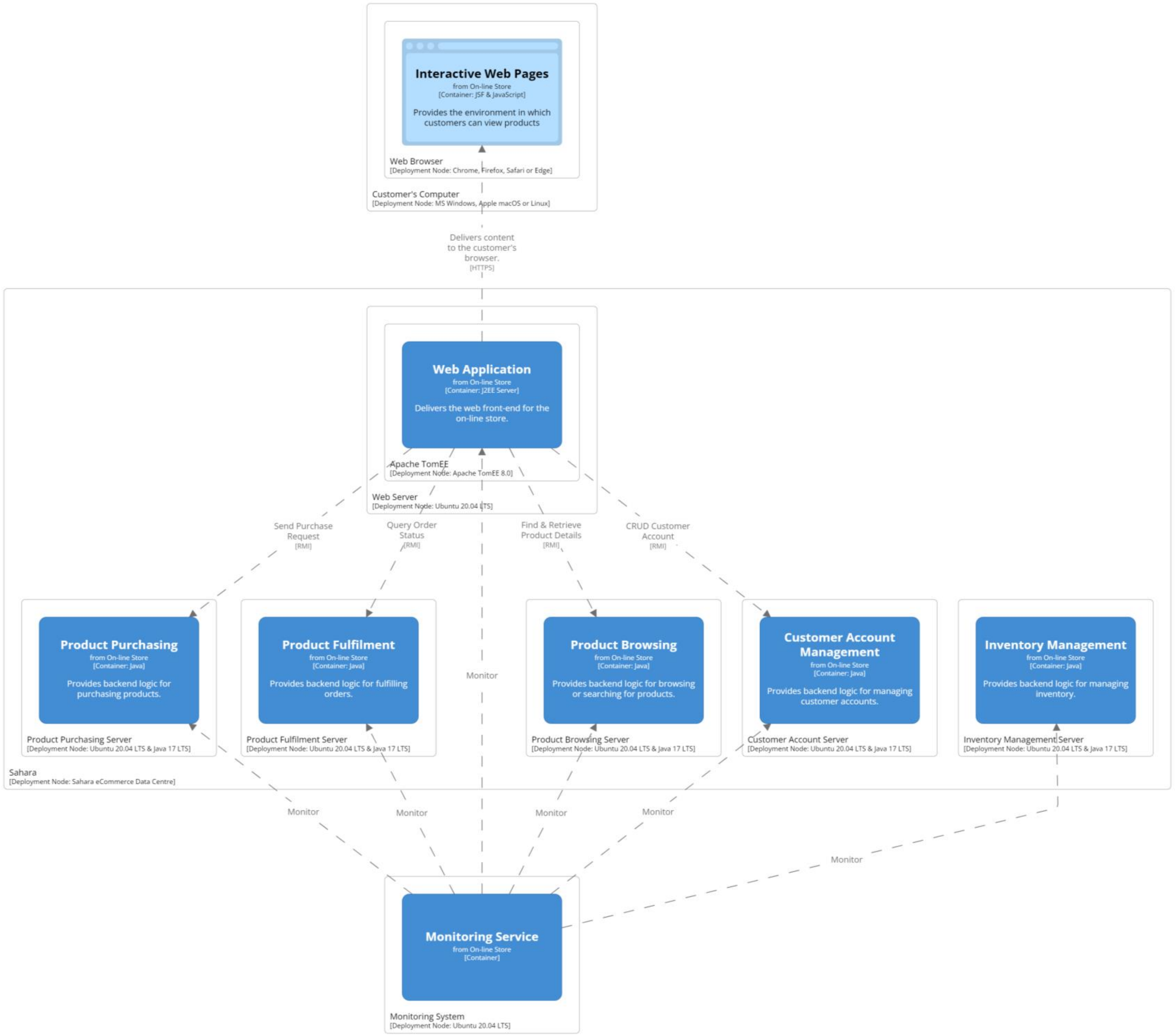
- Data to be monitored
  - e.g. CPU load, memory utilisation, network load, queue length, …
- Consider polling interval
  - Frequently may stress device
  - Infrequently may delay notice
- Record data
  - How much
  - How long

Trigger

- Condition requiring action
  - e.g. CPU load too high, queue too long, …
- Severity
  - *Info* through to *Urgent*
    - 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


Interactive Web Pages
from On-line Store
[Container: JSF & JavaScript]
Provides the environment in which customers can view products
Web Browser
[Deployment Node: Chrome, Firefox, Safari or Edge]
Customer's Computer
[Deployment Node: MS Windows, Apple macOS or Linux]
Delivers content to the customer's browser.
[HTTPS]
Web Application
from On-line Store
[Container: J2EE Server]
Delivers the web front-end for the on-line store.
Apache TomEE
[Deployment Node: Apache TomEE 8.0]
Web Server
[Deployment Node: Ubuntu 20.04 LTS]
Send Purchase Request
[RMI]
Query Order Status
[RMI]
Find & Retrieve Product Details
[RMI]
CRUD Customer Account
[RMI]
Product Purchasing
from On-line Store
[Container: Java]
Provides backend logic for purchasing products.
Product Purchasing Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Product Fulfilment
from On-line Store
[Container: Java]
Provides backend logic for fulfilling orders.
Product Fulfilment Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Monitor
Product Browsing
from On-line Store
[Container: Java]
Provides backend logic for browsing or searching for products.
Product Browsing Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Customer Account Management
from On-line Store
[Container: Java]
Provides backend logic for managing customer accounts.
Customer Account Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Inventory Management
from On-line Store
[Container: Java]
Provides backend logic for managing inventory.
Inventory Management Server
[Deployment Node: Ubuntu 20.04 LTS & Java 17 LTS]
Sahara
[Deployment Node: Sahara eCommerce Data Centre]
Monitor
Monitor
Monitor
Monitor
Monitor
Monitoring Service
from On-line Store
[Container]
Monitoring System
[Deployment Node: Ubuntu 20.04 LTS]

