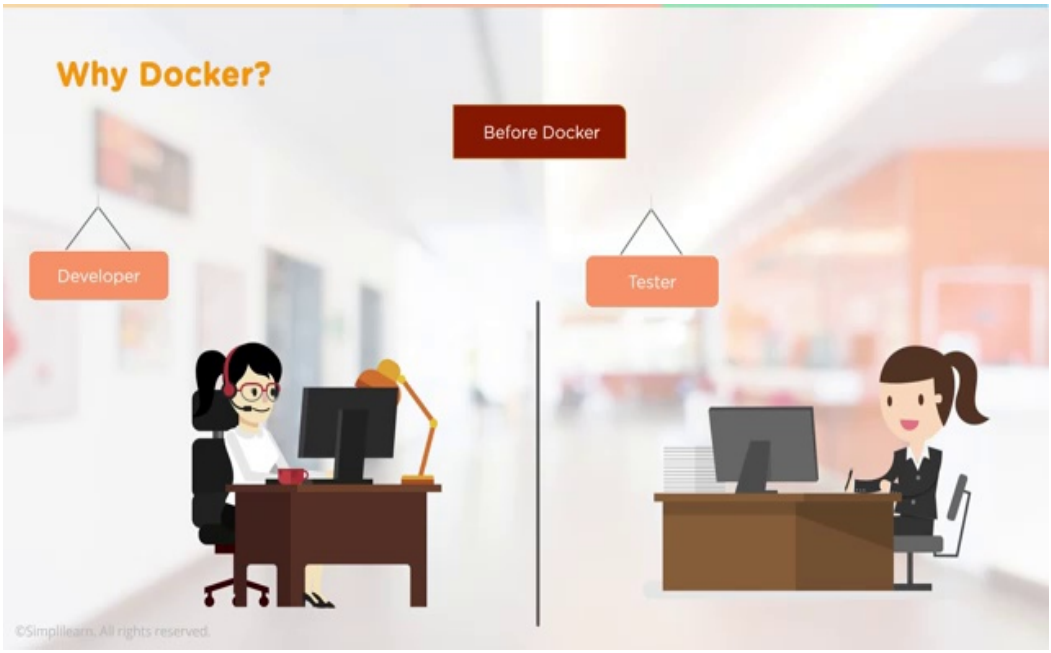


Video Title:

What Is Docker? | What Is Docker And How It Works? | Docker Tutorial For Beginners | Simplilearn

Video Duration:

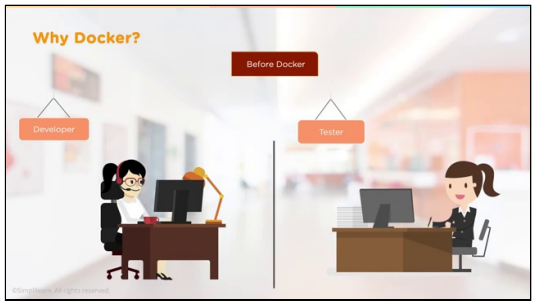
00:15:52

of Extracted Keyframes:

31

Date of Extraction:

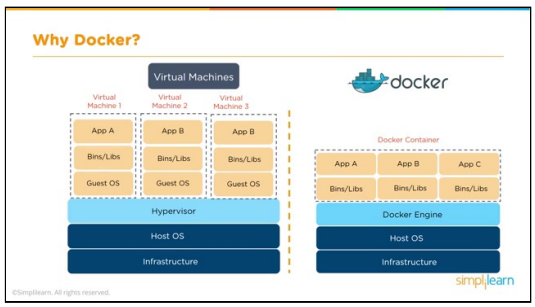
2021/04/16



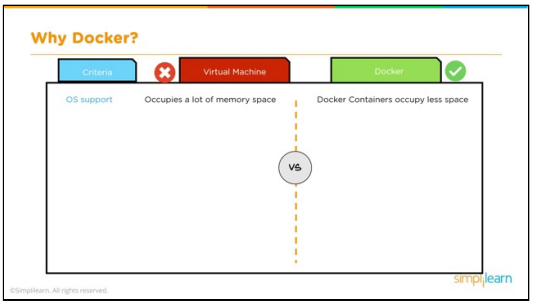
hello this is Matthew from simply learn and today we're gonna cover what is docker and why it should be of value to you as somebody who works in DevOps so

let's take a throw scenario a very developer Etna testa before you had the world of docker a developer would actually build their code and then they'd send it to the tester but then

