

Kubernetes Dockerize Application on Amazon Web Service Using KOPS

ARUN KUMAR K¹, VINAYADITYA B V², VINUTHA B S³

¹Assistant Professor, School of CS & IT, Jain University, Bangalore, Karnataka, India

^{2,3}PG Scholar – School of CS & IT, Jain University, Bangalore, Karnataka, India

Abstract - In this paper we deployed a containerized application on to a Kubernetes cluster managed by KOPS (Kubernetes Operation Tool's). Is a fully managed service that makes it easy to deploy, manage, and scale containerized papers using Kubernetes on AWS. KOPS runs the Kubernetes control plane for you across multiple AWS availability zones to eliminate a single point of failure. In this paper, we will use Amazon Kops to deploy a highly available Kubernetes control plane. We will then configure 'kubectl', an open source command line tool to interact with your Kubernetes infrastructure. Using AWS CloudFormation, you will launch a cluster of worker nodes on Amazon EC2, and then launch a containerized guest book paper onto your cluster.

Keywords: AWS – Amazon Web Services, Docker image, KOPS

1. INTRODUCTION

Containers are a method of operating system virtualization that allows you to run an paper and its dependencies in resource – isolated processes. Containers allow you to easily package an paper's code, configurations, and dependencies into easy to use building blocks that deliver environmental consistency, operational efficiency, developer productivity, and version control. Containers can help ensure that papers deploy quickly, reliably, and consistently regardless of deployment environment. Containers also give you more granular control over resources giving your infrastructure improved efficiency. Running containers in the AWS Cloud allows you to build robust, scalable papers and services by leveraging the benefits of the AWS Cloud such as elasticity, availability, security, and economies of scale. You also pay for only as much resources as you use. Any containerized paper typically consists of multiple containers. There are containers for the paper itself, a database, possibly a web server, and so on. During development, it's normal to build and test this multi-container paper on a single host. This approach works fine during early dev and test cycles but becomes a single point of failure for production, when paper availability is critical.

In such cases, a multi-container paper can be deployed on multiple hosts. Customers may need an external tool to manage such multi-container, multi - host deployments. Container orchestration frameworks provides the capability of cluster management, scheduling containers on different hosts, service discovery and load balancing,

crash recovery, and other related functionalities. There are multiple options for container orchestration on Amazon Web Services: Amazon ECS, Docker for AWS, and DC/OS. Another popular option for container orchestration on AWS is Kubernetes.

2. BACK END LANGUAGE:

In this paper we used PHP programming language because, PHP is a general - purpose scripting language that is especially suited to server-side web development, in which case PHP generally runs on a web server. Any PHP code in a requested file is executed by the PHP runtime, usually to create dynamic web page content or dynamic images used on websites

- i. Version: PHP 7.0
- ii. Java Script with bootstrap
- iii. Back end: kubectl v1.3

2.1 DEPLOYMENT PLATFORM:

For deploying of this application, we used amazon web Services. Amazon web services provide servers on rent to deploy application. Amazon web services is the one of the popular cloud based platform that Provide on-demand cloud computing platforms to Individuals, companies and governments, on a paid Subscription basis.

Platform: amazon web services (EC2 instance – Ubuntu 16.4 servers)

3. EXISTING SYSTEM AND PROBLEM STATEMENT

Before the use of the Docker containers, we were using VM cluster to host our applications. Here are some of the limitations of the same:

Limitations no 1: Implementation and configuration complexity

Limitations no. 2: Update and upgrade factors

Limitations no. 3: Cluster cost factors

PROBLEM STATEMENT:

Before the use of the Docker containers, we were using VM cluster to host our applications. Whenever start to run our application or website on a single EC2 instance and over time, traffic increases to the point that you require

more than one instance to meet the demand. And also it will be like single point of failure. This causes major drawback to run our business.

4. FLOW CHART

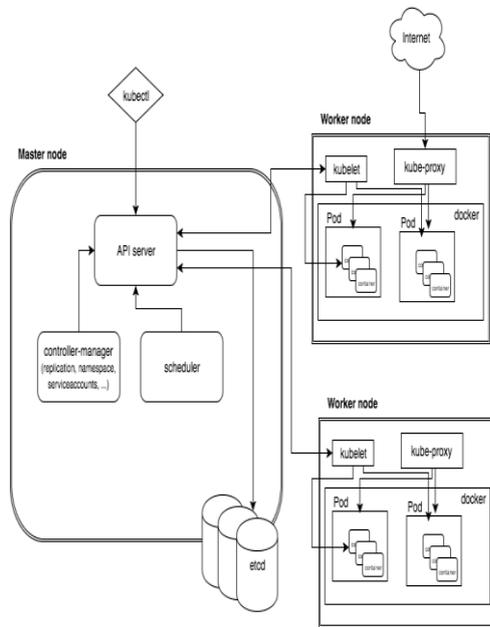


Fig 4.1: Master and Worker nodes API server

The API server is the entry points for all the REST commands used to control the cluster. It processes the REST requests, validates them, and executes the bound business logic. The result state has to be persisted somewhere, and that brings us to the next component of the master node.

