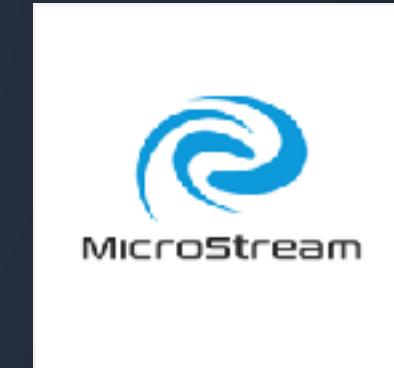




**JAVA PRO**



# Develop modern cloud-native microservices with Payara Micro





# Agenda

- 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





# 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



# 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

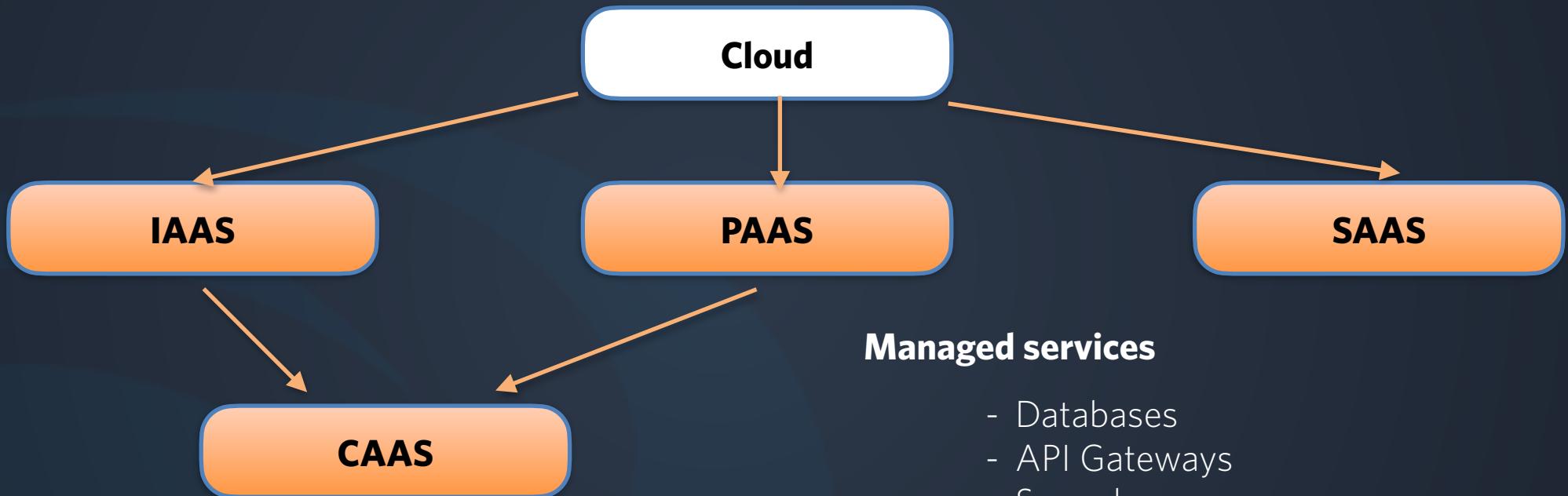


# On-premise to Cloud

- 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 😊)

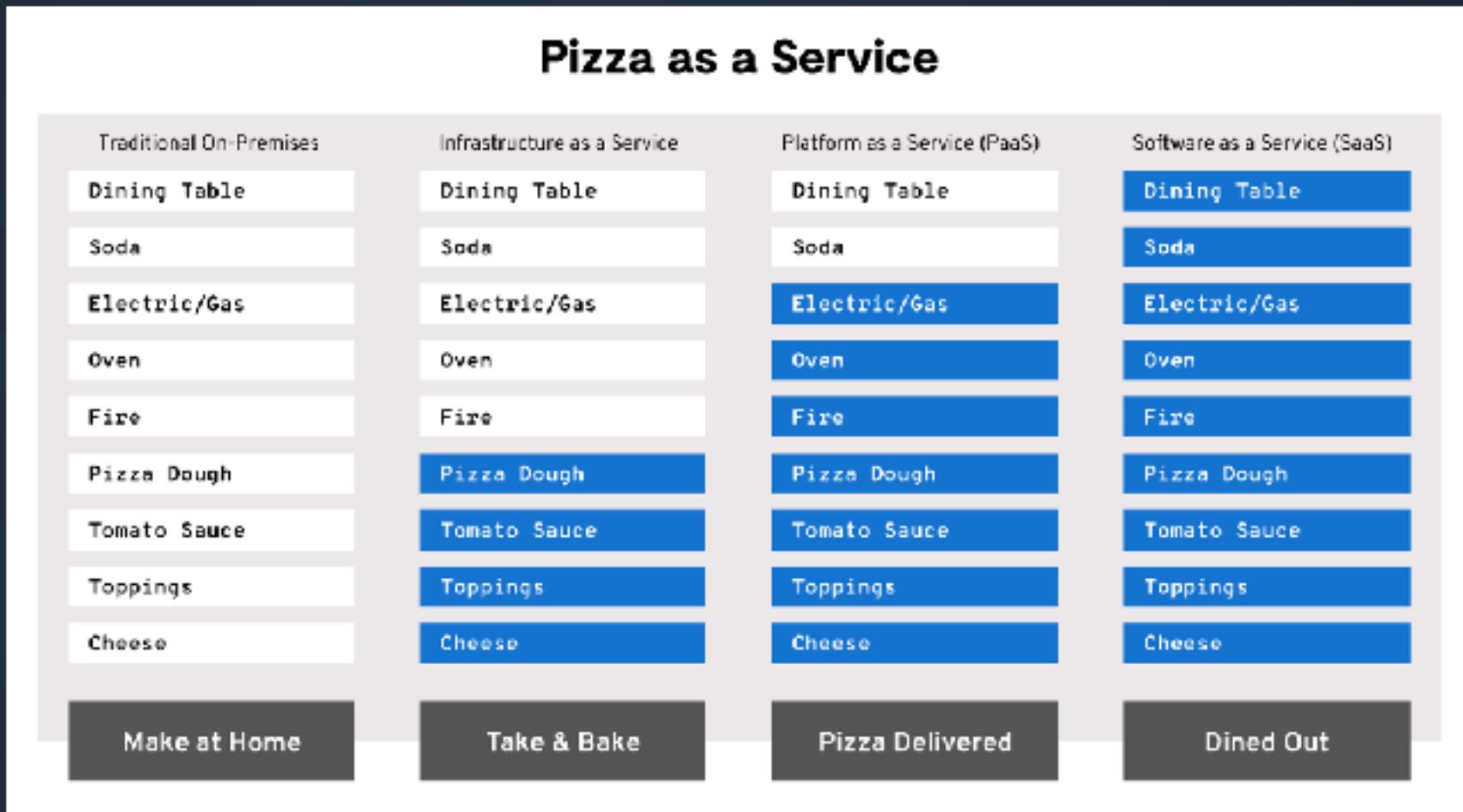


# Cloud Models





# Cloud Models





# Cloud Native

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

The screenshot shows the Pivotal website, which is now part of VMware. The header includes the Pivotal logo and navigation links for Why Pivotal, Kunden, Produkte und Dienstleistungen, Ressourcen, Unternehmen, Support, Anmelden, and a search icon. The main content features a dark background with a central graphic of overlapping colored squares (green, red, blue) and a white play button icon. The text "TECH INSIGHTS" is above the title "Cloud-Native Applications: Ship Faster, Reduce Risk, Grow Your Business". Below the title are links for Cloud-Native Apps, Perspectives, At Pivotal, Customer Stories, and Next steps. The section "What are Cloud-Native Applications?" provides a definition and notes that organizations require a platform for building and operating cloud-native applications.

