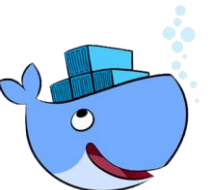


# TechDay#6 – Docker & Spring Boot

Abril 2017    Madrid – Barcelona – Jerez

# Contenido de la Sesión

- Introducción a Spring Boot (30 min.)
- Desarrollo de un servicio con Spring Boot.
- Introducción a Docker (30 min.)
- Encapsulación y ejecución del servicio desarrollado en un contenedor Docker.





# Introducción

Spring Boot es el módulo de Spring encargado de **facilitar los desarrollos**

- Tendencia -> Nuevas aplicaciones desarrolladas en el framework de Spring
- Reduce mucho los tiempos de desarrollo
- Incrementa la productividad
- Facilita disponer de funcionalidad out-of-the-box dependiente de la naturaleza del proyecto -> Mínimo esfuerzo para usar librerías de terceros
- No sólo lo pueden utilizar principiantes de Spring
- Aplicaciones autónomas (stand-alone) e industrializadas (production-grade)

**NO** es una herramienta de **prototipado**

- No crea arquetipos
- Facilita la creación de un fichero de configuración

Permite la **ejecución** de aplicaciones

- JAR : java -jar -> "Script de Spring"
- WAR tradicionales

Proporciona una serie de características **NO funcionales** comunes a la gran mayoría de proyectos

- Seguridad
- Servidores Embebidos
- Métricas
- Seguridad
- Etc.



# Introducción

## Objetivos

Spring Boot reduce los tiempos de desarrollos, test unitarios / integración y facilita el desarrollo de aplicaciones listas para ejecutarse en producción

**Nota :** Reduce mucho los tiempos respecto a hacerlo de forma “tradicional” con el framework Spring

- Evita la configuración “completa” con ficheros XML
- Mantiene el uso de anotaciones
- Evita escribir muchas declaraciones de importación
- Proporciona valores predeterminados
- Etc.



Principal Queja de  
Spring : Excesiva  
Configuración XML



Configuración de  
aplicaciones “legacy”

### Problema :

Requiere MUCHO TIEMPO y es muy COSTOSO el convertir aplicaciones heredadas a Spring Boot

- Se aconseja utilizar con aplicaciones nuevas



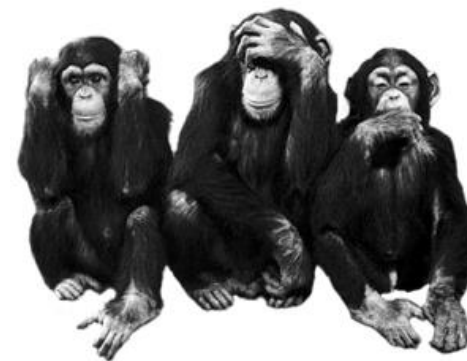
# Introducción

Al eliminar ciertas actividades “muy pesadas” facilita en cierta manera los desarrollos

- Permite centrarse más en la implementación
- Reduce compilación, control de dependencias, despliegue, etc.

Proporciona ciertas **facilidades de integración** :

- Ecosistema Spring : Spring Security, Spring JDBC, Spring ORM, Spring Data, etc.
- Testing
- JMS
- NoSQL : Redis, MongoDB, Solr, ElasticSearch, etc
- Plantillas : Thymeleaf , etc.
- Etc



“Pensamos menos pero no por ello es más fácil”



## Conceptos Generales

### Conocimientos “básicos” de Spring

Se requiere tener claros los siguientes conceptos :

- Bean
- Inyección de dependencias (DI)
- Contenedor de IoC
- ApplicationContext
- Autowiring
- Configuración : XML , Anotaciones y Java



“Un framework para dominarlos a todos”



# Starters

Normalmente tenemos que definir **muchas dependencias** en nuestras aplicaciones

Tarea muy pesada para el desarrollador -> Labor de CSI para descubrir las “cosas raras”

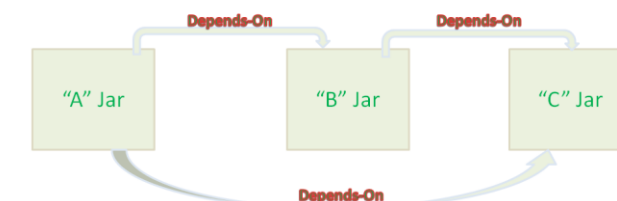
Incrementa el tamaño de los ficheros de configuración

```

14<parent>
15<groupId>org.springframework.boot</groupId>
16<artifactId>spring-boot-starter-parent</artifactId>
17<version>1.3.1.RELEASE</version>
18<relativePath/> <!-- lookup parent from repository -->
19</parent>
20
21<properties>
22<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
23<java.version>1.8</java.version>
24</properties>
25
26<dependencies>
27<dependency>
28<groupId>org.springframework.boot</groupId>
29<artifactId>spring-boot-starter-thymeleaf</artifactId>
30</dependency>
31
32<dependency>
33<groupId>org.springframework.boot</groupId>
34<artifactId>spring-boot-starter-test</artifactId>
35<scope>test</scope>
36</dependency>
37</dependencies>
38
39</build>
  
```

Los starter o inicializadores se encargan principalmente de **combinar un grupo de dependencias comunes/afines en dependencias individuales**

- Pueden ser utilizadas con herramientas de construcción automáticas como :  
Maven o Gradle



Objetivo : **Agrega varias dependencias en una única dependencia**

- Reduce la lista de dependencias de un proyecto



vs.



# Introducción

## Lista de Starter

Starter	Composición
<b>spring-boot-starter</b>	Núcleo de Spring Boot que incluye el soporte a la autoconfiguración, logging y YAML
<b>spring-boot-starter-actuator</b>	Soporte a la monitorización y gestión de una aplicación en producción
<b>spring-boot-starter-batch</b>	Soporte a Spring Batch e incluye HSQLDB
<b>spring-boot-starter-data-jpa</b>	Soporte a “Java Persistence API” incluyendo spring-data-jpa, spring-orm y Hibernate
<b>spring-boot-starter-data-rest</b>	Soporte para publica repositorios Spring Data como REST con spring-data-rest-webmvc
<b>spring-boot-starter-jdbc</b>	Soporte para JDBC
<b>spring-boot-starter-security</b>	Soporte para spring-security
<b>spring-boot-starter-web</b>	Soporte para full-stack web , Tomcat y spring-mvc
<b>spring-boot-starter-jetty</b>	Soporte para servidor Jetty
<b>spring-boot-starter-log4j</b>	Soporte para logging log4j
<b>spring-boot-starter-logging</b>	Soporte para logging logback
...	





# Introducción

## Lista de Starter : Spring Boot Providing

Starter	Composición
<b>spring-boot-starter</b>	spring-boot, spring-boot-autoconfigure, spring-boot-starter-logging
<b>spring-boot-starter-tomcat</b>	tomcat-embed-core, tomcat-embed-logging-juli
<b>spring-boot-starter-web</b>	spring-boot-starter, spring-boot-starter-tomcat, jackson-databind, spring-web, spring-webmvc
<b>spring-boot-starter-batch</b>	spring-boot-starter, spring-core, spring-batch-core, spring-jdbc, HSQLDB
<b>spring-boot-starter-data-jpa</b>	spring-boot-starter, spring-bootstarter-jdbc, spring-boot-starteraop, spring-core, Hibernate EntityManager, spring-orm, spring-data-jpa, springaspects
.....	



