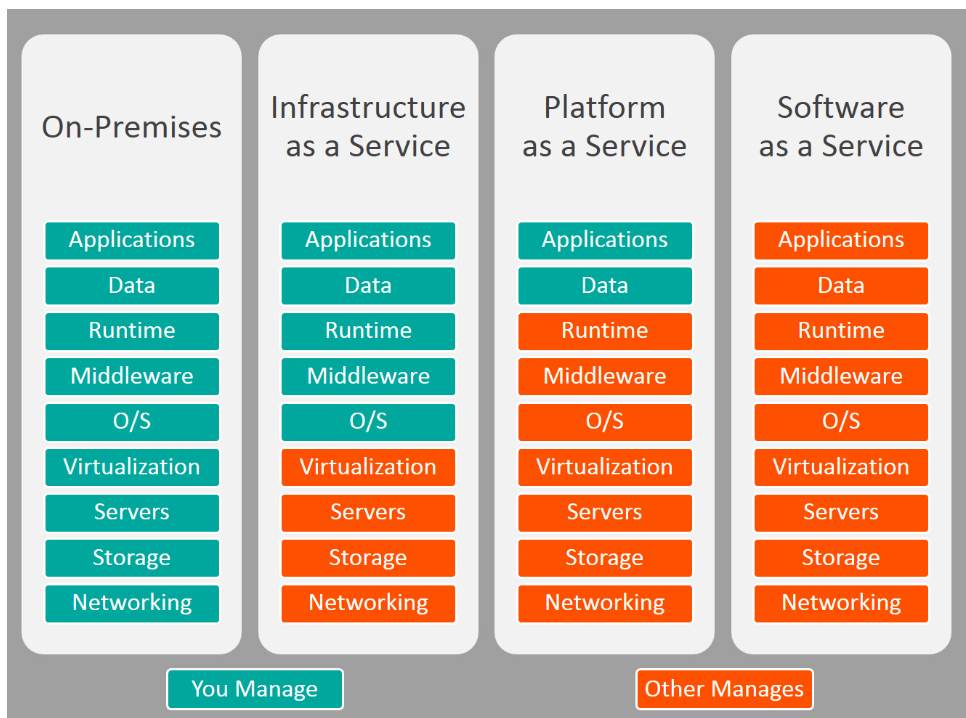
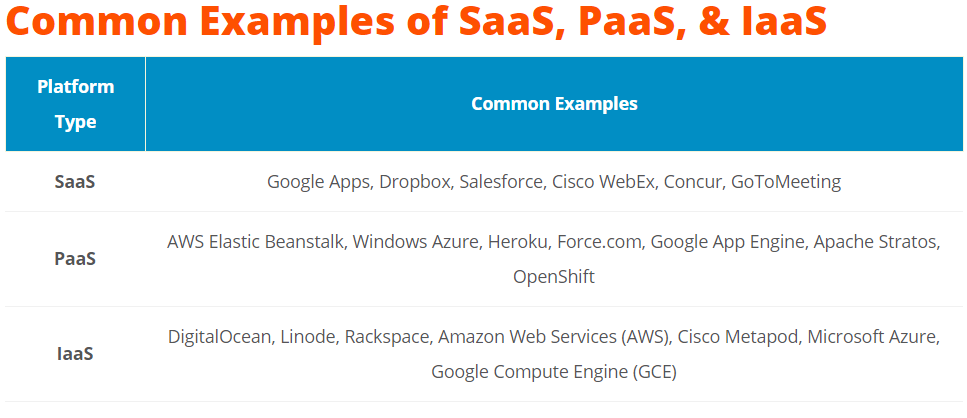
While the cloud is an extremely hot topic from small businesses all the way to global enterprises, it is still a pretty broad concept that covers a lot of online territory. As you begin to consider switching your business to the cloud, whether it be for application or infrastructure deployment, it is more important than ever to understand the differences and advantages of the various cloud services.

There are usually three models of cloud service to compare: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Each of these has its own benefits as well variances and it is necessary to understand the differences among SaaS, PaaS, and IaaS to know how to best choose one for your organization.

**Summary of Key Differences**



**Common Examples of SaaS, PaaS, & IaaS**

**SaaS: Software as a Service**

Software as a Service, also known as cloud application services, represent the most commonly utilized option for businesses in the cloud market. SaaS utilizes the internet to deliver applications to its users, which are managed by a third-party vendor. A majority of SaaS applications are run directly through the web browser, and do not require any downloads or installations on the client side.

**SaaS Delivery**

Due to its web delivery model, SaaS eliminates the need to download and install applications on each individual computer; a nightmare for IT staff. With SaaS, vendors manage all of the potential technical issues, such as data, middleware, servers, and storage, while businesses can simply streamline their maintenance and support.

**SaaS Advantages**

SaaS provides numerous advantages to employees and companies by greatly reducing the time and money spent on tedious tasks such as installing, managing, and upgrading software. This frees up a lot of time for technical staff to spend on more pressing matters and issues within the organization.

**SaaS Characteristics**

There are a few ways to help you determine when SaaS is being utilized:

* Managed from a central location
* Hosted on a remote server
* Accessible over the internet
* Users not responsible for hardware or software updates

**When to Use SaaS**

There are many different situations in which SaaS may be the most beneficial, including:

* If you are a startup or small company that needs to launch ecommerce quickly and don’t have time for server issues or software
* For short-term projects that require collaboration
* If you use applications that aren’t in-demand very often, such as tax software
* For applications that need both web and mobile access

**Examples of SaaS**

Google Apps, Dropbox, Salesforce, Cisco WebEx, Concur, GoToMeeting

**PaaS: Platform as a Service**

Cloud platform services, or Platform as a Service (PaaS), provide cloud components to certain software while being used mainly for applications. PaaS provides a framework for developers that they can build upon and use to create customized applications. All servers, storage, and networking can be managed by the enterprise or a third-party provider while the developers can maintain management of the applications.

**PaaS Delivery**

The delivery model of PaaS is similar to SaaS, except instead of delivering the software over the internet, PaaS provides a platform for software creation. This platform is delivered over the web, and gives developers the freedom to concentrate on building the software while still not having to worry about operating systems, software updates, storage, or infrastructure.

PaaS allows businesses to design and create applications that are built into the PaaS with special software components. These applications, or middleware, are scalable and highly available as they take on certain cloud characteristics.

