NEC

Adopting Agile Environment:

Utilizing docker Docker Containers for Accelerating & Optimizing Testing

-Authors:
Mr. Khelender Sasan, Sr. Solutions Architect
Ms. Divya Saxena, Test Lead

1 Abstract

The current technology business landscape is characterized by scenarios where time to market and resource cost optimization are crucial. Additionally several new paradigms (cloud, distributed computing, DevOps etc) are evolving that are going to impact development methodologies and going to impact testing methodologies as well.
Looking at testing domain specifically, in past decade, virtualization technologies (Hypervisor based virtualization) have helped in accelerating and optimizing the pace by better utilization of advancements in compute technologies. Still, the main bottleneck of such setups was that bulk of resources (hardware and software level) are consumed for simulation of virtual machines, where in-spite of having best server class hardware, over-all utilization for end application purpose is not optimal and hence has a bearing on overall expenditure utilization as well as pace of test execution.

In addition to Hypervisor based virtualization, OS level virtualization (Containers) was being researched and developed, but so far hasn't been used much for testing. The recent introduction of Docker Container platform has provided an excellent framework (to use containers) for enhancing (in resource usage perspective) and accelerating the pace of multiple kinds of testing.
Docker framework builds upon strength of Linux Containers and Advanced Union File System to provide a platform where-in testing can be further accelerated, optimized and streamlined for CI/CD targets.

Additionally, Docker container also is being researched for application into newer paradigms discussed above.This paper provides a brief idea of Docker Containers technology, and discusses technology & business developments around it. It also proposes ways in which testing teams can optimize resource usage for testing environment and as well accelerate their pace of testing. It tries to explain various scenarios use cases that may be addressed by Docker framework.

2 Docker Technology

In computing space, key shifts in technology have been characterized by redefinition of unit model for computing. During the legacy OS era, threading provided a key shift in application implementation aspect which provided a credible alternative to everything being done in process space. Subsequently VMs (virtual machines) started acting as a segregated compute units as per user's expectations (to serve needs of multiple end customers that required it information security needs). The next analogous revolution is going to be caused in VM space by Docker Container that has provided a credible alternative for segregating such spaces by using a very light weight contained environments for similar purpose.

From application perspective, Docker is a framework built on top of containers (LXC or otherwise), that can package an application and its dependencies in an image format that can be launched in form of containers on any Linux server (providing portability with respect to underlying platform, viz. cloud, bare-metal system, servers, desktop or laptop).

> *LXC (*LinuX Containers*) is an operating system–level virtualization method for running multiple isolated Linux systems (containers) on a single control host.*

2.1 Introduction

From technology perspective, *Docker extends Linux container format (LXC), with a high-level API providing lightweight virtualization that runs processes in isolation. Docker uses LXC, cgroups, AUFS (file system with copy-on-write capabilities), and the Linux kernel itself. Unlike traditional virtual machines, a Docker container does not run a separate operating system. Instead, it relies on the kernel's functionality provided by the underlying cgroups and LXC mechanisms that provide resource isolation (CPU, memory, block I/O, network, etc.) and separate namespaces to completely isolate the application's view of the operating environment.*

