|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name:** WASIMA QAYYUMUDDIN SHAIKH  **RollNo:**6220071  **Class:** T.E.I.T  **Sem:** V  **Subject:** ADVACE DEVOPS LAB **(Addevops**)  **EXPERIMENT NO: 09**  **1.** **What is Containerization / Docker? Explain Docker Architecture with the help of diagram**  Docker is the containerization platform which is used to package your application and all its dependencies together in the form of containers so to make sure that your application works seamlessly in any environment which can be development or test or production. Docker is a tool designed to make it easier to create, deploy, run applications by using containers.    Docker is the world’s leading software container platform. It was launched in 2013 by a company called Dotcloud, Inc which was later renamed as Docker, Inc. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VM’s. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of systems. Docker comes into play at the deployment stage of the software development cycle.    **Docker Architecture**  Docker architecture consists of Docker client, Docker Daemon running on Docker Host, and Docker Hub repository. Docker has client-server architecture in which the client communicates with the Docker Daemon running on the Docker Host using a combination of REST APIs, Socket IO, and TCP. If we have to build the Docker image, then we use the client to execute the build command to Docker Daemon then Docker Daemon builds an image based on given inputs and saves it into the Docker registry. If you don’t want to create an image then just execute the pull command from the client and then Docker Daemon will pull the image from the Docker Hub and finally if we want to run the image then execute the run command from the client which will create the container.   Docker follows Client-Server architecture, which includes the three main components that are Docker Client, Docker Host, and Docker Registry.  Docker Architecture  **1. Docker Client**  Docker client uses commands and REST APIs to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.  Docker Client uses Command Line Interface (CLI) to run the following commands -  docker build  docker pull  docker run  **2. Docker Host**  Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.  **3. Docker Registry**  Docker Registry manages and stores the Docker images.  There are two types of registries in the Docker -   1. Pubic Registry - Public Registry is also called as Docker hub. 2. Private Registry - It is used to share images within the enterprise.   **2. Compare Containers vs VMs**  In traditional virtualization, a hypervisor virtualizes physical hardware. The result is that each virtual machine contains a guest OS, a virtual copy of the hardware that the OS requires to run and an application and its associated libraries and dependencies. VMs with different operating systems can be run on the same physical server. For example, a VMware VM can run next to a Linux VM, which runs next to a Microsoft VM, etc.  Instead of virtualizing the underlying hardware, containers virtualize the operating system (typically Linux or Windows) so each individual container contains only the application and its libraries and dependencies. Containers are small, fast, and portable because, unlike a virtual machine, containers do not need to include a guest OS in every instance and can, instead, simply leverage the features and resources of the host OS.  Just like virtual machines, containers allow developers to improve CPU and memory utilization of physical machines. Containers go even further, however, because they also enable [microservice](https://www.ibm.com/cloud/learn/microservices" \t "_blank" \o "microservices) architectures, where application components can be deployed and scaled more granularly. This is an attractive alternative to having to scale up an entire monolithic application because a single component is struggling with load.     |  |  |  | | --- | --- | --- | | **Sr No.** | **Virtual Machines(VM)** | **Containers** | | 1 | VM is piece of software that allows you to install other software inside of it so you basically control it virtually as opposed to installing the software directly on the computer. | While a container is a software that allows different functionalities of an application independently. | | 2. | Applications running on VM system can run different OS. | While applications running in a container environment share a single OS. | | 3. | VM virtualizes the computer system. | While containers virtualize the operating system only. | | 4. | VM size is very large. | While the size of container is very light; i.e., a few megabytes. | | 5. | VM takes minutes to run, due to large size. | While containers take a few seconds to run. | | 6. | VM uses a lot of system memory. | While containers require very less memory. | | 7. | VM is more secure. | While containers are less secure. | | 8. | VM’s are useful when we require all of OS resources to run various applications. | While containers are useful when we are required to maximise, the running applications using minimal servers. | | 9 | Examples of VM are: KVM, Xen, VMware. | While examples of containers are: RancherOS, PhotonOS, Containers by Docker. |   **3. Why are Containers lightweight?**  [Containers](https://www.ibm.com/in-en/cloud/learn/containers) are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.  With virtualization technology, the package that can be passed around is a virtual machine, and it includes an entire operating system as well as the application. A physical server running three virtual machines would have a hypervisor and three separate operating systems running on top of it.  By contrast a server running three containerized applications with Docker runs a single operating system, and each container shares the operating system kernel with the other containers. Shared parts of the operating system are read only, while each container has its own mount (i.e., a way to access the container) for writing. That means the containers are much more lightweight and use far fewer resources than virtual machines.  Containers share the machine OS kernel, eliminating the need for a full OS instance per application and making container files small and easy on resources. Their smaller size, especially compared to virtual machines, means they can spin up quickly and better support [cloud-native](https://www.ibm.com/cloud/learn/cloud-native) applications that scale horizontally.  **4. Deploy a containerized web Application on AWS EC2 Linux. [install Docker ,pull nginx image and run it]. Pull python images and run the command to list all the locally stored docker images**   1. First of all, we have to create IAM role Service 🡪 IAM 🡪 Create role 2. In this choose EC2 🡪 Next      1. Then in Permission section type ssm and choose “AmazonEC2RoleForSSM”. 2. Then in review section type the role name and click on create role button      1. After successfully creating role we have to create EC2 instance Service 🡪 EC2 🡪 Launch Instance      1. Then choose AMI here we are choosing Amazon Linux 2      1. Then in configure instance detail 🡪 IAM role section choose the IAM role which we have already created by above step. 2. Then in Configure Security Group 🡪 Add Rule 🡪 (HTTP) 3. Then review and Launch by downloading key-value pair      1. After launching the instance, we need to connect to our instance by clicking on connect. Then we need to select SSH client.      1. After that open Termux qpp -> locate private key file by using cd command 🡪 Execute chmod 400 wasima12.pem for publically viewable to connect instance execute command ssh -I “wasima12.pem” and it’s Public DNS. 2. After connecting run the following command    1. sudo su // For entering root user    2. yum install docker -y // To install docker    3. service docker start // To start the services of docker      1. Now for downloading docker image go to chrome 🡪 hub.docker.run 2. In search bar type nginx on right hand side there will be one command copy that and run-in terminal (To install the docker image)      1. Again, in search bar type python on right hand side there will be one command copy that and run-in terminal (To install the docker image)        1. Then type “docker images” to see your images      1. To run the image type “docker run -p 80:80 nginx”      1. After that go to instance 🡪 details 🡪 Public DNS 🡪 Copy that Address and paste in browser     **Note: After successfully running docker image, terminate the EC2 Instance** |