**PaaS Advantages**

No matter what size of company you may be in, there are numerous advantages for using PaaS:

* Makes the development and deployment of apps simple and cost-effective
* Scalable
* Highly available
* Gives developers the ability to create customized apps without the headache of maintaining the software
* Greatly reduces the amount of coding
* Automates business policy
* Allows easy migration to the hybrid model

**PaaS Characteristics**

PaaS has many characteristics that define it as a cloud service, including:

* It is built on virtualization technology, meaning resources can easily be scaled up or down as your business changes
* Provides a variety of services to assist with the development, testing, and deployment of apps
* Numerous users can access the same development application
* Web services and databases are integrated

**When to Use PaaS**

There are many situations that utilizing PaaS is beneficial or even necessary. If there are multiple developers working on the same development project, or if other vendors must be included as well, PaaS can provide great speed and flexibility to the entire process. PaaS is also beneficial if you wish to be able to create your own customized applications. This cloud service also can greatly reduce costs and it can simplify some challenges that come up if you are rapidly developing or deploying an app.

**Examples of PaaS**

AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos, OpenShift

**IaaS: Infrastructure as a Service**

Cloud infrastructure services, known as Infrastructure as a Service (IaaS), are made of highly scalable and automated compute resources. IaaS is fully [self-service](http://www.bmc.com/blogs/self-service-thrives-clouds/) for accessing and monitoring things like compute, networking, storage, and other services, and it allows businesses to purchase resources on-demand and as-needed instead of having to buy hardware outright.

**IaaS Delivery**

IaaS delivers Cloud Computing infrastructure to organizations, including things such as servers, network, operating systems, and storage, through virtualization technology. These cloud servers are typically provided to the client through a dashboard or an API, and IaaS clients have complete control over the entire infrastructure. IaaS provides the same technologies and capabilities as a traditional data center without having to physically maintain or manage all of it. IaaS clients can still access their servers and storage directly, but it is all outsourced through a “virtual data center” in the cloud.

As opposed to SaaS or PaaS, IaaS clients are responsible for managing aspects such as applications, runtime, OSes, middleware, and data. However, providers of the IaaS manage the servers, hard drives, networking, virtualization, and storage. Some providers even offer more services outside of the virtualization layer, such as databases or message queuing.

**IaaS Advantages**

There are many benefits of choosing IaaS, such as that it:

* Is the most flexible cloud computing model
* Easily allows for automated deployment of storage, networking, servers, and processing power
* Hardware can be purchased based on consumption
* Gives clients complete control of their infrastructure
* Resources can be purchased as-needed
* Is highly scalable

**IaaS Characteristics**

Some characteristics to look for when considering IaaS are:

* Resources are available as a service
* The cost varies depending on consumption
* Services are highly scalable
* Typically includes multiple users on a single piece of hardware
* Provides complete control of the infrastructure to organizations
* Dynamic and flexible

**When to Use IaaS**

Just as with SaaS and PaaS, there are specific situations when it is the most advantageous to use IaaS. If you are a startup or a small company, IaaS is a great option so you don’t have to spend the time or money trying to create hardware and software. IaaS is also beneficial for large organizations who wish to have complete control over their applications and infrastructures, but are looking to only purchase what is actually consumed or needed. For rapidly growing companies, IaaS can be a good option as you don’t have to commit to a specific hardware or software as your needs change and evolve. It also helps if you are unsure what demands a new application will need as there is a lot of flexibility to scale up or down as needed.

**Examples of IaaS**

DigitalOcean, Linode, Rackspace, Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)

**SaaS vs PaaS vs IaaS**

Overall, each cloud model offers its own specific features and functionalities, and it is crucial for your organization to understand the differences. Whether you are looking for cloud-based software for storage options, a smooth platform that allows you to create customized applications, or are wanting complete control over your entire infrastructure without having to physically maintain it, there is a cloud service for you. No matter which option you choose, [migrating to the cloud](http://www.bmc.com/blogs/new-multi-cloud-world-means/) is the future of business and technology as we know it, and it is necessary to be properly informed.

**Emulation** refers to the ability of a computer program in an electronic device to **emulate** (or imitate) another program or device. Many printers, for example, are designed to **emulate** Hewlett-Packard LaserJet printers because so much software is written for HP printers.

1. Emulation, in a software context, is the use of an [application program](https://searchsoftwarequality.techtarget.com/definition/application-program) or device to imitate the behavior of another program or device.

Common uses of emulation include:

* Running an [operating system](https://whatis.techtarget.com/definition/operating-system-OS) on a hardware platform for which it was not originally engineered.
* Running arcade or console-based games upon desktop computers.
* Running legacy applications on devices other than the ones for which they were developed.
* Running application programs on different operating systems other than those for which they were originally written.
* A common example of that last type of emulation is running Windows applications on [Linux](https://searchdatacenter.techtarget.com/definition/Linux-operating-system) computers. [Virtual PC](https://searchservervirtualization.techtarget.com/definition/Virtual-PC) is another example of an [emulator](https://whatis.techtarget.com/definition/emulator) that allows Macs to run [Windows XP](https://searchenterprisedesktop.techtarget.com/definition/Windows-XP), though the addition of [Boot Camp](https://whatis.techtarget.com/definition/Boot-Camp) to next-generation Intel-based Macs has removed the need for that application in the Macintosh environment in the future.

2. In server virtualization, emulation is a synonym for virtual environment. A virtual environment can also be referred to as a partition, guest, instance or container.

3. In hardware, emulation is the use of hardware to imitate the function of another hardware device for the purpose of connecting devices to one another or connecting to a [mainframe](https://searchdatacenter.techtarget.com/definition/mainframe) computer.

Hardware emulation can be useful when a hardware device or peripheral like a printer, scanner or digital camera becomes obsolete. A printer emulator, for instance, lies within the printer's ROM. In many organizations, legacy applications persist that must still communicate with the device. For this purpose, [terminal emulation](https://searchnetworking.techtarget.com/definition/terminal-emulation) allows a computer [terminal](https://searchnetworking.techtarget.com/definition/terminal) to appear and function like another, usually older type of terminal so that a user can access legacy programs.