Pivotal  
Now part of VMware

Why Pivotal Kunden Produkte und Dienstleistungen Ressourcen Unternehmen Support Anmelden

TECH INSIGHTS

## Cloud-Native Applications: Ship Faster, Reduce Risk, Grow Your Business

Cloud-Native Apps Perspectives At Pivotal Customer Stories Next steps

**What are Cloud-Native Applications?**

Cloud-native is an approach to building and running applications that exploits the advantages of the cloud computing model. The cloud has redefined the competitive landscape across virtually all industries by eliminating the focus on capital investment and staff to run an enterprise data center, replacing it with limitless computing power on-demand and pay-as-you-go. Reduced IT spend means a lower barrier to entry with competitive advantage becoming a function of the speed teams can bring new ideas to market, which is why software is eating the world and startups are using cloud-native approaches to disrupt traditional industries.

Yet organizations require a platform for building and operating cloud-native applications and services that automates and integrates the concepts of DevOps, continuous delivery, microservices, and containers.



## 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.”

Book Collection

### Learning Path Architecting Cloud Native Applications

Design high-performing and cost-effective applications  
for the cloud

Kamal Arora, Erik Farr, John Gilbert  
and Piyum Zonooz

Packt  
[www.packt.com](http://www.packt.com)



# 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 LOG IN SEARCH MENU

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



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.

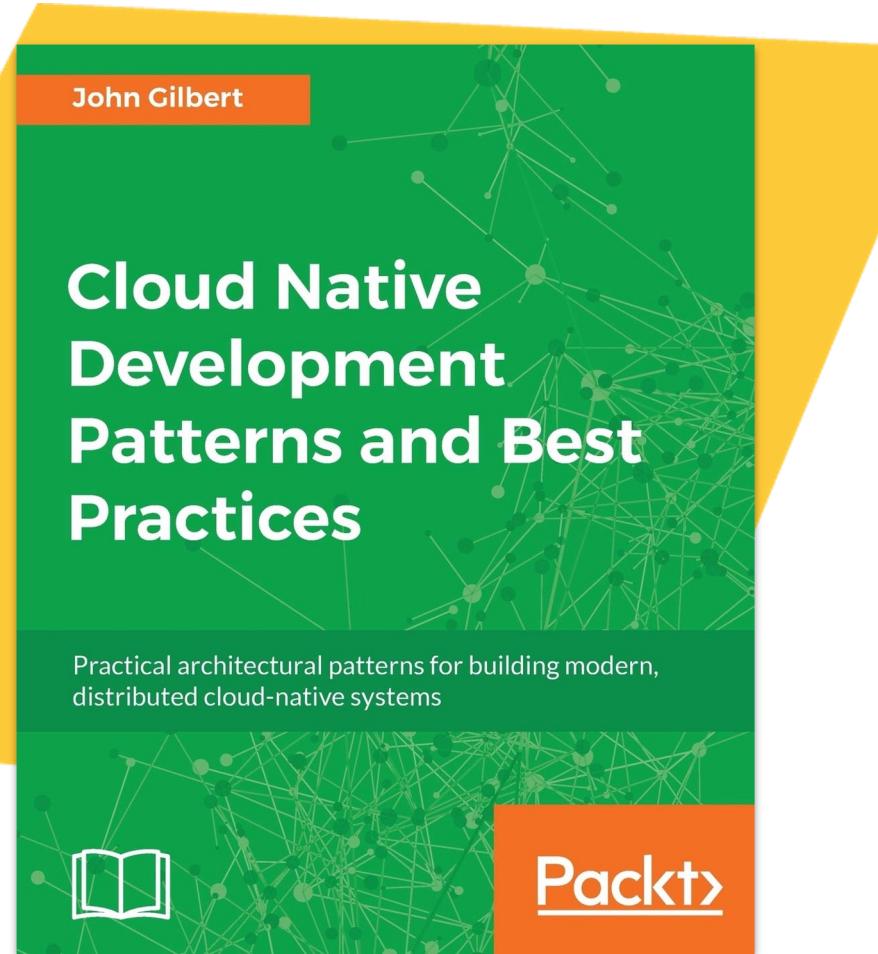
TABLE OF CONTENTS

- ‘Cloud-native’ defined
- Differences between cloud-native and on-premises applications



# Cloud Native

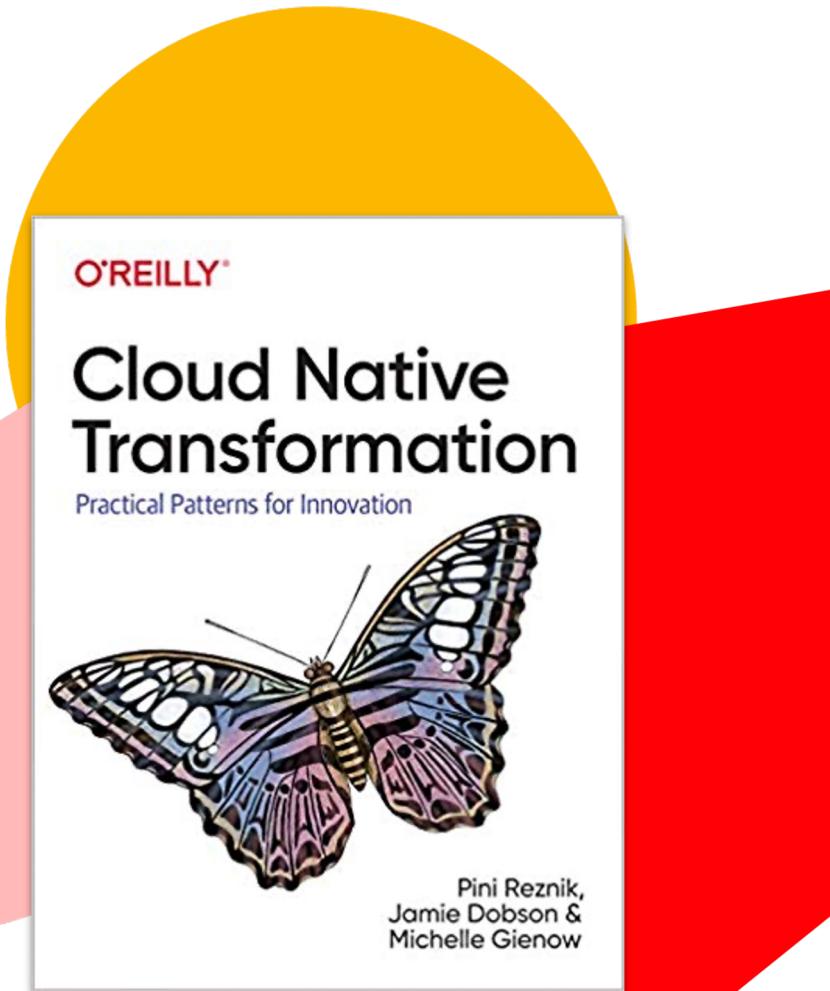
“Independent DURS ultimately comes up in every discussion on cloud-native concepts; to independently Deploy, Update, Replace and Scale.”