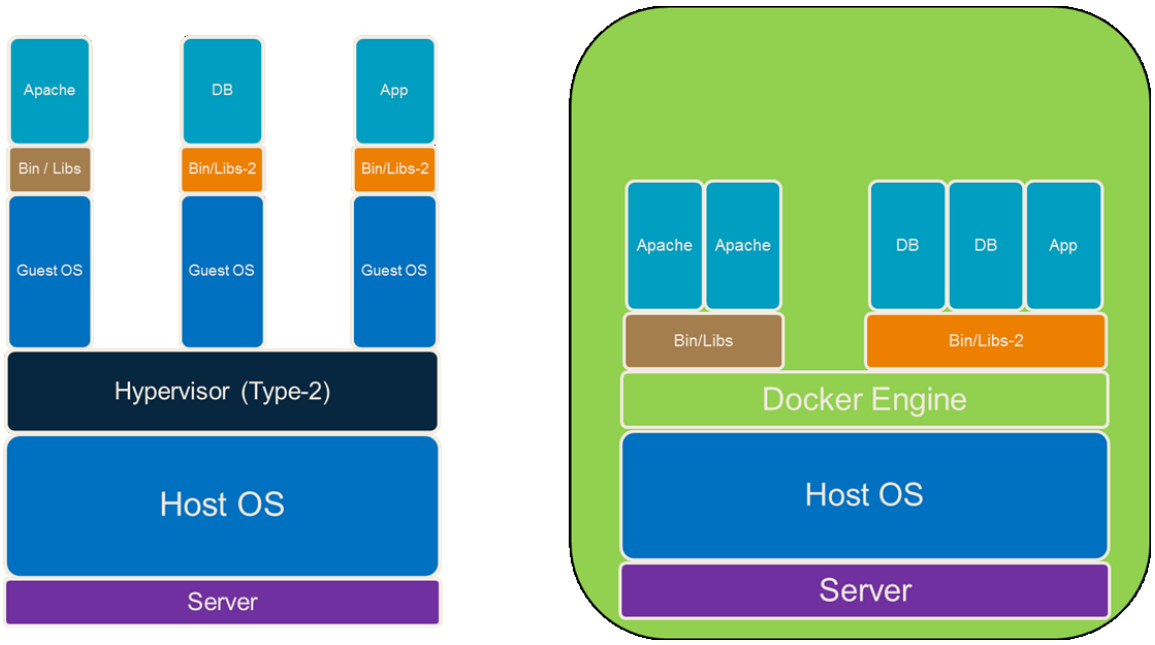


Figure 1: VM versus Docker Containers

From application hosting perspective, Docker container (that comprises of just the application and its dependencies) runs as an isolated process in user space on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient. (In terms of software stack for simulating segregated environment, docker containers require only few MB of capacity.)

In contrast, in case of VM, each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh few GBs. (In terms of software stack for simulating segregated environment, VMs typically require several GBs, and in most optimal case possibly few 100MBs of capacity)

2.2 Docker Architecture

Docker framework involves following component:

1) Docker Engine (framework on host machine)
    a. Docker daemon
    b. Local Images
    c. AUFS
2) Docker containers (running instances of images)

3) Docker client (command line interface for interacting with docker)
4) Docker Registry (Global or private repository that hosts pre-built images)

