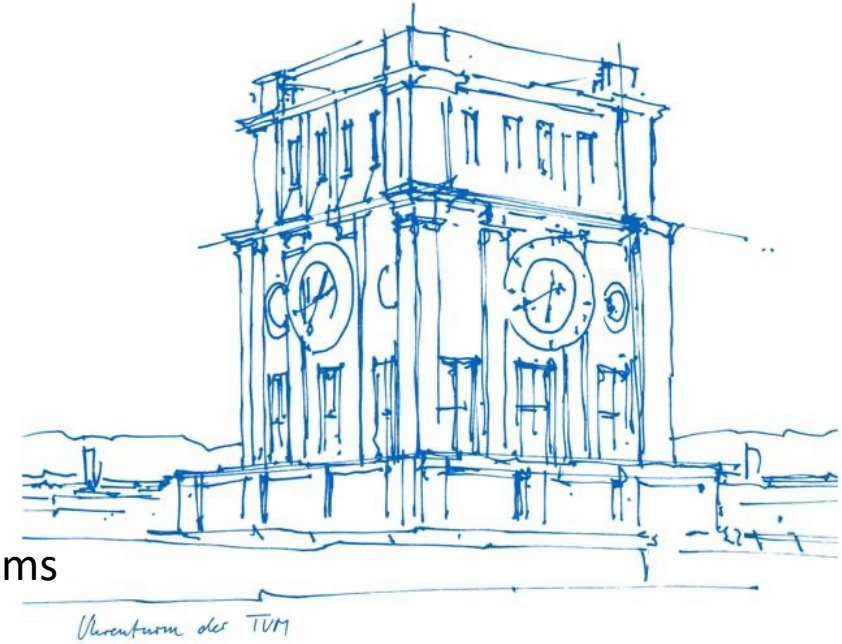
Cloud Computing Exercise – 4

Application Deployment using Kubernetes



Mohak Chadha (M.Sc. Informatics)