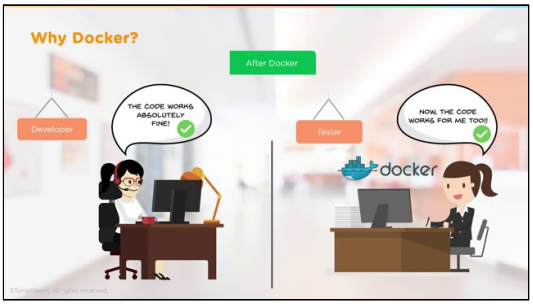
the code wouldn't work on their system coders are worldly a system due to the differences in computer environments so what could be the solution to this well you could go ahead and create a virtual



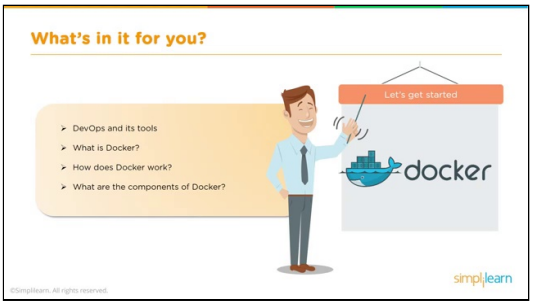
machine to be the same of the solution



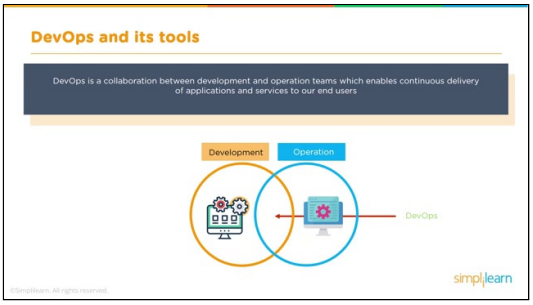
in both areas what do you think docker is an even better solution so let's kind of break out what the main big differences are between docker and virtual machines as you can see between the left and the right hand side both look to be very similar what you'll see however is that on the docker side what you'll see as a big difference is that the guest OS for each container has been



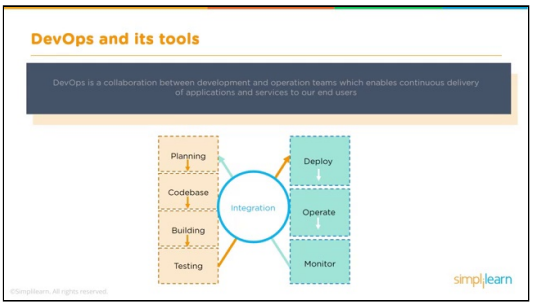
eliminated docker is inherently more lightweight but provides the same functionality as a virtual machine so let's step through some of the pros and cons of a virtual machine versus docker so first of all a virtual machine occupies a lot more memory space on the host machine in contrast Dhaka occupies significantly less memory space the boot up time between both is very different docker just boots up faster the performance of the docker environment is actually better and more consistent than the virtual machine docker is also very easy to set up and very easy to scale the efficiencies therefore a much higher with a docker environment versus a virtual machine environment and you'll find it is easier to port docker across



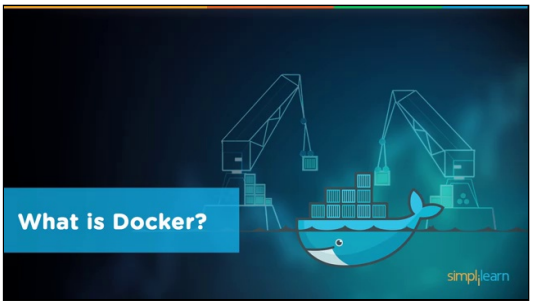
multiple platforms than a virtual machine finally the space allocation between docker and a virtual machine is



