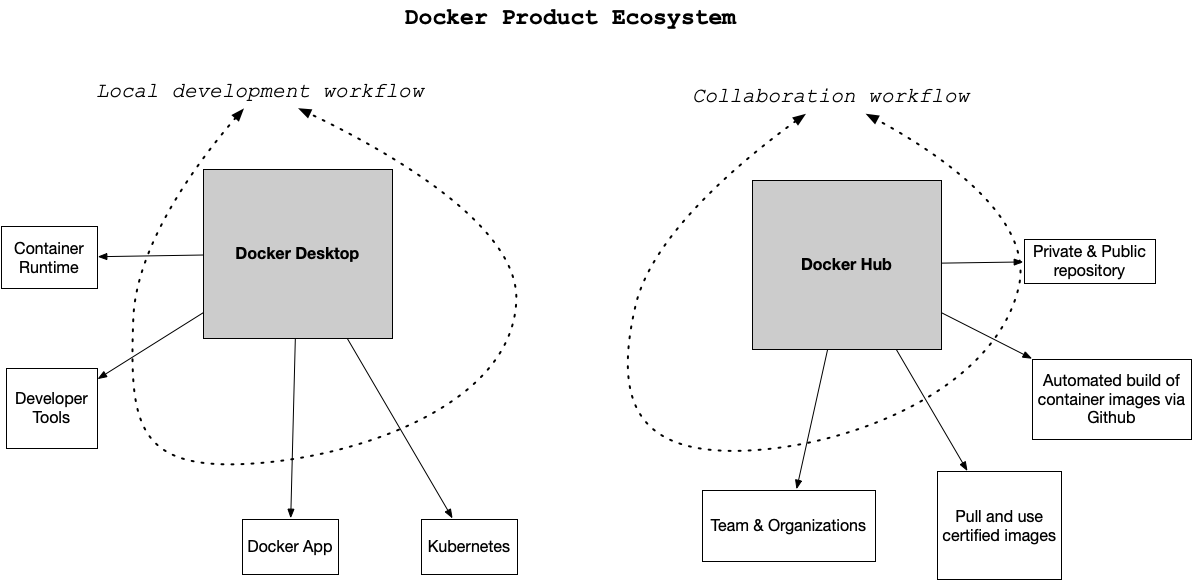
**Docker Containers**

**Getting started with Docker**

There are two main components of Docker: [Docker Desktop](https://www.docker.com/products/docker-desktop) and [Docker Hub](https://www.docker.com/products/docker-hub).



**Docker Desktop Overview**

The desktop application contains the container runtime which allows containers to execute. The Docker App itself orchestrates the local development workflow including the ability to use [Kubernetes](https://github.com/kubernetes/kubernetes" \t "_blank), which is an open-source system for managing containerized applications that came out of Google.

**Docker Hub Overview**

So what is Docker Hub and what problem does it solve? Just as the [git](https://git-scm.com/" \t "_blank) source code ecosystem has local developer tools like [vim](https://www.vim.org/" \t "_blank), [emacs](https://www.gnu.org/software/emacs/" \t "_blank), [Visual Studio Code](https://code.visualstudio.com/" \t "_blank) or [XCode](https://developer.apple.com/xcode/" \t "_blank) that work with it, Docker Desktop works with Docker containers and allows for local use and development.

When collaborating with git outside of the local environment, developers often use platforms like [Github](https://github.com/) or [Gitlab](https://about.gitlab.com/) to communicate with other parties and share code. [Docker Hub](https://hub.docker.com/) works in a similar way. Docker Hub allows developers to share docker containers that can serve as a base image for building new solutions.

These base images can be built by experts and certified to be high quality: i.e. the [official Python developers have a base image](https://hub.docker.com/_/python). This allows a developer to leverage the expertise of the true expert on a particular software component and improve the overall quality of their container. This is a similar concept to using a library developed by another developer versus writing it yourself.

