JEC Overview

JEC

JavaScript Entreprise Container

Last update: 09/18/2017

What is JEC?

JEC stands for JavaScript Entreprise Container.

A JavaScript Standard For Entreprises:

- JEC is defined by its specification
- the specification defines APIs and their interactions

JEC applications are run on an application server, which handle transactions, security, scalability, concurrency and management of the components it is deploying.

TypeScript Integration:

- JEC APIs and applications are written in TypeScript
- TypeScript could become an abstration layer for any Web technologies

JEC is an Open-source Project.

Why JEC?

Node.js as starting point of the project.

JEC aims to fill the gap between JAVA and Node.js ecosystems.

Entreprises need guarantees to cross the line:

- industrialization
- migration costs
- technical support
- quality / stability
- performances vs. maintainability
- long term vision
- flexibility

JEC fulfills all conditions to help companies move to Node.js.

Node.js Without JEC

Many problems make companies reluctant to migrate from JAVA to Node.js:

- **Unstable API**
  → APIs change frequently with some regression issues
- **Development time**
  → you basically need to write everything from scratch
- **Immaturity of tooling**
  → 3rd party packages are still of poor quality or are not documented
- **Hard to make things fault-tolerant**
  → lack of tools to make systems reliable and fault-tolerant
- **Growing demand for experienced professionals**
  → server side JavaScript programming requires strong skills

JEC is the only efficient solution to solve all of these issues!

Made For Developers

JAVA EE is the model used to initially design JEC.

Benefits:

- largely adopted standard
- well structured and proven
- portability

Disadvantages:

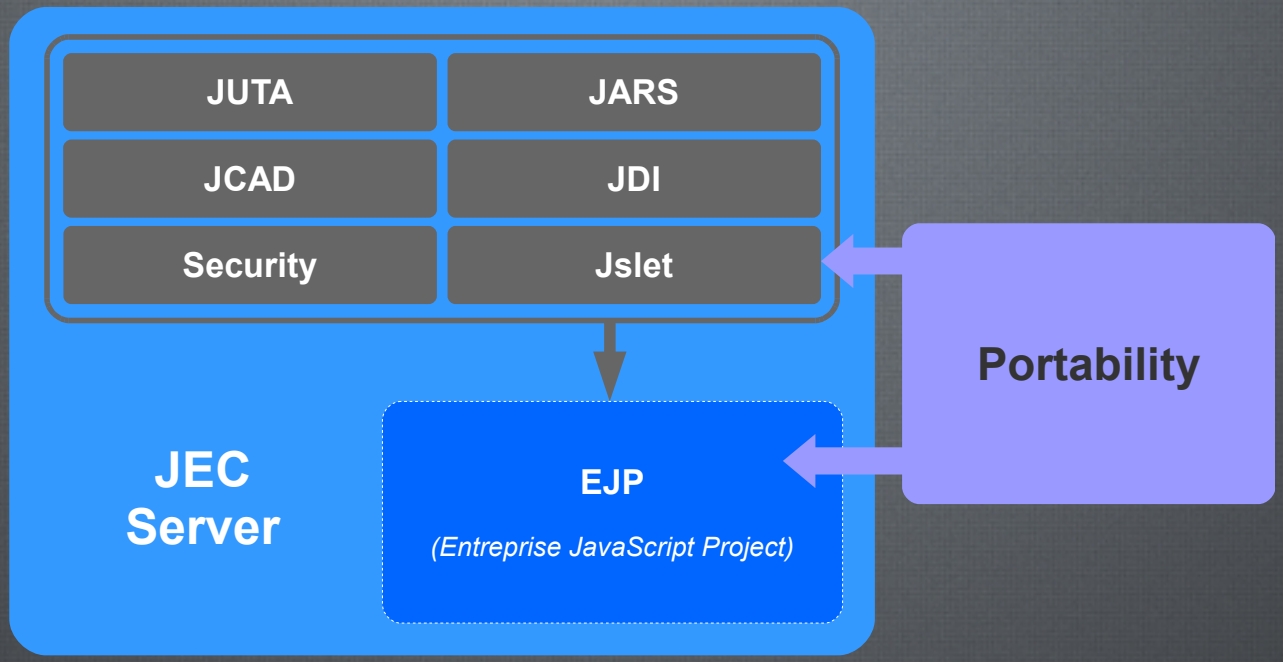
- complexity
- microservices paradigm issues
- stateful vs. stateless architecture

JEC's philosophy:

- keep benefices, remove disadvantages
- developer-centric

Portability

JEC inherits JAVA EE portability principles:



The facts:

- JAVA ecosystems abandon portability to adapt to micro-environments
- JEC has been designed to create and deploy portable micro-applications

Entreprise JavaScript Projects

JEC applications are structured through Entreprise JavaScript Projects (*EJPs*).

EJPs allow to:

- standardize jslet architecture
- share and deploy projects
- embed 3rd party modules

EJPs are:

- familiar to JAVA developers
- compatible with integrated solutions (archetypes, PaaS...)

EJP specification is not finalized yet:

- community feedbacks matter
- configuration is faster than autowiring
- monolithic vs. open architectures

JEC Ecosystem

JEC is defined by its specification.

The specification defines APIs and their interactions:

- The jslet specification defines a set of APIs to service mainly HTTP requests.
- JARS provides support for annotations that make it easy for developers to build RESTful web services.
- The JavaScript Dependency Injection (*JDI*) framework provides the ability to inject components into a JEC application in a typesafe way.
- JUTA is a simple API to write repeatable tests over common JavaScript unit testing frameworks.

Each API comes with a standard implementation

- GlassCat is the default JEC implementation
- Sandcat is the default JARS implementation
- Tiger is the default JUTA implementation
- Bobcat is the default JDI implementation

Rapid App Prototyping

JEC produces content from scratch with rapid results.

JEC has been designed for productivity:

- annotations hide the complexity of configuration
- TypeScript is the unique programing language you need
- Visual Studio Code Editor integration is optimal

JEC default implementation ships with built-in tools:

- EJP archetypes allow to create every kind of projects
- everything can be managed by using the Command Line Interface (*CLI*)

GlassCat focuses on ease-of-use:

- server start and stop immediately
- server configuration is human-readable
- you can emulate container-based microservice APIs