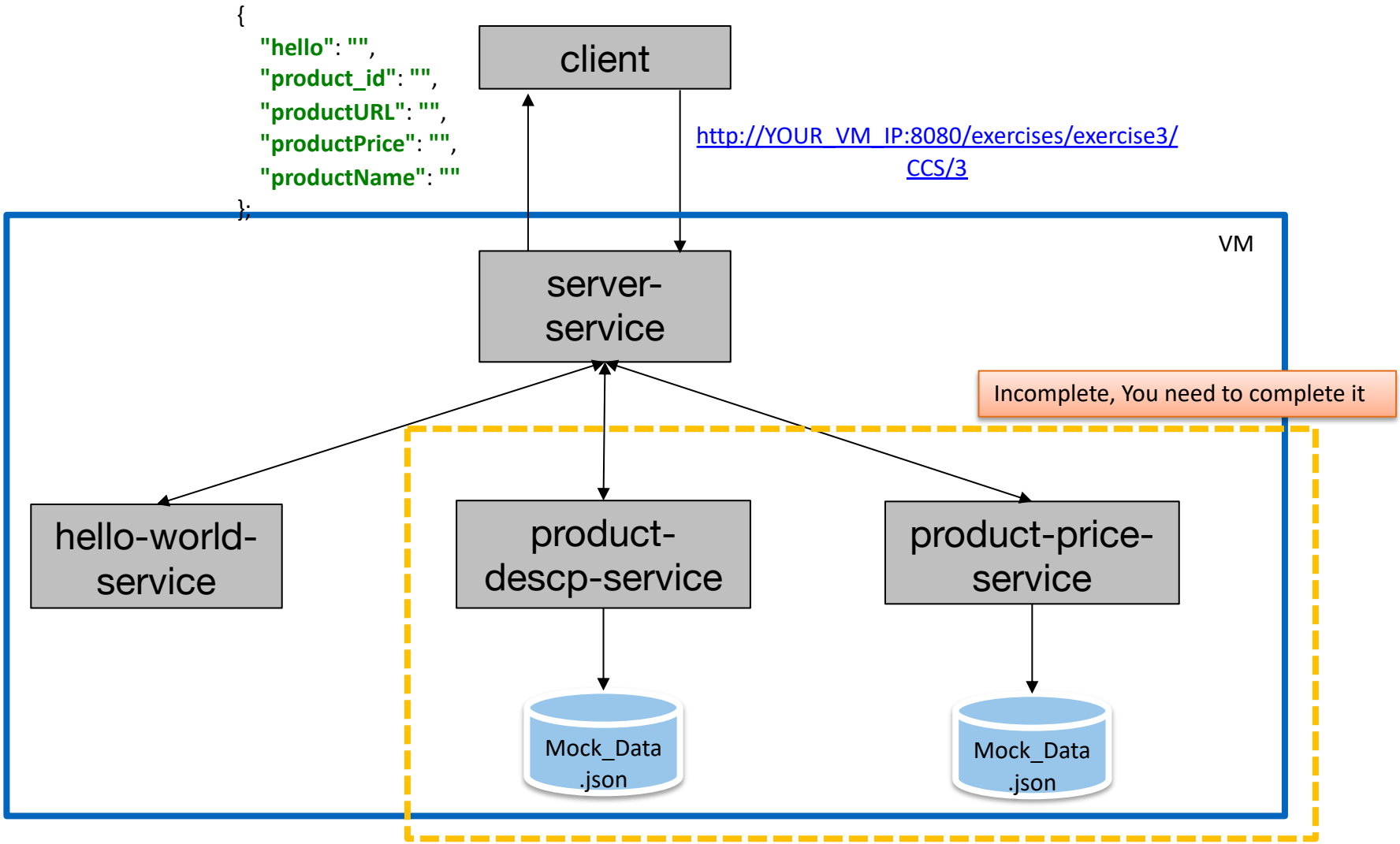
mohak.chadha@tum.de

Chair of Computer Architecture and Parallel Systems

Technical University of Munich (TUM), Germany

Exercise 3 Solution

To develop Microservice Architecture



product-descp-service/product_descp.js

```
module.exports = function (options) {
    //Import the mock data json file
    const mockData = re                    _DATA.json'
    //Add the patterns and their corresponding functions
    this.add('role:product,cmd:getProductURL', productURL);
    this.add('role:product,cmd:getProductName', productName);

//Describe the logic inside the function
function productURL(msg, respond) {

    var myFoundProduct = '';
    for(var i=0; i <mockData.length;i++ ) {
        if(mockData[i].product_id == msg.productId ){
            myFoundProduct = i + 1;
            break;
        }
    }
    if(myFoundProduct){
        respond(null, { result: mockData[myFoundProduct - 1].product_url});
    }
    else {
        respond(null, { result: ''});
    }
}[…]
```

Pattern

Action

Loop to iterate over all the values

Find the correct product based on id

Send the product_url

Error message can be sent here

product-descp-service/product_descp.js continued..

```
//Describe the logic inside the function
function productName(msg, respond) {
    var myFoundProduct = '';
    for(var i=0; i <mockData.length;i++ ) {

        if(mockData[i].product_id == msg.productId ){
            myFoundProduct = i + 1;
            break;
        }
    }
    if(myFoundProduct){
        respond(null, { result: mockData[myFoundProduct - 1].product_name});
    }
    else {
        respond(null, { result: ''});
    }
}
```

Loop to iterate over all the values

Find the correct product based on id

Send the product_name

Error message can be sent here

product-price-service/product_price.js

```
function productPrice(msg, respond) {
    var myFoundProduct = '';
    for(var i=0; i <mockData.length;i++ ) {

        if(mockData[i].product_id == msg.productId){
            myFoundProduct = i + 1;
            break;
        }
    }
    if(myFoundProduct){
        respond(null, { result: mockData[myFoundProduct - 1].product_price});
    }
    else {
        respond(null, { result: ''});
    }

}
```

Loop to iterate over all the values

Find the correct product based on id

Send the product_name

Error message can be sent here

server/services/productPrice.js

```
/**
 * Service Method
 */
const GET_PRODUCT_PRICE = { role: 'product', cmd: 'getProductPrice' };
/**
 * Call Service Method
 */

const getProductPrice = (productId) => {
    return act(Object.assign({}, GET_PRODUCT_PRICE, { productId }));
};
```

