Cloud Computing – WS 2017

Exercise 5 : Horizontal Pod Autoscaling in Kubernetes

23rd January 2018

Anshul Jindal

anshul.jindal@tum.de

Some Announcements

Internet of Things Praktikum (IN2106, IN4224)

The major objective of the course is to adapt the existing IoT platform implemented previously to the industry's requirements, make it scalable, and implement industry-relevant applications that use the IoT platform as their backend. More Info

Cloud Computing Course for Bavarian Companies

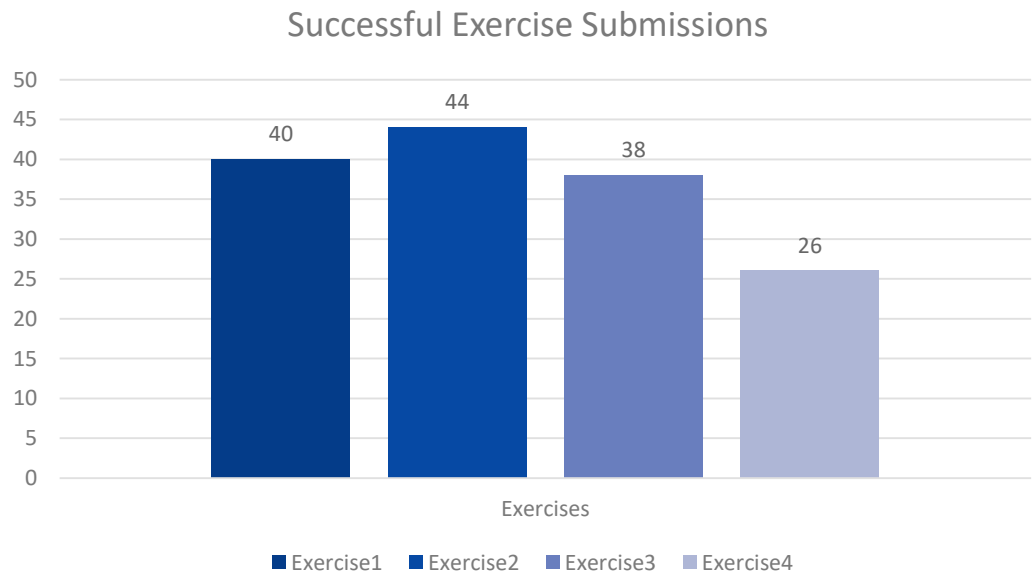
A Cloud Computing course for IT professionals and software architects of Bavarian companies. More Info

Index

Moodle Questions

- **Questions/Exam Pattern**
  - There will be no coding questions in the exam.
  - Theoretical part from the exercises can be asked in the exam(Like about Kubernetes, Microservice vs Monolithic Architecture etc.).
  - There will be mostly short answer (2-3 lines) type questions.
- **Bonus**
  - Maximum of 0.3 bonus will be provided
  - There are total of 6 points (5 exercise + 1 bonus), so 5 points can be used from them to get bonus.
  - Each exercise hold **0.3/5** weightage, so depending upon the correctly submitted exercise you will get the bonus.
- There will be no retake for the exam this semester!

Exercise 4 Solution



Exercise 4 Solution

```
# ------------------- hello-world-service Deployment ------------------- #
kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: hello-world-service
  name: hello-world-service
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: hello-world-service
  template:
    metadata:
      labels:
        k8s-app: hello-world-service
    spec:
      containers:
      - name: hello-world-service
        image: docker.io/DOCKERHUB_ID/microservice:hello
        imagePullPolicy: Always
        ports:
        - containerPort: 9001
          protocol: TCP
---
```

Type of the resource, here Deployment

Name of the application

Configuration

Add your docker hub-id image path

Specify Port Number

Exercise 4 Solution

```yaml
# ------------------- product-descp-service Deployment ------------------- #
kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: product-descp-service
  name: product-descp-service
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: product-descp-service
  template:
    metadata:
      labels:
        k8s-app: product-descp-service
    spec:
      containers:
      - name: product-descp-service
        image: docker.io/DOCKERHUB_ID/microservice:productdescp
        imagePullPolicy: Always
        ports:
        - containerPort: 9002
          protocol: TCP
```

