empowered by

payara

MicroProfile for MicroServices

Ankara Tech Talks #8

Mert ÇALIŞKAN

May 12, 2017

Who am I?

Mert Çalışkan

10+ Years on Enterprise Java

Coder, Author

Part Time Lecturer @ Hacettepe

Founder of AnkaraJUG

/mulderbaba

@mertcal

http://bit.ly/mertcaliskan

Agenda

- What MicroServices are and what it means to you?
- History of Java EE and stairway to MicroProfile effort
- Payara MicroProfile & Payara Micro
- With Samples and Code Walkthrough

Is Single Application enough?

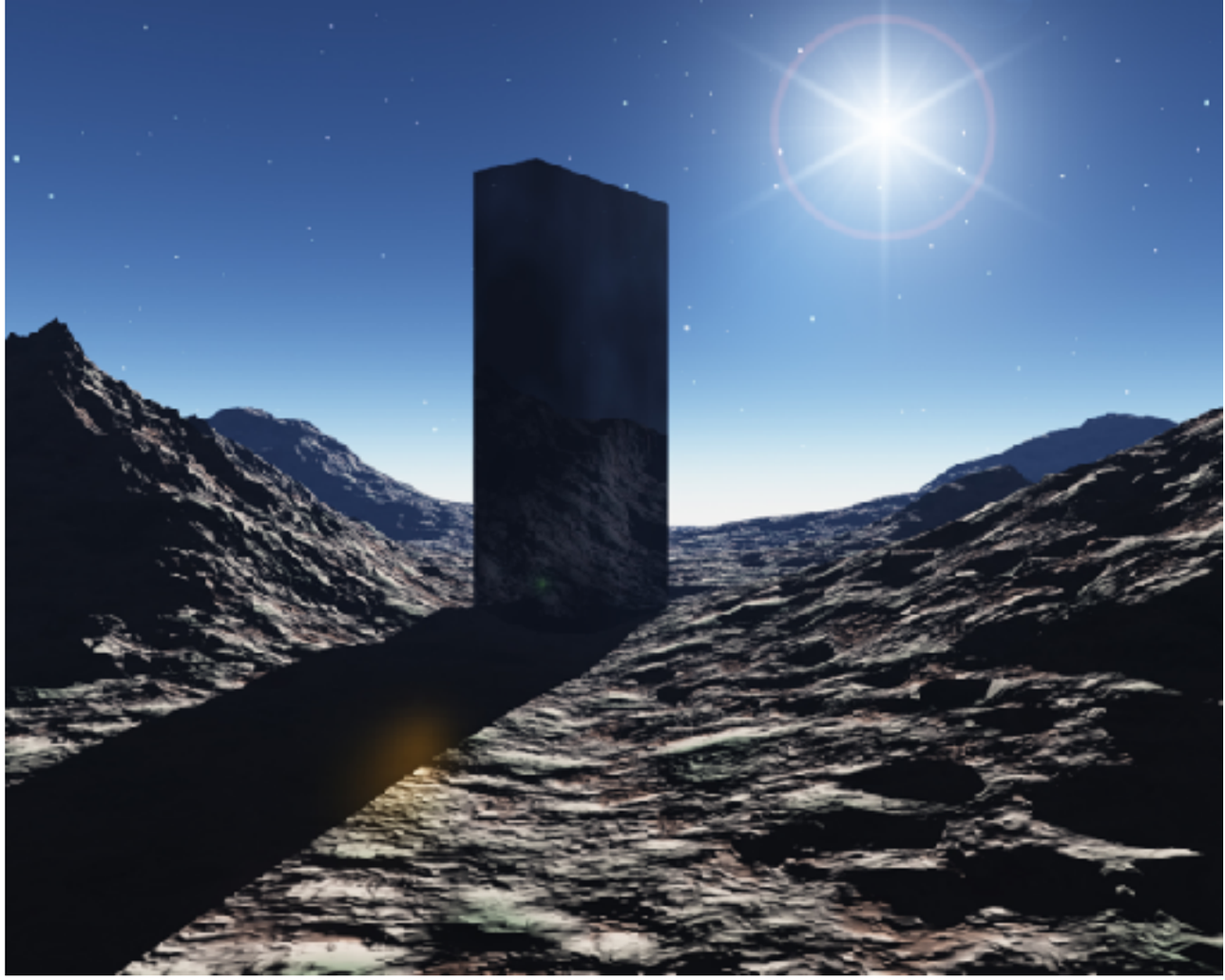
or DO YOU NEED SPECIAL ABILITIES?

Come to the MicroServices side..!

The Monolith

- For decades, we followed the approach of having a single application that implements diverse functions within its layers and deployed onto hardware that is pre-scaled (vertically) for peak loads.

- Business needs agility badly these days so having a monolithic architecture blocks the way we're implementing our applications.

