JAVAPRO

MicroStream

Develop modern cloud-native microservices with Payara Micro

Rudy De Busscher


1

Attendees

Attendees

- Who (can be very very brief)
- Why this course, what do yo want to learn
- Knowledge
  - Cloud, MicroServices, …
  - Payara (Micro)
  - Jakarta EE
  - MicroProfile
  - Java

Recorded - Links will be emailed

Develop modern cloud-native microservices with Payara Micro

Rudy De Busscher

Topics

- Architecture
  - cloud-native microservices
- Runtime
  - Payara Micro
- Framework
  - Jakarta EE
  - MicroProfile
  - MicroStream

Agenda

- Day 1
  - Introduction
  - Cloud Native
  - Payara - Payara Micro intro
  - Jakarta EE Overview
  - CDI
  - Fat Jar - Hollow Jar
  - JAX-RS
  - MicroProfile Overview
  - MicroProfile Config
  - Hands-on exercise

Agenda

- Day 2
  - Review Hands-on exercise
  - Microservices
    - Java In the cloud
  - MicroProfile Rest Client
  - Other MicroProfile specifications
  - Data access
    - JPA
    - MicroStream
  - Integration testing
  - Hands-on exercise

System Requirements

- Java
  - Version 11
  - `java --version`
- Maven
  - Recent
  - `mvn -version`
- Docker Desktop or Docker client
  - Optional
  - For integration testing

Cloud Native

- What is it?

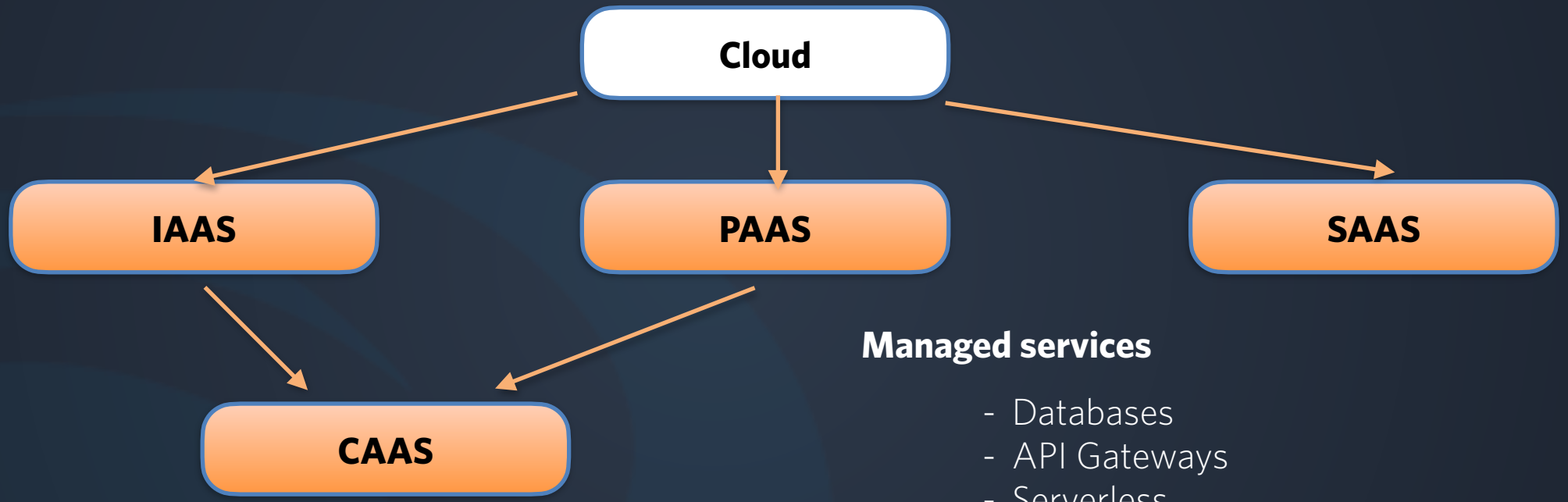
- Cloud
- Cloud-Native

On-premise to Cloud

- It allows organizations to scale, maintain flexibility, and focus their efforts on business operations

- Flexibility (Easier to use new servers)
- Optimal Resource usage
  - Cost efficient
- Automation
- Cloud Platform features
- More secure (should not be the case 😀)