Type of the resource, here Deployment

Name of the application

Configuration

Add your docker hub-id image path

Specify Port Number

Exercise 4 Solution

```yaml
# ------------------- hello-world-service Service ------------------- #
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: hello-world-service
  name: hello-world-service
  namespace: default
spec:
  ports:
    - port: 9001
      targetPort: 9001
  selector:
    k8s-app: hello-world-service
---
# ------------------- server Service ------------------- #
---
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: server
  name: server
  namespace: default
spec:
  ports:
    - port: 8080
      targetPort: 8080
  type: LoadBalancer
  selector:
    k8s-app: server
```

Type of the resource, here Service

Name of the service

Port Configuration

Target Name

Type of the resource, here Service

Name of the service

Port Configuration along with type

Target Name

Solution Demo

Exercise 5

Exercise 5 : Horizontal Pod Autoscaling in Kubernetes

Introduction: What is Autoscaling?

- Imagine you have a 24/7 production service with a load that is variable in time.
- It is very busy during the day, and relatively low at night.
- Ideally, we would want the number of nodes in the cluster and the number of pods in deployment to dynamically adjust to the load to meet end user demand.

Autoscaling is the is a method, whereby the amount of computational resources in a server farm, typically measured in terms of the number of active servers, scales automatically based on the load on the farm [1].

Advantages of Autoscaling

For companies running their own web server infrastructure

- allowing some servers to go to sleep during times of low load, saving on electricity costs (as well as water costs if water is being used to cool the machines)
- can also take care of replacing unhealthy instances and therefore protecting somewhat against hardware, network, and application failures.
- Example: AWS Autoscaling, GCE Autoscale etc

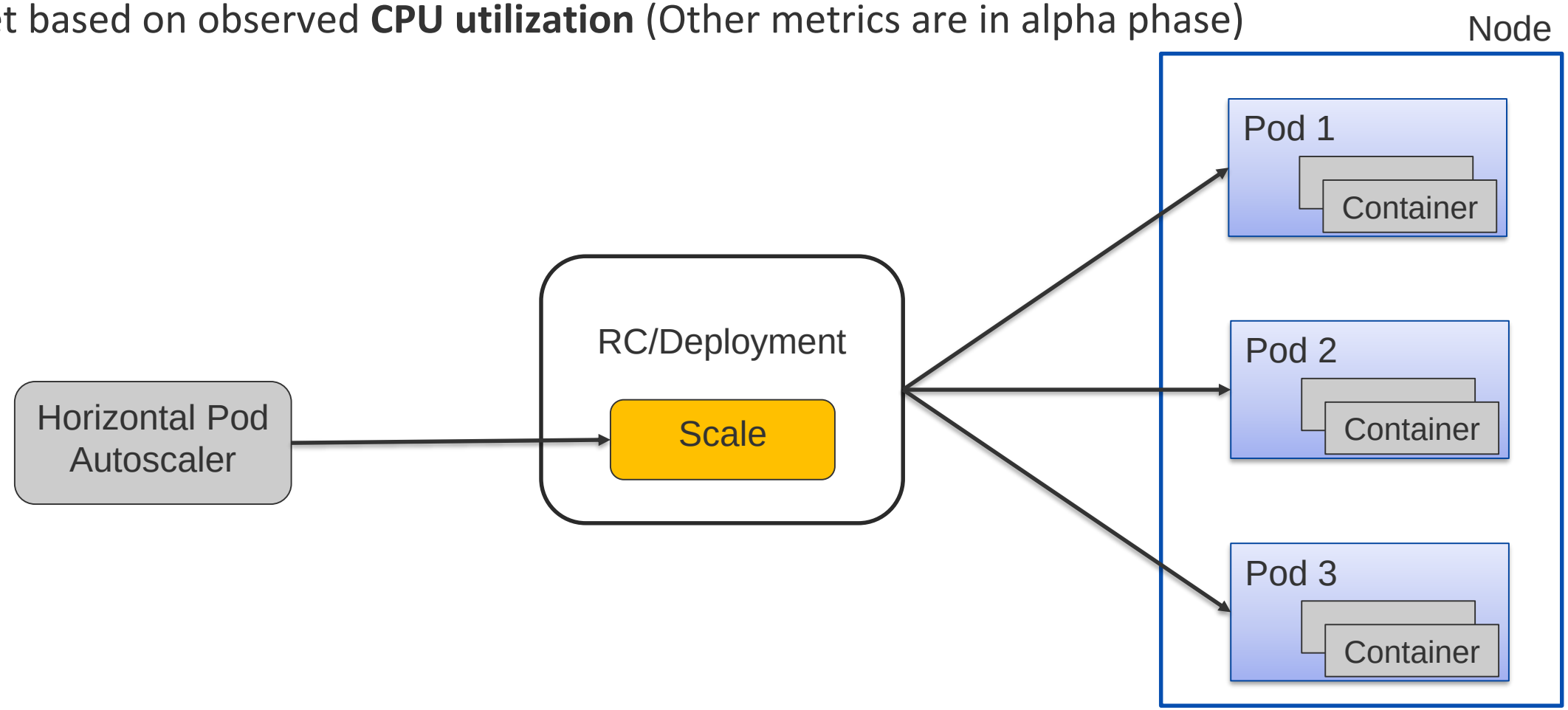
For companies using infrastructure hosted in the cloud