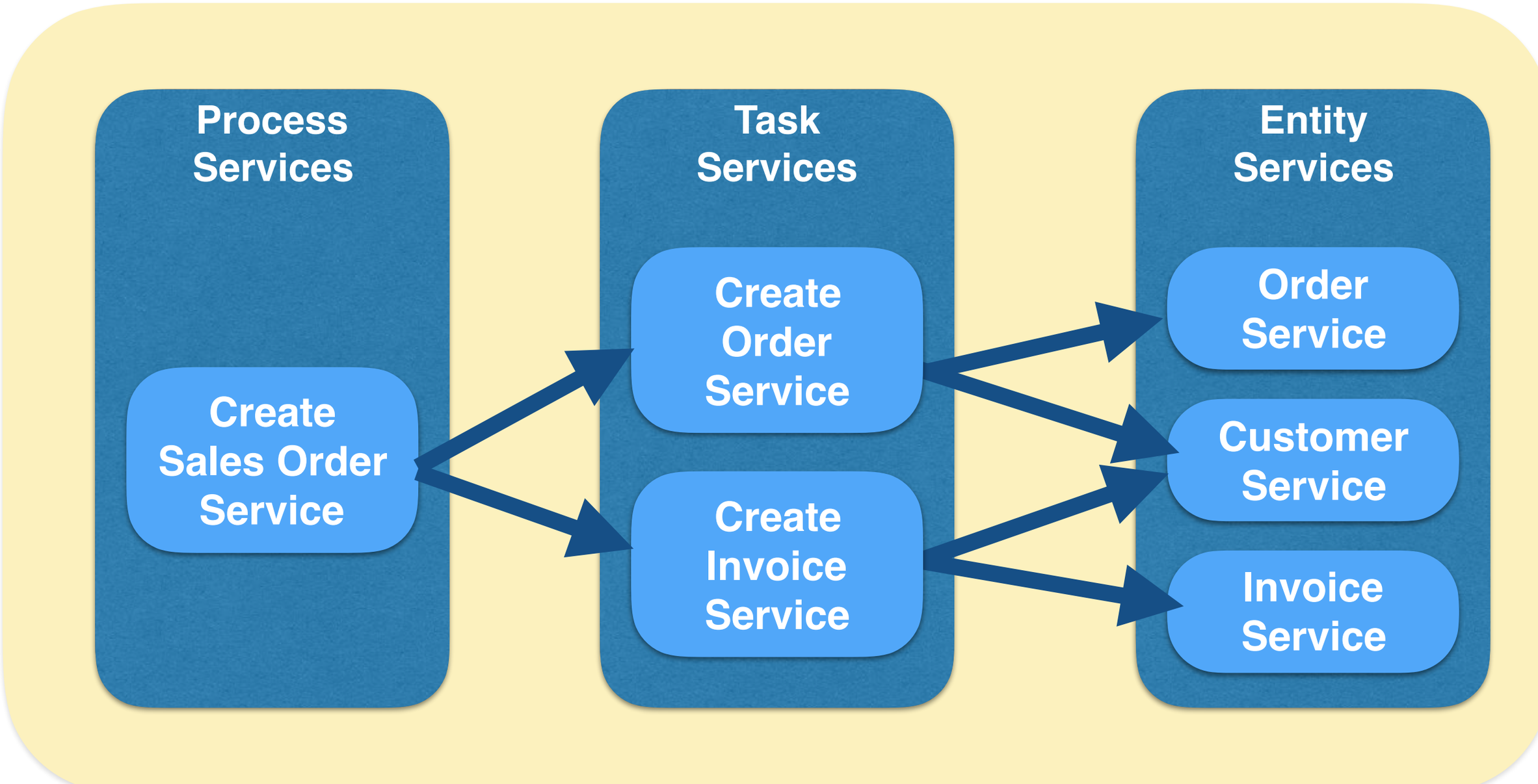


- Dealing with this "big black rock" is an enormous task to achieve indeed.

Breakdown of your Monolith

Create Sales Order Service

Breakdown of your Monolith



Again, What MicroServices are?

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare mininum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

- Martin Fowler, ThoughtWorks

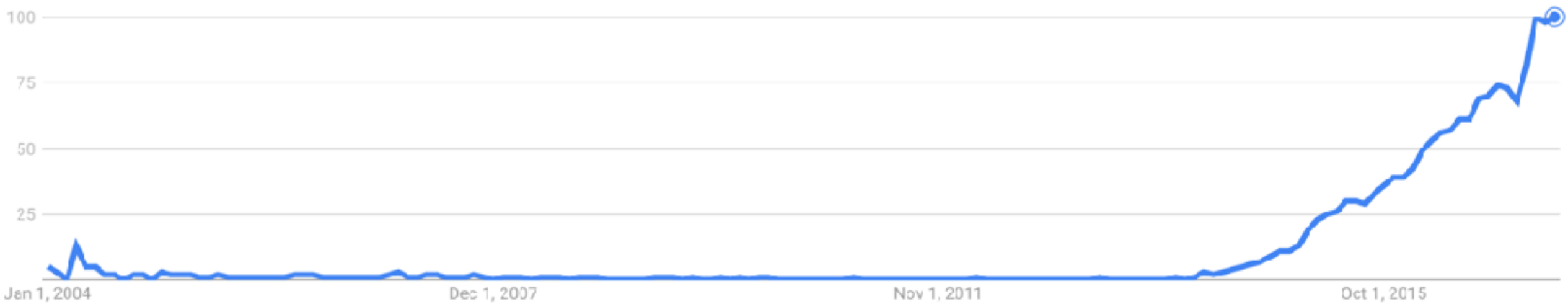
Again, What MicroServices are?

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare mininum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

- Martin Fowler, ThoughtWorks

What MicroServices are?

- There is a growing inclination on MicroServices



- But, it's not a new term actually, **Dr. Peter Rodgers** tossed the coin on the term **Micro-Web-Services** *in 2004* by saying:

  Software components are Micro-Web-Services

- That statement was mentioning that a well-designed service oriented platform applies the underlying architectural principles of the Web and Web services together with **Unix-like pipes** (meaning that services can call services)

And what MicroServices means to you?

