Microservizi: runtime, metodi, pattern ed ultime novità

Ugo Landini - Solution Architect

Giuseppe Bonocore - Solution Architect
“With great power, there must also come—great responsibility.”

—SPIDERMAN (STAN LEE)
It is about the Journey

- Re-Org to DevOps
- Self-Service, On-Demand, Elastic, Infrastructure as Code (Cloud)
- Automation: Puppet, Chef, Ansible and/or Kubernetes
- CI & CD Deployment Pipeline
- Advanced Deployment Techniques
- Silicon Valley DotCom Startup
- Microservices

#RedHatOSD
Microservices play a major role in many organizations today. The movement gained momentum with the seminal article by James Lewis and Martin Fowler, followed by Sam Newman's book and numerous talks and articles by ThoughtWorkers, folks from Netflix, Google, and many others. Microservices quickly made it to the Trial ring on the
You Must Be This Tall

1. Self-Service, on-demand, elastic infrastructure as code
   (how many days/weeks to provision a new VM?)
2. Dev vs Ops
   (who is on the pager for production app outage?)
3. Automation
   (phoenix vs snowflake?)
4. CI & CD
5. Deployment Pipeline

http://martinfowler.com/bliki/MicroservicePrerequisites.html
Maturing The Application Lifecycle

Monolith Java EE Lifecycle
Maturing The Application Lifecycle

Monolith Java EE Lifecycle
Fast Moving Java EE Monolith
Maturing The Application Lifecycle

Monolith Lifecycle
Fast Moving Java EE Monolith
Java EE Microservices
Break things down (organizations, teams, IT systems, etc) down into smaller pieces for greater parallelization and autonomy and focus on reducing time to value.
What’s the difference?

- Same ideas, new technologies (which will evolve in the future)
- But now, we should be able to bring a new feature in production in a few minutes
Microservice Principles/Characteristics

1. Deployment **Independence** - updates to an individual microservice have no negative impact to any other component of the system. Optimized for **Replacement**

2. Organized around **business** capabilities

3. **Products** not Projects

4. **API** Focused

5. **Smart** endpoints and dumb pipes

6. Decentralized Governance

7. Decentralized Data Management

8. Infrastructure Automation (infrastructure as code)

9. Design for failure

10. Evolutionary Design
Microservice ≈ Container

https://www.flickr.com/photos/63975655@N07/
DOCCHER!
Container (no more) = Docker

- LXC Initial release
  - Mar '13
- Open Container Initiative
  - Jun '15
- Docker Initial release
  - Aug '08
- Moby
  - Jun '17
- PodMan
  - Sep '17
- Buildah
  - Aug '17
- CRI-O
  - CRI-O
- CNCF
  - Apr '17
- Cloud Native Computing Foundation
THE CLOUD-NATIVE APP DEV CHALLENGE
41% of enterprises see non-integrated tools as an inhibitor to container adoption.

24% of time spent building and maintaining developer environments.

Source: Cloud Development Survey 2017 - Evans Data Corp
Modern, cloud-native application runtimes and an opinionated developer experience for organizations that are moving beyond 3-tier architectures and embracing cloud-native application development.
Red Hat Hybrid Cloud Development Platform

APPLICATION RUNTIMES
- Red Hat OpenShift Application Runtimes
- OpenJDK
- Red Hat AMQ Broker
- Red Hat Data Grid

INTEGRATION
- Red Hat FUSE
- Red Hat AMQ
- Red Hat 3Scale API Management

AUTOMATION
- Red Hat Process Automation Manager
- Red Hat Decision Manager

APPLICATION SERVICES
- Service Mesh
- Enterprise Kubernetes

RED HAT ENTERPRISE LINUX ECOSYSTEM
Hardware, Virtualization, Cloud and Service Provider Certifications

Infra Monitoring
App Monitoring
#RedHatOSD
Openshift Tested Integration

- 100+ defects fixed between every upstream Kubernetes and commercial OpenShift release
- 140+ combinations of common products tested with every *minor* OpenShift release, incl. Storage drivers, networking, database images, ...
- Tested for performance & scalability, security and reliability

https://access.redhat.com/articles/2176281
Ok, so it’s (also) about being lighter?

Theoretically, yes. But, beware:

- A simple ReST service deployed in EAP used $\frac{1}{5}$ of the memory used by Spring Boot under load and was $2x$ faster!