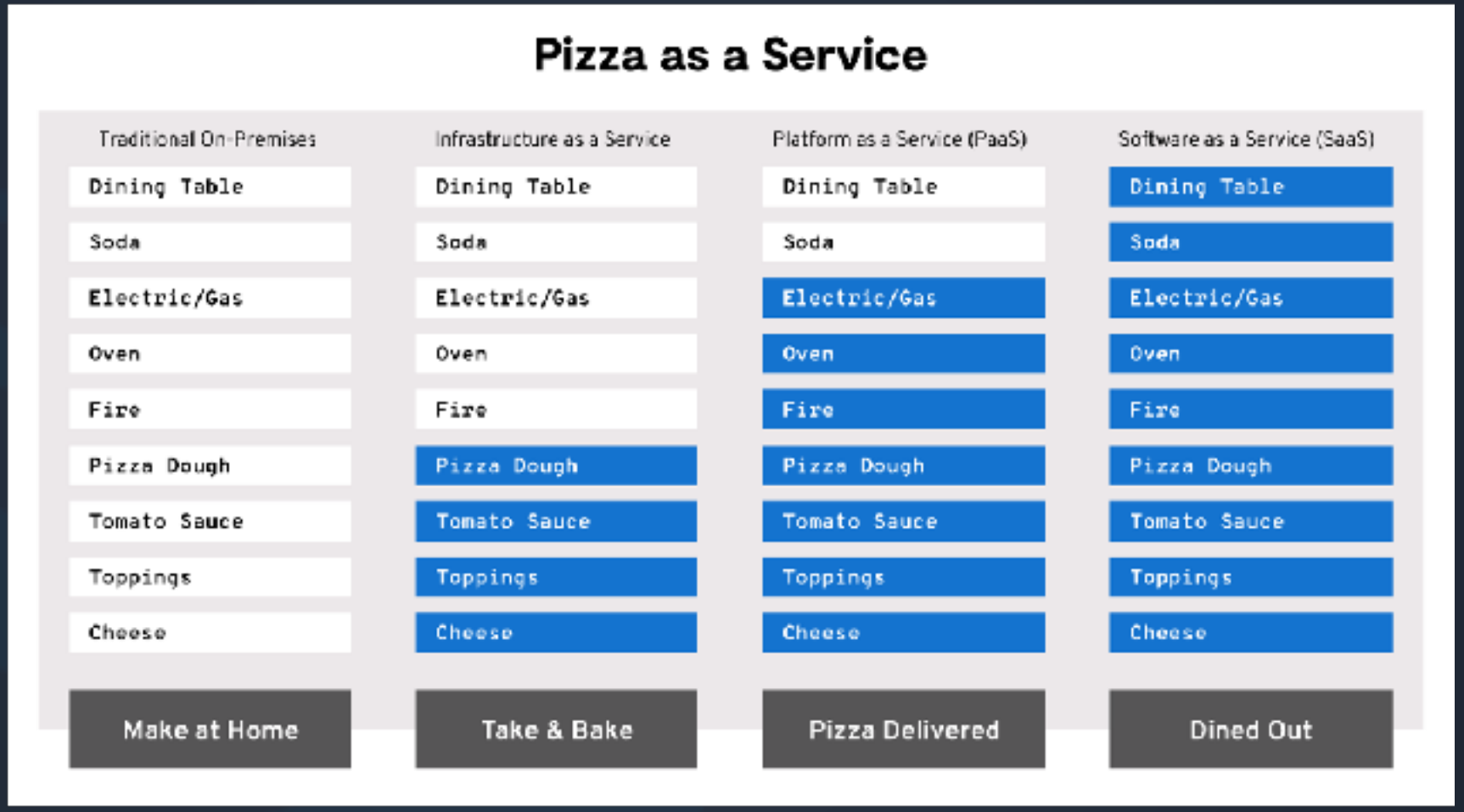
Cloud Models



Managed services

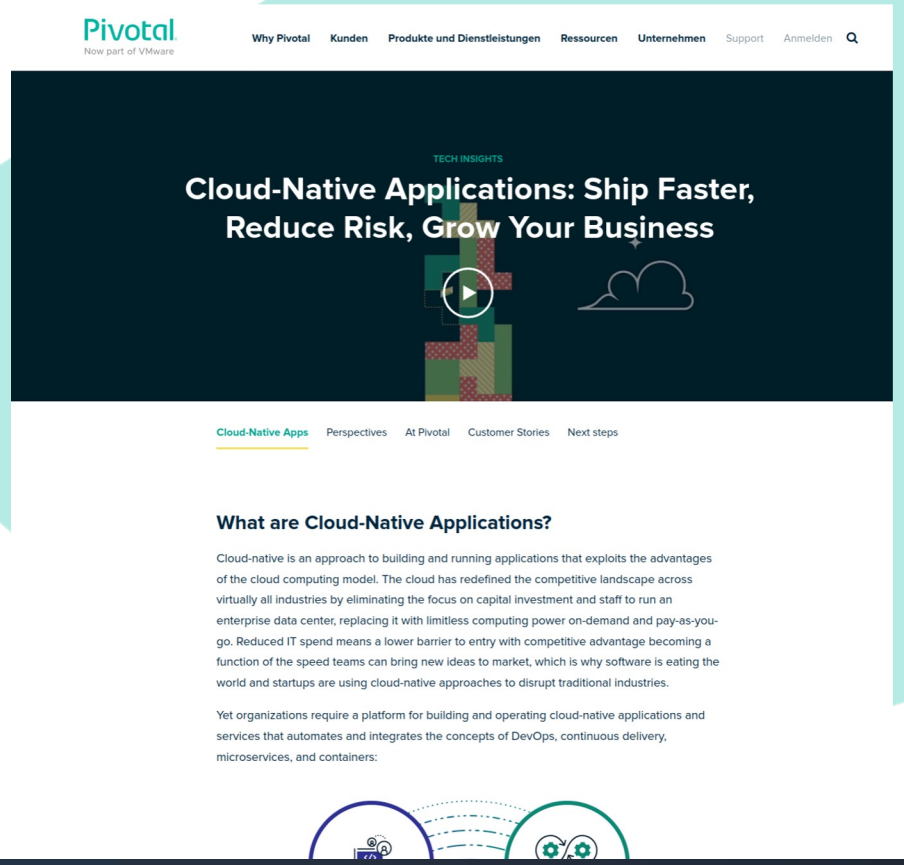
- Databases
- API Gateways
- Serverless

Cloud Models



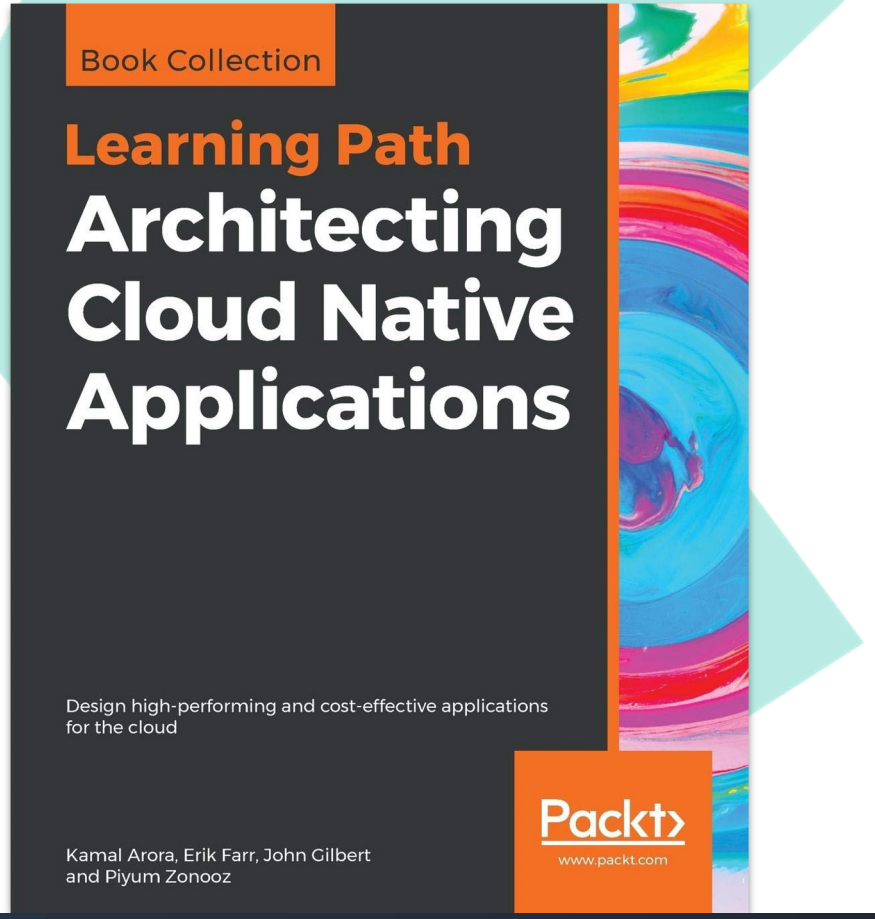
Cloud Native

"Cloud-native is an approach to building and running applications that exploits the advantages of the cloud computing model."



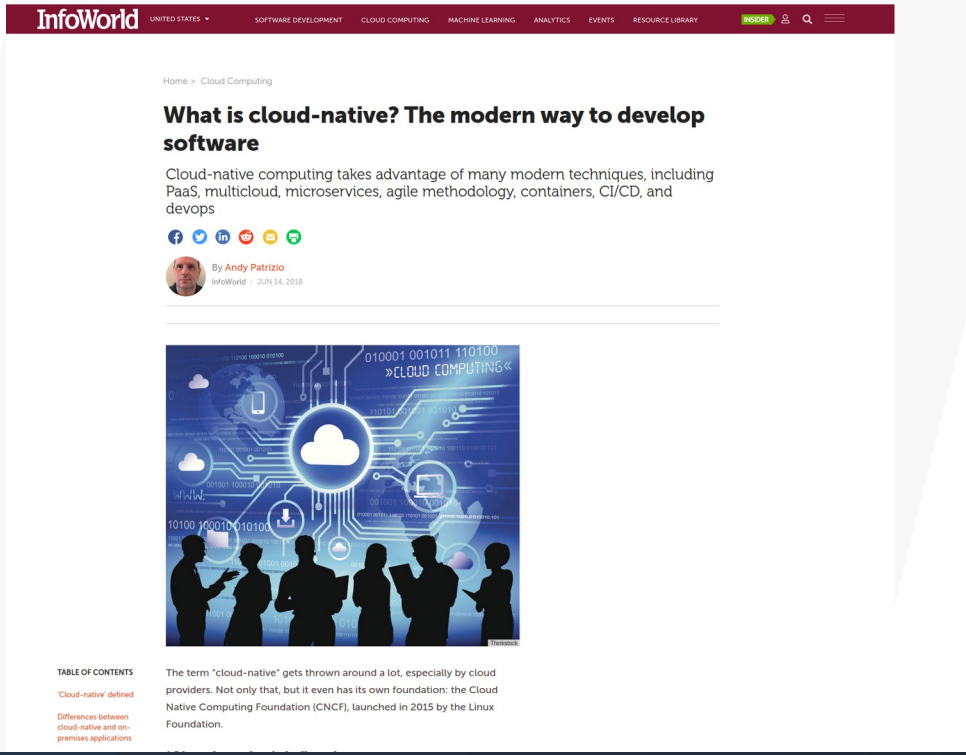
Cloud Native

“Cloud-native is a different way of thinking and reasoning about software systems. It embodies the following concepts: powered by disposable infrastructure, composed of bounded, scales globally, embraces disposable architecture.”



Cloud Native

“In general use, ‘cloud-native’ is an approach to building and running applications that exploits the advantages of the cloud-computing delivery model. ‘Cloud-native’ is about how applications are created and deployed, not where.”



InfoWorld UNITED STATES SOFTWARE DEVELOPMENT CLOUD COMPUTING MACHINE LEARNING ANALYTICS EVENTS RESOURCE LIBRARY INSIDER

Home > Cloud Computing

What is cloud-native? The modern way to develop software

Cloud-native computing takes advantage of many modern techniques, including PaaS, multicloud, microservices, agile methodology, containers, CI/CD, and devops

By Andy Patrizio
InfoWorld | JUN 14, 2018

TABLE OF CONTENTS

‘Cloud-native’ defined

Differences between cloud-native and on-premises applications

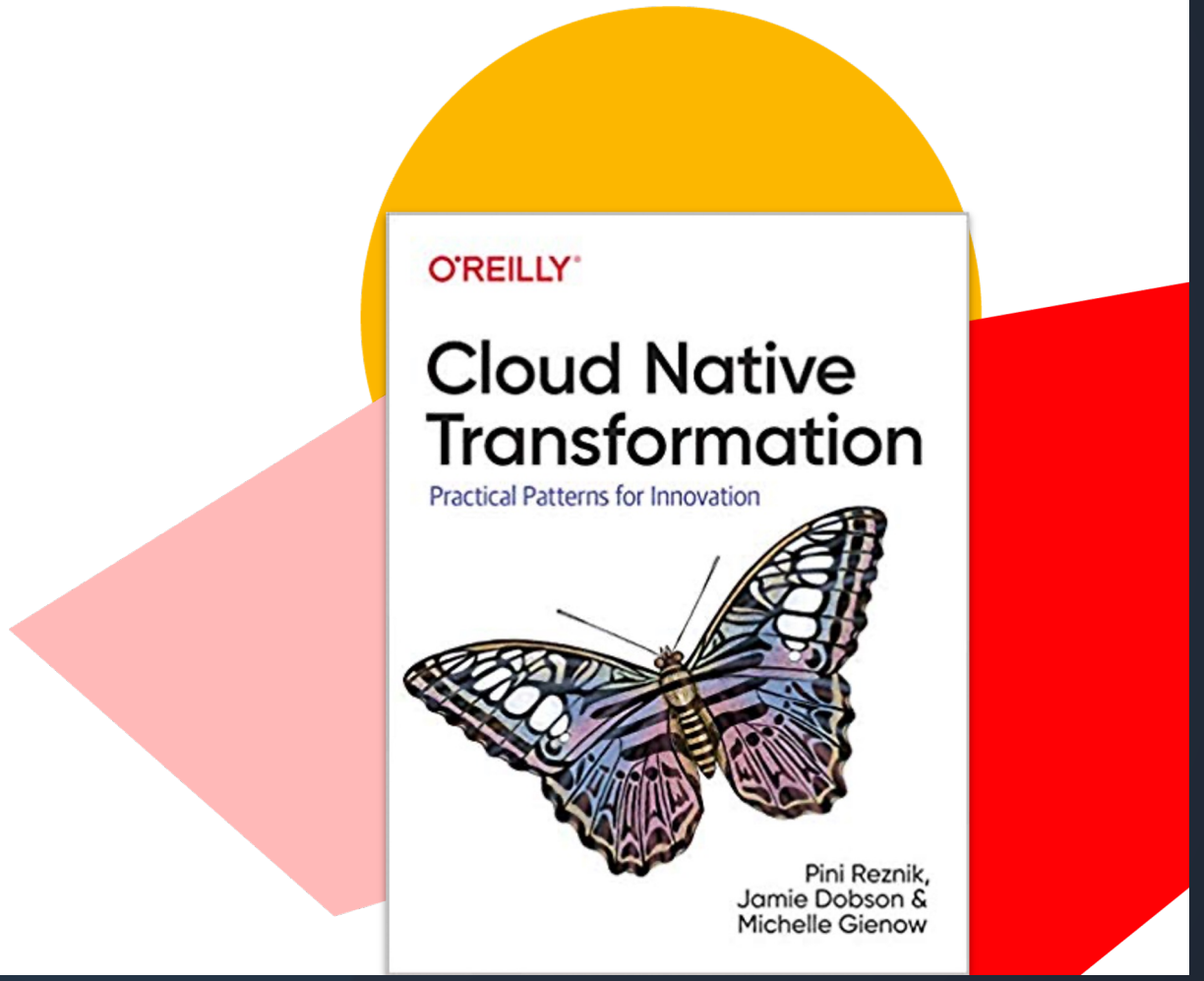
The term “cloud-native” gets thrown around a lot, especially by cloud providers. Not only that, but it even has its own foundation: the Cloud Native Computing Foundation (CNCF), launched in 2015 by the Linux Foundation.

Cloud Native

“Independent DURS ultimately comes up in every discussion on cloud-native concepts; to independently Deploy, Update, Replace and Scale.”

Cloud Native

“Cloud-native is more than a tool set. It is a complete architecture, a philosophical approach for building applications that take full advantage of cloud computing.”



Cloud Native

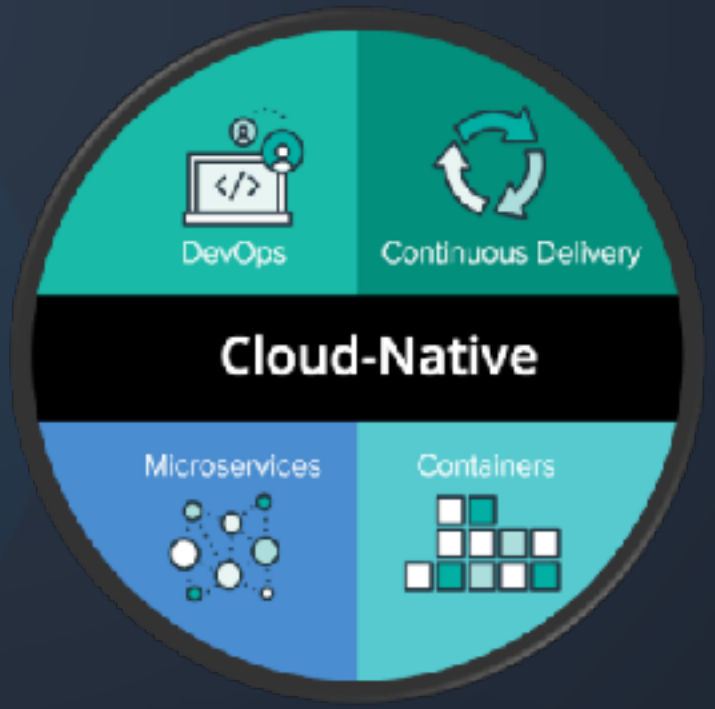
“Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.”