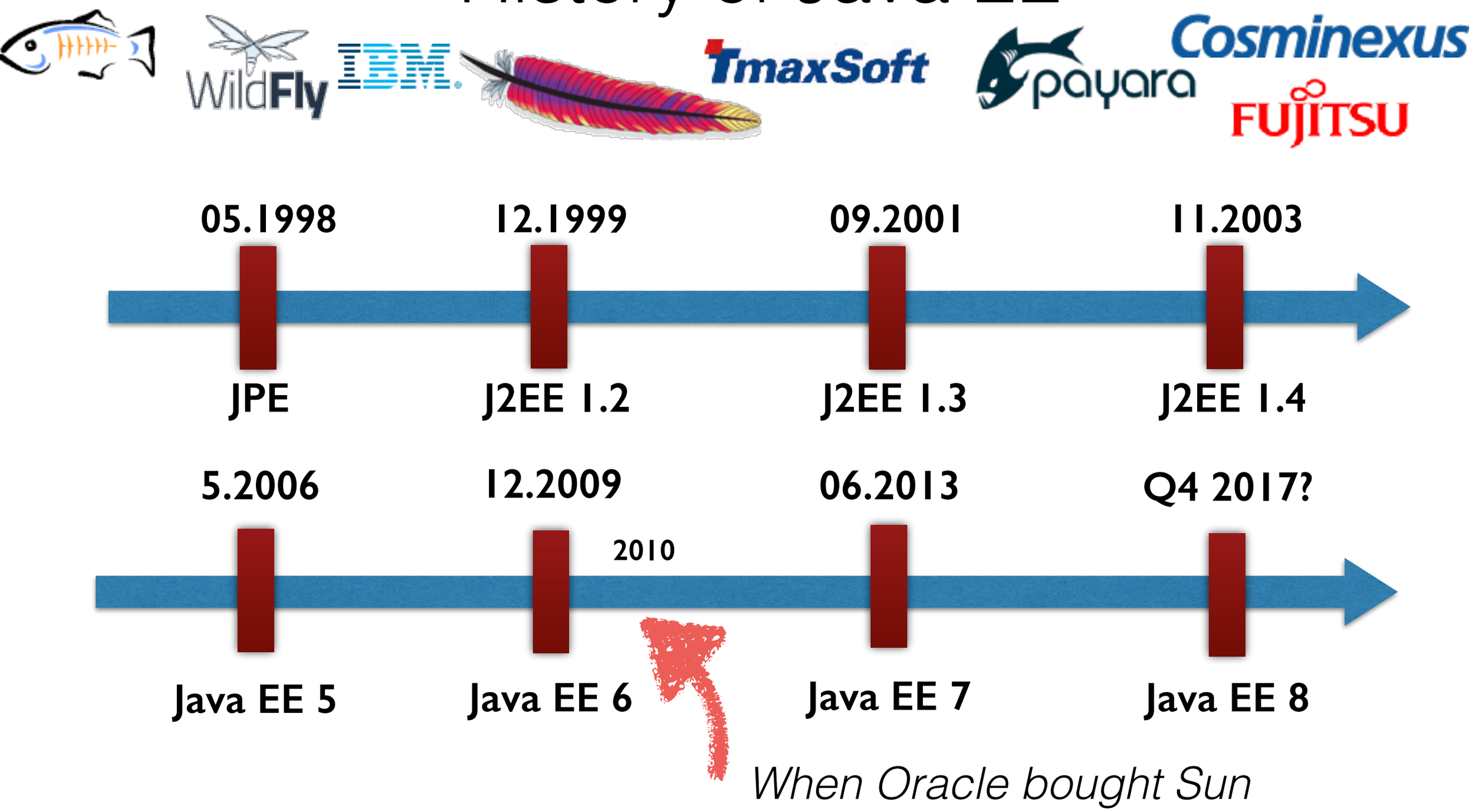
- *Having your software components as:*
  - with single responsibility (tested, deployed independently)
  - scaled and versioned easily.
  - implemented in different languages (polyglot components)
  - developed and maintained by different teams
  - deployed on containers that provide image creation & resource isolation

Fallacies of Distributed Computing

- When dealing with MicroServices, you will have a distributed architecture and these are the fallacies that you should be aware of.

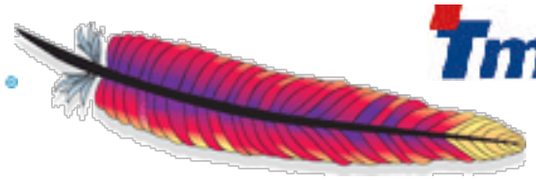
- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

History of Java EE



- All releases clearly defined what specs were included and releases haven't done any statements on the performance measurement or the size of the applications/application servers.

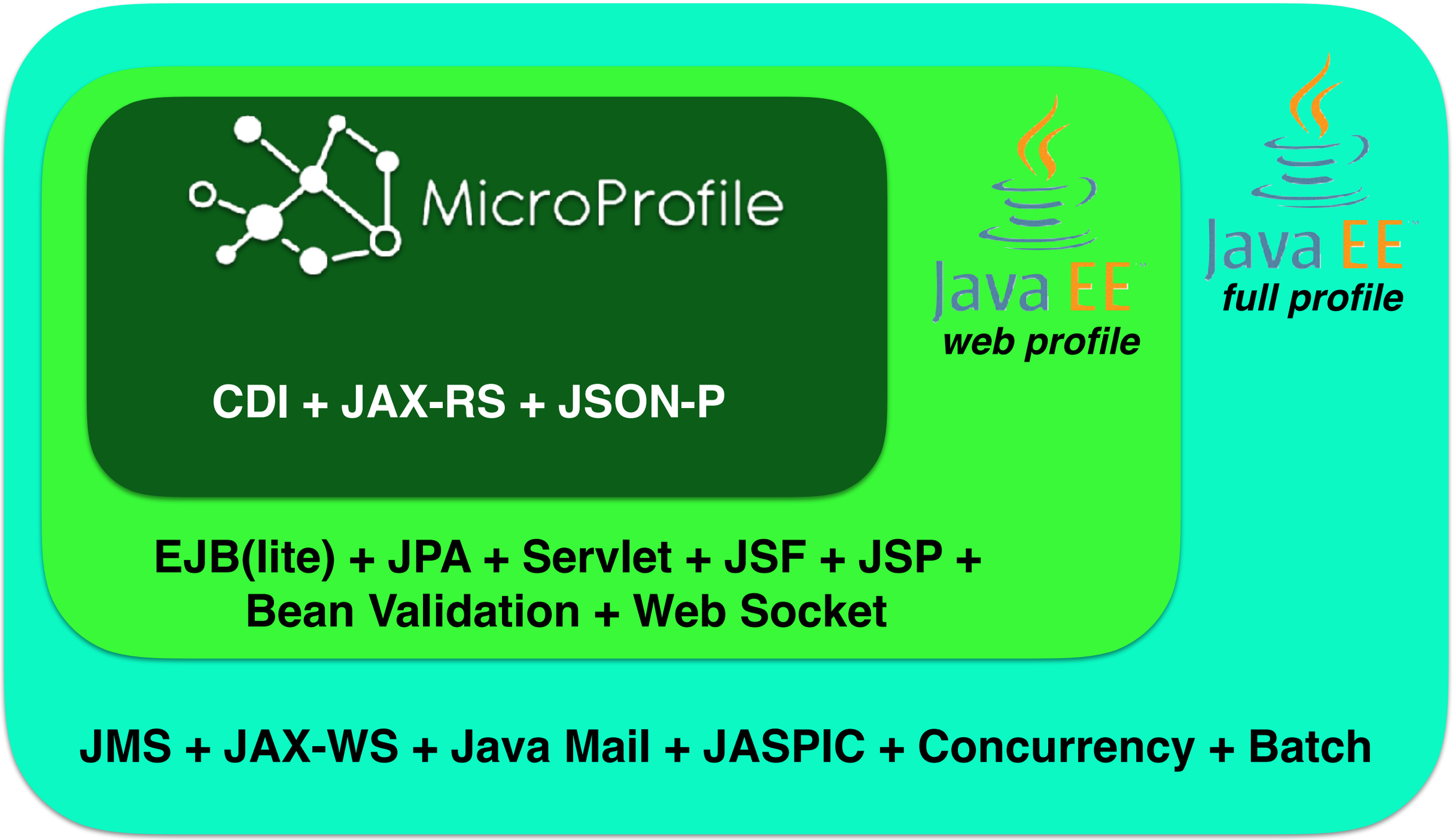
History of Java EE


	JSR Initiation	Early Draft Review	Public Review	Final Release
J2EE 1.4	2001/10	-	2002/5	2003/11
Java EE 5	2004/5	2005/4	2005/6	2006/5
Java EE 6	2007/7	2008/10	2009/1	2009/12
Java EE 7	2011/5	2012/4	2013/1	2013/5
Java EE 8	2014/8	2015/10		2017/8?

