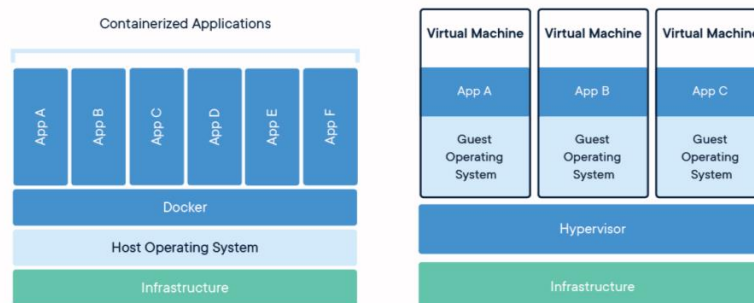


## Background

We will be leveraging Docker to deploy the software that we will be using for the remainder of the course. In order to familiarize ourselves with this environment, we need to contrast containers with the more standard virtual machines (VMs) that we see in the market today (e.g. VirtualBox, VMWare, etc). VMs allow us to run software applications within a guest operating system such as Windows or Linux. The hardware is virtualized and VMs provide process isolation for applications. This means we can run amok within a container without fear of causing problems to the parent system. However, we pay a price in terms of computational overhead for virtualizing the hardware to run a guest operating system.

Containers, such as the Docker system, approaches the problem by providing most of the isolation of VMs through virtualization of the operating system. This reduces the computational overhead. Containers offer a logical packaging mechanism and allows applications to be deployed easily and consistently regardless of target environment. This means you can deploy the identical container in multiple target environments e.g. public cloud or personal desktop. These containers provide isolated environments comprised of everything needed to run an application: code, runtime, system tools, system libraries and settings so the application runs quickly and reliably from one computing environment to another.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.



## Containers

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

## Virtual Machines

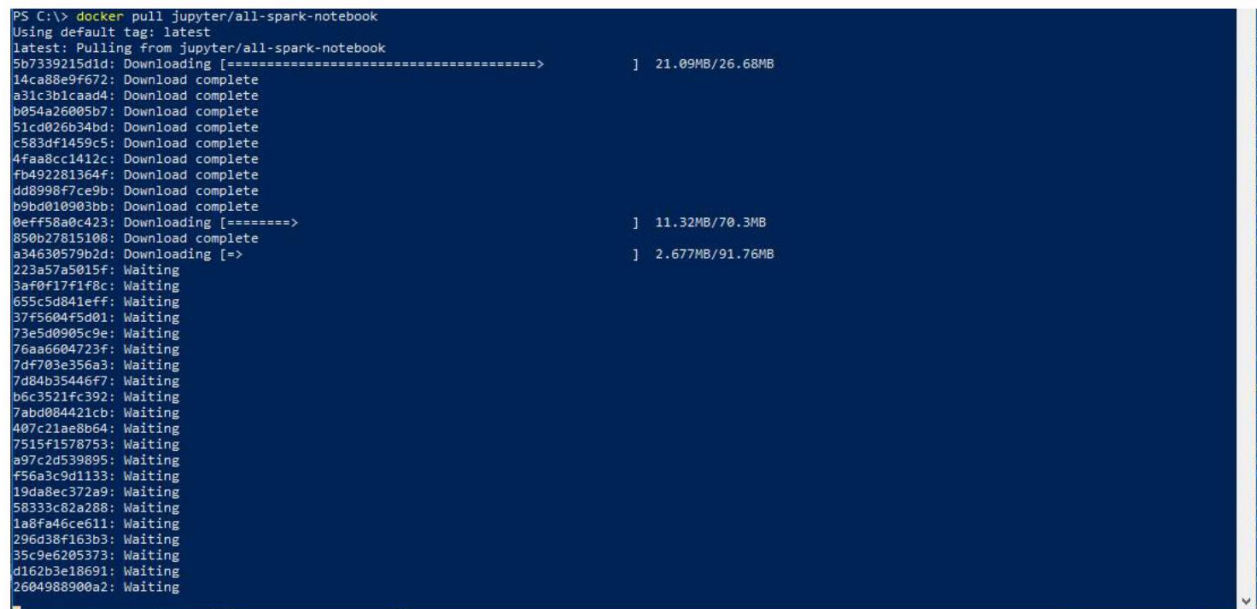
Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot

