1. Learning Summary

- For this week, I learned to how to scale with k8s and understand how we can setup initial requests for both cpu and memory usage as well as setting out a limit to cap these for each of the generated pod. In addition, I learned when is the best time for horizontal scaling or vertically through their pros and cons.
- I am confident that these knowledge will great helpful toward my future career where I not just stop at being a fullstack developer but as well as a devops because I could understand the whole process from production to deployment stage.

2. Lab Activities

Rebuilt the original NodeJS Application and pushed it to localhost:5000

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ docker build -t localhost:5000/node-web .
[+] Building 2.7s (7/7) FINISHED                                                        docker:default
 => [internal] load build definition from Dockerfile                                              0.0s
 => => transferring dockerfile: 106B                                                              0.0s
 => [internal] load metadata for docker.io/library/node:15                                        2.1s
 => [internal] load .dockerignore                                                                 0.0s
 => => transferring context: 2B                                                                   0.0s
 => [internal] load build context                                                                 0.1s
 => => transferring context: 500B                                                                 0.0s
 => CACHED [1/2] FROM docker.io/library/node:15@sha256:608bba799613b1ebf754034ae008849ba51e88b23271412427b76d60ae0d0627  0.0s
 => [2/2] ADD myapp.js /myapp.js                                                                  0.2s
 => exporting to image                                                                            0.1s
 => => exporting layers                                                                           0.1s
 => => writing image sha256:1c7e3ed45d3cfd95f277c60ba0824ca20ce875974d409042c7c89e16beb50447      0.0s
 => => naming to localhost:5000/node-web                                                          0.0s
haydenyeung@HaydenYeung-virtualbox:~/my-container$ docker push localhost:5000/node-web
Using default tag: latest
The push refers to repository [localhost:5000/node-web]
0487314a9f13: Pushed
f92723793659: Layer already exists
f0d8cfcdba81: Layer already exists
4a06816805a3: Layer already exists
b257e69d416f: Layer already exists
1e9c28d06610: Layer already exists
cddb98d77163: Layer already exists
ed0a3d9cbcc7: Layer already exists
8c8e652ecd8f: Layer already exists
2f4ee6a2e1b5: Layer already exists
latest: digest: sha256:3d04ba045998da7fd71ec3d509a4df4b8effe951116dfdd41d1f0fb2b8b972c7 size: 2422
```

Created node-web.yaml and applied it

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
node-web-66f98c7d97-d4ks7  1/1     Running   0          6s
node-web-66f98c7d97-hlbbl  1/1     Running   0          6s
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.152.183.1     <none>        443/TCP        72d
node-web     NodePort    10.152.183.164   <none>        80:31259/TCP   12s
```

“Curl <Cluster-IP>” tests

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ curl 10.152.183.164
Hello ::ffff:10.0.2.15 , this is node-web-66f98c7d97-d4ks7
haydenyeung@HaydenYeung-virtualbox:~/my-container$ curl 10.152.183.164
Hello ::ffff:10.0.2.15 , this is node-web-66f98c7d97-d4ks7
haydenyeung@HaydenYeung-virtualbox:~/my-container$ curl 10.152.183.164
Hello ::ffff:10.0.2.15 , this is node-web-66f98c7d97-hlbbl
haydenyeung@HaydenYeung-virtualbox:~/my-container$ curl 10.152.183.164
Hello ::ffff:10.0.2.15 , this is node-web-66f98c7d97-hlbbl
haydenyeung@HaydenYeung-virtualbox:~/my-container$ curl 10.152.183.164
Hello ::ffff:10.0.2.15 , this is node-web-66f98c7d97-d4ks7
haydenyeung@HaydenYeung-virtualbox:~/my-container$ curl 10.152.183.164
Hello ::ffff:10.0.2.15 , this is node-web-66f98c7d97-hlbbl
```

Applied command "kubectl describe node haydenyeung-virtualbox"