CLOUD NATIVE
COMPUTING FOUNDATION

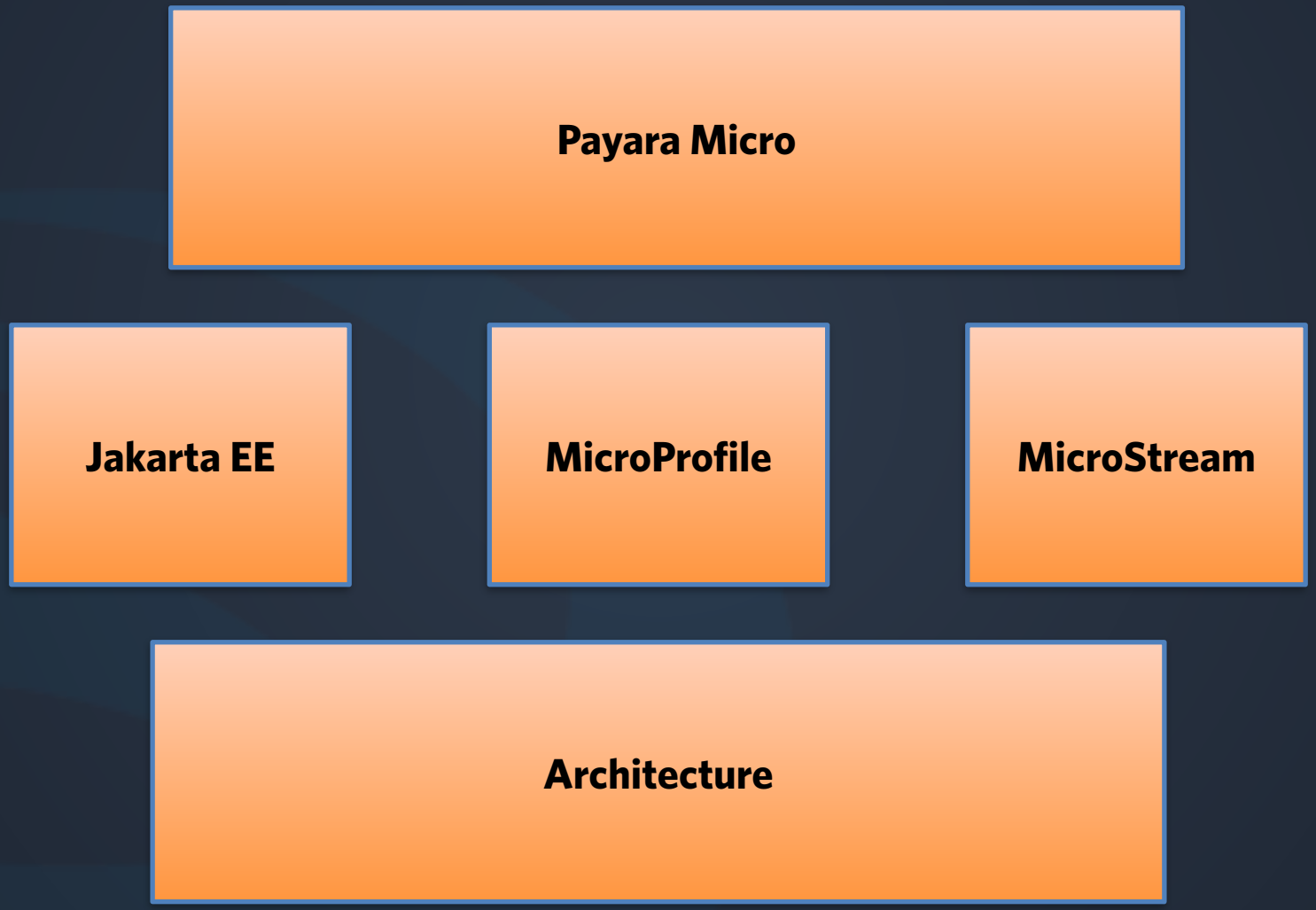
Cloud-Native

- Vague

- A set of best practices to optimize an application in the cloud through:
  - Containerization
  - Orchestration
  - Automation



Our Focus


Payara Micro
Jakarta EE
MicroProfile
MicroStream
Architecture

Jakarta EE

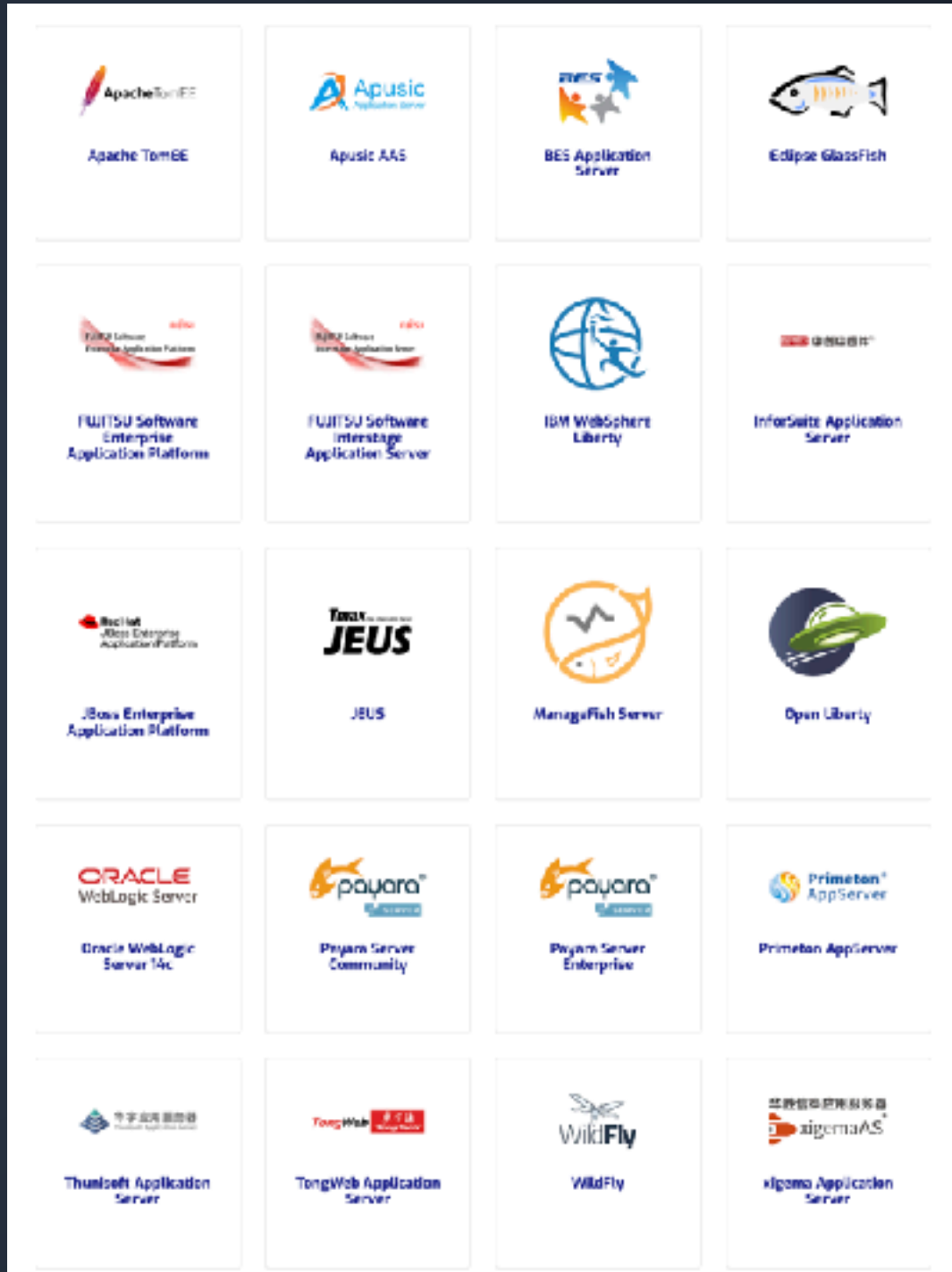
- Is a set of *specifications*, extending Java SE with specifications for enterprise features such as *distributed computing* and *web services*.
- Started by Sun in Dec 1999 (J2EE 1.2)
- As Java EE donated to Eclipse Foundation in September 2017

- Comparable to Spring (Boot)
  - Single vendor

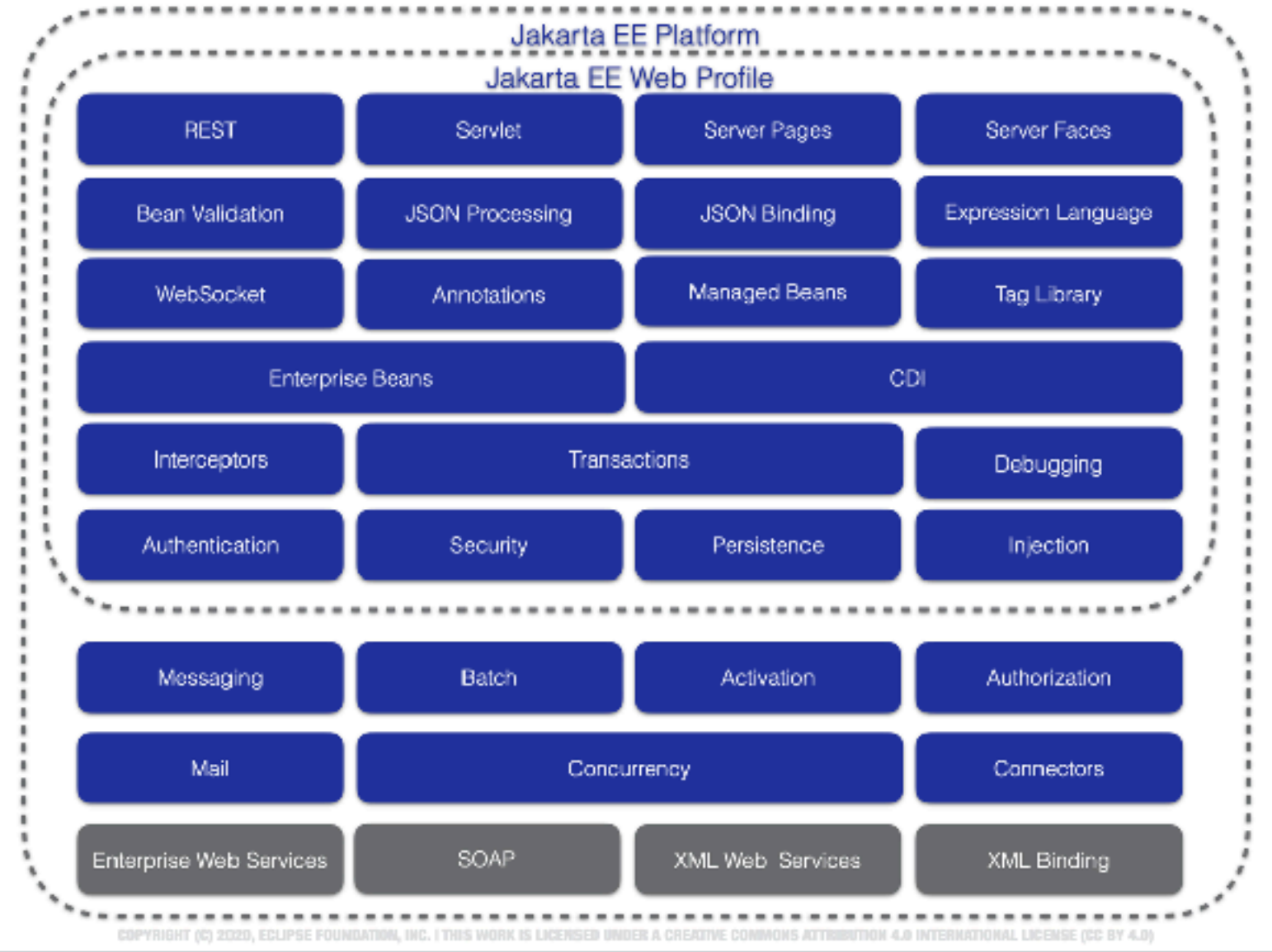
Jakarta timeline

Jakarta EE Specification

- Specification =
  - Document describing functionality
  - API (Java classes)
  - TCK Test Compatibility Kit



