A New Virtualized Environment for Application Deployment Based on Docker and AWS

Gemoh Maliva Tihfon, Jinsul Kim and Kuinam J. Kim

Abstract The setup environment and deployment of distributed applications is a human intensive and highly complex process that poses significant challenges. Nowadays many applications are developed in the cloud and existing applications are migrated to the cloud because of the promising advantages of cloud computing. The very core of cloud computing is virtualization. In this paper, we will look at application deployment with Docker. Docker is a lightweight containerization technology that has gained widespread popularity in recent years. It uses a host of the Linux kernel's features such as namespaces and croup's to sandbox processes into configurable virtual environments. Presenting two common serious challenging scenarios in the software development environment, we propose a multi-task PaaS cloud infrastructure using Docker and AWS services for application isolation/optimization and rapid deployment of distributed applications.

1 Introduction

Software deployment is complex and the diverse computing requirements for applications require complex hardware infrastructure setups and possibly incompatible specific software requirements. Therefore, a platform to automate the deployment and setup of virtual computing is essential. Moreover, it is important to properly and efficiently manage the computing resources so as to reduce additional investment in hardware. All these lead towards the concept of cloud computing.

G. M. Tihfon · J. Kim(✉)
School of Electronics and Computer Engineering, Chonnam National University, Gwangju 500-757, Korea
e-mail: gemohmal@gmail.com, jsworld@chonnam.ac.kr

K.J. Kim(✉)
Convergence Security Department, Kyonggi University, Suwon, Gyonggi-Do, Korea
e-mail: Kuinamj@gmail.com

K.J. Kim and N. Joukov (eds.), *Information Science and Applications (ICISA) 2016*,
Lecture Notes in Electrical Engineering 376,
DOI: 10.1007/978-981-10-0557-2_126

Cloud computing is a paradigm to rapidly provision computing resources such as storage, networks, servers, services, etc., that can be customized and configured to suit a particular user or application demands[1]. Cloud computing paradigm is promising because is changing the way enterprises do their businesses in that dynamically scalable and virtualized resources are provided as a service over the internet. The cloud is enabled by virtualization, automation, standardization. The very core of cloud computing is virtualization, which is used to separate a single physical machine into multiple virtual machines in a cost-effective way. By using virtualization, we're basically getting a lot of the work done for free. With virtualization, a number of virtual machines can run on the same physical computer, which makes it cost-effective, since part of the physical server resources can also be leased to other tenants [2] [4]. Such virtual machines are also highly portable, since they can be moved from one physical server to the other in a manner of seconds and without downtime; new virtual machines can also be easily created. Another benefit of using virtualization is the location of virtual machines in a data center. It does not matter where the data center is located and the virtual machine can also be copied between the data centers with ease [3]. VM's in the cloud offer rapid elasticity and it is pay as you go model.

One thing to keep in mind before pushing forward is that, it's all about the applications and not the operating system. We need operation system to facilitate the applications. Virtual machines (VM) reduce capex and opex but also have some limitations that can be address. VM still requires a CPU, storage allocation, RAM, and an entire guest Operating system (OS). OS consume a lot CPU, RAM, Disk storage, and increase overhead. The more VM's you run, the more resources you need. Also some operating systems might need individual licensing. Moreover application portability is not guaranteed. The VM module does nothing to help but with docker we don't have to worry about all the issues mentioned above.

IaaS cloud computing is hugely influence by hypervisor virtualization [6]. Lightweight virtualization technologies such as Docker, LXC, and Open VZ etc, seems to be a good fit for the cloud although lightweight virtualization is limited but they provide a better hosting density. In general it is possible to host more lightweight virtualizations on a physical host than with hypervisor based virtualizations [5]. Docker can be deployed to any environment or device being public or private cloud because it is super lightweight. A docker container does not include the full OS as mention earlier, but shares the OS of its host. As a result, Docker containers can be faster and less resource-draining than virtual machines. A full virtual machine can take a couple of minutes to launch because of boot time and other stuff, however a container can be initiated in a blink of an eye (seconds). Containers also offer superior performance for the application they contain, compared to running the application within a virtual machine.