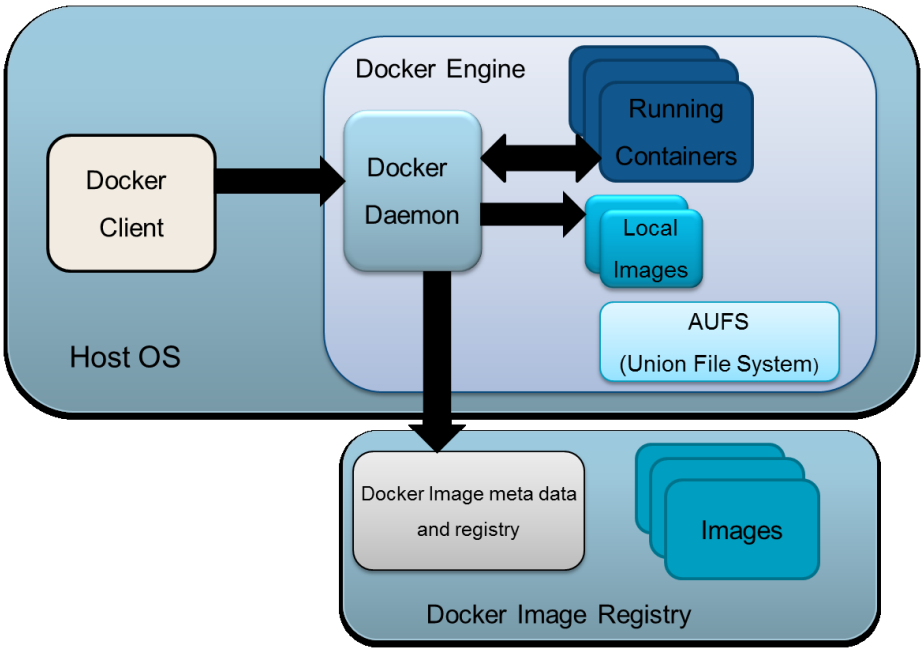


Figure 2: Docker Architecture

Docker Engine refers to the core framework involving Docker daemon, aufs file system, local images cache. Docker daemon provides the main interface for setting up (fetching images, launching), monitoring and managing the Docker Containers. The images fetched from remote registry repository are maintained locally. Docker uses aufs to expose underlying file-system for containers, and manage the layers and changes being done to storage system by containers instances. It provides facility to inspect changes in the container w.r.t. underlying image from which container was launched, and a facility to commit the changes as a new image (in delta form) to docker registry.

Docker containers are the actual instances of images running on the host machine that are sort of equivalent to VM.

- In hosting terms, containers basically runs as separate process groups in user space of host OS, such that they are completely isolated from host OS processes and other containers (using process control groups).

- Container's file-system isolation is implemented with the help of chroot and aufs functionality (such that it remains isolated as well as light-weight)
- Container's other resources (such as network, I/O) are isolated with the help of name-space isolation.

Docker Registry is a system that provides a repository of images and interface to explore & fetch available images as well as commit new images to the repository. By default, docker daemon connects to main docker.io registry repository for fetching, and storing public images. Facility for private images is also available on paid basis. Alternatively, one can also host the Docker Registry framework on a private host (including the host on which Docker containers are running)

Docker framework and containers can be run almost on any kind of system including desktop, physical servers, virtual machines, and Mobile systems.

2.3 Technology behind additional strength

Apart from virtual environment & resource segregation facilities provided by containers, Docker has built upon its strength based on AUFS, as detailed below.

2.3.1 Advanced Union File System

Union file system consists of several layers of file / directory contents, such that the user of file system

1) Sees a final file system that is a union of all layers
2) In case of contents present in multiple layers, top layer contents are visible

The Union file system has following kind of layers:

1) Layers created from Image instances: All the layers that are instantiated from an image layers are read-only, and no changes can be made to these.
2) Layers providing volume mapping: These are the layers that are created to provide mapping of mount points to corresponding mapped points in file system of underlying host OS. They can be in either read-write or read-only mode.
3) The top layer captures any contents being added / changed / deleted in projected file system. Any change that is not being written to a layer mapped to underlying file system (Layer #2) is captured in this layer.

2.3.2 Configuration Management of running environment

Biggest benefit of file system is that all delta changes of file system are captured in one of layer (which is writable). The configuration item check-in operation means merely pushing changes from this layer into the repository.

(This is in sharp contrast to VM technology, where for any snapshot, you need to preserve complete contents)

2.3.3 File System sharing

Spawning of multiple containers from same image is also very effective, as the read-only contents across containers are used seamlessly. This leads to huge benefits in terms of storage space.

2.3.4 Volume sharing

Union file system also supports the concept of mapping of another read-write layer from existing volumes on host machine or from other containers. This leads to a benefit that any dynamic content of a container (e.g. databases, dynamic input & output data) can be directly retrieved / stored onto an underlying directory point, another container (possibly responsible for data processing) or a mapped file system (nfs etc).

3 Docker's impact on technology & business world

This section presents some general impact on technology world along with impacts on testing domain as applicable.

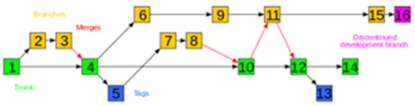
3.1 Agility

Today's technology world is characterized by frequently changing requirements and time to market is crucial for any organization. Newer paradigms of software (agile methodology, agile environment, CI/CD and DevOps) have emerged to attain such level of pace.

