**Welcome**

- Docker is sweeping the software industry and changing how applications are deployed by simplifying the overall application delivery experience. It increases your ability to deliver software more rapidly, and provides a common platform for dev and ops to collaborate. It also maximizes efficiency by integrating seamlessly with existing tool chains.

You'll learn: how to run your very first container,

 build your very first image and most commonly-used options around them.

how to deploy multi-container applications on multiple hosts and take them from dev to production.

  monitoring Docker containers as well.

**What you should know**

this course is targeted toward Java developers, but you can certainly apply those concepts towards your own programing language as well….let's say I'm using a Wildfly image, which is a Java application software, but you want to use a Node.js server, you are welcome to replace it with your own programing language in that sense.

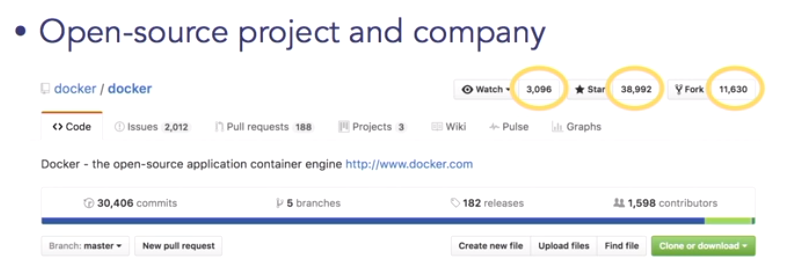
### How to use the GitHub repo

arun-gupta/docker-for-java

**Introduction to Docker**

Company Dark Cloud was changed to the name Docker and now is an open-source project

Literally on GitHub. So it's http://www.github.com/docker. Just look at the number of watches, number of stars, and number of forks on the GitHub repo.



**Well, think of it as an application deployment, or application delivery technology. It is used to build, deploy, and run containers for software applications.**

What are containers good for? Well, go back to the analogy many, many years ago when a wholesaler will produce goods and he has to send it to the actual retailer. Well, the wholesaler will produce goods. He will pack it in whatever form he can.It will go on a cart, go to the shipping container, or go to the shipyard basically, and from there the person who is responsible for taking the ship will collect hundreds of such containers from everybody in different shapes and forms, then he will figure out how the stacking needs to happen, then he will go across the sea, then he will unboard it. And then again, it goes in different carts, or trucks, or trains, and so on.



That is a nightmare for the shipment guy, because he has to accommodate for different formats. Take the same analogy for software application.Well, before that, what they did is, they essentially said, "You know what? You can bring whatever goods you want to bring to me, but I am only going to take these goods in a certain container format.

I'm only going to take in container boxes. I don't care what you bring in the container, but this is the container that I'm going to take." That makes it a lot easier, not just to ship the goods from the wholesaler to the ship, shipping in the ship, and then from the ship to the retailer.

-🡪Take that analogy for software. Now, we want to build software, essentially. What Docker gives you is a standard way to package your software, deploy the software, and run the software. What goes in the container, which is called as a container, it is completely your application developer’s responsibility, but Docker just gives you that standard format by which you can easily do this.

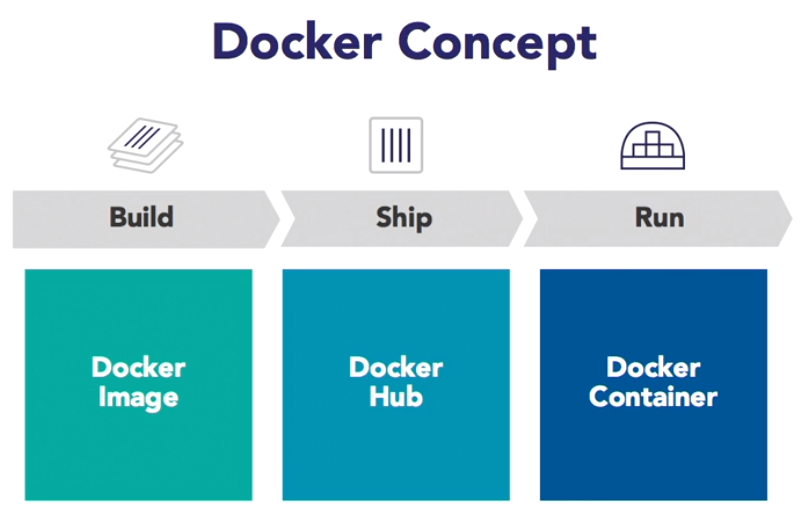
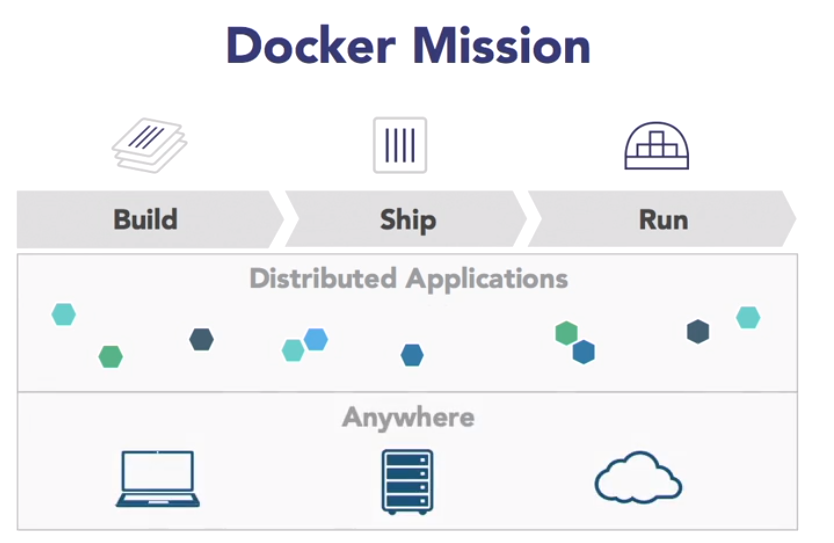
So, what is Docker Mission at a very high level? Well, essentially it wants you to be able to build, ship, and run distributed applications.

Think about this. Now, when you are building an application you have multiple components to the application. You have a web server. You have an application server. You have a database server. You have a caching layer. You have a messaging server. Multiple components. And, typically, you want to run these applications in a distributed way. Why would you run in a distributed? Because you want to avoid a single point of failure.

And for a web server, or a database server, or an application server, you will run multiple instances of those. And you want to run multiple instances again, to avoid a single point of failure. Also, to be able to meet the scalability needs of your application. So that in case more database concurrent accesses are required you can do the horizontal sharding as some of the databases allow. And then these distributed applications should be completely transparentwhether they are running on your laptop, or on Cloud, or a on-premise data center.

-🡪 So if you think about Docker, the mission is to build, ship, and run distributed applications across anywhere. Whether it's your local laptop, whether it's your Cloud, or on-premise

"Here is what my application component going to look like, and here is what the configuration of the application look like, and I'm going to package this together, and I'm going to call that as Docker Image." Now once you have built a Docker image, you want to share it with somebody else, or you want to run it in production, or you want to give it to somebody else who can then exactly try the same image.

****

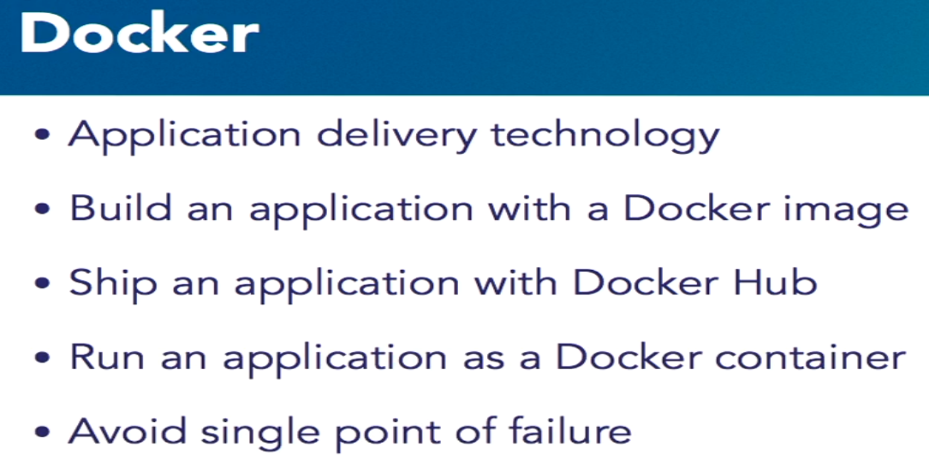
For that, the shipment concept comes in. Now, by default, that is Docker Hub, or on https://www.hub.docker.com, So once you have built the image, you can share the image using Docker Hub, and once you are at Docker Hub, you can look at what the image looks like, and then you can download the image from Docker Hub. By default, as I said, there's a Docker Hub,but there are lots of other options by which you can run a private registry, inside firewall, outside firewall, depending upon the access and stuff like that, that can be easily done.

