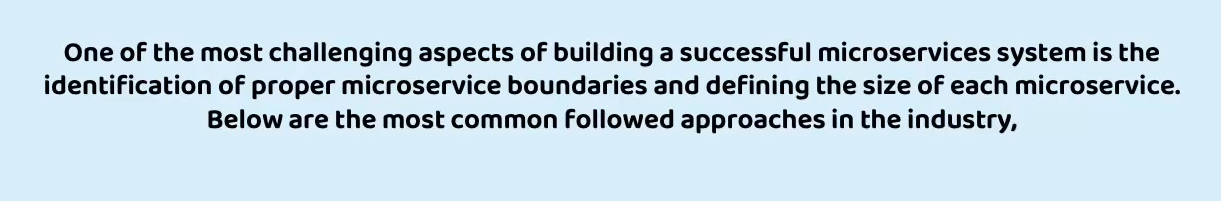
Challenges to develop the microservices:

1. Build the micro services
2. Sizing the micro services
3. Deployment, Portability and Scalability

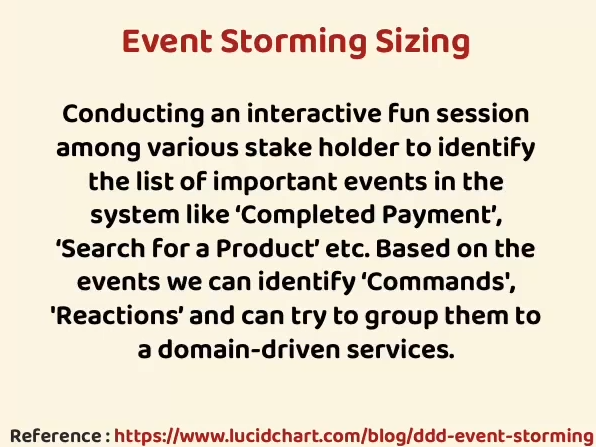
2. **sizing the microservices**



i.e. microservices should not be so big that is cannot grab the whole advantages of microservices system, also it should not be so small or it should not have very less business logic due to which, we will end up having a greater number of microservices within organization. If onboard a greater number of small microservices then there will be operational overhead to make sure all these microservices are interconnected. That’s why its required to ensure that all microservices should have a proper size and their boundaries should be clearly define.

Solutions:

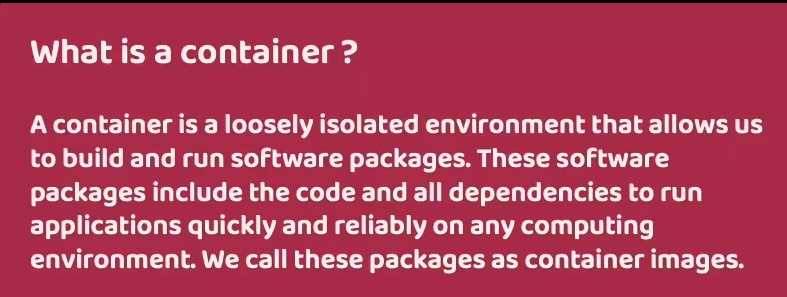
1. Doman driven sizing -> an application consists multiple domains or subdomains. To get a clear picture about the operations and services of a specific domain, its required to discuss with stakeholders, product owners, technical people, business analyst, QAs etc, then we can have some idea about the sizing of microservices. But this process is very time consuming.
2. Event storming sizing->



1. **Deployment, Portability and Scalability**->

**Portability means** how we can move 100s of microservices across environments with less effort, configuration and cost.

Container ->



1. Docker commands to **create and running the container**:
2. Docker images
3. Docker run -p 8181:8080 <image\_name >:tag , or

Docker run -p 8181:8080 <image\_id>

The first port **8181** : the Docker host ( you can use this port to access to your container) to access to the container from the **outside**.

the second one **8080** : is the port used by your application (or where the running instance of application i.e. container is going to start)

**So after starting the container, we should use 8181 to access the application.**

**IMP-> We already start and exposed container on the Port 8181 of our system, therefor we can not reuse the same port for the start and running another container so we have to use another port of our system. And why don’t need to change the second port which is 8080 because this second port number indicates the port number where your container is going to start inside the docker network.**

1. Docker run -p 8181:8080 <image\_name>:tag ,

here **p** indicates for port mapping, actually by default all docker containers starts running inside own isolated network, and we can not access the services inside their network therefor in order to get the access from external network like from local system or from any other system, we need to expose them explicitly with the help of this port mapping.

