Utilizing Hardware's used in Distributed Applications using Docker Container

Khushbu S. Patil [1], Kajal B. Gaikwad[2], Gayatri D. Borawake [3], Kajal S. Kale [4], Prof.Mrs. Pooja Mishra[5]
Department of Computer Engineering
[1, 2, 3, 4, 5] DYPIEMR, Akurdi

Abstract- *Docker is an open platform for developing, shipping and running various applications in faster way. Docker helps to ship code faster. Docker also helps to test fast, deploy fast and make shorts the cycle between writing and running code. Docker use client server architecture. It improves the process of management of application which we are developing. Ex. If we want to restart one application without disturbing other application, we can do it easily; Docker contains 3 components – Docker image, Registries, Containers.*

Keywords- Docker tool, Java, Virtualization, Web Technology (HTML)

I. INTRODUCTION

Docker is an open platform for developing, shipping, and running various applications in a faster way. Docker enables the applications to run separately from the host infrastructure and treat the infrastructure like a managed application. By taking advantages if docker's methodologies for shipping, testing and developing code quickly, one can significantly reduce the delay between writing code and running it in production [1]. For example - In the previous years, if one user wants to run more than one operating system then he has to use different hardware system.After that the concept of virtualization was came in picture.

In virtualization one can use more than one operation system on same machine. But this increases the machine overhead. In virtualization same amount of memory, CPU cycles were taken by virtual system [2]. Hence, docker is new technology which comes over the drawbacks of virtualization.

A. Docker Platform

Docker provides functionality to package and run an application in container i.e. a loosely isolated environment. Many containers can be run simultaneously on a given host usingisolationand security [4]. One can run more containerson a given hardware without creating extra load on hypervisor due to lightweight nature of container.

B. Docker Engine

Docker engine is a client-server application having mainly following components-

1. A server which is a type of long running program called daemon process.
2. A REST API which specifies interface that programs can use to talk to the daemon and instruct it what to do.
3. A command line interface client.

Docker is an integrated application. It is easy to deploy into environment for building and shipping application. Docker for windows is a native application with a native user interface. It is having auto-update capability with native virtualization, Hyper-v, networking and file system. For making it faster and more reliable than previous ways, getting Docker on a Windows PC.

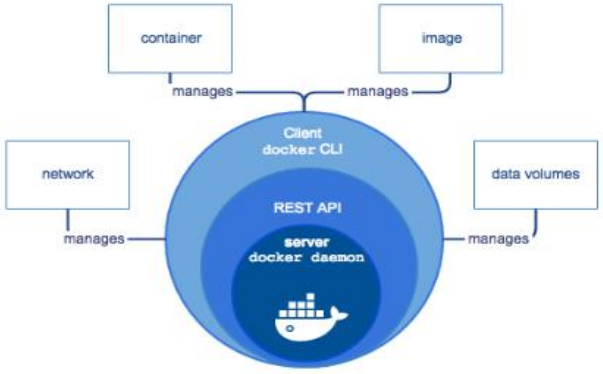

Fig. 1.1. Docker Engine

II. LITERATURE SURVEY

For developers and system administrator Docker provides an open platform to build, ship, and run distributed applications in a faster way. Main components of Docker are Docker images, registries, and containers. Different applications can be run over docker container without dependency of languages [5]. Docker uses resources isolation features of Linux kernel and kernel namespaces (identifier or variables) that allows independent containers to run within a single Linux instance, this will avoids the overhead

of virtual machine. The security and isolation provided by Docker allows running many containers simultaneously on a single host. Multiple containers may share the same kernel but each container can use a defined amount of resources available in the host machine [2]. Critical applications and time-sensitive applications hosted in high priority containers will get a great share of network bandwidth without starving other containers [1]. Docker test and deploy the applications fast and make a short cycle between writing and running code. Docker uses the client server architecture and improves the process of management of application which we are deploying [3].

III. HYPERVISOR BASED AND CONTAINER BASED VIRTUALIZATION

There are two models are used to deploy virtualized instances. Which are the hypervisor-based and container -based platform [1]. For VMs, hypervisor known as a layer to deploys, allocates operation space of instances. A hypervisor, a piece of computer software, hardware or firmware creates and runs VM's. Host machine is a computer on which hypervisor runs one or more virtual machines and virtual machine is called guest machine. Virtual operating platform is present by hypervisor to the guest OS and manage execution of it. Multiple instances of different operating systems may share the virtualized hardware resources.