Pattern

Created a function to call the service action

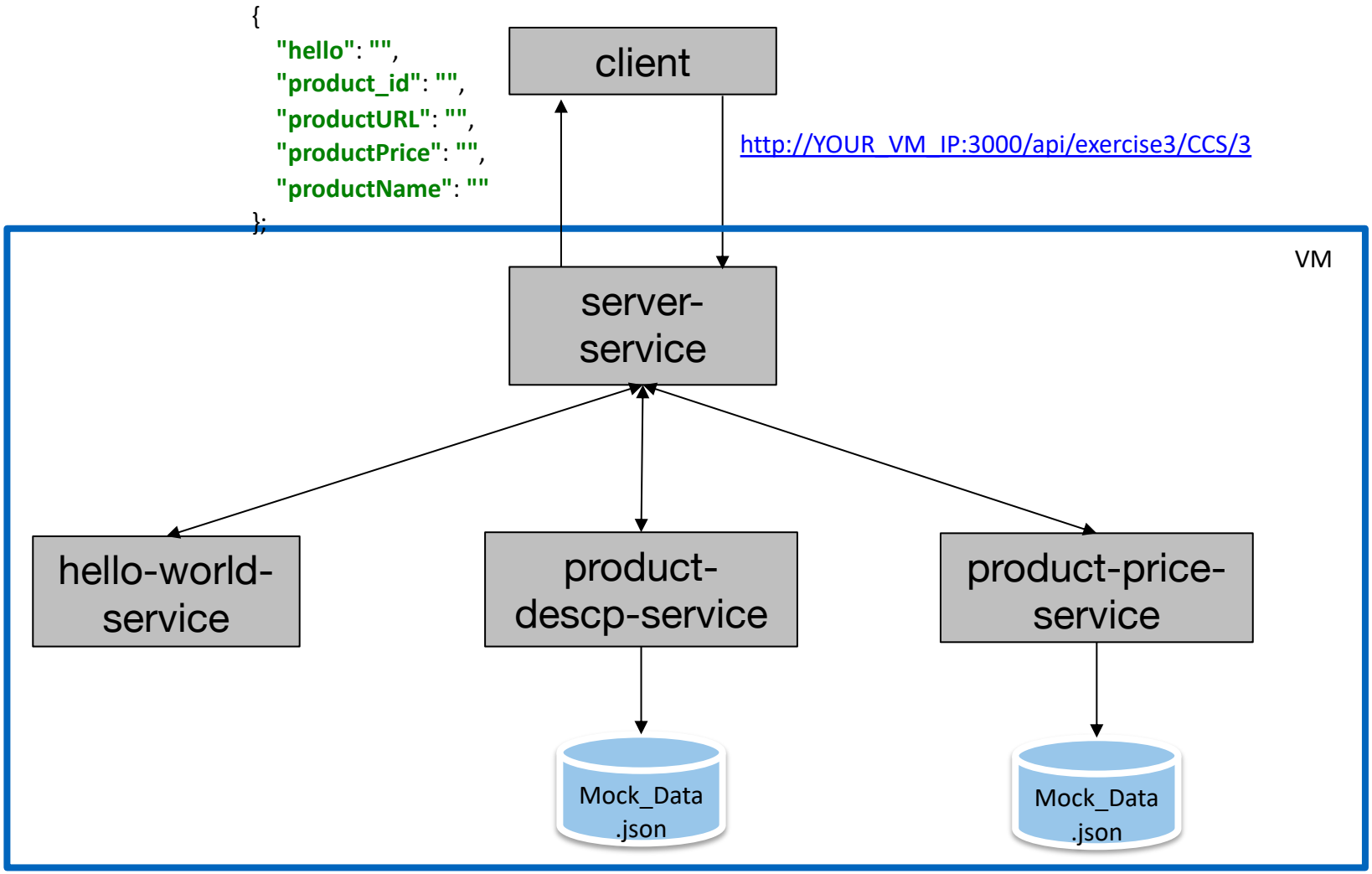
productId is sent to the service

```
module.exports = {
    getProductPrice
};
```