For those who has Docker installed already, you may go straight to [Step 6](#)

## Setup

1. Click “Desktop and Take a Tutorial” on <https://www.docker.com/get-started>
2. Sign up for a docker id account. Docker ID grants you access to Docker services such as the Docker Store, Docker Cloud, Docker Hub repositories, and some beta programs. Your Docker ID becomes repository namespace used by hosted services such as Docker Hub and Docker Cloud. All you need is an email address. This account also allows you to log in to services such as the Docker Support Center, the Docker Forums, and the Docker Success portal. If you already have an account, simply login.
3. Download Docker Desktop and run the installer
4. Run Docker Desktop and login with the Docker ID from above.
5. In your Settings under Resources > File Sharing select the local drive to be available to your container. This will allow you to access and save files outside of the container.
6. Open a Powershell (on Windows) or a Terminal (on Mac) and run the following to retrieve a container to run Spark and R for this class:

```
docker pull jupyter/all-spark-notebook
```

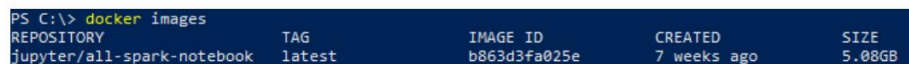


```
PS C:\> docker pull jupyter/all-spark-notebook
Using default tag: latest
latest: Pulling from jupyter/all-spark-notebook
5b7339215d1d: Downloading [=====] 21.09MB/26.68MB
14ca88e9f672: Download complete
a31c3b1caad4: Download complete
b054a2e005b7: Download complete
51cd026b34bd: Download complete
c583df1459c5: Download complete
4faa8c1412c: Download complete
f4a9281364f: Download complete
dd8998f7ce9b: Download complete
b9bd010903bb: Download complete
9eff58a0c423: Downloading [=====] 11.32MB/70.3MB
850b27815108: Download complete
a34630579b2d: Downloading [=====] 2.677MB/91.76MB
223a57a5015f: Waiting
3af0f17f1f8c: Waiting
655c5d841eff: Waiting
37f5604f5d01: Waiting
73e5d0905c9e: Waiting
76aa6604723f: Waiting
7df703e356a3: Waiting
7d84b35446f7: Waiting
b6c3521fc392: Waiting
7abd084421cb: Waiting
407c21ae8b64: Waiting
7515f1578753: Waiting
a97c2d539895: Waiting
f56a3c9d1133: Waiting
19da8ec372a9: Waiting
58333c82a288: Waiting
1a8fa46ce611: Waiting
296d38f163b3: Waiting
35c9e6205373: Waiting
d162b3e18691: Waiting
2604988900a2: Waiting
```

Wait for this process to finish, you should see something similar when this completes:

7. When this process completes, we can confirm that the image is available by running:

```
docker images
```



```
PS C:\> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jupyter/all-spark-notebook	latest	b863d3fa025e	7 weeks ago	5.08GB

8. To run this image, we use the “docker run” command. This will start the image mapping port 8888 in the container to an external port 8888 which will allow us to connect to the container. Port 8888 is used for Jupyter notebook. Similarly, port 4040 which is also being exposed here is used for the SparkUI (Spark Monitoring and Instrumentation UI) which will be very helpful for debugging as we get to later lectures.