- lower bills, because most cloud providers charge based on total usage rather than maximum capacity
- can handle unexpected traffic spikes better.

greater uptime and more availability

Introduction: What is the Horizontal Pod Autoscaler?

- Automatically scales the number of pods in a replication controller, deployment or replica set based on observed **CPU utilization** (Other metrics are in alpha phase)



Autoscaling Algorithm

- The autoscaler is implemented as a control loop and periodically queries pods which are part of **Scale subresource**, and collects their CPU utilization.

- Then, it compares the **arithmetic mean of the pods' CPU utilization*** with the target defined. Target here is threshold cpu utilization

- Then adjusts the replicas of the Scale if needed to match the target (preserving condition: MinReplicas <= Current Replicas <= MaxReplicas).

TargetNumOfPods = ceil(sum(CurrentPodsCPUUtilization) / Target)

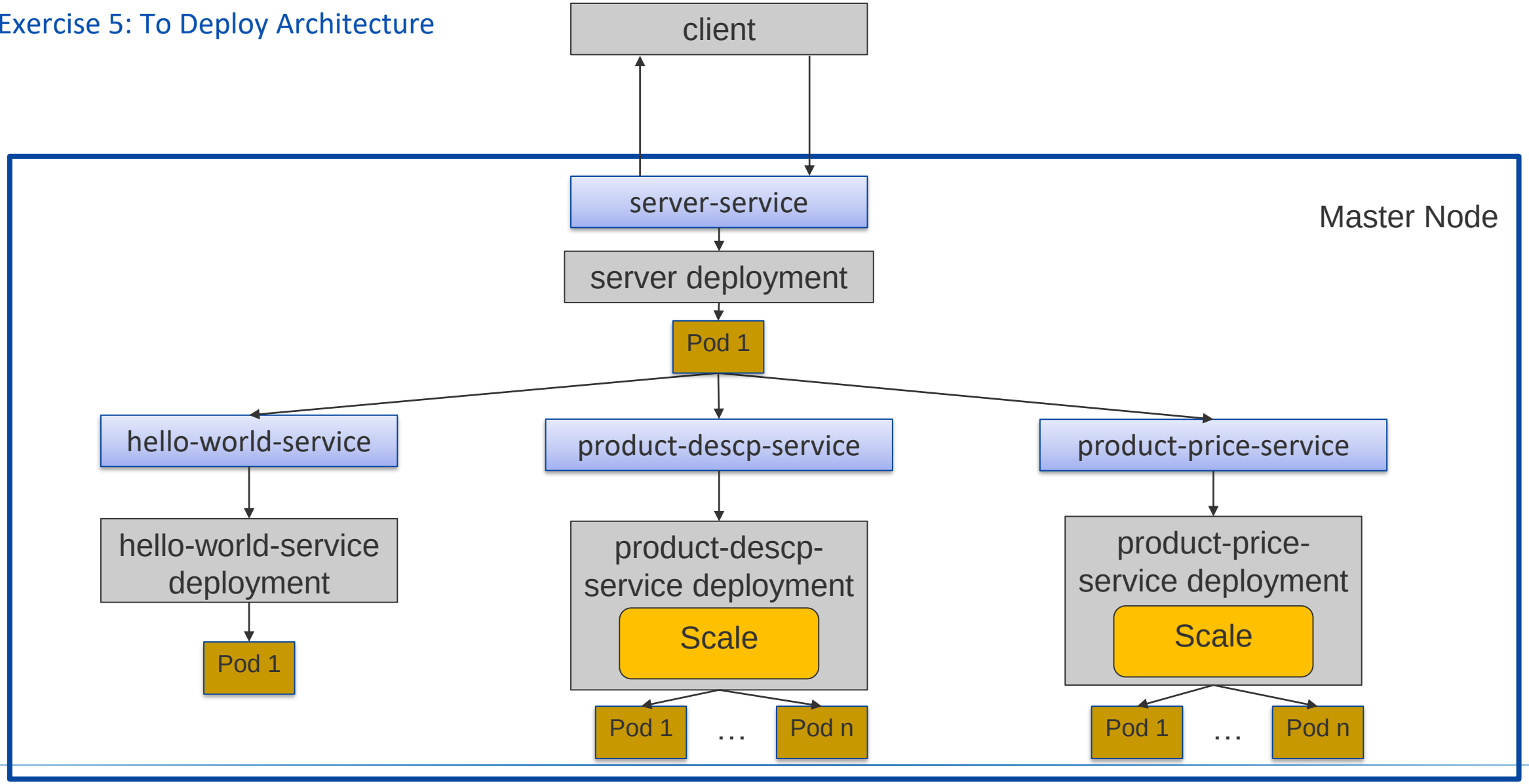
*CPU utilization is the recent CPU usage of a pod (average across the last 1 minute) divided by the CPU requested by the pod