**Real-World Examples of Containers**

What problem do [Docker format containers](https://docs.docker.com/engine/docker-overview/) solve? In a nutshell, the operating system runtime can be packaged along with the code, and this solves a particularly complicated problem with a long history. There is a famous meme that goes "It works on my machine!". While this is often told as a joke to illustrate the complexity of deploying software, it is also true. Containers solve this exact problem. If the code works in a container, then the container configuration can be checked in as code. Another way to describe this concept is that the actual Infrastructure is treated as code. This is called IaC (Infrastructure as Code).

Here are a few specific examples:

**Developer Shares Local Project**

A developer can work on a web application that uses flask (a popular Python web framework). The installation and configuration of the underlying operating system is handled by the Docker container file. Another team member can checkout the code and use docker run to run the project. This eliminates what could be a multi-day problem of configuring a laptop correctly to run a software project.

**Data Scientist shares Jupyter Notebook with a Researcher at another University**

A data scientist working with [jupyter](https://jupyter.org/) style notebooks wants to share a complex data science project that has multiple dependencies on C, Fortran, R, and Python code. They package up the runtime as a Docker container and eliminate the back and forth over several weeks that occurs when sharing a project like this.

**A Machine Learning Engineer Load Tests a Production Machine Learning Model**

A Machine learning engineer has been tasked with taking a new model and deploying it to production. Previously, they were concerned about how to accurately test the accuracy of the new model before committing to it. The model recommends products to paying customers and, if it is inaccurate, it costs the company a lot of money. Using containers, it is possible to deploy the model to a fraction of the customers, only 10%, and if there are problems, it can be quickly reverted. If the model performs well, it can quickly replace the existing models.

**Why Docker Containers vs Virtual Machines?**

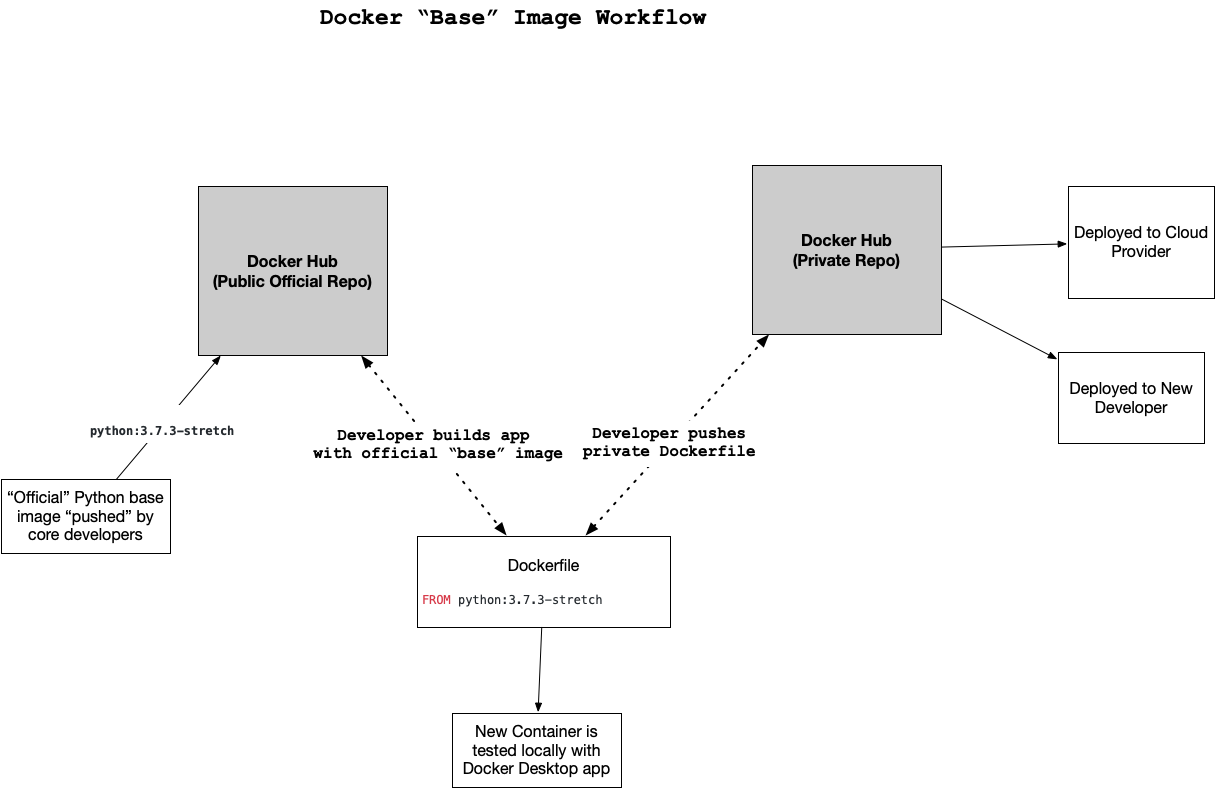
What is the difference between a container and a virtual machine? Here is a breakdown:

* Size: Containers are much smaller than Virtual Machines (VM) and run as isolated processes versus virtualized hardware. VMs can be GBs while containers can be MBs.
* Speed: Virtual Machines can be slow to boot and take minutes to launch. A container can spawn much more quickly typically in seconds.
* Composability: Containers are designed to be programmatically built and are defined as source code in an Infrastructure as Code project (IaC). Virtual Machines are often replicas of a manually built system. Containers make IaC workflows possible because they are defined as a file and checked into source control alongside the project’s source code.

## Running Dockerfiles

### Using "base" images

One of the advantages of the Docker workflow for developers is the ability to use certified containers from the "official" development teams. In this diagram a developer uses the official Python base image which is developed by the core Python developers. This is accomplished by the FROM statement which loads in a previously created container image.



**Docker "Base" Image Workflow**

As the developer makes changes to the Dockerfile, they test locally, then push the changes to a private Docker Hub repo. After this, the changes can be used by a deployment process to a Cloud or by another developer.

### Key Terms:

#### Container

A container is a set of processes that are isolated from the rest of the operating system. They are often megabytes in size.

#### Virtual Machine

A virtual machine is the emulation of a physical operating system. They can be Gigabytes in size.

#### Docker Format Container

There are several formats for containers. An emerging form is Docker, which involves the definition of a Dockerfile.

#### pip

The pip tool installs Python packages.

#### pylint

The pylint tool checks the Python source code for syntax errors.

#### black

The black tool formats the text of Python source code automatically.

#### pytest

The pytest tool is a framework for running tests on Python source code.

#### IPython

The ipython interpreter is an interactive terminal for Python. It is the core of the Jupyter notebook.

#### Makefile

A Makefile is a file that contains a set of directives used to build software. Most Unix and Linux operating systems have built-in support for this file format.

#### CircleCI

A popular SaaS (Software as a Service) build systems used in DevOps workflows.

#### Docker

Docker is a company that creates container technology, including an execution engine, collaboration platform via DockerHub and a container format called Dockerfile.

#### Amazon ECR

Amazon ECR is a container registry that stores Docker format containers.

**AWS Tips**

After creating a Cloud9 environment, try to clone the github repo. We need to create ssh key to Access the repo so run the command below.

**ssh-keygen -t rsa**

When we open it, it will gives a puplic key. Then we need to define that to Github SSH Keys.

If we define an alias in ~/.bashrc, we can use that command system to do what we want.

**alias hello=”cd /usr/bin/hello && source ~/.hello/bin/activate”**

When we type the hello, it will get in the path and activate the python virtual env.

### Common Issues Running a Container

There are a few common issues that crop up when starting a container or building one for the first time. Let's walk through each problem and then present a solution for them.

#### What Goes in a Dockerfile if You Need to [Write to the Host Filesystem](https://docs.docker.com/storage/volumes/" \t "_blank)?

In the following example the docker volume command is used to create a volume and then later it is mounted to the container.