```
Non-terminated Pods:          (10 in total)
  Namespace                   Name                                            CPU Requests  CPU Limits  Memory Requests  Memory Limits
 Age
  ---------                   ----                                            ------------  ----------  ---------------  -------------
 ---
  container-registry          registry-579865c76c-4cjf6                       0 (0%)        0 (0%)      0 (0%)           0 (0%)
 33d
  default                     node-web-66f98c7d97-d4ks7                       0 (0%)        0 (0%)      0 (0%)           0 (0%)
 3m39s
  default                     node-web-66f98c7d97-hlbbl                       0 (0%)        0 (0%)      0 (0%)           0 (0%)
 3m39s
  kube-system                 calico-kube-controllers-5947598c79-gbn5l        0 (0%)        0 (0%)      0 (0%)           0 (0%)
 72d
  kube-system                 calico-node-tv9mz                               250m (12%)    0 (0%)      0 (0%)           0 (0%)
 72d
  kube-system                 coredns-79b94494c7-fwpqf                        100m (5%)     0 (0%)      70Mi (0%)        170Mi (2%)
 72d
  kube-system                 dashboard-metrics-scraper-5bd45c9dd6-592rr      0 (0%)        0 (0%)      0 (0%)           0 (0%)
 28d
  kube-system                 hostpath-provisioner-c778b7559-z7d7k            0 (0%)        0 (0%)      0 (0%)           0 (0%)
 28d
  kube-system                 kubernetes-dashboard-57bc5f89fb-sh7p2           0 (0%)        0 (0%)      0 (0%)           0 (0%)
 28d
```

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests     Limits
  --------           --------     ------
  cpu                450m (22%)   0 (0%)
  memory             270Mi (3%)   170Mi (2%)
  ephemeral-storage  0 (0%)       0 (0%)
  hugepages-2Mi      0 (0%)       0 (0%)
Events:              <none>
```

It was found that those pods required 22% of CPU, and 3% of Memory

```
Capacity:
  cpu:                 2
  ephemeral-storage:   50749700Ki
  hugepages-2Mi:       0
  memory:              7937668Ki
  pods:                110
Allocatable:
  cpu:                 2
  ephemeral-storage:   49701124Ki
  hugepages-2Mi:       0
  memory:              7835268Ki
  pods:                110
```

Pod Access & Checked on both CPU and Memory Usage

```
haydenyeung@HaydenYeung-virtualbox:~$ kubectl exec node-web-66f98c7d97-d4ks7 -it
 -- bash
root@node-web-66f98c7d97-d4ks7:/# cd /sys/fs/cgroup
root@node-web-66f98c7d97-d4ks7:/sys/fs/cgroup# cat cpuacct/cppuacct.usage
cat: cpuacct/cppuacct.usage: No such file or directory
root@node-web-66f98c7d97-d4ks7:/sys/fs/cgroup# cat cpu.stat
usage_usec 5940586
user_usec 2269739
system_usec 3670847
core_sched.force_idle_usec 0
nr_periods 0
nr_throttled 0
throttled_usec 0
nr_bursts 0
burst_usec 0
```

```
root@node-web-66f98c7d97-d4ks7:/sys/fs/cgroup# cat memory.current
42500096
```

Applied new changes to “node-web.yaml” and applied it

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl apply -f node-web.yaml
deployment.apps/node-web configured
service/node-web unchanged
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get replicasets
NAME                  DESIRED   CURRENT   READY   AGE
node-web-5dc95c64df   2         2         2       30s
node-web-66f98c7d97   0         0         0       33m
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
node-web-5dc95c64df-lqhhf   1/1     Running   0          36s
node-web-5dc95c64df-q2gb7   1/1     Running   0          39s
```

Information of the new pod

```
    Limits:
      cpu:  100m
    Requests:
      cpu:        100m
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-ktnm2 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-ktnm2:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   Burstable
```

Applied new changes to “node-web.yaml” – added memory: 10Mi

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl apply -f node-web.yaml
deployment.apps/node-web configured
service/node-web unchanged
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get replicasets
NAME                  DESIRED   CURRENT   READY   AGE
node-web-5dc95c64df   2         2         2       11m
node-web-647fcc5d7c   1         1         0       6m9s
node-web-66f98c7d97   0         0         0       44m
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                        READY   STATUS             RESTARTS     AGE
node-web-5dc95c64df-8zf4p   1/1     Running            0            90s
node-web-5dc95c64df-bk76p   1/1     Running            0            88s
node-web-647fcc5d7c-xzzzc   0/1     CrashLoopBackOff   1 (4s ago)   13s
```