Heapster and InfluxDb

- Heapster
  - Heapster enables Container Cluster Monitoring and Performance Analysis for Kubernetes.
  - Heapster collects and interprets various signals like compute resource usage, lifecycle events, etc.
  - Heapster require InfluxDB to store the data.

- **InfluxDB**
  - It is an open-source time series database developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics [2].

Exercise 5: To Deploy Architecture



Exercise 5: Step 1 - **Installation**

We will be using kubeadm to deploy the kubernetes Cluster.

1. Install Docker, Kubernetes, Kubeadm and Kubectl on Master and Slave nodes (As part of the exercise we are not using slave nodes because of the limited number of VMs on LRZ Cloud)
2. Check the Installation by running kubectl command, you would get something like this

```
kubectl controls the Kubernetes cluster manager.

Find more information at https://github.com/kubernetes/kubernetes.

Basic Commands (Beginner):
  create         Create a resource by filename or stdin
  expose         Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout        Manage a deployment rollout
  rolling-update Perform a rolling update of the given ReplicationController
  scale          Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
  autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes
```

Exercise 5: Step 2 - **Configuring Kubernetes**

Initialize the Master Node using kubeadm init command **(need to be run as root)**

```
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/et
[init] Waiting for the kubelet to boot up the control plane as Static Pods from directory "
[init] This might take a minute or longer if the control plane images have to be pulled.
[apiclient] All control plane components are healthy after 48.504239 seconds
[uploadconfig] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-sy
[markmaster] Will mark node vm-10-155-208-213 as master by adding a label and a taint
[markmaster] Master vm-10-155-208-213 tainted and labelled with key/value: node-role.kubern
[bootstraptoken] Using token: e1e847.e8e8b3eda94f8587
[bootstraptoken] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order
[bootstraptoken] Configured RBAC rules to allow the csrapprover controller automatically ap
[bootstraptoken] Configured RBAC rules to allow certificate rotation for all node client ce
[bootstraptoken] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join --token e1e847.e8e8b3eda94f8587 10.155.208.213:6443 --discovery-token-ca-cer
c307fa17285eaeedaa556d4
```

To be run as the normal user

To be run on the slave nodes for joining the kubernetes cluster

Exercise 5: Step 2 - **Configuring Kubernetes**

Before going forward, you should create a new user (as described in exercise-1), add it to sudoers and run the following commands on it:

```
sudo mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Check everything is running fine by running command kubectl get nodes