4.1 Etcd storage

Etcd is a simple, distributed, consistent key-value store. It's mainly used for shared configuration and service discovery. It provides a REST API for CRUD operations as well as an interface to register watchers on specific nodes, which enables a reliable way to notify the rest of the cluster about configuration changes.

An example of data stored by Kubernetes in etcd is jobs being scheduled, created and deployed, pod/service details and state, namespaces and replication information, etc.

4.2 Scheduler

The deployment of configured pods and services onto the nodes happens thanks to the scheduler component. The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.

4.3 controller-manager

Optionally you can run different kinds of controllers inside the master node. Controller - manager is a daemon embedding those controllers. A controller uses Api server to watch the shared state of the cluster and makes corrective changes to the current state to change it to the desired one. An example of such a controller is the Replication controller, which takes care of the number of pods in the system. The replication factor is configured by the user, and it's the controller's responsibility to recreate a failed pod or remove an extra-scheduled one. Other examples of controllers are endpoints controller, namespace controller, and service accounts controller, but we will not dive into details here.

4.4 Worker node

The pods are run here, so the worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled.

4.4.1 Docker

Docker runs on each of the worker nodes, and runs the configured pods. It takes care of downloading the images and starting the containers.

4.4.2 kubelet

kubelet gets the configuration of a pod from the apiserver and ensures that the described containers are up and running. This is the worker service that's responsible for communicating with the master node. It also communicates with etcd, to get information about services and write the details about newly created ones.

4.4.3 kube-proxy

kube-proxy acts as a network proxy and a load balancer for a service on a single worker node. It takes care of the network routing for TCP and UDP packets.

4.4.4 kubectl

A command line tool to communicate with the API service and send commands to the master node.

5. IMPLEMENTATION:

The implementation involves

- I. create Docker Image
 - Install Docker
 - Install docker.io with this apt command
 - Create Dockerfile

- Edit the 'Dockerfile' with vim

- Save the file and exit

II. Launch an Amazon EC2 Instance

- Launch Instance from console
- Configure your Instance
- Connect to your Instance

III. Enable Auto scaling and Load balancer for High availability

- After launching instance enable auto scaling and load balancer for high availability of server

IV. Containerized Application Support

- Specify dynamic ports in the ECS container task definition
- When a new task is added to the fleet, the ECS schedule auto-assigns it to the ALB using that port
- Share the ALB amongst multiple services using path-based routing
- Improve cost efficiency by running more components of your application per EC2 fleet.

V. Use KOPS commands on Ubuntu instance

- To become root user
- To get node information
- To get deployment information

6. FRONT END SCREEN (SCREENSHOT)

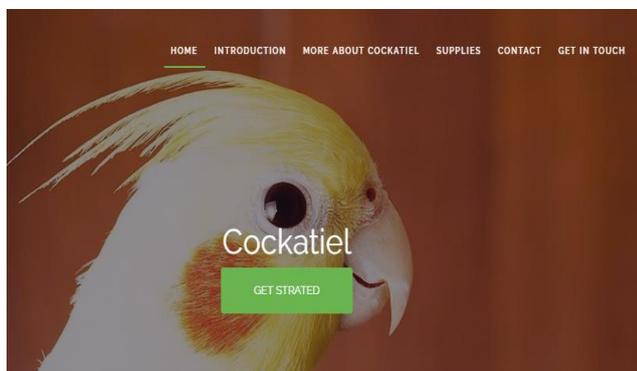


Fig 6.1 WEB Application front Page 1

This page consists of information about the web application Birdlovers.ga where it provides all information about the cockatiel bird.



Fig6.2 WEB application front Page 2

This page speaks about the bird origin where it is from and how to take care of this bird in the captivity.

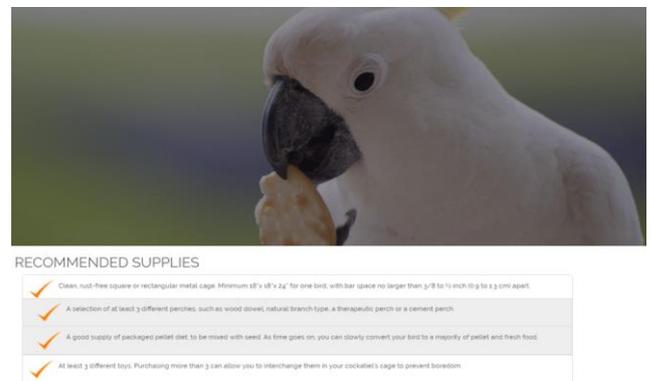


Fig 6.3 WEB Application front Page 3

This page speaks about the life span of this bird and about the food diet of this bird and possible medications for this bird.

