# **Docker: The Container Engine**

Advances in Data Sciences and Architecture

Under the guidance of Prof- Srikanth Krishnamurthy

Docker is an open source project to pack, ship and run any application as a lightweight container.

Docker began as an open-source implementation of the deployment engine which powered [dotCloud](http://web.archive.org/web/20130530031104/https:/www.dotcloud.com/), a popular Platform-as-a-Service. It benefits directly from the experience accumulated over several years of large-scale operation and support of hundreds of thousands of applications and databases.

## **Better than VMs**

A common method for distributing applications and sandboxing their execution is to use virtual machines, or VMs.

Docker relies on a different sandboxing method known as containerization. Unlike traditional virtualization, containerization takes place at the kernel level. Most modern operating system kernels now support the primitives necessary for containerization, including Linux with [openvz](https://openvz.org/), [vserver](http://linux-vserver.org/) and more recently [lxc](https://linuxcontainers.org/), Solaris with [zones](https://docs.oracle.com/cd/E26502_01/html/E29024/preface-1.html#scrolltoc), and FreeBSD with [Jails](https://www.freebsd.org/doc/handbook/jails.html).

Docker builds on top of these low-level primitives to offer developers a portable format and runtime environment that solves all four problems. Docker containers are small (and their transfer can be optimized with layers), they have basically zero memory and CPU overhead, they are completely portable, and are designed from the ground up with an application-centric design.

Perhaps best of all, because Docker operates at the OS level, it can still be run inside a VM!

## **Plays well with others**

Docker does not require you to buy into a particular programming language, framework, packaging system, or configuration language.

## **Escape dependency problems**

A common problem for developers is the difficulty of managing all their application's dependencies in a simple and automated way.

Docker solves the problem of dependencies by giving the developer a simple way to express all their application's dependencies in one place, while streamlining the process of assembling them. Docker doesn't replace your favorite packaging systems. It simply orchestrates their use in a simple and repeatable way. How does it do that? With layers.

Docker defines a build as running a sequence of Unix commands, one after the other, in the same container. Build commands modify the contents of the container (usually by installing new files on the filesystem), the next command modifies it some more, etc. Since each build command inherits the result of the previous commands, the order in which the commands are executed expresses dependencies.

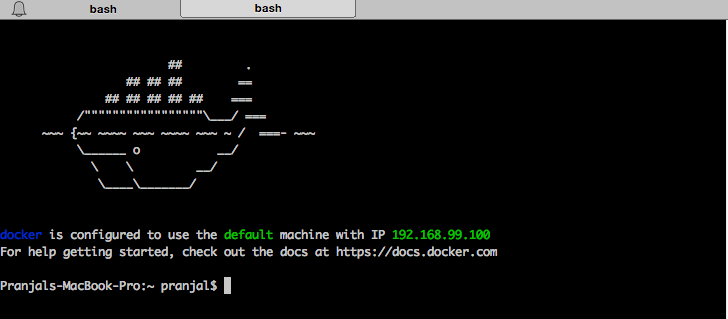
## **GETTING STARTED**

**Installation**

1. Go to <https://www.docker.com/products/overview> and download docker based on your operating system.
2. Install docker like a normal software program.

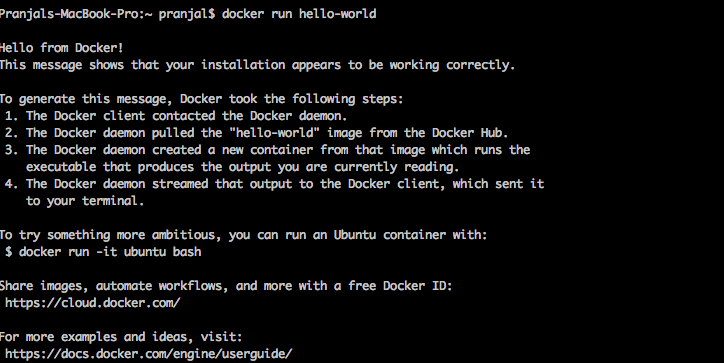
**How to start docker?**

1. Type docker in quickstart terminal . It starts the docker in terminal. First time it will install the docker image, create a virtualbox to manage various containers.