In past decade and half, virtualization has been used by industry to attain the increase pace of execution, still the deployment and maintenance of VMs has been a quite challenging task.

The advent of Docker Framework that uses light weight containers has made available an alternative where applications can be run almost at near native level performance. It is characterized by following agile behavior:

1. Docker containers have sub-second launch times (as compared to few seconds to minute for VMs), reducing the application deployment and execution time. As **several test environments use virtual environment**, this can lead to significant time saving to execute all test cases for such deployments.
2. Smaller size of containers also implies much lesser time for system scale up and down by launch / shutdown of containers.
   Specifically in case of *destructive test scenario* that need tearing down of environment, docker containers can be a blessing.
3. Docker's use of **AUFS file system** helps in capturing all persistent changes in a differential file system layer. This is in sharp contrast to VMs where complete image has to be stored requiring significant time in storing intermediate / final snapshots.
   It has really enabled Configuration Management of environment. (Docker Registry is such a store house where changes in environment can be committed with much ease. This can greatly accelerate the job of DevOps engineer that can further enhance the release life cycle being controlled by CI/CD methodology.)
   Specifically in testing domain, test cases execution reproduction requires configuration control on test case program / scripts, test data and related package dependencies. Focus so far has been on configuration control of test program / scripts and test data, however ensuring versioning control of package dependencies and environment was not easy, which has now been made possible with Docker containers.



4. Memory allocation and deallocation of containers is quite easier as compared to VMs, where deflating memory balloon used to be a quite challenging task.

The **key conceptualization point** of Docker Container was ensuring replica environment across all stages of development, QA & production. Motto statement is "*Getting people to agree on something*!" This replica environment in-fact ensures minimal failures across stages and problems in production environment that leads to much shorter Software Release Cycle time.

Further, several frameworks have got developed around Docker Container for further usage scenarios. Examples involve orchid & fig that helps boot up docker containers on cloud environment. CoreOS is a product that is going to further optimize the server

footprint in Linux market. All applications typically supported by servers shall run in a dockerized form and the underlying OS will be a minimalistic one.

3.2 Cost Optimization

During the past decade, lots of improvements have happened in VMs technology leading to reduction in the size of VM down from GB to several 100MBs range. Still the size of VMs is a constraining factor in virtual entities densities.

Docker Container has suddenly provided a disruptive technology that has reduced the size of containers to just few MBs. It has happened because of OS level virtualization, which has provided following advantages:

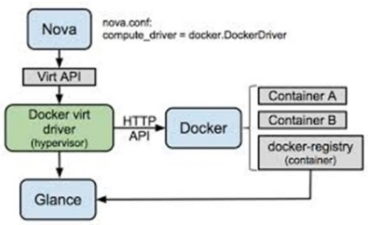
- There is no need to handle virtual machine simulation for guest OS execution. Rather host and guest OS share the same kernel.
- Dedicated hardware resources (e.g. memory, disk, CPU resources) are not a must requirement in docker containers. The resources can be shared / used among multiple instances as per need.

This has led to server hardware resources being meaningfully utilized to support large number of containers. This has an implication that the density attained by virtual containers can be of several times that of VM for a given hardware (typically in size of ~10x – 100x), leading to significant cost advantages.

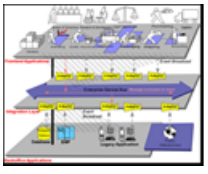
Apart from reduction in hardware requirement, no license cost for Virtual Containers and lower support & maintenance acts as an icing on the cake.

3.3 Strengthening new paradigms

Cloud has cut cost by ensuring optimal utilization of resources, but so far they have traditionally used VMs. The argument given above regarding agility of containers versus VMs also applies to Cloud frameworks. ***OpenStack*** has included a **docker component** that is going to support containers spawning in cloud environment, leading to benefits of agility applied to applications run & testing being done in Cloud environment.

