Azure Kubernetes Service

Pods (Basics reference [here](https://azure.microsoft.com/mediahandler/files/resourcefiles/phippy-goes-to-the-zoo/Phippy%20Goes%20To%20The%20Zoo_MSFTonline.pdf))

* Pods are basic unit for running containers inside of Kubernetes.
* A pod provides a way to set environment variables, mount storage, and feed other information into a container.
* In Kubernetes, Pods are responsible for running your containers. Every Pod holds at least one container, and controls the execution of that container. When the containers exit, the Pod dies too.

Replica sets

* Replica Sets are considered as low-level type of Kubernetes. Often Kubernetes opt for higher level abstractions like deployments and daemon sets.
* A Replica Set ensures that a set of identically configured Pods are running at the desired replica count. If a Pod drops off, the Replica Set brings a new one online as a replacement.

Secrets

* Secrets are base 64 encoded “at rest” but the data is automatically decoded when attached to a pod.
* Secretes are attached as files or environment variables.
* Use add-on encryption providers for locking your data.
* Secrets are used to store non-public information, such as tokens, certificates, or passwords. Secrets can be attached to Pods at runtime so that sensitive configuration data can be stored securely in the cluster.

Deployments

* Deployments supports rolling updates and rollbacks. Rollouts can even be paused.
* A Deployment is a higher-order abstraction that controls deploying and maintaining a set of Pods. Behind the scenes, it uses a Replica Set to keep the Pods running, but it offers sophisticated logic for deploying, updating, and scaling a set of Pods within a cluster.

Daemon Sets

* DaemonSets provide a way to ensure that a copy of a Pod is running on every node in the cluster. As a cluster grows and shrinks, the DaemonSet spreads these specially labeled Pods across all of the nodes.

Ingress

* Ingresses provide a way to declare that traffic ought to be channeled from the outside of the cluster into destination points within the cluster. One single external Ingress point can accept traffic destined to many different internal services.

Cron Jobs

* CronJobs provide a method for scheduling the execution of Pods. They are excellent for running periodic tasks like backups, reports, and automated tests.

CRD

* CustomResourceDefinitions, or CRDs, provide an extension mechanism that cluster operators and developers can use to create their own resource types.

Azure Kubernetes Basic Setup in Windows.

Limitations

The following limitations apply when you create and manage AKS clusters that support multiple node pools:

* You can't delete the first node pool.

The following additional limitations apply to Windows Server node pools:

* The AKS cluster can have a maximum of 10 node pools.
* The AKS cluster can have a maximum of 100 nodes in each node pool.
* The Windows Server node pool name has a limit of 6 characters.

1. Install Azure cli from [here](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest)

Or

Using Power shell command

Invoke-WebRequest -Uri https://aka.ms/installazurecliwindows -OutFile .\AzureCLI.msi; Start-Process msiexec.exe -Wait -ArgumentList '/I AzureCLI.msi /quiet'; rm .\AzureCLI.msi

1. Install aks-preview cli extension.

# Install the aks-preview extension

az extension add --name aks-preview

# Update the extension to make sure you have the latest version installed

az extension update --name aks-preview

1. Create a resource group.

az group create --name <resource group name> --location eastus

1. Create AKS Cluster. (adds linux node pool by default)

az aks create --resource-group <resource group name> --name <cluster name> --node-count 1 --enable-addons monitoring --kubernetes-version 1.16.7 --generate-ssh-keys --enable-vmss --vm-set-type VirtualMachineScaleSets --load-balancer-sku standard --network-plugin azure

1. Add windows cluster node pool.

az aks nodepool add --resource-group <resource group name> --cluster-name <cluster name> --os-type Windows --name npwin --node-count 1 --kubernetes-version 1.16.7

1. Connect to AKS Cluster by locally installing kubectl.

az aks install-cli

1. Add kubectl path from control panel by going to control panel ->system ->advanced settings and add path “C:\Users\Administrator\.azure-kubectl”
2. Download credentials to aks.

az aks get-credentials --resource-group myResourceGroup --name myAKSCluster

1. Verify nodes.

kubectl get nodes.

1. Running Containers/pods on nodes Eg: aspnet sample application
2. Create sample.yaml file and place below code into it.

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample

labels:

app: sample

spec:

replicas: 1

template:

metadata:

name: sample

labels:

app: sample

spec:

nodeSelector:

"beta.kubernetes.io/os": windows

containers:

- name: sample

image: mcr.microsoft.com/dotnet/framework/samples:aspnetapp

resources:

limits:

cpu: 1

memory: 800M

requests:

cpu: .1

memory: 300M

ports:

- containerPort: 80

selector:

matchLabels:

app: sample

---

apiVersion: v1

kind: Service

metadata:

name: sample

spec:

type: LoadBalancer

ports:

- protocol: TCP

port: 80

selector:

app: sample

1. Run Yaml file using

kubectl apply -f sample.yaml

1. Run service using

kubectl get service sample –watch.(This will give external IP copy that)

1. Check the status of pods.

Kubectl get pods. (If running Open IE in next step)

1. Open IE and type

http://<externalIP>:port (80)

1. Delete resource group using.

az group delete --name <resource group name> --yes --no-wait

Azure Kubernetes with Akure containers Registry in Windows

1. Execute first 6 steps from above scenario.
2. Create ACR (azure container registry) using command

az acr create --resource-group <group name> --name <registry name> --sku Standard --location eastus

1. Copy application docker file to c:\<application folder> (locally) Eg : Nats application (c:\nats)
2. Build and push image to ACR using.

az acr build --registry <registry name> --image caddy:v1 c:\caddy --platform=windows