## AutoConfigurador

Spring Boot AutoConfigurator es un submódulo de Spring Boot que tiene por objetivo **reducir la configuración de Spring**

- Permite NO definir un fichero de configuración XML
- Ninguna o una mínima configuración por anotaciones

Los starter saben que beans “iniciales” son necesarios

- Son los encargados de disparar la autoconfiguración

Propone realizar toda la configuración de Spring basada en Java

- Utilización de la anotación @Configuration

Ejemplo :

- Una aplicación basada en Spring MVC requiere definir : views, view resolvers, etc.
- Spring Boot usará el starter adecuado : “spring-boot-starter-web”
- Con dicho starter los elementos requeridos por Spring MVC serán configurados automáticamente

Importante :

La gran mayoría de las **críticas** que recibe el framework de Spring están relacionadas con la **cantidad de código relacionado con configuración** que requiere el desarrollo de cualquier aplicación basada en este framework

- Problema de configuración : XML y Anotaciones



# Conceptos

## Configuración basada en Java

Uso de anotaciones :

- **@Configuration** : Identifica las clases de configuración de beans -> También se registra como bean
- **@Bean** : Identifica los beans implicados en una configuración -> Permite darles nombre



# Conceptos

## Ejemplo de Clase de Configuración



*@Configuration*

```
public class AcmeConfiguration {
```

*@Bean*

```
    public AccountDao accountDao() {  
        AccountDaoInMemoryImpl dao = new AccountDaoInMemoryImpl();  
        //Incluir aqui las dependencias del DAO  
        return dao;  
    }
```

*@Bean*

```
    public AccountService accountService() {  
        AccountServiceImpl service = new AccountServiceImpl();  
        //Incluir aqui las dependencias del Servicio  
        service.setAccountDao(accountDao());  
        return service;  
    }
```





## Conceptos

### Anotación @EnableAutoConfiguration

Anotación que permite activar todas las opciones de configuración automática de una aplicación

- Utilizada en las aplicaciones basadas en Spring Boot
- Configuración de Spring en base a las dependencias JAR que tiene añadidas
- Ejemplo : spring-boot-starter-web añade Spring MVC y Tomcat -  
> Realiza su configuración

Mucha de estas configuraciones vienen dadas principalmente por el classpath y por el uso de un fichero de configuración específico : application.properties

Se pueden desactivar partes de la configuración mediante el parámetro : exclude

Se puede ver un informe de la autoconfiguración realizada si se ejecuta en modo debug

Se coloca en la clase principal y define la base de “paquetes de búsqueda”

- Evita tener que configurar basePackage en @ComponentScan

Ejemplo :

```
@Configuration
@EnableAutoConfiguration
public class Application {
    ...
}
```



# Conceptos

## Anotación @SpringBootApplication

Spring Boot reduce la configuración basada en anotaciones.

Esta anotación a nivel de clase hace que Spring Boot AutoConfigurator añada todas las anotaciones requeridas sobre la clase ByteCode

Definición oficial :

`@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoConfiguration`

```
@Target(value=TYPE)
@Retention(value=RUNTIME)
@Documented
@Inherited
@Configuration
@EnableAutoConfiguration
@ComponentScan
public @interface SpringBootApplication
```

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Proporciona "alias" para poder personalizar los atributos de las anotaciones que lo componen



# Propiedades

Mucha de las configuraciones de los elementos de Spring Boot vienen dadas principalmente por el **classpath** y por el uso de un **fichero de configuración específico**

- Externalización de configuración
- Los ficheros se suelen localizar en el área de recursos de un proyecto (Por ejemplo : src/main/resources)

## Fichero de configuración basado en propiedades

- Suele ser el sistema por defecto -> application.properties
- Clásica estructura : clave – valor
- Ubicación determinada : classpath, /config, etc.

## Fichero de configuración basado en YAML

- Formato de serialización de datos que simplifica XML –
- También se le conoce como YML
- Utiliza el fichero por defecto : application.yml

### Orden de sobreescritura de valores

1. Argumentos línea de comandos
2. Propiedades SPRING\_APPLICATION\_JSON
3. Atributos JNDI
4. Propiedades sistema Java
5. Variables entorno OS
6. Valores random: RandomValuePropertySource
7. Properties específico-perfil externo al JAR
8. Properties específico-perfil interno al JAR
9. Properties específico externo al JAR
10. Properties específico interno al JAR
11. @PropertySource en @Configuration
12. SpringApplication.setDefaultProperties



# Propiedades

## Variables de Entorno

- Argumentos de la JVM
- Ejemplo de variable con el tipo de entorno : `-DtargetPlatform=dev`
- Ejemplo Maven : `mvn spring-boot:run -Drun.jvmArguments='-Dserver.port=8081'`
- Se recuperará mediante el uso de la anotación `@Value` haciendo referencia al argumento : `@Value("${param1}")`

## Argumentos de línea de comandos

- Se indica en la ejecución de la aplicación
- Ejemplo : `java -jar <aplicación>.jar -param1="test"`
- Se recuperará mediante el uso de la anotación `@Value` haciendo referencia al argumento : `@Value("${param1}")`

## Ejemplo

- **Argumento Línea de comandos**

```
java -jar ejemplo.jar --spring.application.json='{ "username": "admin" }'
```

- **SPRING\_APPLICATION\_JSON**

```
SPRING_APPLICATION_JSON='{ "datasource": { "username": "admin" } }'  
java -jar ejemplo.jar
```

- **Variable del sistema**

```
java -Dspring.application.json='{ "username": "admin" }' -jar ejemplo.jar
```

- **RandomValuePropertySource** (application.properties)

```
ejemplo.password=${random.value}
```

```
ejemplo.id=${random.int(15)}
```





# Propiedades

## Anotación @ConfigurationProperties

Anotación que permite configurar los beans automáticamente con las propiedades de configuración

- Se le puede indicar un tipo de propiedades en concreto mediante el parámetro : **prefix**

Ejemplo :

```
@ConfigurationProperties(prefix="datasource")
public class DataSourceSetting {
    @NotNull
    private String jndi;
}
```

### application.properties

```
datasource.jndi=java:comp/env/miDatasource
datasource.user=admin
...
```

### application.yml

```
datasource:
  jndi:java:comp/env/miDatasource
  user:admin
...
```

### Importante :

Existen propiedades específicas para hacer referencia al fichero de configuración

- spring.config.name : Identifica el nombre del fichero
  - spring.config.location : Identifica la ubicación
- Orden de búsqueda : classpath:,classpath:/config,file:file:config/,...



