

Title: Externalising Spring Boot properties when deploying to Docker

Post Body:

In my Spring Boot app I want to externalise the properties to run in a Docker container. When first deployed, the properties that are currently in `my-server/src/main/resources/application.yml` are loaded and used by the application as expected. All works fine.

However, my problem is that I need these properties to be updatable as needed, so I need access to the `application.yml` file once on the Docker container. But at this point, it's not included in the `build/docker/` directory before running the `buildDocker` task, so won't be copied over or accessible after first deployment.

So, what I have tried is to copy the Yaml file into the `docker/` build directory, copy it to an accessible directory (`/opt/meanwhileinhell/myapp/conf`), and use the `spring.config.location` property to pass a location of the config to the Jar in my Dockerfile:

```
ENTRYPOINT ['java', \ ... '-jar', '/app.jar', \ '--spring.config.location=classpath:${configDirectory}']
```

Looking at the Command running on the Docker container I can see that this is as expected:

```
/app.jar --spring.config.location=classpath:/opt/meanwhileinhell/myapp/conf]
```

However, when I update a property in this file and restart the Docker container, it isn't picking up the changes. File permissions are:

```
-rw-r--r-- 1 root root 618 Sep 5 13:59 application.yml
```

The [documentation](#) states:

When custom config locations are configured, they are used in addition to the default locations. Custom locations are searched before the default locations.

I can't seem to figure out what I'm doing wrong or misinterpreting, but probably more importantly, is this the correct way to externalise the config for this type of Docker scenario?

Accepted Answer:

DOCKER IMAGE CONFIGURATION

If you look to [the way Spring recommends](#) to launch a Spring Boot powered docker container, that's what you find:

```
FROM openjdk:8-jdk-alpine VOLUME /tmp ARG JAR_FILE COPY ${JAR_FILE} app.jar ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/
```

That means your image extends openjdk and your container has its own environment. If you're doing like that, it would be enough to declare what you want to override as **environment properties** and Spring Boot will fetch them, since [environment variables take precedence](#) over the yml files.

Environment variables can be passed in your docker command too, to launch the container with your desired configuration. If you want to set some limit for the JVM memory, see the link below.

DOCKER COMPOSE SAMPLE

Here you have an example of how I launch a simple app environment with docker compose. As you see, I declare the `spring.datasource.url` property here as an environment variable, so it overrides whatever you've got in your `application.yml` file.

```
version: '2' services:      myapp:      image: mycompany/myapp:1.0.0      container_name: myapp      depends_on:
```

See also:

- [How do I pass environment variables to Docker containers?](#)
- [Limit JVM memory consumption in a Docker container](#)

Highest Rated Answer:

I personally would consider two options:

Using an environment variable per config

```
app:      image: my-app:latest      ports:      - '8080:8080'      environment:      SPRING_DATASOURCE_URL=jdbc:mysql://db:3306/table
```

Using `SPRING_APPLICATION_JSON`

```
app:      image: my-app:latest      ports:      - '8080:8080'      environment:      SPRING_APPLICATION_JSON: '{      'spring.datasou
```