<table>
<thead>
<tr>
<th>Runtime (framework)</th>
<th>Boot time server only</th>
<th>Boot time including app deployment</th>
<th>Memory usage without load</th>
<th>Memory usage under load</th>
<th>Measured throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBoss EAP (Java EE)</td>
<td>2 - 3 sec</td>
<td>3 sec</td>
<td>40 MB</td>
<td>200 - 400 MB</td>
<td>23K req/sec</td>
</tr>
<tr>
<td>JBoss EAP (Spring)</td>
<td>2 - 3 sec</td>
<td>7 sec</td>
<td>40 MB</td>
<td>500 - 700 MB</td>
<td>9K req/sec</td>
</tr>
<tr>
<td>JBoss WS/Tomcat (Spring)</td>
<td>0 - 1 sec</td>
<td>8 sec</td>
<td>40 MB</td>
<td>0.5 - 1.5 GB</td>
<td>8K req/sec</td>
</tr>
<tr>
<td>Fat JAR (Spring Boot)</td>
<td>N/A</td>
<td>3 sec</td>
<td>30 MB</td>
<td>0.5 - 2.0 GB</td>
<td>11K req/sec</td>
</tr>
<tr>
<td>Fat JAR (Thorntail)</td>
<td>1-2 sec</td>
<td>5 sec</td>
<td>30 MB</td>
<td>250 - 350 MB</td>
<td>27K req/sec</td>
</tr>
</tbody>
</table>
# Decision Points For Selecting The Runtimes

<table>
<thead>
<tr>
<th>Selection Consideration</th>
<th>Project Type</th>
<th>Framework Pref</th>
<th>Learning Effort</th>
<th>Deployment Pkg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtimes</td>
<td>Cloud Native (new)</td>
<td>Cloud Enable (existing)</td>
<td>Java EE</td>
<td>Non-Java EE</td>
</tr>
<tr>
<td>EAP</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Thorntail</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Vert.x</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Node.js</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Tomcat</td>
<td>+</td>
<td>Spring Boot</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Application Runtimes

**LAUNCH SERVICE**

- **JAVA SE** OPENJDK
- **JAVA EE** JBOSS EAP
- **JAVA WEB** JBOSS WS
- **SERVERLESS** CLOUD FUNCTION*
- **SPRING** SPRING BOOT
- **JAVASCRIPT** NODE.JS
- **MICROPROFILE** THORNTAIL
- **REACTIVE** VERT.X
- **DISTRIBUTED DATA GRID**
- **MESSAGING AMQ BROKER**

**SECURITY**

- Pre-configured Missions and Boosters
- Integration with RH Developer, CI/CD tools, Security Services
- Optimized for OpenShift / Kubernetes Services
- Available Application Migration Toolkit
- Python, Go and .Net also supported by Red Hat (with a different SLA)

**RED HAT SSO**

- Fully supported JDK

Facilitate cloud native app development ON THE HYBRID CLOUD:

- ✓ Faster getting started
- ✓ Simplify container dev
- ✓ Automate DevOps
- ✓ Standardize tools/processes

*Coming Soon
LAUNCH
Continuous application delivery, built and deployed on OpenShift.

LAUNCH YOUR PROJECT

Supported Runtimes

**THORNTAIL**
Thorntail offers an innovative approach to packaging and running Java EE applications by packaging them with just enough of the server runtime to "Java out" your application.

Learn more ▶

**VERT.X**
Eclipse Vertx is a tool-kit for building reactive applications on the JVM.

Learn more ▶

**Spring Boot**
Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

**RED HAT FUSE**
Red Hat® FUSE is a lightweight, flexible integration platform that uses Apache Camel at its core.

Learn more ▶

**node**
Node.js® is a JavaScript runtime built on Google's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

Learn more ▶
Openshift-do: A CLI For Developers

Openshift-DO ("odo") is a new CLI plugin for OpenShift 3.9+ that is tailored for developer syntax and workflows.

Goal is to make it simple for a developer to create an app, add components (like a database) and expose it without needing to know Kubernetes.

In tech preview now.

```
> odo create wildfly backend
Component ‘backend’ was created.
To push source code to the component run ‘odo push’

> odo push
Pushing changes to component: backend

> odo storage create backend-store --path /data --size 100M
Added storage backend-store to backend

> odo create php frontend
Component ‘frontend’ was created.
To push source code to the component run ‘odo push’

> odo push
Pushing changes to component: frontend

> odo url create frontend - http://frontend-myproject.192.168.99.100.nip.io

> odo watch
Waiting for something to change in /Users/tomas/odo/frontend
```
The books you’ll need to read

- The DevOps Handbook
- Migrating to Microservice Databases
- Building Microservices
- Release It!
- Microservices for Java Developers
- Continuous Delivery
- The Phoenix Project
- Building Reactive Microservices in Java
- Domain-Driven Design

#RedHatOSD
GRAZIE PER L’ATTENZIONE

Ugo Landini - Solution Architect

Giuseppe Bonocore - Solution Architect