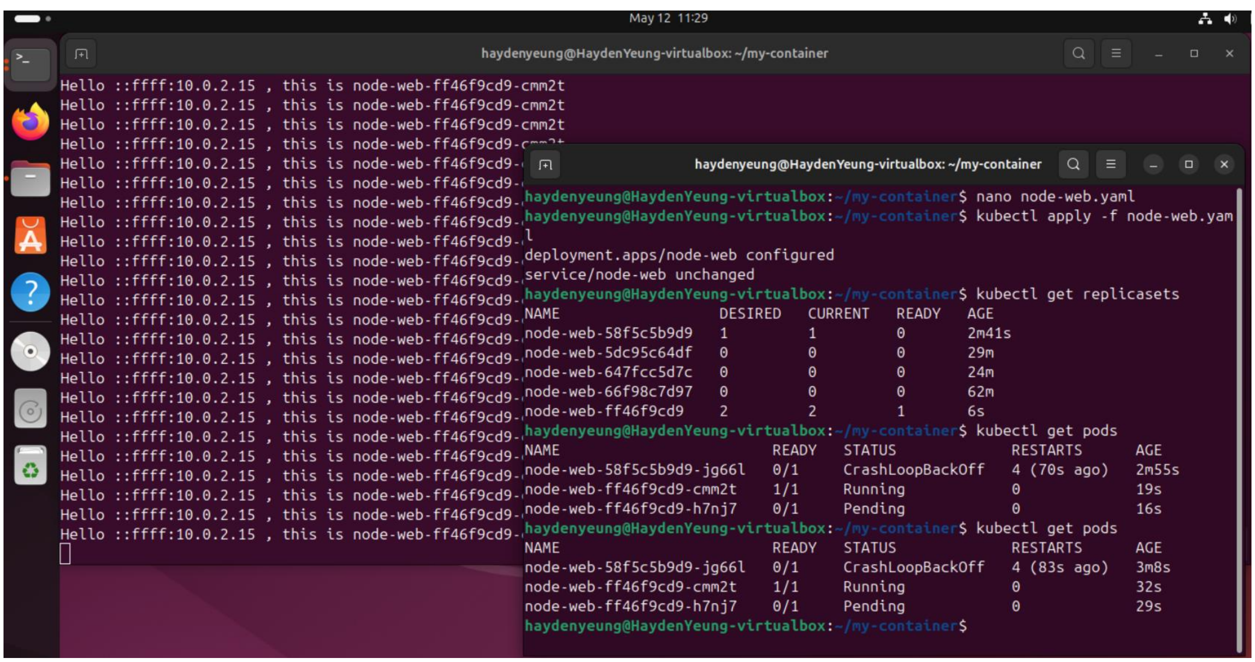
Used “kubectl describe pod …”

```
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl describe pod node-web-647fcc5d7c-xzzzc
Name:             node-web-647fcc5d7c-xzzzc
Namespace:        default
Priority:         0
Service Account:  default
Node:             haydenyeung-virtualbox/10.0.2.15
Start Time:       Mon, 12 May 2025 11:10:32 +1000
Labels:           app=node-web
                  pod-template-hash=647fcc5d7c
Annotations:      cni.projectcalico.org/containerID: 02d3e4449868fa348312e741e48059ef81df859a919410067be22bc3732a1118
                  cni.projectcalico.org/podIP: 10.1.186.19/32
                  cni.projectcalico.org/podIPs: 10.1.186.19/32
Status:           Running
IP:               10.1.186.19
IPs:
  IP:           10.1.186.19
Controlled By:  ReplicaSet/node-web-647fcc5d7c
Containers:
  node-web:
    Container ID:   containerd://c74ce175aff5845a3be403c62e2e12fabda26d78f89d41a9dc91a217a0be3357
    Image:          localhost:5000/node-web
    Image ID:       localhost:5000/node-web@sha256:3d04ba045998da7fd71ec3d509a4df4b8effe951116dfdd41d1f0fb2b8b972c7
    Port:           <none>
    Host Port:      <none>
    State:          Waiting
      Reason:       CrashLoopBackOff
    Last State:     Terminated
      Reason:       StartError
      Message:      failed to create containerd task: failed to create shim task: OCI runtime create failed: runc create failed: unable
to start container process: container init was OOM-killed (memory limit too low?): unknown
```

- It was found that this pod was kept on "CrashLoopBackOff' because the instructed value for memory was too low.