Jakarta EE Specifications



Payara

- Jakarta EE & MicroProfile compatible, supported application servers for production and containerized deployments.
- Payara Server
  - Typical Application Server
  - Full Profile
  - Install, Configure, Run
- Payara Micro
  - Application runtime
  - Web Profile
  - Designed for containerised environments

Payara Get Started

- Download Payara Micro
  - https://www.payara.fish/downloads/payara-platform-community-edition/
  - Search for “Payara Download”. And click on community downloads
- `java -jar payara-micro.jar`
  - Renamed for easier typing

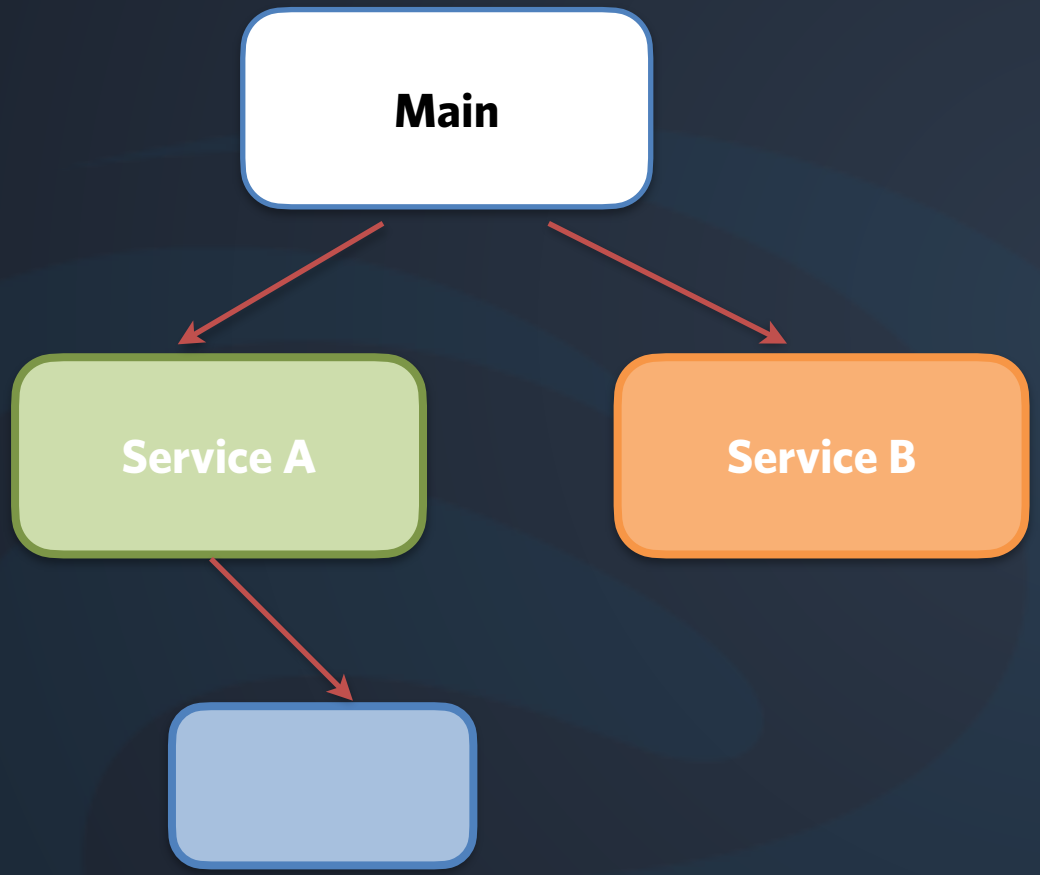
Why Payara?

- Jakarta / MicroProfile implementation
- Cloud / Container features
- Clustering capabilities
- Good support

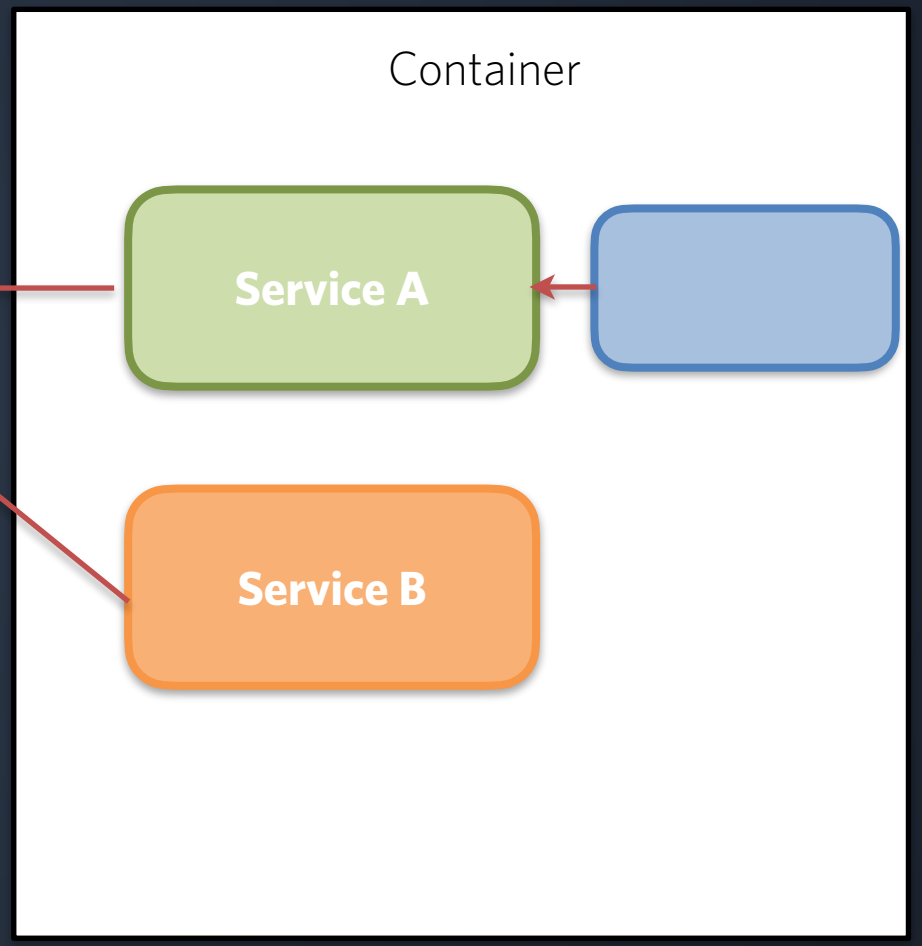
Context and Dependency Injection

- Having well-defined lifecycle for stateful objects, beans
- Bound objects to lifecycle contexts = different lifecycles
- Creating decoupled and flexible code
  - Dependencies are provided by container
- Decorating injected objects
  - Interceptors
- Using events and notifications

Inversion of Control



Classic



Demo

CDI Demo

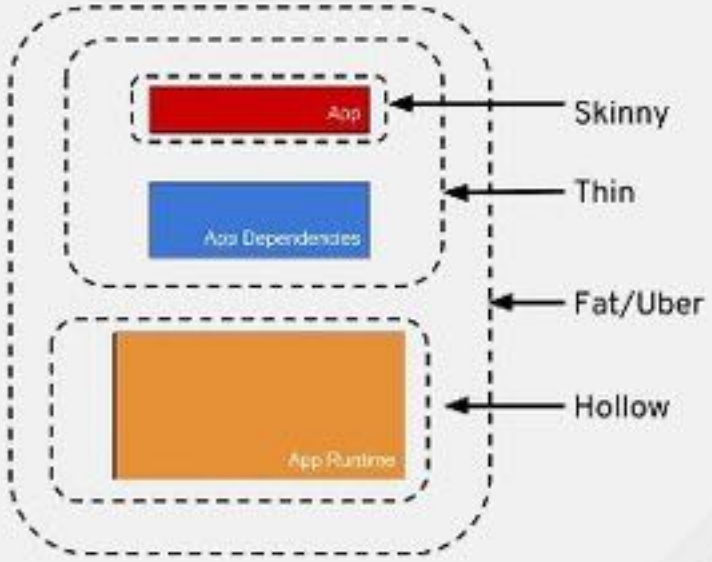
- Using Java SE (works the same in Payara Micro)
  - Define CDI beans
  - Injection of dependencies
  - Scope
  - Qualifier
  - Optional
  - Producer
  - Event
  - Decorator
  - Interceptor

Payara Micro - Different options

- Fat jar vs Hollow jar

Fat, Thin, Skinny, Hollow

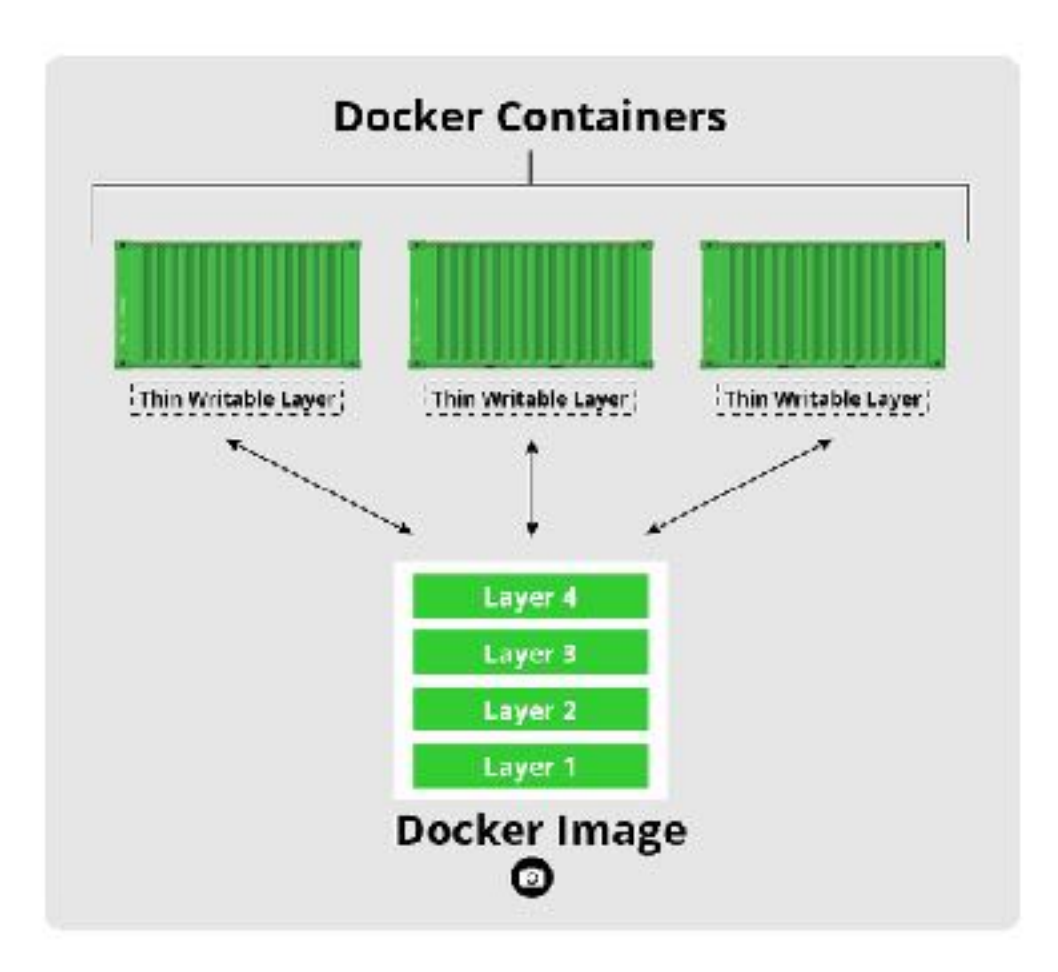
- Thin
  - App + Direct Dependencies (ex: traditional Java EE .WAR files)
- Skinny
  - App only, dependencies satisfied externally
  - Must be "deployed" to hollow app
- Hollow
  - App runtime only, contains app dependencies



developers.redhat.com/blog/2017/08/24/the-skinny-on-fat-thin-hollow-and-uber/

Fat vs Hollow

	Fat	Hollow
Pro	Easy (all-in-one)	Ideal for Containers
Contra	Redundancy in built Less suited for Containers	Separate runtime