And finally, once you have shared the application, then you want to be able to run the application.

So think of Docker Image as the build-time component and when you want to run the image, that's called as a Docker Container. So, you might have built an application, which you'll say, for example, using WildFly. I know WildFly's... That has, open source application server. So you have, say WildFly in there and in the WildFly you have (mumbles) application. You will have a single image, but if you want to run a container out of it you may want to be able to run multiple instances of that application.

So you will ship the application once, but once you are running the application, you may run multiple containers of it.

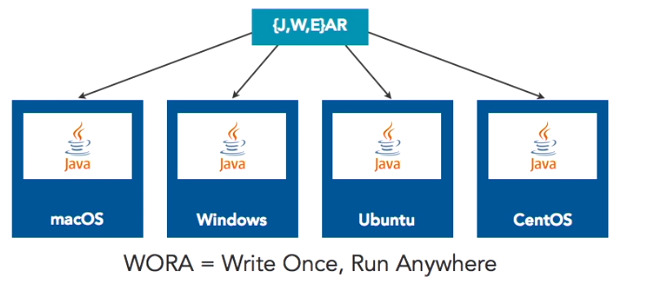
summarize, **Docker is an application delivery technology. It gives you the ability to build an application using Docker image, ship an application using Docker Hub, or registry is another term for shipping, and then finally, be able to run the application as a Docker container, potentially on multiple hosts so that there is no single point of failure and even multiple instances of a container just to avoid a single point of failure as .**

****

**Docker and Java**

WORA, or Write Once, Run Anywhere. How does that work? Well, I mean, you have a Java source file. You take the Java source file. You give it to the Java compiler.The Java compiler then takes it, generates a .class file. Then, you have different tools like Maven, Gradle, so on, so forth. You take those class files and you pack them into a JAR or a WAR or an EAR file.

Okay, cool. So, you've taken the Java source file, compiled into class, and then packaged it into a JAR, WAR, or EAR.

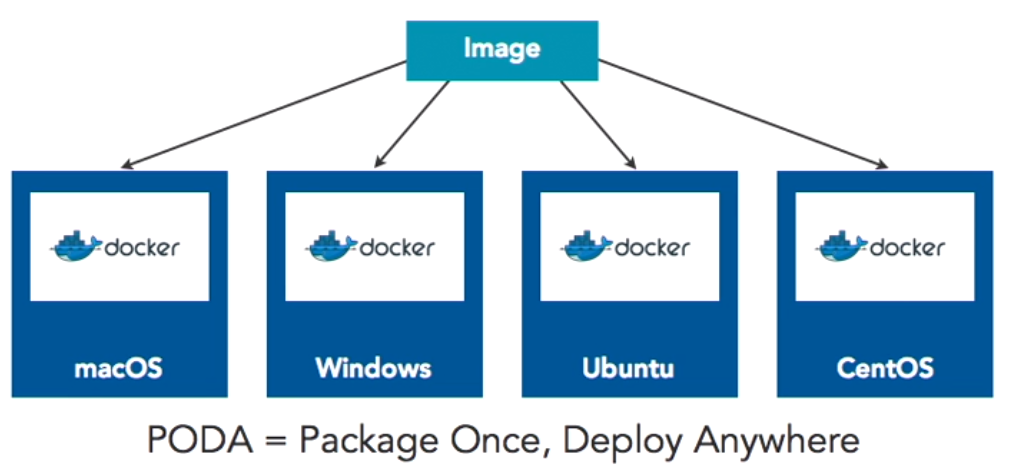


Now, on a variety of operating systems, whether it's Mac or Windows or Ubuntu or any flavor of Linux for that sake, there is a JVM, or Java Virtual Machine that is running. That JVM now understands the format of this class file, which is a Java bytecode. And, that's what gives you the power of Write your source code once, and the JVM will understand the underlying bytecode and it will run out of the box.

Now, Docker is very similar, but not exactly same.

And, I'll highlight the differences in a second, but let's understand what the similarity is first. As we talked about, Docker has this capability of creating an image. Images where all of your application configuration data is put together. Now, you have created that image once on a variety of operating systems. Once again, OS X, Windows, a variety of Linux. You have this thing called as running as Docker engine, or a Docker host.

Now, the Docker engine understands the image format. You have given me the image format. Just like your JVM understands the class format, the Docker understands the image format. It takes that image format and runs the container for you. Now, Docker is essentially a native Linux technology. It's built on concepts like cgroups, namespaces, et cetera. Those inherently exist in Linux, so on OS X and Windows, you do have to run sort of a virtual machine which we'll talk about a little bit later in the chapter.