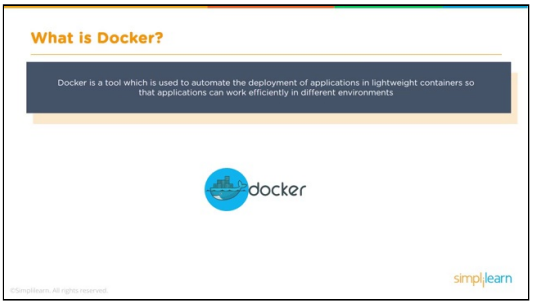
significant when you don't have to include the guest Oh you're eliminating a significant amount of space and the dock environment is just inherently smaller so after darker as a developer you can build out your



solution and send it to a tester and as long as we're all running in the doctor environment everything will work just great so let's step through what can I cover in this presentation we're gonna look at the DevOps tools and where docker fits within that space we'll examine what docker actually is and how docker works and then finally we'll step



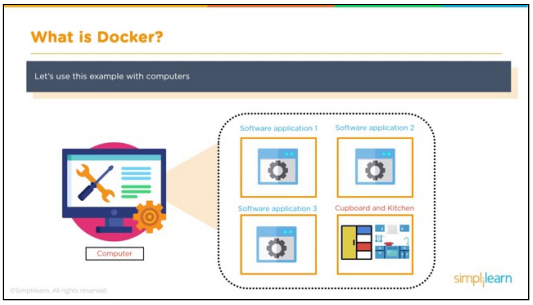
through the different components of the docker environment so what is DevOps DevOps is a collaboration between the development team the operation team allowing you to continuously deliver solutions and applications and services that both delight and improve the efficiency of your customers if you look at the Venn diagram that we have here on the left hand side we have development on the right hand side we have operation and then there's a cross over in the middle and that's where the DevOps team sits if we look at the areas of integration between both groups developers are really interested in planning code building and testing and



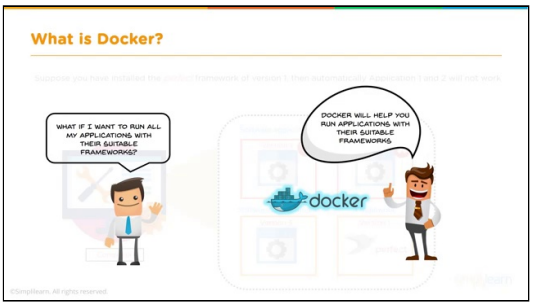
operations want to be able to efficiently deploy operate a monitor

when you can have both groups interacting with each other on these seven key and elements then you can have the efficiencies of an excellent DevOps team so planning in codebase we use tools like JIT and Guerra for building we use Gradle and mavin testing we use selenium the integration between dev and ops is through tools such as Jenkins and then the deployment operation is

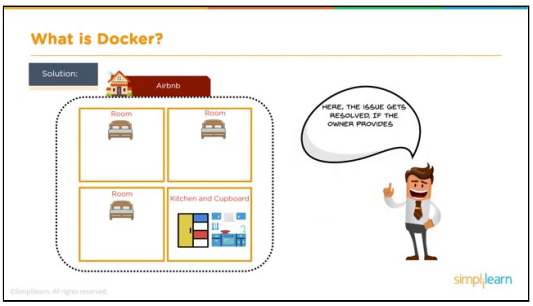
done with tools such as docker and share finally nagas is used to monitor the



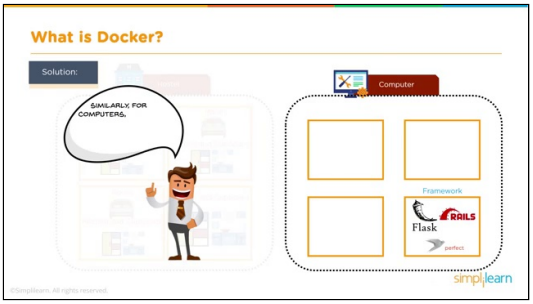
entire environment so let's step deeper into what docker actually is so docker is a tool which is used to automate the deployment applications in a lightweight container so the application can work efficiently in different environments no



