Week 5 lab

Ansible and Kubernetes Autoscaling

1. Introduction

For this assignment, you will learn how to use Ansible to setup and configure software on multiple remote hosts. You will create a playbook and use Ansible to set up and configure a Kubernetes cluster on top of a set of VMs provisioned in Amazon. You will also use the deployed Kubernetes cluster to practise the service autoscaling.

1.1 Reporting

At the end of this assignment, you (individually) should:

- create playbooks, using the information from this tutorial, that will:
  - Install and configure a Kubernetes cluster;
  - Install and configure a Kubernetes Metrics Server;
- write a short report (min 2 pages, not more than 6 pages):
  - Report the results of the following tasks, e.g., screenshots for each of the steps and/or performance measurement etc.
    - Install and configure a Kubernetes cluster;
    - Stress test a simple Nginx server with and without autoscaling and report and present the results;
  - Discuss how Ansible can be used during DevOps lifecycle, e.g., which stages? What are the advantages, alternatives of Ansible?
  - Discuss what are the benefits of auto-scaling in Cloud applications, and in DevOps? Based on the experiments, discuss in what other scenarios can autoscaling be used?

1.2. Assessment

If your Ansible files perform the steps defined above and you have performed the stress test you will receive 80% Your report will determine the rest of 20%.

To be given a grade, you must submit the following:

- Written report (see above for details)
- Ansible Playbooks (git link or zip archive)

1.3. Technologies Overview

- Ansible: Provisioning, configuration management, and application-deployment, https://www.ansible.com/
- Kubernetes: Container-orchestration, https://kubernetes.io/

2. Ansible

Ansible uses the following terms:

- Controller Machine: the machine where Ansible is installed. It manages the execution of the Playbook. It can be installed on your laptop or any machine on the internet
- Inventory: provides a complete list of all the target machines on which various modules are run by making an ssh connection and install the necessary software’s
- Playbook: consists of steps that the control machine will perform on the servers defined in the inventory file
- Task: a block that defines a single procedure to be executed, e.g., install a package
- Module: the main building blocks of Ansible and are reusable scripts that are used by Ansible playbooks. Ansible comes with many reusable modules. These include functionality for controlling services, software package installation, working with files and directories etc.
- Role: a way for organizing playbooks and other files to facilitate sharing and reusing portions of a provisioning
- Facts: global variables containing information about the system, like network interfaces or operating system
- Handlers: used to trigger service status changes, like restarting or stopping a service

3. Preparation

3.1 Install Ansible

For Linux and macOS follow these instructions:

- https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html For windows you can use subsystem:
- https://docs.ansible.com/ansible/latest/user_guide/windows_faq.html#can-ansible-run-on-windows

Or Cygwin:

- https://everythingshouldbevirtual.com/automation/ansible-using-ansible-on-windows-via-cygwin/ You can always use a VM from EC2 or on your own machine

4. Lab assignments of week 5

Make sure Ansible is working by executing the following command:

```
ansible all --inventory "localhost," --module-name debug --args
"msg='Hello'"
```

Here is a break down of Ansible the command:

- all: this means do run the module on all machines that are listed in the "inventory" file, which is the next part of the command
- --inventory "localhost,": The inventory is where all details of the machines are listed such as IP addresses, usernames, etc. In this case, we only use our local computer. This may also be a file
- --module-name debug: Specify which module to use. In this case the "debug" module, prints statements during execution and can be useful for debugging variables
- --args "msg='Hello'": Part of the debug module. In this case 'Hello' is the customized massage that is printed. If omitted, prints a generic message.

