Week 4: Service-to-Service Communication
Unit 1: Logging and Tracing Using SLF4J

Logging and Tracing Using SLF4J
Motivation

No debugging of running microservice in production

- Debug port is closed by default
- Applications must be SOX-compliant, i.e. immutable
- Multiple instances, hard to target specific instance

"I saw some error message two weeks ago"

- Do you know which code ran at that time?
- Can you identify the corresponding request?
- Can you locate the problem if it happened in a downstream service?
- Do you have the data necessary to understand/reproduce the error?
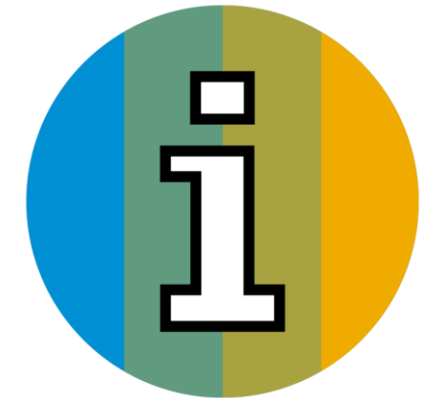
Logging and Tracing Using SLF4J

What should be logged?

Basics	timestamp (UTC), thread, logger name, severity
Operational	traces (for important methods), errors, configuration,…
Categories	useful for filtering: SQL, Authentication, Order,…
Business	used features, purchases, response times (SLA?),…
Security	login, denied access, privileged account actions,…

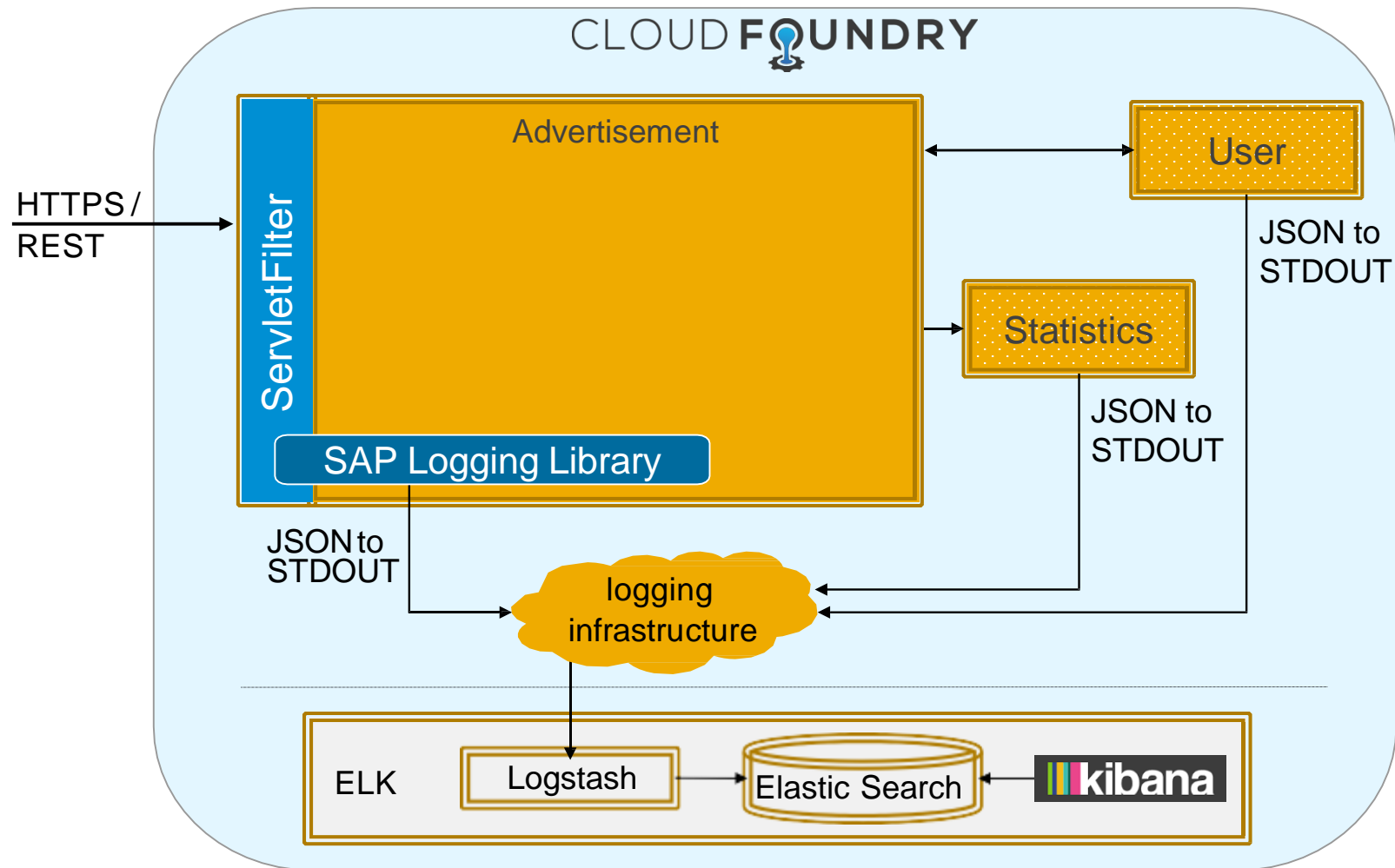# Logging and Tracing Using SLF4J
What should NOT be logged?

