

# DevOps: Week 5

Matthijs Bos  
10073558

March 2020

GitHub	<a href="https://github.com/matthijsbos/DevOpsLabWeek5">https://github.com/matthijsbos/DevOpsLabWeek5</a>
Playbook	kubernetes.playbook.yml

## 1 Experiment

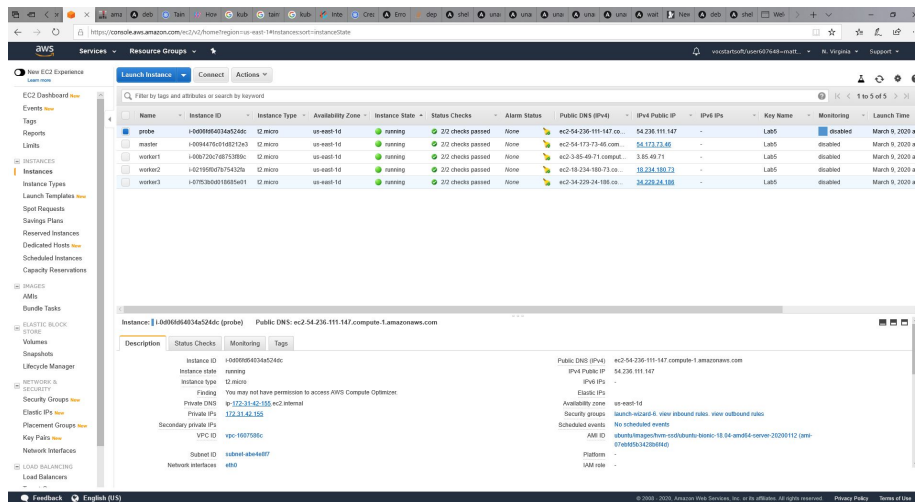


Figure 1: Created a master control node, three worker nodes and an additional node for load testing. The security group inbound traffic rules were relaxed to allow traffic on all ports.

```
ubuntu@ip-172-31-42-15% ~
PLAY [kubernetes_masters] *****
TASK [Gathering Facts] *****
[ec2-54-173-73-46.compute-1.amazonaws.com] *****ok:
TASK [shell] *****
[ec2-54-173-73-46.compute-1.amazonaws.com] *****changed: [ec2-54-173-73-46.compute-1.amazonaws.com]
TASK [debug] *****
[ec2-54-173-73-46.compute-1.amazonaws.com] -> {
  "k8s_nodes.stdout_lines": [
    "NAME          STATUS    ROLES    AGE   VERSION",
    "ip-172-31-33-44 Ready     <none>   17m   v1.17.3",
    "ip-172-31-34-253 Ready     <none>   17m   v1.17.3",
    "ip-172-31-39-81 Ready     master   102m   v1.17.3",
    "ip-172-31-44-14 Ready     <none>   17m   v1.17.3"
  ]
}
PLAY [kubernetes_masters] *****
TASK [Gathering Facts] *****
[ec2-54-173-73-46.compute-1.amazonaws.com] *****ok:
TASK [Install packages] *****
```

Figure 2: Created playbook to deploy kubernetes cluster

```
ubuntu@ip-172-31-39-81: ~
ubuntu@ip-172-31-39-81:~$ kubectl top nodes
NAME          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-33-44 25m          2%     514Mi           58%
ip-172-31-34-253 25m          2%     510Mi           57%
ip-172-31-39-81 80m          8%     729Mi           82%
ip-172-31-44-14 23m          2%     503Mi           56%
ubuntu@ip-172-31-39-81:~$ kubectl -n kube-system top pods
NAME          CPU(cores)   MEMORY(bytes)
coredns-6955765f44-j7k08 3m            11Mi
coredns-6955765f44-v7s2c 2m            7Mi
etcd-ip-172-31-39-81 9m            41Mi
kube-apiserver-ip-172-31-39-81 26m          321Mi
kube-controller-manager-ip-172-31-39-81 11m          41Mi
kube-proxy-5fp4b 1m            10Mi
kube-proxy-8rksw 1m            16Mi
kube-proxy-f8gd9 1m            15Mi
kube-proxy-xccmc 1m            9Mi
kube-scheduler-ip-172-31-39-81 3m            16Mi
metrics-server-7799bf6bb-g7s2n 1m            11Mi
weave-net-4jb8b 1m            51Mi
weave-net-gdmxn 2m            53Mi
weave-net-gm9lg 1m            69Mi
weave-net-gnplb 2m            39Mi
ubuntu@ip-172-31-39-81:~$
```