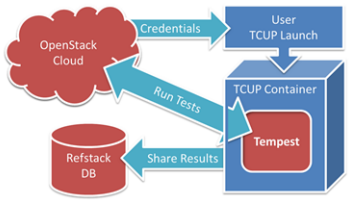


Further, Docker has created a global repository of images where several **pre-built images** are readily available, thus avoiding need of separate download and configurations at OS level for every release configuration. This is going to increase pace of application integration and deployment from CI/CD and



DevOps perspective.

OpenStack community visualizes there are going to hundreds of deployments of OpenStack across world, and sees that **interoperability** of such clouds may pose a challenge. It is working towards a framework called ***RefStack*** that is going to use docker container to launch set of test suites that shall figure out working of the cloud w.r.t. expected standards and reporting the results to a central repository.



Docker container images also inherently support the cause of pets versus cattle, where server farms can be deployed with ease by simple configuration setups (rather than requiring elaborate setups for every server in sense of a pet). Google's Kubernetes is an effort towards providing Cluster as a Service.

3.4 Docker Containers and technology business ecosystem

Docker containers have created a budge in the technology world in a very short period of time (they were introduced around May'13). Their emergence has been noticed by all major corporations, and several strategic partnerships have been announced in recent past. This section tries to present few important ones.

3.4.1 Microsoft & Docker

Docker and Microsoft have partnered to bring container applications across platforms (15-Oct-2014). As per the release

"Microsoft Corp. and Docker Inc., the company behind the fast-growing Docker open platform for distributed applications, on Wednesday announced a strategic partnership to provide Docker with support for new container technologies that will be delivered in a future release of Windows Server. Developers and organizations that want to create container applications using Docker will be able to use either Windows Server or Linux with the same growing Docker ecosystem of users, applications and tools"

3.4.2 Google & Docker

Google has launched Managed Service for running docker based applications on its Cloud Platform (4-Nov-2014). Google specified the new service is a "Cluster as a Service" platform based on Google Kubernetes. It helps developer manage their container clusters. In this service, Kubernetes dynamically manages the different Docker containers that make up an application for the user. (Docker as a Service)

3.4.3 Linux Distros & Docker

Canonical has released Ubuntu 14.10 with tighter docker integration (24-Oct-2014). SUSE has bundled docker and processor optimizations into its major new Linux distro (27-Oct-2014). Red Hat is working on bringing docker

support to Enterprise Linux (RHEL 6.5) as well as its PaaS offering OpenShift (15-Apr-2014)

3.4.4 Docker acquisitions

Docker has acquired Orchard that has developed components around Docker for Orchestration & integration of platforms. Docker has also recently acquired Koality that provides continuous integration solutions. (7-Oct-2014)

3.4.5 CoreOS: Planning to serve a diet Linux Platform (server env. optimization)

CoreOS is creating a distribution based on Chrome OS, and it has done away with package manager concept, and instead has replaced package deployments with provisioning of docker containers to run various services required by the system.

3.5 Visualizing newer solutions & Paradigms

3.5.1 QA Automation framework enhancement for distributed environment

Test Automation framework can be enhanced to use docker for orchestrating system tests in a distributed environment.

OpenStack community has visualized *"Data Center will become one big COMPUTER, and OpenStack will be the OS for it."*

For such a distributed system, test automation frameworks have to scale up and start treating complete Data Center as a single system to ensure they are able to orchestrate test suites across full deployment. Such orchestration will require docker container for isolation of development and test environments.

Additionally, such systems will require testing in a non-invasive manner, i.e. without disrupting (integrated) development / production environment (similar to OpenFlow in networking domain), and that can be done by Docker only. In long term, it might require **nested virtualization** features to do such kind of testing, and Docker is already geared up for it.

3.5.2 More possibilities

Docker can be used to do multiple kind of testing much efficiently that could have not been done so far due to constraint of resources needed, e.g.

- Interoperability testing
- Application Stress behavior
- Network DNS features testing

Additionally Docker can help developers in bringing fresh hardware setups, where frequent installs are required that may be replaced with container images run. These use cases have been detailed out in next section.