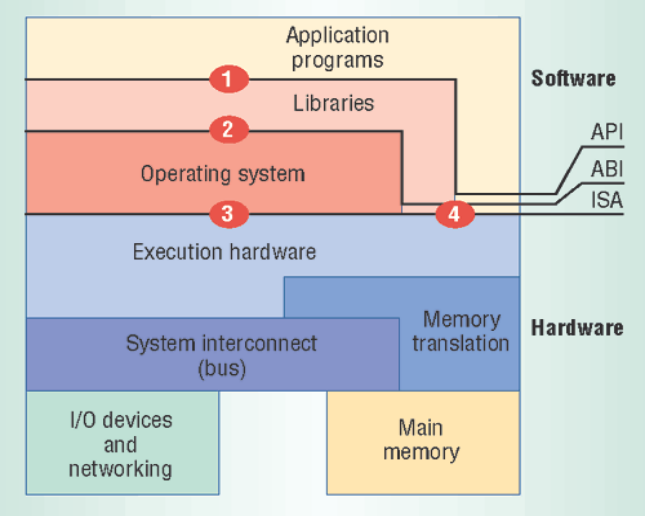

Fig.3.1. Computer system architecture

In the architecture of real computers, we need an interaction between software and hardware to operate a system. There are three important interfaces, application binary interface (ABI), instruction set architecture (ISA) and application programming interface (API).

Above figure shows computer system architecture which is used for communicating with key implementation layers via interfaces. In virtual machine, a process or system running as guest and the underlying platform support virtualized instances is host. Hence, they need hypervisor [10] or VM monitor for deploying and managing VMs. From the approach of the operating system and the application supports, a produced virtual machine has a whole execution environment which can perform many processes cumulatively. It can allocate individual I/O resources and memory to the processes. The VMM has to match the hardware ISA that the guest software can execute.

IV. HOW DOCKER IS BETTER THAN VM'S

1) Virtual machines runs on virtual hardware and guest operating system will be loaded in its own memory. In Docker, guests share same operating system, which is the Host operating system, is loaded in the physical memory.
2) Communication between guests is done through the network devices in virtual machines while in Docker Communication between guests is done through pipes, bridges, sockets, etc.
3) Booting is faster in Docker where in VM it takes time in booting.
4) Due to light weight containers less overhead are occur in Docker. Due to complexity in virtual machine more overheads are occurs.
5) In virtual machine sharing of libraries and files are not possible while in Docker sharing of files is possible.
6) Docker uses less memory as it shares host operating system. Virtual machine uses more memory as it has to store complete operating system for each guest.
7) Docker provides a way to run almost any application securely isolated in container [2].

V. ARCHITECTURE

It is a client server architecture in which client talks to the docker daemon, which does the heavy lifting of building, running and distributing docker containers. The client and daemon can be run on same system or one can connect a docker client to a remote Docker daemon. The client and daemon communicate using a REST API, over UNIX socket or a network interface [7].

A. Docker Image

It is a read only template having instructions for creating a Docker container [5]. For example an image can be the modules of student, teacher and administration. One can build or update image from scratch or download and use other's image. An image can be based on, or may extend one or more images. A docker image is described in text file called Dockerfile. Docker images are the build component of docker.

B. Docker Containers

These are the runnable instance of image. Run, stop, move or delete these operations can be performed on containers using Docker APT or CLI commands [5]. Each container is isolated and secure platform but can be privileged with access to resources running in different host or container as well persisting storage or databases.

It is an isolates application from each other on a shared operating system. This approach standardizes application program delivery, allowing apps to run in any Linux environment, whether physical or virtual. Because they share the same operating system, containersare portable among different Linux distributionsand are significantly smaller than virtual machine (VM) images.

C. Docker Registries

It is the library of the images. Registry can be public or private, or can be on same server as the Docker daemon or Docker client, or on totally separated server [3].

D. Union File System

Union file systems, or UnionFS, are operated by creating layers, making them fast and very lightweight. UnionFs is a file system for Linux, NetBSD, and FreeBSD that will implement a union mount for other file systems. UnionFs allows directories and files of separate file system to form a single coherent file system. Directory contents which have same path within merged branches that will be merged in a single directory within new virtual file system.

UnionFs is used for creating a single common template for number of file systems and also for security reasons. Docker Engine uses Union file system. That provides the building blocks for containers. Docker Engine can use multiple UnionFS variants, which includes, btrfs, vfs, AUFSand DeviceMapper [3].With the help of UnionFs Docker can layer Docker images. Actions are done to base images then layers get created and documented, such that each layer completely describes how to recreate an action. This will enables Docker's lightweight images, as only layer updates need to be propagated.