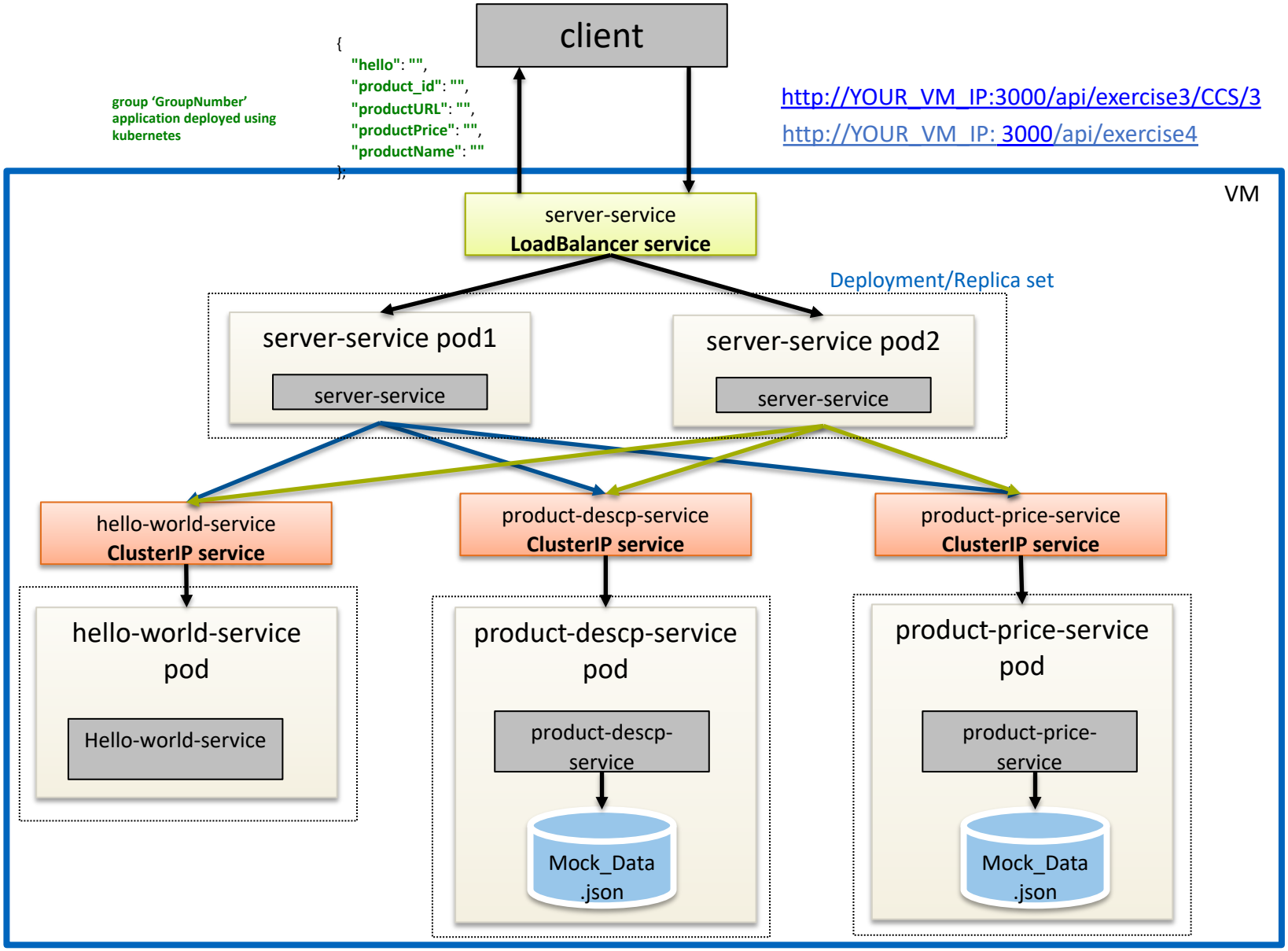
Functions exposed for accessible by server/app.js