- Confidential information

    - Passwords, credit card details,…

- Sensitive data, personal information

- Respect rules set up by regulations and legislation, e.g. EU GDPR

- Instead log IDs: e.g. log *user ID* instead of *name, email,…*

Attention: Sensitive data in REST URIs or stack traces

Logging and Tracing Using SLF4J
Big picture

Logging and Tracing Using SLF4J

Logging library and SLF4J – Simple logging facade for Java

SLF4J Implementations: <u>Logback</u>, Log4J 2,…

Java Logging Support for Cloud Foundry ([GitHub](#))

- JSON formatting
- Servlet filter: logs performance data and more

Example: expose log message

```
import org.slf4j.*;

// instantiate logger
Logger logger = LoggerFactory.getLogger(this.getClass());

// log message with log level "info"
logger.info("incoming request: {}", request);
```

> use getClass() to avoid copy-paste errors (in non-static context)

> message gets formatted, placeholder gets replaced only when log is logged

Logging and Tracing Using SLF4J
Overview log levels and when to use what?

The levels in descending order are:

error	for critical exceptions that need immediate attention

warn	for exceptions that do not trigger alerts, e.g. NotFound

info	for (API, REST) requests, used features

debug	for information helpful only when debugging, e.g. subtotals

trace	for very detailed information, e.g. parameter values

Typically only messages of severity INFO or higher are logged in production

Logging and Tracing Using SLF4J
Correlation ID and MDC – Mapped Diagnostic Context

Correlation ID

- Correlates log messages of different microservices

- Automatically added to MDC using servlet filter (when using Java Logging Support for Cloud Foundry)

- Add manually to outgoing request headers

Provide information in MDC – once per thread…

```
MDC.put("endpoint", "api/v1/any" + id);
```

...so that it is included in all messages of the same thread

```
logger.info("customer {} logged in", customer.getId());
// 1441720072  [INFO]  [c.s.f.Bar] [endpoint=api/v1/any/1234]  customer 1234 logged in
logger.info("customer used feature");
// 1441720073  [INFO]  [c.s.f.Bar] [endpoint=api/v1/any/1234]  customer used feature
```