4.1 Controlling Hosts

Start 3 t2.micro Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-0080e4c5bc078760e VMs using EC2. Make sure that port 22 is open (see security groups)

Create a text file named 'aws_hosts1' that looks like this:

```
[aws]

[aws:vars]
ansible_ssh_user=ec2-user
ansible_ssh_private_key_file=/home/<USER>/vms.pem
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

Note that the ansible_ssh_private_key_file has to correspond to the key that you use to ssh to the provisioned VMs.

After all theVMs have started, get their public IP (DNS) and add them under the [aws] heading in the file above. So the file may look like this:

```
[aws]
ec2-xx-xx-xx-xx.compute-1.amazonaws.com
ec2-xx-xx-xx-xx.compute-1.amazonaws.com
ec2-xx-xx-xx-xx.compute-1.amazonaws.com

[aws:vars]
ansible_ssh_user=ec2-user
ansible_ssh_private_key_file=$HOME/vms.pem
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

You will notice that this file has some heading in brackets [aws] and [aws:vars]. The first heading in brackets is a group name. You can have more than one group name, which is used to classify systems and decide what systems you are controlling at what times and for what purpose. So, in this case, we only have specified [aws] group. To assign variables to hosts, you can use the [aws:vars] group variables. In this case, we set the VMs username and the location of the key. For more information on inventories see here:
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

Next, run:

```
ansible aws --inventory aws_hosts1 -m setup
```

```
Note: after entering “ansible aws --inventory aws_host1 -m setup”, it
requests “Enter passphrase for key” of each compute node in the [aws]
bracket. However, users usually didn’t have that, an old passphrase when
using the keypair that was created by the AWS platform. That will lead this
command not to be successfully executed.  To solve this issue, it is
suggested to re-create a key-pair with an enter passphrase, and then
replace the original keypair. After fixing it, re-run the above command.
```

The setup module will gather information about the target machines.

The output should be similar to this:

```
ec2-xx-xx-xx-xx.compute-1.amazonaws.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.31.86.189"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::104f:5aff:fedb:8ee"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "08/24/2006",
        "ansible_bios_version": "4.2.amazon",
        "ansible_cmdline": {
            "console": "ttyS0",
            "nvme_core.io_timeout": "4294967295",
            "root": "LABEL=/",
            "selinux": "0"

......
}
```

Terminate all VMs from the EC2 console. Start 2 t2.micro Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0ac019f4fcb7cb7e6 VMs using EC2.

Change your aws_hosts1 to reflect the new VMs and run the setup module again:

```
ansible aws --inventory aws_hosts1 -m setup
```

This time the command should fail. To get more information on what Ansible is doing, run the same command with verbose enabled:

```
ansible aws --inventory aws_hosts1 -m setup -vvv
```

The output should give a hint on what’s wrong:

```
ansible 2.7.7
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/user/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.15rc1 (default, Nov 12 2018, 14:31:15) [GCC 7.3.0]
Using /etc/ansible/ansible.cfg as config file
/home/user/aws_hosts1 did not meet host_list requirements, check plugin
documentation if this is unexpected
/home/user/aws_hosts1 did not meet script requirements, check plugin
documentation if this is unexpected
Parsed /home/user/aws_hosts1 inventory source with ini plugin
META: ran handlers
<ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com> ESTABLISH SSH CONNECTION FOR
USER: ec2-user
<ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com> SSH: EXEC ssh -C -o
ControlMaster=auto -o ControlPersist=60s -o
```

```
'IdentityFile="/home/user/k8.pem"' -o KbdInteractiveAuthentication=no -o
PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -
o PasswordAuthentication=no -o User=ec2-user -o ConnectTimeout=10 -o
StrictHostKeyChecking=no -o ControlPath=/home/user/.ansible/cp/6fc6074a80
ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com '/bin/sh -c '"'"'echo ~ec2-user
&& sleep 0'"'"''
<ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com> ESTABLISH SSH CONNECTION FOR
USER: ec2-user
<ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com> SSH: EXEC ssh -C -o
ControlMaster=auto -o ControlPersist=60s -o
'IdentityFile="/home/user/k8.pem"' -o KbdInteractiveAuthentication=no -o
PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey -
o PasswordAuthentication=no -o User=ec2-user -o ConnectTimeout=10 -o
StrictHostKeyChecking=no -o ControlPath=/home/user/.ansible/cp/8a14c3c900
ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com '/bin/sh -c '"'"'echo ~ec2-user
&& sleep 0'"'"''
<ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com> (255, '', 'ec2-user@ec2-xx-xx-
xxx-xxx.compute-1.amazonaws.com: Permission denied (publickey).\r\n')
<ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com> (255, '', 'ec2-user@ec2-xx-
xxx-xxx-xxx.compute-1.amazonaws.com: Permission denied (publickey).\r\n')
```

