Cloud Computing – WS 2017

Exercise 4 : Kubernetes - Container Orchestration

19th December 2017

Anshul Jindal

anshul.jindal@tum.de

Index

Exercise 3 Solution

1.Complete the microservices **product-description-service(to get product name and URL)** and **product-price-service(to get product price),**

```
module.exports = function (options) {
    //Import the mock data json file
    const mockData = require('./MOCK_DATA.json');
    //Add the patterns and their corresponding functions
    this.add('role:product,cmd:getProductPrice', productPrice);
    //Describe the logic inside the function
    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: ''});
        }

    }
}
```

Pattern

Action

Import the data file

Add the pattern and Corresponding Function

Add the function logic
To search for the product based upon the ID

Respond back with the Product Price

product-price-service/product_price.js

```
module.exports = function (options) {
    //Import the mock data json file
    const mockData = require('./MOCK_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: ''});
        }

    }
    //Describe the logic inside the function
    function productName(msg, respond) {
```

Pattern

Action

Import the data file

Add the patterns and Corresponding Functions

Add the function logic
To search for the product based upon the ID

Respond back with the Product URL

Same Logic for Product Name function

product-descp-service/product_descp.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 }));
};
module.exports = {
    getProductPrice
};
```

Pattern

Service Method with Same pattern

Function to call Service method with the Product Id as the argument

server/services/product_price.js

```
/**
 * Service Method
 */
const GET_PRODUCT_URL = { role: 'product', cmd: 'getProductURL' };
const GET_PRODUCT_NAME = { role: 'product', cmd: 'getProductName' };
/**
 * Call Service Method
 */
const getProductURL = (productId) => {
    return act(Object.assign({}, GET_PRODUCT_URL, { productId }));
};
const getProductName = (productId) => {
    return act(Object.assign({}, GET_PRODUCT_NAME, { productId }));
};
module.exports = {
    getProductURL,
    getProductName
};
```

Pattern

Service Method with Same pattern

Functions to call Service method with the Product Id as the argument

server/services/product_descp.js

