

Dropwizard & Spring

The perfect Java REST server stack

Jacek Furmankiewicz
Enterprise Architect
PROS, Houston, TX

Dropwizard

What is it and why is it so exciting?

Dropwizard

Created by Coda Hale @ Yammer

<http://dropwizard.codahale.com>

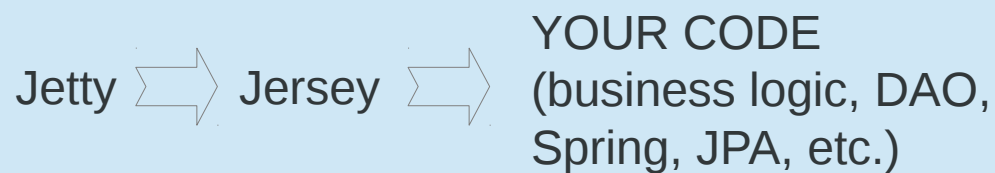
Best-of-breed Java libraries

- Embedded Jetty (no WAR, no deployment to external servlet container)
- JAX-RS (Jersey)
- JSON (Jackson)
- Logging (Logback / SLF4J)
- SLA Tracking (Metrics)
- Hibernate Validators
- etc.

Embedded Jetty

- Restart your code in seconds
- No WAR to recompile
- No WAR to redeploy
- Debug from your IDE (you have a *main()*), no need to attach to separate process
- No need to share heap and GC issues with other apps running in the same servlet container
- Total process isolation (one mis-behaving WAR cannot affect others as much)

Anatomy of a Dropwizard app



Multiple apps on same box

pid 1843

Jetty \Rightarrow Jersey \Rightarrow APP 1

Own JVM
4 GB heap

pid 1407

Jetty \Rightarrow Jersey \Rightarrow APP 2

Own JVM
4 GB heap

pid 1976

Jetty \Rightarrow Jersey \Rightarrow APP 3

Own JVM
4 GB heap

etc.

Operations-friendly

- Opens 2 HTTP ports: one for public APIs (i.e. your REST services), one for admin APIs (e.g. run GC, refresh internal caches, etc)
- Admin port can be closed off on the firewall and inaccessible to outside world
- Health Check APIs to allow easy monitoring from external tools like Nagios
- @Timed annotation on any single REST API allows to track its SLA using Metrics library

Ease of deployment

- Dropwizard apps can be easily compiled into a single JAR with all dependencies (e.g. using One-Jar)
- Your entire app consists of two files: the YAML config + single JAR
- Trivial to run from command line:

```
java -server myapp.jar server myapp.yml
```

Ease of deployment (part 2)

- Can be wrapped in an RPM or DEB to install on Linux clusters
- Can be registered as a Linux daemon, e.g.
 - *sudo service myapp start*
 - *sudo service myapp stop*

Dropwizard and Spring

Adding dependency injection

Spring DI

- Create Spring context first and wire all your components
- Query the Spring context and pull out all the parts Dropwizard cares about: JAX-RS resource classes, JAX-RS @Provider classes, Dropwizard HealthCheck and Task classes, etc.
- Register each of them with the Dropwizard runtime

Spring DI (part 2)

- Link the embedded Jetty with the Spring context, which makes it think it is running within a regular servlet container

```
environment.addServletListeners(new SpringContextLoaderListener(springContext));
```

Spring DI (part 3)

See the example application on github:

|

| <https://github.com/jacek99/dropwizard-spring-di-security-onejar-example>

|

Dropwizard and Spring

Adding Spring Security

Spring Security

- Add a Spring Security XML config file to your context @Configuration class, e.g.

@Configuration

@ImportResource("classpath:myapp-security.xml")

*@ComponentScan(basePackageClasses =
MyAppSpringConfiguration.class)*

public class MyAppSpringConfiguration {}

Spring Security (part 2)

- Unlike the rest of Spring, Spring Security does not support Java @Configuration yet, hence an XML file is required.
- This should be the ONLY XML file you should need to integrate Spring.
- Everything else in Spring can be done via pure Java @Configuration classes

Dropwizard & Spring

Q&A

Jacek Furmankiewicz
Enterprise Architect
PROS, Houston, TX