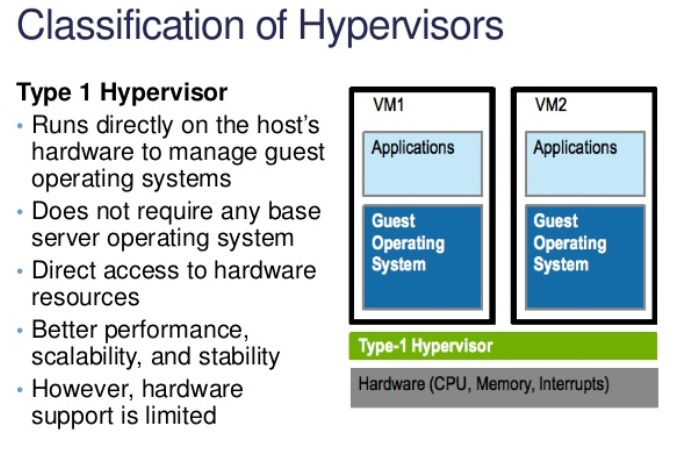
While in theory any application may be rewritten to work on another device or architecture, in practice such [reverse engineering](https://searchsoftwarequality.techtarget.com/definition/reverse-engineering) is quite difficult to accomplish successfully, resulting in significant performance penalties or crashes due to [bug](https://searchsoftwarequality.techtarget.com/definition/bug)gy programming.

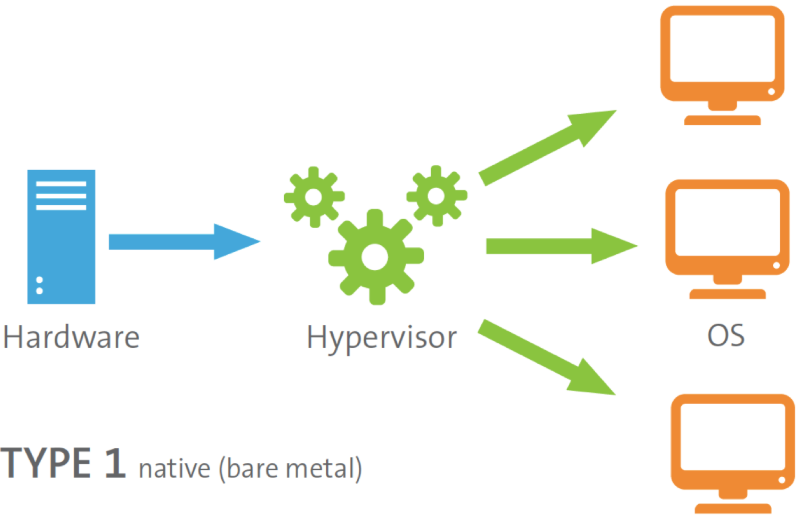
A **hypervisor** or virtual machine monitor (VMM) is computer software, firmware or hardware that creates and runs virtual machines. A computer on which a **hypervisor** runs one or more virtual machines is called a host machine, and each virtual machine is called a guest machine.

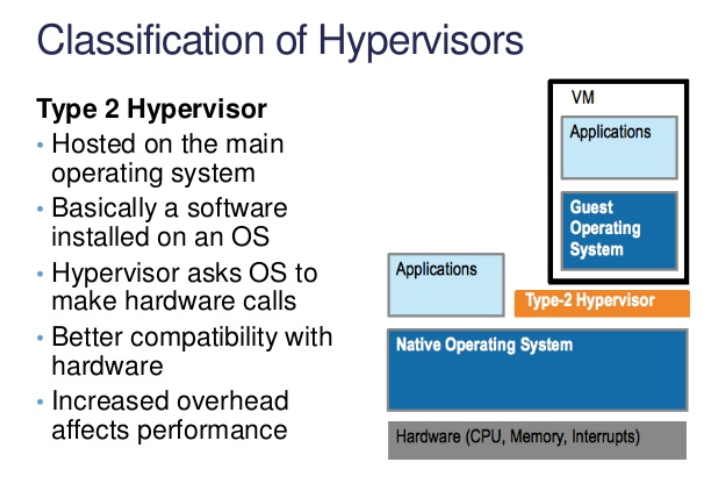
In computing, **virtualization** means to create a virtual version of a device or resource, such as a server, storage device, network or even an operating system where the framework divides the resource into one or more execution environments.

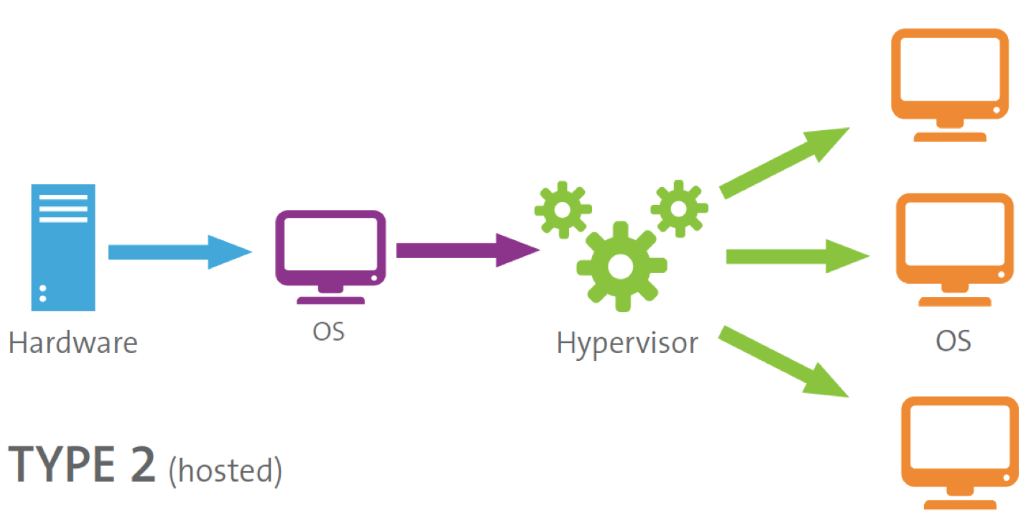
"**Cloud computing** is the delivery of shared **computing** resources, software or data — as a service and on-demand through the Internet." Most of the confusion occurs because **virtualization** and **cloud computing** work together to provide different types of services, as is the case with private clouds