Exercise 4

Exercise3 Application Architecture



Exercise 4 Application Architecture



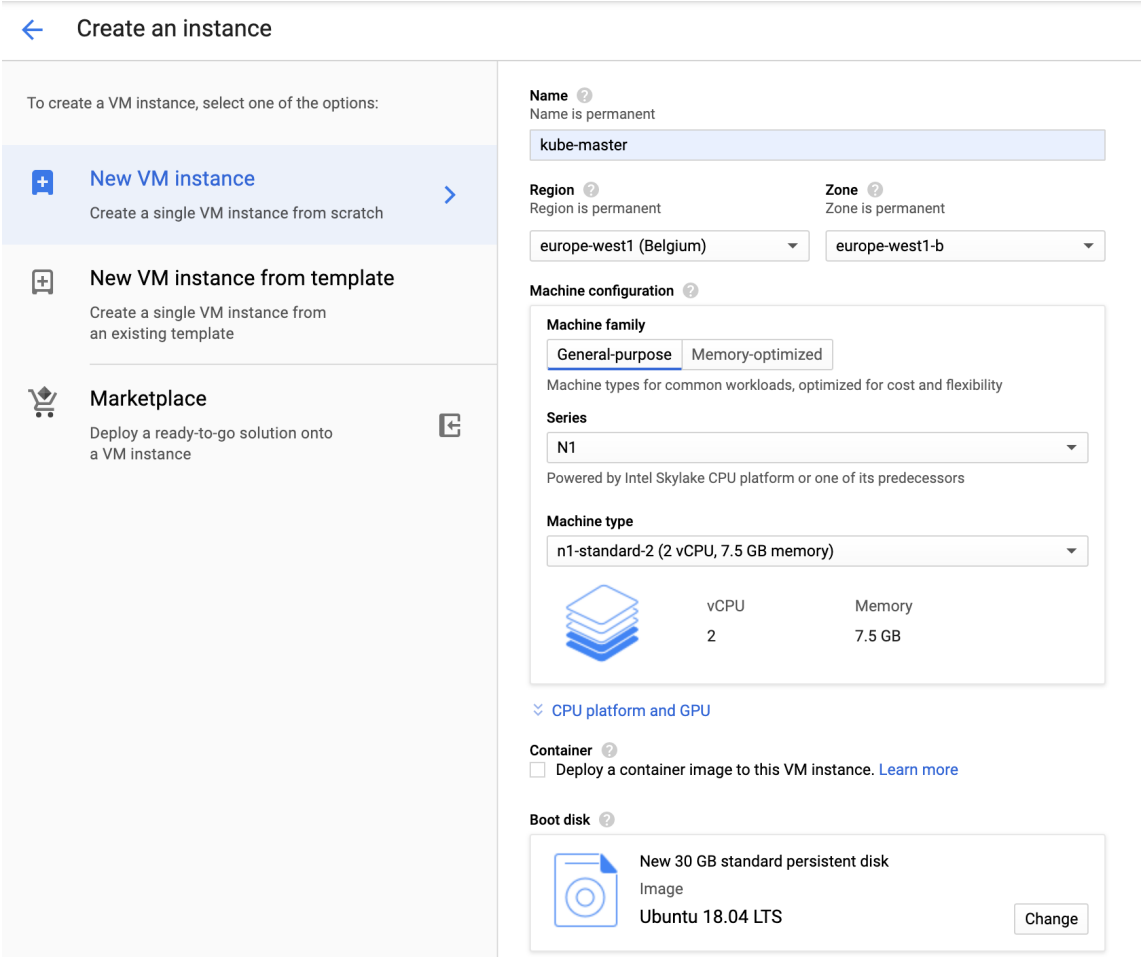
Kubernetes Installation

and

Running the application

Launching an EC2 Master Instance

- Create a new VM on GCP (select instance > = n1-standard-2)



- Kubernetes cluster operate on the below mentioned ports, so enable these.
  - 30000-32767 (node port range)
  - 8001, 443, 6443 (for Kubernetes communication)

SSH to Master

SSH into Master VM

```
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Jun 12 15:18:10 UTC 2023

  System load:                    2.0
  Usage of /:                     5.6% of 96.75GB
  Memory usage:                   17%
  Swap usage:                     0%
  Processes:                      113
  Users logged in:                0
  IP address for ens3:            192.168.130.79
  IP address for docker0:         172.17.0.1
  IP address for br-2102ff17c7d0: 172.18.0.1
  IP address for br-b3a448807fc0: 172.19.0.1

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

108 packages can be updated.
1 update is a security update.

New release '20.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Mon Jun 12 14:17:18 2023 from 131.159.85.223
ubuntu@testcc:~$
```

Install docker and Kubernetes

1. Install packages to allow apt to use a repository over HTTPS

```
sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
software-properties-common
```

2. Add Docker's official GPG (GNU Privacy Guard) key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

3. Use the following command to set up the stable repository.

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

Install docker and Kubernetes Cont..

4. Switch to root user

```
sudo su root
```

5. Add Kubernetes repositories

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
```

6. Switch to the normal user

```
su <orginal user name>
```

7. Update the apt package index.

```
sudo apt-get update
```

8. Install the latest version of Docker by using this command.

```
sudo apt-get install -y docker-ce
```

9. Installation kubeadm, kubernetes and kubectl

```
export K8S_VERSION=1.23.17-00
sudo apt-get install –y cri-tools kubelet=$K8S_VERSION kubeadm=$K8S_VERSION kubectl=$K8S_VERSION kubernetes-cni
```

Install docker and Kubernetes Cont..

10. Change docker control group to systemd from cgroupfs

```
sudo vim /lib/systemd/system/docker.service
Add “--exec-opt native.cgroupdriver=systemd” to ExecStart
sudo systemctl daemon-reload
sudo systemctl restart docker
```

11. Installation kubeadm, kubernetes and kubectl