4 Use cases for application of Docker Containers in testing domain

This section outlines various use cases that may be applicable for testing domain. This is in addition to usage of Docker in any testing environment that involves usage of virtualized environment for test system deployment of any type.

4.1 Optimize frequent install & setups during hardware bring-up

In a case where frequent installs, setups and/or configurations are required for reason of hardware bring-up, faulty hardware (e.g. legacy) etc, setup contents can be captured in form of some container images, that may be deployed very quickly on every fresh install

saving significant hours of installation, need for internet connectivity to download packages etc.

4.2 Non-invasive testing

In lot of scenarios, test cases & related environment setup require setting up of a minimal to elaborate environment setup that sometimes can have a bearing on conflict of QA environment w.r.t. production environment and ultimately on test results. In a manner analogous to non-invasive surgery, docker container framework offers a minimal setup that can be done on target platform and it can house any kind of QA environment and related tests suites offering QA environment that is almost equivalent to pre-production / production environment.

This is especially true for Distributed systems testing, where for ST (System Testing), number of setups will be minimal and the tests need to run on components across physical deployments and run maximum tests without changing anything in system environment.

4.3 Scalability testing

As Docker container can spawn several "named" instances of an application running under specifically numbered (named) container, it is very easy to launch new containers and provide its detail (host name, port connections to other containers, shared storage volumes across containers, number of processors, memory limits etc) to configuration scripts of such kind of application.

4.4 Interoperability testing across Linux Distributions and application packages

Interoperability testing across various software package versions and Linux distributions currently is quite resource and effort consuming.
In Docker container framework, several Linux distributions (where application is not dependent upon any version specific kernel features) may be run very efficiently. This coupled with the fact that images for specific software packages versions may be made available from Docker registry repository, Docker platform can support interoperability testing with numerous software / library packages.

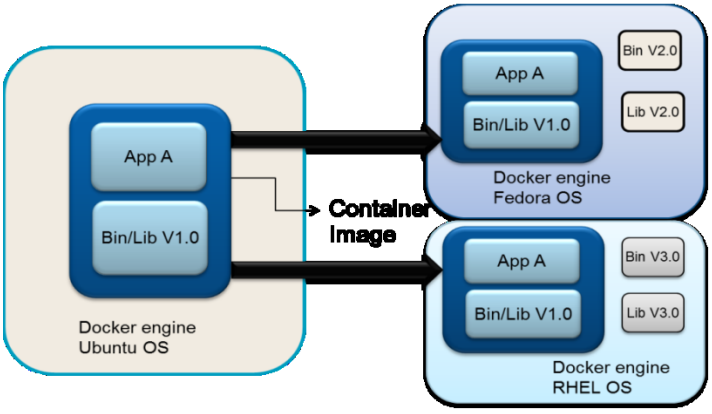


Figure 3 : Interoperability Testing across different Linux distributions / versions

4.5 Application Stress testing: behavior validation

Application behavior testing under stress condition can be done using docker. As Docker technology matures, it will be possible to launch containers with dedicated CPU and memory limit. Application running inside the container can be then stressed via imposing high load or stress within the container.

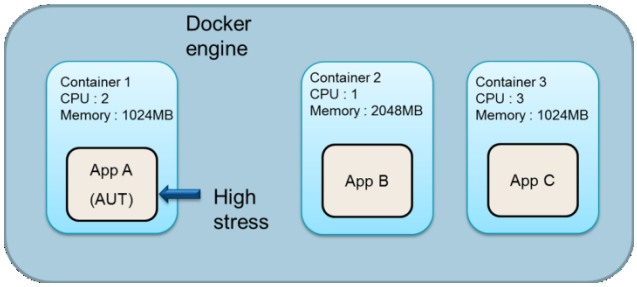


Figure 4 : Stress test using containers

Test suite execution on AUT Under high stress condition can be performed to judge its behavior. This type of containerize test has advantage that it will not affect rest of the applications running on other containers.

4.6 Implementation optimizations

A test setup that has any one of following feature can be very easily optimized by use of docker container images.