# Perfiles

En función del **entorno de ejecución** se pueden necesitar unos beans u otros

Un clásico ejemplo de este aspecto es la definición de un : datasource

- La definición de este bean suele ser diferente en DEV y PRO
- En DEV suele hacer referencias a bases de datos embebidas (Por ejemplo : H2)
- En PRO suele hacer referencias a recursos JNDI



Una buena práctica para esto suele venir dada por la **creación de ficheros de configuración diferentes** que son **específicos del entorno**

- Por ejemplo : datasource-dev.xml y datasource-pro.xml
- Para Spring Boot ocurre lo mismo con su fichero : application.properties
- Utilizará la convención : application-{profile}.properties
- El perfil por defecto se considera : default



# Perfiles

## Soluciones :

- Disponer de ficheros INDEPENDIENTES de configuración específicos de cada entorno
- Ejemplo : application-dev.properties y application-prod.properties
- Disponer de una configuración de ficheros parametrizable según el entorno :
- Ejemplo : application-{profile}.properties

Requiere establecer el parámetro en la JVM : `-Dspring.profiles.active=<VALOR>`

Invocación mediante la propiedad de Spring : `spring.profiles.active`

- Fichero de configuración con la propiedad -> Ejemplo : `--spring.profiles.active=dev,hsqldb`
- Línea de comandos -> Ejemplo `--spring.profiles.active=dev,hsqldb`



# Perfiles

Se permite el uso de Placeholders en properties

- Dentro del fichero de propiedades Utilizando propiedades de maven

Spring proporciona soporte para ayudar a **establecer perfiles** en los beans declarados en los fichero XML y  
Anotaciones

- Ficheros XML : Utilización de la etiqueta <beans> con el atributo profile
- Anotaciones : Uso de la anotación @Profile()
- Utilización de la clase ConfigurableEnvironment

Lo habitual es aplicar perfiles sobre @Component y @Configuration

- Propiedad “spring.profiles.active” en application.properties





# Perfiles

Ejemplo :

```
System.getProperty("spring.profiles.active")
```

```
<beans profile="prod">  
  <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">  
    <property name="jndiName" value="java:comp/env/jdbc/DS" />  
  </bean>  
</beans>
```

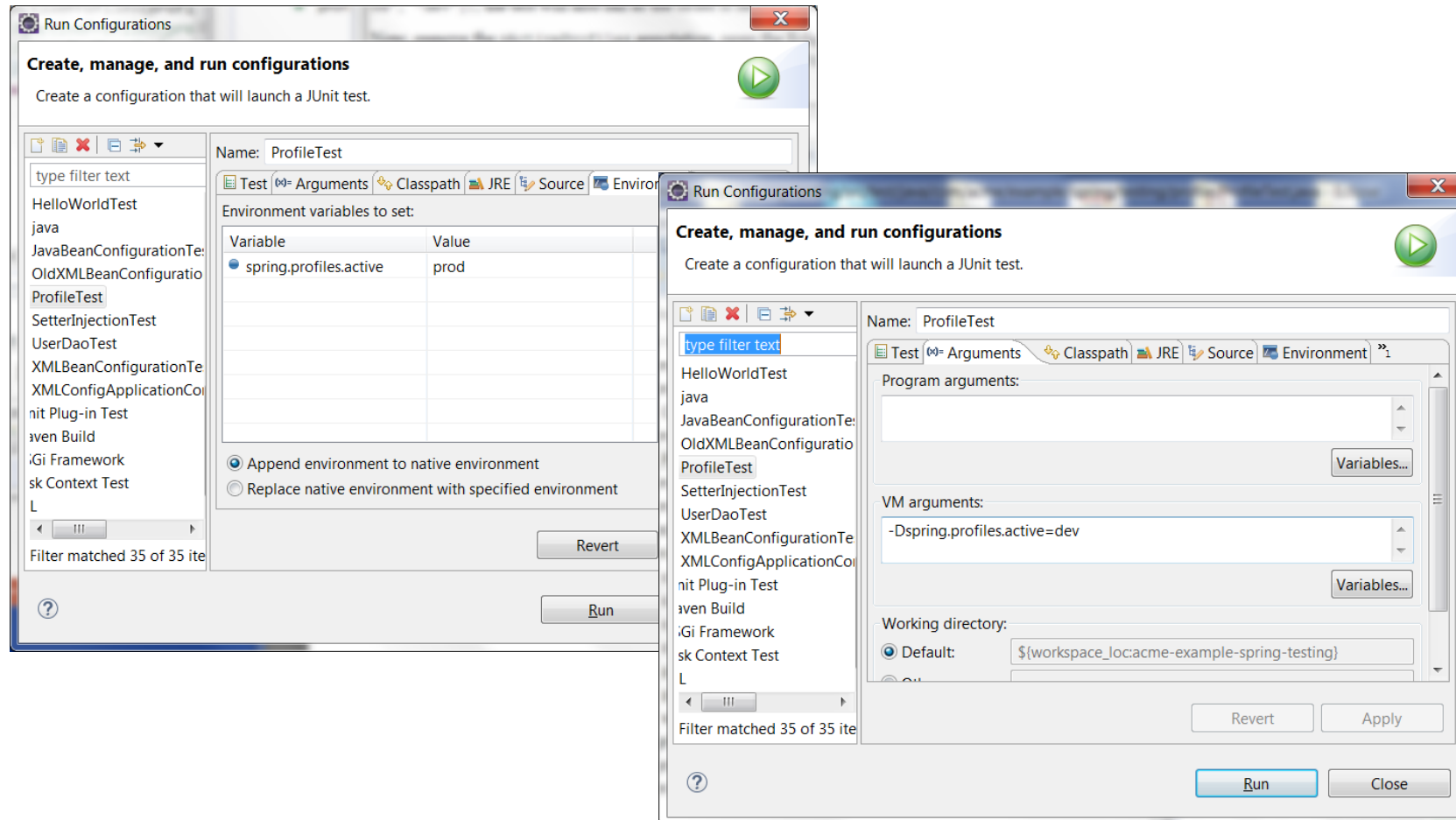
```
@Bean  
@Profile("dev")  
public Example example() {  
  ...  
}
```

```
ConfigurableEnvironment environment = applicationContext.getEnvironment();  
environment.setActiveProfiles("dev");
```



# Perfiles

## Configuración entorno mediante : spring.profiles.active



# Perfiles

## Ejemplo de Uso de Perfiles

...

```
<!-- Configure properties files -->
<context:property-placeholder location="classpath*:properties/database.properties" ignore-unresolvable="true" order="1"/>

<!-- Configure datasource -->
<jee:jndi-lookup id="dataSource" jndi-name="${database.dataSource.jndi.name}" />

<!-- Configure database -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<beans profile="development">

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${database.dataSource.driverClassName}" />
        <property name="url" value="${database.dataSource.url}" />
        <property name="username" value="${database.dataSource.username}" />
        <property name="password" value="${database.dataSource.password}" />
    </bean>
</beans>
```

### Ejemplo de Configuración Maven:

- Project -> Properties
- Seleccionar Maven
- En el campo de texto "Active Maven Profiles" introducir : development



# Servidores Embebidos

Proporciona **soporte** para el uso de **contenedores embebidos**

- Tomcat , Jetty o Undertow

Se requiere indicar su uso mediante los “**starters**” adecuados

Se puede adaptar la configuración del servidor embebido

- *Fichero de configuración : Mediante propiedades (Por ejemplo : server.port, etc)*
- *Por defecto escuchan las peticiones HTTP en el puerto 8080*
- *server.port : Puerto de escucha de las peticiones HTTP*
- *server.address : Dirección utilizada*
- *server.sesión.timeout : Timeout de la sesión*
- *Parámetros*
- *Implementando EmbeddedServletContainerCustomizer*
- *Programáticamente*





## Servidores Embebidos

**Spring Boot** proporciona **soporte** a diferentes bases de datos en memoria :

- **H2** : [www.h2database.com](http://www.h2database.com)
- **HSQLDB** : <http://hsqldb.org/>
- **DERBY** : <https://db.apache.org/derby/>



**Importante** : No proporcionan almacenamiento persistente “in-memory”

Para su uso requiere únicamente incorporar su dependencia “starter”



# Persistencia

Spring Boot proporciona **soporte a diferentes tipos de persistencia**

Para su uso se requiere únicamente incorporar su dependencia “starter”