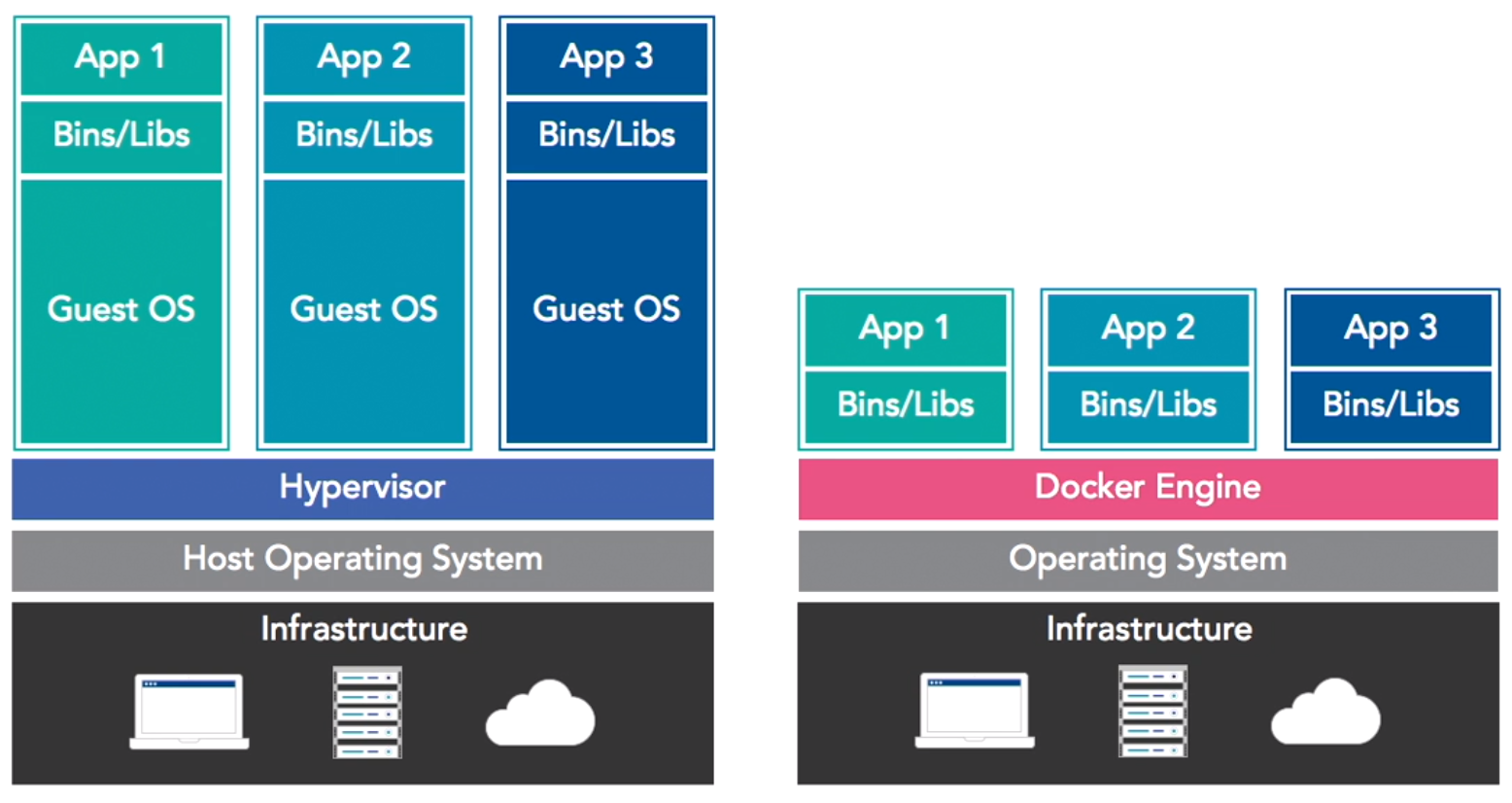


But, the point is, once you have the image ready, you can easily run it on a variety of operating systems. We'll talk a little bit later on how we create an image. So, as I said, **PODA which is Package Once, Deploy Anywhere**, **is very similar to WORA, but it's not exactly similar because based upon what is the base operating system that you're using for an image, it could run only on that operating system.** And, we'll talk about this differentiation and will understand this better when we go later in the chapter of creating an image essentially.

So, just the way you do WORA in Java, the same concept is of PODA in Doc.

**how is Docker really difference from virtual machines??** A lot of people call Docker as a virtualization technology. I would like to be a myth-buster for that. It's not a virtualization technology. Docker is an application delivery technology. Well, let's try to understand what that means. On the left, what you see is a very classical virtual machine environment.So, you have your infrastructure on-premise, cloud, laptop, whatever it is. You have a host operating system running, which is whatever your base operating system is. On a host operating system, you would run a hypervisor. Let's say, a type two hypervisor like Virtual Box or anything. Let's say my host operating system is Windows or Mac or Linux, and on top of that,I went on a different operating system. For example, let's say my host operating system is OS X and, on top of that, I want to run a Windows operating system. So for that, I would need to run a type two hypervisor and then, using Virtual Box, I can create a Windows VM on a Mac OS X.

Now, if I want to build an application around an application within Windows, then I would have to packet my application as a full-blown operating system sitting over there. It's literally delegating for networking and hard drives. Very basic needs are being filled by the underlying host operating system. So, on a given infrastructure, if you have multiple virtual machines running, there's a limit because each operating system has its own memory requirements, CPU requirements, et cetera. So, you are stalled by the capacity of the underlying operating system. The density is very  low..

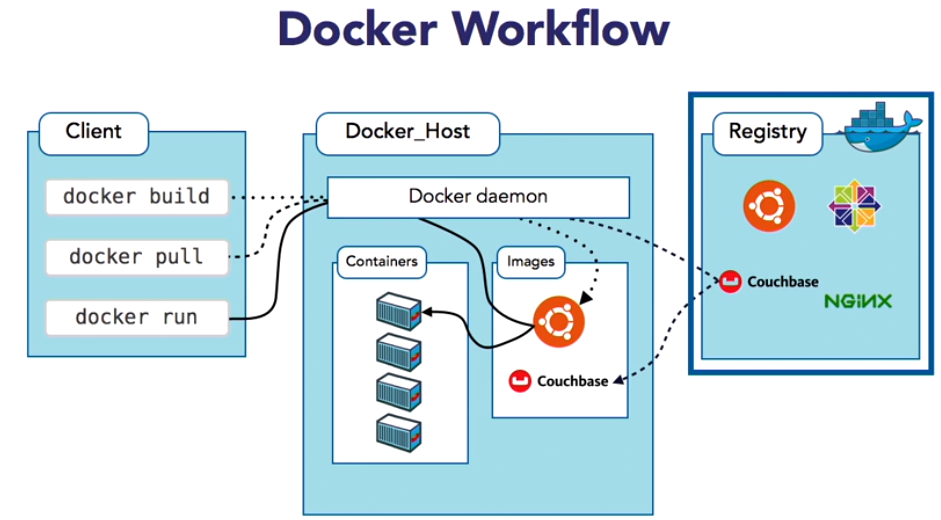


Look on the right-hand side now, which is, you have same underlying infrastructure. This is not a host operating system. It's just a simple operating system on which your Docker engine is running. Now, the Docker engine has a whole bunch of images. In that image, instead of a full-blown operating system, you have a minimal operating system. What does it mean by minimal operating system? As we discussed earlier, Docker relies upon basic Linux capabilities like cgroups and namespaces.

So, instead of having one gigabyte Ubuntu guest operating system on a typical virtual machine,I can use 120 megabytes Ubuntu base operating system on the Docker side and then packet my application in there. Everything else is provided by Docker engine. What that gives you? Well, first of all, that gives you a much smaller image, but because of the smaller image and lesser requirements, you can have much more densely packed.

You can have agility of the application. You can have the scalability of the application achieved much more easily. So, essentially, Docker which is an application delivered technology, is very different from a virtual machine technology. So, I think understanding the concept that how PODA matches with WORA, and how Docker is different from virtual machines, is fundamentally important. Once you understand that, then we can go forward and look a little bit more concepts of Docker.

**Docker workflow**

****

it's got three big blue boxes. On the left what you see is Docker "Client", also called Docker "CLI". In the middle what you see is a Docker "Host" or a Docker "Engine", those terminologies are typically interchangeable, and in the far right is what you call a "Registry".

1.) Now, lets go back to our build, ship and run concept here. **Okay, so the "build" part as we said is an image. "ship" is a registry, and a registry is sitting on the right. And "run" is a container.**

2.) So, as you can see, the images and the containers are sitting on the Docker host over here.

3.) So, first of all Docker client. We'll talk a out how Docker client can be installed, but lets say you have the Docker client on your machine, then the Docker client is configured to talk to a Docker host.

It could be a single docker host, or it could be a cluster of Docker hosts.

In later chapters we'll talk about how we set up a cluster of Docker hosts, but the important part to understand is 7 client is a "dumb client". It's a stateless client. There is no state on the client at all. The client says "I want to run a container", which is a Docker run command. Now, the client is configured to talk to Docker\_Host it gives the command to Docker\_Host, on Docker\_Host there is a Docker daemon running, which is listening for that command. The client gives the command "docker run", but if you have to look under the hood, it is basically a REST API call, which goes as a REST API and Docker daemon is listening on a specific port for that REST request.

It understands that request. So, the client said "docker run", okay? Docker says, "In order to run a container I need to have the image". By default Docker daemon, or Docker\_Host is configured to talk to a registry, which is Docker hub, which is hub.docker.com. So it queries the registry,says, "Hey do you have this image? Because I need to run this container". Lets say you want to run a simple Nginx container or a simple Couchbase container, or an Ubuntu container. So it queries the registry.

Registry has, well first of all, registry is where anybody can publish a Docker image. So typically if it's an N ginx container it is published by the entity behind it. For example for Couchbase,Couchbase has an image on Docker hub. So anybody and everybody can push an image on the public Docker hub. So now, Docker\_Host is saying, "I want to download an image". It downloads the image on the Docker\_Host. Remember we said Client is stateless? The state is maintained on Docker\_Host. You can download as many images on Docker\_Host via the client.