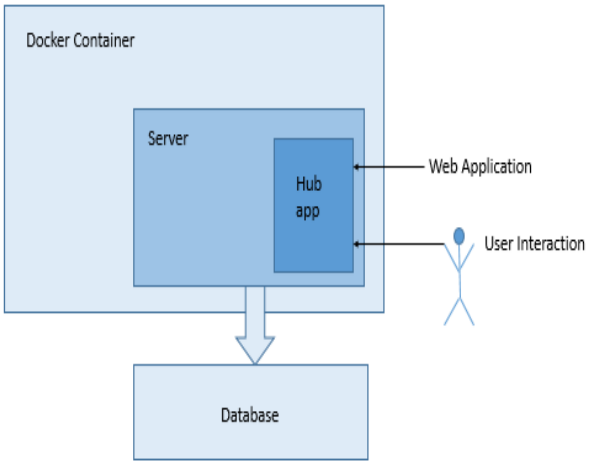

Fig. 5.1. Architecture Diagram

VI. IMPLEMENTATION

Each image consists of series of layers. These layers are the reason of Docker being lightweight. When one change the image by updating the application the newer version, a new layer is build and replaces only the layer it updates. The other layers remain intact. One needs to transfer only the updated layer to distribute it.

Each layer is having the base layer as Ubuntu image or fedora image. One can also create his own base image. For example creating the images of student module, teacher module and administration module.

Docker registry stores Docker images. After building image one can push it on public registry i.e. Docker Hub or the private registry running behind firewall. One can also use existing image by pulling it from registry to a host. Docker Hub is having large collection of existing images.

For example for updating the student image in above system one need not to update all the system modules. One just need to pull or create new student image and run it on container.

A container uses the host machine’s Linux kernel and consists of any extra files added along with metadata associated with container at the time of creation or when the container is started. Container build of images which define container’s contents which will be running when the container is launched. Images are read only by default but it adds read write layer on the top of the image in which application is running.

When one runs Docker through CLI command or equivalent API, the engine client instructs the daemon to run the container.

So working of docker can be concluded as follow:

1) Create the Docker image that holds your Application
2) Create Docker Container from those images to run yourapplications
3) Share those Docker Images via Docker Hub or on your own registry.

Public and private both storage for image is provided by Docker Hub. Public storage can be searched and downloaded by anyone. Private storage is only for limited users; is excluded from search results and only you and your users can pull image down and used to build container.

When you run the container, either by using the docker binary or via the API, docker daemon is ask to run a container.

Using run command docker client launches a new container of the docker binary.

A. Operations performed by docker during launch

1) Pulls the image:
Docker checks initially whether the requested image is present locally on host machine and, if not then it will download from Docker Hub[3]. If the image is already present on host then it will use the image to create new container.

2) Creates a new container:
If docker is having the image then it will create a container.

3) Allocation of filesystem ant mounting read-write layer:
Container is created in file system and read-write layers are added to the image.
4) Allocation of network/bridge interface:
Network interface is getting created so the docker container can talk to the local host machine.

5) Setting up an IP address:
From a pool an IP address is fetched and allocate.

6) Execute the process that used requests:
Run application that is requested.

7) Captures and provides application output:
Docker will connect and log standard input, output and display errors made to see how application is running.

B. Docker Events

The events API is the fantastic feature of docker that enable tools like Registrator and Logspout for listing the container events like starting and stopping [8].

Docker API documentation which provides list of available events without knowing what they mean and when will occur.

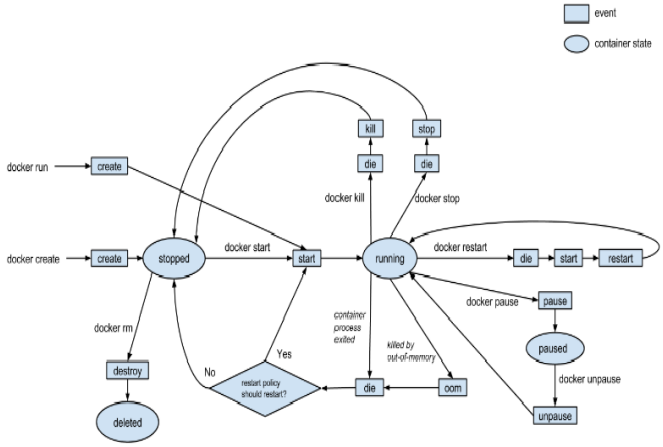

Fig.6.1 Events of Docker

Some container related events are:
Export: emitted by docker export
Exec_create: emitted by docker exec
Exec_start: also emitted by docker exec after exec_create

VII. DEPLOYING DISTRIBUTED APPLICATIONS ON DOCKER

Many types of applications can be configured with different approaches depending on virtualized architecture. For e.g., VMs as the hypervisor-based instances have full components emulated by hypervisor layer, e.g., OS, hardware libraries. Hypervisor has to deploy an entire filesystem and OS in each VM. This results in the overhead of emulating libraries and OS when producing a large range of VMs. The advantage of VMs is isolation, and disadvantage is overhead when running distributed applications. This is also one of problem that developers have to consider in PaaS field. On the basis of container-based architecture, Docker is a platform supporting containers that can share the same related libraries and OS kernel.