- All releases clearly defined what specs were included and releases haven't done any statements on the performance measurement or the size of the applications/application servers.

Profiles


MicroProfile
CDI + JAX-RS + JSON-P
Java EE
web profile
Java EE
full profile
EJB(lite) + JPA + Servlet + JSF + JSP +
Bean Validation + Web Socket
JMS + JAX-WS + Java Mail + JASPIC + Concurrency + Batch

The MicroProfile Effort

- A new community collaboration first announced at DevNation'16 (held in June) in San Francisco.
- Parties were payara IBM redhat LJC Tomitribe
- The mission was optimizing Enterprise Java for a MicroService architecture.
- At JavaOne'16 MicroProfile v1.0 was released with 4 implementations.
- More joined to the effort: SOUJava sociedade de usuários java hazelcast kumuluzEE FUJITSU payara

Implementations of MicroProfile

WildFly SWARM

```
<dependency>
   <groupId>org.wildfly.swarm</groupId>
   <artifactId>microprofile</artifactId>
   <version>2017.4.0</version>
</dependency>
```

```
<dependency>
   <groupId>com.kumuluz.ee</groupId>
   <artifactId>kumuluzee-microProfile-1.0</artifactId>
   <version>2.2.0</version>
</dependency>
```

kumuluz EE

WebSphere Liberty

```
<dependency>
   <groupId>com.ibm.websphere.appserver.runtime</groupId>
   <artifactId>wlp-microProfile1</artifactId>
   <version>17.0.0.1</version>
</dependency>
```

payara

Payara Microfile

```
<dependency>
   <groupId>fish.payara.extras</groupId>
   <artifactId>payara-microprofile</artifactId>
   <version>1.0-4.1.1.171.1</version>
</dependency>
```

Apache TomEE™

```
<plugin>
        <groupId>org.apache.tomee.maven</groupId>
        <artifactId>tomee-maven-plugin</artifactId>
        <version>7.0.3</version>
        <configuration>
                <tomeeClassifier>webprofile</tomeeClassifier>
        </configuration>
</plugin>
```

Payara MicroProfile

- MicroProfile distribution from the Payara Team!
- One JAR under 40mb. Available via Maven as:

```
<dependency>
    <groupId>fish.payara.extras</groupId>
    <artifactId>payara-microprofile</artifactId>
    <version>1.0-4.1.1.171.1</version>
</dependency>
```

- v1.0 released on September'16 and as of March'17 we have v1.0-4.1.1.171.1, which aligns with the Payara Full itself.
- Deploying applications easy as from command line:

```
java -jar payara-microprofile-1.0-4.1.1.171.1.jar
     --deploy app1.war --deploy app2.war
```

Payara MicroProfile

- You can even deploy with GAV coordinates as:

```
java -jar payara-microprofile-1.0-4.1.1.171.1.jar
 --deployfromGAV "fish.payara.examples,app,1.0.0-SNAPSHOT"
```

- We have moved onto nested jar structure from the shaded jar structure, which makes the artefact more modular and simple to work with.

- It's important to mention that ***MicroProfile is not to compete directly with Java EE***, but to come up with innovative ways of implementing MicroServices with Enterprise Java

- As of December'16, after reaching a consensus in the community, MicroProfile was moved to **Eclipse Foundation** with *Apache License.*

- This brings the vendor neutrality, meaning that no vendor would own the MicroProfile.

- MicroProfile 1.1 is underway and planned to be released at 2nd quarter of '17 with features:
    - Configuration 1.0
    - Health Check
    - Security (JWT)
    - Fault Tolerance

- And MicroProfile 1.2 is under way on Q3'17 with HealthCheck, Security and Fault Tolerance. MicroProfile 2.0 will probably be released at the same time Q3'17 but with alignment of JavaEE8 specs like CDI 2.0, JAX-RS 2.1, JSON-P 1.1

Configuration 1.0

- Aim is to separate a MicroService from its configuration.

- This would enable to secure the configuration or having it configured dynamically.

- No need to rebuild/redeploy the code when
  - every time there is a change in the configuration
    (Dynamic Configuration influenced by Netflix - archaius)
  - moving to another environment

- Properties could either be defined in the following order *(ordinal)*
  1. System Properties - *400*
  2. Environment Variables - *300*
  3. Configuration File META-INF/microprofile-config.properties - *100*
  4. Custom ConfigSource objects

- Multiple configuration sources can be merged into a single configuration and can be accessed with one API

Configuration 1.0

- The whole configuration can be injected (via CDI) as follows:

```
@Inject
Config config;
String appName =
       config.getRawString("APP_NAME").orElse("MicroDemo");
```

- Or a single property could be injected as:

```
@Inject
@ConfigProperty
String PROPERTY_NAME1;

@Inject
@ConfigProperty(name = "PROPERTY_NAME2")
String propertyTwo;
```

- Or get property programmatically

```
Config config = ConfigProvider.getConfig();
String appName = config.getRawString("APP_NAME").orElse("MicroDemo");
```

Configuration 1.0

- Built-in converters are provided for:
  `String`, `Boolean`, `Integer`, `Long`, `Short`, `Byte`, `Double`, `Float`, `BigInteger`, `BigDecimal`, `AtomicInteger`, `AtomicLong`, `Duration`, `Period`, `LocalDateTime`,`LocalDate`, `LocalTime`, `OffsetDateTime`, `OffsetTime`, `ZonedDateTime`, `Instant`, `Date`, `Currency`, `BitSet`, `URI`, and `URL`

- Custom converters can also be implemented via interface:
  `org.eclipse.microprofile.config.spi.Converter<T>`

- Dynamic Property Updating can be enabled on `ConfigSource` impl by setting `refreshInterval`

```
@Inject
@ConfigProperty
ConfigValue<MyClass> PROPERTY_NAME3;
MyClass mc = PROPERTY_NAME3.getValue();
```

Configuration 1.0

- 4 implementation exist:
  - IBM WebSphere Liberty Profile
  - Apache Tamaya
  - WildFly Swarm
  - Apache Geronimo
- First version planned to be released at the time of Devoxx.UK (11-12th of May) and it will require JDK 8.