2 Problem Statement and Description

Scenario 1: Microservice architecture is an approach that makes web based development more agile and code bases easier to maintain. This architecture enables developers to be highly productive and to quickly iterate and evolve a code base. For fast moving startup companies, the microservices architecture can really help dev teams be quick and agile in their development efforts. The disadvantage of microservices is that, because services are spread across multiple hosts, it is difficult to keep track of which hosts are running certain services.

Linux containers can help mitigate many of these challenges with the microservices architecture. Linux containers make use of kernel interfaces such as cgroups, namespaces, and unionfiles, which allow multiple containers to share the same kernel while running in complete isolatioin from one another. The Docker execution environment uses a module called Libcontainer, which standardizes these interfaces. It is this isolation between containers running on the same host that makes deploying microservices code developed using different languages and frameworks very easy. Using Docker, we could create a DockerFile describing all the languages, framework, and library dependencies for that service. The container execution environment isolates each container running on the host from one another, so there is no risk that the language, framework, library dependencies used by one certain container will collide with that of another.

Scenario 2: You have written a code for some website or developed a mobile app for a game using development environment on your laptop. After thorough testing and realize that your code is ready to be deploy in the working environment or in your working organization. The system admin dutifully deploys the most recent build to the test environment and in no time notice that your recently developed REST endpoint is broken. After uncountable hours of troubleshooting with the system admin, you discover that the test environment is using an outdated version of third-party library, and this was causing the REST endpoints to break. Differences between developments, test, stage, and production environments is a familiar problem in today's rapid build and deploy cycles.

The solution is to find a way to transfer from one environment to another seamlessly and eliminating error prone resource provisioning and configuration. Services like Amazon EC2, AWS CloudFormation, and Docker provide reliable and efficient way to automate the creation of an environment. Amazon EC2 makes web-scale cloud computing easier for developers. AWS CloudFromation gives developers and system admins an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable manner. You simply create or use prebuilt template which is a JSON file that serves as a blueprint to define the configuration of all the AWS resources that make up your structure and application stack. On a plus, CloudFormation is free of charge and you pay only for the AWS resources needed to run your application. Docker takes the concept of declarative provioning a step further. Docker provides a declarative syntax for creating containers. However, Docker containers don't depend on any specific virtualization platform; neither do they need a separate operating system to run. A container simply requires a Linux kernel in order to run. This means dockerized apps can run anywhere on anything being desktop, laptops, VMs, datacenter or instances in the cloud. Docker containers use an execution environment called Libcontainer, which is an interface to various Linux kernel isolation features, like cgroups, namespace, and unionfiles. This architecture allows for multiple containers to be run in complete isolation from one another while sharing the same Linux kernel. Because a Docker container instance doesn't require a dedicated OS, it is much more portable and lightweight than a virtual machine. The core components of Docker are the Docker daemon and Docker client. Docker daemon is the engine that runs on the host machine and it is a server process that manages all the containers. Docker client is a CLI used to interact with the daemon. The key concepts to understand the workflow of Docker as shown in Figure 1 are its workflow components. Docker images, registry, containers, and Dockerfile.

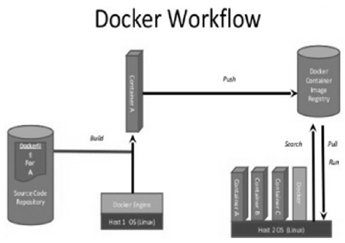


Fig. 1 Docker workflow diagram

Docker containers are becoming the go ahead for all distributed systems because they are scalable in the sense that these containers are extremely lightweight which make scaling up and scaling down very fast and easy. Dockerized applications are extremely portable; we can move them very easy. With the isolated containers, we can put more than one into a machine thereby making more efficient use of our resources. Another huge plus point of Docker is the Docker community. This community is one of the fastest growing open source communities out there. Chef, Puppet, Cloud providers such as AWS, OpenStack Azure, and Rackspace are just a few of the recognized members. There are many more benefits, but what all this mean is that it dramatically reduces the entire development life cycle from development, to testing, and then deployment.