If you notice the line *<ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com> ESTABLISH SSH CONNECTION FOR USER: ec2-user You will see that Ansible tries to connect using the username: ec2-user. However, the Ubuntu instances use ubuntu as username. To fix this, we should change the inventory file, but we want to be more flexible so we should create two separate heading to include both Amazon Linux and Ubuntu.

```
[aws-ubuntu]
ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com
Ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com

[aws-amazon]


[aws:children]
aws-ubuntu
aws-amazon


[aws:vars]
ansible_ssh_private_key_file=/home/<USER>/k8.pem
ansible_ssh_common_args='-o StrictHostKeyChecking=no'

[aws-amazon:vars]
asible_ssh_user=ec2-user

[aws-ubuntu:vars]
ansible_ssh_user=ubuntu
```

As you can see we now use 3 different variables: [aws:vars] concerns all hosts under [aws]. [aws-amazon:vars] is about the Linux amazon instances i.e. [aws-amazon] and [aws-ubuntu:vars] is for [aws-ubuntu]. However, there is no [aws] in this file. The heading [aws:children] has been set as the parent of both [aws-ubuntu] and [aws-amazon] so both are now referred to as [aws].

Even if we replace the ansible_ssh_user and run this command again, it will also fail. Ansible works by connecting to the hosts over SSH and pushing out scripts called "Ansible Modules". Ansible then executes and removes them when finished. These modules are simple Python scripts and Ansible is agent-less, so the target hosts only require an SSH connection and Python installed. Therefore, Ansible requires SSH server and Python on every host. However, the Ubuntu instances in Amazon don't have python installed. You must install python on each VM.

On each VM type:

```
sudo apt install python
```

Then retry:

```
ansible aws --inventory aws_hosts1 -m setup
```

This time it should work without any problems. We can now terminate the VM's.

4.2 Using Playbooks

Ansible Playbooks are like a to-do list for Ansible that contains a list of tasks. They are written in YAML format and run sequentially.

4.3 Playbook Structure

Each playbook is an aggregation of one or more plays, and there can be more than one play inside a playbook. A play maps a set of instructions defined against a particular host.

4.4 Create a Playbook

Start 2 t2.micro Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0ac019f4fcb7cb7e6

VMs using EC2 and make sure that ports 22 and 80 are open.

After all VMs have started, get their public IP (DNS) and add them and update the aws_hosts1.

Create a playbook that will install in both VMs java:

```
---
- hosts: all
  become: true
  gather_facts: False
  tasks:

    - name: Bootstrap a host without python2 installed
      raw: test -e /usr/bin/python || (apt -y update && apt install -y
python)

    - name: Update apt-cache
      apt: update_cache=yes

    - name: Install openjdk-11-jdk
      apt: name=openjdk-11-jdk state=latest
```

Execute the play book:

```
ansible-playbook -i aws_hosts1 playbook_example1.yml
```

Based on the previous section this command should fail since Ubuntu instances in EC2 don't have python installed. If you notice the first task, use the module 'raw'. This module directly executes commands via ssh. Since we don’t have python installed yet, we need to disable 'gather_facts'. More information on the module can be found here: https://docs.ansible.com/ansible/latest/modules/raw_module.html

4.4.1 Execute plays on different hosts

If we want to execute different plays on different hosts, for example, we need to install Apache on one host and MySQL on another we need to specify that in the playbook.