## 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

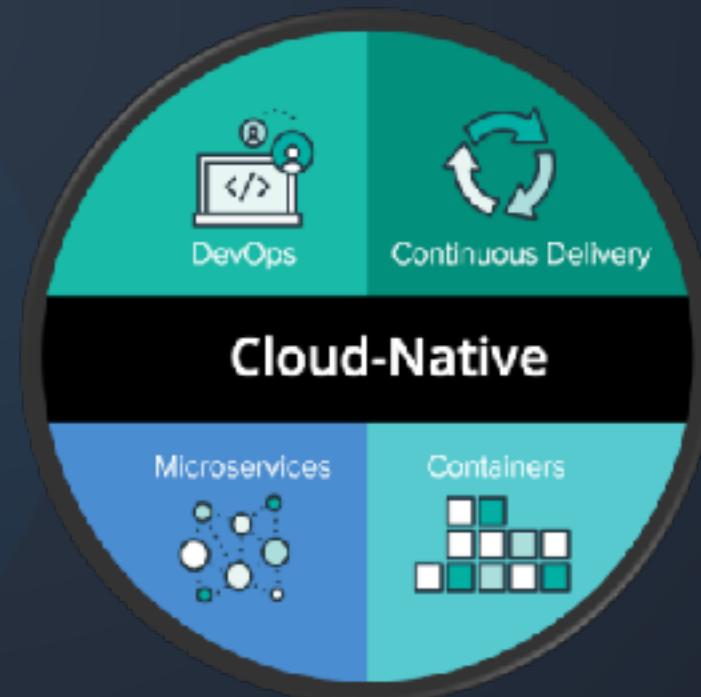
“Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.”