Note: default platform is Linux

1. Integrate ACR with aks cluster using.

az aks update -n <aks name> -g <resource group name> --attach-acr <acrName>

1. Execute 7,8,9 steps from above.
2. Running Containers/pods on nodes Eg: caddy application
3. Create caddy.yaml file and place below code into it.

apiVersion: apps/v1

kind: Deployment

metadata:

name: caddy

labels:

app: caddy

spec:

replicas: 1

template:

metadata:

name: caddy

labels:

app: caddy

spec:

nodeSelector:

"beta.kubernetes.io/os": windows

containers:

- name: caddy

image: containerregistrysupmek.azurecr.io/caddy:v1

resources:

limits:

cpu: 1

memory: 800M

requests:

cpu: .1

memory: 300M

ports:

- containerPort: 80

selector:

matchLabels:

app: caddy

---

apiVersion: v1

kind: Service

metadata:

name: caddy

spec:

type: LoadBalancer

ports:

- protocol: TCP

port: 80

selector:

app: caddy

1. Run Yaml file using

kubectl apply -f caddy.yaml

1. Run service using

kubectl get service caddy –watch.(This will give external IP copy that)

1. Check the status of pods.

Kubectl get pods. (If running Open IE in next step)

1. Open IE and type

http://<externalIP>:port(80)

1. Delete resource group using.

az group delete --name <resource group name> --yes --no-wait

Multiple Containers Running in Azure Kubernetes.

1. Execute first 6 steps from above scenario.
2. Create ACR (azure container registry) using command

az acr create --resource-group <group name> --name <registry name> --sku Standard --location eastus

1. Copy application docker file to c:\<application folder> (locally) Eg : Nats application (c:\nats)
2. Build and push 2 or more images to ACR using.

az acr build --registry <registry name> --image caddy:v1 c:\caddy --platform=windows

az acr build --registry <registry name> --image nats:v1 c:\nats --platform=windows

Note: default platform is Linux

1. Integrate ACR with aks cluster using.

az aks update -n <aks name> -g <resource group name> --attach-acr <acrName>

1. Execute 7,8,9 steps from above.
2. Running Containers/pods on nodes Eg: caddy and nats application
3. Create multi.yaml file and place below code into it.

apiVersion: apps/v1

kind: Deployment

metadata:

name: caddy

labels:

app: caddy

spec:

replicas: 1

template:

metadata:

name: caddy

labels:

app: caddy

spec:

nodeSelector:

"beta.kubernetes.io/os": windows

containers:

- name: caddy

image: containerregistrysupmek.azurecr.io/caddy:v1

resources:

limits:

cpu: 1

memory: 800M

requests:

cpu: .1

memory: 300M

ports:

- containerPort: 80

selector:

matchLabels:

app: caddy

---

apiVersion: v1

kind: Service

metadata:

name: caddy

spec:

type: LoadBalancer

ports:

- protocol: TCP

port: 80

selector:

app: caddy

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nats

labels:

app: nats

spec:

replicas: 1

template:

metadata:

name: nats

labels:

app: nats

spec:

nodeSelector:

"beta.kubernetes.io/os": windows

containers:

- name: nats

image: containerregistrysupmek.azurecr.io/nats:v1

resources:

limits:

cpu: 1

memory: 800M

requests:

cpu: .1

memory: 300M

ports:

- containerPort: 8222

selector:

matchLabels:

app: nats

---

apiVersion: v1

kind: Service

metadata:

name: nats

spec:

type: LoadBalancer

ports:

- protocol: TCP

port: 8222

selector:

app: nats

1. Run Yaml file using

kubectl apply -f multi.yaml

1. Run service using

kubectl get service caddy –watch.(This will give external IP copy that)

kubectl get service nats –watch

1. Check the status of pods.

Kubectl get pods. (If running Open IE in next step)

1. Open IE and type

http://<externalIP>:port(80) for caddy

http://<externalIP>:port(8222) for nats

1. Delete resource group using.

az group delete --name <resource group name> --yes --no-wait

Scaling.

1. Scaling pods manually Eg: caddy deployment.

kubectl scale --replicas=5 deployment.apps/caddy

1. Auto scaling pods.

kubectl autoscale deployment caddy --cpu-percent=50 --min=3 --max=10

1. Scale Nodes.

az aks scale --resource-group <resource group name> --name <aks cluster name> --nodepool-name npwin --node-count 3.

1. Scaling pods using yaml files.(scale.yaml)

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: caddy-hpa

spec:

maxReplicas: 10 # define max replica count

minReplicas: 3 # define min replica count

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: caddy

targetCPUUtilizationPercentage: 50 # target CPU utilization

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: nats-hpa

spec:

maxReplicas: 10 # define max replica count

minReplicas: 3 # define min replica count

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: nats

targetCPUUtilizationPercentage: 50 # target CPU utilization

kubectl apply -f scale.yaml

kubectl get hpa

Upgrade the application.

1. Build nats application by changing version in dockerfile.

az acr build --registry containerregistrysupmek --image nats:v2 c:\nats --platform=windows

1. Set deployment to latest version.

kubectl set image deployment nats nats=<aks name>.azurecr.io/nats:v2

1. Kubectl get pods. (this will show process of terminating old pod and creating new pod version)

Upgrade the Cluster.

1. Get available cluster versions using.

az aks get-upgrades --resource-group <resource group> --name <aks name> --output table

1. Upgrade a cluster using.

az aks upgrade --resource-group <resource group> --name <aks name> --kubernetes-version 1.15.5.

1. Validate updated cluster using.

az aks show --resource-group <resource group> --name <aks name> --output table