1. Requires regular clean-up after each test case run
2. Needs some kind of common setup steps

Basically, the docker container images are responsible for providing the initial setup / environment expected, and quick stopping/removal of containers removes the requirement of writing and executing cleanup steps. Apart from fast execution, it also helps in avoiding costly human errors done during setup etc.

4.7 Networking test cases

Test setups for networking (using VM setups) that test network dis-connect behavior can be easily implemented using docker. Additionally, network tests that require

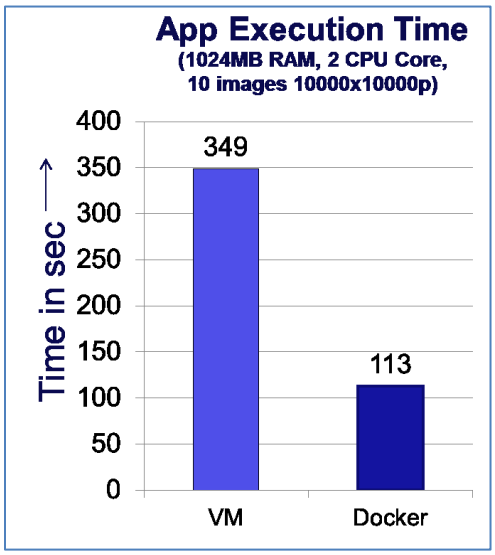
elaborate virtual machine setups for DNS configuration testing can also be tested very efficiently using Docker.

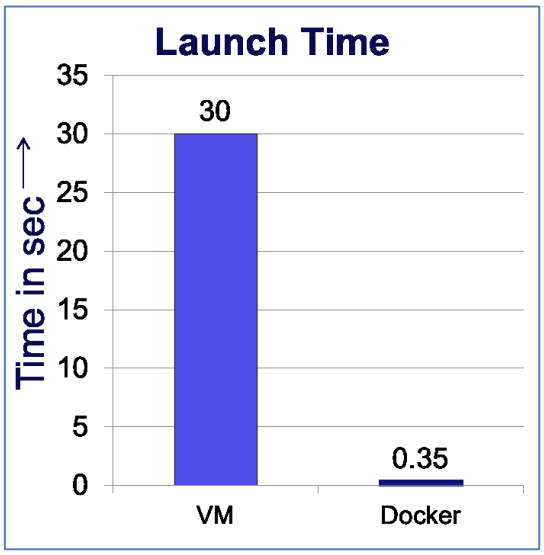
4.8 Enhancement of Test Automation Frameworks

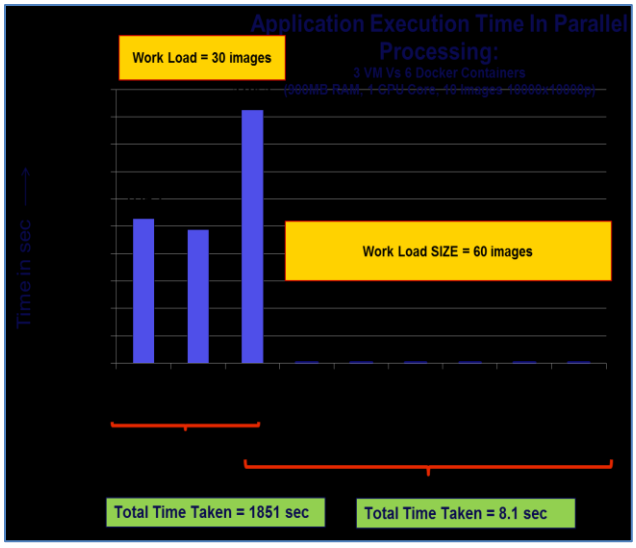
Docker container framework can be used to enhance existing test automation frameworks. Currently test automation frameworks do testing on target system only. They can be enhanced wherein specific parts of test automation framework can be residing in multiple component deployments across systems, such as providing a single coherent view to test automation frameworks.