HealthCheck

- Aim is to collect the health information by probing the state of a computing node from another other machine (as in Kubernetes service controller)

- REST endpoints should be implemented by delivering collected health information.

- With this way, services can also be defined as *Cloud-Native*, that is if a service is unhealthy, it can be restarted, re-created or failed by the resource manager (could be a part of the cloud provider).

- context: **/health**, GET
    - 200 - for a health check with a positive outcome
    - 204 - in case no health check procedures are installed into the runtime
    - 503 - in case the overall outcome is negative
    - 500 - in case the consumer wasn't able to process the health check request (i.e. error in procedure)

HealthCheck

- A sample REST resource would be as follows:

```
@Path("/app")
public class HealthCheckResource {

    @GET
    @Path("/diskspace")
    @Health
    public HealthStatus checkDiskspace() {
        File path = new File(System.getProperty("user.home"));
        long freeBytes = path.getFreeSpace();
            long threshold = 1024 * 1024 * 100; //100mb
            return freeBytes> threshold?
            HealthStatus.named("diskspace").
                up().withAttribute("freebytes", freeBytes) :
            HealthStatus.named( "Diskspace").
                down().withAttribute("freebytes", freeBytes);
    }
}
```

Security (JWT Token Exchange)

- The security requirements that involve MicroService architectures are strongly related with RESTful Security
- In a REST based architecture, services are usually **stateless** and security state associated with client is sent to target service on every request to re-create the context.
- This is a perfect job for security tokens, like it's being done in OAuth2, OpenID Connect, SAML, WS-Trust and others.
- Currently there is a proposal for using OpenID Connect(OIDC) based JSON Web Tokens(JWT) for role based access control(RBAC) of MicroService endpoints.

Security (JWT Token Exchange)

- Goals:
  - Services don’t need to store any state about clients or users
  - Services can verify the token validity if token follows a well known format. Otherwise, services may invoke a separated service.
  - Services can identify the caller by introspecting the token. If the token follows a well known format, services are capable to introspect the token by themselves, locally. Otherwise, services may invoke a separated service.
  - Services can enforce authorization policies based on any information within a security token
  - Support for both delegation and impersonation of identities

Security (JWT Token Exchange)

- JWT (JSON Web Tokens) is well-defined and a known standard for protecting the services.
- It's not just tokens, it also provides features like encryption, signing or expiration checks.
- They are JSON based, so it's so easy to parse for us w/ JSON-P.
- Token validation doesn't require an additional trip and can be validated locally by each service, so that's how JWT got popular.

Security (JWT Token Exchange)

```
{
   "iss": "https://server.example.com",
   "sub": "24400320",
   "preferred_username": "jdoe",
   "aud": "s6BhdRkqt3",
   "nonce": "n-0S6_WzA2Mj",
   "exp": 1311281970,
   "iat": 1311280970,
   "auth_time": 1311280969,
   "realm_access": {
      "roles": [
         “role-in-realm", “user", "manager"
      ]
   },
   "resource_access": {
      "my-service": {
         "roles": [
            "role-in-my-service"
         ]
      }
   }
}
```

Encoded version

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsInN1YiI6IjI0NDAwMzIwIiwicHJlZmVycmVkX3VzZXJuYW1lIjoiamRvZSIsImF1ZCI6InM2QmhkUmtxdDMiLCJub25jZSI6Im4tMFM2X1d6QTJNaiIsImV4cCI6MTMxMTI4MTk3MCwiaWF0IjoxMzExMjgwOTcwLCJhdXRoX3RpbWUiOjEzMTEyODA5NjksInJlYWxtX2FjY2VzcyI6eyJyb2xlcyI6WyJyb2xlLWluLXJlYWxtIiwidXNlciIsIm1hbmFnZXIiXX0sInJlc291cmNlX2FjY2VzcyI6eyJteS1zZXJ2aWNlIjp7InJvbGVzIjpbInJvbGUtaW4tbXktc2VydmljZSJdfX19.nvb6lDXc7I8jdFpcNoL4XMJwU7NVGcy_r20CIFdIG3I

Fault Tolerance

- The aim is to separate the application execution logic from error handling execution and make service invocation more resilient.
- Provided patterns are: TimeOut, RetryPolicy, Fallback, BulkHead and CircuitBreaker.

Fault Tolerance

- Creating a retryPolicy with max of 3 retries and 2 seconds of delay between retries.

```
// Acquire retryPolicy from FT Factory
RetryPolicy retryPolicy =
FaultToleranceFactory.getFaultToleranceType(RetryPolicy.class);

RetryPolicy rp = retryPolicy
    .retryOn(Exception.class) // retry condition
    .withDelay(2, TimeUnit.SECONDS) // retry interval
    .withMaxRetries(3); // retry count Connection

//An inconsistent service, sometimes throws an exception
Runnable mainService = () -> serviceA();

//This is the fallback service
Runnable fallbackService = () -> serviceB();
```

Fault Tolerance

- So next step is to create an executor that will run serviceA and then fallback onto serviceB with a retryPolicy.

```
// Create a FaultTolerance Executor.
Executor executor =
    FaultToleranceFactory.getFaultToleranceType(Executor.class);

// Configure it to execute "serviceA",
   with our RetryPolicy and with a fallback to "serviceB"
executor.with(retryPolicy)
    .withFallback(fallbackService)
    .run(mainService);
```

Fault Tolerance

- A fault tolerance bulkhead limits the number of concurrent calls to a service.

```
// Create a Bulkhead
Bulkhead bulkhead = FaultToleranceFactory.getFaultToleranceType(Bulkhead.class);

// Create a ThreadPoolExecutor
int poolSize = 5; // Poolsize of 5
ThreadPoolExecutor tpexecutor =
      new ThreadPoolExecutor(poolSize, poolSize, 0, TimeUnit.SECONDS,
                             new ArrayBlockingQueue<Runnable>(10));

// Configure the Bulkhead to support the ThreadPoolExecutor
bulkhead = bulkhead.withThread(tpexecutor);

// Create a FaultTolerance Executor
Executor executor = FaultToleranceFactory.getFaultToleranceType(Executor.class);

// Spin off a number of services. It will be observed that services are executed
// in batches of 5.
for (int i = 0; i < 50; i++) {
    // Create an instance of Runnable
    Runnable mainService = new MyRunnableService("" + i);

    executor.with(bulkhead)
        .run(mainService);
}
```