## •Spring-JDBC

- Incluir “starter” : spring-boot-starter-jdbc
- Configura automáticamente el bean : JdbcTemplate

## •Spring-Data-JPA

- Incluir “starter” : spring-boot-starter-data-jpa
- Escanea todos los paquetes en busca de las clases Entity y los repositorios de Spring Data



# Logging

**Spring Boot** utiliza **por defecto Commons Logging** para el logging interno pero facilita utilizar diferentes implementaciones :

- *Otras implementaciones : Java Util Logging, Log4J, Log4J2 ,Logback (Opción por defecto si se usan Starter POM)*

Por defecto cada logger se **configura** para usar la **consola** de salida con posibilidad de utilizar un **fichero opcional de salida**

- *Configurar un fichero de salida con propiedades : logging.file o logging.path*

Se permite **establecer un formato** de Log

Gestión de **niveles de log**

- Mediante propiedad : logging.level.\*=<LEVEL> donde LEVEL es : TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF

- El root también puede ser configurado mediante logging.level.root

Se puede ejecutar una aplicación en modo debug

- Ejecutando como parámetro : java -jar ejemplo.jar -debug

- Especificar una propiedad (application.properties) : debug=true



# Logging

## Personalización de la configuración de log

Logging	Personalización
JDK (Java Util Logging)	Logging.properties
Logback	logback-spring.xml, logback-spring.groovy, logback.xml o logback.groovy
Log4j	log4j-spring.properties, log4jspring.xml, log4j.properties o log4j.xml
Log4j2	log4j2-spring.xml o log4j2.xml
....	

Se recomienda siempre que se pueda usar las variantes que contienen –spring

Nota : Si se usan las ubicaciones estándar de las configuraciones, Spring no puede tener el control completo sobre el log

Existen ciertas propiedades de Spring que han sido convertidas a propiedades del sistema

Por ejemplo : logging.file -> LOG\_FILE

### Importante:

- Activar el modo “debug” cuando se ejecuta una aplicación Spring Boot -> java -jar <nombre>.jar --debug
- Especificar la propiedad : debug=true en el fichero application.properties



# Seguridad

## Características

- Seguridad básica HTTP para todos los “endpoints”
- *Ignorar acceso a localizaciones de recursos estáticos*
- *Proporcionar AuthenticationManager*
- *Uso de OAuth2*
- *Securización de Actuadores*
- *Etc.*



# Seguridad

## Integración con Spring Security

**Spring Security** es el módulo de Spring encargado de **proporcionar** a las **aplicaciones web** multiusuario que utilizan Spring sus **requerimientos de seguridad**

- *Proporciona soporte a los servicios de autenticación y autorización*
- *Ready-to-use*





# Testing

Spring Boot proporciona **soporte para test unitarios y de integración**

Para su uso se requiere únicamente incorporar su dependencia “starter” : **spring-boot-starter-test**

Esta dependencia incluye las dependencias :

- Spring Test*
- JUnit*
- Hamcrest*
- Mockito*

Se utilizará `@SpringApplicationConfiguration` como alternativa a `@ContextConfiguration`

- Así quedará configurado para mantener las características de Spring Boot*

Para test de integración

- Uso de anotación `@IntegrationTest` que permite levantar la aplicación para escuchar en los puertos por defecto*
- Se pueden configurar los puertos para que no estén ocupados*
- Uso de la clases `TestRestTemplate` que es una subclase de `RestTemplate`*



# Testing

## Ejemplo de configuración vía Java



```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(AcmeConfiguration.class)
@WebIntegrationTest({"server.port=0", "management.port=0"})
public class ExampleIntegrationTest {

    ...

}
```



# Actuator

**Spring Boot Actuator** es un **submódulo** de Spring Boot que **proporciona** muchas características , pero principalmente destaca por :

- Proporciona la **gestión de EndPoints** a las aplicaciones Spring Boot (tiene elementos por defecto)
- Proporciona **métricas / diferentes tipo de información** para las aplicaciones Spring Boot ejecutadas
- Tipos : health, metrics, info, dump, shutdown, trace, etc
- Permite monitorizar, auditar y administrar aplicaciones

Para su uso se requiere incorporar su dependencia “starter” : **spring-boot-starter-actuator**

Objeto : **Expone diferentes tipos de información sobre las aplicaciones en ejecución**

- A lo anterior se le suele denominar : production-ready
- Suele ser un buen punto de partida para tener una solución de monitorización de producción

**No requiere implementación** por parte del programador



## Endpoints

## Actuator

Permiten **monitorizar la aplicación** y en algunos casos incluso **interactuar con ella**

Spring Boot por defecto proporciona muchos endpoint, pero estos también se pueden ampliar -> Incorporar propios (custom)

Algunos Endpoints :

- /health** : Muestra información sobre la salud de la aplicación
- /info** : Muestra información arbitraria de la aplicación
- /metrics** : Muestra información de métricas
- /trace** : Muestra información de seguimiento
- /shutdown** : Permite tirar la aplicación
- ....

Existen muchos más en la **documentación oficial**



# Spring Boot Initializr con Web Interface

URL : <http://start.spring.io>

Initializr

The screenshot displays the Spring Initializr web interface. At the top, it says "SPRING INITIALIZR" and "Bootstrap your application now". Below this, there are two main sections: "Project metadata" and "Project dependencies".

**Project metadata:**

- Group:
- Artifact:
- Name:
- Description:
- Package Name:
- Type:
- Packaging:
- Java Version:
- Language:
- Spring Boot Version:

**Project dependencies:**

- Core: ☐ Security, ☐ AOP, ☐ AspectJ (LTA), ☐ Biometric LTA
- Web: ☐ Web, ☐ Websocket, ☐ WS, ☐ Jersey (API-RG)

Below these sections, there is a "Generate a" dropdown menu set to "Maven Project" and a "with Spring Boot" dropdown menu set to "1.3.1".

**Project Metadata:**

Artifact coordinates

Group:

Artifact:

**Dependencies:**

Add Spring Boot Starters and dependencies to your application

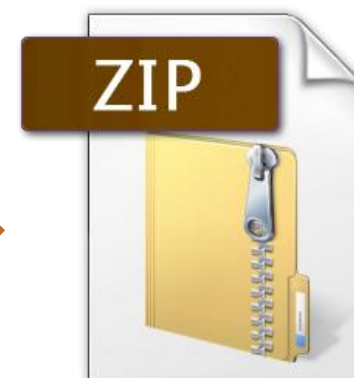
Search for dependencies:

Selected Starters:

At the bottom, there is a green button labeled "Generate Project" with a keyboard shortcut "alt + g".