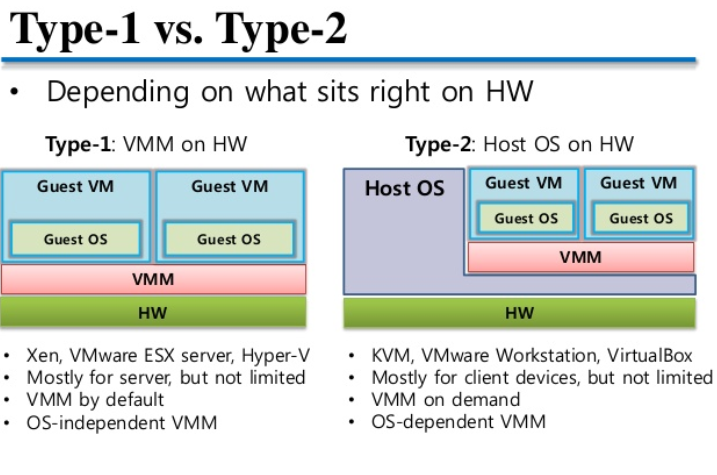
The most **important** function of **virtualization** is the capability of running multiple operating systems and applications on a single computer or server. This means increased productivity achieved by fewer servers

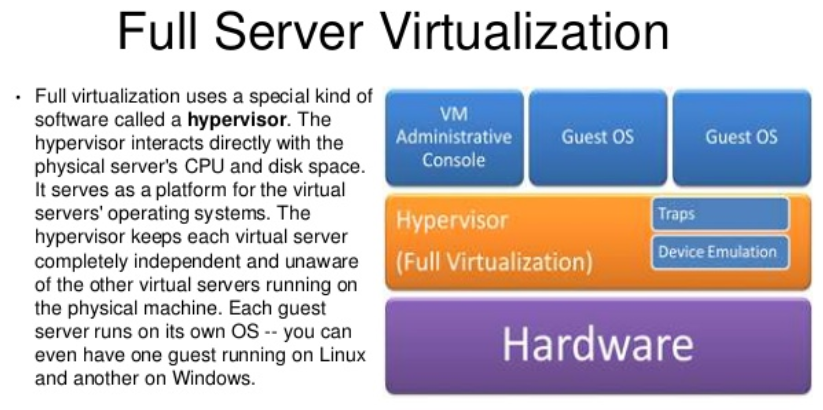


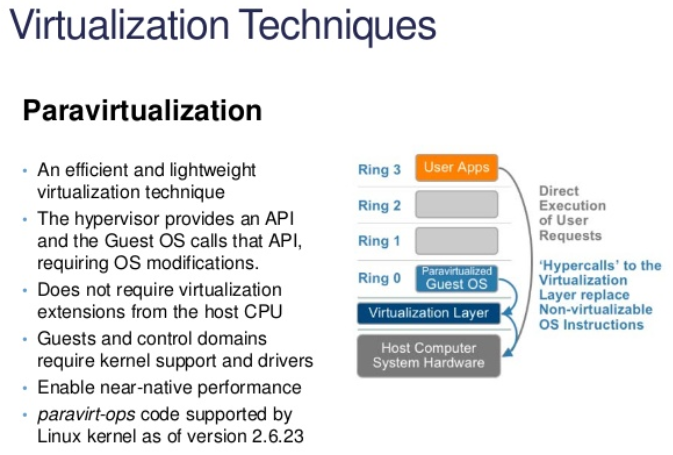


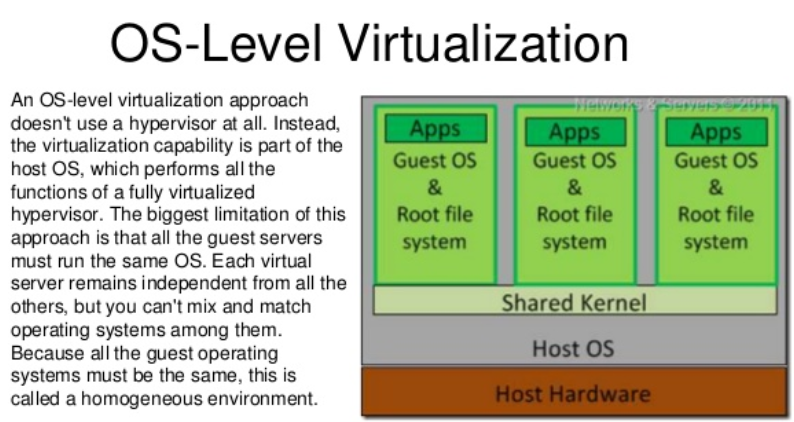


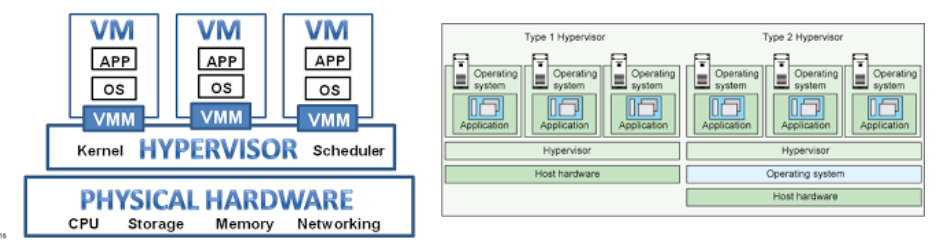




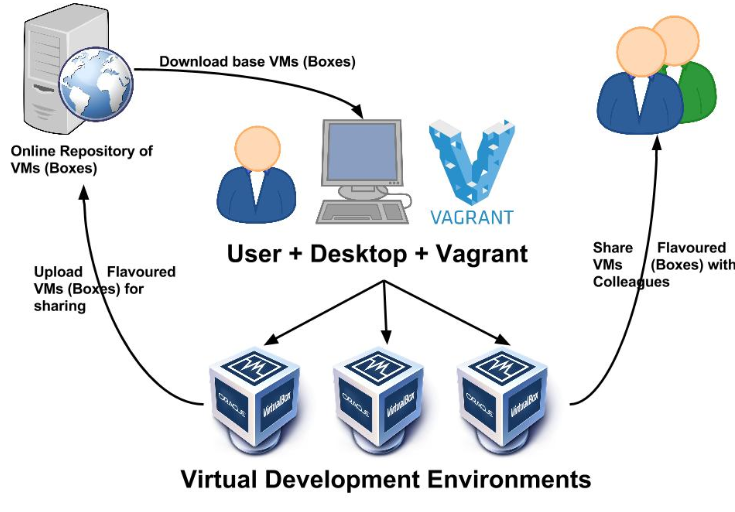


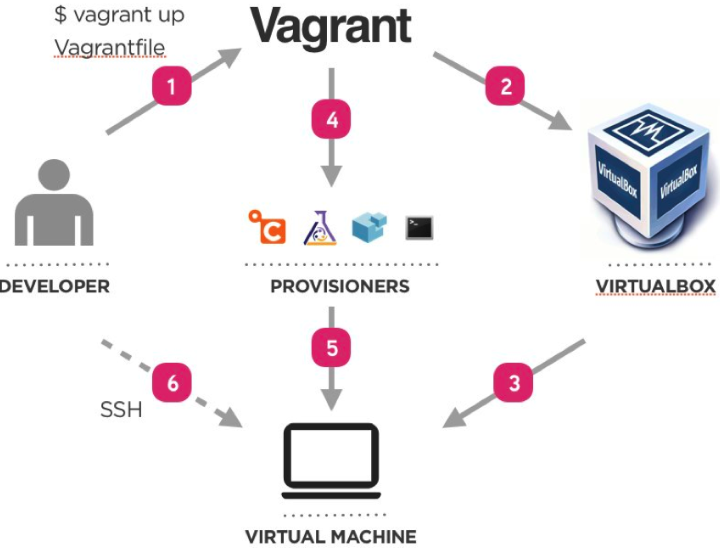






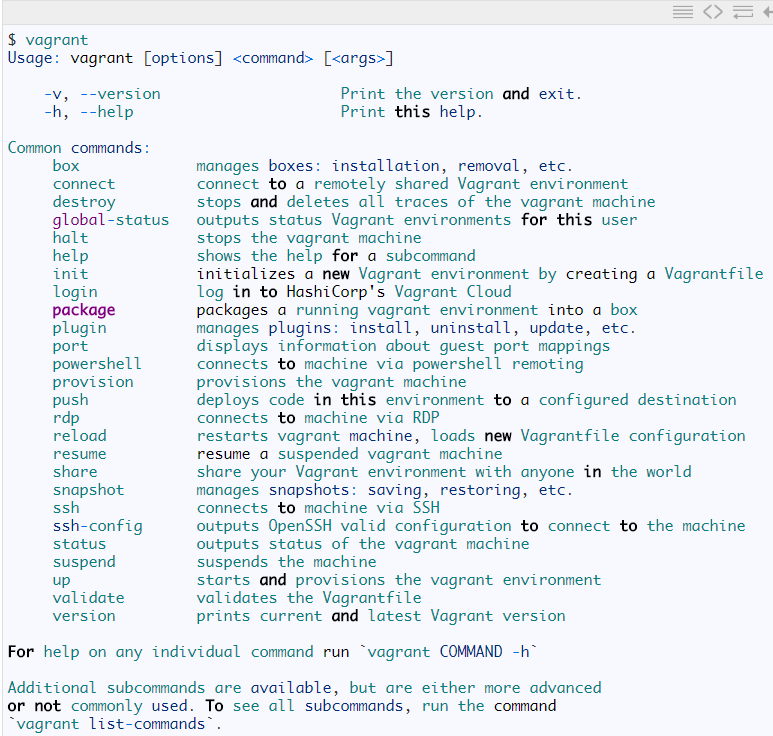
**Vagrant** is an [open-source software](https://en.wikipedia.org/wiki/Open-source_software) product for building and maintaining [portable](https://en.wikipedia.org/wiki/Software_portability) [virtual](https://en.wikipedia.org/wiki/Virtualization) software development environments, e.g. for [VirtualBox](https://en.wikipedia.org/wiki/VirtualBox" \o "VirtualBox), [Hyper-V](https://en.wikipedia.org/wiki/Hyper-V), [Docker containers](https://en.wikipedia.org/wiki/Docker_(software)), [VMware](https://en.wikipedia.org/wiki/VMware), and [AWS](https://en.wikipedia.org/wiki/AWS_EC2) which try to simplify [software configuration management](https://en.wikipedia.org/wiki/Software_configuration_management) of [virtualizations](https://en.wikipedia.org/wiki/Hardware_virtualization) in order to increase development productivity. Vagrant is written in the [Ruby language](https://en.wikipedia.org/wiki/Ruby_(programming_language)), but its ecosystem supports development in a few languages.

