**A small Azure and AKS challenge to facilitate the discussion**

Dear Candidate,

We want to have an open and honest discussion with you to assess your level of resourcefulness and flexibility when it comes to DevOps and cloud, and in order to facilitate that discussion we have thought of a small challenge for you…

The challenge will be to deploy a K8s cluster from scratch in Azure, using the AKS service.

The files used are all in [git@github.com:gabrielbcn/test.git](mailto:git@github.com:gabrielbcn/test.git) which you can clone. Since we’re giving you this document in advance it is fine if you get it done first and just comment it to us. If you prefer, we will give you time to do it and check back with you later.

This exercise is not about replicating the steps but about discussing the process, showing understanding and discussing possible ways of improving it.

In order to deploy the k8s cluster, you will be given an Azure account

Username: XXXXXXX

Password: XXXXXXX

You should use cloud shell, and, once in cloud shell, you should use automation to deploy a cluster.

You can choose other ways of automating it, and if you’re more comfortable with other tools, by all means use them, but we’re proposing you use the following ansible playbook:

# ===========

# This playbook first creates an Azure Kubernetes Service cluster of 2 nodes with provided username and ssh key and scales it to 3 nodes.

# Change variables below to customize your AKS deployment.

# It also requires a valid service principal to create AKS cluster, so fill:

# - client\_id

# - client\_secret

# This sample requires Ansible 2.6

- name: Create Azure Kubernetes Service

hosts: localhost

connection: local

vars:

ssh\_key: ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDH14vPP1NE+qc3SgpoSPYMVaPLj8ouA1azujnMPbVKtN8mSg0KBWVWebx+GLqGI9nrKwmVwPHVzM8LhbbpYhfnN0UA2OV7sxBIshy7BshDTewiLyR9FXuNjuf8Cwd7WI+vgkKO0Yh6EdgwZ2hz+yVTn+5umCRDOg0ht0i2kyPMn1MHv61aN3noYp5zV8W3Tg8oJ2DPei6FxcNheRWkyvjan73/rISQJYWURlI90b3MCQOTXuI2w6BIxF3kHIHLUtzD7FqQhZdEkM/8cA9wUDgnS6+3ySkGOuQazkEWYn60Ll/597u2fLsq6P/pOCM1+KWLf2kY6mTl/dNxiIdV1d01

resource\_group: "{{ resource\_group\_name }}"

location: eastus

aks\_name: myAKSCluster

username: azureuser

client\_id: "{{ lookup('env', 'AZURE\_CLIENT\_ID') }}"

client\_secret: "{{ lookup('env', 'AZURE\_CLIENT\_SECRET') }}"

tasks:

- name: Create resource group

azure\_rm\_resourcegroup:

name: "{{ resource\_group }}"

location: "{{ location }}"

- name: Create a managed Azure Container Services (AKS) cluster

azure\_rm\_aks:

name: "{{ aks\_name }}"

location: "{{ location }}"

resource\_group: "{{ resource\_group }}"

dns\_prefix: "{{ aks\_name }}"

linux\_profile:

admin\_username: "{{ username }}"

ssh\_key: "{{ ssh\_key }}"

service\_principal:

client\_id: "{{ client\_id }}"

client\_secret: "{{ client\_secret }}"

agent\_pool\_profiles:

- name: default

count: 2

vm\_size: Standard\_D2\_v2

tags:

Environment: Production

- name: Scale created Azure Container Services (AKS) cluster

azure\_rm\_aks:

name: "{{ aks\_name }}"

location: "{{ location }}"

resource\_group: "{{ resource\_group }}"

dns\_prefix: "{{ aks\_name }}"

linux\_profile:

admin\_username: "{{ username }}"

ssh\_key: "{{ ssh\_key }}"

service\_principal:

client\_id: "{{ client\_id }}"

client\_secret: "{{ client\_secret }}"

agent\_pool\_profiles:

- name: default

count: 3

vm\_size: Standard\_D2\_v2

For client ID and secret, you should use the following:

Client ID: XXXX

Secret: XXXX

We don’t want this cluster to be created as it is in the script by default.

* Our applications are memory intensive, we are not sure we have chosen the right kind of machine.
* We are in Singapore, and we are not sure we have the proper region for the cluster.
* We are not sure what the value of the RSA key is, what’s your recommendation?
* We would also like to choose proper names for the cluster and the group of resources, these should be based on your username
* We are not happy with a cluster that has a fixed number of nodes: we would like the cluster to autoscale, with a minimum of two.
* We would like to build more resilience to our cluster and we’re thinking of availability zones.