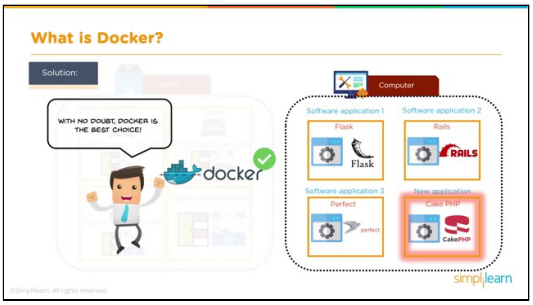
it's important to note that the container is actually a software package that consists of all the dependencies required to run the application so multiple containers can run on the same hardware the containers are maintained in isolated environments



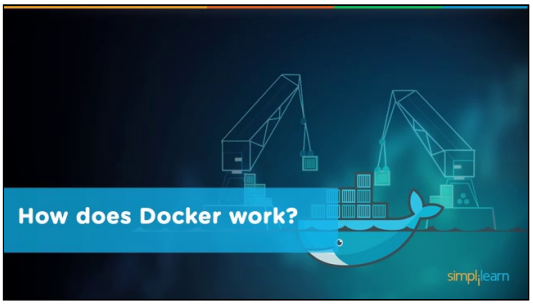
they're highly productive and they're



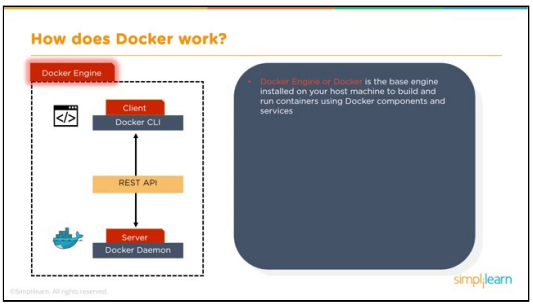
quick and easy to configure so let's take an example of what dogger is by using a house that may be rented for



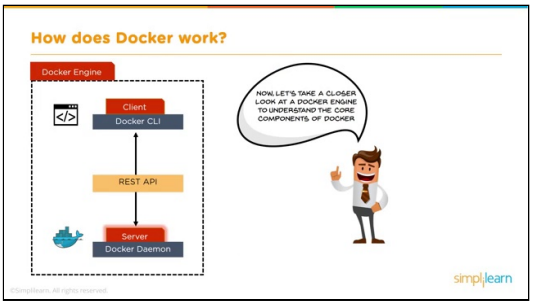
someone using Airbnb so in the house there are three rooms and only one cupboard and kitchen and the problem we have is that none of the guests are really ready to share the cupboard and kitchen because every individual has a different preference when it comes to how the cupboard should be stocked and how the kitchen should be used this is very similar to how we run software applications today each of the applications could end up using different frameworks so you may have a



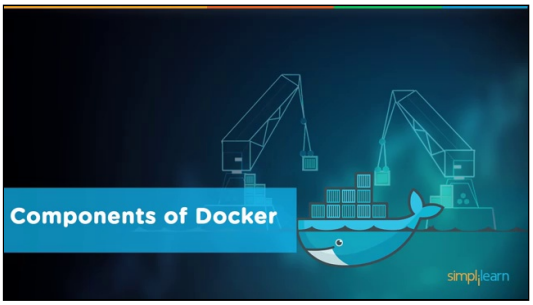
framework such as rails perfect and flask and you may want to have them



running for different applications for different situations this is where



docker will help you run the applications with the suitable frameworks so let's go back to our Airbnb example so we have three rooms and a kitchen and cupboard how do we resolve this issue well we put a kitchen in covered in each room we can do the same thing for computers docker provides the suitable frameworks for each different application and since every application has a framework with a suitable version this space can also then be utilized for putting in Suffern applications that are long and since every application has its own framework and suitable version the area that we had previously stored for a framework can be used for something else now we can create a new application in this instance a fourth application that uses its own resources you know what with these kinds of abilities to be able to free up space on the computer it's no wonder docker is the right choice so let's take a closer look to how docker actually works so when we



look at docker and we call something Dokka we're actually referring to the base engine which actually is installed on the host machine that has all the different components that run your docker environment and if we look at the