1. To check the installation was successful. Let’s run the hello-world container from docker.

|  |
| --- |
| docker **run** hello-world |



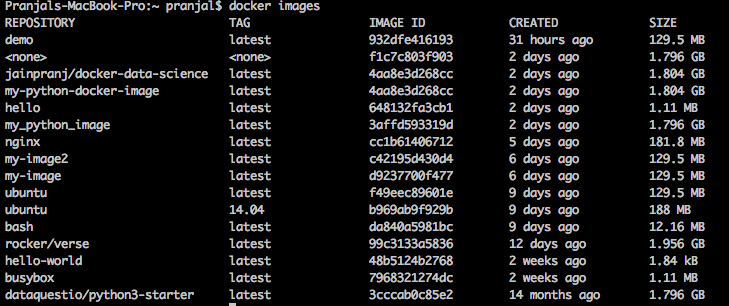
3. Run a docker container with ubuntu image.

|  |
| --- |
| docker **run** -ti unbuntu bash |



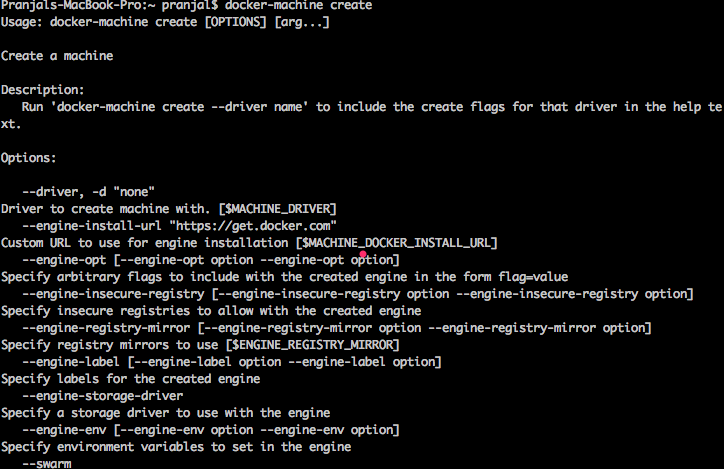
4. List all the docker images

|  |
| --- |
| docker images |



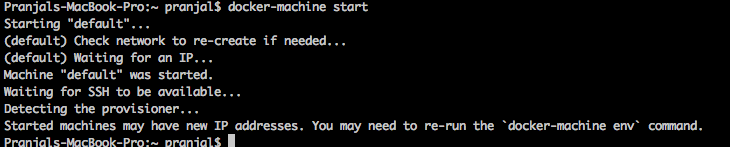
5. Create a new docker machine

|  |
| --- |
| docker-machine create |



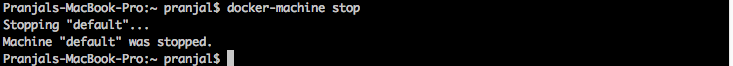
6. Start docker-machine

|  |
| --- |
| docker-machine start |



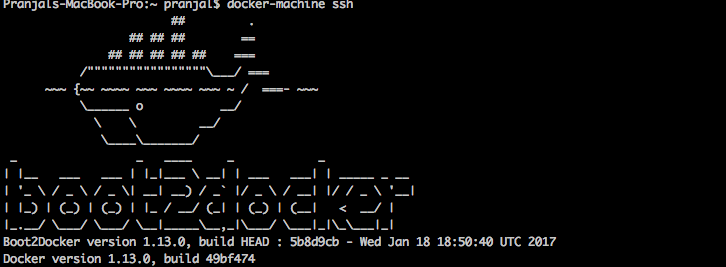
7. Stop docker-machine

|  |
| --- |
| docker-machine stop |



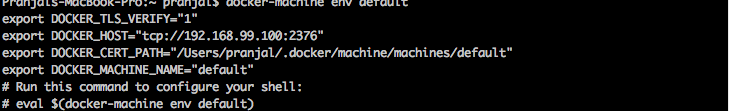
8. Connect to docker machine

|  |
| --- |
| docker-machine ssh |



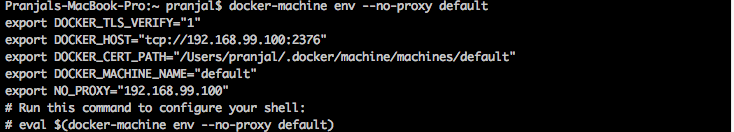
9. Setting docker machine as default.

