*Docker*

**Docker for the Absolute Beginner – Hands On - DevOps**

(Tutorial – Channel Name: Udemy)

Videos - #1 - #27

**First Published On: 14th Jun 2021**

**Last Updated On: 15th Jun 2021**

**Docker**

|  |  |
| --- | --- |
| Installing Docker (Udemy Video #5)                  Docker Commands (Udemy Video #8)                            **Commands Hands On (Udemy Video #9)**            **Docker Run**            **Docker Run Advances Features (Udemy Video #17)**        **Installing Jenkins (Skip below to next section for shortcut way of installing and running jenkins)**          **Installing Jenkins and mapping ports directly**            **Docker Images (Udemy Video #21)**        **Creating a new Docker image (Udemy Video #22)**                            **Environment Variables (Udemy Video #24)**        **Command vs Entrypoint** | * Previously, to install different software and services, they need to have correct OS versions, dependencies, correct version of library, etc. * Due to this limitation it was difficult to install and maintain such applications * Using Docker, now all the application can run in its own separate Isolated Container with the resources required by them individually * Here the Os like Ubuntu, Fedora, ets share the same Kernel but the OS are different like – UI, drivers, etc * So, Docker can run all the flavors/distro of OS if the underlying Kernel is same i.e. Linux in this case * This will not work for Windows OS if underlying kernel is Linux * Although, windows OS is supported on Docker, however, in background it creates a virtual m/c of Linux and then you can run linux containers on windows * Ideally its Linux on Linux (VM) on Windows * Virtual Machines have separate OS for each VM and are installed on hypervisor like ESXi * Containers are installed on Docker and are isolated. Docker share the same OS * You have readymade docker templates/images of OSs configured and installed and are available on dockerhub or docker store * You can just install them and run docker run <<application>> easily. * Containers are running instances of Images which are isolated and run on their own env with set of processes * Goto docs.docker.com * Enter this command to check if there are any older Docker images already installed * Run the 1st command to download the copy of the script from internet to local server * And second command (sudo sh get-docker.sh) to execute the script * Lets now run a simple Container * Goto hub.docker.com * Search for whalesay * Type this command * Whalesay image will be pulled and will give a whale output as shown * Runs nginx * If its not available, it will be pulled from dockerhub * *docker ps* – list of containers * *docker ps -a* – list of containers started and stopped * to stop a container, use *docker ps* to find the name * then do *docker stop <<name>>* * *docker rm <<name>>* * Get list of images – *docker images* * Once all the containers are stopped and removed * You can get rid of the downloaded image file as well * *docker rmi nginx* * As *docker run <<name>>* command is used to run a container * If you just want to download the image, *use docker pull <<name>>* * To go into any container and do some operations, *docker exec <<name>> cat <<path>>* * This means you are opening a file available under image Ubuntu name: distracted\_mcclintock * If you run the 1st command shown, this means it will run and show the execution on screen and you wont be able to perform any other operations unless this current operation is completed * To let it run in background, just add a -d to the command, and it will work in background * Later if you want to see the execution, just do docker attach <<first few chars of image name>> * Note that first few letters of image also work unless it is unique than others * To install centos, use *docker run centos* * It will first check locally, if not available, it will download from official store * In this case it downloaded from official store – “library” (check this line in the image – line #3) * To run this os and enter in the container, use this command * Run centos container for 20 secs * It will automatically stop after 20 secs * Run centos for 2000 secs * Stop/kill the container while active * Remove the completely completely from the disc * You can use container ID or image name to remove * rm is used to remove containers * rmi is used to remove images * Remove multiple images in a single command * To go inside a container and read a file * 1st command is from a file and is not dockerized so it asks for input and prompt * 2nd command - Docker by default is not interactive, so it does not ask for input or prompt * 3rd command – to ask for input use *–i* tag * 4th command – for the prompt, use *–t* i.e. –*it*  together * Basically, if you run web application and want other users to access it, you need to map the port with IP * But which port and IP? There is internal ip for each container, however we cannot use that * We need to use ip of Docker Host (underlying docker having these containers) * To map ports use commands as shown in image * *docker logs <<container name or id>>* * Appending tags * Tag in this case is the version appended using a colon : * You can use any tag from the following to run container of that version (e.g. 17.10 or artful or devel) * Run Jenkins container * To find the ip address, use this command, this command gives all the detailed information about the specified image * IP address * Accessing Jenkins Internally * To access the Jenkins from external clients, do the port mapping using this command. * Done using single command * Mapping Volume command * This is require if you want to backup all the data and configurations * This will be useful when the container is destroyed, we can restore this backup * Use this command to map the volume * Install the Jenkins, you can continue to install plugins and create build jobs * Next time, when you deleted the container, and want to run again, use this command again, it will automatically bring the plugins and build jobs * This is also helpful if you want to run same instance of Jenkins in different container with this preconfigured data * For example, we want to create an image for a project * You first create a docker file, and give instructions * Then you build this file * Finally you push it to Docker Registry * The instructions gets executed in layered architecture, i.e. 1st executed, then 2nd which will take resources of 1st execution and so on * Here we are just creating a container of OS Ubuntu * *-it* is used for attaching our input and entering into prompt * bash is used to open terminal so that we can stay always in the container * These are the steps to install the flask app * Flask app is a simple web server hosted application * Create a folder first * Start writing into dockerfile usin cat command * FROM – this is always required to tell dockerfile which OS to run * RUN – to tell the dockerfile that these instructions should be executed * COPY – to copy contents of file (currently app.py file is not available, we will add this file in next step) * ENTRYPOINT – tell dockerfile that this is the command to run the application. * Create a file app.py and copy the complete code here * Build the image * Give a name to the image * Run the created image * We also need to tag this image to our repository. Replace mmumshad with your repo name * Next, login to your dockerhub * Push this to the dockerhub * Instead of giving color in code, you can set an environment variable, which acts like a parameter * Use this command to know the environment variables * Basically, in any dockerimage, if we want to overwrite some commands, we can do it using CMD <<command>>, however this is static command * Although we can overwrite this as well using typing our command after the docker run, as shown * In order to give the parameters on runtime, you use ENTRYPOINT, and pass the parameter * But in this case it will give error if no parameters are passed * To overcome above issue, use both CMD and ENTRYPOINT as shown, if no parameter is passed it will execute CMD |