Logging and Tracing Using SLF4J

Exercise 12 + 13

Exercise 12:
Setup SLF4J Logger

Exercise 13:
Use SLF4J Features
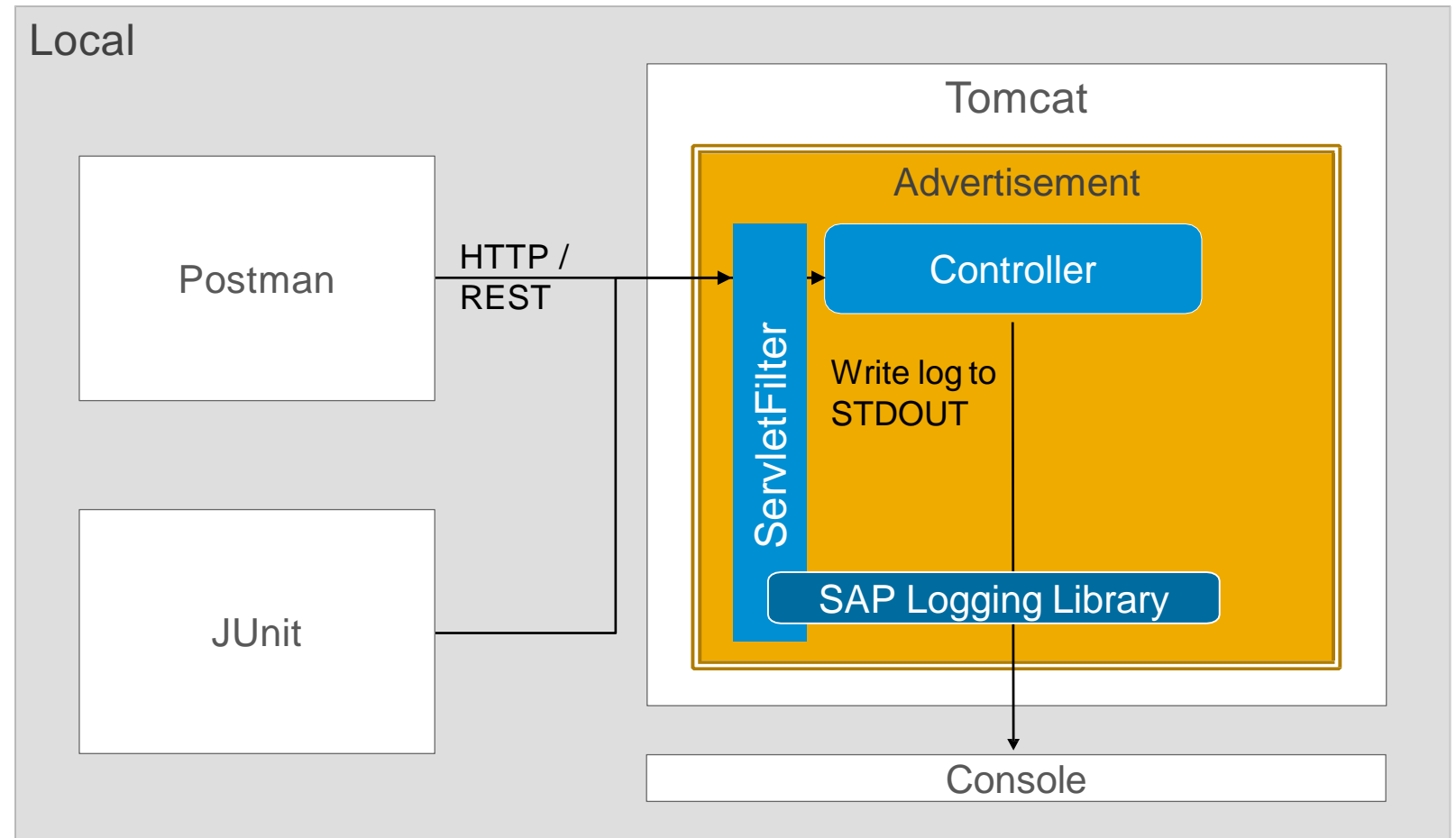
Logging and Tracing Using SLF4J