Don't know what to look for? Want more options? [Switch to the full version.](#)

start.spring.io is powered by [Spring Initializr](#) and [Pivotal Web Services](#)



# Configuración

## Tipos de ejecución

### IDE

- Lanzar como una aplicación Java la clase : Application

### Maven

- **Directa** : mvn spring-boot:run

### JAR (Se requiere crear un JAR ejecutable)

- Plugin de empaquetado que sea compatible con Spring Boot : spring-boot-maven-plugin

```
java -jar target/<ejemplo>-0.0.1-SNAPSHOT.jar
```

#### Importante :

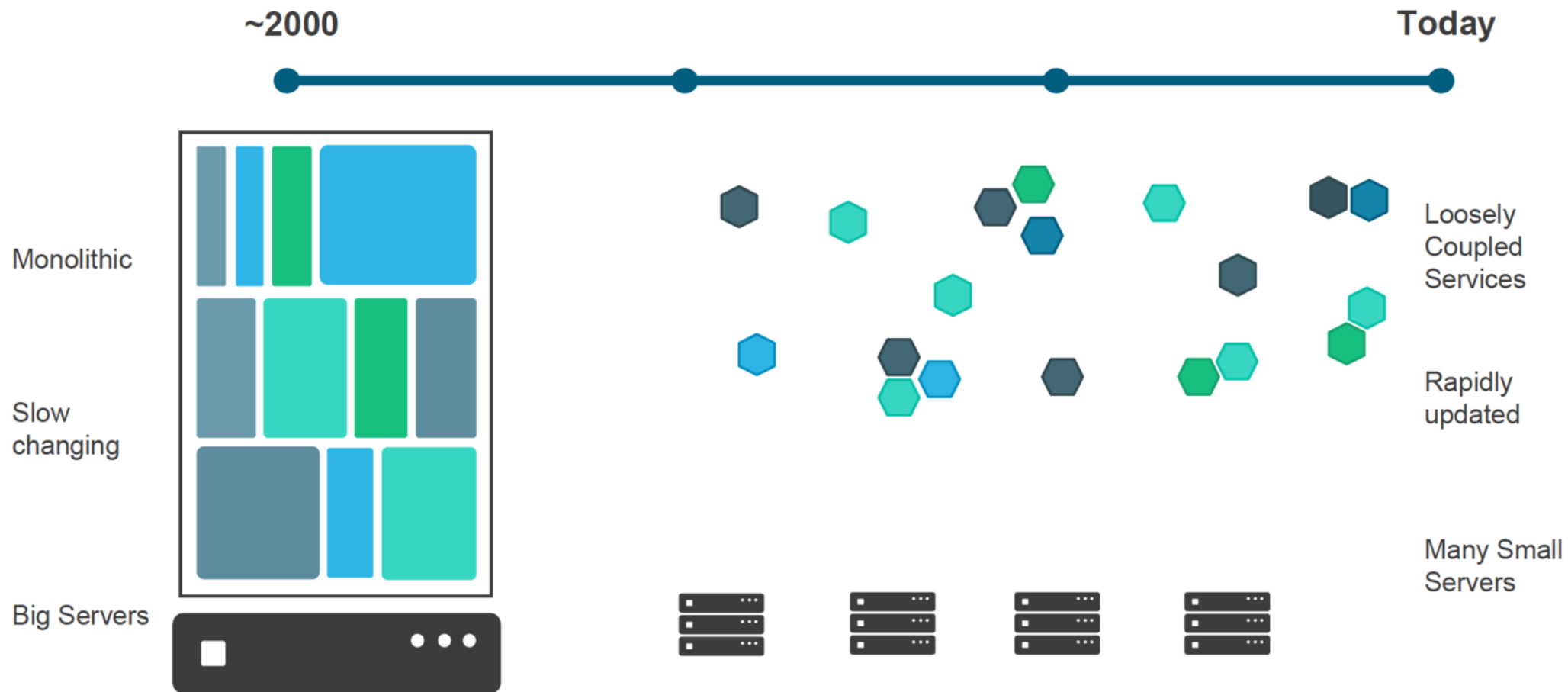
Java NO proporciona ninguna forma estándar para cargar archivos JAR anidados. Esto es un problema si se quiere construir la aplicación de forma “autocontenida”  
Concepto “uber” (El más grande) jar : Un único archivo con todas las clases y todos los JARs necesarios



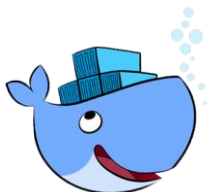
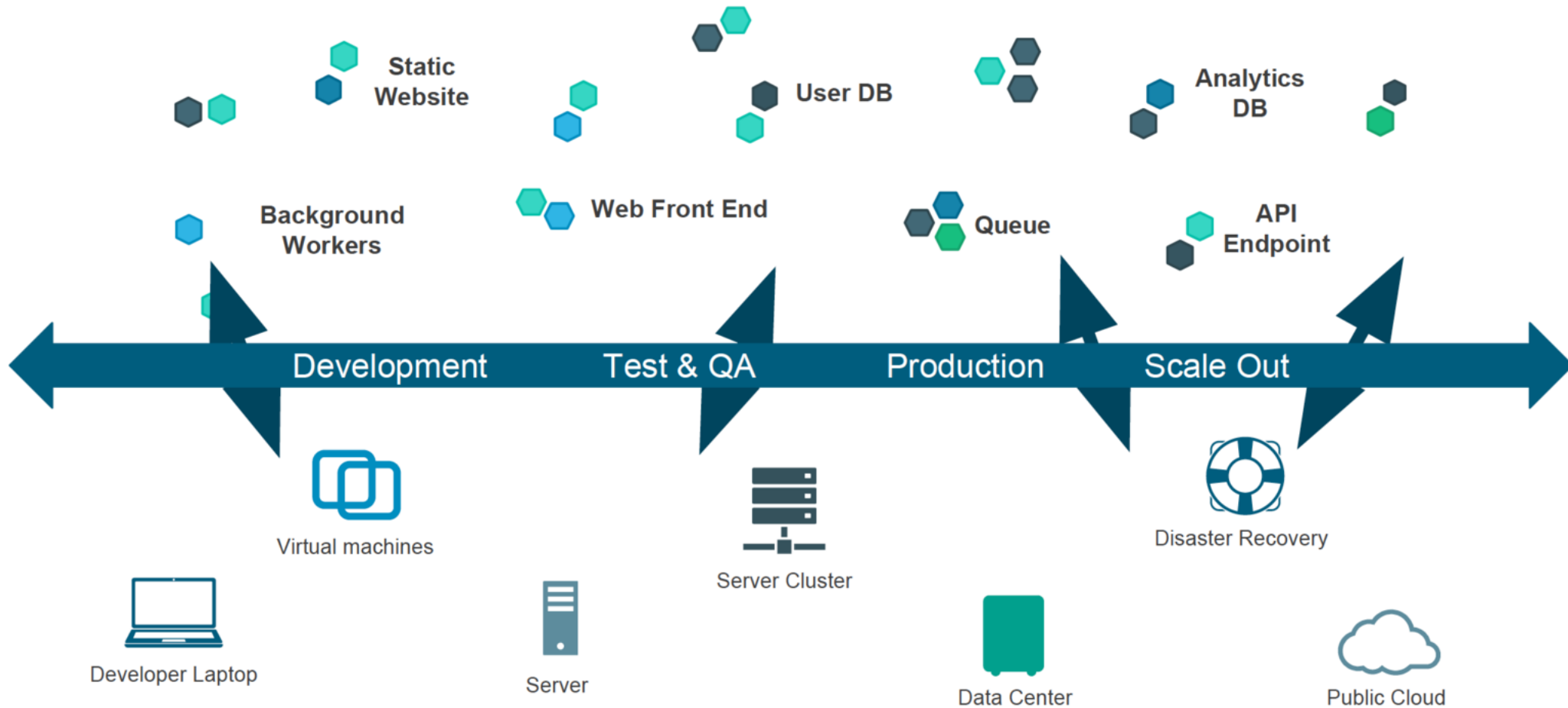