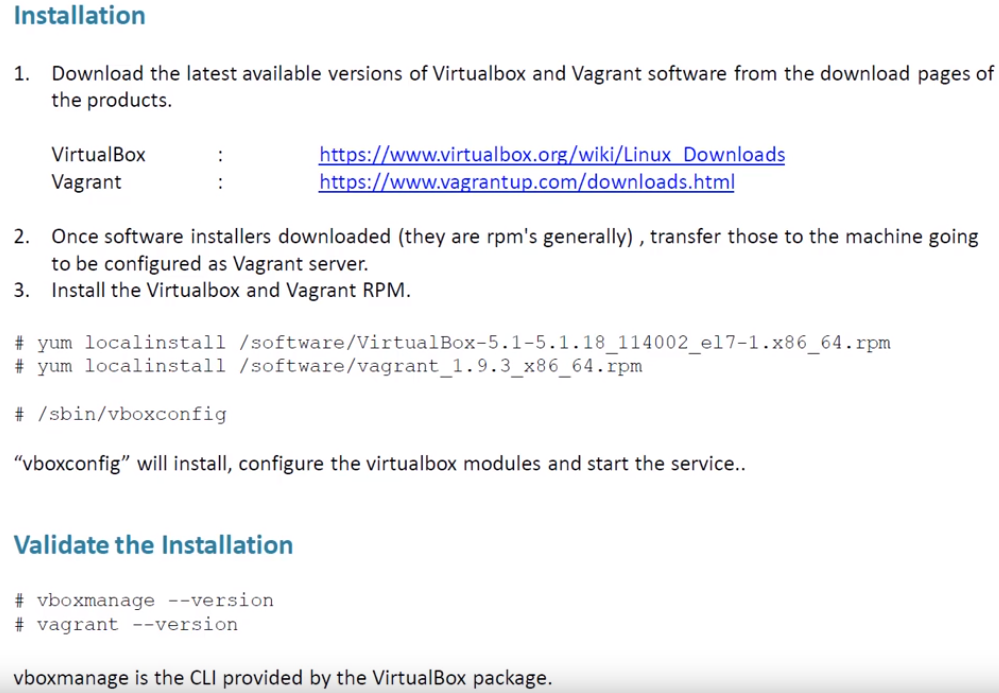


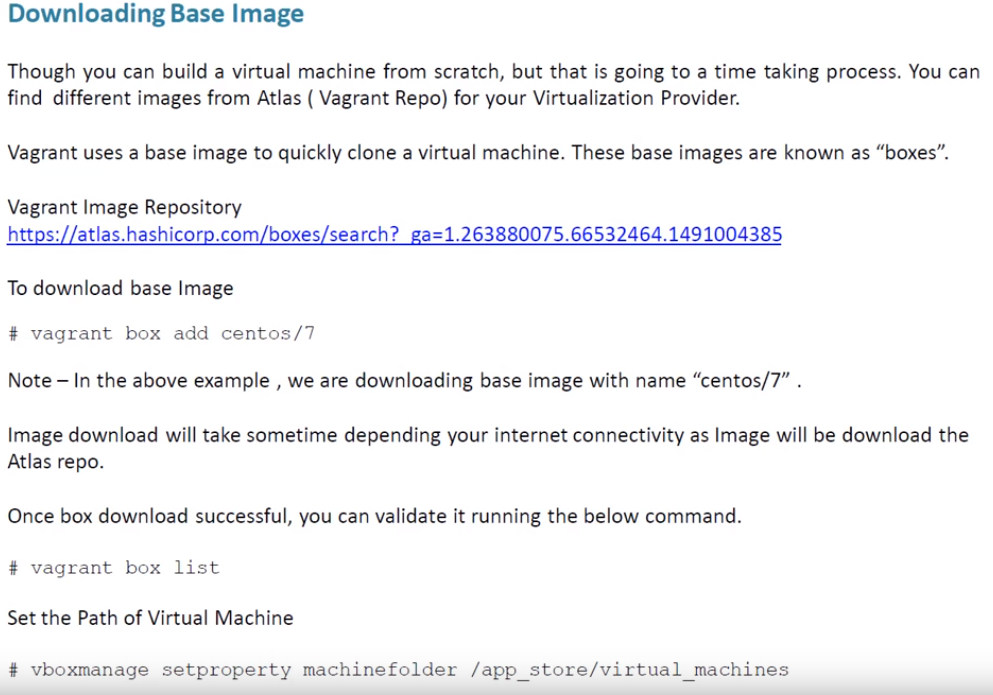


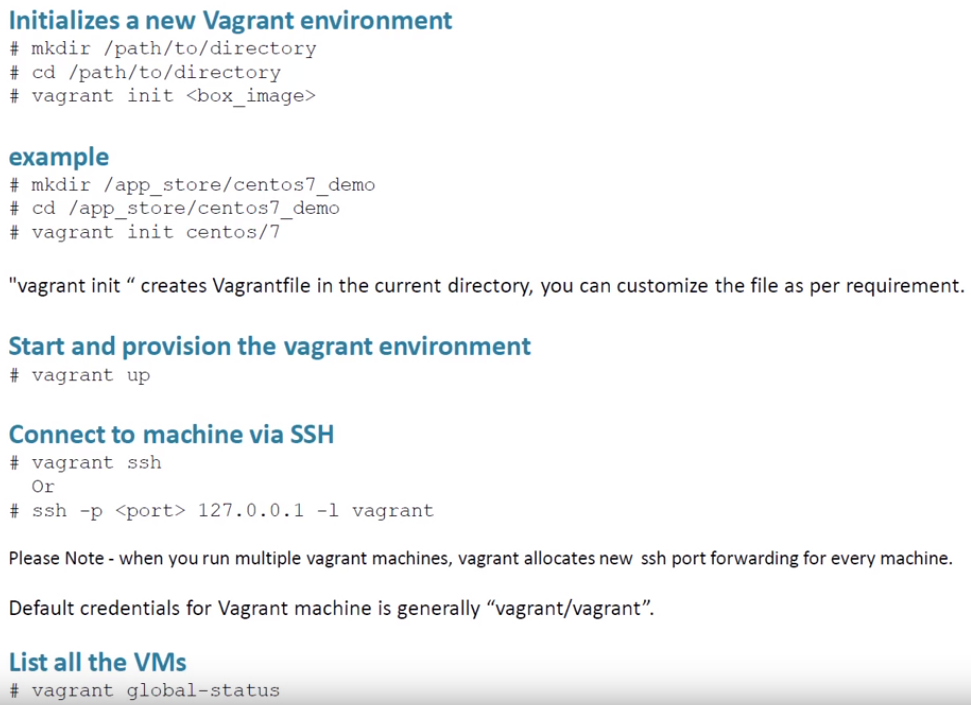
<https://www.vagrantup.com/downloads.html>

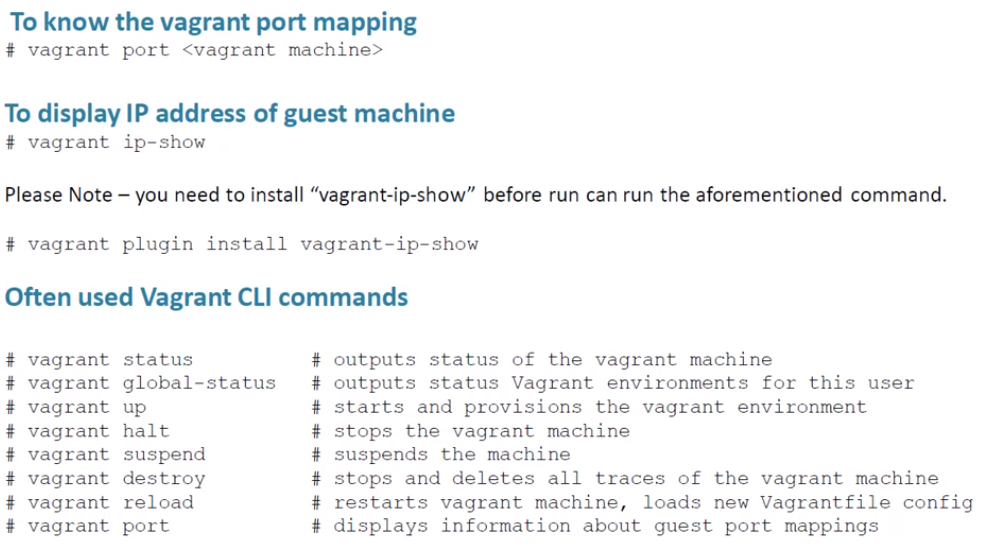
<https://www.youtube.com/watch?v=IXjhfsIocnw>