Review of sample solution
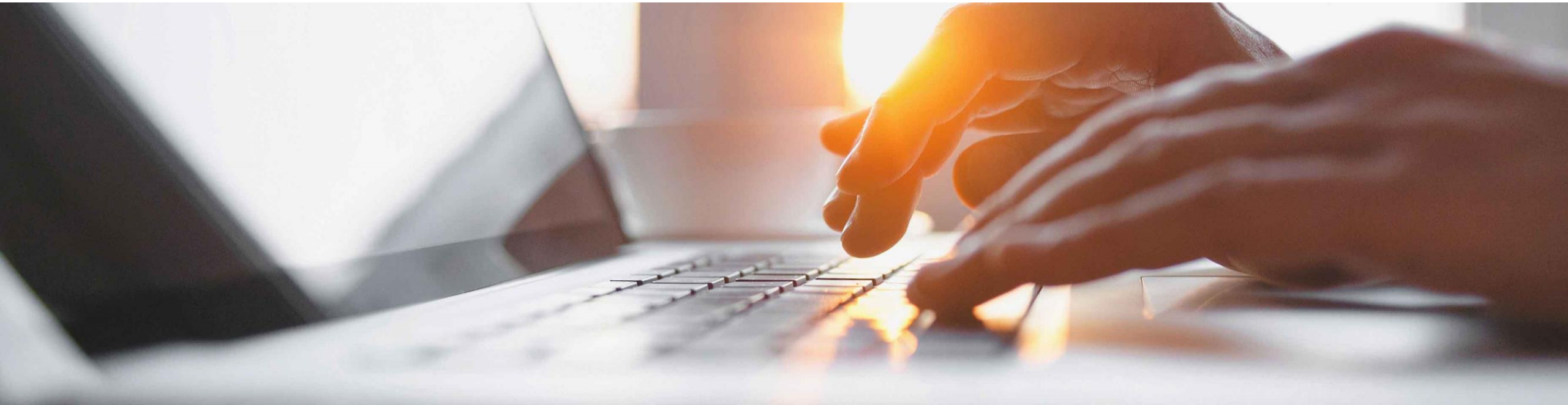
Logging and Tracing Using SLF4J
Demo: SAP logging support library

Logging and Tracing Using SLF4J
Further reading

- Java Logging Support for Cloud Foundry
- Simple Logging Facade for Java (SLF4J)
- Further SLF4J performance hints
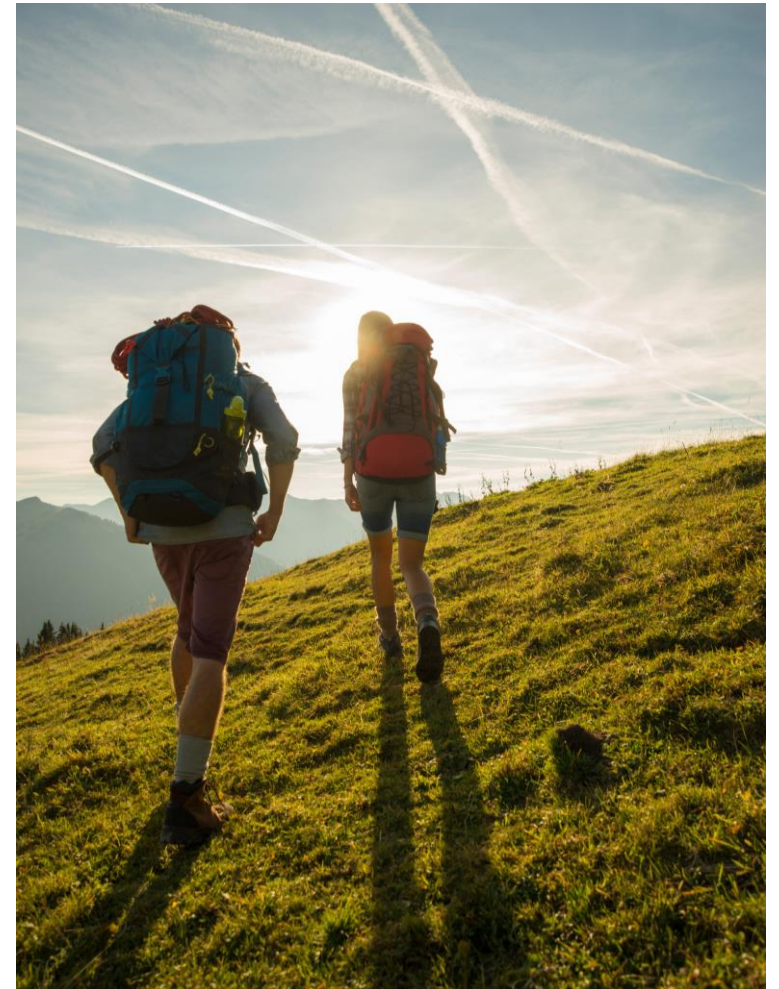
Logging and Tracing Using SLF4J