> /tmp docker volume create docker-data

docker-data

> /tmp docker volume ls

DRIVER VOLUME NAME

local docker-data

> /tmp docker run -d \

--name devtest \

--mount source=docker-data,target=/app \

ubuntu:latest

6cef681d9d3b06788d0f461665919b3bf2d32e6c6cc62e2dbab02b05e77769f4

#### How Do You [Configure Logging](https://docs.docker.com/config/containers/logging/configure/" \t "_blank) for a Docker Container?

You can configure logging for a Docker container by selecting the type of log driver, in this example json-file, and whether it is blocking or non-blocking. This example shows a configuration that uses json-file and mode=non-blocking for an Ubuntu container. The non-blocking mode ensures that the application won't fail in a non-deterministic manner. Make sure to read the [Docker logging guide](https://docs.docker.com/config/containers/logging/configure/" \t "_blank) on different logging options.

> /tmp docker run -it --log-driver json-file --log-opt mode=non-blocking ubuntu

root@551f89012f30:/*#*

#### How do You Map Ports to the External Host?

The Docker container has an internal set of ports that [must be exposed to the host and mapped](https://docs.docker.com/engine/reference/commandline/port/" \t "_blank). One of the easiest ways to see what ports are exposed to the host is by running the docker port <container name> command. Here is an example of what that looks like against a foo named container.

$ docker port foo

7000/tcp -> 0.0.0.0:2000

9000/tcp -> 0.0.0.0:3000

What about actually mapping the ports? You can do that using the -p flag as shown. You can read more about [Docker run flags here](https://docs.docker.com/engine/reference/commandline/run/" \t "_blank).

docker run -p 127.0.0.1:80:9999/tcp ubuntu bash

#### What about Configuring Memory, CPU and GPU?

You can configure docker run to accept flags for setting Memory, CPU and GPU. You can read [more about it here](https://docs.docker.com/config/containers/resource_constraints/" \t "_blank) in the official documentation. Here is a brief example of setting the CPU.

docker run -it --cpus=".25" ubuntu /bin/bash

This tells this container to use at max only 25% of the CPU every second.

**Build and Deploy**

### Instructions

* Run docker build --tag=api . from the directory containing your Dockerfile. If you want to use a different tag name, feel free to do so.
* Wait for awhile as your Docker image is built (note: you are welcome to get a jump start on the next exercise, if desired, while this completes).
* Use docker image ls to make sure your new Docker image is shown. You won't see the other containers in Noah's video - those are all of his other Docker images on his computer.
* Run docker run -p 8000:5001 api. If you changed the tag name in the first step, make sure to replace api here with your tag name. Note above that the -p notes port 5001 from the Docker container (as specified in web.py for our flask app) is exposed on port 8000 on the host computer.
* The container will tell us the Flask app is running on port 5001, but since we exposed port 8000 on our host to it, we will actually access the running app using port 8000. We haven't looked at Swagger documentation before here, but you can access it at http://localhost:8000 in your browser when the docker container is running. Note that Swagger is part of the implementation of this specific Flask app - if you make your own Flask app, Swagger won't be included unless you include it in your own code.
* Test out one of the Swagger commands from the running containerized app.
* HINT: Remember that you can access the underlying container by opening up a web browser on your host machine and typing in http://localhost:8000. The host container maps a port to the internal container and this is how the web service is exposed to your browser.

### More on Swagger

[Swagger](https://swagger.io/solutions/api-documentation/) helps provide automated documentation for your APIs using the [OpenAPI specifications](https://github.com/OAI/OpenAPI-Specification).

Check out an example of a Swagger-based API [here](https://petstore.swagger.io/#/). Udacity actually uses Swagger for its internal-facing APIs!

### Key Terms:

#### Swagger

A swagger tool is an open-source framework that simplifies the creation of API documentation.

#### Data Engineering

Data Engineering is the process of automating the flow of data.

#### Ports

A port is a network communication endpoint. An example of a port is a web service running on port 80 via the protocol HTTP.

#### JSON

JSON stands for JavaScript Object Notation, and it is a lightweight, human-readable data format used heavily in web services.