```
export K8S_VERSION=1.23.17-00
sudo apt-get install –y cri-tools kubelet=$K8S_VERSION
kubeadm=$K8S_VERSION kubectl=$K8S_VERSION kubernetes-cni
```

Installation

We will be using kubeadm to deploy the kubernetes Cluster.

- Install Docker, Kubernetes, Kubeadm and Kubectl on Master and worker nodes (As part of the exercise we are not using worker nodes)
- Check the Installation by running kubectl command, you would get something like this

```
ubuntu@testcc:~$ kubectl --help
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
  create        Create a resource from a file or from 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):
  explain       Get documentation for a resource
  get           Display one or many resources
  edit          Edit a resource on the server
  delete        Delete resources by file names, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout       Manage the rollout of a resource
  scale         Set a new size for a deployment, replica set, or replication controller
  autoscale     Auto-scale a deployment, replica set, stateful set, or replication controller

Cluster Management Commands:
  certificate   Modify certificate resources.
  cluster-info  Display cluster information
  top           Display resource (CPU/memory) 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

Troubleshooting and Debugging Commands:
  describe      Show details of a specific resource or group of resources
  logs          Print the logs for a container in a pod
  attach        Attach to a running container
  exec          Execute a command in a container
  port-forward  Forward one or more local ports to a pod
  proxy         Run a proxy to the Kubernetes API server
  cp            Copy files and directories to and from containers
  auth          Inspect authorization
  debug         Create debugging sessions for troubleshooting workloads and nodes
```

Step 2 - **Configuring Kubernetes**

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

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

```
ubuntu@testcc:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
I0612 15:40:40.477296   11284 version.go:256] remote version is much newer: v1.27.2; falling back to: stable-1.23
[init] Using Kubernetes version: v1.23.17
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 23.0.6. Latest validated version: 20.10
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local testcc] and IPs [10.96.0.1 192.168.130.79]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [localhost testcc] and IPs [192.168.130.79 127.0.0.1 ::1]
```

```
Your Kubernetes control-plane 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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

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/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.130.79:6443 --token a87oq5.i7n8dipmrt8337gr \
        --discovery-token-ca-cert-hash sha256:c92d556e421817737ff5f240f516cdff335808e8ae69bfadd72379a3df8b0f6f
```

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

Step 2 - **Configuring Kubernetes Cont..**

- Before going forward, you should create a new user and 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

```
ubuntu@testcc:~$ kubectl get nodes
NAME      STATUS   ROLES                  AGE   VERSION
testcc    Ready    control-plane,master   82m   v1.23.17
ubuntu@testcc:~$
```

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. To install an add-on run this command: Example: kubectl apply -f <add-on-name.yaml>
- We will be installing flannel, which provides networking and network policy.

kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml

```
ubuntu@testcc:~$ kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
namespace/kube-flannel created
serviceaccount/flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

Step 4 – **Status Check**

- Check the status of pods run the following command.

kubectl get pods --all-namespaces

```
ubuntu@testcc:~$ kubectl get pods --all-namespaces
NAMESPACE      NAME                              READY   STATUS    RESTARTS   AGE
kube-flannel   kube-flannel-ds-dvkn5             1/1     Running   0          2m14s
kube-system    coredns-bd6b6df9f-nbwwg           1/1     Running   0          12m
kube-system    coredns-bd6b6df9f-pb96s           1/1     Running   0          12m
kube-system    etcd-testcc                       1/1     Running   0          13m
kube-system    kube-apiserver-testcc             1/1     Running   0          13m
kube-system    kube-controller-manager-testcc    1/1     Running   0          13m
kube-system    kube-proxy-g476v                  1/1     Running   0          12m
kube-system    kube-scheduler-testcc             1/1     Running   0          13m
```

- You can also run this command to check the status of pods:

watch kubectl get pods –all-namespaces

It automatically gets refreshed after 2 seconds

- Check the status of node using the command kubectl get nodes

```
ubuntu@testcc:~$ kubectl get nodes
NAME     STATUS   ROLES                  AGE   VERSION
testcc   Ready    control-plane,master   14m   v1.23.17
```

Step 5 – **Joining the nodes**

- By default, your cluster will not schedule pods on the master for security reasons.
- If you want to be able to schedule pods on the master, e.g. a single-machine Kubernetes cluster for development, run the following command on master:

```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

- Worker nodes can be joined by running the kubeadm join command as taken note while doing kubeadm init on master node.

Kubernetes Workloads

- Workloads within Kubernetes are higher level objects that manage Pods or other higher level objects.

ReplicaSet

- Primary method of managing pod replicas and their lifecycle.
- Includes their scheduling, scaling, and deletion.
- Their job is simple: Always ensure the desired number of pods are running.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
template:
    <pod template>
```

Deployment

- Declarative method of managing Pods via ReplicaSets.
- Provide rollback functionality and update control.
- Updates are managed through the pod-template-hash label.