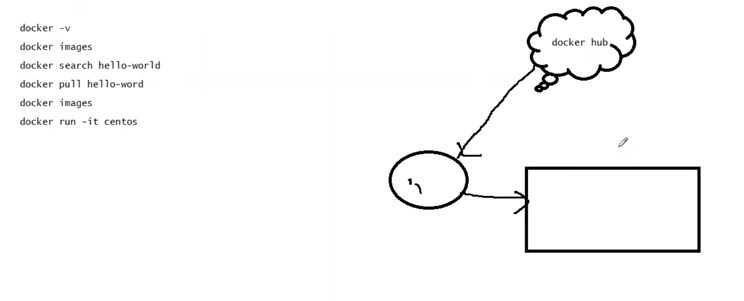


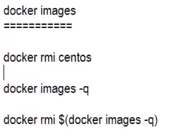
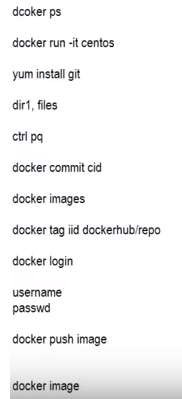


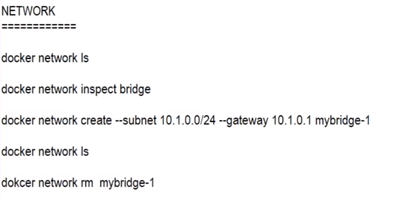


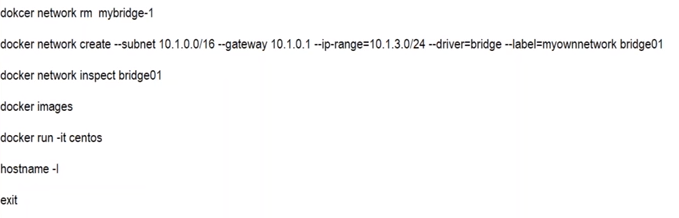
Docker training in telugu by Prasad good session

**Docker user – someshdevops2018/ pass – Welcome123# https://hub.docker.com/r/someshdevops2018/test/**

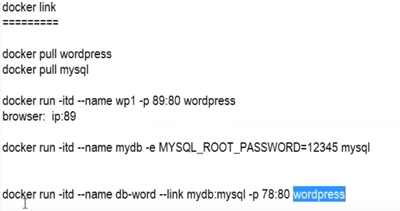
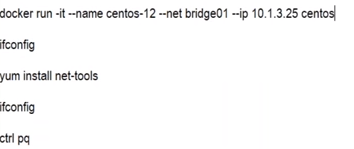
<https://hub.docker.com/> <https://www.youtube.com/watch?v=PU_UU6uHRg4&t=416s>



