Good Morning Gentlemen.. Today We are going to explore some of the features of Kubernetes (which is a powerful open-source container orchestration system originally developed by Google) and We are going to see kubernetes in action on AKS, which is the kubernetes service offered by Azure. First of all lets look into some of great features provided by Kubernetes (though these are also offered by Docker swarm in a way or another, however, docker swarm is basically for smaller setup and has limited functionality as compared to kubernetes). So some of the kubernetes features are:

1. Service discovery and load balancing

That basically means In Kubernetes, Pods are assigned with their own IP addresses and for a set of Pods a single DNS name is offered, which can help to load-balance across them.

If someone who is not familiar with kubernetes, let me tell you, POD is the smallest object in kubernetes which is used to hold a container inside it.

2. Automated rollouts and rollbacks.

It is the default strategy and here pods will be taken down once at a time and corresponding new pod will be rolled out.

By doing so the application remains up all the time and will not cause any downtime in between.

3. Horizonal scaling

It allows to Scale your application up and down with a simple command line or with a UI, or automatically based on resource utilization. That auto scaling feature is unavailable in docker swarm by default.

Next 3 features are self-explanatory that it provides the capabilities for

4. Batch execution by creating Jobs and CronJobs in Kubernetes.

5. Self-Healing

6. And it also provides a nice integrated Dashboard where we can monitor the Kubernetes objects.

Few mins back just to save time, I have setup the AKS cluster using Terraform. Let me do a quick walkthrough with terraform code and resources.

We are going to discuss about 3 prominent features today:

1. Service discovery and load balancing.

2. Horzitonal scaling through Horzitonal Pod Autoscalar.

3. Kubernetes Dashboard.

Initially I'm going to show you load balancing capabilities by using AKS native load balancer just by deploying a sample application.

The first thing is to create an object of type "deployment".

A deployment allows you to create an application, by maintaing details such as:

1. the number of containers

2. selector which can basically select pods based on labels.

3. the image to be used and etc.

Now to enable communication between various components within and outside of an application, we have to deploy another Kubernetes object of type service.

Here, you can see there is a selector, which basically selects pods based on the labels.

The type of the service is loadbalancer, which will give us an external IP and we will able to browse this app in our browser.

At first Let's create the deployment object. It has been created and if we can check, we can see that few more objects has been created.

The first is the deployment itself, the second is the replicaset which basically takes care of ensuring that the required number of pods are always running and finally the pods. Now let’s create the service object of type loadbalancer. It gives us an external IP. let's open that in browser.

Frequently, the page is served by a new pod and it shows efficient distribution of incoming network traffic across a group of Pods.

==========

Now I'm going to show you another coolest feature of kubernetes in AKS cluster, that is, auto scaling.

HPA or Horizontal Pod Autoscaler automatically scales the number of pods based on observed CPU utilization.

I'm going to enable Horizontal Pod Autoscaler for the php-apache server.

=> Run & expose php-apache server:

To demonstrate HPA we will use a custom docker image based on php-apache.

php:5-apache

Inside the image it defines an index.php page which performs some CPU intensive computations:

First, we will start a deployment running the image and expose it as a service on port 80

kubectl create -f php-apache.yaml

kubectl get deployments

=> Create Horizontal Pod Autoscaler:

Now that the server is running, we will create the autoscaler.

HPA will increase and decrease the number of replicas to maintain an average CPU utilization of 50% across all Pods.

kubectl create -f hpa.yaml

kubectl get hpa -w

=> Now To Increase load , We are going to create another deployment of load generator which runs a busybox image.

We will start a container and send an infinite loop of wget queries to the php-apache service.

we will see how the autoscaler reacts to increased load.

kubectl create -f load-generator-deployment.yaml

Within few minutes or so, we should see the higher CPU load by executing:

kubectl get hpa -w (in one terminal)

kubectl get po -w (in second terminal)

=>Delete Load

Now We will scale down the number of replicas, there by deleting the load-generator.

kubectl delete deployment load-generator

kubectl get deployment php-apache

Last but not the least, lets have a look at kubernetes Dashboard.