# Cloud-Native

- 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**

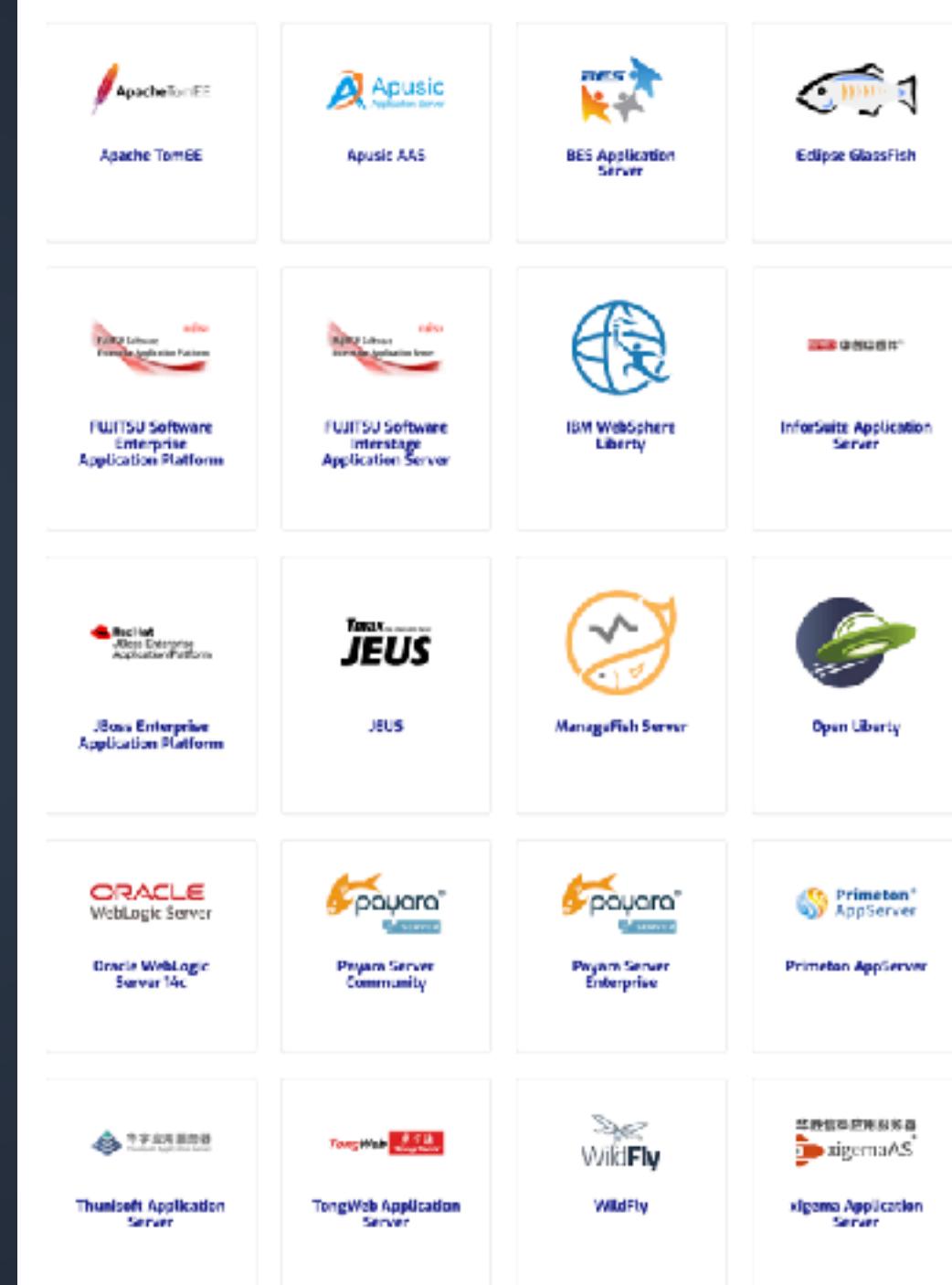


# Jakarta EE

- 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

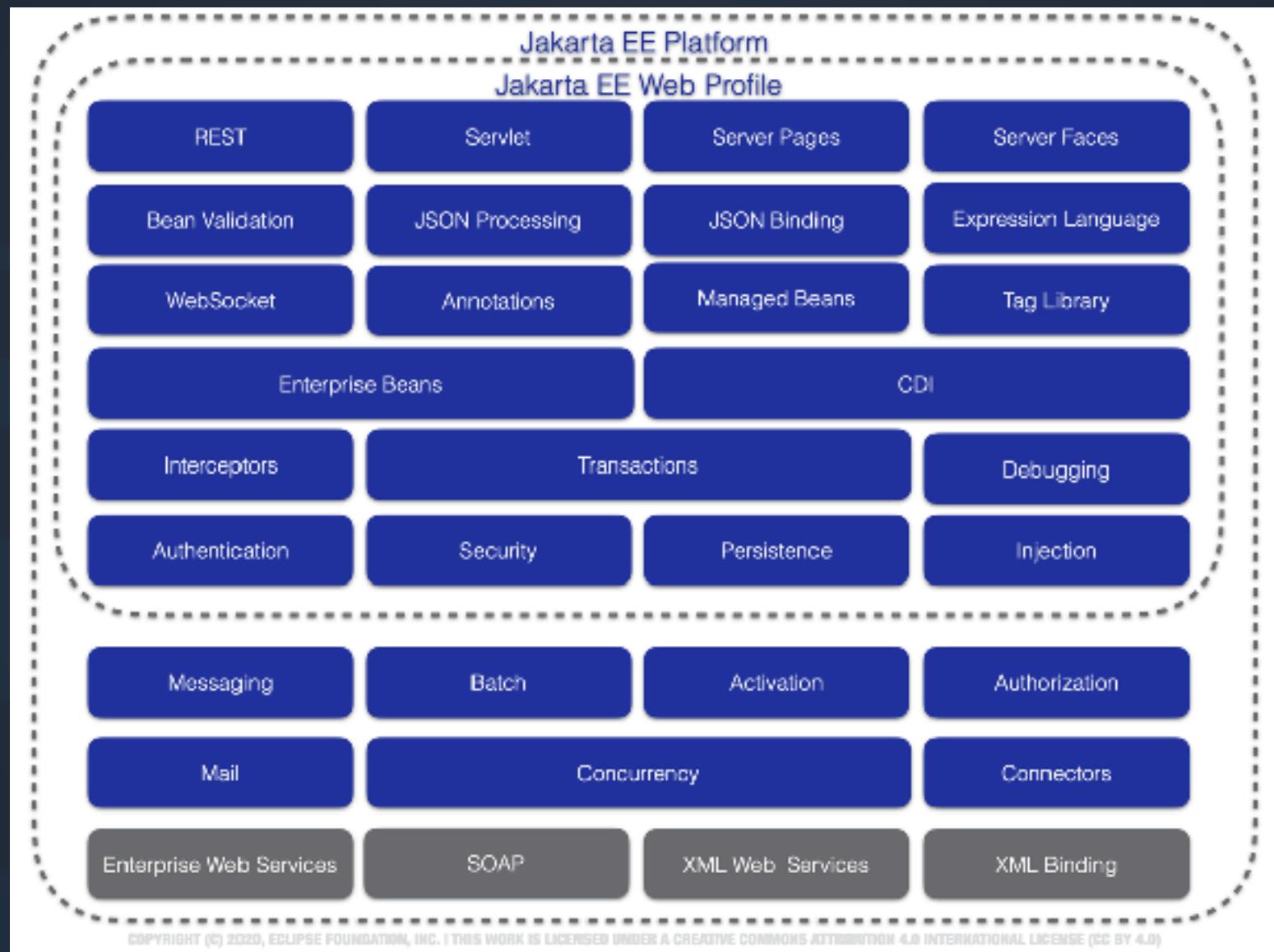
# Jakarta EE Specification

- 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

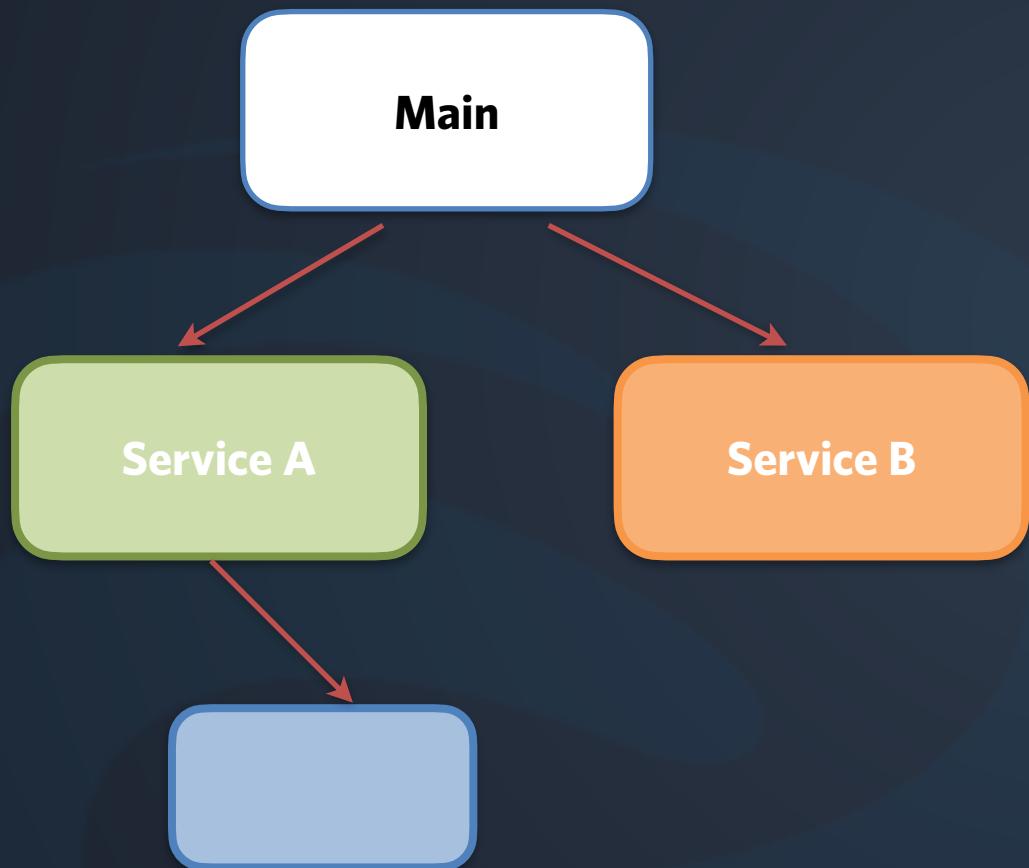


# Context and Dependency Injection

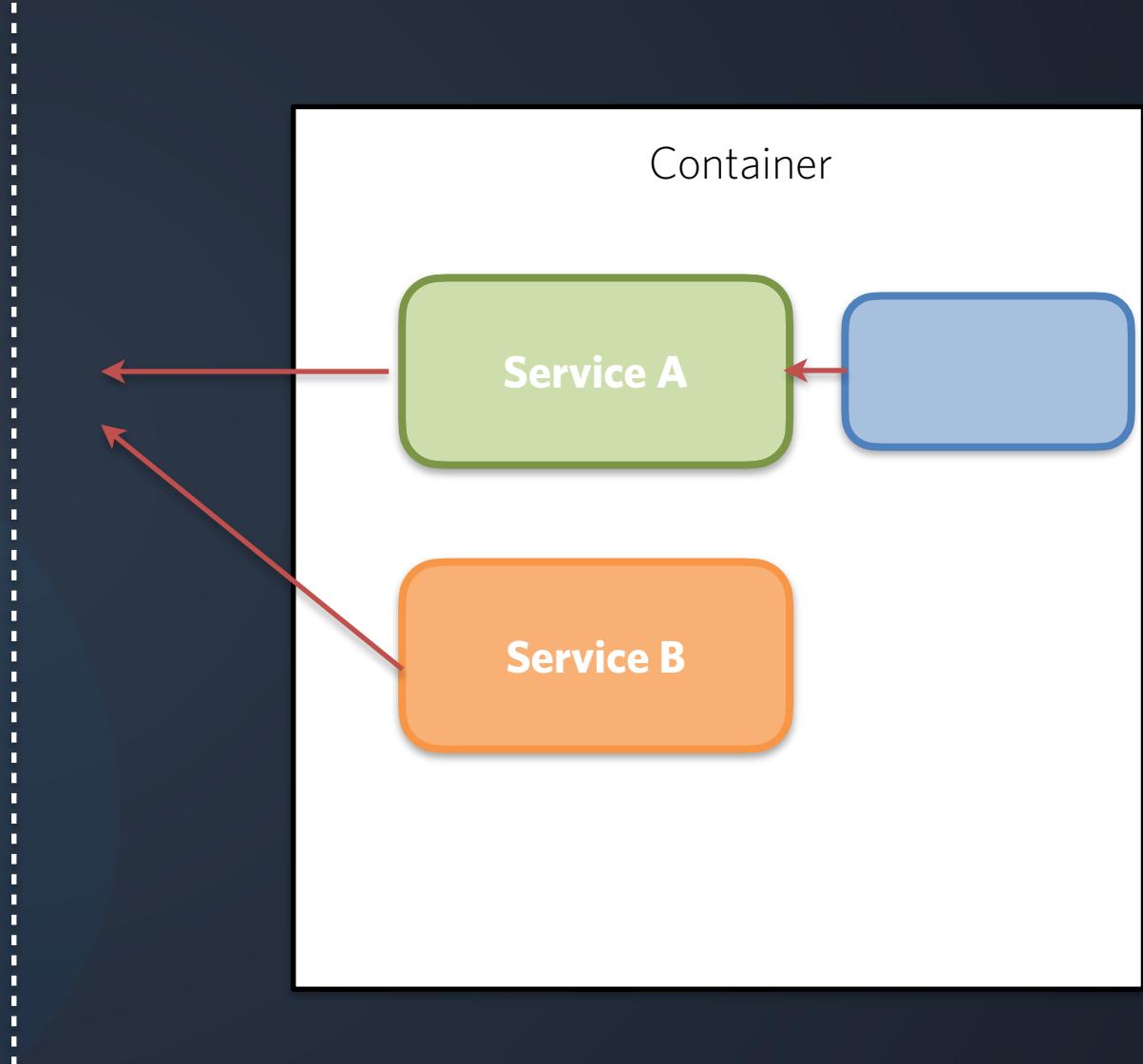
- Having well-defined lifecycle for stateful objects, beans
- Bound objects to lifecycle contexts = different lifecycles
- Creating decoupled and flexible code
  - Dependencies are provided by container
- Decorating injected objects
  - Interceptors
- Using events and notifications



# 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

The diagram illustrates the four types of Java application packaging:

- Skinny:** Represented by a red rectangle labeled "App". It is contained within a dashed outer boundary.
- Thin:** Represented by a blue rectangle labeled "App Dependencies". It is also contained within the same dashed outer boundary as the Skinny app.
- Fat/Uber:** Represented by an orange rectangle labeled "App Runtime". It is contained within a larger dashed outer boundary.