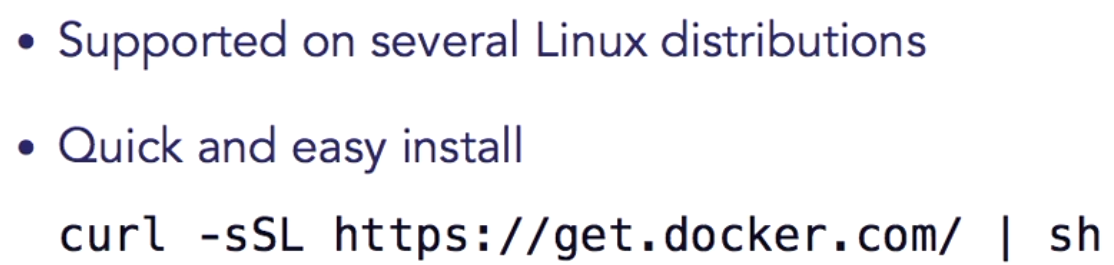
**The client gives a command, the image is stored on the Docker host. Now, once you have the host and you want to run the container. Sure, the image is downloaded once, but you can run as many instances of the container. So lets think about the flow once again. The client makes a request, docker run, a container, goes to Docker\_Host. Docker\_Host says, "I don't have the image. If I have the image I will run the container. If I don't have the image I will download the image from the Docker hub, and once the image is downloaded I will run the container. All this happens behind the scenes.**

You don't need to worry about it, but sometimes understand if we look under the hood on what's happening and what's going on. Now there are lots of varieties and flavors that we can do to this. As we said, the first flavor is **"I don't want to have a single Docker host, because that's a single point of failure for me".** So I can have a cluster of Docker hosts, and later in this course we will look at "**How can we have a cluster of Docker hosts?"** Now, the second flavor is where you can say, "I don't want to have**, I'm not creating images that should not be available on the public Docker hub. I want to have an internal registry".**

So, Docker gives you this commercial product called "Docker Trusted Registry" or there is an open source project that you can deploy internally, and similarly there are registries by other companies like JFrog, which you can deploy inside your firewall. So, that's another variation that's possible. In your typical day-to-day workflow that is your typical model, where the client makes a request to the host, which downloads an image from the registry, and runs the container.

**Get started with Docker**

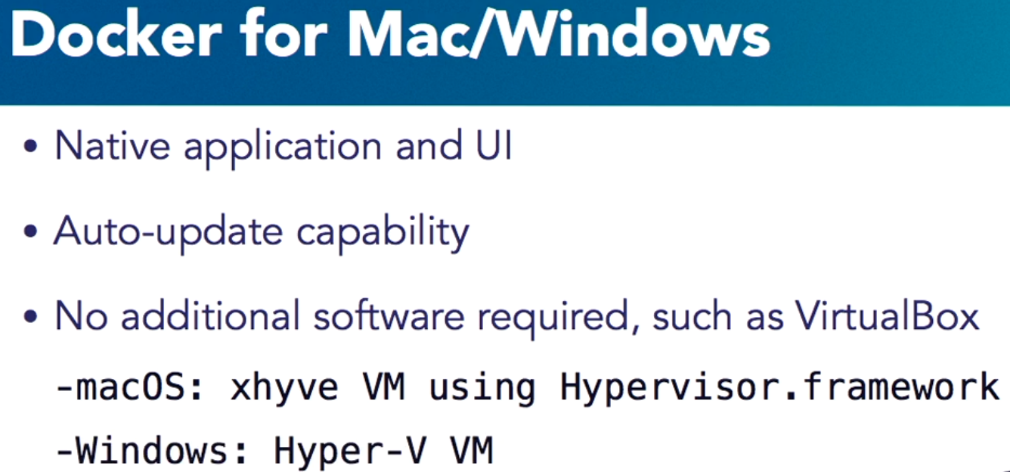
run the script. It'll download the the latest Docker version for you. It'll install the Docker engine.



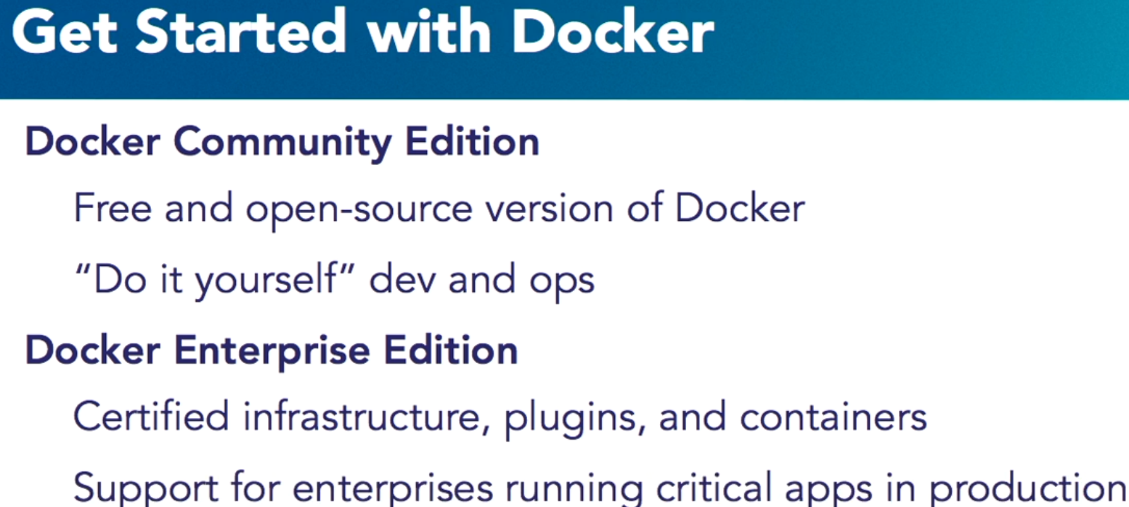
Prod: add the repository. You want to make sure you download from the right repository and stuff like that.

Docker for Mac and Windows. It’s a native application and a user interface. What that means is you go to a URL, you download the Docker for Mac or Windows; that provides a very native, seamlessly integrated experience on that particular operating system.

It's a very Mac-like experience, very Windows-like experience; and we'll see that a little bit later in this chapter. The UI or the tool is automatically updated, so let's say you have a particular version of Docker and a new version is released in the back end, you know, by Docker, then the tool will automatically update. So you don't have to worry about it. You can configure it to be auto update, or you can manually update it. The good thing about Docker for Mac and Windows is there are no additional software required like VirtualBox. We will look at Docker toolbox,which will require an additional software.



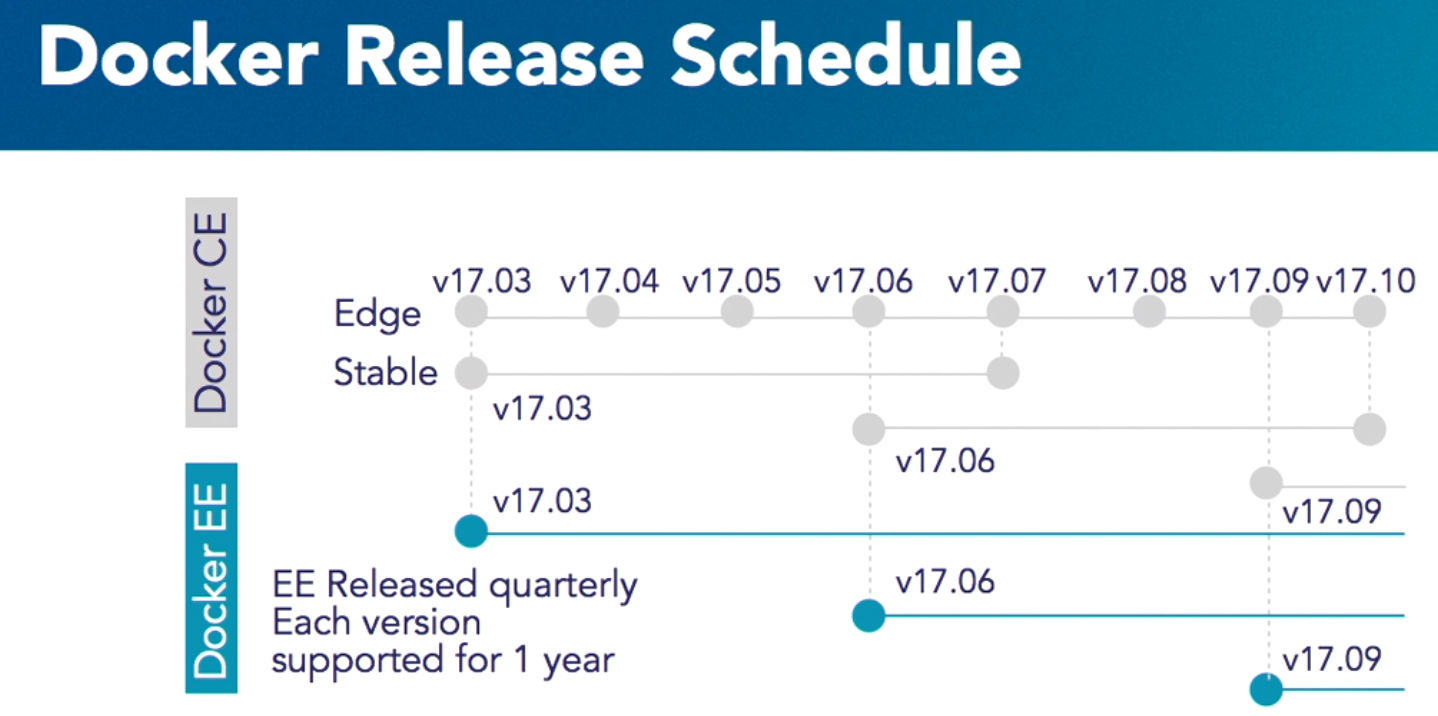
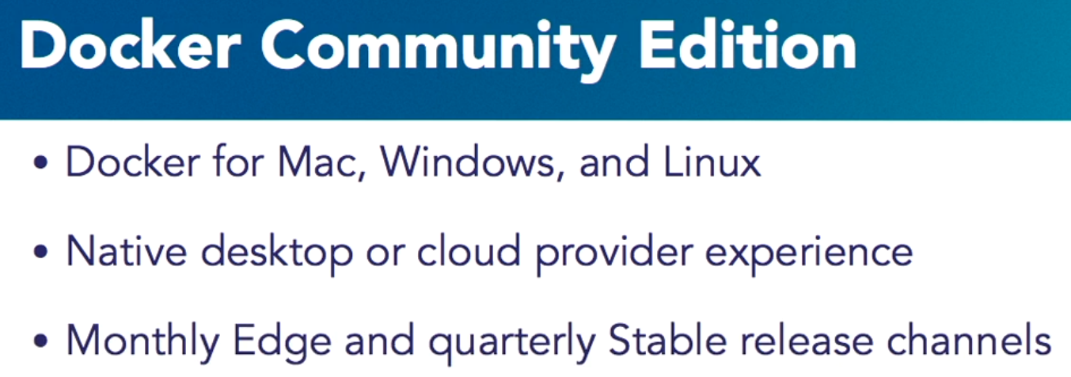
That's how everybody is starting these days. They just download Docker for Mac or Windows, and they get started with that. And how does it work? Well, because Docker is a native Linux technology. Well, on Mac, particularly starting Yosemite, and they have hypervisor.framework.So on OS X essentially, it creates a xhyve VM; and on Windows, it uses Hyper-V VM. Those are both hypervisor frameworks built into the operating system, and that's why you don't need any additional software for installation of Docker.



