Monitoring and Deployment of Web Services Using Docker as PAAS

Mr. Tondare Brijesh B.[1], Ms. Sagun Bisht[2], Ms. Patole Priyanka[3], Ms safina Shaikh[4], Mrs.Aarti Gaikwad[5]
Department of Information Technology
[1, 2, 3, 4] D Y PATIL College of Engineering, Aurdi,Pune
[5]Assistant Professor,D Y PATIL College of Engineering, Aurdi,Pune

Abstract-*This poster presents an extension to the currently limited Docker's networks, to guarantee quality of service (QoS) on the network, our extension allows users to assign priorities to Docker's containers or configures the network to service these containers based on their assigned priority. Providing QoS not only improves the user experience but also decrease the operation value by allowing for the efficient use of resources. Our implementation ensures that time-sensitive, critical applications, hosted in high-priority containers and get a greater share of network bandwidth, without starving other containers.*

I. INTRODUCTION

In order to ensure high-quality performance for serious applications executed in Docker's on tainers, a required level of service should be ensured without expanding or over-provisioning the network. unfortunately Docker's networks are currently configured to provide the "good effort" to all the traffic; and parameters such as Bandwidth, reliability, and packets per second for a specific application cann't be guaranteed. Consequently a single bandwidth-intensive application results in poor performance for any other application sharing the Docker network. That problem can be solved by introducing quality of service mechanisms that provide preferential treatment to traffic and applications. Since the Docker networking is in its infancy , it does not provide QoS yet. In this poster, we address this problem by proposing a QoS mechanism that enable spreferential delivery service for critical applications in Docker, while providing sufficient bandwidth, controlling latency and delay, and reducing data loss.

A virtual Ethernet bridge,is created (that is Docker0) when Docker boots up. By default, all the containers are configured to be in the same no of subnet and to use docker0 so that they can communicate with one another. For every container, a pair of virtual Ethernet interfaces is created, and an IP address is assigned to the them. Currently, no options are available to configure network shares, bandwidth, and most imp priority, as in the case of other resources that is CPU and memory. Thus each container gets an equal use of the bandwidth. Assuming that each container is dedicated to host only a one application, a container is for a real-time application requires higher priority over regular applications hosted on other containers. Thus, Docker needs QoS(quality of service) mechanisms that provide differential services to containers with respect to network bandwidth and that are based on priority criteria, as our proposed mechanism does.

following dig(Fig 1) shows the architecture of Docker. The implementation consists of a packet classifier or priority scheduler. The packets in the flows are classified and added to one of the three available priority queues. The scheduler dequeues the packets and sends that each packet to a container according to the queues priority. Our implementation provides the functionality to assign this priorities to containers. The priority values are high, medium, and low, where medium is the fix default value that is assigned to a container. The packet

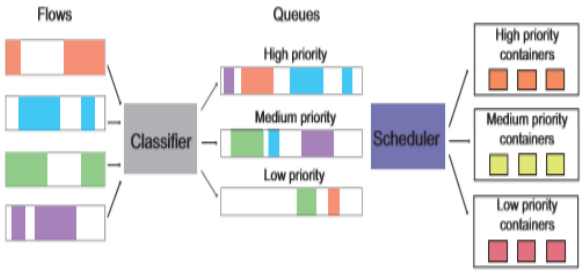


Fig. 1. Architecture of our priority queue scheduler.

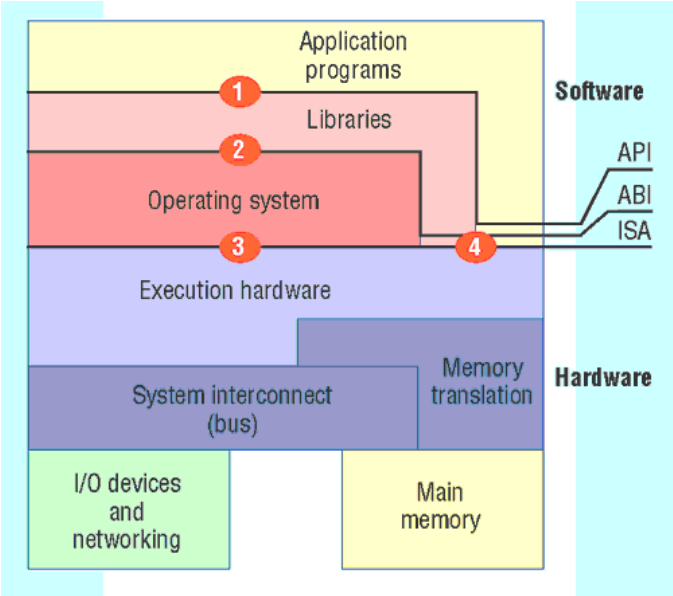
classifier and scheduler are built on the top of the docker0 bridge, which prioritizes the network access that containers. A higher share of the total available network bandwidth is provided to the containers with highest priority.

II. HYPERVISOR BASED AND CONTAINER BASED

VIRTUALIZATION There are two models are used to deploy virtualized instances. Which are the container and hypervisor based platform . For Virtual Machines, hypervisor known as a layer to deploys, allocates operation space of instances. A hypervisor, a part of computer software, hardware or firmware creates and runs Virtual Machine's. Host machine is a computer on which hypervisor runs one or more than one virtual machines and virtual machine is called guest machine.