Having the following inventory:

```
[web-server]
ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com

[db]
ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com

[cluster:children]
web-server
db

[cluster:vars]
ansible_ssh_private_key_file=$HOME/k8.pem
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
ansible_ssh_user=ubuntu
```

We will use the following playbook:

```
---
- hosts: web-server
  become: true
  tasks:
      - name: Install apache2
        apt: name=apache2 state=latest

- hosts: db
  become: true
  tasks:
      - name: Install mysql
        apt: name=mysql-server state=latest
```

And execute:

```
ansible-playbook -i aws_hosts1 playbook_example2.yml
```

If we open a browser to [web-server] we should see Apache running.

4.4.2 Pass Variables Between Plays

Sometimes it is necessary to pass variables between plays. Consider the following playbook:

```
---
- hosts: web-server
  tasks:
     - name: generate secret
       shell: date +%s | sha256sum | base64 | head -c 32 ; echo
       register: command_output

     - name:
       debug:
         msg: "Secret password is {{ command_output.stdout }}"

- hosts: db
  tasks:
     - name: print paswd
       debug:
         msg: "Secret password is {{ command_output.stdout }}"
```

The play with the ‘db’ hosts will fail. The variable stored on one play is not visible on the next. Instead, we need to use 'hostvars':

```
---
- hosts: web-server
  tasks:
    - name: generate secret
      shell: date +%s | sha256sum | base64 | head -c 32 ; echo
      register: command_output

    - name:
      debug:
        msg: "Secret password is {{ command_output.stdout }}"

    - name: Add command_output to dummy host
      add_host:
        name: "command_output_holder"
        paswd: "{{ command_output.stdout }}"

    - name:
      debug:
        msg: "Secret password is
{{ hostvars['command_output_holder']['paswd'] }}"

- hosts: db
  tasks:
    - name: print paswd
      debug:
        msg: "paswd is {{ hostvars['command_output_holder']['paswd'] }}"
```

If we execute this playbook, we’ll see that the variable is now available to the 'db' hosts as well. More information about variables and 'hostvars' can be found here: https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#accessing-information-about-other-hosts-with-magic-variables.

5. Exercises

5.1 Ansible

Create a playbook that will:

- Install and configure a Kubernetes cluster

Process

Write the Kubernetes setup playbook on one file with the following plays/tasks:

- For all VMs of the cluster. For all this play you will have to execute all commands as root by using become: yes
    - Use the raw module to install python. Use module: https://docs.ansible.com/ansible/latest/modules/raw_module.html
    - Update the repositories and Install the packages docker.io, apt-transport-https. Use module:

    https://docs.ansible.com/ansible/latest/modules/apt_module.html?highlight=apt
    - Add the https://packages.cloud.google.com/apt/doc/apt-key.gpg apt signing key. Use module: https://docs.ansible.com/ansible/latest/modules/apt_key_module.html?highlight=apt_key
    - Add kubernetes http://apt.kubernetes.io/ kubernetes-xenial main. Use module: https://docs.ansible.com/ansible/latest/modules/apt_repository_module.html
    - Install the packages kubelet, kubeadm, kubernetes-cni. Use module: https://docs.ansible.com/ansible/latest/modules/apt_module.html?highlight=apt
- For the master
    - Run kubeadm init. Use module: https://docs.ansible.com/ansible/latest/modules/shell_module.html?highlight=shell
    - Create the directory $HOME/.kube . Use module: https://docs.ansible.com/ansible/latest/modules/file_module.html#file-module
    - Copy /etc/kubernetes/admin.conf to /home/Ubuntu/.kube/config. Use module: https://docs.ansible.com/ansible/latest/modules/copy_module.html?highlight=copy. Hint:
        - To get the ssh user, you can use the variable {{ ansible_ssh_user }},https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html
        - To copy a file or execute any command as root use become: yes
        - To the file has to be owned by the Ubuntu user (chown)
    - Use sysctl to set up network bridge with name net.bridge.bridge-nf-call-iptables to 1Use module: https://docs.ansible.com/ansible/latest/modules/sysctl_module.html?highlight=sysctl. Hint:
        - Recall the command: sudo sysctl net.bridge.bridge-nf-call-iptables=1
    - Install the Weave Net addon, using the command kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')". Use module: https://docs.ansible.com/ansible/latest/modules/shell_module.html?highlight=shell
    - Allow muster to schedule pods by executing kubectl taint nodes --all node-role.kubernetes.io/master-. Use module:

    https://docs.ansible.com/ansible/latest/modules/shell_module.html?highlight=shell
  - Print the 'join' command to be used in the next play by all the workers, kubeadm token create --print-join-command. Use module: https://docs.ansible.com/ansible/latest/modules/shell_module.html?highlight=shell. Hint:
    - You have to register the output to a variable, and you have to make that variable accessible to the next play. To do that recall the section above on how to pass variables between plays
- For the workers:
  - Execute the join command you got from the previous play. Use module: https://docs.ansible.com/ansible/latest/modules/shell_module.html?highlight=shell. See the section above on how to pass variables between plays
    - To print variables and other debug messages you can use the debug module, https://docs.ansible.com/ansible/latest/modules/debug_module.html?highlight=debug
- For the master:
  - Verify that workers have joined the cluster, kubectl get nodes and register a variable for the output. Use module: https://docs.ansible.com/ansible/latest/modules/shell_module.html?highlight=shell
  - Print the output of the task above. Use module: https://docs.ansible.com/ansible/latest/modules/debug_module.html?highlight=debug

Start one t2.medium Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0ac019f4fcb7cb7e6

and 4 t2.micro Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0ac019f4fcb7cb7e6

VMs using EC2.

Obtain their public IP (DNS) and add the t2.medium as the master and the rest as workers. Use the following inventory file:

```
[k8-master]
ec2-x-x-x-xxx.compute-1.amazonaws.com

[worker]
ec2-x-xx-xxx-xxx.compute-1.amazonaws.com

[cluster:children]
```

```
k8-master
worker
`
[cluster:vars]
ansible_ssh_private_key_file=/home/$HOME/vms.pem
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
ansible_ssh_user=ubuntu
```

When you have your playbooks, ready execute the Kubernetes setup playbook.

5.2 Kubernetes Autoscale

To be able to set the minimum and maximum utilization levels (for CPU, mem. etc.) that will trigger autoscaling you'll need to install the Kubernetes Metrics Server

Process

Write a playbook to install the Kubernetes Metrics Server with the following plays/tasks:

- For the master:
    - Install zip. Use module: Use module: https://docs.ansible.com/ansible/latest/modules/apt_module.html?highlight=apt
    - Download and unzip the Metrics Server files from https://docs.ansible.com/ansible/latest/modules/unarchive_module.html . Use the module: https://docs.ansible.com/ansible/latest/modules/unarchive_module.html
    - Deploy Metrics Server in the kubernetes folder (kubectl create -f ~/kubernetes/). Use module: https://docs.ansible.com/ansible/latest/modules/shell_module.html

When the Metrics Server is installed on the master, log in and test if metrics are gathered by typing:

```
kubectl top nodes
```

and

```
kubectl -n kube-system top pods
```

If you don't get any results you may wait for several minutes for the server to deploy.

Run a simple Nginx server:

```
kubectl run nginx --image nginx
```

To make Nginx accessible, you should expose port 80:

```
kubectl expose deploy nginx --port 80 --type NodePort
```

Check that the deployment and pods are present:

```
kubectl get all
```

You should see something like this:

```
NAME                             READY   STATUS    RESTARTS   AGE
pod/nginx-6db489d4b7-4crjs       1/1     Running   0          3m20s

NAME                   TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/kubernetes     ClusterIP    10.96.0.1       <none>        443/TCP
4h37m
service/nginx          NodePort     10.107.13.92    <none>        80:32063/TCP
6s

NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx    1/1     1            1           3m20s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-6db489d4b7    1         1         1       3m20s
```

Note that Nginx is running on port 32063. In your case, it may be different. Test Nginx by opening your browser at http://MATER-IP:PORT.