Docker comes in two editions: Docker Community Edition and the Docker Enterprise Edition.Docker Community Edition is ideal for developers and small teams that are looking to get started with Docker and experimenting with container-based apps. It's a free and open-source version of Docker, as it's more popularly known; and it meant for teams that are do-it-yourselffor developer and ops. Docker Enterprise Edition on the other side is designed for enterprise development and IT teams who build, ship, and run business critical applications and productions are scaled.

It gives you certified infrastructure, plug-ins, and containers; and there is commercial support available for enterprises running critical applications in production for you.

 Let's take a look at Docker Community Edition, since that's the one that we going to be using for this course. Well, the Docker Community Edition is available for Mac, Windows, and Linux; the three primary platforms that we would care about. It gives you a native desktop experience. That means you can run it on your local desktop or in the cloud, where there is AWS or Azure.



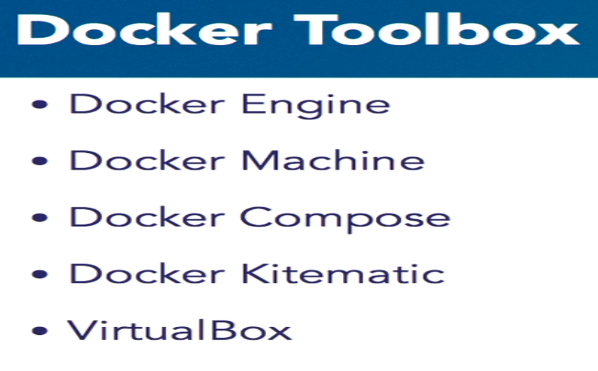
It provides all those experiences very seamlessly. In terms of the release cycle, there is a monthly Edge release cycle and a quarterly Stable release cycle. So every month, there is a bleeding edge built that is pushed out by Docker. So if you're using Docker for Mac, for example,then you will have the Docker for Mac native application installed on your machine; and every month Docker will push that bill to you. Same thing for Windows and Linux as well. Or if you want a Stable bill, where you don't want to be in the bleeding edge, then you can install the Stable download; and Docker will push you updates typically about every quarter.

Docker Enterprise Edition, on the other side, is doing releases every quarter; and each release is supported for a year. Now the actual release cycle, the support lifecycle, that may change; so I would recommend going back to Docker.com for the latest updates on that. Just to give a better idea, in a given calendar year, say January through December, you know, you will have a CE Edge release every month. Every third release in the year, starting from March 2017 essentially, will be Docker CE Stable; and then there will be a corresponding Docker EE as well.

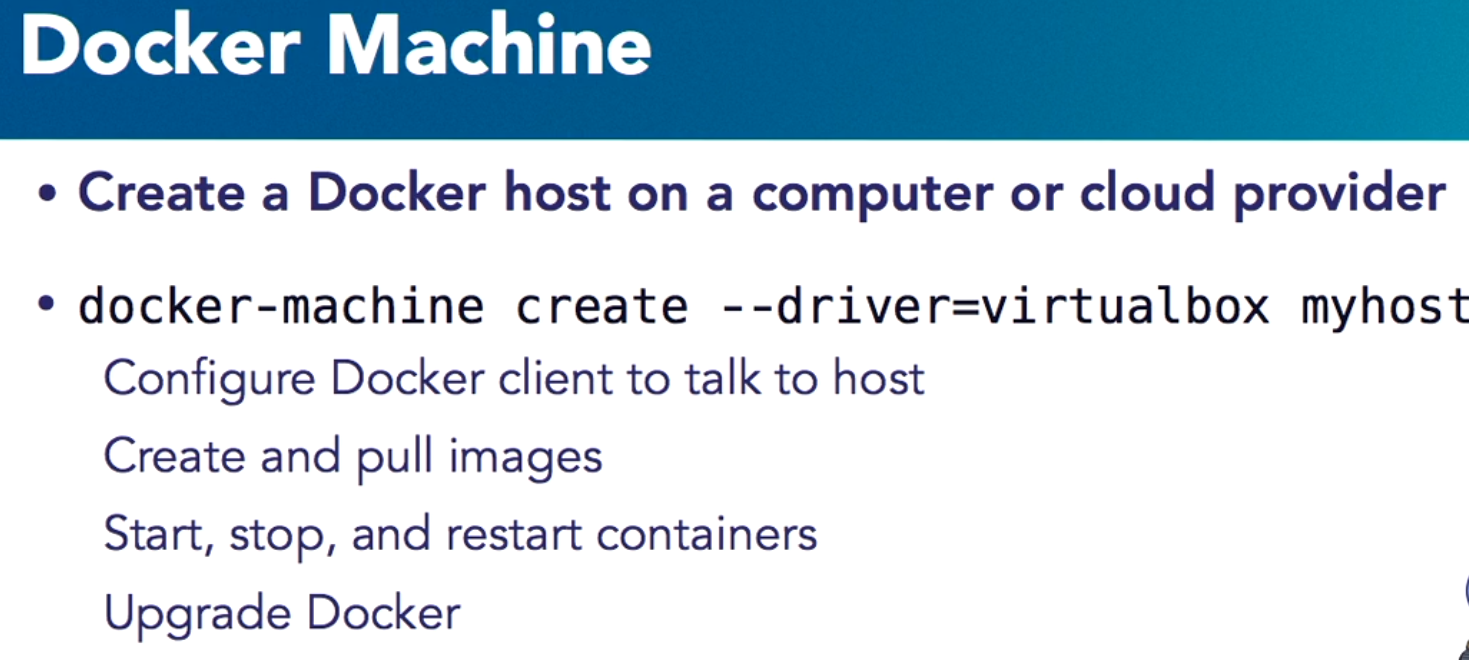
**Now the CE Stable will be supported for about four months; and then, in the meanwhile, the next CE Stable to release would be available to you as well. Docker EE, as I said earlier, is currently supported for year. Now, I'm on Docker.com website.**

In order to get started, you click on view pricing. You can either get started for free, there's a 30-day evaluation version; or you can buy online in Docker store. For this video course, we will only use the Docker Community Edition; and that version is available for free on the Docker store.Now, as I said, Docker for Mac and Windows have little clear requirements; and if you are somehow not able to meet those requirements, then you can download Docker toolbox. What is Docker toolbox? Docker toolbox is a loosely composed set of tools put together.