```yaml
version: '2'
services:
  server:
    build: ./server
    image: ansjin/microservice:server
    ports:
      - "8080:8080"
  hello-world-service:
    build: ./hello-world-service
    image: ansjin/microservice:hello
  product-descp-service:
    build: ./product-descp-service
    image: ansjin/microservice:productdescp
  product-price-service:
    build: ./product-price-service
    image: ansjin/microservice:productprice
```

Server Service

Exposed to port 8080

hello-world-service

product-descp-service

product-price-service

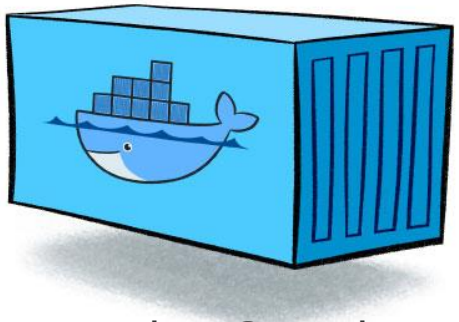
docker-compose.yml

Solution Demo

Exercise 4

Exercise 4 : Kubernetes - Container Orchestration

Introduction: What is Container Orchestration ?



A Docker Container

Single VM

Container Deployment

Many Docker Containers

100s of VMs

?

How to deploy and manage them ?

This management and deployment of containers is called Container Orchestration

Introduction: Benefits of Container Orchestration

- **Management of multiple containers** : Logically Grouping of containers into one entity.
- **Container placement :** Selects a specific host for a specific container or a set of containers using different rules.
- **Container Scaling** : Scaling of containers up or down based upon the requirements.
- **Resource usage monitoring** : Resource usage like CPU and RAM is required at different levels – at the container level, at the logical group level and at the cluster level.
- **Health Checks:** Used to check container's liveness or readiness status.
- **Networking:** isolate independent containers, connect coupled containers and provide access to containers from external clients (Remember **Service discovery**)

Introduction: Different Container Orchestration Tools

Amazon Elastic Container Service (Amazon ECS)