|  |
| --- |
| docker-machine **env** default |



10. Enabling internet access on docker machine by setting it to no proxy.

|  |
| --- |
| docker-machine **env** –no-proxy default |



11. Run a nginx server on docker machine

|  |
| --- |
| docker run -d -p 8000:80 nginx |

-p implies the port number. The first port number is internal port and other is external port.

-d-implies running container in detached mode.



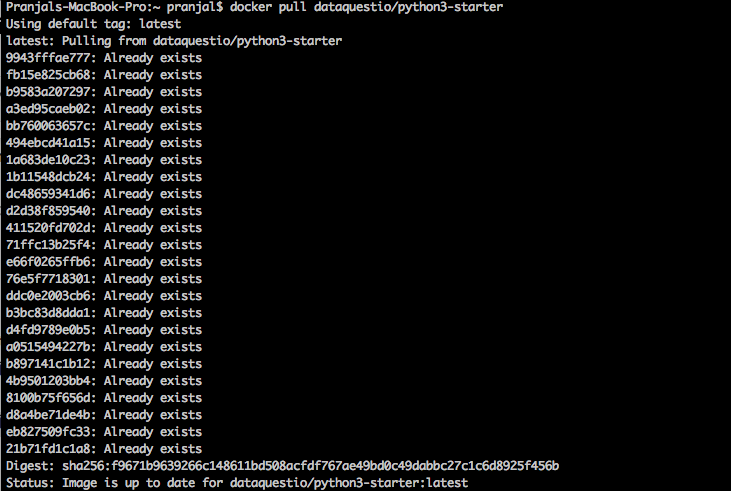
12. Connecting to nginx server

$(docker-machine ip default):8000

**Python Data Science setup with docker**

1. Download python image. The following is an illustration from dataquestio.

docker pull dataquestio/python3-starter



2. Creating a local directory to store python notebooks

|  |
| --- |
| cd docker/ mkdir notebooks\_docker |

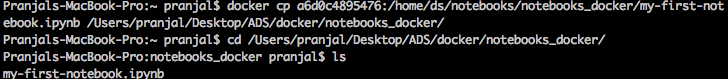
3. Running juypter from the image downloaded

|  |
| --- |
| docker **run** -d -p 8888:8888 -v /Users/pranjal/Desktop/ADS/docker:/home/ds/notebooks dataquestio/python3-starter |



4. Copying notebooks from docker container to local.

|  |
| --- |
| docker cp a6d0c4895476:/home/ds/notebooks/notebooks\_docker/my-first-notebook.ipynb /Users/pranjal/Desktop/ADS/docker/notebooks\_docker/ |