Task 1 – Experiment with the resource requirements

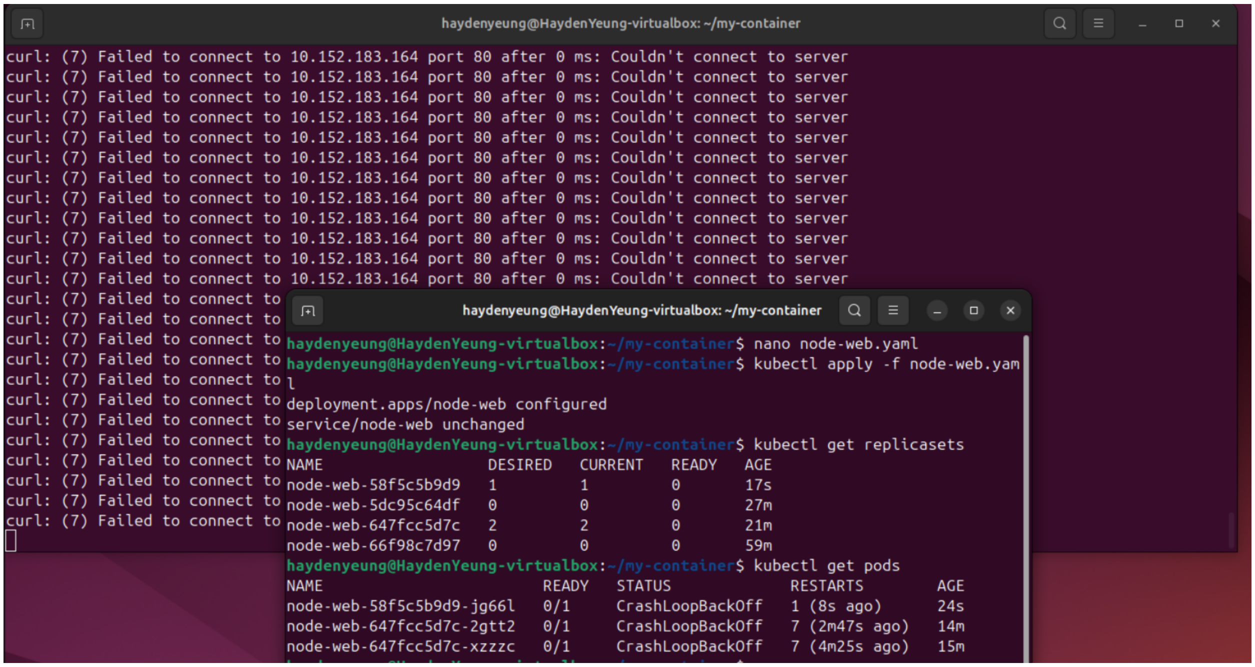
A/ Both High: set CPU = 1000M, memory = 512Mi




```
Started container node-web
90s          Warning    FailedScheduling          pod/node-web-ff46f9cd9-h7nj7
0/1 nodes are available: 1 Insufficient cpu. preemption: 0/1 nodes are available
: 1 No preemption victims found for incoming pod.
```

- Due to high CPU requirement, one of the new pod remained in “pending” state due to insufficient CPU resource to handle another pod from running side-by-side with the first running one.

B/ Both Low: set CPU = 50M, memory = 5Mi



- Because the CPU was set too low, thus pods cannot be created due to insufficient resource to “housing” them.

C/ Low Request, High Limit:

Requests:

- Cpu: 100m
- Memory: 10mi

Limits:

- Cpu: 500m
- Memory: 50mi

```
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c-t6chk
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c-56tj6
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c-56tj6
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c-
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
Hello ::ffff:10.0.2.15 , this is node-web-6d54f5886c
```

```
haydenyeung@HaydenYeung-virtualbox: ~/my-container
haydenyeung@HaydenYeung-virtualbox:~/my-container$ nano node-web.yaml
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl apply -f node-web.yam
l
deployment.apps/node-web configured
service/node-web unchanged
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get replicasets
NAME                  DESIRED   CURRENT   READY   AGE
node-web-58f5c5b9d9   0         0         0       9m4s
node-web-5dc95c64df   0         0         0       35m
node-web-647fcc5d7c   0         0         0       30m
node-web-66f98c7d97   0         0         0       68m
node-web-6d54f5886c   2         2         2       16s
node-web-ff46f9cd9    0         0         0       6m29s
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods
NAME                        READY   STATUS        RESTARTS   AGE
node-web-6d54f5886c-56tj6   1/1     Running       0          16s
node-web-6d54f5886c-t6chk   1/1     Running       0          21s
node-web-ff46f9cd9-cmm2t    1/1     Terminating   0          6m33s
haydenyeung@HaydenYeung-virtualbox:~/my-container$
```

```
QoS Class:                   Burstable
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists fo
r 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists
for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  2m59s  default-scheduler  Successfully assigned default/nod
e-web-6d54f5886c-56tj6 to haydenyeung-virtualbox
  Normal  Pulling    2m58s  kubelet            Pulling image "localhost:5000/nod
e-web"
  Normal  Pulled     2m58s  kubelet            Successfully pulled image "localh
ost:5000/node-web" in 74ms (74ms including waiting). Image size: 361305196 bytes
.
  Normal  Created    2m58s  kubelet            Created container: node-web
  Normal  Started    2m57s  kubelet            Started container node-web
```