What you've learned in this unit

- Why you want to do logging
- What you should log and what not
- What SAP offers in terms of application logging
- What SLF4J is and how to use it
- How the SAP logging library supports you and how to make use of it
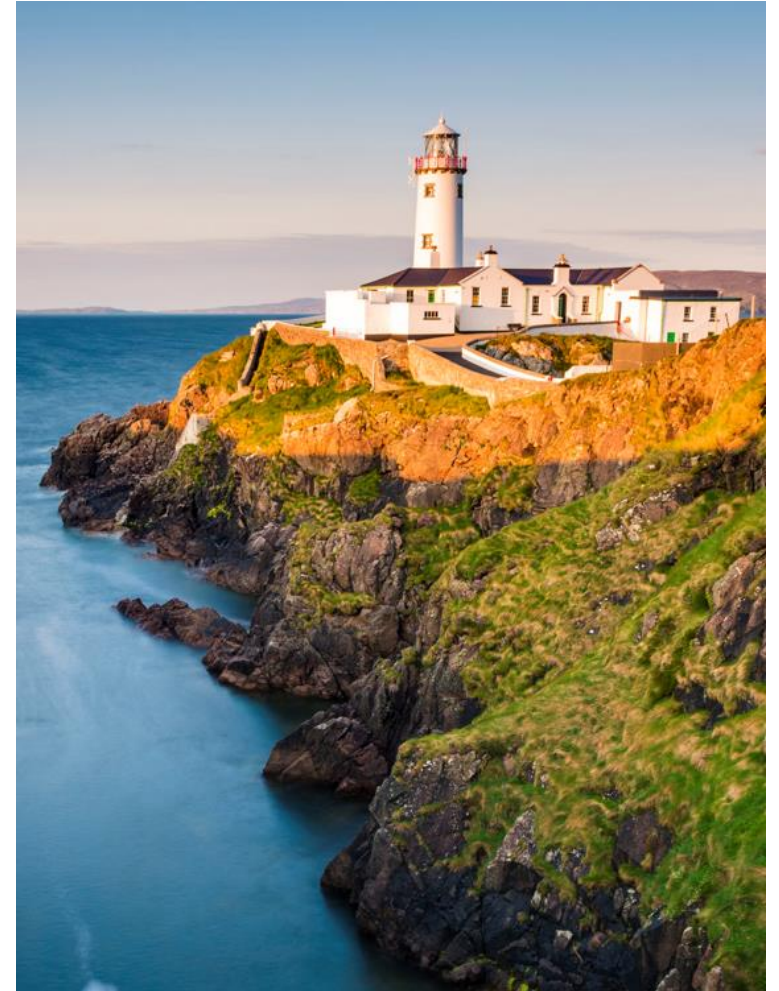
Thank you.

Contact information:

open@sap.com

© 2018 SAP SE or an SAP affiliate company. All rights reserved.