**Pushing a Docker image to Docker Hub**

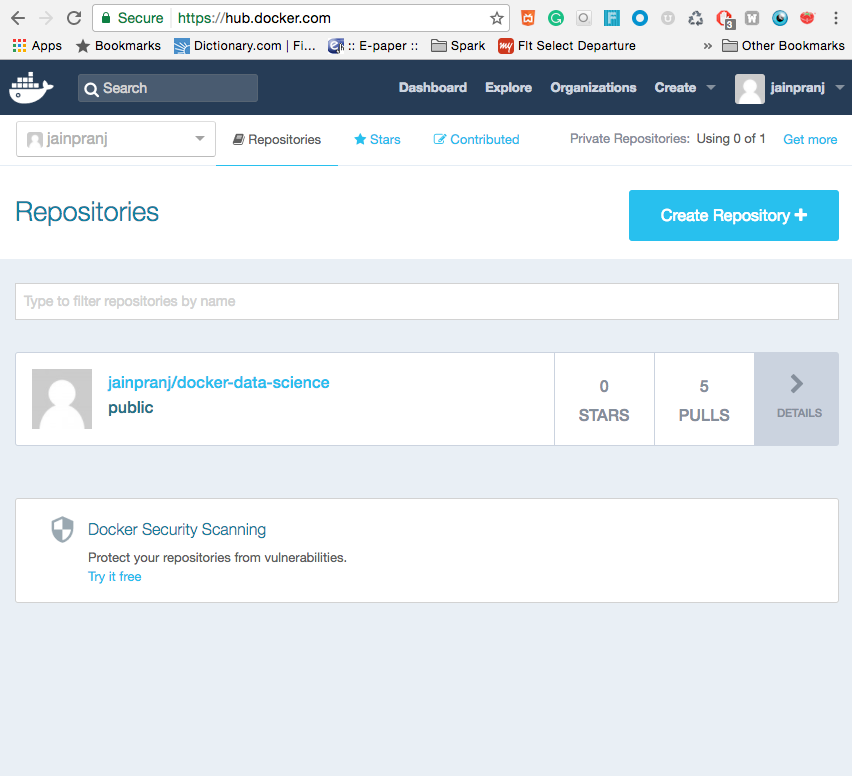
Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

Docker Hub provides the following features:

* Image Repositories: Find and pull images from community and official libraries, and manage, push to, and pull from private image libraries to which you have access.
* Automated Builds: Automatically create new images when you make changes to a source code repository.
* Webhooks: A feature of Automated Builds, Webhooks let you trigger actions after a successful push to a repository.
* Organizations: Create work groups to manage access to image repositories.
* GitHub and Bitbucket Integration: Add the Hub and your Docker Images to your current workflows

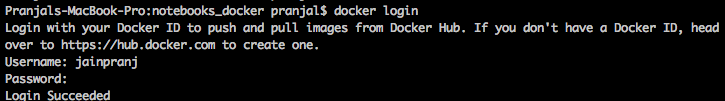
**STEPS TO PUBLISH AN IMAGE ON DOCKERHUB**

1. Create an account on Docker Hub. Create a repository.



2. Connect to Docker Hub from Docker. It prompts for username and password.

|  |
| --- |
| docker login |



3. First commit the image with changes using the following command:

|  |
| --- |
| docker commit a6d0c4895476192605cd0309d5f8181c49f2117185d1841e6f16e8ccb621ef13 |

|  |
| --- |
| docker commit container-name |



4. Tagging the committing image with repository

|  |
| --- |
| docker tag my-python-docker-image jainpranj/docker-data-science:latest |

|  |
| --- |
| docker tag <image-to-be-committed> <repository name> |



5. Pushing the image to Docker Hub

|  |
| --- |
| docker push <repository-name> docker push jainpranj/docker-data-science:latest |



**DOCKERFILE**

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

The following are the steps to create a DockerFile:

1. Create a file and enter following command

|  |
| --- |
| touch Dockerfile **FROM** debian:sid **RUN** echo "building simple docker image" **CMD** echo "Hello Container" |

|  |
| --- |
| **FROM**- implies repository **from** which image needs to be downloaded **RUN**- to run the docker command **CMD**- To run other linux command **EXPOSE**- to **expose** any ports **ENV**- to set environment variables |