1. When container runs using below commands , no other docker commands can be run

Docker run -p 8181:8080 <image\_name >:tag , or

Docker run -p 8181:8080 <image\_id>

Therefor, to start containers in **detached mode** below command can be used

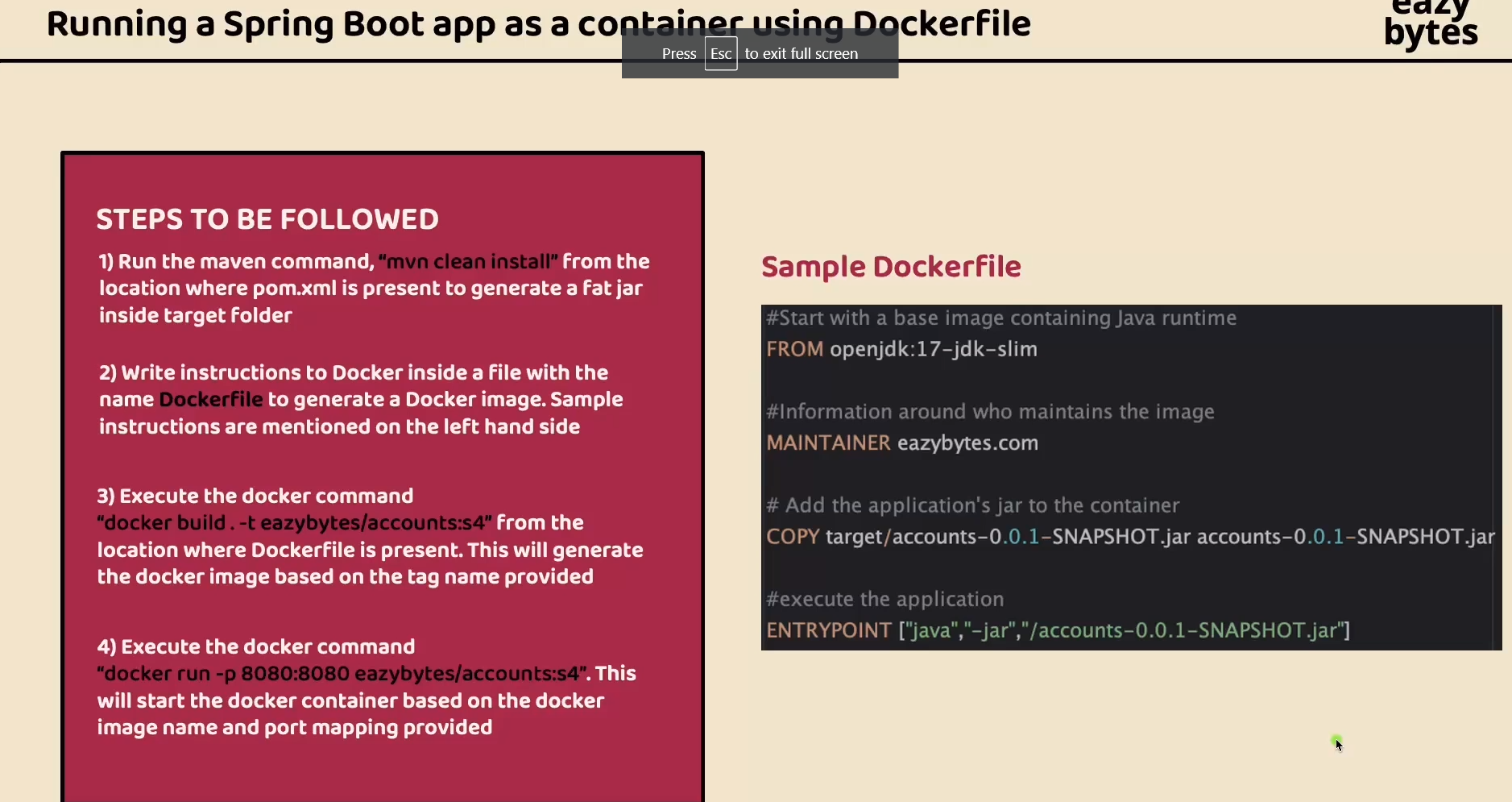
Docker run -d -p 8181:8080 <image\_name>:tag , or