```
docker run -p 8888:8888 -p 4040:4040 jupyter/all-spark-notebook
```

Optionally, if you would like to mount a drive so that it is available to your Docker container you can use the -v argument (in this case I map my local c:/ directory to the image's /mnt/c folder)

```
docker run -p 8888:8888 -p 4040:4040 -v c:/:/home/jovyan jupyter/all-spark-notebook
```

Since I'm running this on Windows, this is saying mount the volume c: / on the local file system to the /home/jovyan folder, which is the default home folder for Jupyter Notebook in the Docker container. Now I can reference files on my local computer in Jupyter notebooks, etc.

```
I 2022-03-17 18:15:24.033 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.9/site-packages/jupyterlab
I 2022-03-17 18:15:24.033 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
I 2022-03-17 18:15:24.042 ServerApp] jupyterlab | extension was successfully loaded.
I 2022-03-17 18:15:24.062 ServerApp] nbclassic | extension was successfully loaded.
I 2022-03-17 18:15:24.064 ServerApp] Serving notebooks from local directory: /home/jovyan
I 2022-03-17 18:15:24.064 ServerApp] Jupyter Server 1.15.5 is running at:
I 2022-03-17 18:15:24.064 ServerApp] http://499eb4a0c466:8888/lab?token=065b9ab6b59b1d6e364744c8614ce58a5075dec370642d37
I 2022-03-17 18:15:24.065 ServerApp] or http://127.0.0.1:8888/lab?token=065b9ab6b59b1d6e364744c8614ce58a5075dec370642d37
I 2022-03-17 18:15:24.065 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
C 2022-03-17 18:15:24.076 ServerApp]
```

To access the server, open this file in a browser:

file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html

Or copy and paste one of these URLs:

http://499eb4a0c466:8888/lab?token=065b9ab6b59b1d6e364744c8614ce58a5075dec370642d37

or http://127.0.0.1:8888/lab?token=065b9ab6b59b1d6e364744c8614ce58a5075dec370642d37

9. Take note of the URL below, in this

case <http://127.0.0.1:8888/?token=e9e624471aecc522dbe355ef27a02329f973c385a7bd419e>

. The token will be different based on your run. Copy the URL that shows up on your screen and paste it into a browser.

```
[1]: from pyspark.sql import SparkSession
MAX_MEMORY = "8g"
spark = SparkSession \
    .builder \
    .appName("SparkPiTest") \
    .config("spark.executor.memory", MAX_MEMORY) \
    .config("spark.driver.memory", MAX_MEMORY) \
    .master("local[*]") \
    .getOrCreate()
```

```
[2]: spark
```

```
[2]: SparkSession - in-memory
```

**SparkContext**

Spark UI

Version	v3.2.1
Master	local[*]
AppName	SparkPiTest

```
[*]: from random import random
```

## Useful Docker commands

Command	Description
<b>Running Docker Containers</b>	
docker start [container]	Start a particular container. If your container was stopped from earlier, you can use this command to restart it.
docker stop [container]	Stop a particular container.
docker exec -ti [container] [command]	Run a shell command inside a particular container
docker run -ti --image [image] [container] [command]	Create and start a container at the same time, and then run a command inside it.
docker run -ti --rm --image [image] [container] [command]	Create and start a container at the same time, run a command inside it, and then remove the container after executing the command.
docker pause [container]	Pause all processes running within a particular container.
docker attach [container]	Attaches the local standard input, output and error streams to a running container. If you are restarting a stopped container, you can use this command to get the standard output.
<b>Using Docker Utilities</b>	
docker history [image]	Display the history of a particular image.
docker images	List all of the images that are currently stored on the system.
docker inspect [object]	Display low-level information about a particular Docker object.
docker ps	List all of the containers that are currently running.
docker version	Display the version of Docker that is currently installed on the system.
<b>Cleaning Up Your Docker Environment</b>	
docker kill [container]	Kill a particular container.
docker kill \$(docker ps -q)	Kill all containers that are currently running.
docker rm [container]	Delete a particular container that is not currently running.
docker rm \$(docker ps -a -q)	Delete all containers that are not currently running.