18. Deploying Spring

# 18.1 Weighting development options

-We can build and run SB apps in several ways:

+Run directly in IDE

+From command line using Maven(spring-boot:run) or Gradle (bootrun)

+Use Maven or Gradle to produce an executable JAR file that can be run at command line or deployed in cloud.

+Use Maven or Gradle to produce WAR that can be deployed to a traditional Java app server

+Using Maven or Gradle to produce a container image that can be deployed anywhere that containers are support, including Kubernetes environments

-What about when you’re ready to deploy the app into a production or other nondevelopment environment. Jar or WAR files are valid options for developing apps to production environment.

+Deploy to a **Platform as a Service (Paas) cloud**: Cloud Foundry (<https://www.cloudfoundry.org/>), use **JAR**

+Deploy to **Java app servers**: Tomcat, WebSphere, WebLogic, use **WAR**

+Deploy to **Kubernetes**: Modern cloud platforms are based on Kubernetes (<https://kubernetes.io/>). Build your app into a **container image**.

# 18.2 Building executable JAR files

-Produce executable JAR file:

+Maven: 

After build, Jar file will placed into /target/<artifactId>-<version>-SNAPSHOT.jar

+Gradle: 

Jar file will placed into /libs: rootProject.name property in settings.gradle + version property in build.gradle

-Run JAR file: 

+The app will run and start up an embedded server(Netty/Tomcat) and listen for requests on server.port (8080 by default) -> **Run app locally**

-Deploy to Cloud Foundry foundation:



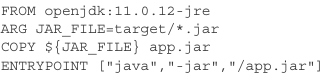
+1st arg is the name given to app in Cloud Foundry. This name is used to reference the app in Cloud Foundry and cf CLI, and as a subdomain at which the app is hosted.

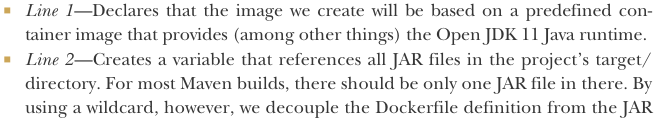
# 18.3 Building container images

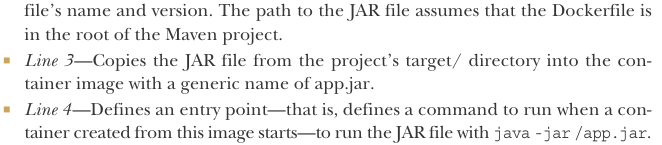
-**Docker** (<https://www.docker.com/>): the de facto standards for distributing apps for all kinds for deployment in the cloud. Many cloud environments (AWS, Azure, Google Cloud) accept Docker containers for deploying apps.

-**Containerized** **apps**, created with Docker, draws analogies from real-world **intermodal containers**. They share a common container format that can be deployed and run anywhere, regardless of the app inside.

-The most basic way to create an image from SB app is to use **docker build command** and a **Dockerfile** that copies **JAR file** into container image. This is Dockerfile:







-With this Dockerfile, create image using Docker cmd:



+Thi . references the relative path to location of Dockerfile. If you running docker build from a different path, replace . with the path to Dockerfile



+The value after –t is the image tag: made up of a name and version.

-docker run: run created image:



+-p8080:8080 forwards requests to port 8080 on host machine(your machine where running Docker) to the container’s port 8080 (where Tomcat or Netty listening for requests).

-While building a Docker image this way is easy if you already have JAR file, it’s not the easiest way to create image from SB app.

-SB build plugins for Maven and Gradle support the building of container images directly.

+Build Maven-built Spring project into container image:



+Gradle-bult project: 

+This builds an image with a default tag based on <artifactId> and <version>.

+SB’s build plugins rely on Docker to create images. You need to have the Docker runtime installed on machine building this image:



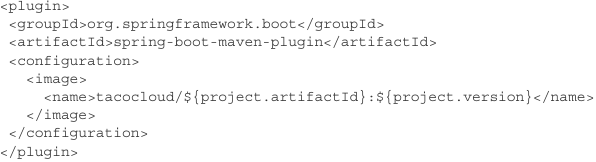
+This runs the image and exposes image’s port 8080 (the embedded Tomcat or Netty server lisening) to host machine’s port 8080.

+The default format of tag is docker.io/library/<artifactId>:<version>. Change it:





Or specify the image name as part of the build:





+With this configuration, you can build image without specifying the image name, you can run image with docker run or use docker push to push image to an image registry such as DockerHub:



-Once the image is in an image registry, it can be pulled and run from any environment that has access to that registry.

18.3.1 Deploying to Kubernetes

-Kubernetes is powerful platform on which to deploy apps. Check out: **Kubernetes in Action**

-You need a Kubernetes environment to deploy app: Amazon’s AWS EKS, Google Kubernetes Engine. For experimentation locally: MiniKube (https://minikube.sigs.k8s.io/docs/), MicroK8s (<https://microk8s.io/>) , Kind (<https://kind.sigs.k8s.io/>)

-The 1st thing is create a deployment manifest: a YAML file that describes how an image should be deployed:



+This manifest can be named anything: deploy.yaml and placed it in k8s at the root of project

+Our deployment is named taco-cloud-deploy and is set to deploy and start a container based on the image. By giving “latest” , we can know that the very latest image pushed to container registry will be used.

+replicas:3 -> 3 instances of container running.

-Apply the deployment: 

-See the deployment in action: 

+You can see 3 pods, each one running a container instance.

-Expose a port from one of the pods on your machine:



18.3.2 Enabling graceful shutdown

18.3.3 Working with app liveness and readiness

# 18.4 Building and deploying WAR files

# 18.5 The end is where we begin