Essentially it has Docker engine; it has Docker machine. Well, Docker machine is a CLI that is included as part of Docker toolbox that creates a VirtualBox VM for you or that allows you to simulate Docker engine environment on your Mac or your Windows. It could use VirtualBox as a virtualization provider, and there are other virtualization providers that are available as well, which we'll talk in a second. It also has Docker Compose, which we'll cover in a later chapter,but allows you to essentially run multi container applications very easily.

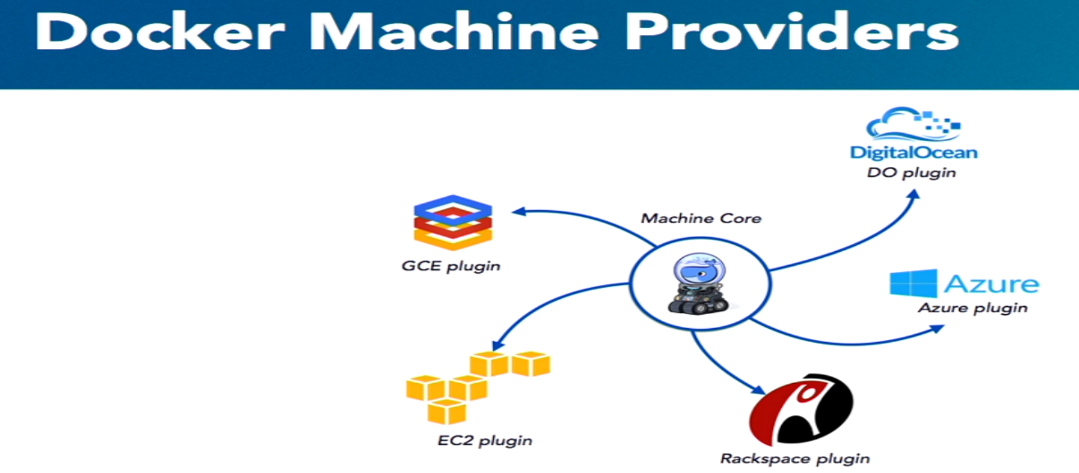


Docker Kitematic is a simple UI that allows you to manage your VMs and VirtualBox is included for the simplicity of it, so that you have one seamless experience over there. And then there is a Quickstart Terminal that is already available for you, which is pre-configured to talk to that Docker machine. Now we talked about Docker machine. Let's take a look at how do we create this Docker Machine. Well, the command line is included for you, which is docker-machine; and you say docker-machine create, and then you need to specify a driver. So you can say the driver is VirtualBox, for example.



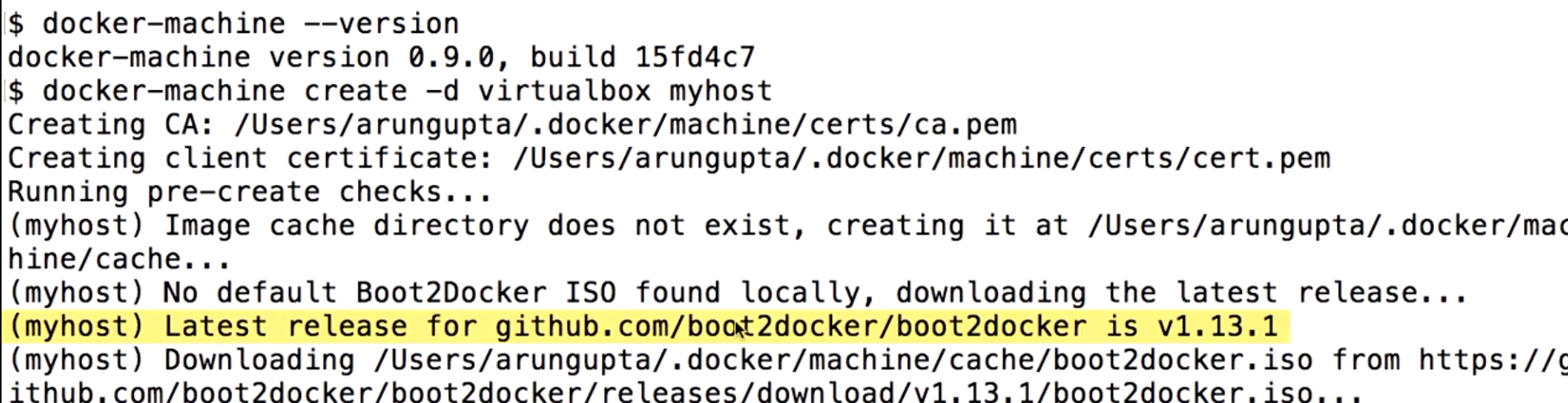
In this case, the Docker Machine will create a VirtualBox VM for you. It'll install all the Docker components as part of it, and the machine name is specified at that last part of the CLI. In this case, it's myhost. Then, you can configure your Docker client to talk to this host. You can create and pull images. Now in this case what's happening is your client and your Docker host or the Docker engine are sitting on the same machine, which is pretty good actually for local development.

You can start, stop, restart containers; and it gives you an ability to test everything on your local laptop without thinking about cloud or any other environment. You can also upgrade Docker because let's say you create a Docker machine using a certain version and now Docker has released a new version that is the capability by which the Docker machine can be upgraded or the docker daemon version can be upgraded as well. One of the things that I love about Docker machine is how not only it allows you to create Docker engine using VirtualBox provider, but look at the variety of providers that it gives you.



So you can use Docker machine, for example, to say, okay, create a Docker machine using AWS provider or EC2 provider or a GCE provider or an Azure provider. So, essentially you can create Docker machine using VirtualBox on your local box; that is cool. And now you're ready to test on a cloud. You just change the provider, and now it's going to get provisioned on EC2 or Google Cloud or Rackspace. So that capability and that functionality is very, very handy. So, let's take a look at it on how we can easily get started with Docker machine.

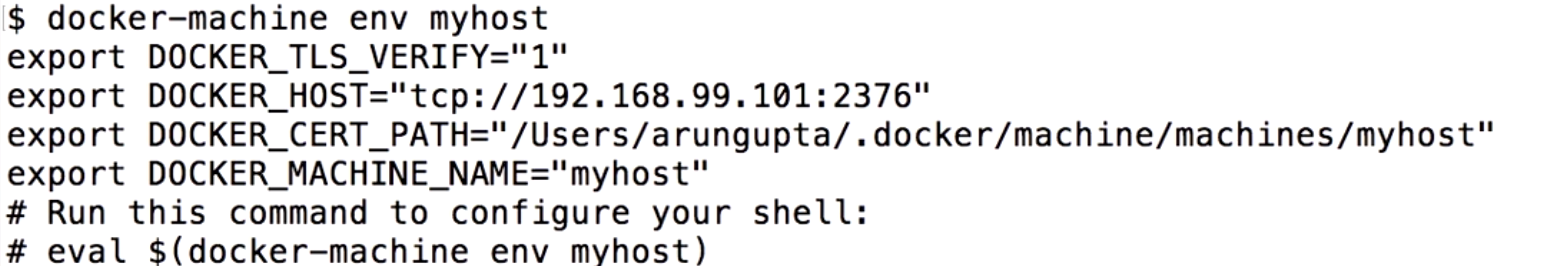
-🡪Now

I press an enter.Now here, as you can see, it's downloading github.com/boot2docker version.

Once the VM is downloaded-- well, boot2docker.iso essentially is the ISO, which is a tiny Linux VM that is actually used to create the VirtualBox VM. So ISO is the basic image. It uses VirtualBox and ISO combination to create the VM for you. Now this will take some time to create and provide an IP, so we'll just move forward. All right, now our Docker machine is created. You can see the commands are prepended by myhost, and now the operating system is identified, you know, it uses boot2docker to provision the machine; and now everything is running.

