Practical Cloud-native Java with Eclipse MicroProfile
Background to Cloud-native
Agile
Agile & DevOps
Agile & DevOps & Cloud
Agile & DevOps & Cloud & Microservices
Good characteristics of a cloud-native environment
Provides APIs for distributed computing
Starts fast and shuts down clean
Has a (proportionately) small footprint
Facilitates **dev/prod parity**, including through externalized config
Can be easily containerized
Being **fast, small and open**
Liberty Features

server

<featureManager>
  <feature>jaxrs-2.0</feature>
  <feature>openapi-3.0</feature>
</featureManager>

build

<packaging.type>minify, runnable</packaging.type>
Wildfly Swarm Thorntail Fractions

```xml
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>jaxrs</artifactId>
</dependency>
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>swagger</artifactId>
</dependency>
```
What does it mean to provide microservice technologies?
Eclipse MicroProfile

Optimizing Enterprise Java for a microservices architecture
There’s a good chance you’ll use REST APIs.
Eclipse MicroProfile

- Rest Client 1.1
- CDI 2.0
- JSON-P 1.1
- JSON-B 1.0
- JAX-RS 2.1
Demo of REST support
Demo of REST support

Service A
/props/{propName}

Service C
/props/{propName}
Handling “100s” of collaborating services requires a strong operations focus
Eclipse MicroProfile

- Open API 1.0
- Open Tracing 1.1
- Health Check 1.0
- Metrics 1.1
- Rest Client 1.1
- CDI 2.0
- JSON-P 1.1
- JSON-B 1.0
- JAX-RS 2.1
Demo openapi, health & metrics
Handling "100s" of collaborating and frequently evolving services requires new APIs
Demo of config and Fault Tolerance
Fault Tolerance in microservices

Circuit breaker

```java
@CircuitBreaker(
    failOn=IOException.class,
    delay = 500)
public void callServiceC() {
    // call the service
}
```
Fault Tolerance in microservices

```java
public void callServiceC() {
    // call the service
}
```

**Bulkhead**

```java
@Asynchronous
@Bulkhead(value = 10,
    waitingTaskQueue = 15)
public void callServiceC() {
    // call the service
}
```
Can’t I do all this with a service mesh?
Fault Tolerance in microservices

New capabilities for microservices

App Container can defer to Cloud Platform

@Retry
@Timeout
@CircuitBreaker
@Bulkhead

Application still provides:
@Fallback
What is next?
• Reactive
• Data access
• Istio integration
• Updates to existing specs
Packaging for deployment
Making the most of Docker

- Application
- App Server
- JVM
- O/S

thin war

fat jar
Summary

• Consider both organizational and technological changes to increase likelihood of success
• Leverage MicroProfile to solve cloud-native challenges
• Choose appropriate packaging for your cloud
• Reduce overheads and cost with a right-sizeable runtime
• In Docker, strive for a thin application layer
• Learn more with Open Liberty guides https://ibm.biz/mpGuides