```
ansjin@vm-10-155-208-213:/$ sudo kubectl get nodes
NAME                  STATUS     ROLES     AGE       VERSION
vm-10-155-208-213     NotReady   master    14m       v1.9.0
ansjin@vm-10-155-208-213:/$
```

Exercise 5: Step 3 - **Installing the Pod Network**

- Master is up so we need to install the pod network.
- It is necessary to do this before you try to deploy any applications to your cluster, and before kube-dns will start up.
- See the add-ons page for a complete list of available network add-ons.
- We will be installing weave net, which provides networking and network policy.

kubectl apply -f <add-on.yaml>

```
ansjin@vm-10-155-208-213:/$ kubectl apply -f "https://cloud.weave.works/k8s/net?
serviceaccount "weave-net" created
clusterrole "weave-net" created
clusterrolebinding "weave-net" created
role "weave-net" created
rolebinding "weave-net" created
daemonset "weave-net" created
ansjin@vm-10-155-208-213:/$
```

Exercise 5: Step 4 – **Status Check**

- Check the status of pods run the following command.

kubectl get pods --all-namespaces

```
ansjin@vm-10-155-208-213:/$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY     STATUS    RESTARTS   AGE
kube-system   etcd-vm-10-155-208-213                        1/1       Running   0          35s
kube-system   kube-apiserver-vm-10-155-208-213              1/1       Running   0          39s
kube-system   kube-controller-manager-vm-10-155-208-213     1/1       Running   0          39s
kube-system   kube-dns-6f4fd4bdf-t67wg                      0/3       Pending   0          1m
kube-system   kube-proxy-jz4s5                              1/1       Running   0          1m
kube-system   kube-scheduler-vm-10-155-208-213              1/1       Running   0          34s
kube-system   weave-net-7w48x                               2/2       Running   0          42s
```

Try this command: watch kubectl get pods --all-namespaces

Deployment using YAML file

```yaml
---
# ------------------- hello-world-service Deployment ------------------- #
kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: hello-world-service
  name: hello-world-service
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: hello-world-service
  template:
    metadata:
      labels:
        k8s-app: hello-world-service
    spec:
      containers:
      - name: hello-world-service
        image: docker.io/DOCKERHUB_ID/microservice:hello
        imagePullPolicy: Always
        ports:
        - containerPort: 9001
          protocol: TCP
        resources:
          limits:
            cpu: 200m
            memory: 400Mi
          requests:
            cpu: 100m
            memory: 200Mi
---
```

Type of the resource, here Deployment

Name of the application

Configuration

Add your docker hub-id where images are present

Without this information HPA will not work

```yaml
# ------------------- HPA ------------------- #
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: product-price-service
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: product-price-service
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 20
```

Type of the resource, here HPA

Name

Target Configuration

Number of Replicas

Autoscaling Parameter and Target Utilization

Exercise 5: Step 5 – **Running Everything using yaml file**

- We provide you with a yaml file which contain the deployments for
  - InfluxDB
  - Heapster
  - Dashboard
  - Metrics Collection
  - Application (deployment, Services and HPA)

- Run the following command from inside the source directory.

```
kubectl apply -f exercise5.yaml
```

- Enable the kube API for external access using following command.

```
sudo kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='^*$'&
```

- Dashboard is available at

http://YOUR_VM_IP:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

Exercise 5: Step 6 – **Running your containerized image**

- Check the status of all the deployments by running the command

kubectl get deployments --all-namespaces

```
ansjin@vm-10-155-208-213:/$ kubectl get deployments --all-namespaces
NAMESPACE     NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
default       hello-world-service    1         1         1            1           4m
default       product-descp-service  1         1         1            1           4m
default       product-price-service  1         1         1            1           4m
default       server                 1         1         1            1           3m
kube-system   kube-dns               1         1         1            1           1h
kube-system   kubernetes-dashboard   1         1         1            1           1h
```

Exercise 5: Step 6 – **Status of HPAs**

- Check the status of all the deployments by running the command

kubectl get hpa --all-namespaces

```
root@vm-10-155-208-160:~# kubectl get hpa --all-namespaces
NAMESPACE   NAME                    REFERENCE                          TARGETS    MINPODS   MAXPODS   REPLICAS
default     product-price-service   Deployment/product-price-service   0% / 20%   1         5         1
```

Exercise 5: Step 7 – **Status of Services**

- you can view the services by running the command

kubectl get services –all-namespaces