Figure 3: Kubernetes cluster status

```
ubuntu@ip-172-31-39-81: ~  
ubuntu@ip-172-31-39-81:~$ kubectl run nginx --image nginx  
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.  
deployment.apps/nginx created  
ubuntu@ip-172-31-39-81:~$ kubectl expose deploy nginx --port 80 --type NodePort  
service/nginx exposed  
ubuntu@ip-172-31-39-81:~$ kubectl get all  
NAME                READY   STATUS    RESTARTS   AGE  
pod/nginx-6db489d4b7-pl9w5  1/1     Running   0           15s  
  
NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE  
service/kubernetes  ClusterIP     10.96.0.1    <none>        443/TCP          106m  
service/nginx       NodePort      10.105.29.215 <none>        80:31733/TCP     6s  
  
NAME                READY   UP-TO-DATE   AVAILABLE   AGE  
deployment.apps/nginx  1/1     1             1           15s  
  
NAME                DESIRED   CURRENT   READY   AGE  
replicaset.apps/nginx-6db489d4b7  1         1         1       15s  
ubuntu@ip-172-31-39-81:~$
```

Figure 4: Deploy nginx on cluster

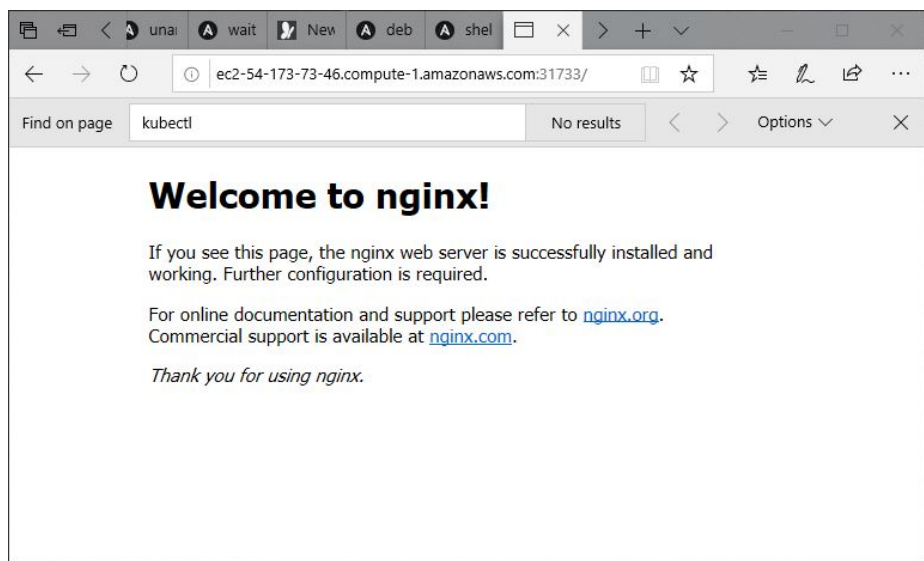


Figure 5: Validated nginx working

```
ubuntu@ip-172-31-42-155: ~  
Concurrency Level:      500  
Time taken for tests:    38.298 seconds  
Complete requests:      50000  
Failed requests:        0  
Total transferred:      42250000 bytes  
HTML transferred:       30600000 bytes  
Requests per second:    1305.55 [#/sec] (mean)  
Time per request:       382.979 [ms] (mean)  
Time per request:       0.766 [ms] (mean, across all concurrent requests)  
Transfer rate:          1077.34 [Kbytes/sec] received  
  
Connection Times (ms)  
  min  mean[+/-sd] median  max  
Connect:    1   159 498.3     8   15342  
Processing: 14   148 268.8    83   10342  
Waiting:    14   147 268.5    82   10342  
Total:      25   306 590.7    90   15646  
  
Percentage of the requests served within a certain time (ms)  
 50%    90  
 66%   99  
 75%  267  
 80%  305  
 90% 1101  
 95% 1271  
 98% 1875  
 99% 3124  
100% 15646 (longest request)  
ubuntu@ip-172-31-42-155:~$
```