Fault Tolerance

- A fault tolerance circuit breaker provides a way for systems to fail-fast. It temporarily disables the running of a service to prevent the service from overloading a system.

```
// Create a CircuitBreaker
CircuitBreaker circuitBreaker =
    FaultToleranceFactory.getFaultToleranceType(CircuitBreaker.class);

// Set up a Delay of 3 seconds
Duration delay = Duration.ofSeconds(3);

// Set up a Timeout of 3 seconds
Duration timeout = Duration.ofSeconds(3);

// Configure Fault Tolerance CircuitBreaker
circuitBreaker = circuitBreaker.withTimeout(timeout) // a timeout after 3
seconds counts as a failure
    .withFailureThreshold(3) // Open circuit after 3 failures
    .withSuccessThreshold(2) // set breaker half-open and if 2 trial
                             // executions succeed then close the breaker
                             // allow normal execution
    .withDelay(delay);       // after circuit broken, delay for 3 seconds
```

Fault Tolerance

- A fault tolerance circuit breaker provides a way for systems to fail-fast. It temporarily disables the running of a service to prevent the service from overloading a system.

```
// This unreliable Service sometimes succeeds but sometimes times out.
Callable<Object> mainService = () -> serviceA();

// Create a FaultTolerance Executor
Executor executor =
        FaultToleranceFactory.getFaultToleranceType(Executor.class);

// Executor with circuit breaker, can be called many times
for (int i = 0; i < 50; i++)
{
    executor.with(circuitBreaker).get(mainService);
    // A method to report the state of the CircuitBreaker using
circuitBreaker.isOpen(), etc calls
    checkBreakerState(circuitBreaker);
    // as serviceA() is unreliable the code can observe the Circuit
Breaker transition between CLOSED,
    // HALF-OPEN and OPEN states.
}
```

Feature Backlog*

- Distributed Logging
- Distributed Tracing
- Service Discovery
- MicroService Security
- Metrics/Monitoring
- Testing
- JCache
- Bean Validation
- Big Data / NoSQL support
- JPA
- JTA
- WebSockets
- Servlets
- OAuth2/OpenID Connect
- Concurrency Utilities for Java EE
- Asynchronous / Reactive Support / Patterns
- Java 9 Modularity

** items are subject to move due to lack of interest*

MicroProfile Conference Sample App

- Collaborative sample app that consists of several MicroServices and a Web-Application managing a conference.

1

`microservice-schedule` : Schedule of the conference - On Payara Micro
`microservice-session` : Sessions of the conference - On Wildfly Swarm
`microservice-speaker` : Speakers of the conference - On TomEE
`microservice-vote` : Votes for each session - On WebSphere Liberty
`web-application` : Frontend Angular2 | Bootstrap4

- Execute all with: `mvn clean package -P start,ui`

- All sources available at:
  https://github.com/eclipse/microprofile-conference

Payara Micro

- Application server as executable JAR.
- Big brother of Payara MicroProfile!
- It is small <70mb. with v171 it also has a nested JAR architecture.
- It’s based off of GlassFish Embedded.
- Deploy your WARs easily from command line.
- It provides automatic and elastic clustering
    - spawn many MicroServices dynamically
    - replication using distributed cache

Payara Micro

- It's using **Hazelcast** integration, each Payara Micro process will automagically cluster with other Payara Micro processes on the network, giving web session resilience and a fully distributed data cache using Payara's JCache support.

- *Payara MicroProfile* release is a cut-down of Payara Micro.

- Bundled specs inside are:

JPA
WebSockets
Bean Validation
Servlets
JCache
JSF
EJB Lite
CDI
Interceptors
JTA
JAX-RS
JSP
JBatch
Concurrency

@mertcal

payara

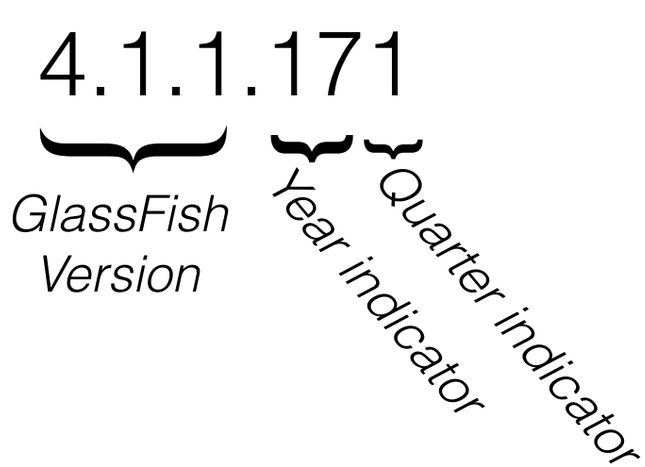
Streaming Events on Payara Micro

- This example demonstrates Clustered CDI EventBus of Payara Micro where a CDI event gets fired from one instance and received from another one inside the Payara Micro Cluster.

- Code available at: https://github.com/payara/Payara-Examples/tree/master/Payara-Micro/cdi-clustered-events

And the big old brother of all, Payara Server

- Payara Server is a drop-in replacement of GlassFish Server
- We provide enhancements, bug fixes and patches to upstream of GlassFish Server and dependent libraries as well like, Eclipselink, Tyrus, Jersey, Weld, Hibernate and etc.
- Quarterly releases since October 2014. Versions are defined as:



Resources

- Microprofile.io - The community landing site.
  all related materials are available at: https://github.com/microprofile

- Google+ Microprofile groups - Public discussion lists
  bit.ly/MicroProfileForum

- Eclipse github repositories - https://github.com/eclipse/microprofile

- Release Repo - https://repo.eclipse.org/content/repositories/microprofile-releases/

- Snapshot Repo - https://repo.eclipse.org/content/repositories/microprofile-snapshots/

Where to fetch MicroProfile artifacts

- Payara.Fish Downloads
  http://www.payara.fish/downloads

- WildFly Swarm MicroProfile Distro Generator
  http://wildfly-swarm.io/generator

