Deployment

Packaging

Packaging as JAR

- Default artefact
- Maven or Gradle plugin
  (using spring-boot-starter parent)

```
<packaging>jar</packaging>
…
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

```
$ mvn package

$ java -jar demo.jar
```

Packaging as WAR

- Maven or Gradle plugin
  (using spring-boot-starter parent)
- Deployable in Servlet Container (Jetty, Tomcat,...) and executable from command line

```
<packaging>war</packaging>
…
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

```
$ mvn package

$ java -jar demo.war
```

Hybrid Apps – Running as a JAR or WAR

- Extend app launcher with SpringBootServletInitializer
- Runnable on CLI or Tomcat/Jetty

```
@SpringBootApplication
public class DemoApplication extends
    SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(
        SpringApplicationBuilder application) {
        return application.sources(DemoApplication.class);
    }

    public staticvoid main(String[] args) {
        SpringApplication.run(DemoApplication.class);
    }
}
```

Logging

Logging

- Spring Boot provides default configuration files for four logging frameworks: Logback, Log4j, Log4j2 and java.util.Logging
- Starters use Logback with colour output
- Default log level set to INFO
- Debug output can be easily enabled using --debug
- Log to console by default
- For custom Logging in APP use SLF4j API
- logging.file and logging.path property to enable file
- Logging levels can be customized in application.properties

```
logging.level.com.example=DEBUG
```

More on Properties

YAML instead of Properties

- SnakeYAML dependency
  - Already available with starters
  - File extension .yaml
  - Replacement for application.properties
  - @PropertySource does not support yaml
  - Tabs vs. Spaces

application.yaml

```yaml
spring:
 application:
   name:DemoApplication
server:
 port:9000
```

Properties from CLI Arguments

- Spring Boot style

```
$ java -jar demo.jar --property.name="value"
```

- Java Style

```
$ java -Dproperty.name="value" -jar demo.jar
```

Properties from Environment Variables

- Environments variables are available as properties
- Relaxed conversion from uppercase to java style
  - E.g. JAVA_HOME ->. @Value("${java.home}")

```
$ export NAME=value

$ java -jar demo.jar
```

Randomizing Property Values

- RandomValuePropertySource can randomize values

```
random.number=${random.int}
random.long=${random.long}
random.uuid=${random.uuid}
```

Need a Configuration Server?

- Have a look at Spring Cloud Config!
  - https://cloud.spring.io/spring-cloud-config/

Property Precedence

Config Server	key1=**kiwi** key2=**flamingo** key3=**dolphin**
Command line arguments	key1=apple
System Properties	key2=dragon
System Environment	key1=banana
classpath: application.yml	key1=coconut key2=elephant
classpath: bootstrap.yml	key3=shark

Profiles

@Profile

- Used to limit the availability of @Component or @Configuration
- With one or more parameters
  - @Profile("prod") or @Profile({"dev", "prod"})
- Negation possible
  - @Profile("!dev")

```
@Profile("!dev")
@Configuration
public class PrdConfiguration{…}
```

```
@Service
@Profile("dev")
public class DevEmployeeService implements EmployeeService {

    @Override
    public List<Employee> getAllEmployees() {
        return Arrays.asList(new Employee("carmen"));
    }
}
```

Selecting Profiles

- Command Line

```
$ java -Dspring.profiles.active=dev,staging -jar demo.jar
```

- Property File (application.properties or application.yaml)

```
spring.profiles.active=dev,staging
```

- Programmatically setting profile

```
public static void main(String[]args) {
    SpringApplication app =
        new SpringApplication(DemoApplication.class);
    app.setAdditionalProfiles("dev","staging");
}
```

Profile Specific Configurations

- Application-{profile}.properties
- Loaded the from the same location as application.properties
- Will override default application.properties

- application-staging.properties

```
db.driver=oracle.jdbc.driver.OracleDriver
db.username=<username>
db.password=<password>
db.tableprefix=
```

- application-dev.properties

```
db.url=jdbc:hsqldb:file:configurations
db.driver=org.hsqldb.jdbcDriver
db.username=sa
db.password=
db.tableprefix=
```

YAML and Profiles

- A YAML file might contain more than one profile
- Default profile not named

```
# default profile
server:
 port:9000
---
spring:
 profiles:dev
server:
 port:8000
---
spring:
 profiles:staging
server:
 port:8080
```