Both pods were working well. QoS Class was found to be Burstable

Challeng Task

Re-edit “myapp.js”

- Create an array named memoryHog.
- For every request, 1 string of 1 MB of letter A (~1 million letters A) is added to this array.
- This array is not edit-able → memory keeping increasing with incoming requests.
- The console.log helps tell the current memory usage.

```
haydenyeung@HaydenYeung-virtualbox: ~/my-container
GNU nano 7.2                                   myapp.js
const http = require('http');
const os = require('os');

const listenPort = 8080;

console.log('Server starting on host ' + os.hostname() + 'port ' + listenPort + '...');

let memoryHog = [];

const server = http.createServer((req, res) => {
        let clientIP = req.connection.remoteAddress;
        console.log('Processing request for ' + req.url + ' from ' + clientIP);

        const largeData = 'A'.repeat(1024*1024);
        memoryHog.push(largeData);

        console.log('Memory hog array size: ' + memoryHog.length + ' MB');

        res.writeHead(200);
        res.end('Hello ' + clientIP + ' , this is ' + os.hostname() + '\n');
});

server.listen(listenPort);
```

Re-edit “node-web.yaml”

```
spec:
  containers:
  - image: localhost:5000/node-web:v9
    name: node-web
    resources:
      requests:
        cpu: 200m
        memory: 20Mi
      limits:
        cpu: 300m
        memory: 25Mi
```

Result:

```
Memory hog array size: 3704 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3705 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3706 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3707 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3708 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3709 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3710 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3711 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3712 MBcurl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
```

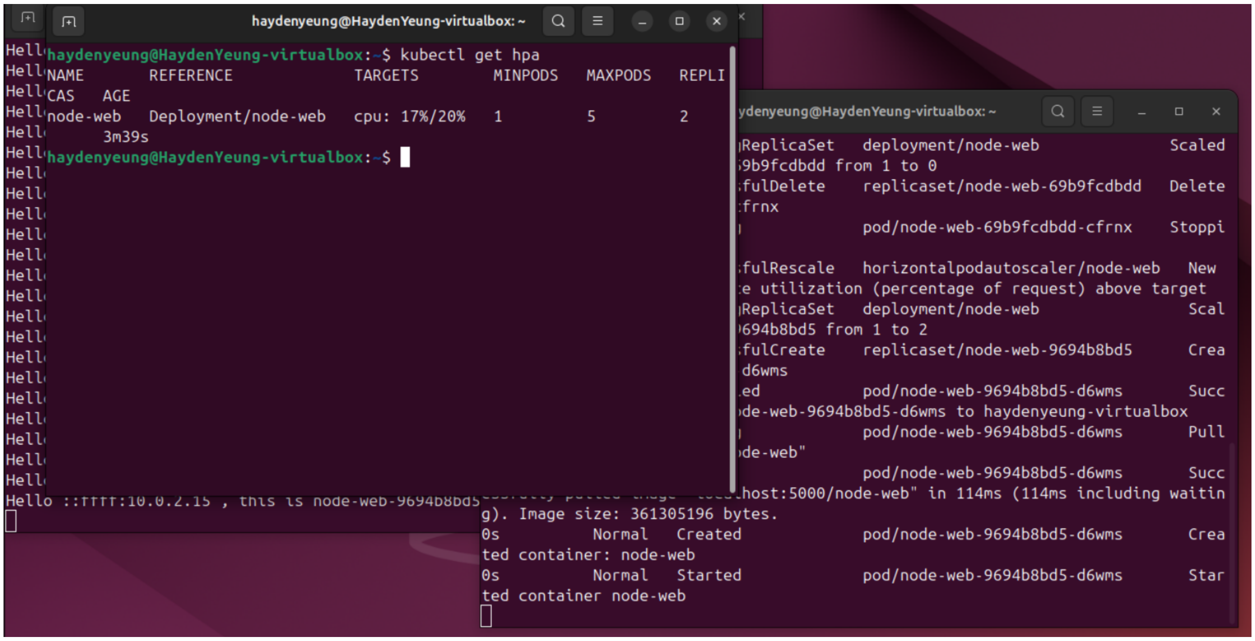
```
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
curl: (7) Failed to connect to 10.152.183.236 port 80 after 0 ms: Couldn't connect to server
Hello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 1 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 2 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 3 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 4 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 5 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 6 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 7 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 8 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 9 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
Memory hog array size: 10 MBHello ::ffff:10.0.2.15 , this is node-web-69b9fcdbdd-cfrnx
```