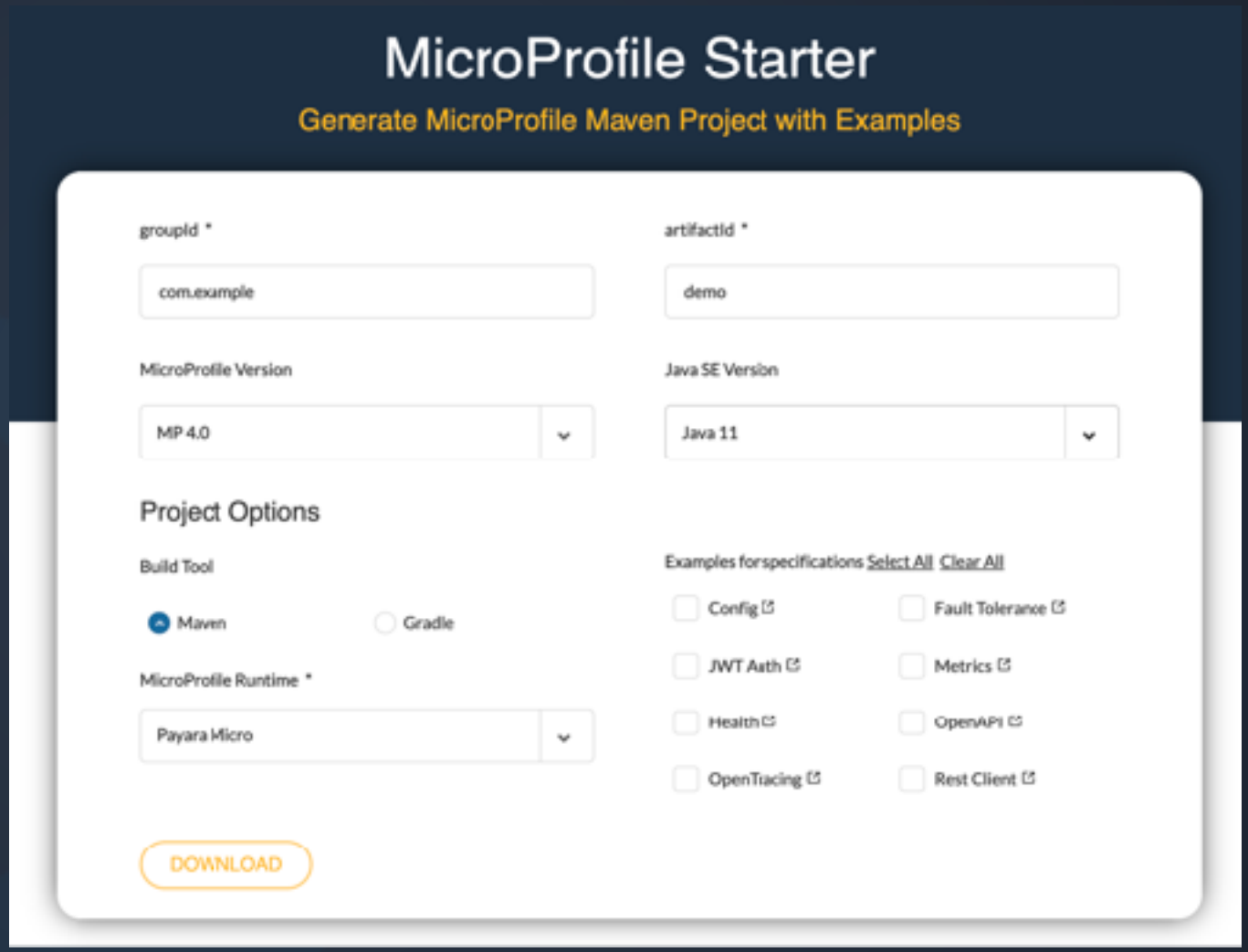



```
practical-devsecops@docker-Lab:~$ docker build -t mynmap .
Sending build context to Docker daemon  80.56MB
Step 1/5 : FROM alpine:latest
 ---> 9617b9b7b411
Step 2/5 : RUN apk update
 ---> Using cache
 ---> 8d85a77f247e
Step 3/5 : RUN apk ad
 ---> Using cache
 ---> b95843fff997
Step 4/5 : ENTRYPOIN
 ---> Using cache
 ---> a511166c832a
Step 5/5 : CMD ["-h", "127.0.0.1"]
 ---> Running in 56adc0d5506e
Removing intermediate container 56adc0d5506e
 ---> 0abe23abfb5a
Successfully built 8abe23abfb5a
Successfully tagged mynmap:latest
practical-devsecops@docker-Lab:~$
```

Using cache and reusing the existing layer

Fat vs Hollow : Example with Payara Micro

- MicroProfile Starter
- start.microprofile.io

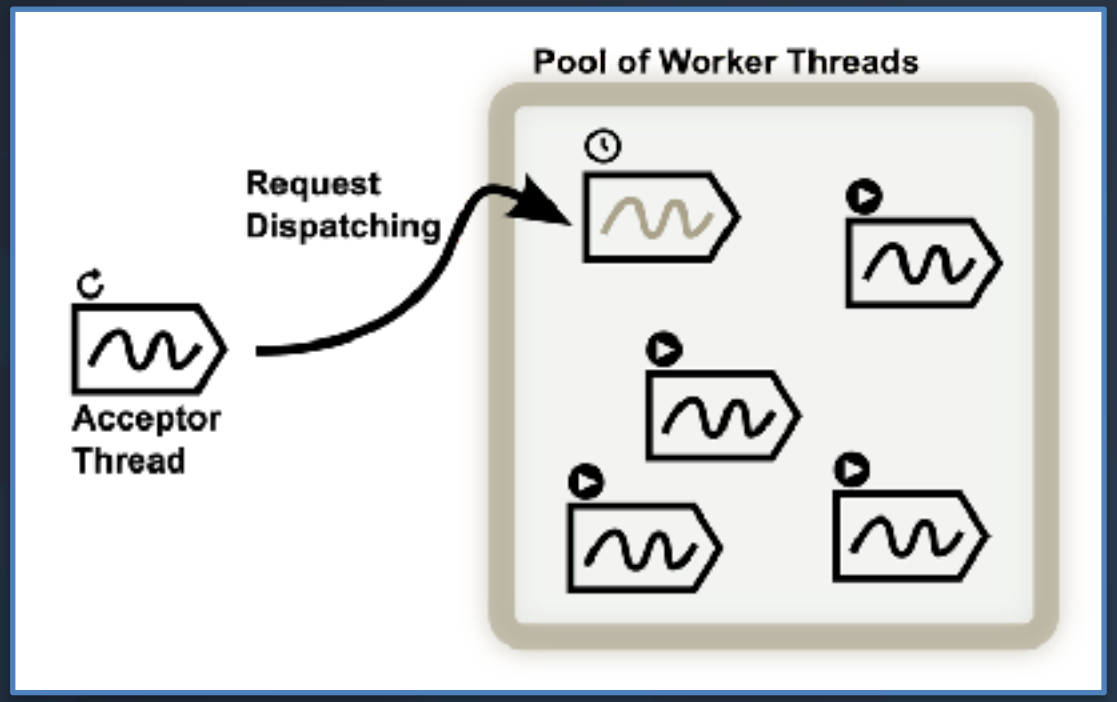


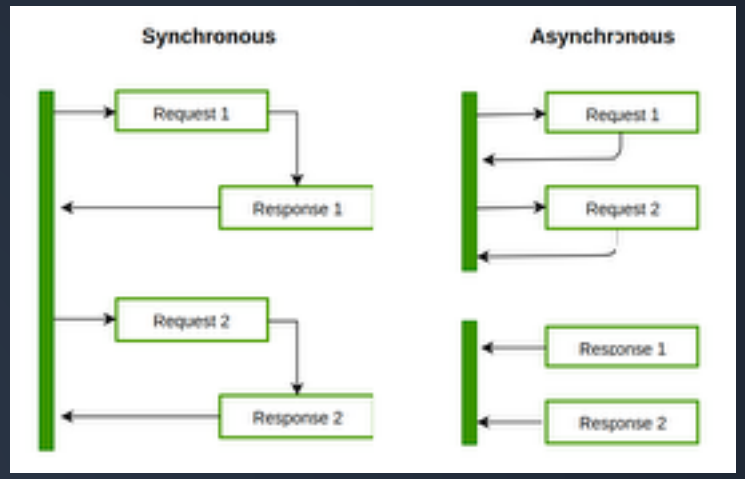
JAX-RS

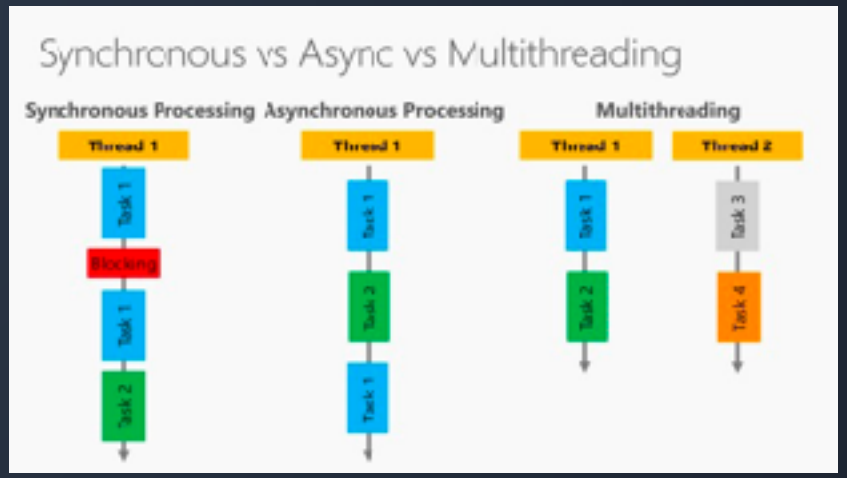
- Specification that provides support in creating web services according to the Representational State Transfer architectural pattern. (REST)
- Stateless
- Communication between client and server
- Based on HTTP methods.
- Content type not fixed
  - Based on request information
  - JSON is common