- Hollow:** This label points to the space between the "Thin" and "Fat/Uber" components, indicating that the hollow app runtime does not contain its own dependencies but rather depends on the external "Thin" layer.

[developers.redhat.com/blog/2017/08/24/the-skinny-on-fat-thin-hollow-and-uber/](https://developers.redhat.com/blog/2017/08/24/the-skinny-on-fat-thin-hollow-and-uber/)

redhat

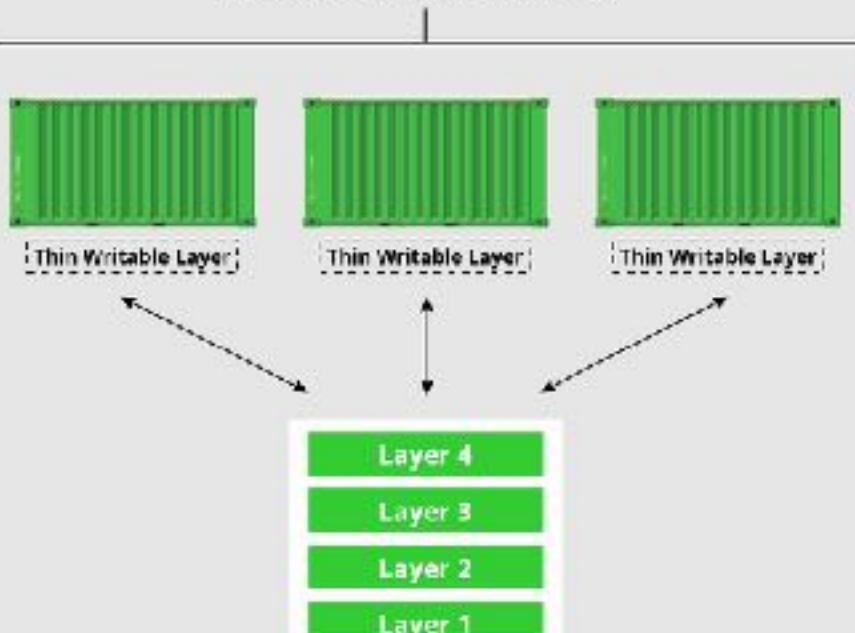


# Fat vs Hollow

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



## Docker Containers



## Docker Image



```
practical-devsecops@docker-Lab:~$ docker build -t myimap .
Sending build context to Docker daemon 89.56MB
Step 1/5 : FROM alpine:latest
--> 961789878411
Step 2/5 : RUN apk update
--> Using cache
--> 8d85a77f247c
Step 3/5 : RUN apk add imap
--> Using cache
--> b95843ffff997
Step 4/5 : ENTRYPOINT ["imap"]
--> Using cache
--> a511166c832a
Step 5/5 : CMD ["-h", "127.0.0.1"]
--> Running in 56adc0d5506e
Removing intermediate container 56adc0d5506e
--> Babe23abfb5a
Successfully built Babe23abfb5a
Successfully tagged myimap:latest
practical-devsecops@docker-Lab:~$
```