- It was found that upon reaching the CPU limit, curling became failed and after a short period of time, this pod this terminated and a new pod (with the same name) was created and took over the continuous process of "curling".

Task 2 – Experiment with horizontal scaling

```
GNU nano 7.2                                   autoscale.yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: null
  name: node-web
spec:
  maxReplicas: 5
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: node-web
  targetCPUUtilizationPercentage: 20
status:
  currentReplicas: 0
  desiredReplicas: 0
```

Wrote autoscale according to instructions.



2 Replicas has been created, because, having additional replicas → it took longer time to requiring an additional replica.

3. Benefits of Scalable Resources for Applications

Scalable resources are critical for modern applications as they enable systems to adapt dynamically to varying workloads, ensuring optimal performance and cost efficiency. Scalability allows applications to handle increased user demand, such as during traffic spikes, by allocating additional resources, thereby maintaining responsiveness and preventing downtime (Buyya et al., 2018).

For example, in cloud environments like Kubernetes, scalable resources ensure that applications can access sufficient CPU and memory to meet demand without over-provisioning, which reduces costs. Additionally, scalability supports fault tolerance by redistributing workloads across resources when failures occur, enhancing reliability. This adaptability is particularly beneficial for applications with unpredictable usage patterns, such as e-commerce platforms during sales events, where scalable resources ensure seamless user experiences while optimizing operational expenses (Mell & Grance, 2011).

4. Vertical Scaling in Applications

Vertical scaling involves increasing the capacity of a single server by adding more resources, such as CPU, memory, or storage, to handle increased application demand.

Applications implement vertical scaling by upgrading hardware or allocating additional virtual resources in cloud environments, allowing them to process more tasks without changing the application architecture (Buyya et al., 2018).

- Advantages include simplicity, as it requires minimal changes to application code, and lower latency, since all resources are co-located on a single machine. For example, a database application can benefit from vertical scaling by adding more memory to handle larger datasets.
- However, disadvantages include limited scalability, as there is a physical or virtual cap on how much a single server can be upgraded. Additionally, vertical scaling can lead to downtime during upgrades and creates a single point of failure, reducing fault tolerance (Mell & Grance, 2011).

This approach is less suitable for highly distributed applications requiring massive scalability.

5. Horizontal Scaling in Applications

Horizontal scaling involves adding more servers or instances to distribute an application's workload across multiple nodes, commonly used in cloud-native environments like Kubernetes.

Applications achieve horizontal scaling by deploying additional containers or pods, managed by tools like the HorizontalPodAutoscaler, which adjusts the number of instances based on metrics like CPU utilization (Burns et al., 2019).

- Advantages include high scalability, as adding more nodes can handle virtually unlimited demand, and improved fault tolerance, as failures in one node do not disrupt the entire system. For instance, a web application can scale horizontally to handle traffic spikes during peak hours.
- However, disadvantages include increased complexity, as applications must be designed to handle distributed processing, often requiring load balancers and data consistency mechanisms. Additionally, horizontal scaling may introduce higher operational costs due to managing multiple instances (Buyya et al., 2018).

This approach is ideal for stateless applications but challenging for stateful systems.

References

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2019). Borg, Omega, and Kubernetes. *Communications of the ACM*, 62(5), 50-57. https://doi.org/10.1145/3308560

Buyya, R., Srirama, S. N., Casale, G., & Buyya, R. (2018). *Cloud computing: Principles and paradigms*. Wiley.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 50. https://doi.org/10.6028/NIST.SP.800-145