Pod Template

```yaml
template:
  metadata:
    labels:
      app:
nginx
  spec:

containers:
    - name:
nginx
```

Kubernetes Workloads Cont..

Deployment

- revisionHistoryLimit: The number of previous iterations of the Deployment to retain.
- strategy: Describes the method of updating the Pods based on the type. Valid options are
  - **Recreate:**
    All existing Pods are killed before the new ones are created.
  - **RollingUpdate:**
    Cycles through updating the Pods according to the parameters:
    - maxSurge: how many additional replicas to spin up while updating.
    - maxUnavailable: how many may be unavailable during the update.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3

revisionHistoryLimit:
3
  selector:
    matchLabels:
      app: nginx
strategy:
    type:
RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable:
0
  template:
    <pod template>
```

More..

Step 7 – **Running your containerized services**

- We create deployments for each service, the hello-world deployment file looks like : (kubernetes_files/deployments/hello-world.yml)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-deployment
  labels:
    app: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hello-world
          image: HUB_ID/microservice:hello
          ports:
            - containerPort: 9001
```

Type of workload

Name of deployment

Labels for referring

Specification about pod

Number of replicas

Template of the pod

Specification of the container

Container Name

Image name

Container Port

Step 7 – **Running your containerized images**

Before creating deployments :

- Add the image name in kubernetes_files/deployments/hello-world.yml file.
- Complete the missing product-descp.yml, product-price.yml and server.yml files.

After that run, the deployments for the microservices using the command

```
kubectl apply -f kubernetes_files/deployments/< file_name>.yml
```

Step 7 – **Running your containerized images Cont..**

- Check the status of all the pods in the deployments by running the command.

kubectl get pods --all-namespaces

```
ubuntu@testcc:~$ kubectl get pods --all-namespaces
NAMESPACE      NAME                                       READY   STATUS    RESTARTS      AGE
default        hello-world-deployment-7b7d996656-5s9hd    1/1     Running   0             103s
default        product-descp-deployment-6b4f9d8ff7-sswrg  1/1     Running   0             94s
default        product-price-deployment-79b99fddcd-52g5w  1/1     Running   0             86s
default        server-deployment-6cb7978998-rvmq5         1/1     Running   0             80s
kube-flannel   kube-flannel-ds-dvkn5                      1/1     Running   0             47m
kube-system    coredns-bd6b6df9f-nbwwg                    1/1     Running   0             57m
kube-system    coredns-bd6b6df9f-pb96s                    1/1     Running   0             57m
kube-system    etcd-testcc                                1/1     Running   0             58m
kube-system    kube-apiserver-testcc                      1/1     Running   0             58m
kube-system    kube-controller-manager-testcc             1/1     Running   1 (39m ago)   58m
kube-system    kube-proxy-g476v                           1/1     Running   0             57m
kube-system    kube-scheduler-testcc                      1/1     Running   1 (39m ago)   58m
```

- Check the status of deployments: kubectl get deployments --all-namespaces

```
ubuntu@testcc:~$ kubectl get deployments --all-namespaces
NAMESPACE      NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
default        hello-world-deployment     1/1     1            1           3m4s
default        product-descp-deployment   1/1     1            1           2m54s
default        product-price-deployment   1/1     1            1           2m47s
default        server-deployment          1/1     1            1           2m41s
kube-system    coredns                    2/2     2            2           59m
```

Step 7 – **Running your containerized images Cont..**

- As all the microservices are running in different pods so we need to create kube-services for each of them to complete the interaction.
- All the kube-services are in (kubernetes_files/services/)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-world-service
spec:
  selector:
    app: hello-world
  ports:
    - protocol: TCP
      port: 9001
      targetPort: 9001
```

Type of workload

Name of the kube-service. It is same as in the docekr-compose.yml file for last exercise

Name of the pod to connect this with.

Container Port and VM port

Step 7 – **Running your containerized images Cont..**

Before creating services :

- Complete the missing product-descp.yml, product-price.yml files in kubernetes_files/services/.
- We expose our server microservice to the outside world so ist type is **LoadBalancer**.

After that, run the kube-services for the microservices using the command :