Figure 6: Initial load testing

```
ubuntu@ip-172-31-39-81: ~  
Metrics:  
  resource cpu on pods  (as a percentage of request): ( current / target )  
Min replicas:          1  
Max replicas:          5  
Deployment pods:       0 current / 0 desired  
Events:                <none>  
ubuntu@ip-172-31-39-81:~$ kubectl delete hpa nginx  
horizontalpodautoscaler.autoscaling "nginx" deleted  
ubuntu@ip-172-31-39-81:~$ kubectl edit deploy nginx  
error: deployments.apps "nginx" is invalid  
deployment.apps/nginx edited  
ubuntu@ip-172-31-39-81:~$ kubectl autoscale deployment.apps/nginx --cpu-percent=10 --min=1 --max=5  
horizontalpodautoscaler.autoscaling/nginx autoscaled  
ubuntu@ip-172-31-39-81:~$ kubectl describe hpa nginx  
Name:      nginx  
Namespace: default  
Labels:    <none>  
Annotations: <none>  
CreationTimestamp: Mon, 09 Mar 2020 21:01:26 +0000  
Reference:  Deployment/nginx  
Metrics:  
  resource cpu on pods  (as a percentage of request): ( current / target )  
Min replicas:          1  
Max replicas:          5  
Deployment pods:       0 current / 0 desired  
Events:                <none>  
ubuntu@ip-172-31-39-81:~$
```

Figure 7: Configured autoscaling

```
ubuntu@ip-172-31-42-155: ~  
Concurrency Level:      500  
Time taken for tests:    22.948 seconds  
Complete requests:      50000  
Failed requests:         0  
Total transferred:      42250000 bytes  
HTML transferred:       30600000 bytes  
Requests per second:    2178.79 [#/sec] (mean)  
Time per request:       229.485 [ms] (mean)  
Time per request:       0.459 [ms] (mean, across all concurrent requests)  
Transfer rate:          1797.93 [Kbytes/sec] received  
  
Connection Times (ms)  
            min  mean[+/-sd] median  max  
Connect:     0   167 610.4    17   15460  
Processing:   1    46 184.5    18   17129  
Waiting:     0    46 184.1    18   17129  
Total:       1   213 642.7    39   17144  
  
Percentage of the requests served within a certain time (ms)  
 50%    39  
 66%    48  
 75%    56  
 80%    70  
 90%   1033  
 95%   1067  
 98%   1301  
 99%   3080  
100%  17144 (longest request)  
ubuntu@ip-172-31-42-155:~$
```

Figure 8: Second load test

```
ubuntu@ip-172-31-39-81:~$ kubectl describe hpa nginx  
Name: nginx  
Namespace: default  
Labels: <none>  
Annotations: <none>  
CreationTimestamp: Mon, 09 Mar 2020 21:01:26 +0000  
Reference: Deployment/nginx  
Metrics:  
  resource cpu on pods (as a percentage of request): 100% (100m) / 18%  
Min replicas: 1  
Max replicas: 5  
Deployment pods: 4 current / 5 desired  
Conditions:  
  Type      Status Reason Message  
  ----      -  
  AbleToScale True SucceededRescale the HPA controller was able to update the target scale to 5  
  ScalingActive True ValidMetricFound the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)  
  ScalingLimited True TooManyReplicas the desired replica count is more than the maximum replica count  
Events:  
  Type Reason Age From Message  
  ---- -  
  Warning FailedGetResourceMetric 99s (x2 over 114s) horizontal-pod-autoscaler unable to get metrics for resource cpu: no metrics returned from resource metrics API  
  Warning FailedComputeMetricsReplicas 99s (x2 over 114s) horizontal-pod-autoscaler invalid metrics (1 invalid out of 1), first error is: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from resource metrics API  
  Normal SuccessfulRescale 22s horizontal-pod-autoscaler New size: 4; reason: cpu resource utilization (percentage of request) above target  
  Normal SuccessfulRescale 5s horizontal-pod-autoscaler New size: 5; reason: cpu resource utilization (percentage of request) above target  
ubuntu@ip-172-31-39-81:~$
```

Figure 9: Observed auto scaling events

## 1.1 Load Testing Observations

In table and figure below, one can clearly observe a significant effect of the auto scaling on the response time during load testing. For up to 80% of the requests, the auto scaling was able to almost minimize the measured effects.