-🡪Now, 

Docker; there is nothing over there, okay? **But how do I configure my client to talk to Docker host? So** what I do is I say docker-machine env myhost,



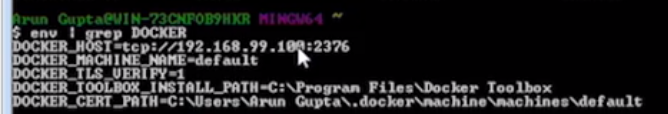
But, essentially, my Docker CLI is looking at all of these different environment variables to configure on where the Docker host is living. So what I'm going to do here is I am going to literally copy this command here, and I'm going to paste it here. All it's saying is, okay, take those environment variables and set it as part of my environment. And I eval it, and now when I do a env and grep on Docker, those variables are set for me. So now when I say Docker image ls,and we'll talk about these commands in the next chapter; but, basically**, this command show show can you show the listing of all the images on the Docker host, which in our case is Docker machine.**

### Boom, it shows no images exist; and that is true because we haven't created any images. We haven't downloaded any images so far, but at least it shows how can you configure your Docker CLI to talk to the Docker machine very easily.

### Docker Toolbox for Windows demo

start our Quickstart Terminal. One of the first things it does is it tries to download the latest boot2docker.iso from GitHub to make sure it has the latest VM for you. Then it tries to create a VirtualBox VM, creates the SSH key, which will allow you to SSH into the VM itself. Certain times, you may get an error as shown on the screen over here, so the easiest way to go past this error is click on any key to continue.

Let’s say we start (mumbles) container here, the container will not be accessible to us on localhost. Instead, wherever localhost is used in rest of the video, if you're using Docker Machine, then you will have to use this particular IP address. In our case, it's 192.168.99.100. If it differs in your case, so make sure you use that IP address.

And sometimes you may wonder, what is the IP address that is assigned to me? So, in that case, you can look at the Docker environment variables, and, here, it gives you the Docker host, and this is exact IP address that is to your machine. So make sure, any time you see localhost, if you're not using Docker for Mac or Docker for Windows, and you're using Docker Machine, use this IP address. In this video we saw how to install Docker Toolbox on a Windows virtual machine. Make sure to use VMware Fusion instead of VirtualBox to have success.

### Docker CLI

### Docker info

Okay. Well, let's see what it shows us here. When I see Docker info, it says 0 containers are there,no running, no pause, no stop, rightly so, because we haven't done anything. It talks about, no images are there. It talks about certain storage drivers. It talks about plugins and we'll talkabout them a little bit later. It talks about swarm. Now, we haven't talked about swarm yet but essentially it gives you multi-node Docker cluster. We'll talk about that a little bit later. It talks about the run times, and out runC's before run time baked into Docker itself.

It talks about other options. It talks about certain kernel version, operating system. It talked about the OS type, the architecture,

**Docker version**

Now we saw the Docker info here, but I want to see what client version and the server version I'm using. So for that, I'm going to say, Docker version. Now, here they're saying, the client is using version 1.13.1. As we talked about in the Docker work flow, the client makes a request to the server, but underlying it's a REST API.

So it even gives a REST API version here. And it gives you some other detail about client. And similarly, it gives you the server version, and the server API version as well.

DOCKER\_HIDE\_LEGACY\_COMMANDS=true

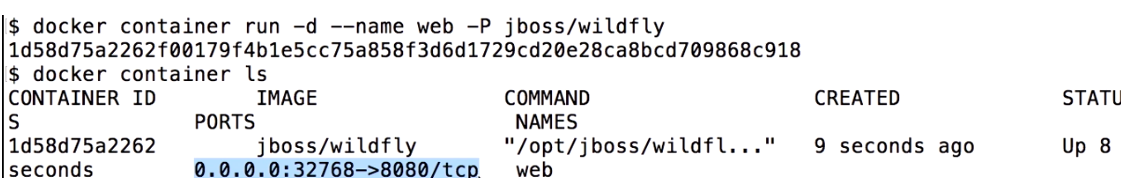
**Run your first Docker container**

----USE –it

**Run container (ports and volumes)**

Running the container has been no good to us, because we have not been able to access it yet. Well for that, you need to be able to expose ports. And attach volumes to it so that it can deploy .war files to it.

WildFly we know by default runs on 8080. By saying -P you pick a random port in a certain range and expose me, basically make myself available.

take all the network interfaces on the host, which is indicated by 0.0.0, port 32768 over there and redirect itto port 8080 in the container.

$ docker-machine.exe ip

192.168.99.100

$ docker-machine.exe ls

NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS

default \* virtualbox Running tcp://192.168.99.100:2376 v17.10.0-ce

test1 - virtualbox Error Unknown machine does not exist

**docker container run -d --name web2 -p 8080:8080 jboss/wildfly**

**docker logs db74a50e**

Instead of capital P, which says take all exposed ports and publish them in a predefined range, I'm going to use a lowercase p. And I will say take port 8080 on the host and map it to port 8080 on the container.

**HOW TO DEPLOY WAR FILE**

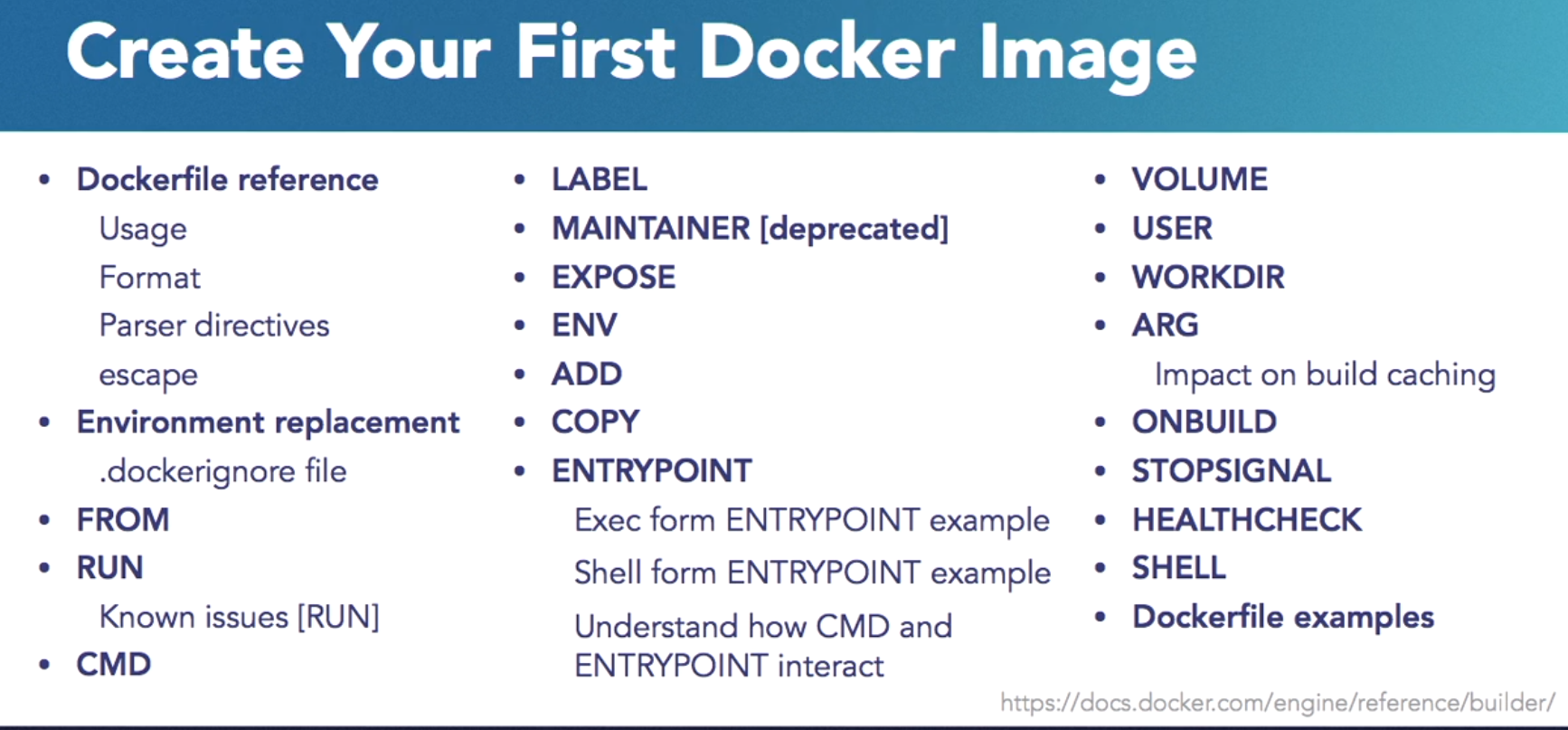
I'm going to add a volume mapping. What that means is, I can take a directory on my local machine, and my laptop in this case, and mount it as a point in my container. And let's see how we're going to use that effectively to deploy the .war file. So I'm going to say -v, which is how I'm going to do the volume mapping.

