Title: How to understand that spring boot application is ready to work?

Post Body:

I have several microservices on spring. One of them is config server. I try to start services altogether with docker-compose. But there is problem. Microservices try to get configs from config server before it had been initialized. I want to write script for microservice's startup preventing false start. I have to now how i can determinate moment when my config server is ready. Port listening isn't working. Docker hides info about his networking. I believe there is better way than standard output parsing.

Does somebody know how i can determine time of spring boot application starting?

Accepted Answer:

> Does somebody know how i can determine time of spring boot application starting?

Your application must be ready when it returns code `200` from `/health` endpoint with a payload like that

```
{   'status': 'UP' }
```

It means that your application have no issues and is ready to work, anyway this endpoint is normally used by a monitor application to know about your application health that way it can maintain the cluster, this endpoint is not commonly used by others apps, this monitor commonly

- Uses that endpoint to know when your application is ready to receive world requests, that way it can register your application to a public address for example
- Try to restart that application expecting it can solve it's problem when `/health` returns **DOWN** status

Take a look at docker healthcheck, it uses the same concept as spring

To have that endpoint available you will need to add spring actuator dependency, here is a gradle sample

```
compile group: 'org.springframework.boot', name: 'spring-boot-starter-actuator', version: '1.5.10.RELEASE'
```

> Microservices try to get configs from config server before it had been initialized ....

Here some important points to let clear

- Docker compose don't grants startup order unless you are using depends_on clause, anyway docker will never wait to first container completely start up (event using healthcheck) to then start the second container
- If your microservice **A** depends microservice **B** then **A** must be prepared to deal with **B** failure and unavailability, it's a microservice premise, because it will just happens, someday or even worse, in a unexpected moment when it is not supposed to. What about **config-server** restarts in some moment? What will happen with the dependent apps?

So my advise is to you let your application just fail when it tries to get information from **config server** app, if it fails you can do some things:

- Do some kind of retry using spring-retry for example
- If you can deal with some temporary **config server** unavailability then my suggestion is to add a custom check in your `/health` exposing this information to some monitor microservice
- If your app can't work without **config server** then just call `System.exit(-1)` and let docker-compose restart your application again and again until **config server** answer something helpfull

Here some compact example simulating your case and solving that using docker

- config server
- app-1

config-server will ever take longer than **app-1** to get ready, then **app-1** will stay **unhealthy** until **config-server** responds correctly

```
version: '3.4' services:   mg-config-server:     image: nginx:1.10     healthcheck:       test: ['CMD', 'bash', '-c', 'sleep 1
```

Then just start it

```
docker-compose up docker ps | grep 'mg'
```

Anyway it makes more sense to use docker swarm in that case, because it will check healthcheck endpoints and restart the containers if it's not healthy

```
docker swarm init --advertise-addr <your-machine-ip> docker stack deploy --compose-file docker-compose.yml my-stack && docker
```

Docker version: `18.02.0-ce`

Sorry for the too long answer, hope it helps

Highest Rated Answer:

If your issue is specific to Config server, You can also implement spring retry mechanism

```
spring:    application:       name: test-service    cloud:       config:          enabled: true          uri: ${CONFIG_SERVER_
```

You need to have spring-retry dependency also

```
// https://mvnrepository.com/artifact/org.springframework.retry/spring-retry compile group: 'org.springframework.retry', name:
```