3 Related Works

There are many works in software management and tools that address the deployment of virtual infrastructures and applications. Numerous cloud providers, such as AWS provide tools to deploy virtual infrastructures, applications and websites. In particular, CloudFormation and OpsWorks provides developers and systems administrators with an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion [8].

The Nimbus project team group has developed a set of tools to deploy virtual infrastructures: the Context Broker [9] and cloudinit.d [10]. In particular, the last tool submits, controls, and monitors Cloud applications. It automates the virtual machine (VM) creation process, the contextualization, and the coordination of service deployment[7]. It supports multiple clouds and the synchronization of different 'runlevels' to launch services in a defined order. Furthermore, it provides a system to monitor the services that uses user-created scripts to ensure that they are running. This system checks for service errors, re-launching failed services or launching new VMs. However, it enables the contextualization of VMs using simple scripts, which are insufficient in complex scenarios with multiple VMs and different Operating Systems.

One common limitation of all the above systems is the usage of manually selected base images to launch the VMs. This is an important limitation because it implies that users must create their own images or they must previously know the details about software and configuration of the image selected. This limitation affects the reutilization of the previously created VMIs, forcing the user to waste time testing the existing images or creating new ones. Another issue is that most of them need to use a VMI specifically configured to support their environment, requiring specific software installed or a set of scripts prepared.

The next section is our proposed cloud platform to address and improve most of these related works. Also in the next section is an algorithm we created to effectively and fully utilize the available resources in any data center or organization setup environment.

4 Proposed Cloud Infrastructure

Based on the numerous advantages of Docker containers, ease of deployment in the development, test, stage, and production environment and how Docker containers fit well in the distributed systems architecture (microservices). We propose a private-multitask PaaS cloud system. This PaaS cloud system using Docker is for infrastructure virtualization and application isolation/deployment. In a multitask environment, the number of containers will be increasing, and this becomes increasingly difficult to manage manually. This is where the services of Amazon EC2 Container Service (ECS) steps in to help our container management framework (cluster computing). With ECS, we effectively abstract the low-level resources such as CPU, memory, and storage, allowing for highly efficient usage of the nodes in a compute cluster.

Initially the idea we had was to use the Docker swarm which is a native clustering solution provided by Docker. It takes the Docker Engine and extends it to enable you to work on a cluster of containers. Using Swarm we can manage a resource pool of Docker hosts and schedule containers to run transparently on top, automatically managing workload and providing failover services. Swarm uses an algorithm called Bin Packing Scheduling algorithm (they also support Random and Spread algorithms) and some scheduler filters (Constrain, Affinity, Port, Dependency, and Health filters) to effectively manage the containers on a subset of nodes. Swarm is the future native clustering for Docker. Currently swarm has many limitations such as it doesn’t support image management yet, it is still beta and not recommended for production. So using Amazon EC2 Container service is the right choice for scalability and management of Docker containers.

EC2 Container Service is a cluster management framework that uses optimistic, shared state scheduling to execute processes on EC2 instances using Docker containers. Amazon ECS makes it easy to launch containers across multiple hosts, isolate applications and users, and scale rapidly to meet changing demands of your applications and users. Using ECS incurs no extra charges, apart from the cost of spinning up EC2 instance. The ECS takes care of many of the challenges in running a distributed system. Customers need not about monitoring the health and availability of nodes that provide the scheduling and resource management capabilities. ECS also provides a robust solution to the very challenging problem of storing state information in a distributed system. Lastly, ECS is designed to scale horizontally and for high availability. Container instances, clusters, tasks, and task definition are the key components of ECS.