Demo

Processing

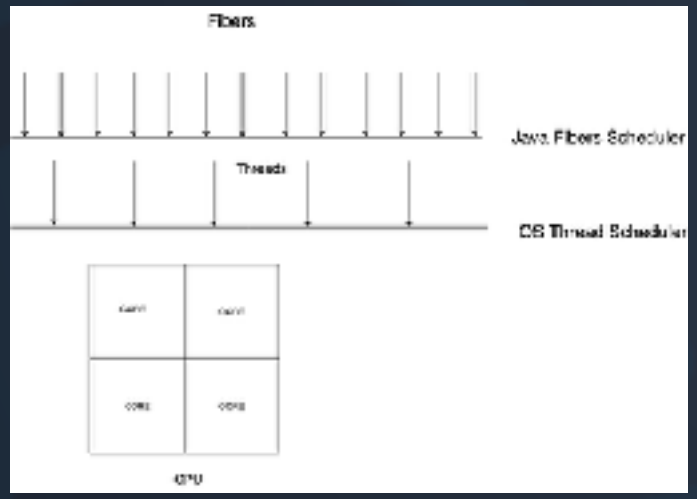

Pool of Worker Threads
Request
Dispatching
Acceptor
Thread

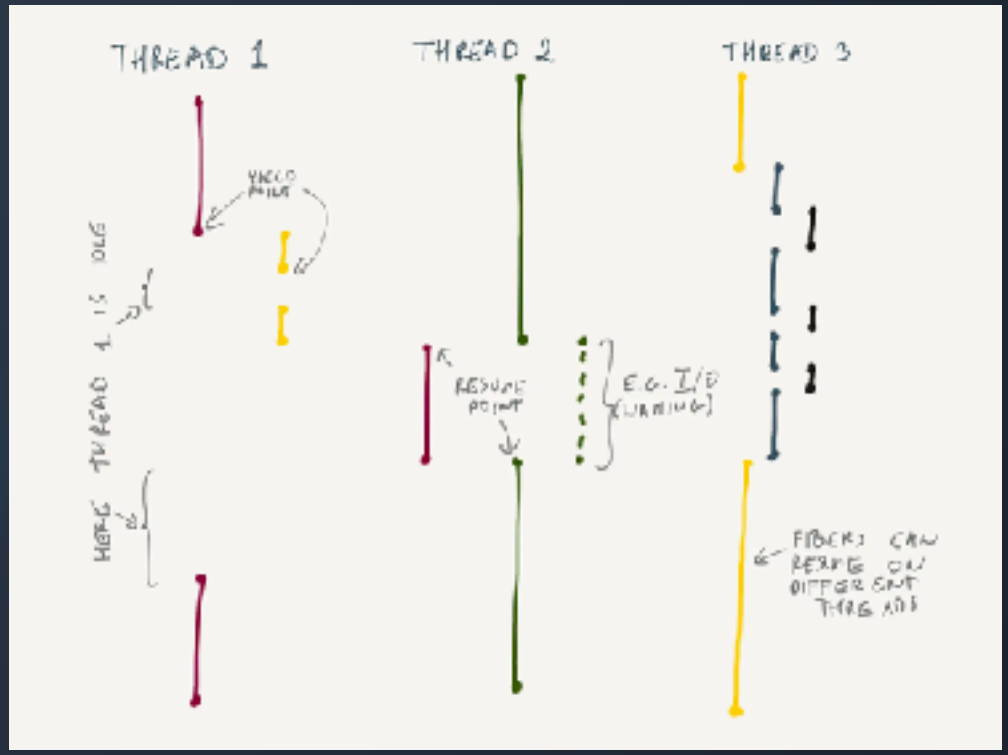

Synchronous
Asynchronous
Request 1
Response 1
Request 2
Response 2
Request 1
Request 2
Response 1
Response 2


Synchronous vs Async vs Multithreading
Synchronous Processing
Asynchronous Processing
Multithreading
Thread 1
Thread 1
Thread 1
Thread 2
Task 1
Blocking
Task 1
Task 2
Task 1
Task 2
Task 1
Task 3
Task 4

Project Loom

- JDK 19 preview
  - Not production ready
- Not used by Payara (yet)





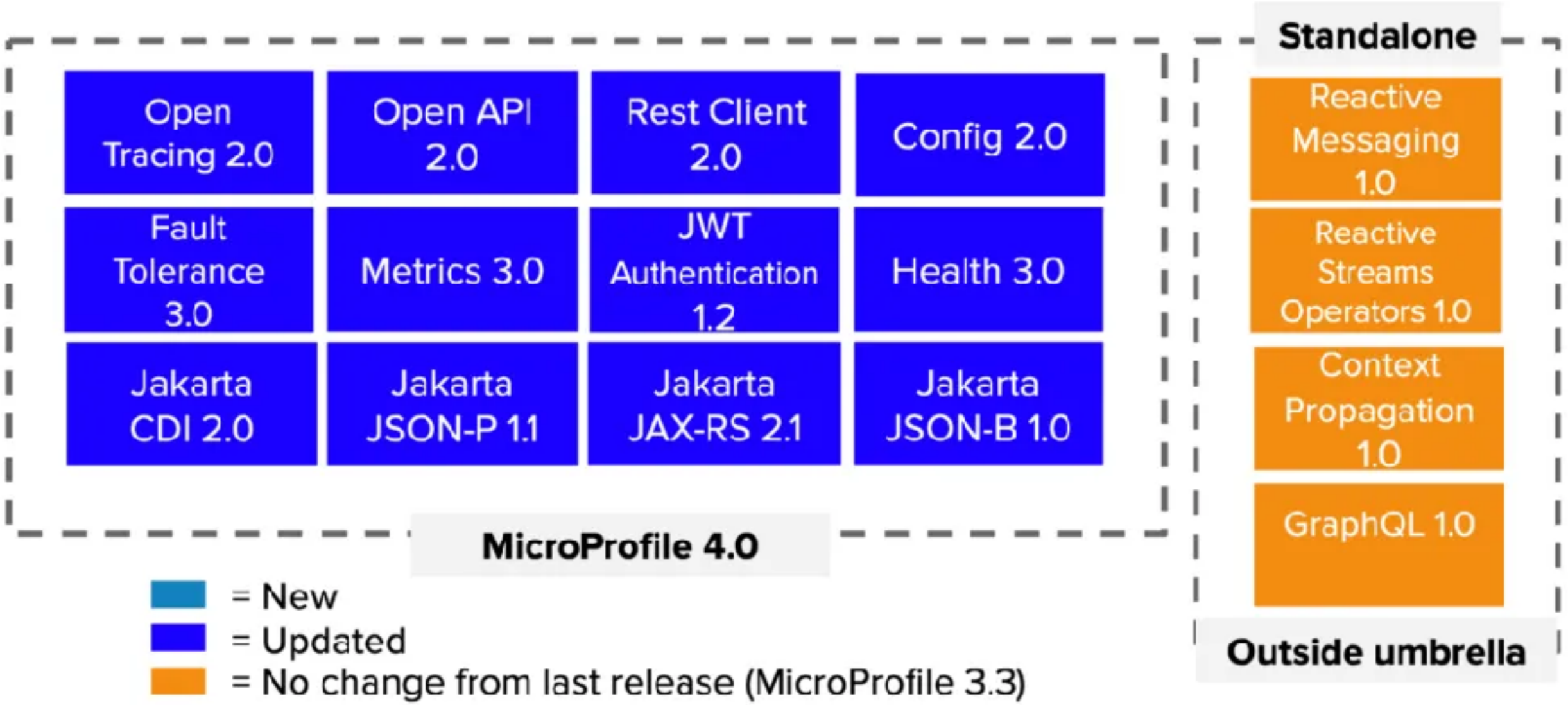
JAX-RS Demo

- Mapping endpoints
- JSON support
- Define response status
- Conversion
- Validation
- Access CDI
- Exception
- Filters

MicroProfile

- Optimizes the Enterprise Java for microservices architecture and delivers application portability across multiple MicroProfile runtimes.
- Jakarta EE not suitable?
- Missing some goodies
  - From 12 factor application methodology
  - Observability
  - Fault tolerance (distributed)

MicroProfile Specifications



MicroProfile Config

- From 12-factor App:
  - Store config in the environment
  - Single artefact for Dev and Production

- MP Config
  - Defines ConfigSources
  - Priorities
  - Conversion
  - Injection as CDI beans

Demo

MP Config demo

- Declarative and Programmatic Access
- Priority
- Optional values
- Conversion
- Variable replacement

- Caching with Payara Micro
- Custom Source

Configure Payara Micro

- No install, configure, run steps, just *execute*
- Command line parameters
  - Payara Micro Command Line Options (Documentation)
  - --port, --context root, --addlibs, --systemproperties, --deploy, --deploydir
  - --logtofile, --logproperties, --enablerequesttracing
  - --rootdir, --postbootcommandfile
  - --warmup, --outputlauncher, --outputuberjar

Post boot command file

- Asadmin command
  - Commands of the ASADMIn Payara Server tool
- Steps
  - Unpack JAR
  - Execute pre boot command file
  - Adapt JVM parameters, including Classpath and start runtime
  - Execute post boot command file
  - Deploy Application(s)
  - Execute post deploy command file
  - Instance is ready to server requests

Asadmin recorder of Payara Server

Challenge Day 1

Hands-on

- Jakarta Hands-on
- Endpoints for Web shop
- Light - Full (advanced) assignment
- https://github.com/rdebusscher/payara-micro-course/labs

JAVAPRO

MicroStream


2


Certificate

Agenda

- Day 2
  - Review Hands-on exercise
  - Microservices
    - Java In the cloud
  - MicroProfile Rest Client
  - Other MicroProfile specifications
  - Data access
    - JPA
    - MicroStream
  - Integration testing
  - Hands-on exercise


REVIEW

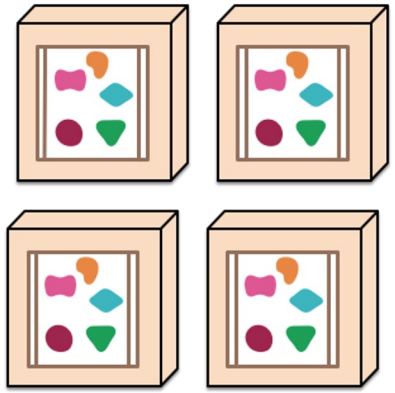
MicroServices

- Service focussed on a single business domain
  - Nothing to do with size.
- Connected with other services through REST
  - Ideally not RPC over HTTP
- Independently upgradeable and scalable
  - Backwards compatibility
  - Multiple versions
- Mainly for very large teams and applications

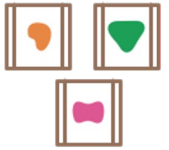
Monoliths and Microservices

A monolithic application puts all its functionality into a single process...