Sahara Monitoring

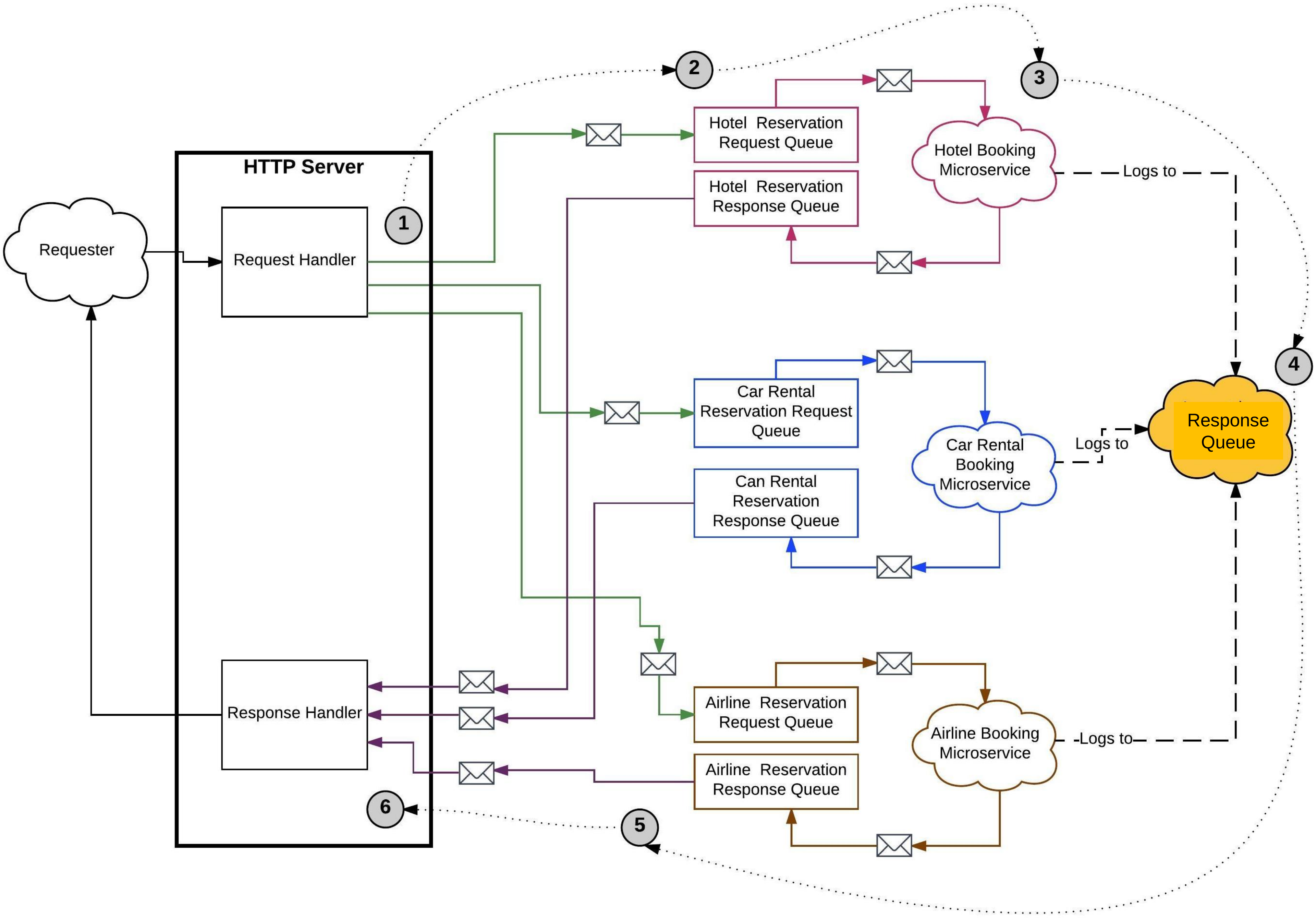
- Items
  - Purchasing: CPU load, network connections
  - Browsing: DB response time, dropped connections
  - Web App: JSF thread count, memory usage
- Triggers
  - Memory Usage > 90%
- Actions
  - Memory Usage > 90%
    - Dashboard: Set web server icon orange
  - Memory Usage > 90% for > 5 minutes
    - 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
  - Dealing 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



2
3
HTTP Server
Hotel Reservation Request Queue
Hotel Booking Microservice
Logs to
Hotel Reservation Response Queue
1
Requester
Request Handler
Car Rental Reservation Request Queue
4
Response Queue
Car Rental Booking Microservice
Logs to
Can Rental Reservation Response Queue
Response Handler
Airline Reservation Request Queue
Airline Booking Microservice
Logs to
Airline Reservation Response Queue
6
5

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/