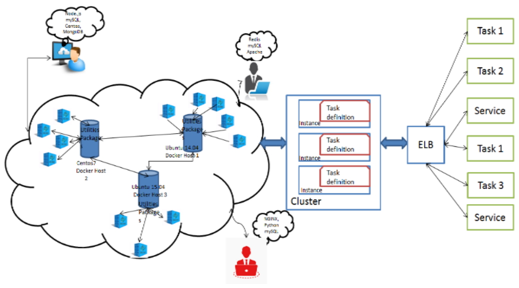


Fig. 2 Propose multi-task cloud

5 System Workflow

The propose platform allow organizations/sysadmins/developers to focus on building products rather than building infrastructure. As mention earlier, we can build, test, and debug our code on any machine capable of running Docker. When the code is ready, we can package it up into the Docker image by building the image from a Dockerfile and storing it in Docker Hub (repository). Next, we need to provide the compute resources required to run containers. In ECS, this is called a cluster, and it consists of EC2 instances called “container instances” that are running the ECS agent. To create an ECS cluster of container instances, we simply launch one or more EC2 instances using the Aazon ECS-Optimized Amazon Linux AMI. The instance will need to be associated with an IAM role that allows the agent running on the instance to make the necessary API calls to ECS. The next step is to tell ECS how to run the containers. We use an entity called a “task definition.” ECS task definition can be thought of a prototype for running an actual task. For any given task definition, there can be zero or more task instances running in the cluster. The Task definition allows for one or more containers to be specified. ECS has another entity called a "service," which is useful for long running tasks, like web applications. A service allows multiple instances of a task definition to be run simultaneously. It also provides integration with Elastic Load Balancing (ELB) service. The ELB is used to distribute tasks and services to different containers efficiently.

6 Schedule Algorithm

The schedule algorithm below aims to schedule applications on VMs based on the user deployment request and deploys VMs on physical resources based on resource availability. This strategy optimizes the application performance. Additionally, the load-balancer ensures high and efficient resource utilization in the cloud environment.

Schedule Algorithm for VMs

```
1: Input: UserDeploymentRequest
2: get Resources&AvailableVMList
3: // find applicableVMList
4: if AVM(UDR, ARS) != 0 then
5:     //call the load balancer
6:     deployableVM = load-balance(AVM(UDR, ARS))
7:     deploy UserRequest on deployableVM;
8:     deployed = true;
9: else
10:    if ResourceForExtraVM    then
11:            start newVMInstance;
12:            add VMToApplicableVMList;
13:            deploy UserRequest on newVM;
14:            deployed = true;
15:    else
16:            queue UserRequest until
17:            queueTime > waitingTime
18:            deployed = false;
19:    end if
20: end if
21: if deployed    then
22:    return success;
23:    terminate;
24: else
25:    return fail;
26:    terminate;
27: end if
```

As shown in the Schedule Algorithm, the scheduler receives as input the Users' Deployment Requests (UDR) and the application data to be provisioned (line 1 in the schedule algorithm). The output of the scheduler is the confirmation of successful or failure deployment.

In the first step, the user request is extracted, which then forms the basis for finding the VM with appropriate resources for deploying the application. Next, it collects information about the Available Resource (ARS) and the number of running VMs in the data center (line 2). The UDRs are used to find a list of Applicable/Apposite VMs (AVM) capable of provisioning the requested user request (lines 3-4).

When the list of VMs is found, the load-balancer decides on which particular VM to deploy the application in order to balance the load in the data center; in our case the ELB (line 5-8).

In case there is no VM with the appropriate resources running in the data center, the scheduler checks if there is resources consisting of physical resources can host new VMs (lines 9-10). If that is the case, it automatically starts new VMs with predefined resource capacities to provision the user requests (lines 11-14).

When the resources cannot host extra VMs, the scheduler queues the provisioning of service requests until a VM with appropriate resources is available (lines 15-16). If after a certain period of time, the user requests cannot be be scheduled and deployed, the scheduler returns a scheduling failure to the cloud admin, otherwise it returns success (lines 17-27).