**Using cache and reusing the existing layer**



# Fat vs Hollow : Example with Payara Micro

- MicroProfile Starter
- start.microprofile.io

### MicroProfile Starter

Generate MicroProfile Maven Project with Examples

groupid \*

artifactid \*

MicroProfile Version

Java SE Version

Project Options

Build Tool

 Maven  Gradle

MicroProfile Runtime \*

Examples for specifications [Select All](#) [Clear All](#)

Config ↳  Fault Tolerance ↳

JWT Auth ↳  Metrics ↳

Health ↳  OpenAPI ↳

OpenTracing ↳  Rest Client ↳

**DOWNLOAD**



# 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



# JAX-RS Demo

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

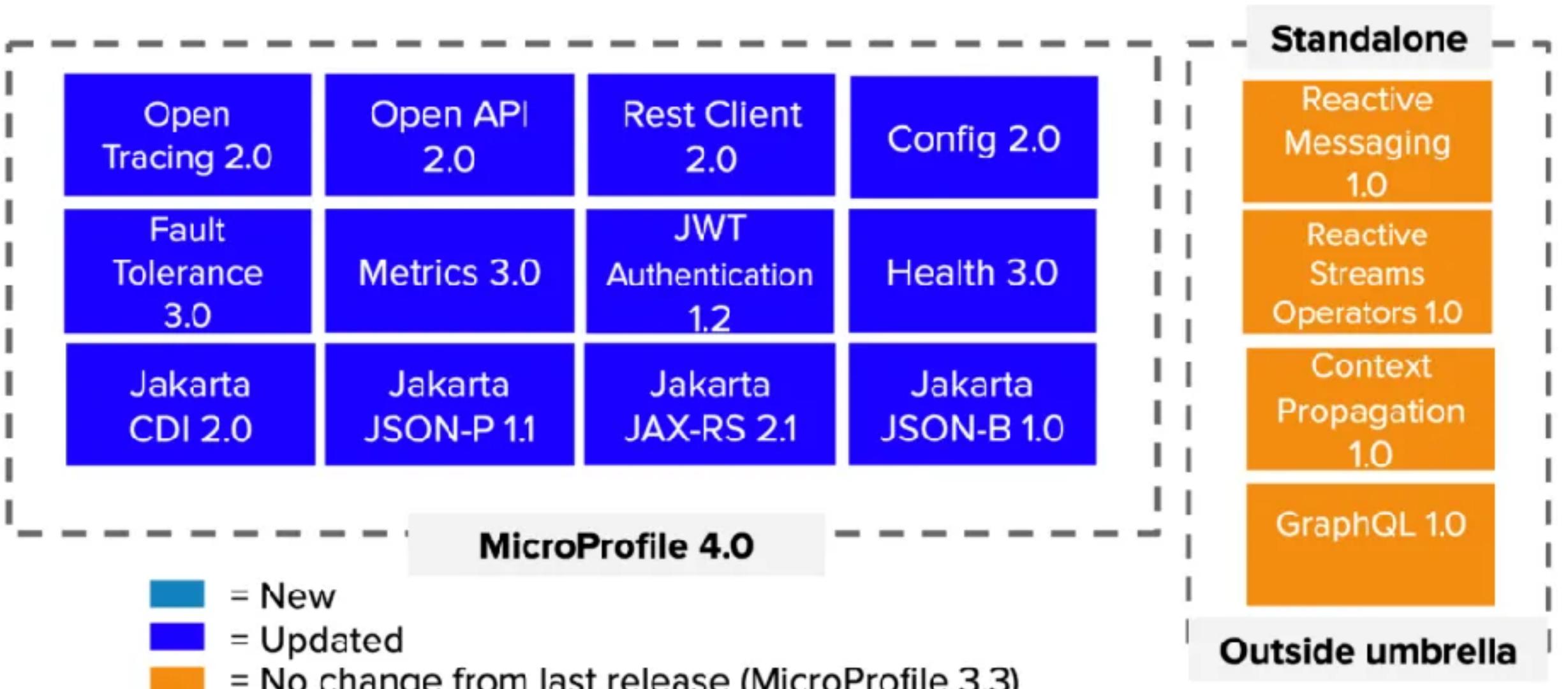


# MicroProfile

- 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
  - <https://docs.payara.fish/community/docs/documentation/payara-micro/appendices/cmd-line-opts.html>
  - --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 assignment
- <https://github.com/rdebusscher/payara-micro-course>





# 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

The word "REVIEW" is displayed in large, bold, purple 3D letters. A magnifying glass with a black handle is positioned over the letter "V", focusing on it. The background behind the text is white, and the overall design is set against a dark blue gradient background.



# 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

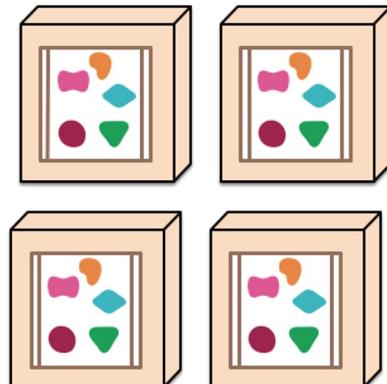


# Monoliths and Microservices

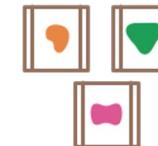
*A monolithic application puts all its functionality into a single process...*



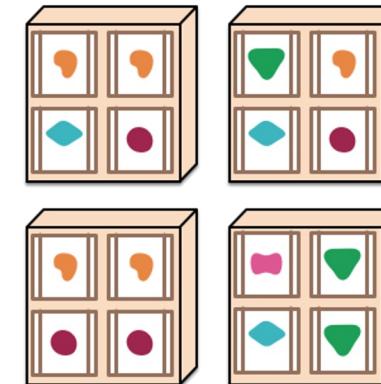
*... and scales by replicating the monolith on multiple servers*