```
ansjin@vm-10-155-208-213:/$ kubectl get svc --all-namespaces
NAMESPACE     NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
default       hello-world-service    ClusterIP      10.101.168.134   <none>        9001/TCP
default       kubernetes             ClusterIP      10.96.0.1        <none>        443/TCP
default       product-descp-service  ClusterIP      10.99.204.194    <none>        9002/TCP
default       product-price-service  ClusterIP      10.105.146.127   <none>        9003/TCP
default       server                 LoadBalancer   10.104.32.226    <pending>     8080:32400/TCP
kube-system   kube-dns               ClusterIP      10.96.0.10       <none>        53/UDP,53/TCP
kube-system   kubernetes-dashboard   ClusterIP      10.102.132.97    <none>        443/TCP
```

Here the external-IP is your **VM public IP** and the port number is **32400**.

Your Application would be running at address

http://VM_IP:PORTNUMBER/exercises/exercise3?name=CCS&productId=3

http://VM_IP:PORTNUMBER/exercises/exercise4

Exercise 5: Step 8 – **Deleting and Resetting the Cluster**

- To delete the service and deployment you can run the following command:

```
kubectl delete service,deployment <deployment_Name>
```

- Reset all kubeadm installed state, run the following command on master

```
sudo kubeadm reset
sudo rm -r $HOME/.kube/
```

Short Demo

Tasks To be Completed

Tasks to be completed

As part of the exercise5, following are the tasks need to be completed:

1. Install docker on the VM.
2. Install Kubernetes as done in last exercise.
3. Run the Kubernetes Cluster
4. Install the Pod network
5. Enable pod Scheduling on Master using command : kubectl taint nodes --all node-role.kubernetes.io/master-
6. Run the following commands to expose the Kubernetes APi
   a. sudo kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='^*$'&
   b. Enable Port **8001** in IPTables so that it can be accessed from outside.

Deadline for submission: 29th January 2018 23:59

Tasks to be completed

7. Add 1 More Horizontal pod Autoscalers in **exercise5.yaml**:
    a. product-descp-service
        a. MinReplicas = 1
        b. Max Replicas = 2
        c. Target CPU Utilization = 10%
8. Run exercise5.yaml after replacing **DOCKER-HUB ID** to your ID, using command **kubectl apply –f exercise5.yaml. Same Images will be used as used in last exercise.**
9. Run the following command to check the port number of the Server Service on which it is running and afterwards **enable that port number in IP tables**.
    a. kubectl get services –all-namespaces
10. Visit the URL to test whether the Application is running or not

http://VM_IP:PORTNUMBER/exercises/exercise3?name=CCS&productId=3

Deadline for submission: 29th January 2018 23:59

Submission

Submission

To submit your application results you need to follow this :

1. Open the cloud Class server url
2. Login with your provided username and password.
3. After logging in, you will find the button for **exercise5**
4. Click on it and a form will come up where you must provide
    1. VM ip on which your application is running
    2. Port number of the Server application

Example:

10.0.23.1

32465

5. Then click submit.
6. You will get the correct submission from server if everything is done correctly(multiple productids will be tested while submission of the code).

Remember no cheating and no Hacking ☺

Important points to Note:

1. Make sure your VM and your application is running after following all the steps mentioned in this manual.
2. We will grade you based upon the number of tasks completed by you.
3. You will get to see, what your application has submitted to the server.
4. You can submit as many times until the deadline of exercise.
5. Multiple submission will overwrite the previous results.

Good Luck and Happy Coding☺

Thank you for your attention! ☺

Questions?

Appendix

Node.js Client Application Deployment : Port unblock

- If your request timed out, your VM probably has some firewall rules in place prevent a user to call your web server from the outside.
- The iptables rules are located in the file **/etc/iptables/rules.v4**. Open this file with your favourite editor:
- After line 9 insert a new line allowing incoming connections on port 8080:

-A INPUT -p tcp -m tcp --dport 8080 -m state --state NEW -j ACCEPT

```
# Generated by iptables-save v1.6.0 on Fri May  6 15:32:09 2016
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 4 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -m state --state NEW -j ACCEPT
-A INPUT -p tcp -m tcp --dport 8080 -m state --state NEW -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m limit --limit 3/min -j LOG
COMMIT
# Completed on Fri May  6 15:32:09 2016
~
```

- Save it and to apply the new iptables rules, you need to reload them to your local firewall system.

iptables-restore < /etc/iptables/rules.v4