7 Test Experience

Based on the figure below Fig. 3, the setup environment for testing is pretty simple at this stage of our work. Using Oracle VM VirtualBox manager we setup a 64-bit Ubuntu 14.04 system and Centos7 system with the following features each: 512MB of memory, 2 processor, 12MB of video memory, 2 network adapters, and 16GB of hard drive space.

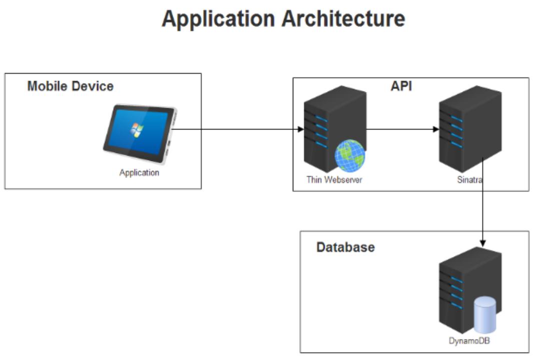
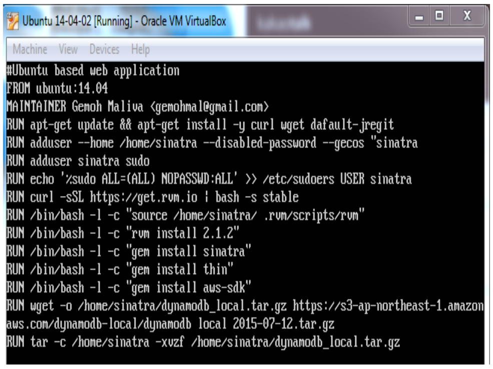


Fig. 3 A Web Application Architecture

First, we define a Docker image for launching a container for running the REST endpoint. I will then use this Docker image to test the code (the source code of a simple productive web application I downloaded from GitHub) on the Centos7 (acting like a laptop in this test environment). Later this created image can be used to test the code in Amazon EC2. The REST endpoints are going to be developed using Ruby and the Sinatra framework, so these will need to be installed in the image. Sinatra is open source software to write web application written in Ruby. We chose Sinatra framework in the test environment because it is an elegant web framework and really tiny. Sinatra is good fit for small scale projects and it does all what other heavy frameworks of the Ruby family such as Rail do. The backend will use Amazon DynamoDB to ensure that the application can be run from both inside and outside AWS web services, the Docker image will include the DynamoDB local database.

The Docker image is created using the DockerFile that contains all the instructions require to build an image. From the file, we will launch containers, install a bunch of software packages using the APT package manager, and then commit those changes to a new Docker image. DockerFile is a more powerful, fast and flexible way of creating Docker images. Here's the DockerFile we created for the app looks like:


Ubuntu 14-04-02 [Running] - Oracle VM VirtualBox
Machine View Devices Help

```
#Ubuntu based web application
FROM ubuntu:14.04
MAINTAINER Gemoh Maliva <gemohmal@gmail.com>
RUN apt-get update && apt-get install -y curl wget dafault-jregit
RUN adduser --home /home/sinatra --disabled-password --gecos "sinatra
RUN adduser sinatra sudo
RUN echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers USER sinatra
RUN curl -sSL https://get.rvm.io | bash -s stable
RUN /bin/bash -l -c "source /home/sinatra/ .rvm/scripts/rvm"
RUN /bin/bash -l -c "rvm install 2.1.2"
RUN /bin/bash -l -c "gem install sinatra"
RUN /bin/bash -l -c "gem install thin"
RUN /bin/bash -l -c "gem install aws-sdk"
RUN wget -o /home/sinatra/dynamodb_local.tar.gz https://s3-ap-northeast-1.amazon
aws.com/dynamodb-local/dynamodb local 2015-07-12.tar.gz
RUN tar -c /home/sinatra -xvzf /home/sinatra/dynamodb_local.tar.gz
```