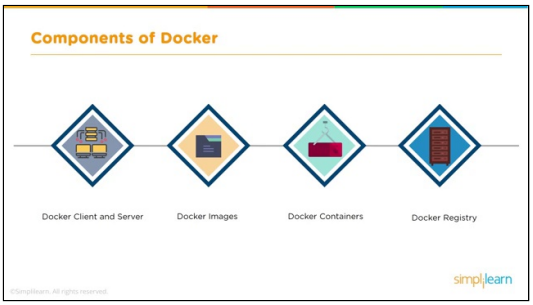
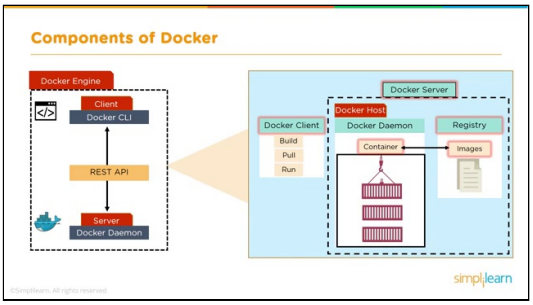
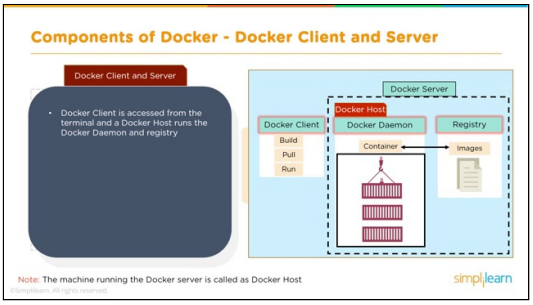


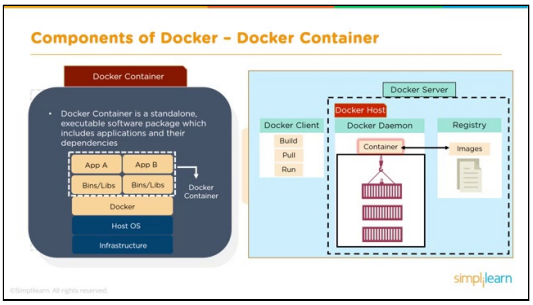
image on the left-hand side of the screen you'll see that docker



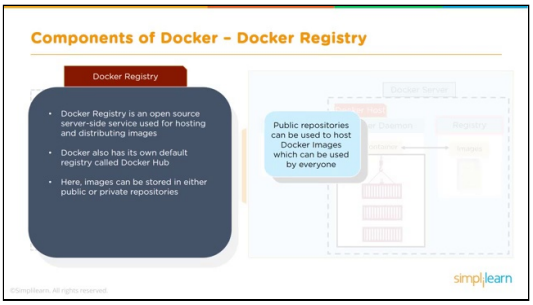
has a client-server relationship there is a client installed on the hardware there is a client that contains the docker product and then there is a



server which controls how that docker client is created the communication that goes back and forth to be able to share the knowledge on that docker client relationship is done through a REST API this is fantastic news because that means that you can actually interface and program that API so we look here in the animation we see that the docker



client is constantly communicating back to the server information about the infrastructure and it's using this REST API as that communication channel the dock a server then we'll check out the requests and the interaction necessary for it to be the docker daemon which runs on the server itself will then check out the interaction and the necessary operating system pieces needed to be able to run the container okay so that's just an overview of the docker engine which is probably where you're going to spend most of your time but there are some other components that form the infrastructure for docker let's dig into those a little bit deeper as well so what we're going to do now is break out the four main components that comprise of the docker environment the four components are as follows the docker clientís server which we've already done a deeper dive on docker images docker containers and the dagger registry so if we look at the structure that we have here on the left-hand side



