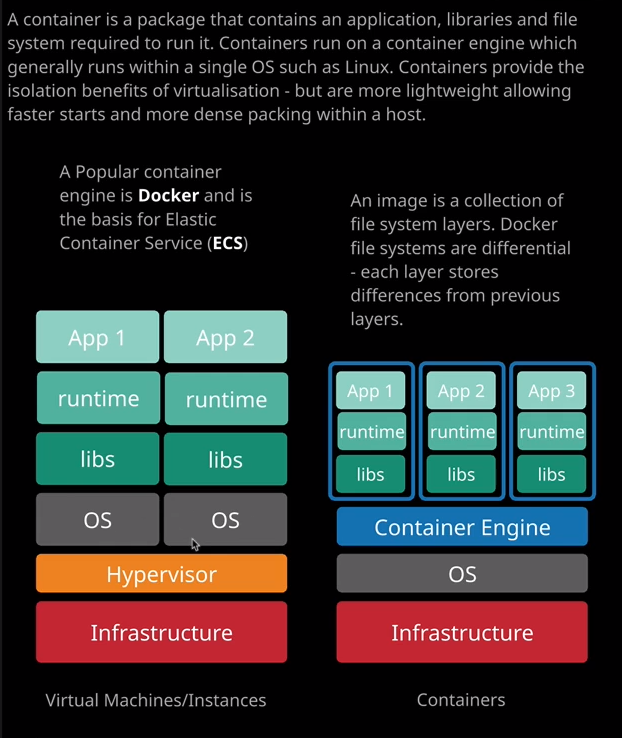
**DOCKER ESSENTIALS**   
  
When you're architecting solutions which need long running compute, you'll generally have two choices: virtual machines or containers. Before I start showing you how containers work, I should establish what containers are. Many people compare containers to virtual machines and while there are some similarities, they are not the same thing.

**Virtual machines, as the name suggests, are a virtual server a virtual grouping of resources. A virtual machine is created by taking physical infrastructure so a physical machine and by using a hypervisor carving that in to one or more virtual machines, each having its own slice of the physical resources of the host. Virtual machines are self-contained. They have their own operating system, libraries, run times, and applications.** So if you run a 100 virtual machines and physical infrastructure, then you'd have 100 operating systems, a 100 instances of libraries and runtimes and 100 or more applications. That is a lot of duplication, but also using virtual machines means that everything is isolated and separate.

Containers, on the other hand, are different. **With containers, the physical infrastructure runs a standard operating system, and on top of this is a container engine. Now the container engine creates a single process in the host operating system for each container that it manages, and this process is isolated. Inside this process are the libraries, runtimes, and applications. It does mean, though, that containers share the same base operating system with the host and other containers just in an isolated way.** Now **containers are quick to start, and you can pack more of them on the same host because they don't also have to run their own individual copies of the operating system**.



Now, an analogy that I've heard before and I'll credit this one to Mike Coleman, who now works for AWS is that virtual machines are like houses. **They're fully self contained. They have their own power, their own plumbing, their own entry and exit controls.** What happens inside your house stays inside your house, and most of the time, if your house burns down or floods, it's just your house that's affected. Lots of houses are the same, and houses can only get so small because you always need that base set of infrastructure, that base set of things inside each dedicated house. Containers on the other hand, they're like apartments in an apartment building. They share certain things. They have a shared entryway, shared power, shared heating, shared plumbing, and this means apartments can be smaller. The smallest apartment could be a lot smaller than the smallest house because they don't need their own dedicated version of these basic infrastructure components and apartments can come in lots of shapes, big and small but if one of the apartments catches fire or floods then it can impact other apartments very easily. So that's a key comparison with containers, **while containers are isolated from other containers because they share the same base operating system, if there are any problems with the containers, those problems can spill across to other containers.** So they do isolate the things that are running inside those containers, just like apartments, isolate their occupants but if somebody in an apartment makes noise then people in other apartments are impacted more than if they were separate houses. So you've got the noisy neighbor problem that's much more prevalent when using containers. **If one container uses a significant amount of resources in a destructive way then it can impact other containers.**