Containers can share common files because their images are constructed from layered filesystems [6]. At the time of running a job, each containers assigned a unique PID, it can be observed equally as a process at the view of host machine. These characteristics of Docker, we deploy applications that share the same dependencies, necessary libraries under the host machine. This method is available for portable computations

and solving scalable problems because we can reduce remarkably the overhead, when comparing to VMs.

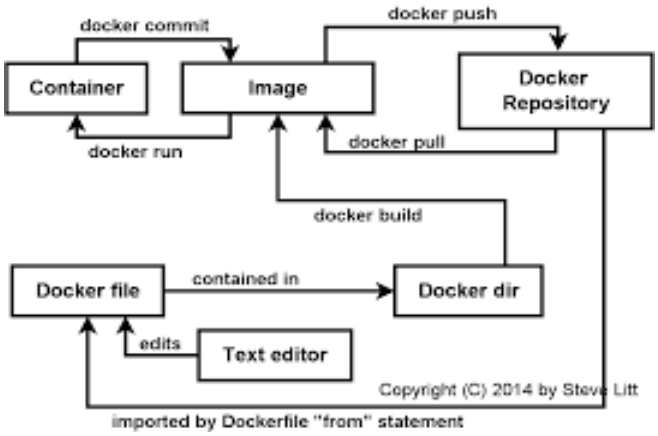

Fig. 7.1. Model for deploying distributed applications on Docker and Virtual Machine

Above Fig shows the model which user deploy Docker container to run applications. VMs provide a complete environment which supports multiple applications as well as users. VMs emulate the full OS and hardware along with individual libraries. Hence, the same way to configure distributed application on VMs and host system. We need to install and configure applications, libraries inside each virtual machine to execute as a cluster. In opposition to VMs, we exploit the sharing ability of Docker with host OS kernel to deploy applications. Every Docker container does not need to set up a whole image or OS with related libraries, they can share the same libraries and binaries during executing.

VIII. DOCKER'S KEY ADVANTAGE

Docker provides lightweight virtualization with almost zero overhead. The effect of this delivers some impactful advantages. Docker can have many more containers running on single machine than we can with virtualization. Another powerful impact is that container bring up and bring down can be accomplished within seconds. It provides Portable deployment of applications as a single object versus process sandboxing, Application-centric versus machine/server-centric. It can Supports for automatic container builds. It Built-in version tracking; having Reusable components; can be Public registry for sharing containers. It is a growing tools ecosystem from the published API.

IX. RESULTS

For updating the module of student, user needs not to update whole system after addition of student image. He just needs to upload and run new update image of student which is not affecting the remaining modules. Similarly, remaining modules can also be updated without disturbing other modules as and when required.

X. CONCLUSION

Docker container technology is become developing platform used in cloud computing field. Our research shows that VMs and Docker container have both positive and negative factors. Thus, we have to consider about the target of utilization and the feature of application type running on them. Also VMs have a strong point about the isolation criterion. Docker containers have numerous benefits in reducing overhead because the architecture allows sharing the OS kernel.

As the Docker's architecture, these characteristics to execute applications efficiently, our evaluation shows that the utilization of VMs and Docker containers have many advantage about probability. Also have more advantages like convenience and scalability. We need to pay attention about the problem sizes, application types and system limitations. For example, Docker is more suitable than virtual machine about data intensive applications.

In the future, we explore the transmission capacity in the cluster system, which is formed by VMs and Docker containers. This future work will support the resource scheduler in allocating VMs and containers. Docker is getting code tested and deployed into production faster than virtualization. Different applications can be run over Docker container with language independency. The results include much less CPU utilization, memory utilization, CPU count; network I/O counter, etc. as compared to the previous virtualization technique.

REFERENCES

[1] Ayush Dusia, Yang, Michela Taufer, "Network Quality of Service in Docker Containers", IEEE International Conference on Cluster Computing, 2015.

[2] Preeth E. N., Fr. Jaison Paul Mulerickaly, Biju Paulz and YedhuSastriz, "Evaluation of Docker Containers Based on Hardware Utilization", International Conference on Control, Communication & Computing India (ICCC), 2015.

[3] https://docs.docker.com/

[4] https://docs.docker.com/engine/understanding-docker/

[5] http://prakhar.me/docker

[6] Minh Thanh Chung, Nguyen Quang-Hung, Manh-Thin Nguyen, Nam Thoai," Using Docker in High Performance Computing Applications".

[7] http://nordlcapls.com/api-driven-devops-spotlight-on-docker/

[8] http://gliderlabs.com/devlog/2015/docker-events-explained/

[9] J.Hwang, S. Zeng, F.Y.Wu and T.Wood, "A component-based performance comparison of four hypervisors", in Integrated Network Management (IM 2013).