you see the relationship between the docker client and the darkest server and then we have the rest api in between now if we start digging into that rest api particularly the relationship with the daka daemon on the server we actually have our other elements that form the different components of the docker ecosystem so the docker client is accessed from your terminal window so if you are using Windows this can be PowerShell on Mac it's going to be your terminal window and it allows you to run the docker daemon and the registry service when you have your terminal window open so you can actually use your terminal window to create instructions on how to build and run your images and containers if we look at the images part of our registry here we actually see that the image is really just a template with the instructions used for creating the containers which you use within docker the document image is built using a file called the docker file and then once you've created that docker file you store that image in the docker hub or registry and that allows other people to be able to access the same structure of a docker environment that you've created the syntax of creating the image is fairly simple it's something that you'll be able to get your arms around very quickly and essentially what you're doing is you're creating the option of a new container you're identifying what the image will look like what are the commands that are needed and the arguments for and then those commands and once you've done that you have a definition for what your

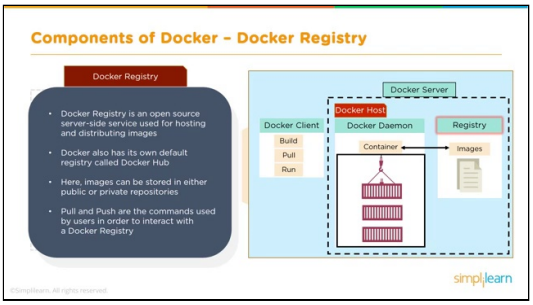
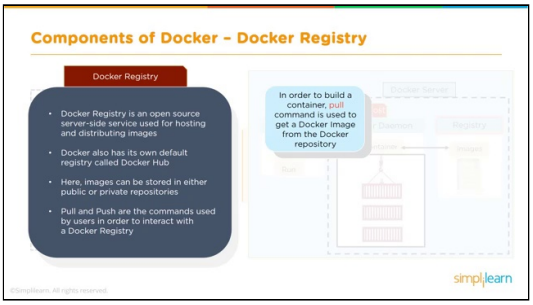
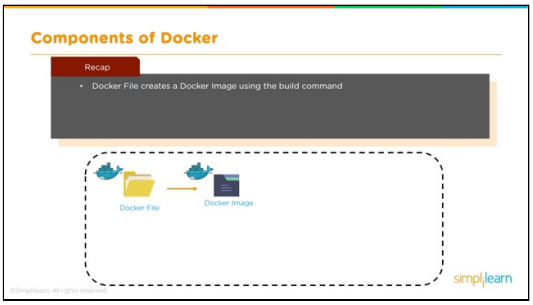


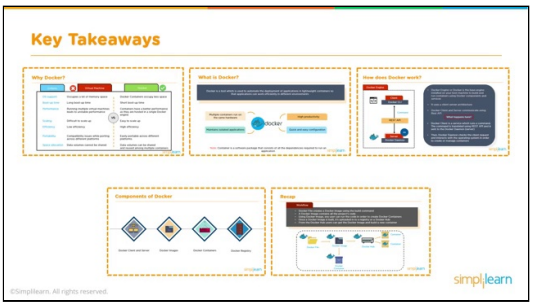
image will look like so if we look here at what the container itself looks like is that the container is a standalone executable package which includes applications and their dependencies it's the instructions for what your environment will look like so you can be consistent in how that environment is shared between multiple developers testing units and other people within your DevOps team now the thing that's



great about working with docker is that it's so lightweight that you can actually run multiple docker containers in the same infrastructure and share the



same operating system this is its strength it allows you to be able to create those multiple environments that you need for multiple projects so you're working on interestingly though within each container that contain it creates an isolated area for the applications to run so while you can run multiple containers in an infrastructure each of those containers are completely isolated they're protected so that you can actually control how your solutions work there now as a team you may start off with one or two developers on your team but when a project starts becoming more important and you start adding in more people to your team you may have 15 people that are offshore you may have 10 people that are local you may have 15 consultants that are working on your project you have a need for each of those two verbs or each person on your team to have access to that docket image and to get access to that image we use a docker registry which is an open source server



site servers for hosting and distributing the images that you have defined you can also use docker itself as its own default Rattray and docker hub now something it has to be very mindful is that for publicly shared images you may want to have your own private images in which case you would do that through your own registry so once again public repositories can be used to host the docket images which can be accessed by