# Applications are changing



# The challenge: new matrix from hell



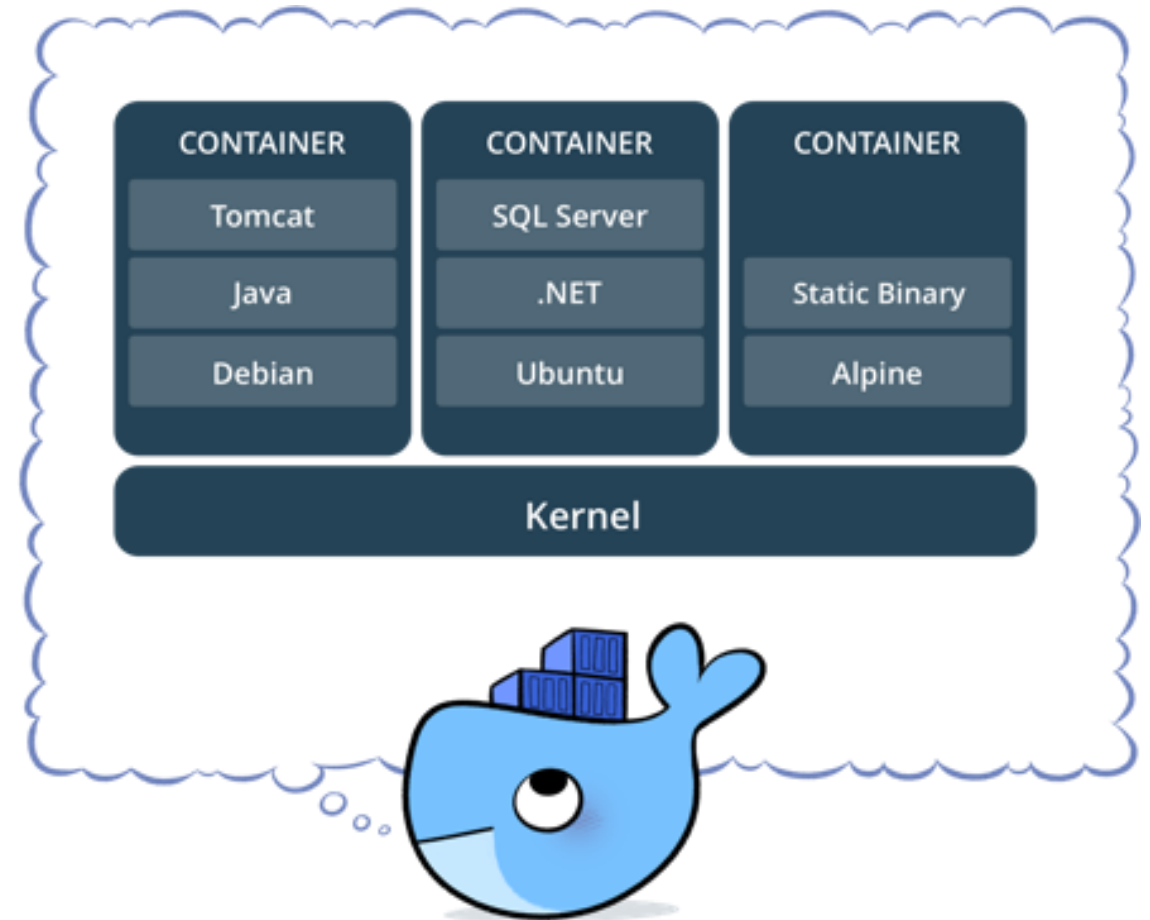


# Linux Containers

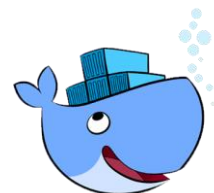
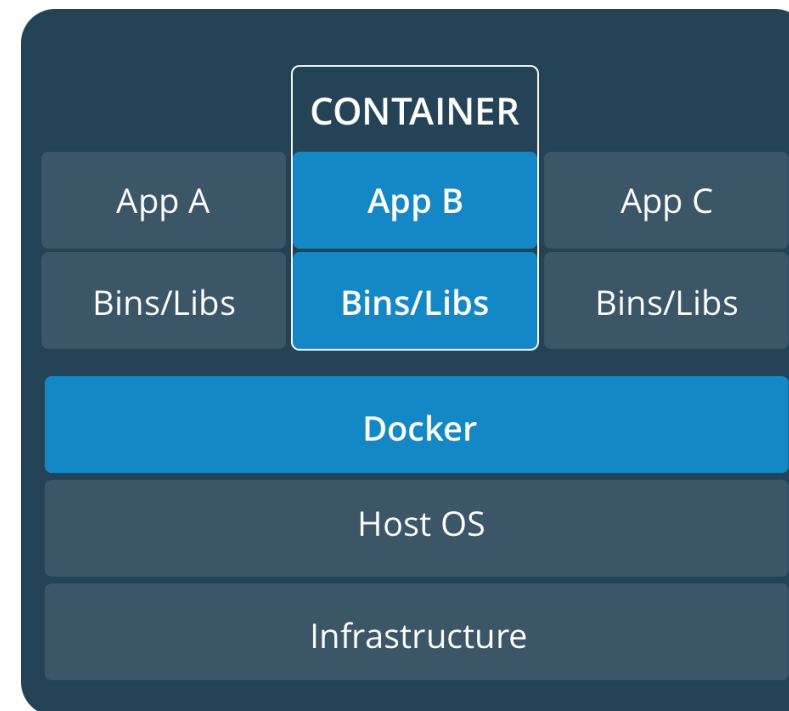
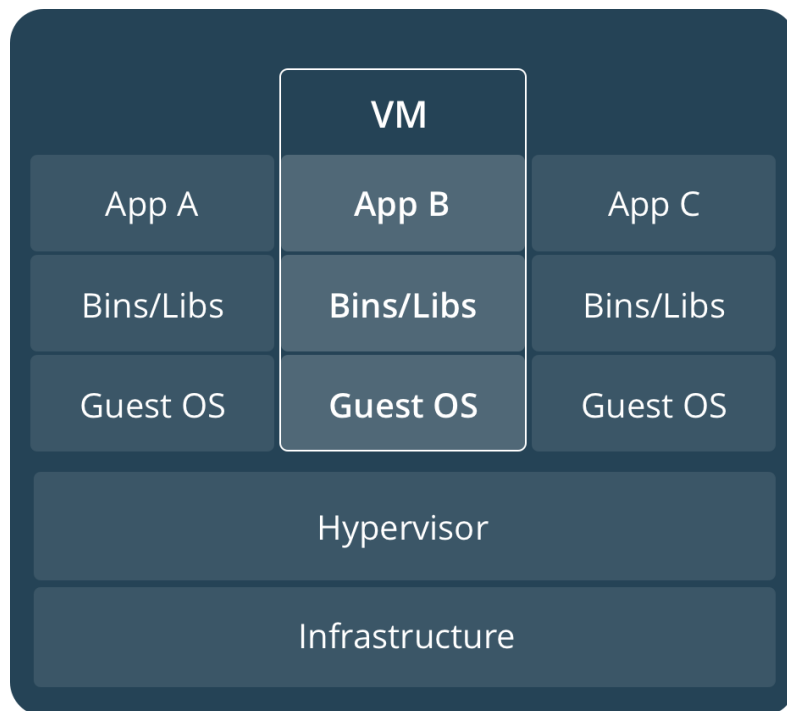
Docker is based on **LXC** (Linux Containers) which allows it to isolate containers from each other.