Virtual OS(operating system) is present by hypervisor to the guest Operating System and manage execution of it. Multiple instances of different OS (operating systems) may share the virtualized hardware resources.

In the architecture of real computers, we need an interaction between hardware and software to operate a system. There are three important Fig. 2. Computer system architecture interfaces, instruction set architecture (ISA)application binary interface (ABI) and application programming interface (API).



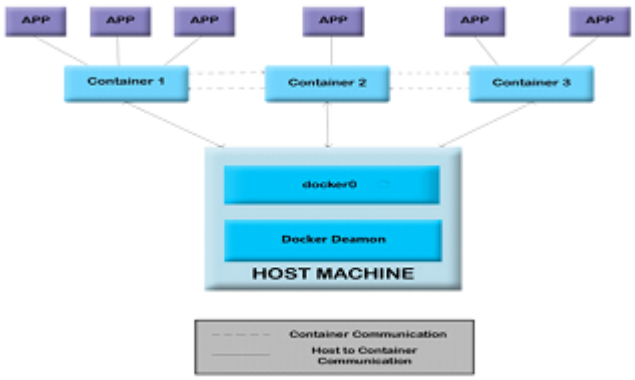
Above fig. shows computer system architecture which is used for communicating with key implementation layers via interfaces. In VM, a process or system running as guest and the underlying platform support virtualized instances is host. So, they need hypervisor or Virtual Machine monitor for deploying and managing VMs. From the approach of the OS(operating system) and the application supports, a produced VM has a whole execution environment which can perform many processes cumulatively. It can allocate individual Input output resources and memory to the processes. The VMM has to match the H/W ISA that the guest software can execute.

III. HOW DOCKER IS BETTER THAN VM'S

VM's runs on virtual hardware and guest OS(operating system) will be loaded in its own memory. In Docker, guests share same OS(operating system), which is the Host operating system, is loaded in the physical memory.

- Communication between guests is done through the N/W devices in virtual machines while in Docker Communication between guests is done through pipes, bridges, sockets,router etc.
- Booting is faster in Docker where in Virtual machine it takes time in booting.
- Due to light weight containers less overhead are occur in Docker. Due to complexity in VM more overheads are occurs.
- In VM sharing of libraries and files are not possible while in Docker sharing of files is possible.
- Docker uses less memory as it shares host OS(operating system). Virtual machine uses more memory as it has to store complete OS(operating system) for each guest.

IV. ARCHITECTURE



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

- Docker Image

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

- Docker Containers

These are the runnable instance of image. Run, move,stop or delete these operations can be performed on containers using Docker CLT or API commands . Each container is isolated and secure platform but can be privileged

with access to resources running in different container or host as well persisting storage or databases.

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

- Docker Registries

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

V. DEPLOYING DISTRIBUTED APPLICATIONS ON DOCKER

Many various types of applications can be configured with different approaches depending on virtualized architecture. For e.g., Virtual machines as the hypervisor-based instances have full components emulated by hypervisor layer,for example, OS(operating system), hardware libraries. Hypervisor has to deploy an entire filesystem and OS(operating system) in each VM. This results in the overhead of emulating libraries and OS(operating system) when producing a large range of VMs. The advantage of Virtual machines is isolation, and disadvantage is overhead when running distributed applications. This is also one of the 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(operating system) kernel.

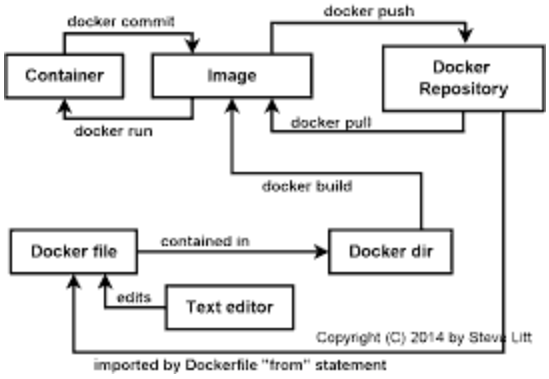


Fig. 3. Model for deploying distributed applications on Docker and Virtual Machine

Containers can share same files because their images are constructed from layered filesystems . At the time of running a work, each containers assigned a unique ID, it can be observed equally as a process at the view of host machine. These property of Docker, we deploy applications that share the same dependencies, necessary libraries under the host machine. This method is available to portable computations and solving scalable problems because we can decrease remarkably the overhead, when comparing to Virtual machine.

VI. DOCKER'S KEY ADVANTAGE

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

VII. CONCLUSION

Our extension to the Docker networking presented here guarantees Quality of service to containers, so that their network bandwidth(BW) matches assigned priority. Providing Quality of service to containers has few advantages: containers hosting user-sensitive applications, such as real-time multimedia or some high-BW applications can now be assigned higher priority; and operating costs can be decreased by using existing N/W resources more efficiently and thus delaying or decreasing the need for expansion or upgrades. Moreover, when containers host applications using UDP, which is not more sensitive to N/W congestion, our QoS implementation allows such containers to be throttled appropriately to achieve the desired levels of BW sharing across all containers. The complete source code of our implementation is available on GitHub.

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/