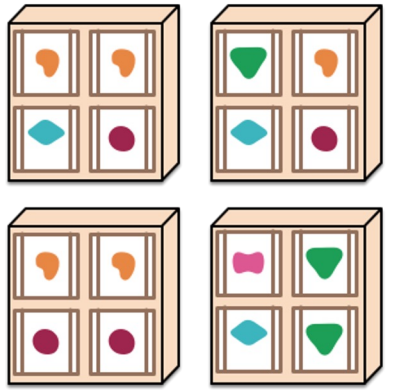
... and scales by replicating the monolith on multiple servers



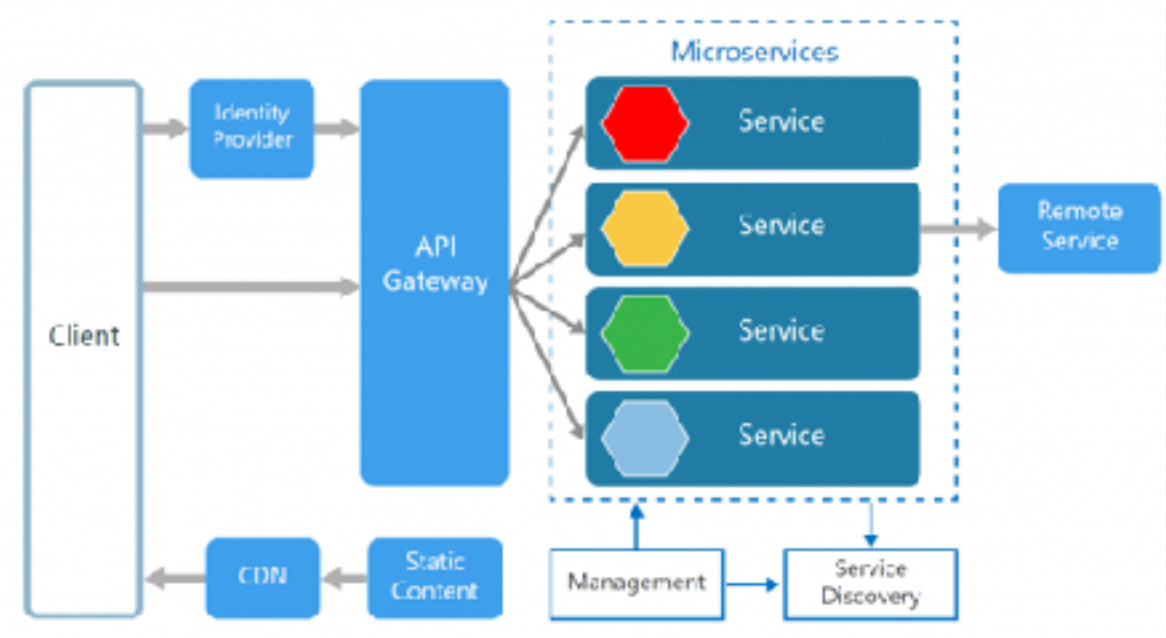
A microservices architecture puts each element of functionality into a separate service...

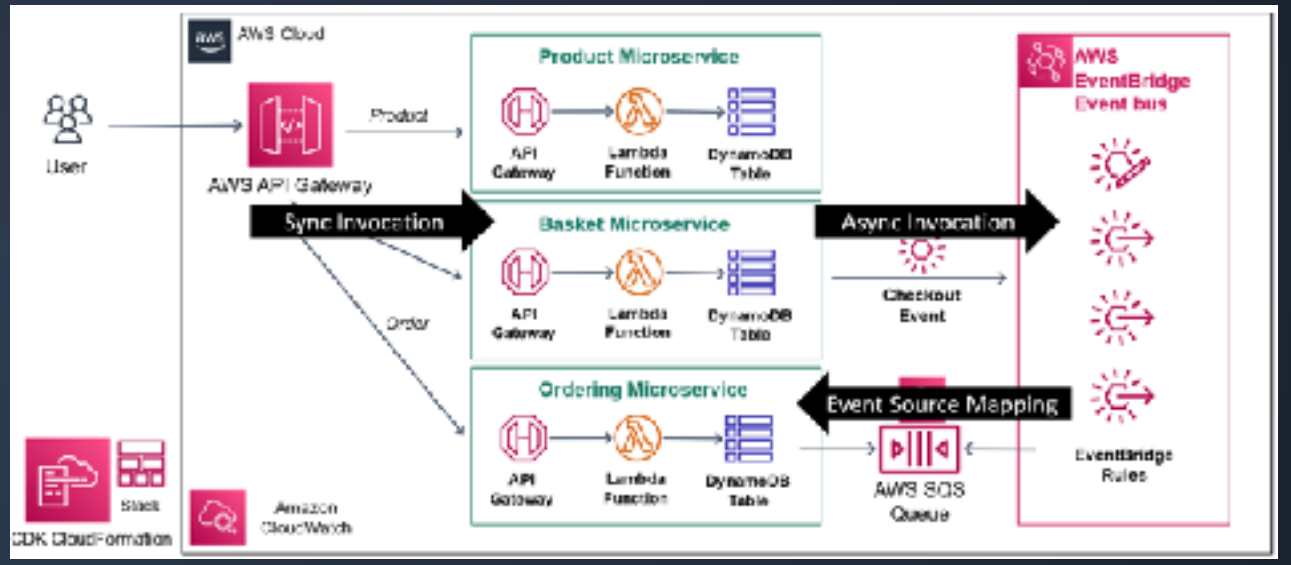


... and scales by distributing these services across servers, replicating as needed.



Source: https://martinfowler.com/articles/microservices.html


Microservices
Client
Identity Provider
API Gateway
Service
Service
Service
Service
Remote Service
CDN
Static Content
Management
Service Discovery


AWS Cloud
User
AWS API Gateway
Product
Sync Invocation
Order
Product Microservice
API Gateway
Lambda Function
DynamoDB Table
Basket Microservice
Async Invocation
Checkout Event
Ordering Microservice
Event Source Mapping
AWS SQS Queue
AWS EventBridge Event bus
EventBridge Rules
Amazon CloudWatch
CDK CloudFormation

Java too heavy for cloud?

- AWS Lambda
  - CPU is related to amount of memory selected
  - Very low memory -> very low CPU
  - Need sufficient CPU for processing -> more memory
  - Memory is more than enough for JVM
- Comparison with JavaScript (NodeJS)
  - Comparable performance.

- Class Data Sharing
  - Prepare Class info in format info JVM format (Read-only memory mapped files)

MicroProfile Rest Client

- Type-safe approach to invoke RESTful services over HTTP.

- Remote call looks like JVM method call.
- Heavy lifting behind the scenes.

Demo

MP Rest Client Demo

- Define Remote endpoint contract
- Call other service
- Make asynchronous call

- Asynchronous JAX-RS endpoint (*go back to JAX-RS demo*)
  - Server side

MicroProfile OpenAPI

- API description format for REST APIs
- Swagger

- Documentation
- Client generation
  - Code Generation project supports MP Rest Client.

- Customise with @Operation, @APIResponse, @Schema, …

MP Fault Tolerance

- Microservices are a distributed system with all their problems (7 fallacies of distributed computing)
- Set of annotations to make call more robust

MP Fault Tolerance Components

- Timeout: Define a duration for timeout.
- Retry: Define a criteria on when to retry.
- Fallback: provide an alternative solution for a failed execution.
- Bulkhead: isolate failures in part of the system while the rest part of the system can still function.
- CircuitBreaker: offer a way of fail fast by automatically failing execution to prevent the system overloading and indefinite wait or timeout by the clients.

MP Health and Metrics

- Observability measures the internal states of a system by examining its outputs. It allows you to see what your system is doing, if it is running fine or needs to be restarted.

- Metrics
  - Provides the (raw) data
- Health
  - Interprets the data

MicroProfile JWT

- Secure the endpoints by passing tokens (JSON Web Tokens) that contain the required information.

- Build on top of JOSE specification (Signed and encrypted tokens)
- Adaption for passing on groups / roles information
- In some areas to restrictive
  - Does not allow some parts or combinations of the JOSE specification o be used
  - Might be that only useable within MP itself (not for tokens from other frameworks and languages)

Data Access

- Classic
  - Database
  - No-SQL solution
- MicroStream
  - JVM memory as database

JPA

- The Java ORM standard for storing, accessing, and managing Java objects in a relational database
- On top of JDBC to avoid boilerplate, low level access and dependencies to product, etc ..

- Sometimes challenging to retrieve the data with acceptable performance (N+1 query problem, mapping, caching, …)
-

JPA requirements

- The ORM tool itself (obviously)
- A JDBC driver supporting the database
- Configuration to access database
  - Database server location
  - Credentials
  - Configuration options.

Demo

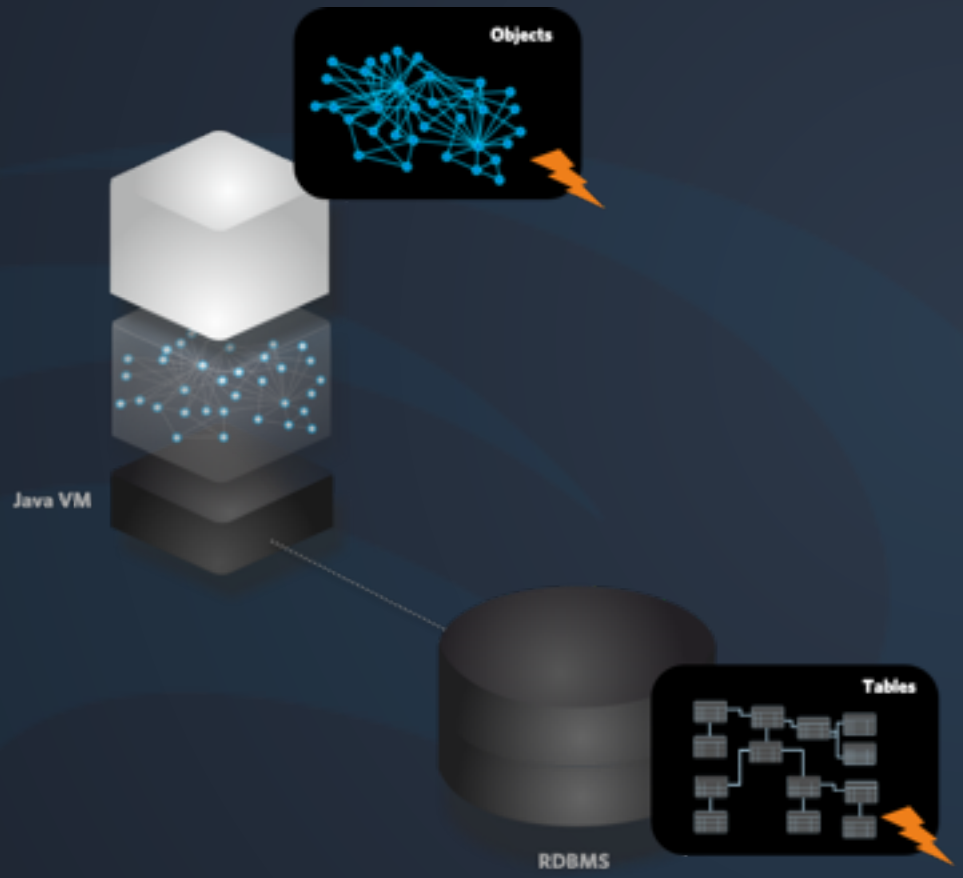
JPA Demo

- Define Datasource with @DataSourceDefinition
- Define configuration properties (through MicroProfile)
- Supply JDBC driver to Payara Micro
- Basic functionality of JPA