**LXC** use mainly two **Linux Kernel** features to achieve it :

- Namespaces (Isolation of resources)
- Cgroups (Isolation of resource usage, as CPU/RAM)



# Virtual Machines vs Containers



## Virtual Machines vs Containers (II)

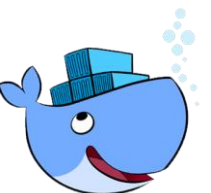
Containers share resources with the host OS, which makes them an order of magnitude more efficient.

Containers can be started and stopped in a fraction of a second.

Applications running in containers incur little to no overhead to applications running natively on the host OS.

The portability of containers has the potential to eliminate a whole class of bugs caused by subtle changes in the running environment and missing libraries

— it could even put an end to the age-old developer refrain of  
“but it works on my local machine!”

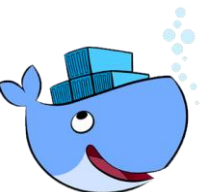




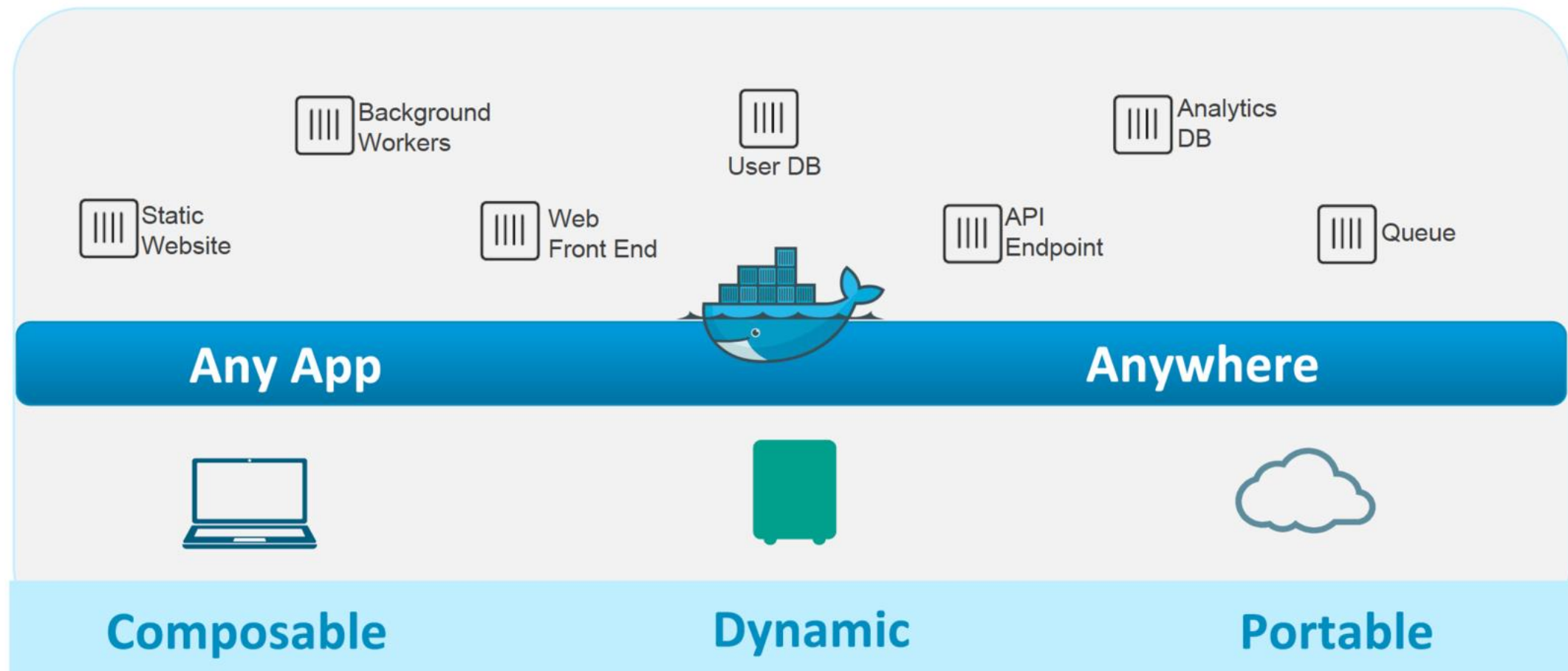
## WHY DO DEVELOPERS CARE?

Build once...(finally) run anywhere

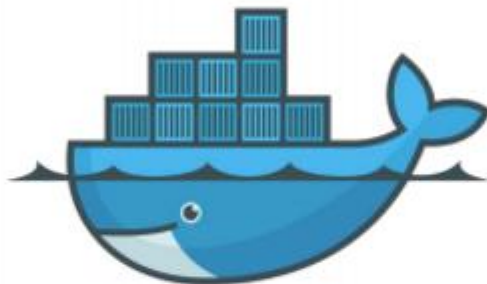
- A clean, safe and portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Automate testing, integration, packaging...anything you can script.
- Cheap, zero-penalty containers to deploy services.
- Instant replay and reset of image snapshots.