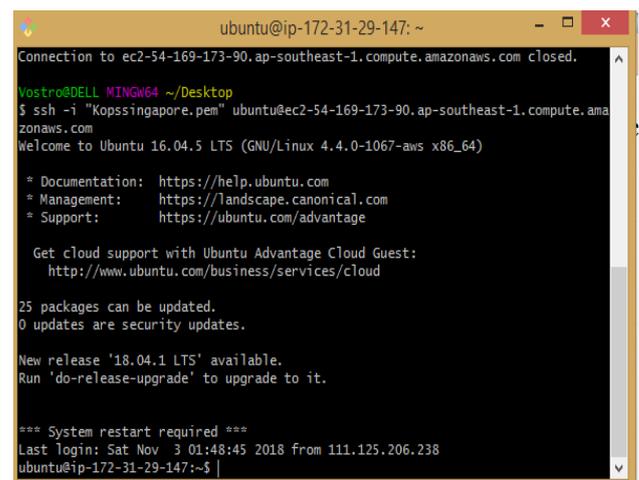
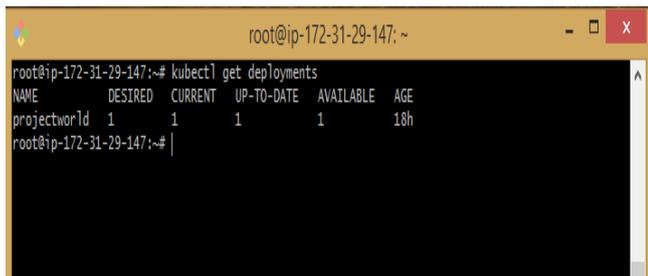


Fig 6.4 Kops server

This screenshot shows all the information about the kops server which is installed on the Ubuntu server machine.



```

root@ip-172-31-29-147:~# kubectl get deployments
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
projectwor1d  1         1         1            1           18h
root@ip-172-31-29-147:~#
    
```

Fig-5 Deployment information

~# kubectl get deployments is a command used to get all the information about the cluster which is deployed in this server machine.

7. CONCLUSIONS AND FUTURE ENHANCEMENT

This project is designed to deploy a containerized application on to a Kubernetes cluster worker node using KOPS. We have created the micro services of our web application using Docker. The web application is running in a container and that container is successfully deployed in a Kubernetes cluster which has a master and a worker node and this web application is successfully deployed and running inside the node.

FUTURE ENHANCEMENT

For future enhancement we suggests to make this work with a multi - container so that they can communicate with each other and host any size huge web application using Docker Networking Technology.

REFERENCES

1. P. Mell and T. Grance, The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-145, 2011.
2. U. Thakrar, "Introducing Right- Scale Cloud Appliance for vSphere," blog, 10 Dec. 2013; www.rightscale.com/blog/enterprisecloudstrategies/introducing-rightscale-cloud-appliance-vsphere.
3. B. Kepes, "VoltDB Puts the Boot into Amazon Web Services, Claims IBM Is Five Times Faster," Forbes, 2014; www.forbes.com/sites/benkepes/2014/08/06/voltdb-puts-the-boot-into-amazon-web-services-claims-ibm-5-faster.
4. J. Petazzoni, "Containers & Docker: How Secure Are They?" blog, 21 Aug. 2013; <http://blog.docker.com/2013/08/containers-docker-how-secure-are-they>.
5. J. Petazzoni, "Linux Containers (LXC), Docker, and Security," 2014; [ww.slideshare.net/jpetazzo/linux-containers-lxc-docker-and-security](http://www.slideshare.net/jpetazzo/linux-containers-lxc-docker-and-security).
6. C. Mcluckie, "Containers, VMs, Kubernetes and VMware," blog, 2014; <http://googlecloudplatform.blogspot.com/2014/08/containers-vm-kubernetes-and-vmware.html>.
7. B. Butler, "Containers: Buzzword du Jour, or Game-Changing Technology?" Network World, 3 Sept. 2014; www.networkworld.com/article/2601925/cloud-computing/container-party-vmware-microsoft-cisco-and-red-hat-all-get-in-on-app-hoopla.html.
8. https://github.com/kubernetes/kops/blob/master/docs/cli/kops_create_secret_encryptionconfig.md
9. <https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/>
10. https://github.com/kubernetes/kops/blob/master/nod-eup/pkg/model/kube_apiserver.go#L61
11. <https://github.com/georgebucknerfield/kops/blob/master/pkg/apis/kops/cluster.go#L162>