MicroStream

- The Java objects are the database
- Reading is just accessing getters or any other object method
- Data is persisted
  - Storage manager
  - Survive process restart
-

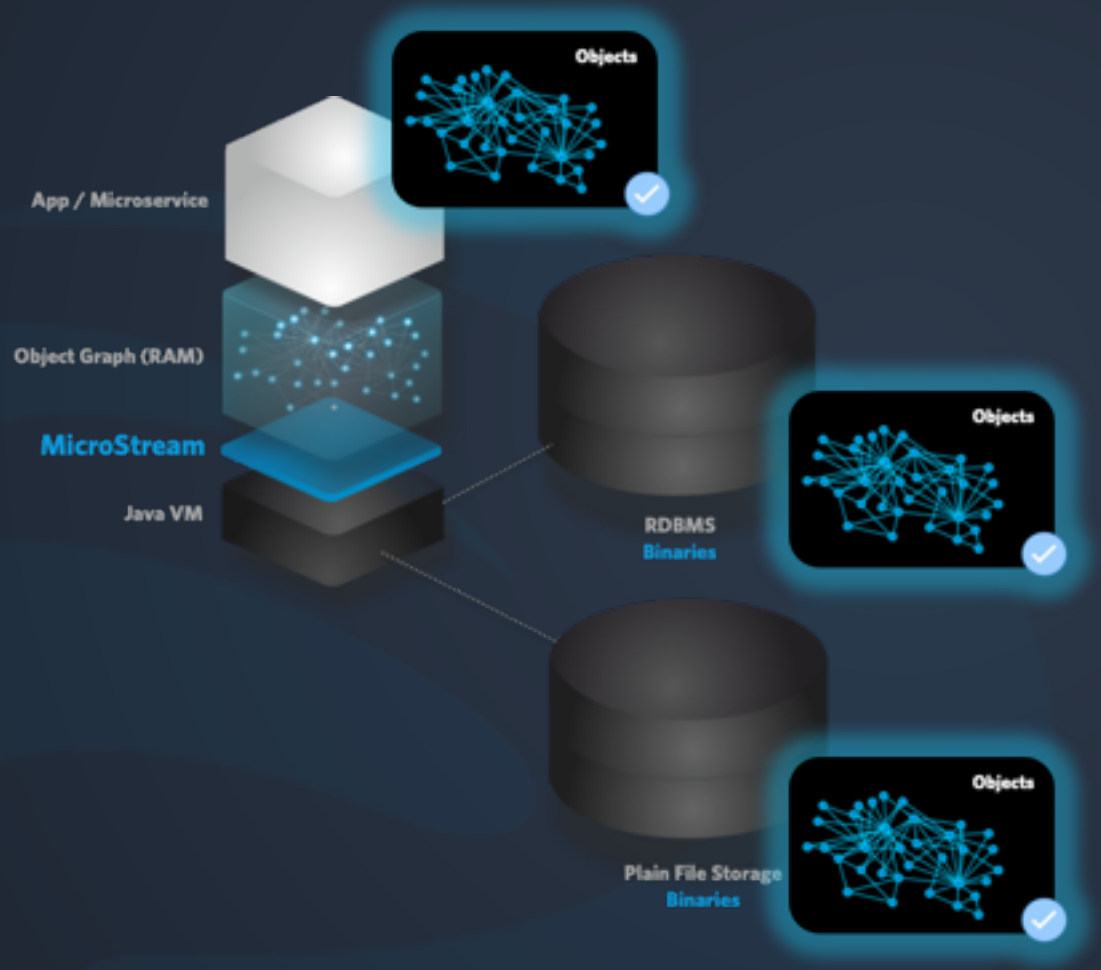
Traditional Java Persistence



Impedance Mismatch

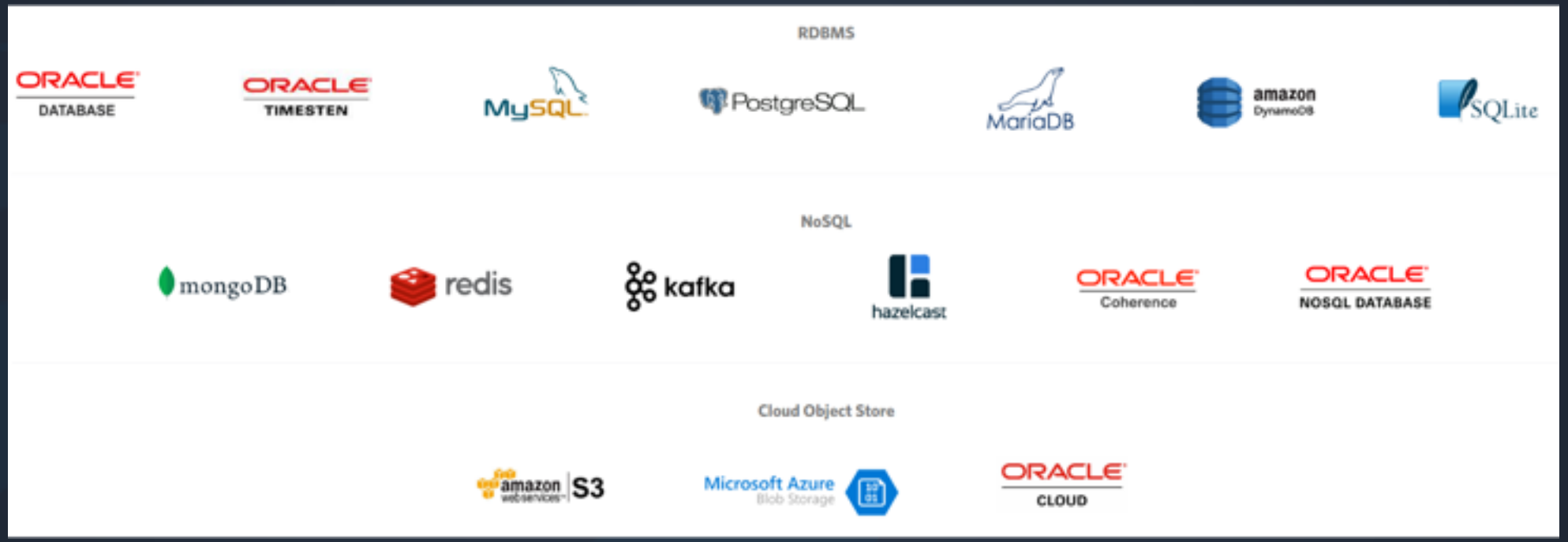
- Granularity mismatch
- Subtypes mismatch
- Identity mismatach
- Associations mismatch
- Data Navigation mismatch
- Data type differences

MicroStream Persistence

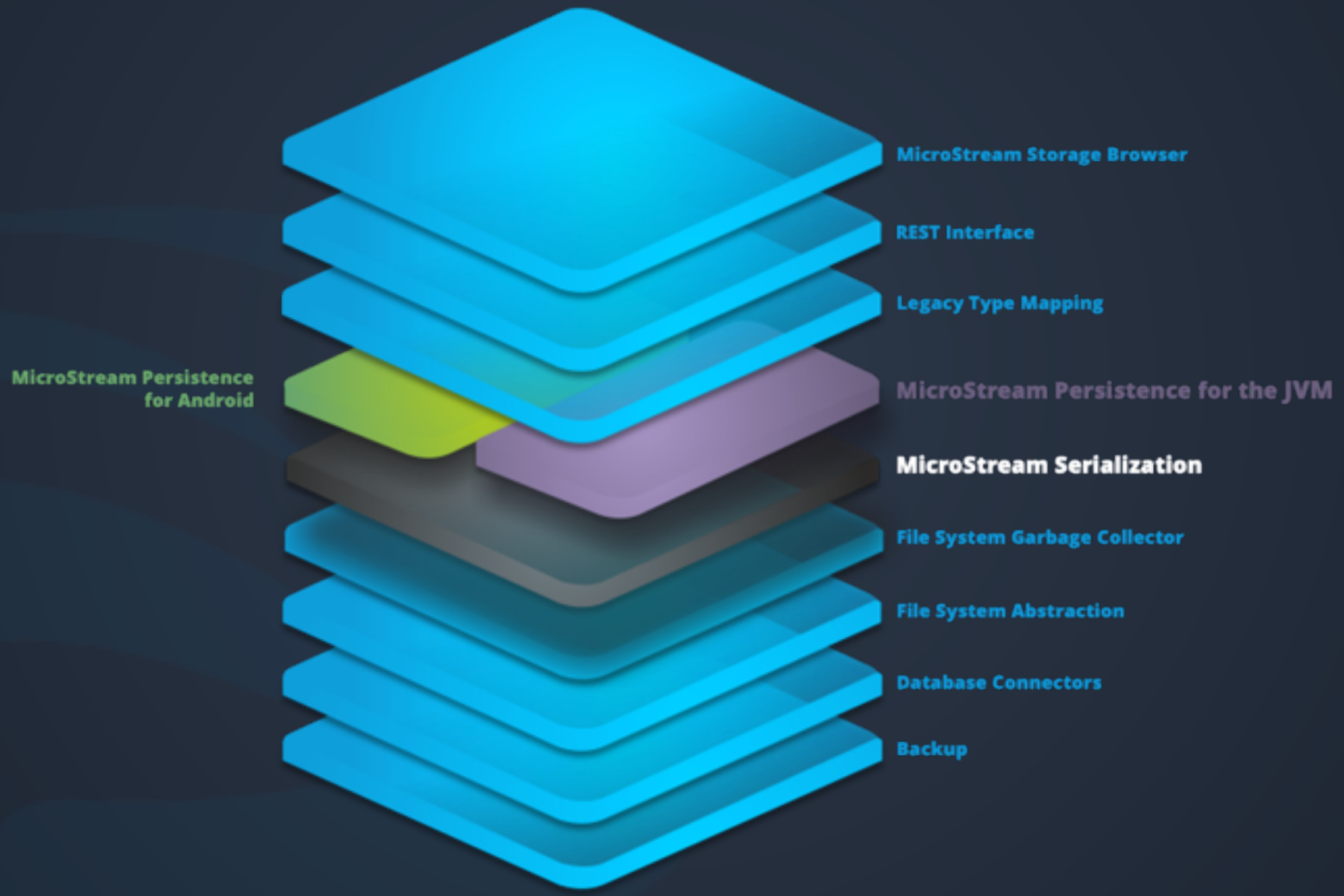


- Simple architecture
  - No Conversions
- Saves lots of vCPU power
- Minimizes latencies
- In-memory queries executed in microseconds

MicroStream Supported Storage



MicroStream Components



Demo

MicroStream Demo

- Define Storage Manager
- Write changed data
- Integrate with CDI and MicroProfile

Integration Testing

- Testing microservices can be challenge as it requires multiple components that work together.
- With TestContainers framework you can setup the service dependencies easily and repeatable within a container environment.

- Unit Tests that start container(s) and call application.

Demo

Challenge Day 2

Hands-on

- Add MicroStream persistence to shop of Jakarta Hands-on
  - Using MicroStream CDI integration
  - Or just plain MicroStream.

Thank you

- Rudy De Busscher - Atbash

- Blog
  - https://www.atbash.be
- Github
  - https://github.com/atbashEE
- Contacts
  - rdebusscher@gmail.com
  - @rdebusscher | @rdebusscher@mastodon.online