I'm going to take my current directory(PWD) , pick up webapp.war from there, and mount it to the WildFly container has a particular directory in which the .war files need to be deployed. So I'm going to specify that directory location here.



docker container run, running in a detached mode, giving it the name web, exposing on port 8080, doing the volume mapping, taking the webapp.war file, deploying it to /opt/jboss/wildfly/standalone/deployments/ .war file, and then the container name.

### Create your first Docker image

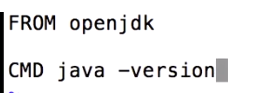


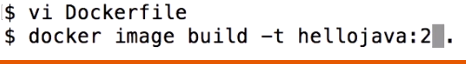
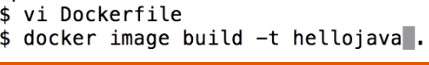
CMD is basically the command that the docker container is going to run.

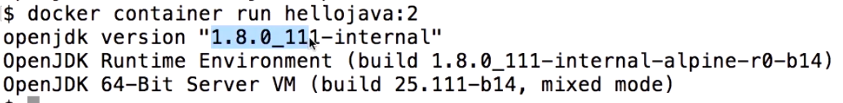
Now, as we said earlier, docker containers can run only one, and one thing, and they do that really well. So, this is my CMD, which is basically my entry point to the docker, and here I'm saying echo hello world. That's it. That is a very simple hello world docker image.

**Docker image –help**

****



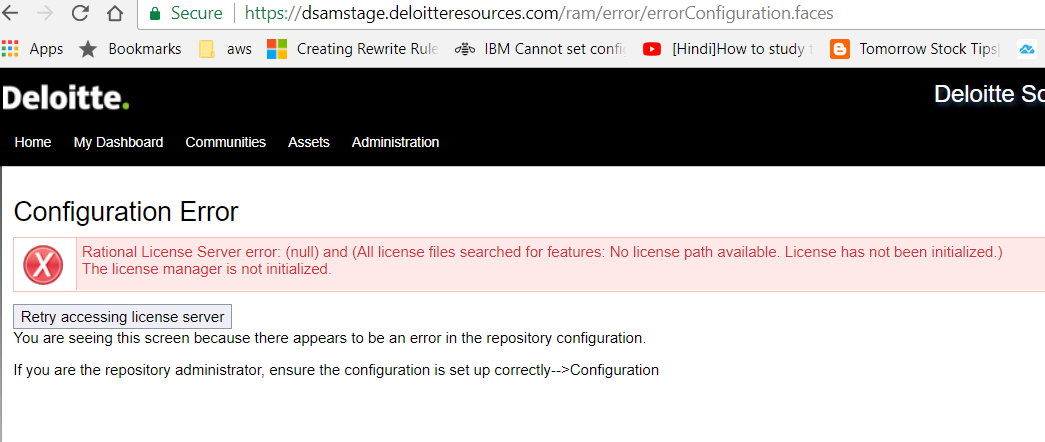
-🡪 2 is tag



**Copy files in the Docker image**

Copy---be it a JAR file, or a WAR file, or something else,you want to copy it in there.

how is it that we can copy the files to a docker image??

and add, and it's often confused. Which one should I used? So let's take a look. Now, COPY instruction, it copy new files or directories from your local machine to the container file system. So let's say you are using a Maven project. From the Maven project, you created an application, and from that application, there is a target directory, there is JAR file, and you want to copy it to the container, you would use a copy instruction. Well, add instruction.

ADD instruction has everything that copy instruction has, but in addition, it also allows tar file auto-extraction in the image. Let's say you have a tar file that you want to get copied to the image and then auto-extract, that's when you would use this. So for example, in this case you can say ADD app.tar.gz, which is on your location machine to opt/var/myapp directory which is in the container file system, so the contents of app.tar.gz will be auto-extracted in that directory.

With add, you can also download a file from a remote URL. Now, that's not typically recommended, because, typically, when you download a file, you want to auto-extract it or do something with it. So the recommendation is, generally, use curl or wget, based upon your operating system or the utilities that you have available. You use one of those tools to download the tar file, and then extract the contents if you need to, and then get rid of the tar file, because, otherwise, it could cause unnecessary bloat in your overall image size.

So that's the important aspect to understand. What we're going to do is we're going to take a look at the copy instruction, and show how that can be used to copy a WAR file into our image, and then run that image. So let's go to our terminal. Just like any docker image, I'll make a new directory here by the name helloweb. Okay, now I'm in the helloweb directory. I'm going to go into helloweb, and in my chapter2, which is my exercise files, I'm going to copy the webapp.war file in this directory. And remember, I am in helloweb.

So now, I'm going to create a Dockerfile here. In this Dockerfile the WAR file that we have is a Java EE application, so we would like to deploy this into the WildFly container. So I can just directly say FROM jboss/wildfly. Okay. And in order to copy the WAR file, from my local file system, because webapp.war file is in my current directory, and this result from the build context that we talked about earlier, I'm going to copy it into my container file system, which isopt/jboss/wildfly/standalone/deployments/webapp.war.

That's it. Now you may wonder, there is no CMD command. How does this work? Well, in jboss/wildfly image, there is a CMD command, which by default fires up WildFly. Let's take a look at that. So let's go to a browser for a second. So I'm going to go to hub.docker.com and I will search for jboss/wildfly here. Here is my image.

And if I look at the corresponding Dockerfile here, if you go all the way to the bottom, here it says it's starting WildFly and binding to all network interfaces. Now, we talked about how image hierarchy is used. The other part that you also need to understand is in multiple Dockerfiles that have been used to create your image, only one CMD command is effective. So if you write a CMD in your Dockerfile, then the CMD in the parent Dockerfile would be overridden. In your Dockerfile, we could have multiple CMD, only the last CMD would be effective.

But in our case, we only care about the default behavior of WildFly starting up, so we are okay.We don't need to override the CMD command. We just used the copy instruction, copy the WAR file into the WildFly standard directory, and then WildFly will do the right thing, so this is the default behavior. So let's go back to our terminal, and build our image. So, back in the terminal here. Let's save the image here. We only have Dockerfile and webapp.war. We're going to say docker image build -t helloweb, and I'm going to go with the build context as the current directory.

Because my JBoss WildFly image is already there, so that layer is already available for me. I just have to copy the WAR file, and that's it. Now, let's take a look at image listing here. It is my helloweb, and this is slightly bigger than the standard WildFly image, but what I can also do is I can say docker image history helloweb. It's showing you lots of details about the WildFly image here, but this is the copy command that we gave, which is showing as the last image in the helloweb directory.

Now, in order to run this container, docker container run, and you guess it right, we're just going to say helloweb, ad we need to expose ports here. So, host port, container port, and run it in detach mode. It's just a simple command, and we can fire up our container here now. Certain times you may get an error that the port is already allocated. Now, that error occurs typically if you have a container that is already running, and using that port.

Well, that can be easily fixed. So, what you need to do is remove the container that is using the port 8080. You can look at the listing of the containers by using docker container ls command,and then use the docker container rm command to remove that specific container. Or for convenience, remove all the containers just to have a clean environment. And once again, that can be done using docker container rm, and listing all the container IDs, space separated. Now, if we look at the list of containers, so docker container ls, that is the only container running here.

Well, if you don't give it a name, then it gives you a nice fancy name. In this case, it's romantic\_poincare. So the key part is now I can access my application on application name was webapp, rest resources and persons. Tada! So, we ran this container earlier by simply copying the WAR file, but we build the image and we run the container ourselves as well. So, you're getting familiar with the entire workflow of docker, building image, running containers,and hopefully getting more and more familiar with it.