Now we can benchmark Nginx. To do that, install apache2-utils:

```
sudo apt-get install apache2-utils
```

Run the benchmark:

```
ab -n 50000 -r -c 500 http://MATER-IP:PORT
```

Use the output for your report. The table you are interested in looks like this:

```
Concurrency Level:        XXXX
Time taken for tests:     XXXXX seconds
Complete requests:        XXXX
Failed requests:          XXX
```

```
   (Connect: 0, Receive: XXX, Length: XXX, Exceptions: XXXX)
Total transferred:      XXXX bytes
HTML transferred:       XXXX bytes
Requests per second:    XXX [#/sec] (mean)
Time per request:       XXXX [ms] (mean)
Time per request:       XXXX [ms] (mean, across all concurrent requests)
Transfer rate:          XXX [Kbytes/sec] received
```

Enable autoscaling with 10% cpu utilization and max 5 pods:

```
kubectl autoscale deployment.apps/nginx --cpu-percent=10 --min=1 --max=5
```

Check that the horizontal pod autoscaler (hpa) is running:

```
kubectl describe hpa nginx
```

You should see something like this:

```
Name:                                                  nginx
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Tue, 25 Feb 2020
19:25:01 +0100
Reference:                                             Deployment/nginx
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  <unknown> / 10%
Min replicas:                                          1
Max replicas:                                          5
Deployment pods:                                       1 current / 0
desired
Conditions:
  Type           Status  Reason                   Message
  ----           ------  ------                   -------
  AbleToScale    True    SucceededGetScale        the HPA controller was
able to get the target's current scale
  ScalingActive  False   FailedGetResourceMetric  the HPA was unable to
compute the replica count: missing request for cpu
Events:
  Type     Reason                        Age                   From
Message
  ----     ------                        ----                  ----
-------
  Warning  FailedComputeMetricsReplicas  43s (x12 over 3m28s)  horizontal-
pod-autoscaler  invalid metrics (1 invalid out of 1), first error is:
failed to get cpu utilization: missing request for cpu
  Warning  FailedGetResourceMetric       28s (x13 over 3m28s)  horizontal-
pod-autoscaler  missing request for cpu
```

Notice that the horizontal pod autoscaler (hpa) has some errors. To fix that we need to set some limits to the Nginx deployment. To do that we first delete the horizontal pod autoscaler (hpa) :

```
kubectl delete hpa nginx
```

Next, set the limits by editing the deployment:

```
kubectl edit deploy nginx
```

This will open a vim editor. Locate the 'containers' line and the limits. The section containers should look like this:

```
     containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        resources:
                limits:
                    cpu: "100m"
                requests:
                    cpu: "100m"
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

More details about limits and requests can be found here:https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

Now we are ready to re-enable autoscaling:

```
kubectl autoscale deployment.apps/nginx --cpu-percent=10 --min=1 --max=5
```

To check that it is running correctly type:

```
kubectl describe hpa nginx
```

You could see something like this:

```
Name:                                                  nginx
Namespace:                                             default
```

```
Labels:                                                        <none>
Annotations:                                                   <none>
CreationTimestamp:                                             Tue, 25 Feb 2020
19:43:28 +0100
Reference:                                                     Deployment/nginx
Metrics:                                                       ( current / target )
  resource CPU on pods  (as a percentage of request):  0% (0) / 10%
Min replicas:                                                  1
Max replicas:                                                  5
Deployment pods:                                               1 current / 1
desired
Conditions:
  Type            Status  Reason                Message
  ----            ------  ------                -------
  AbleToScale     True    ScaleDownStabilized   recent recommendations were
higher than current one, applying the highest recent recommendation
  ScalingActive   True    ValidMetricFound      the HPA was able to
successfully calculate a replica count from CPU resource utilization
(percentage of request)
  ScalingLimited  False   DesiredWithinRange    the desired count is within
the acceptable range
Events:                 <none>
```

Run the benchmark again and record the results: ab -n 50000 -r -c 500 http://MASTER-IP:PORT