Fig. 4 Creating an image using the Dockerfile

To build the image from the above DockerFile, I used this command
$ docker build --tag="aws_activate/sinatra:v1"

The tag option sets an identifier on the images and is usually setup as owner/repository:version. This makes it easy to identify what an image contains and who owns it when viewing the images in a registry.
Next I launched a container from this newly created image:
$ docker run -it aws_activate/sinatra:v1 /bin/bash

This command launches the container and goes into a bash shell. I can interact with the container inside just like I would on a Linux server. Because I'm developing a web application, I cloned the image and commit the changes in the running container to a new image using this commit command.

```
$ docker commit -m “ready for testing” b9d03d60ba89 aws_activate/sinatra:v1.1
```

Version 1.1 of the container includes the Sinatra application that will serve up the REST endpoint.

The web application can be run using this command:

```
$ docker run -d -w /home/sinatra –p 10001:4567aws_activate/sinatra:v1.1
./run_app.sh
```

The shell script starts up the local DYnamoDb database in the container and launches the Sinatra application using the thin webserver on port 4567. The web application can be view from the browser using http://localhost:10001/activity/1 and see the following:

```
{"activity_id":"1", "user_id":" db430d35-92a0-49d6-ba79-
0f37ea1b35f7", "type":"meal", "calories":100, "date":"2015-
10-29 15:47:23 +0000"}
```

The endpoint seems to be working properly. The activity record was pulled from the local DynamoDB database and returned as JSON from the Sinatra application code.

8 Conclusion

In this paper, we looked at application optimization and deployment. Based on the challenges in software deployment environment and the numerous advantages of Docker, we proposed a multi-task cloud infrastructure using Docker and AWS services for rapid deployment, application optimization and isolation. We saw that this platform is for building, shipping, and running our applications. We can build any app in any language using any stack, dockerized the app and the app can run anywhere on anything. Furthermore, we saw how Amazon ECS helps solve challenging problems when running multiple container-based applications and services on a shared compute cluster.

For future work, we intend to fully complete the implementation of our proposed cloud platform and scale it up with Amazon EC2 container service for high performance container management. We will then conduct thorough evaluation to demonstrate the flexibility of our platform and then compare it with related existing platform.

Acknowledgments Following are results of a study on the “Leades INdustry-university Cooperation” Project, supported by the Ministry of Education, Science & Technology (MEST)

References

1. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Application **1**, 7–18 (2010)
2. Yang, T.A., Joshy, N., Rojas, E., Anumula, S., Moola, J.: Virtualization and Data Center Design. Global Journal on Technology **9**, 36–54 (2015)
3. Kratzke, N.: Lightweight Virtualization Cluster How to Overcome Cloud Vendor Lock-In. Journal of Computer and Communications **2**, 1–7 (2014)
4. Kratzke, N.: Cloud Computing Costs and Benefits—An IT Management Point of View (2012). In: Ivanov, I., van Sinderen, M., Shiskov, B. (eds.) Cloud Computing and Services Sciences, pp. 185–203. Springer, New York (2014)
5. Merkel, D.: Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal **2** (2014)
6. Caballer, M., Blanquer, I., Molto, G., de Alfonso, C.: Dynamic management of virtual infrastructures. Journal of Grid Computing **13**(1), 53–70 (2014)
7. Binz, T., Breitenbcher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - a runtime for TOSCAbased cloud applications. In: ICSOC. LNCS, vol. 8274, pp. 692–695. Springer (2013)
8. AmazonWebServices. AWSEC2. http://docs.aws.amazon.com/AmazonECS/latest/developerguide/
9. Keahey, K., Freeman, T.: Contextualization: providing one-click virtual clusters. In: Fourth IEEE International Conference on eScience, Indianapolis, Indiana, USA, pp. 301–308 (2008)
10. Bresnahan, J., Freeman, T., LaBissoniere, D., Keahey, K.: Managing appliance launches in infrastructure clouds. In: Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery, TG 2011, vol. 12, pp. 1–12:7. ACM, New York (2011)