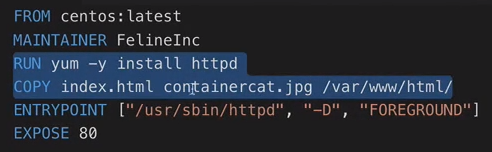
Now choosing between containers and VMs is an important decision. There isn't one size or choice which fits all and you should make sure that you are making the right choice and I hope that throughout this course, I'll give you all of the information that you need to do that. Okay, so it's demo time, and I think it's easier to show you how containers work and what all of the individual components do rather than talk about more theory. So to do that, I'm going to move across to the console and go to EC2 and I'm going to create an EC2 instance. It's going to be a little bit larger than normal, so I'm going to launch instance, going to pick Amazon Linux 2 but this time I'm going to use a T3.medium. So I'll select that, configure instance details. I'll accept those defaults, go to add storage. That also looks good. I'm going to add a tag. I'm going to use the key of name and then the value is go be Docker EC2 manual, because we're going to utilize this **EC2 instance as a Docker host**. So I'm going to go to security groups I'll create a new security group, and I'll call it Docker EC2 manual the same as the instance and use the same for the description. It's already got the SSH rule enabled by default. I'm just going to tighten the security and change this to my IP and then I'm going to add an additional rule, which is to allow HTTP traffic again from my IP address and I'll do that because I'll be creating a container which is a small website. So I'll need to allow HTTP through to demonstrate that. So that looks good both of those are allowed from my IP. So review and launch the instance, select Launch. I'm going to create a new key pair and I'm going to call it Container. I'll download it, save it to my local machine and launch the instance. So I'll move to the instances console I'll just monitor the progress. It will take a couple of minutes to go from pending to available. Okay, so we've got the instance in a running state now. So on the diagram on the right, that's the infrastructure and operating system components. **So the infrastructure is EC2, so it's an EC2 instance. It has some infrastructure. The operating system that's running on this is Amazon Linux 2 and the next step is to connect and install the container engine.**

So I'm going to go ahead and do that. I'm going to right click on the instance and put Connect because this is a new PEM key I'll need to change the permission, so I'll do that first. So copy that into my clipboard, move across to my terminal and paste that in just to set the permissions properly, I'll copy the connection string as well into my clipboard, move back to the terminal and paste that in to connect to this EC2 instance. I'll need to acknowledge the authenticity and as soon as that's done I'll be connected to the shell. So what I want to do at this stage is to install the **Docker engine**, so that's the container engine. It's a specific type of container engine, one of the most popular container engines available and it's called Docker. So I'm going to go ahead and install that. Now to do that,   
sudo amazon-linux-extras install docker //this will go ahead and download and install the Docker container engine. Yes. It'll take a couple of seconds, and once it's finished, it'll be fully installed and ready to go. I'll need //to start the Docker service and to do that, I'll do a sudo service Docker start. Now because I'm logged in as the EC2-user. I'll need //to give that user permissions to interact with the Docker service and we can do that by going sudo usermod -a -G docker ec2-user. Essentially, it's just adding the EC2 user to the Docker group, which will give it the permissions it needs to interact with Docker. I'm going to need to exit and then reconnect to make that work properly.

Connect in to my local machine, reconnect using the PEM key, and then clear the screen to make it easier to see. At this point, I should be able to run a docker ps and get this output so there are no containers running, but it does show that everything's working and ready to go. So now we've got a container engine that's running on this EC2 instance. So now we can create containers, or we can download and use containers. Running Docker on top of an EC2 instance, is a perfectly valid way to run containers, especially in small situations. There are other ways which I'm going to be showing you in the next lesson but you can run **Docker on a single EC2 instance and it's especially useful for any development or testing workloads where you just want to spin up a simple container and test it out.**

Now, I do have some files prepared for this lesson, and so I'm going to download them onto this instance, and to do that, I need //to install git, which allows us to download files from a GitHub repository. So I'm going to do a sudo yum (which is the packet manager) install git. So that will install git. I'll need to acknowledge that and once it's finished, we can use it to download the lesson files. Now to download the lesson files, I'm going to do a get clone <URL>. but you will need to type that press Enter to clone that repositories of all the files for all of the lessons in the course. Once that's done we've got a folder called content AWS 2019 which is the course files. So I'm going //to go into that folder so   
cd content-aws-2019/lesson\_files/03\_compute/Topic number5\_Containers/Docker/.   
So I'm going to a listing and what we should have is containercat.jpeg, Docker file, and index.html and those are the three files that we'll be using for this lesson. Ls -la

Now, we're going to do next is make a **Docker image** and **Docker images are what are essentially used to spin up containers. Docker image is essentially all of the files and all the libraries and any other components that application running inside a Docker container uses.** So if I move back to this architecture diagram, if this blue box is the container than what is used to create this container is the Docker image and a **Docker image is essentially just a file system, and it consists of a number of layers, and each of those layers added to the previous one.** So it starts off with a base layer and then any application configuration or installation that you do adds an additional layer and this layer architecture is really efficient, and it means that when you downloading Docker images, you only have to download the file system layers that you don't already have.



So this is the Docker file we'll use. It starts off with the from Directive, **which specifies the base image**. So the base OS image that's going to be used for this container, The next line is the maintainer and that's **feline incorporated** and then for any run or copy commands within a Docker file, **it's going to create a new layer**. So this first command is going **to install a web server** and the second command is going to **copy the index.html and the containercat.jpeg files there inside this folder,** is going to copy those into that image. Then it's going to set the entry point for this container. So the web server is going to be the first process that starts up in this container, and it's going to expose port 80. So a Docker container, obviously any applications that run inside the container will use network ports, and then the Docker container itself will expose one or more of those ports to the outside world and in this case, we're exposing port 80. So the web server that will be running inside our container will be exposed to port 80 on the outside of the container and then we can access that from a web browser using the EC2 instance.

