DOCKER - 2 Layer

Spring version - 2.3.4.RELEASE

Changes to make in POM.XML file

Make sure your project is using the specified SPRING BOOT version.

Note: this plugin is located just after where the dependency ends.

```
<build>
	<plugins>
		<plugin>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-maven-plugin</artifactId>
			<configuration>
				<layers>
					<enabled>true</enabled>
				</layers>
			</configuration>
			<executions>
				<execution>
					<goals>
						<goal>build-image</goal>
					</goals>
				</execution>
			</executions>
		</plugin>
	</plugins
	<finalName>app</finalName>
 </build>
```

Here, build-image with layer enabled, will enable the feature of breaking the fat jar (single layer jar) into multiple layers.

After that, Update your pom.xml. (right click > maven > update project)

Build you project: clean package -Pproduction

This will create a jar file in the folder name Target in main directory.

Dockerfile

Create a docker file with name Dockerfile with no extension in your current directory.

Add following code:

```
FROM openjdk:8-alpine as builder
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
RUN java -Djarmode=layertools -jar app.jar extract

FROM openjdk:8-alpine
COPY --from=builder dependencies/ ./
COPY --from=builder snapshot-dependencies/ ./
COPY --from=builder spring-boot-loader/ ./
COPY --from=builder application/ ./
EXPOSE 8002
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

We are using Alpine version of java.

Before you build docker image, make sure to add alpine image in your DOCKER DESKTOP

Open your terminal and run: `docker pull openjdk:8-alpine`

Once alpine is in you docker local,

Create docker image using: `docker build -t app .` #here app is a custom name.

After image build is done,

Push Image to DOCKER HUB

#run in terminal

step 1: docker login # to make sure you are logged in.

step 2: docker tag app username/my-app # {username} is you docker hub username and {my-app} is the custom image name you want to create in hub

step 3: docker push username/my-app #this will push image to docker hub

Run Images in Docker

To run images in docker and access it in browser, run following command

```
docker run -p 8080:8001 caelumpirata/docker-image
```

Here, the first port is that you will use in browser after the domain name, and the second port is that is already been exposed while creating the docker image.

Running Images in KUBERNETES Locally

Make sure, HYPERV is enabled in your system.

Start MINIKUBE in terminal running with Administrative privilege using:

```
minikube start --driver=hyperv
```

Once it starts,

Open Minikube Dashboard: minikube dashboard

If you are using INGRESS, enable it using: minikube addons enable ingress

After that, Start Minikube Tunnel: minikube tunnel

Next step, Deployment of our application in local kubernetes :)

Docker Deployment

Create a file in current directory with name deployment.yaml and paste the following code:

```
apiVersion: v1 # Kubernetes API version
kind: Service # Kubernetes resource kind we are creating
metadata: # Metadata of the resource kind we are creating
  name: spring-alarm-app
spec:
  selector:
    app: spring-alarm-app
  ports:
    - protocol: "TCP"
      port: 8003 # The port that the service is running on in the cluster
      targetPort: 8080 # The port exposed by the service
  type: LoadBalancer # type of the service. LoadBalancer indicates that our service will be external.
---
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: spring-alarm-app
spec:
  selector:
    matchLabels:
      app: spring-alarm-app
  replicas: 2 # Number of replicas that will be created for this deployment
  template:
    metadata:
      labels:
        app: spring-alarm-app
```

```
  spec:
    containers:
      - name: spring-alarm-app
        image: username/my-app:latest # Image that will be used to containers in the cluster and don't forget to add ( :latest ) at the end
        imagePullPolicy: Always
        ports:
          - containerPort: 8003
```

Deploy command: kubectl apply -f deployment.

Make sure it is deployed successfully using following command.

All Deployments: kubectl get deployments

Running services: kubectl get services

Get running pods using: kubectl get pods

Ingress

Create a new file with name ingress.yaml in your current directory and paste.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: auth-ingress #ingress name
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: example.com #hostname
      http:
        paths:
          - path: /(.*)
            pathType: Prefix
            backend:
              service:
                name: service1 #deployed service name
                port:
                  number: 8001 #port exposed by that service
          # you can add multiple paths with different services
          - path: /hello/(.*)
            pathType: Prefix
            backend:
              service:
                name: hello-service #deployed service name
                port:
                  number: 8004 #port exposed by that service
```

DEPLOY COMMAND: kubectl apply -f ingress.yaml

Make sure it is deployed successfully, using

command: kubectl get ingress