```
kubectl apply -f kubernetes_files/services/<file_name>.yml
```

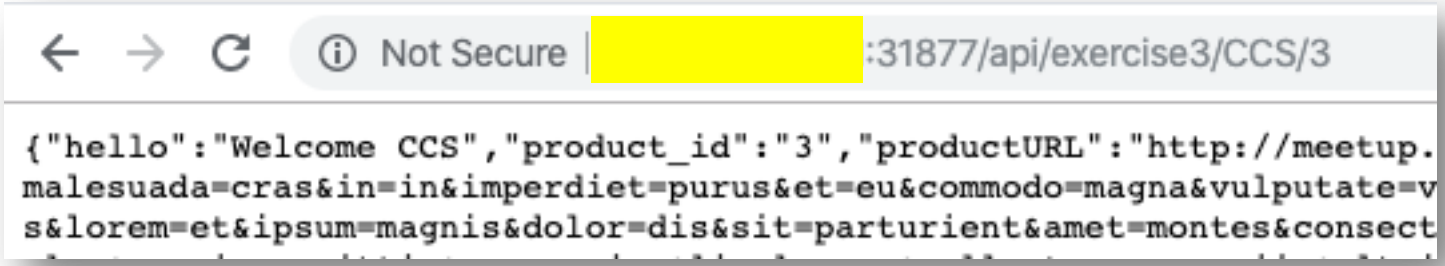
Step 7 – **Running your containerized images Cont..**

- Can the kube-services by running the command

kubectl get services --all-namespaces

```
ubuntu@testcc:~$ kubectl get services --all-namespaces
NAMESPACE     NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                  AGE
default       hello-world-service     ClusterIP      10.99.6.54       <none>        9001/TCP                 50s
default       kubernetes              ClusterIP      10.96.0.1        <none>        443/TCP                  63m
default       product-descp-service   ClusterIP      10.103.130.156   <none>        9002/TCP                 43s
default       product-price-service   ClusterIP      10.98.0.93       <none>        9003/TCP                 40s
default       server-service          LoadBalancer   10.98.50.18      <pending>     3000:32013/TCP           34s
kube-system   kube-dns                ClusterIP      10.96.0.10       <none>        53/UDP,53/TCP,9153/TCP   62m
```

- Here the external-IP is your **VM public IP** and the port number is **32013**.
- Your Application would be running at address http://VM_IP:PORTNUMBER/api/exercise3/CCS/3
- http://VM_IP:PORTNUMBER/api/exercise4



Step 8 – **Scaling your deployment**

- Before Scaling

```
ubuntu@testcc:~$ kubectl get deployment server-deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
server-deployment   1/1     1            1           15m
```

- Scaling can be done by following command:

kubectl scale deployment <deployment_name> --replicas=<replicaNumber>

```
ubuntu@testcc:~$ kubectl get deployment server-deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
server-deployment   2/2     2            2           17m
```

Deleting and Resetting the Cluster

Do this step only if you need to redeploy the Kubernetes cluster or some workloads.

- 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

```
kubeadm reset
```

- Delete the configuration file

```
sudo rm -r $HOME/.kube/config
```

Tasks to be Completed

Tasks to be completed

As part of the exercise4, following are the tasks to be completed:

1. Add an API endpoint in your Server Microservice and push the images to docker-hub:
   1. **/api/exercise4: Send a message "group # application deployed using kubernetes"**
2. Install docker and Kubernetes on the VM.
3. After installation run this application on the VM using Kubernetes as explained in previous slides:
   a. Start Kubernetes Cluster
   b. Install Pod Network
   c. Enable Pod Scheduling on Master node.
   d. Run all microservices deployments.
   e. Create kube services for all the microservices.
   f. Scale Server microservice to have 2 replicas
   g. Expose Kubernetes API : sudo kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='^*$'&
   h. Check the status and port number of microservices.
   i. Visit the URL to test the application: http://YOUR_VM_IP:PORTNUMBER/api/exercise3/CCS/3

http://YOUR_VM_IP:PORTNUMBER/api/exercise4

Submission

Submission Instructions

To submit your application results you need to follow this :

1. Open the Cloud Class server url : https://cloudcom.caps.in.tum.de/
2. Login with your provided username and password.
3. After logging in, you will find the button for **exercise4**
4. Click on it and a form will come up where you must provide
   - VM ip on which your application is running
   - 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).

Deadline for submission: Check the submission server

Thank you for your attention!