2. Build a docker file

|  |
| --- |
| docker build -t hello . |

|  |
| --- |
| docker build -t <container-name> |

. -implies current directory>

3. Run the container

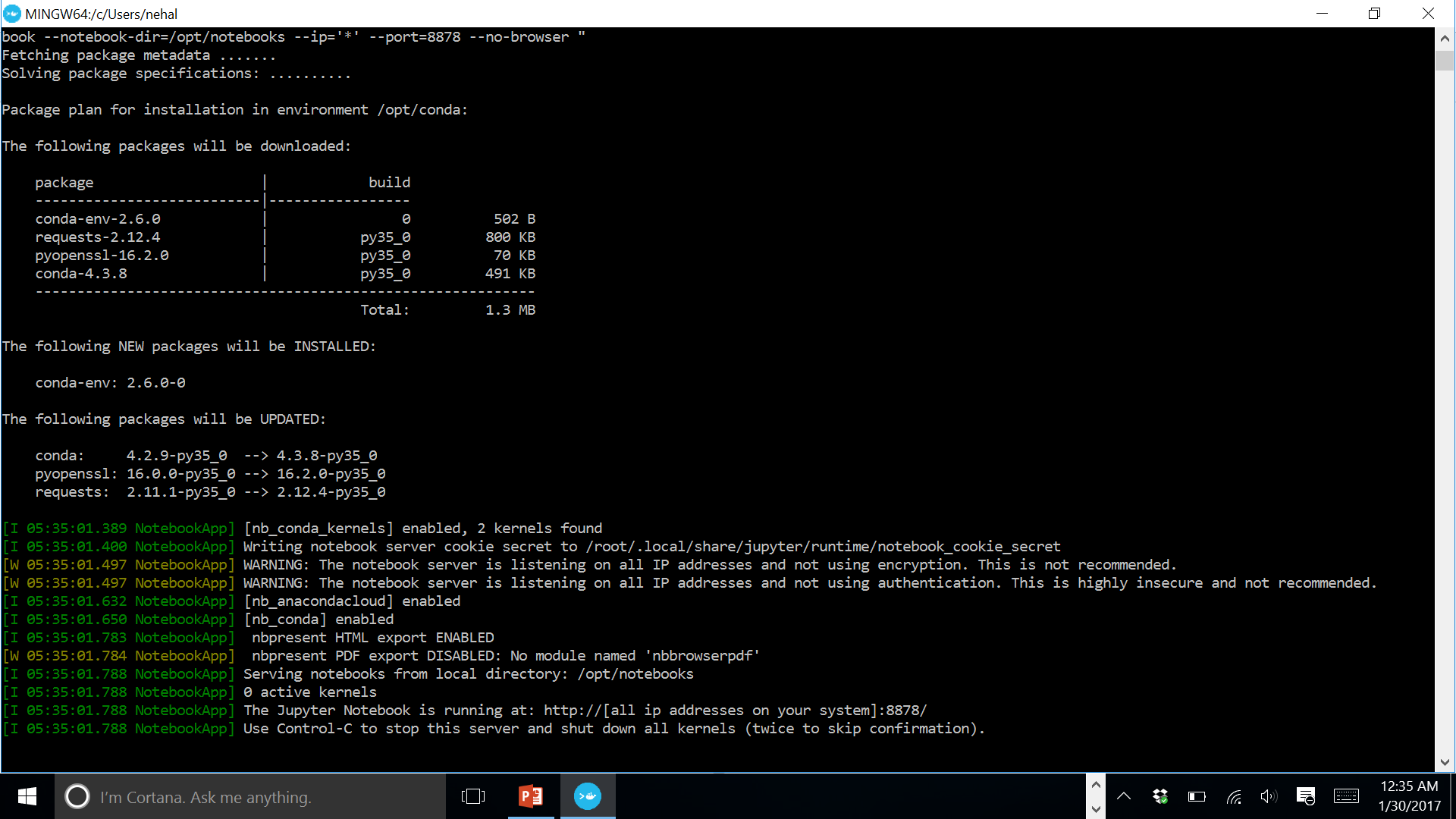
|  |
| --- |
| docker **run** hello |

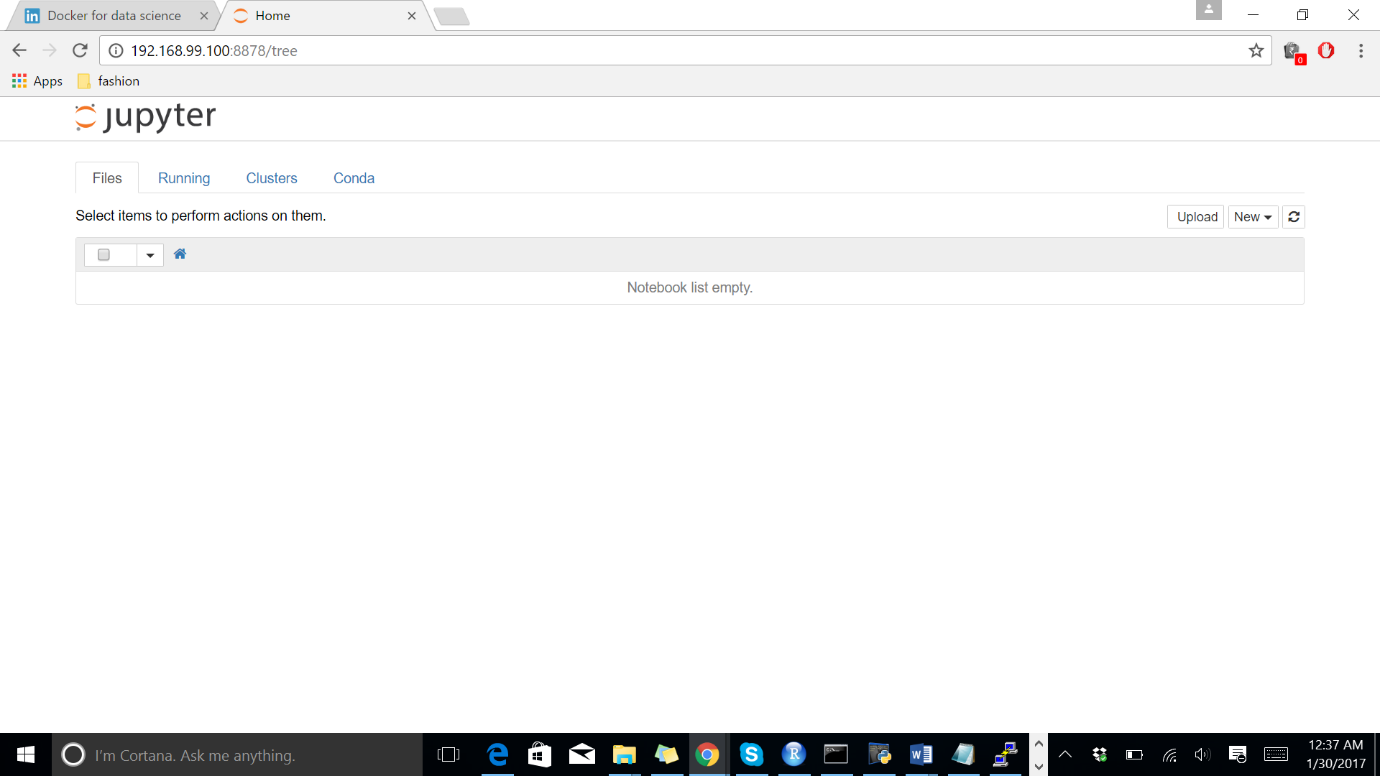
**DOCKER FOR ANACONDA**

Whether you’re a developer, data scientist, or devops engineer, Anaconda and Docker can provide your entire data science team with a scalable, deployable and reproducible Open Data Science platform

Steps for using Anaconda with Docker:

1. Download an Anaconda image from one  of the following resources:
   * [**https://hub.docker.com/r/continuumio/anaconda3/**](https://hub.docker.com/r/continuumio/anaconda3/)
   * [**https://www.dataquest.io/blog/docker-data-science/**](https://www.dataquest.io/blog/docker-data-science/)
2. Pull and Run the image
3. Start a Jupyter Notebook server (optional)  and interact with Anaconda via your browser