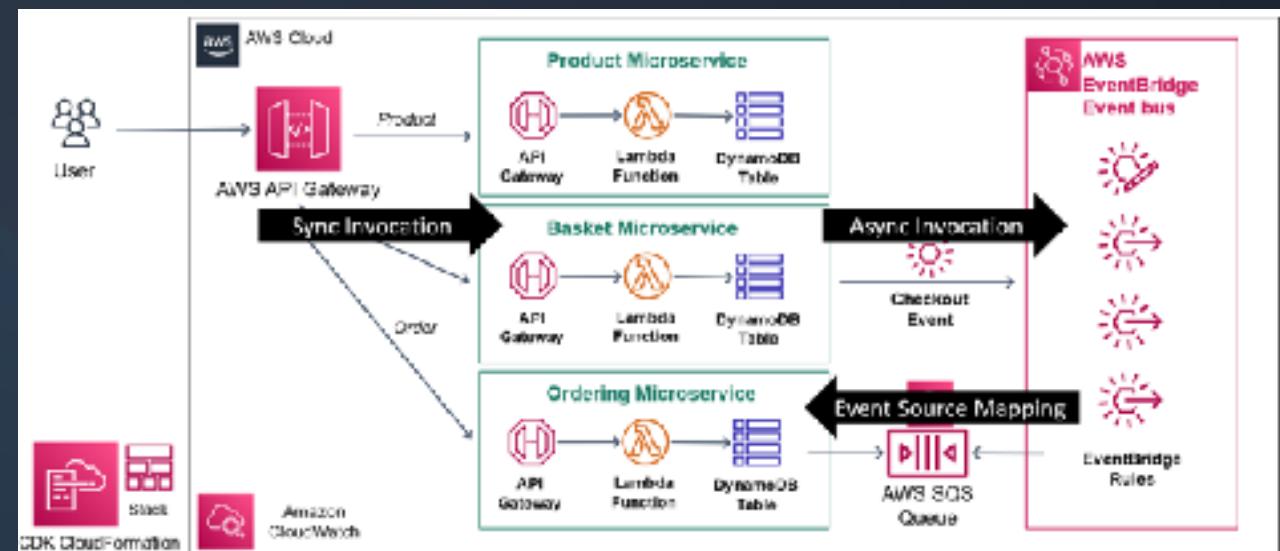
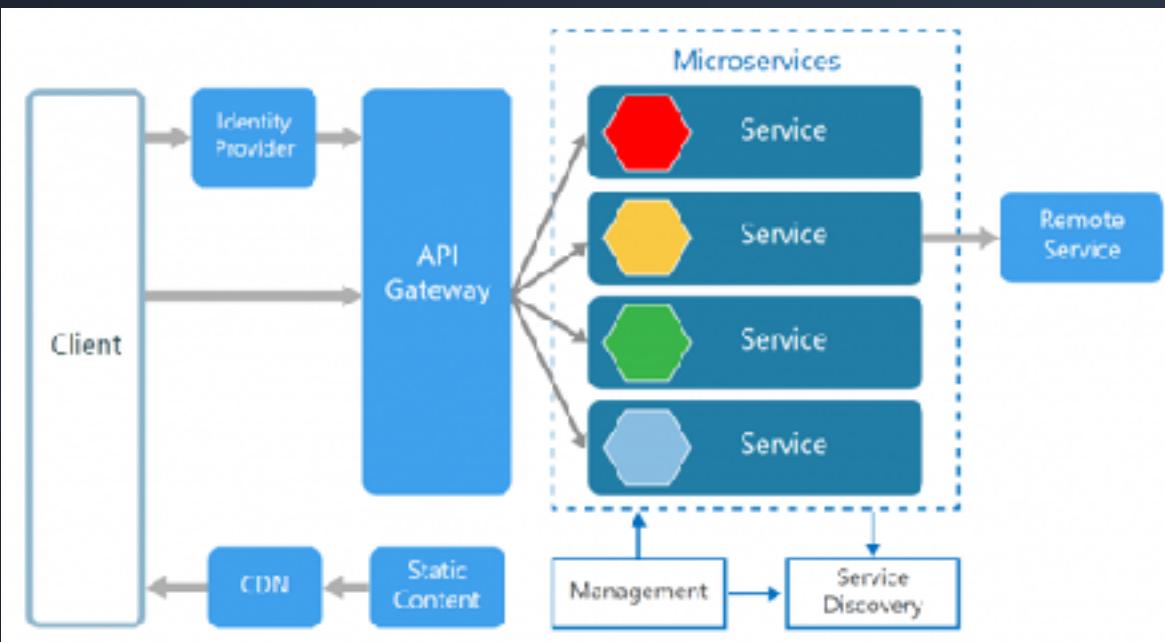
*A microservices architecture puts each element of functionality into a separate service...*



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



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





# 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*)
-



# 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 too restrictive
  - Does not allow some parts or combinations of the JOSE specification to 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
-