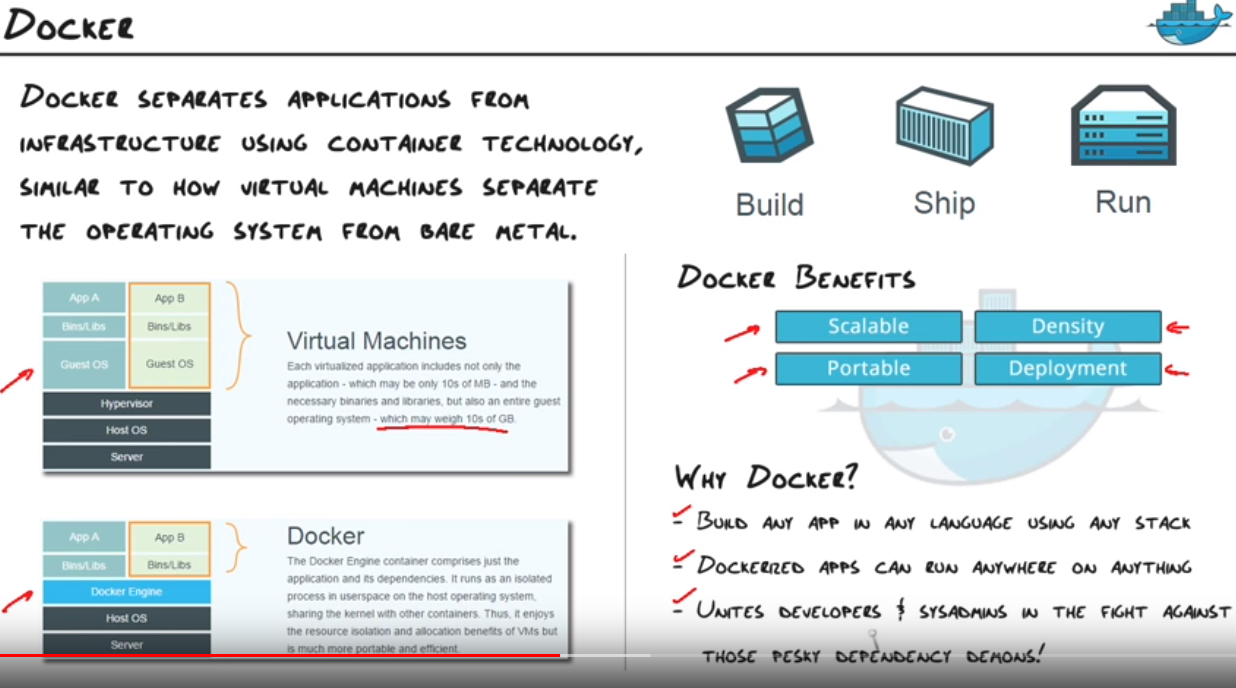
command – docker network rm

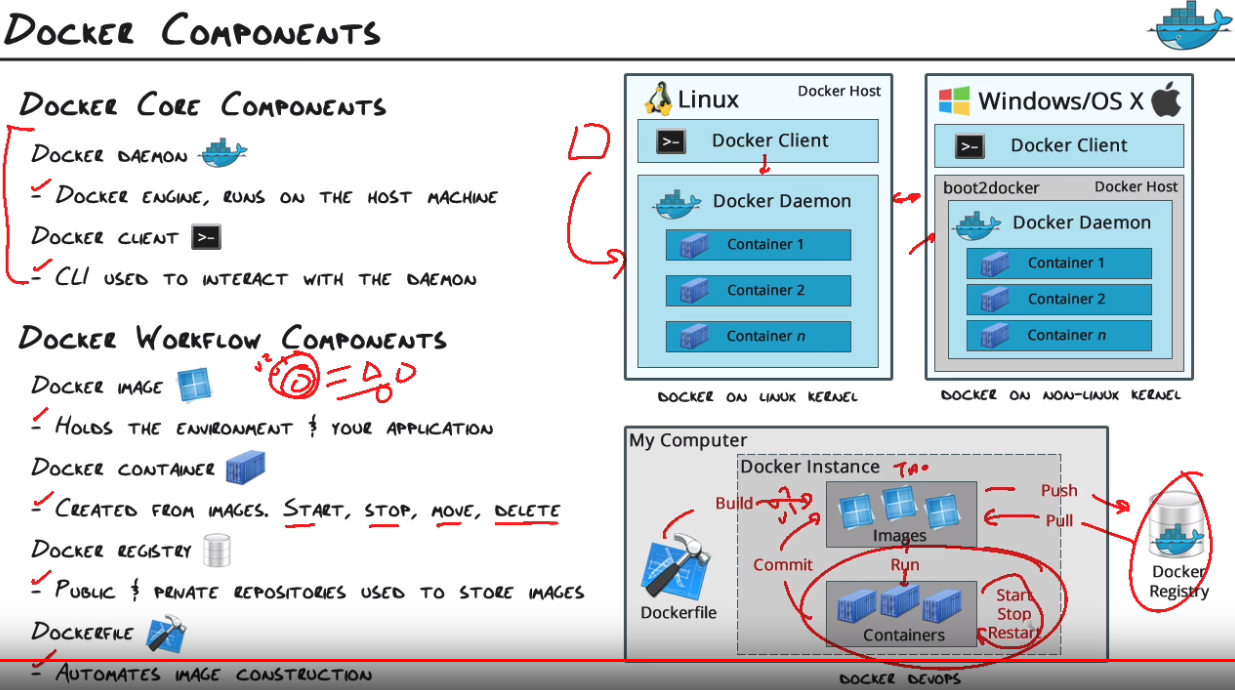


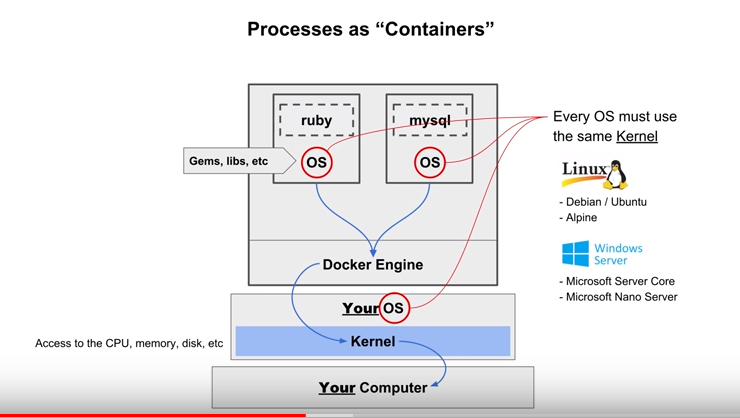
docker run -it --name 1centos --net bridge01 centos



https://www.youtube.com/watch?v=aLipr7tTuA4









Let’s clarify some terminology that is used frequently in the Docker ecosystem.

* Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run docker inspect alpine. In the demo above, you used the docker pull command to download the **alpine** image. When you executed the command docker run hello-world, it also did a docker pull behind the scenes to download the **hello-world** image.
* Containers - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS. You created a container using docker run which you did using the alpine image that you downloaded. A list of running containers can be seen using the docker ps command.
* Docker daemon - The background service running on the host that manages building, running and distributing Docker containers.
* Docker client - The command line tool that allows the user to interact with the Docker daemon.
* Docker Store - A registry of Docker images, where you can find trusted and enterprise ready containers, plugins, and Docker editions. You'll be using this later in this tutorial.

An important distinction with regard to images is between base images and child images.

* **Base images** are images that have no parent images, usually images with an OS like ubuntu, alpine or debian.
* **Child images** are images that build on base images and add additional functionality.

Another key concept is the idea of official images and user images. (Both of which can be base images or child images.)

* **Official images** are Docker sanctioned images. Docker, Inc. sponsors a dedicated team that is responsible for reviewing and publishing all Official Repositories content. This team works in collaboration with upstream software maintainers, security experts, and the broader Docker community. These are not prefixed by an organization or user name. In the list of images above, the python, node, alpine and nginx images are official (base) images. To find out more about them, check out the [Official Images Documentation](https://docs.docker.com/docker-hub/official_repos/).
* **User images** are images created and shared by users like you. They build on base images and add additional functionality. Typically these are formatted as user/image-name. The user value in the image name is your Docker Store user or organization name.

Quick summary of the few basic commands we used in our Dockerfile.

* FROM starts the Dockerfile. It is a requirement that the Dockerfile must start with the FROM command. Images are created in layers, which means you can use another image as the base image for your own. The FROM command defines your base layer. As arguments, it takes the name of the image. Optionally, you can add the Docker Cloud username of the maintainer and image version, in the format username/imagename:version.
* RUN is used to build up the Image you're creating. For each RUN command, Docker will run the command then create a new layer of the image. This way you can roll back your image to previous states easily. The syntax for a RUN instruction is to place the full text of the shell command after the RUN (e.g., RUN mkdir /user/local/foo). This will automatically run in a /bin/sh shell. You can define a different shell like this: RUN /bin/bash -c 'mkdir /user/local/foo'
* COPY copies local files into the container.
* CMD defines the commands that will run on the Image at start-up. Unlike a RUN, this does not create a new layer for the Image, but simply runs the command. There can only be one CMD per a Dockerfile/Image. If you need to run multiple commands, the best way to do that is to have the CMD run a script. CMD requires that you tell it where to run the command, unlike RUN. So example CMD commands would be:

CMD ["python", "./app.py"]

CMD ["/bin/bash", "echo", "Hello World"]

* EXPOSE creates a hint for users of an image which ports provide services. It is included in the information which can be retrieved via $ docker inspect <container-id>.

**Note:** The EXPOSE command does not actually make any ports accessible to the host! Instead, this requires publishing ports by means of the -p flag when using $ docker run.

* PUSH pushes your image to Docker Cloud, or alternately to a [private registry](https://docs.docker.com/registry/)

**Note:** If you want to learn more about Dockerfiles, check out [Best practices for writing Dockerfiles](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/).

<https://github.com/docker/labs>

**Jenkins**

**Jenkins** is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Mac OS X and other Unix-like operating systems.

Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place. The following flowchart demonstrates a very simple workflow of how Jenkins works.

Default below Jenkins packages installed in 2.129