	50%	66%	75%	80%	90%	95%	98%	99%	100%
No auto scaling	90	99	267	305	1101	1271	1875	3124	15646
With auto scaling	39	48	56	70	1033	1067	1301	3080	17144

Table 1: Results for percentage of the requests served within a certain time (ms) with and without auto scaling

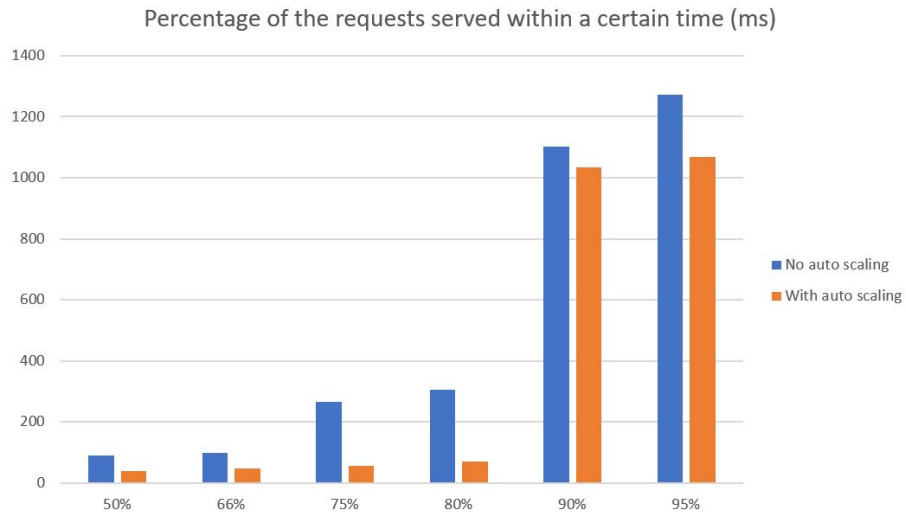


Figure 10: Results for percentage of the requests served within a certain time (ms) with and without auto scaling

## 2 Self Study Questions

### 2.1 Ansible & DevOps

*Discuss how Ansible can be used during DevOps lifecycle, e.g., which stages? What are the advantages, alternatives of Ansible?*

- During the coding phase, an Ansible playbook can be considered a deliverable for implementation of a specific architecture. Playbooks can be checked into revision control for versioning and collaboration.
- During the build phase, Ansible could be used to provision a build toolchain. There may be other tools better suited for this purpose, but the provisioning of a Docker development container for example, could very well be automated using Ansible.
- During the testing phase, Ansible can be used to provision a testing environment that is a realistic copy of a production environment. Given the possibilities for parameterization of playbook, this allows for easy modification of an existing configuration.
- During the deployment phase, one can use ansible to manage the provisioning of a production environment. This is the primary use case for Ansible.
- During the operation phase, the idempotent characteristics of Ansible make it very suitable to make modifications on a live production environment.

### 2.2 Auto Scaling

*Discuss the benefits of using auto scaling for Cloud applications, and for DevOps? Based on the experiments, discuss other scenarios where autoscaling can be used?*

Auto scaling is especially useful during operation of a production environment where the workload of managing the number of workers can be automated. The quality of service can automatically be managed based on preset parameters.

Auto scaling can be useful for any kind of workload that allows for a horizontal growth by dynamically growing and shrinking the pool of workers. This can be the case for front-end servers such as in the experiment, but can also be done for background workers, for example. One example can be a auto scaling pool of background workers that scale based on the number of pending work items in a queue.

### 2.3 Ansible & Azure

*Can you use Ansible with Azure? Is it essential to use Azure DevOps with Ansible?*

Ansible is perfectly suitable for use with Azure, since it doesn't put any requirements on the target hosts other than that these should be accessible through a remote terminal, preferably SSH. There are also specific Ansible modules that interact with Azure APIs for configuration of resources other than compute hosts.

Ansible DevOps doesn't require or restrict the use of Ansible. Azure DevOps is built on top of the regular Azure services that can be provisioned using Ansible.

## 2.4 Azure Resource Manager

*Do you have to use Azure Resource Manager (ARM) when working with Azure resources and deploying VM and services on Azure cloud? Briefly compare ARM and CloudFormation.*

ARM and Cloudformation serve identical purposes in the sense that they are platform-specific methods for provising and configuration of cloud resources. Azure and AWS both offer APIs as well, allowing for external tools to be used as an alternative, not limiting end users to use either of both.

ARM and Cloudformation primarily differ in syntax and used library constructs, but the underlying concepts are the same. Someone switching between the two will mainly be investing in acquiring knowledge on the specific services and configuration options for the different cloud platforms.