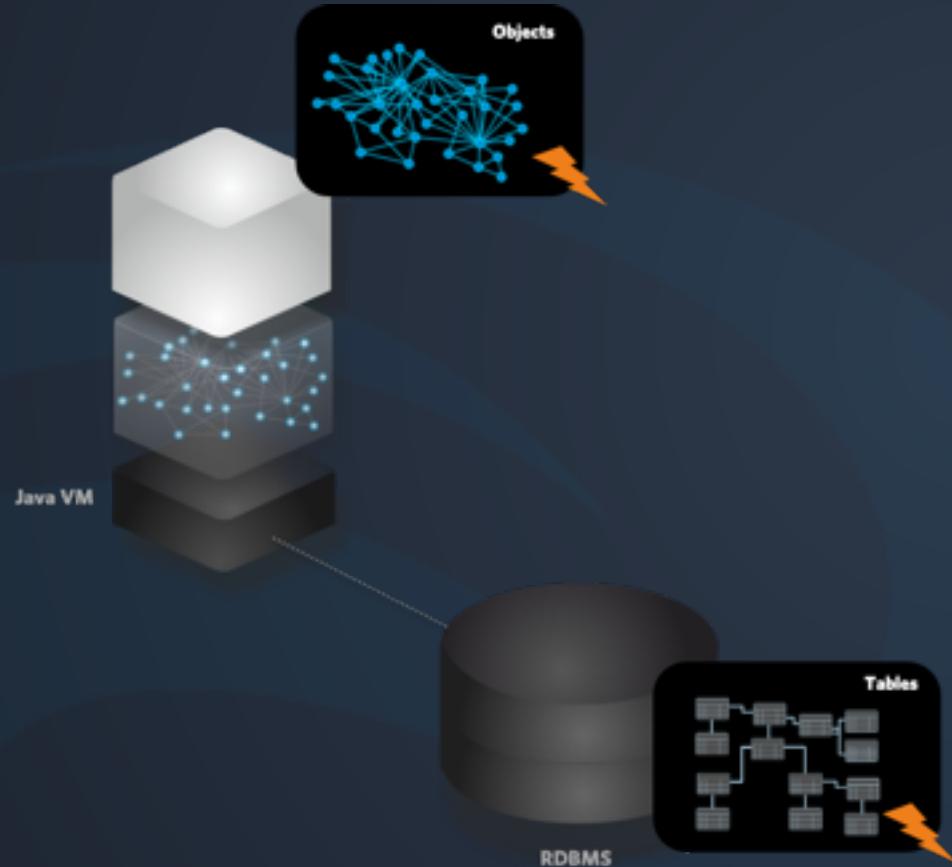


# MicroStream ???

- ??? Basic Slides
-



# Traditional Java Persistence

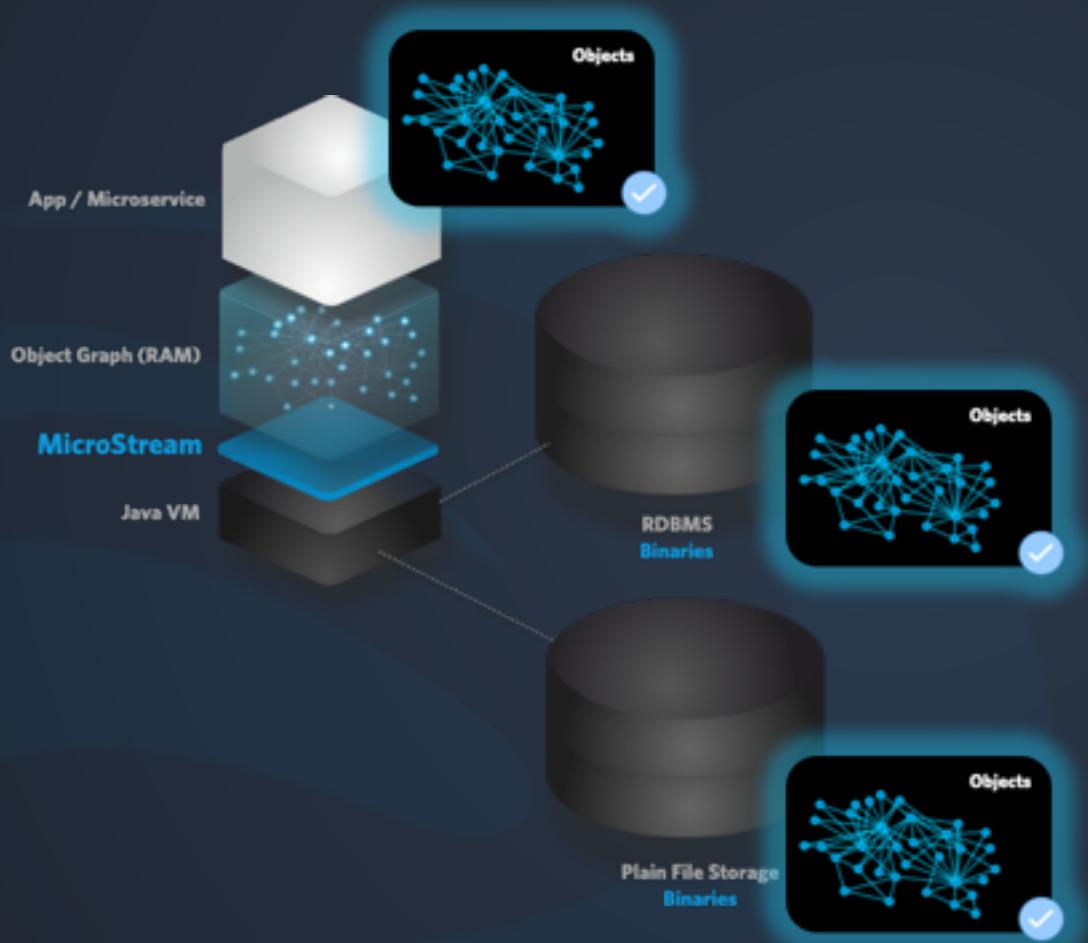


## Impedance Mismatch

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



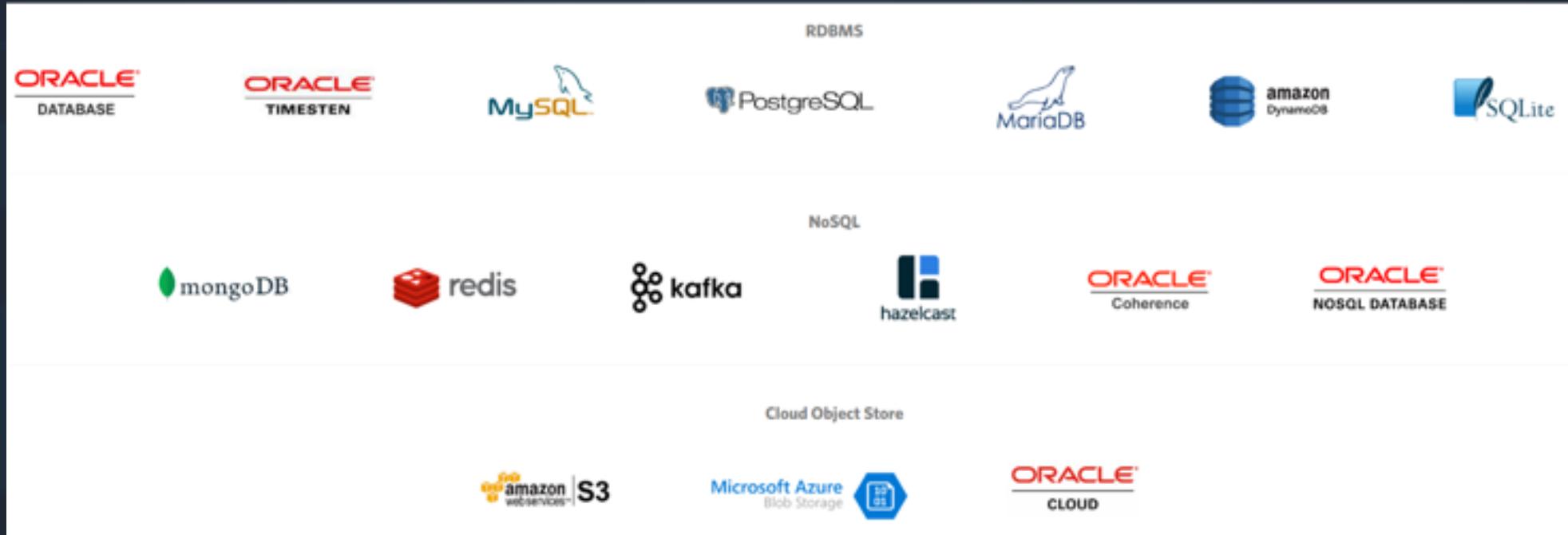
# MicroStream Persistence



- 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.



REVIEW

The word "REVIEW" is displayed in large, bold, purple 3D letters. A magnifying glass with a black handle is positioned over the letter "V", focusing on it. The background behind the text is white, and the overall design is set against a dark blue gradient background.