In order to get the file mentioned, you can use:

git clone <https://github.com/Azure-Samples/ansible-playbooks.git>

You will be able to execute it as:

ansible-playbook aks\_create\_scale.yml

But first please remember to change it to our needs and later on to get credentials

az aks get-credentials --resource-group yourResourceGroup --name yourAKSname

And you should be able to see the nodes created. We leave this to you.

Once we have a Kubernetes cluster we need to give it some content. An example would be a service that would return some useful information. In this case we’re giving you a sample app in go that does a few things:

package main

import (

"fmt"

"log"

"math/rand"

"net"

"net/http"

"os"

"time"

)

const defaultAddr = ":8080"

*// main starts an http server on the $PORT environment variable.*

func main() {

addr := defaultAddr

*// $PORT environment variable is provided in the Kubernetes deployment.*

if p := os.Getenv("PORT"); p != "" {

addr = ":" + p

}

log.Printf("server starting to listen on %s", addr)

log.Printf("http://localhost%s", addr)

log.Printf("http://localhost%s/test", addr)

log.Printf("http://localhost%s/ip", addr)

http.HandleFunc("/", home)

http.HandleFunc("/ip/", getip)

http.HandleFunc("/test/", test)

if err := http.ListenAndServe(addr, nil); err != nil {

log.Fatalf("server listen error: %+v", err)

}

}

*// home logs the received request and returns a simple response.*

func home(w http.ResponseWriter, r \*http.Request) {

log.Printf("received request main: %s %s", r.Method, r.URL.Path)

rand.Seed(time.Now().UnixNano())

answers := []string{

"It is certain",

"It is decidedly so",

"Without a doubt",

"Yes definitely",

"You may rely on it",

"As I see it yes",

"Most likely",

"Outlook good",

"Yes",

"Signs point to yes",

"Reply hazy try again",

"Ask again later",

"Better not tell you now",

"Cannot predict now",

"Concentrate and ask again",

"Don't count on it",

"My reply is no",

"My sources say no",

"Outlook not so good",

"Very doubtful",

}

fmt.Fprintf(w, "Magic 8-Ball says:", answers[rand.Intn(len(answers))])

}

func getip(w http.ResponseWriter, r \*http.Request) {

addrs, err := net.InterfaceAddrs()

if err != nil {

log.Printf("Oops: " + err.Error() + "\n")

}

for \_, a := range addrs {

if ipnet, ok := a.(\*net.IPNet); ok && !ipnet.IP.IsLoopback() {

if ipnet.IP.To4() != nil {

fmt.Fprintf(w, ipnet.IP.String()+"\n")

}

}

}

}

func test(w http.ResponseWriter, r \*http.Request) {

log.Printf("received request test: %s %s", r.Method, r.URL.Path)

fmt.Fprintf(w, "The test page")

}

You should create a Dockerfile that gets a container made from this code, here you have a sample that works:

FROM golang:latest

WORKDIR /app

COPY ./ /app

RUN go run thalestest.go

Time to build the container:

docker build . -t thalestest

Which will start executing instantaneously from cloud shell. You will be able to interact with it with the proper controls. Image should be tagged as thalestest, make sure it has been done.

Now create a deployment, example of yaml in the repo.

apiVersion: apps/v1 *# for versions before 1.9.0 use apps/v1beta2*

kind: Deployment

metadata:

name: frontend

labels:

app: guestbook

spec:

selector:

matchLabels:

app: guestbook

tier: frontend

replicas: 3

template:

metadata:

labels:

app: guestbook

tier: frontend

spec:

containers:

- name: php-redis

image: gcr.io/google-samples/gb-frontend:v4

resources:

requests:

cpu: 100m

memory: 100Mi

env:

- name: GET\_HOSTS\_FROM

value: dns

*# Using `GET\_HOSTS\_FROM=dns` requires your cluster to*

*# provide a dns service. As of Kubernetes 1.3, DNS is a built-in*

*# service launched automatically. However, if the cluster you are using*

*# does not have a built-in DNS service, you can instead*

*# access an environment variable to find the master*

*# service's host. To do so, comment out the 'value: dns' line above, and*

*# uncomment the line below:*

*# value: env*

ports:

- containerPort: 80

Create a service, two nodes min. Expose through load balancer.

apiVersion: v1

kind: Service

metadata:

name: frontend

labels:

app: guestbook

tier: frontend

spec:

*# comment or delete the following line if you want to use a LoadBalancer*

type: NodePort

*# if your cluster supports it, uncomment the following to automatically create*

*# an external load-balanced IP for the frontend service.*

*# type: LoadBalancer*

ports:

- port: 80

selector:

app: guestbook

tier: frontend

Check links and get the ip’s (two), do you get the same answer? Etc…

If you have a problem: check that the ports are right. If you open an interactive session with the container in the node it will tell you to which port is the http server attached. You must properly map this to the externally exposed port. Any will do, no restrictions.