**DOCKER WITH AWS**

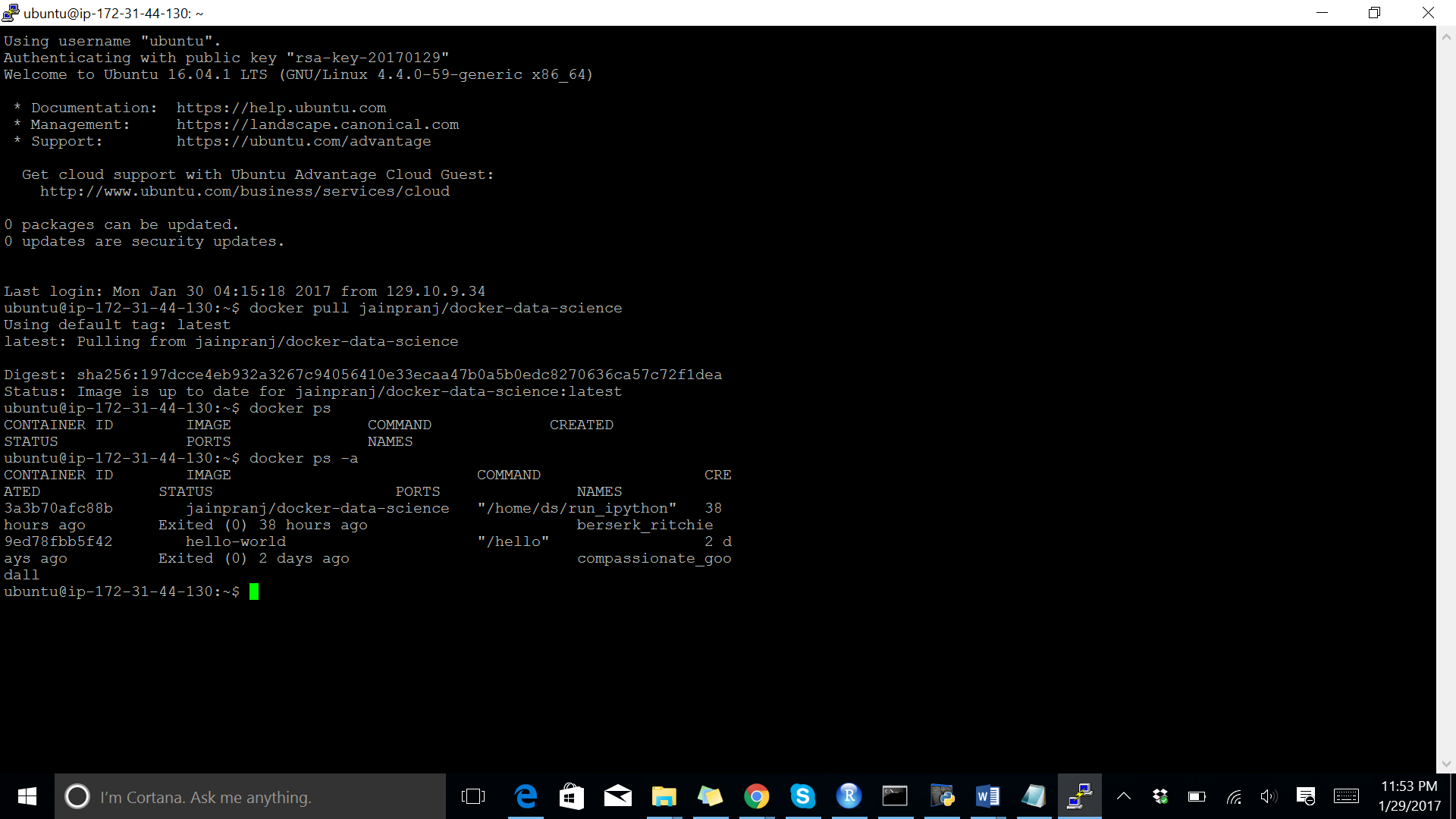
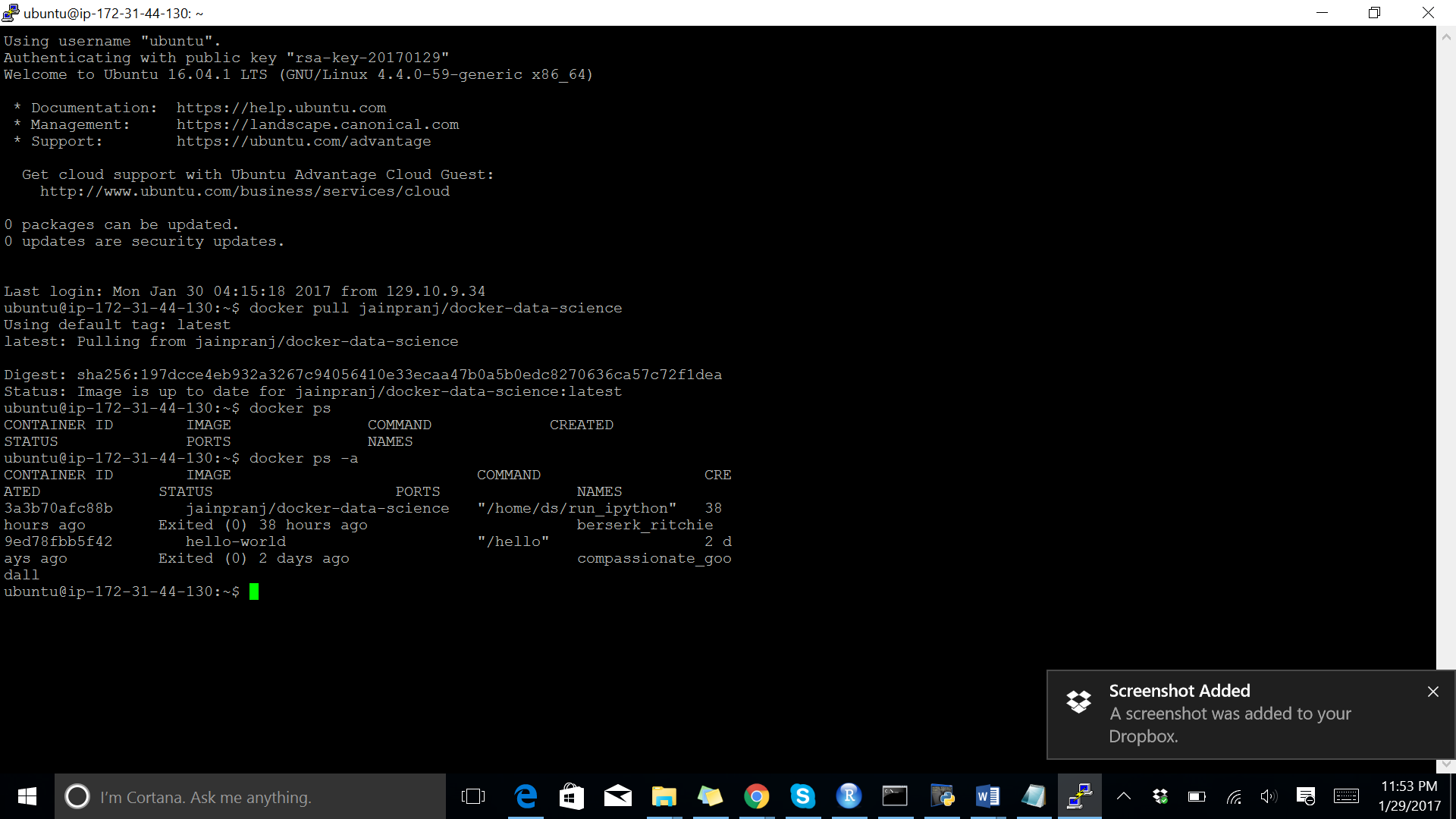
Docker is available on different operating systems like Linux, Ubuntu, Mac OSX and Windows.

Running Docker on AWS provides a highly reliable, low-cost way to quickly build, run, test, and deploy distributed applications at any scale. AWS provides support for Docker open-source and commercial solutions within AWS services.

Using Amazon EC2 install, we can install docker and get started with it.

Steps for using Docker in AWS:

* Launch an EC2 instance with Amazon Linux AMI
* Connect to your instance
* Install docker
* Run docker commands in the AWS instance



**REFERENCES**

1. <https://github.com/docker/docker>
2. <https://en.wikipedia.org/wiki/Operating-system-level_virtualization>
3. <https://docs.docker.com/>
4. <http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html>
5. <https://www.continuum.io/blog/developer-blog/anaconda-and-docker-better-together-reproducible-data-science>
6. <https://www.dataquest.io/blog/docker-data-science/>
7. Cloud Computing  by Professor Dino Konstantopoulos