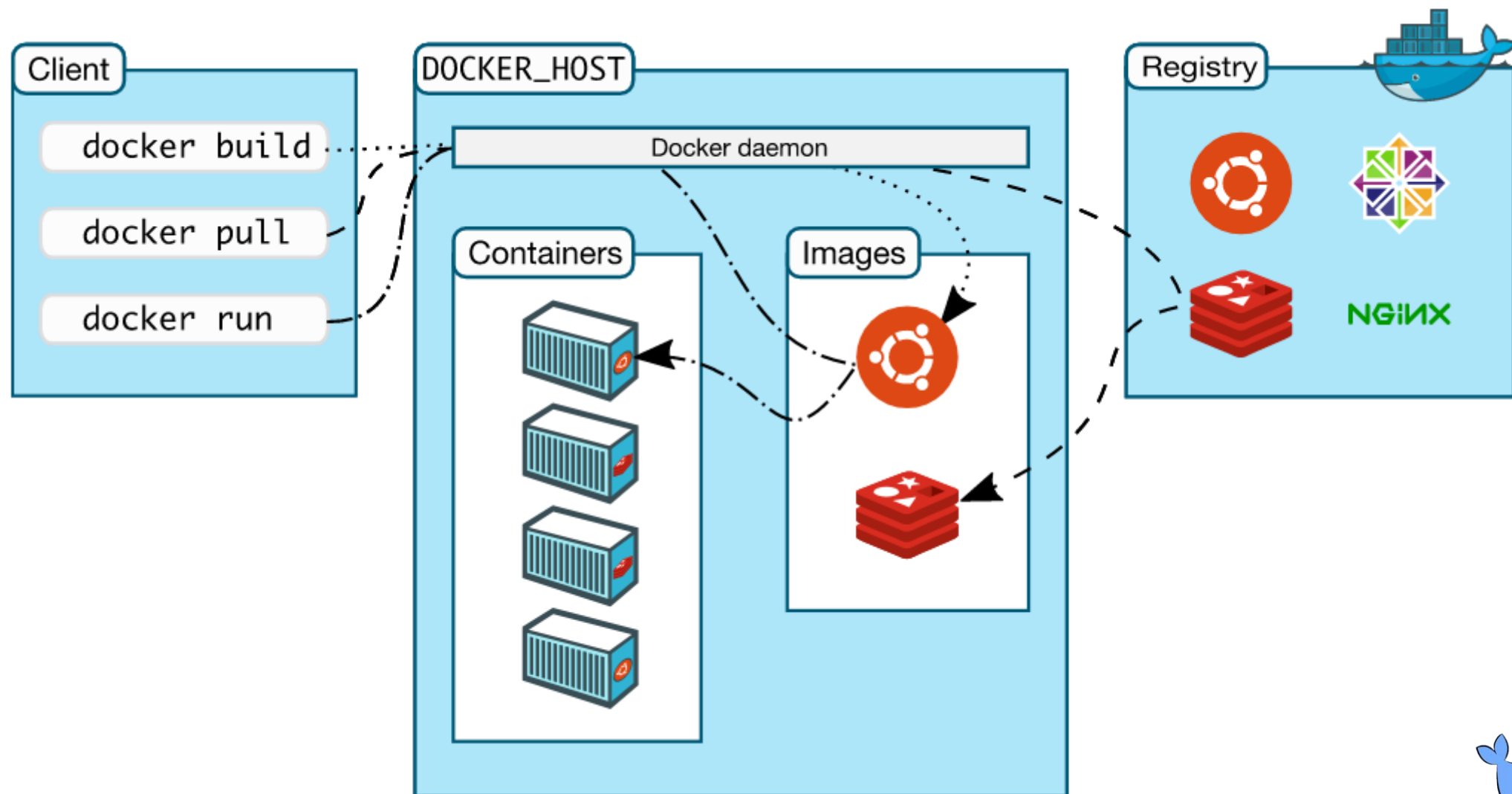
# Dockerized ecosystem



# Docker Ecosystem



# The Docker Architecture



# Anatomy of a Dockerfile

An image is built from a Dockerfile. A file describing how the image is supposed to behave, what it extends from, ...

```
FROM java:8-jre

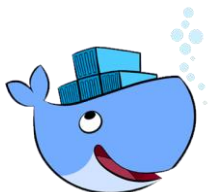
ENV CATALINA_HOME /usr/local/tomcat
ENV PATH $CATALINA_HOME/bin:$PATH
RUN mkdir -p "$CATALINA_HOME"
WORKDIR $CATALINA_HOME

ENV TOMCAT_TGZ_URL https://www.apache.org/tomcat/tomcat-latest.tar.gz

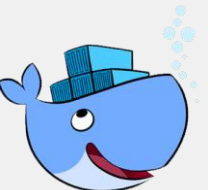
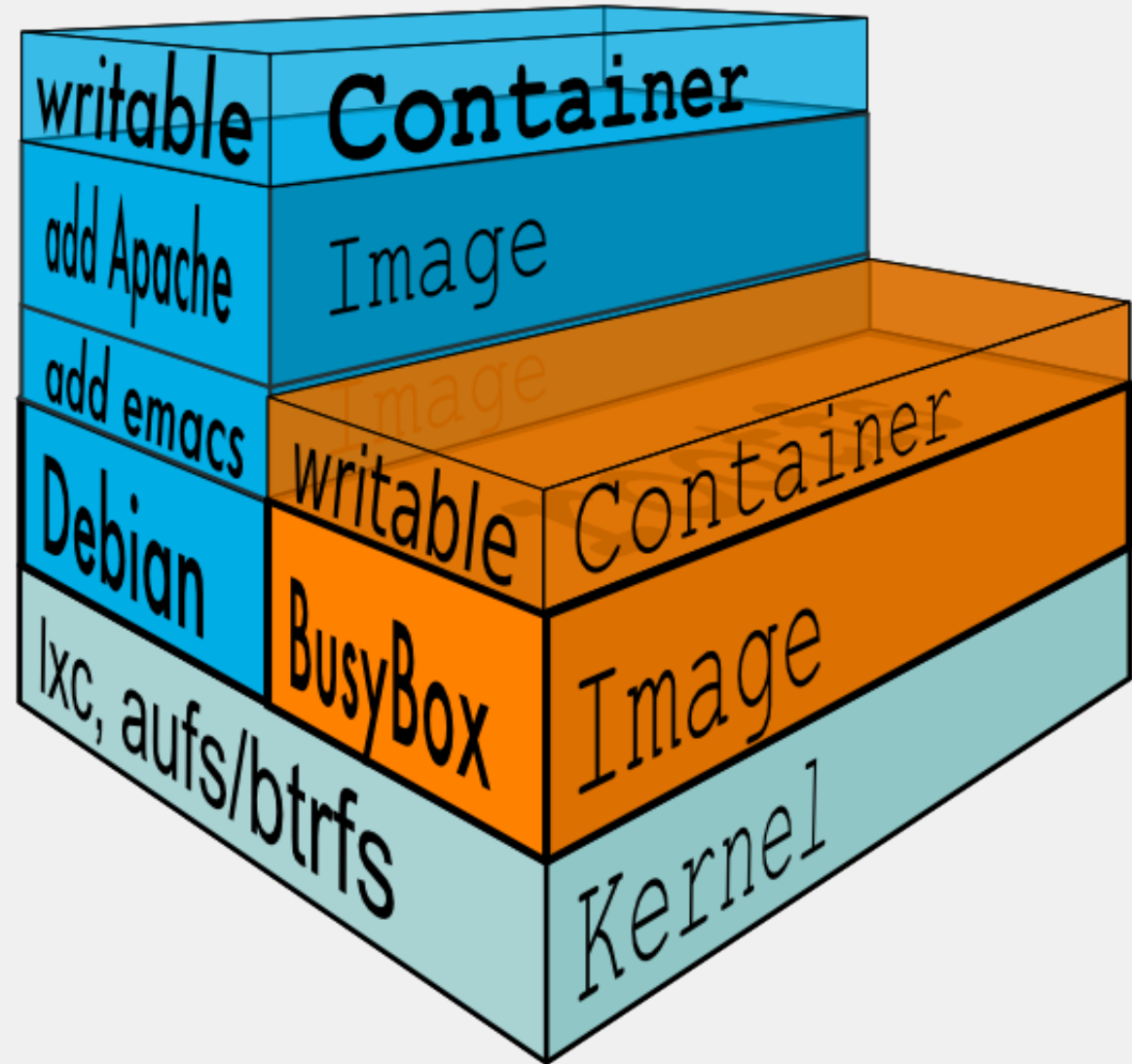
RUN set -x \
    && curl -fSL "$TOMCAT_TGZ_URL" -o tomcat.tar.gz \
    && tar -xvf tomcat.tar.gz \
    && rm tomcat.tar.gz*

EXPOSE 8080

CMD ["catalina.sh", "run"]
```

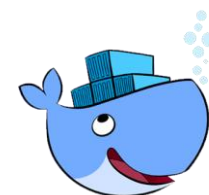


## How images get built





Workshop time!!





# GRACIAS

[www.atsistemas.com](http://www.atsistemas.com)

902 888 902



## **Madrid**

C/Valle de Alcudia.3 Edificio 2,  
planta 1. 28232. Las Rozas, Madrid



## **Barcelona**

Plaça de Catalunya, 21 - 2ª  
08002, Barcelona



## **Cádiz**

Edificio Jerez Parque Empresarial,  
Calle del Desarrollo 2; oficina 12,  
planta 1, 11047, Jerez de la Frontera, Cádiz



## **Zaragoza**

Centro Tecnológico TIC XXI C/Bari, 57  
Plataforma Logística (PLA-ZA),  
50197, Zaragoza



## **A Coruña**

Edificio Mans, Polígono de Pocomaco,  
parcela D22, 15190 A Coruña



## **Palma de Mallorca**

Regus Palma, Gremi de Sabaters, 21,  
Polígono de Son Castello 07009 Palma