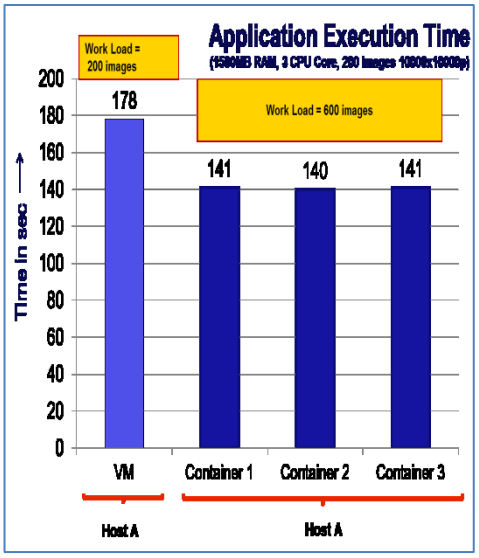
4.9 Performance benchmark findings for VM versus container experiment

An image processing application was run in context of VM and docker container for various work-load conditions and hardware configurations. The corresponding results have been shared below that clearly demonstrates launching and execution time of application on docker container is much higher as compared to traditional VMs.









5 Bibliography

1. Docker (http://www.docker.com/)
   a. Introduction to Docker (November-2013) http://www.slideshare.net/Docker/dockerintronovember-131125185628phpapp02-37588934
   b. Docker: Automated and Consistent Software Deployments (http://www.infoq.com/news/2013/03/Docker)
   c. Running private repository (https://blog.codecentric.de/en/2014/02/docker-registry-run-private-docker-image-repository/)
2. Docker performance characteristics
   a. http://bodenr.blogspot.in/2014/05/kvm-and-docker-lxc-benchmarking-with.html
3. Budge around Docker
   a. Docker and Microsoft partner to bring container applications across platforms **(**http://news.microsoft.com/2014/10/15/dockerpr/**)**
   b. Google to offer Docker as a Service (using Kubernetes) (http://siliconangle.com/blog/2014/11/06/with-docker-as-a-service-google-wants-developers-to-think-beyond-cloud/)
   c. CoreOS (based on docker) as a lean and mean virtualization machine (http://www.networkworld.com/article/2840343/opensource-subnet/coreos-a-lean-mean-virtualization-machine.html)
4. Docker Acquisitions
   a. Docker acquires Koality (http://techcrunch.com/2014/10/07/docker-acquires-koality-in-engineering-talent-grab/)
   b. Docker acquires Orchard (http://www.zdnet.com/docker-acquires-london-startup-orchard-laboratories-7000031921/)
5. Miscellaneous

a. Ubuntu server 14.10 adds new features for docker containers (http://siliconangle.com/blog/2014/10/23/ubuntu-server-14-10-adds-new-features-for-openstack-containers/)
b. Automated Testing of Hardware Appliances with Docker (http://www.appneta.com/blog/automated-testing-with-docker/)

6 Author's Biography

Khelender Sasan

Khelender has been associated with NEC for more than 10 years and currently responsible for Technology Practice team setup at ITPF BU at NEC Technologies India. He has a rich experience of around 19 years across diverse platforms software (OS, servers, storage and cloud frameworks) for both development & QA. He has been also responsible for setting up and mentoring teams in few other engineering domains. In his prior assignments with HCL Technologies, he has handled engineering responsibilities for several MNC projects.

He graduated in Bachelor of Engineering (Electronics & Communication) from Delhi University (NSIT) and also holds PMP certification.

Divya Saxena

With more than 7 years of industry experience, Divya Saxena is presently associated with NEC as Test Lead. Her primarily skill is test automation and she has successfully handled several test automation related assignment by using test automation tools like Selenium, Xnee, and X11-GUI and has been responsible for developing test automation frameworks as well. Apart from it, she has responsible for handling test automation assignments in her past organization Interra System India Pvt. Ltd.

She has graduated in B.Tech. - Electronics & Instrumentation from UP Technical University.

THANK YOU!