Docker run -d -p 8181:8080 <image\_id>

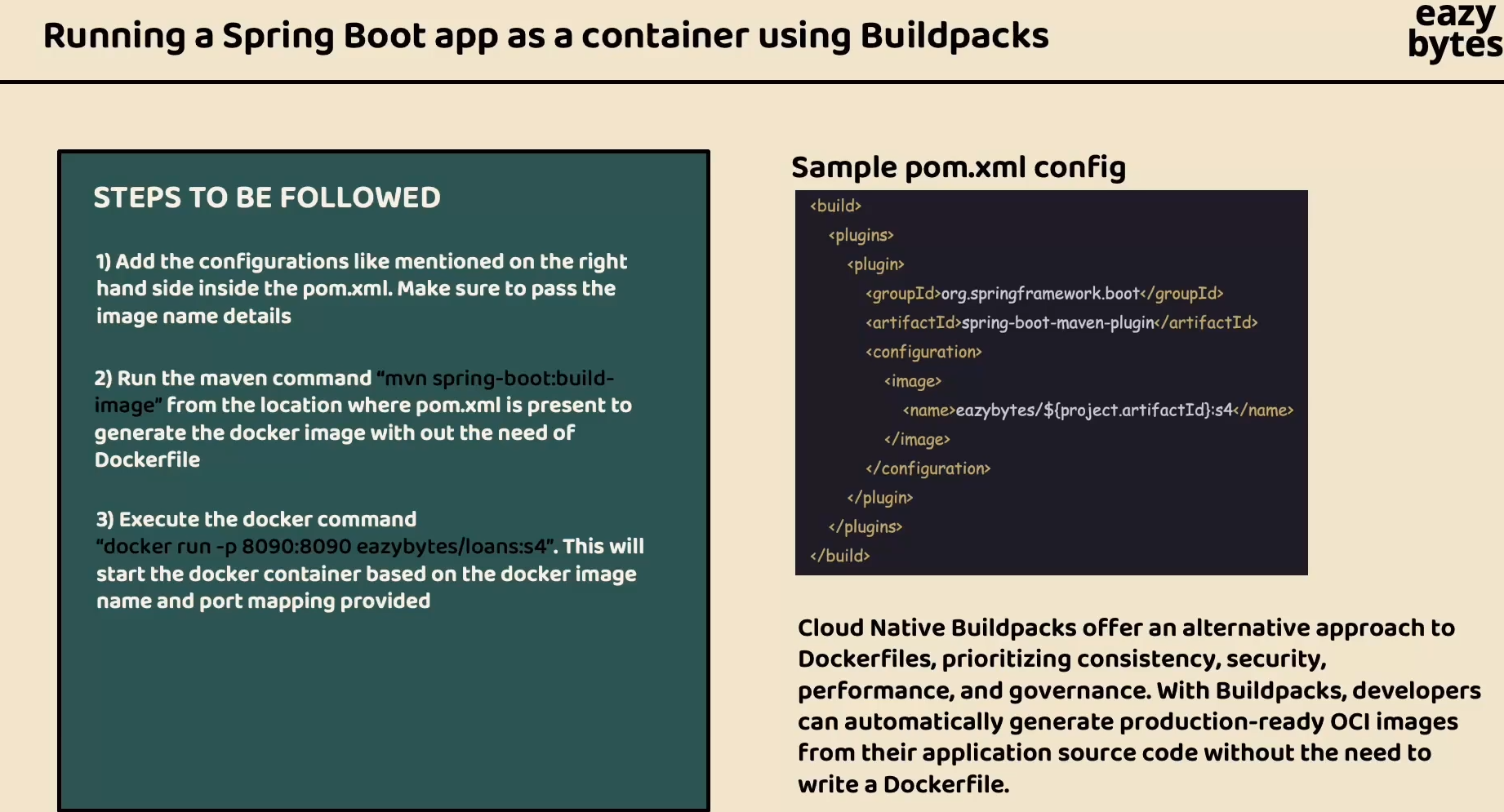
**-d tells docker server that run the containers without display any logs on terminal.**

7. docker ps -> this command is using to **show all running containers**

8. running spring boot app as container using docker file



9. running spring boot app as container using buildpacks



10. mvn spring-boot:build-image

Command to create docker image using buildpacks.