So I think it's easier just to demonstrate exactly how this works and I'm going to do that using the  
docker build -t containercat .  
**//Docker build**. So I'm going to build an image. I'm going to specify -t and I'm going to call the image containercat. Next, I'm going to put the period(.) which indicates the **current directory**. So that's where it's going to look for this Docker file. So it's going to load in this Docker file and based on the contents of this Docker file so these directives, it's going to build the image that we'll use to make container. So I'm going to run this command. It will download this base image, which is CentOS and then for any of the directives inside the Docker file, it will install applications, configure applications, and then it will create our image. So now we've got an image. We've got the container cat image. So if I do a docker images I'll be able to see any images that are on this local machine. We've, of course, got the CentOS base image because this is used as the base image for container cat. So it's downloaded and ready to be used and then container cat which is based on this image with additional installations. So the web server and then our assets that use for our website so index.html and the JPEG.   
docker run -t -i -p 80:80 containercat  
So now we can use this image to run a container, and we can do that by doing a docker run -t -i- p. So this is what controls the port mapping and I want to map port 80 on the Docker host to port 80 on this container and then only to specify container cat. We won't see any other messages. We're good to go. So if I get the public DNS name for this EC2 instance and open it in another tab, we'll see the website that's running inside this Docker container which is a cat in a container in a container because it's running in Docker. Now this is a simple example, **but this website is running inside a container that's running on top of the container engine inside the EC2 instance**, we could have made this as complex or as simple as we've liked. What we’re essentially doing is by using this Docker file, we've defined how to create this container. The benefit of creating a container is we could upload this container, use it on other Docker hosts, and we would know that in every situation this container would run as expected. **Containers provide isolation, so they contain all the libraries and runtimes everything an application needs to run. If we've got another container which needs a different libraries, runtimes, and applications we can run both of those side by side without any conflicts any versioning issues and that's one of the benefits of containers.**

Now as the last step I'm going to cancel out of this container. Clear the screen to make it easier to see. We've also got the concept of a **container registry**. So a **container registry is a way that you can upload your containers to an online or a private repository so all the Docker hosts can utilize the images that you've created to make containers and a popular public version of this is the Docker hub**.   
So what I'm going to do at this point is I'm going to upload the container I've just made to the public Docker Hub so other people can utilize it.   
docker login –username acantril  
Now to do that, I need to use Docker login and then double dash --username, and I need to specify my username on Docker hub, which is a.cantrill. So I'm going to press Enter and it'll ask me for a password and I'm going to enter my password and press Enter. It will store it in an **unencrypted form so you do need to be aware of the security implications of logging in using standard usernames and passwords.** There are many more secure ways you can do this if this is production, but this is just a quick demonstration of how this works. Now, if I wanted Docker images to generate a complete list of images on this machine on the image ID of container cat. So I'm going to copy that into my clipboard.   
docker tag <image ID> <user name>/containercat  
I'm going to run Docker tag and then image ID. So I'm going to tag the container cat image, and I'm going to tag it with acantrill/container cat. So acantrill is my username on Docker hub and then containercat is the name of the image that I'm going to upload. So that's what I'm going to tag it as and then I'll run a Docker push and I'm going to push this to acantrill/container cat.   
docker push acantril/containercat  
**Now that's going to upload all of the layers there in this Docker image up to Docker hub**. Notice one of them is layer already exists, and that is the base file system layer that's used for my container image because it's a popular one because I downloaded it and modified it to make containercat it will already exist on Docker Hub and so it doesn't need to upload it again. **So this is again one benefit of using this lead file system format for images you'll need to ever upload anything that doesn't exist at the remote end.** Now, that does mean that the container cat image is now available for the world to use on Docker hub so I'll make sure I include a link in the lesson description.

Now, one thing that's important to understand for the exam is when you should and should not use Docker or containers in general. **Remember that Docker containers use the same underlying kernel.** **They're just isolated processes. So what ever kernel it's used by the operating system the container engine is running on is also shared by all of the containers. So you can't utilize Docker if you want to use different operating systems, Remember, Docker just provides process isolation it's still to share the underlying resources without the same level of isolation that virtualization provides**. **If you've got any business requirements where you can't utilize the same OS kernel, if you got security issues or governance issues, then you can't utilize containers**. **If all of your compute requirements need long running compute and they all use the same OS then using containers will gain you massive efficiencies because you can pack these containers more densely on a given set of infrastructure and not have to duplicate the operating system.**