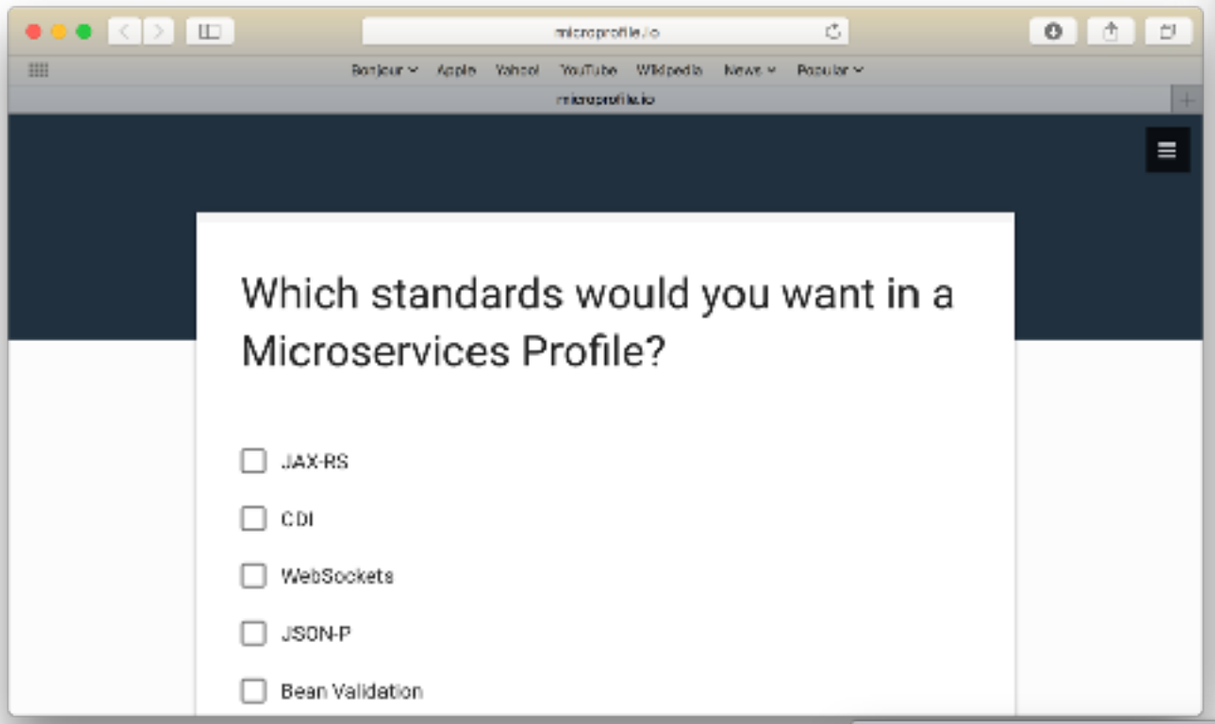
- WebSphere Liberty App Accelerator
  https://liberty-app-accelerator.wasdev.developer.ibm.com/start

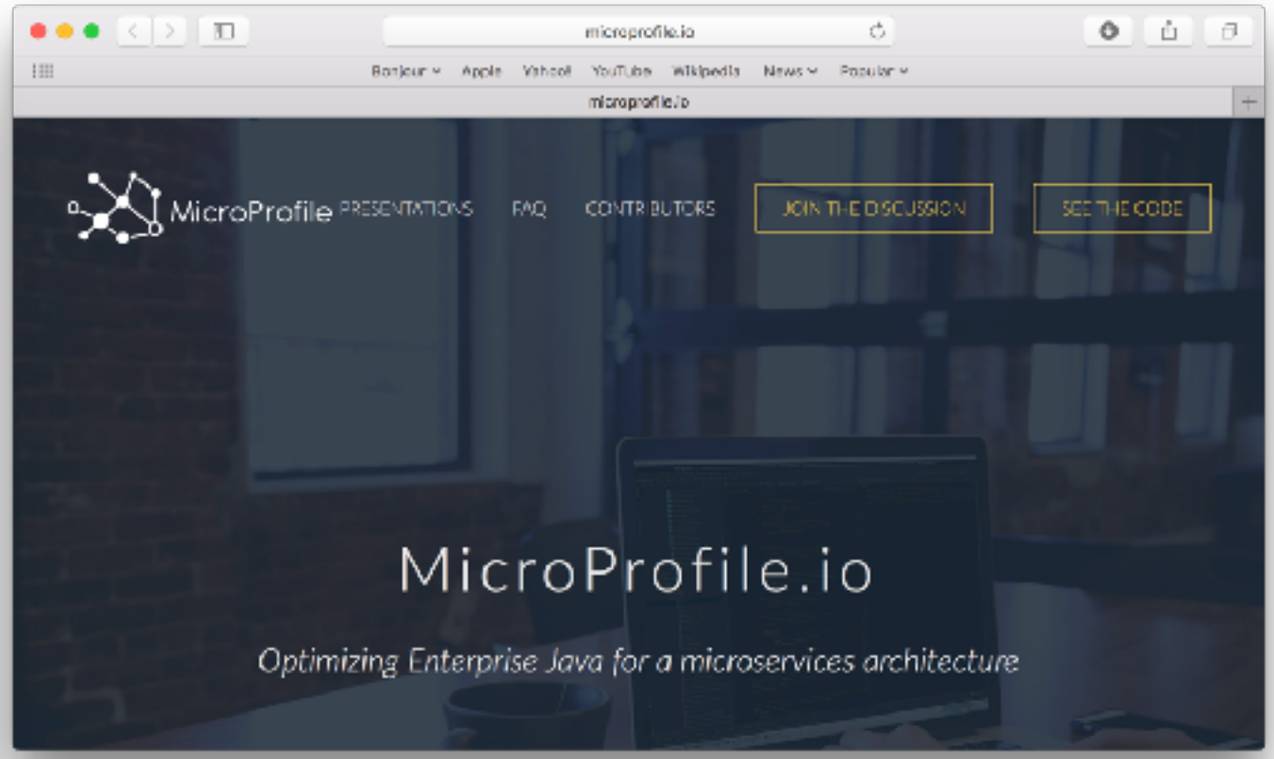
- TomEE
  http://tomee.apache.org/downloads.html

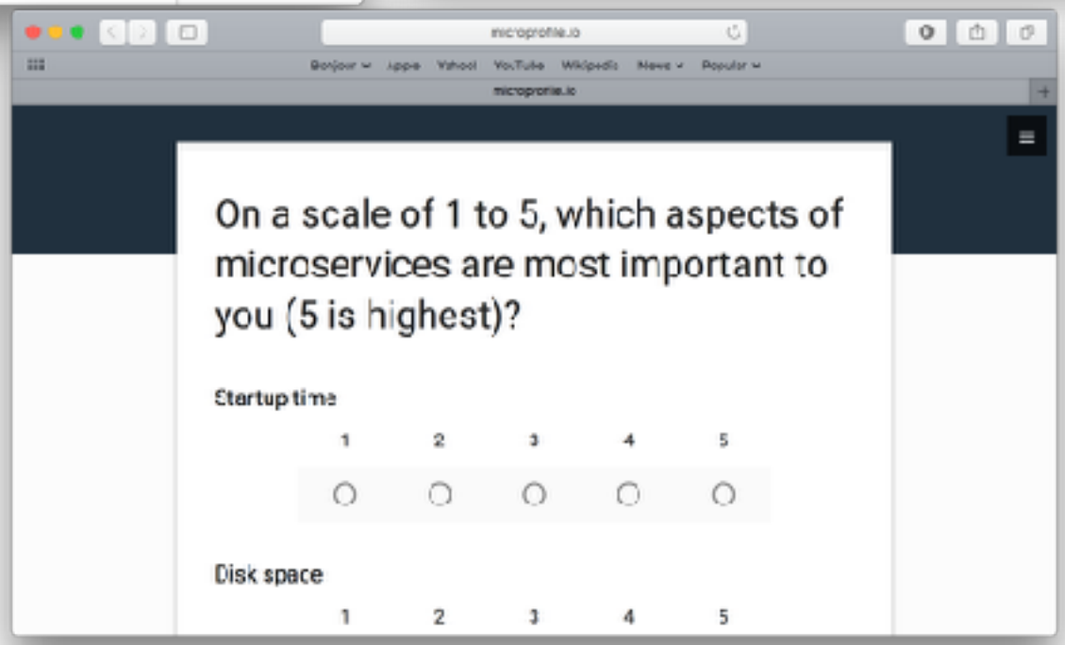
- KumuluzEE
  http://ee.kumuluz.com/generator/

- It is YOU who will be shaping the future of MicroProfile.

- Just type http://microprofile.io into your browser and fill up the survey.



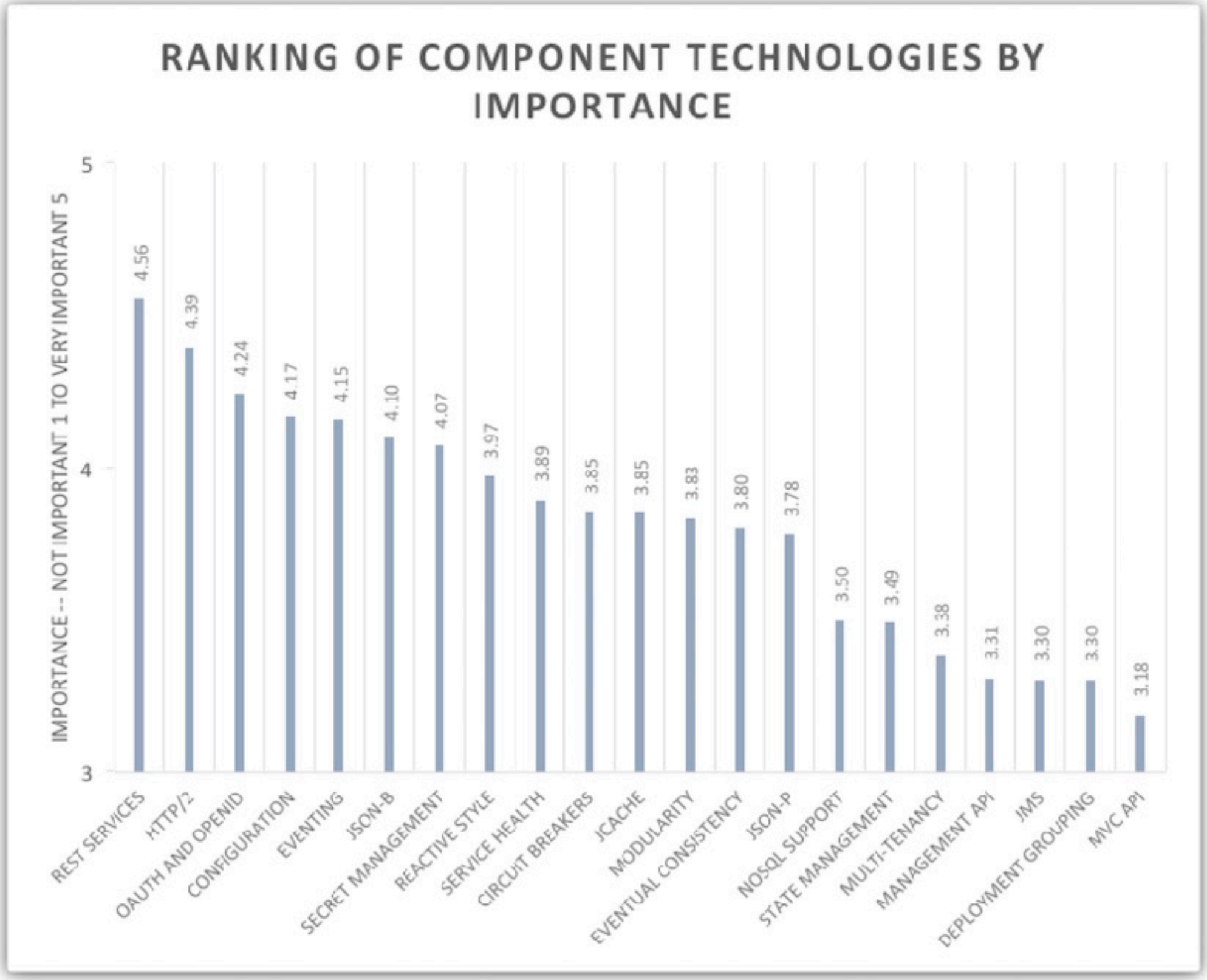




@mertcal

payara

- 1693 person ranked the importance of 21 different technologies that exists on Java EE8 Roadmap.



Thank You

MicroServices Visualised