Azure Container Service

Introduction: What is Kubernetes ?

- The Kubernetes project was started by Google in 2014.
- It is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.
- It is one of the most feature-rich and widely used orchestration frameworks.
- It's key features include:
  - Automated deployment and replication of containers
  - Online scale-in or scale-out of container clusters
  - Load balancing over groups of containers
  - Rolling upgrades of application containers
  - Resilience, with automated rescheduling of failed containers
  - Controlled exposure of network ports to systems outside of the cluster

Introduction: Kubernetes Architecture

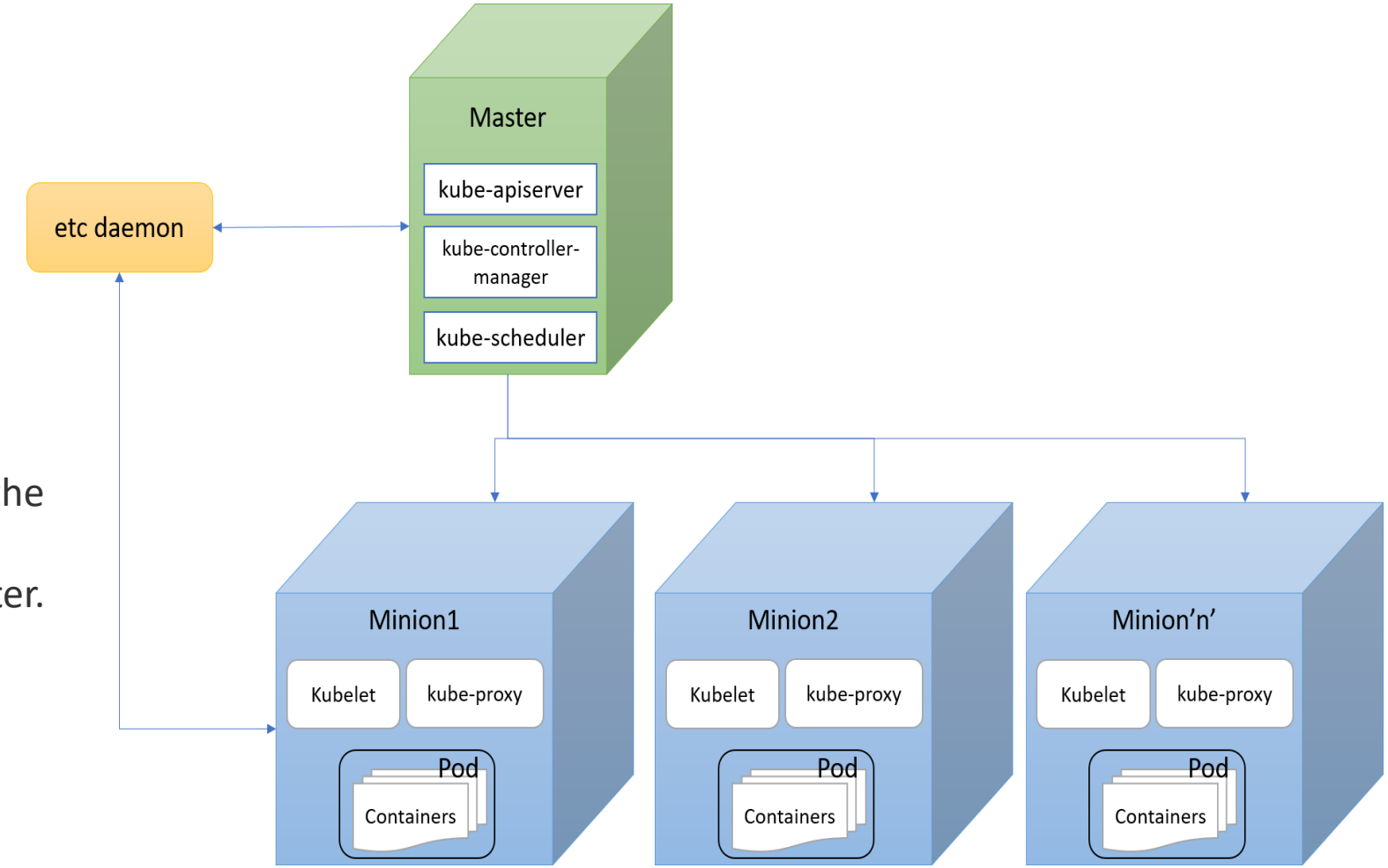
- Based on master-slave Architecture

Master: The server that runs the Kubernetes management processes, including the API service, Controller Manager and scheduler.

Minion(Slave Nodes): The host that runs the kubelet service and the Docker Engine. Minions receive commands from the master.

etcd: etcd is used as Kubernetes' backing store. All cluster data is stored here.

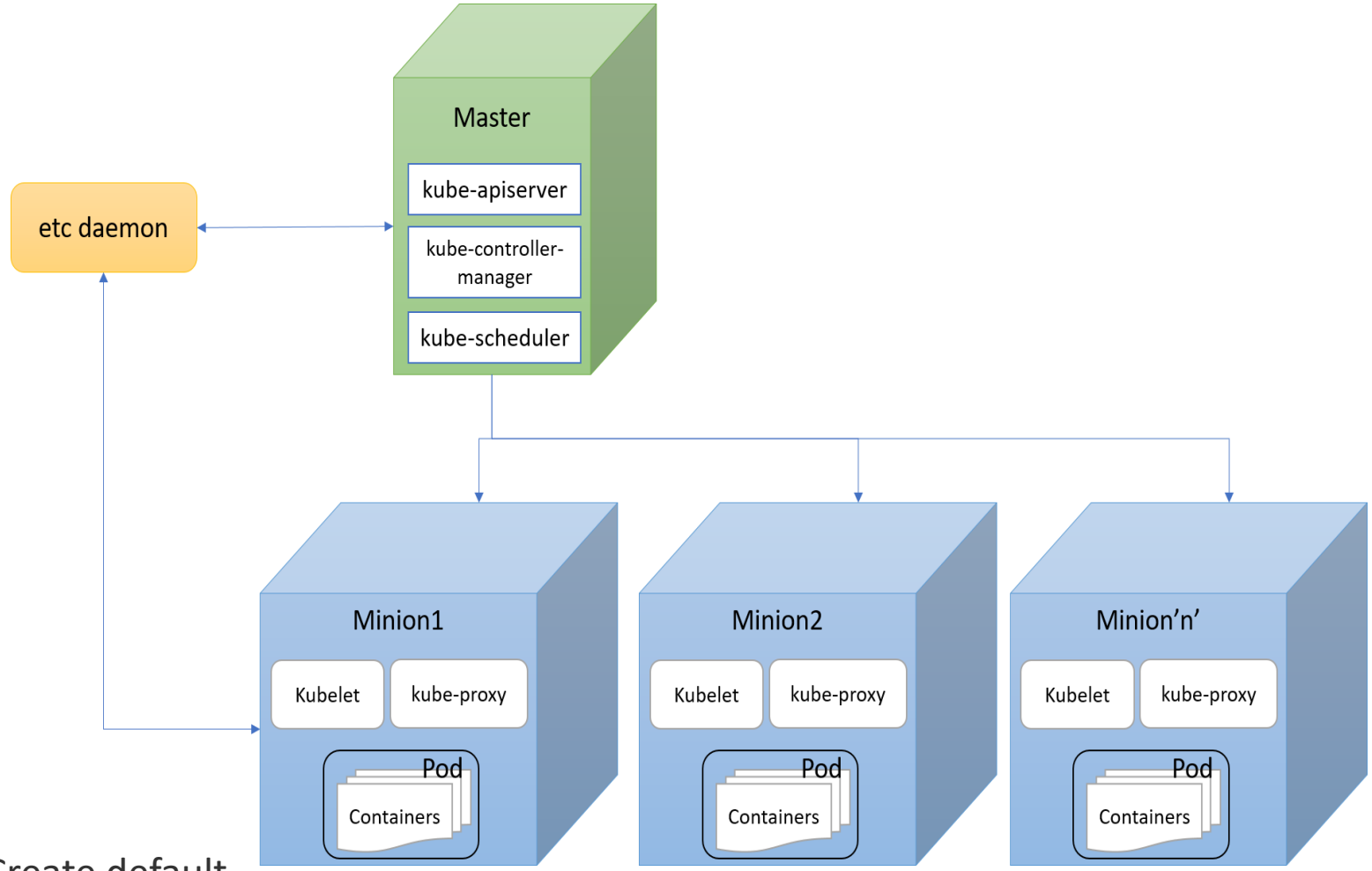
kube-apiserver: kube-apiserver exposes the Kubernetes API;



Introduction: Kubernetes Architecture

Kube-controller-manager:
kube-controller-manager is a binary that runs controllers, which are the background threads that handle routine tasks in the cluster.
These controllers include:

- **Node Controller:** Responsible for noticing & responding when nodes go down.
- **Replication Controller:** Responsible for maintaining the correct number of pods for every replication controller object in the system.
- **Endpoints Controller:** Populates the Endpoints object (i.e., join Services & Pods).
- **Service Account & Token Controllers:** Create default accounts and API access tokens for new namespaces.



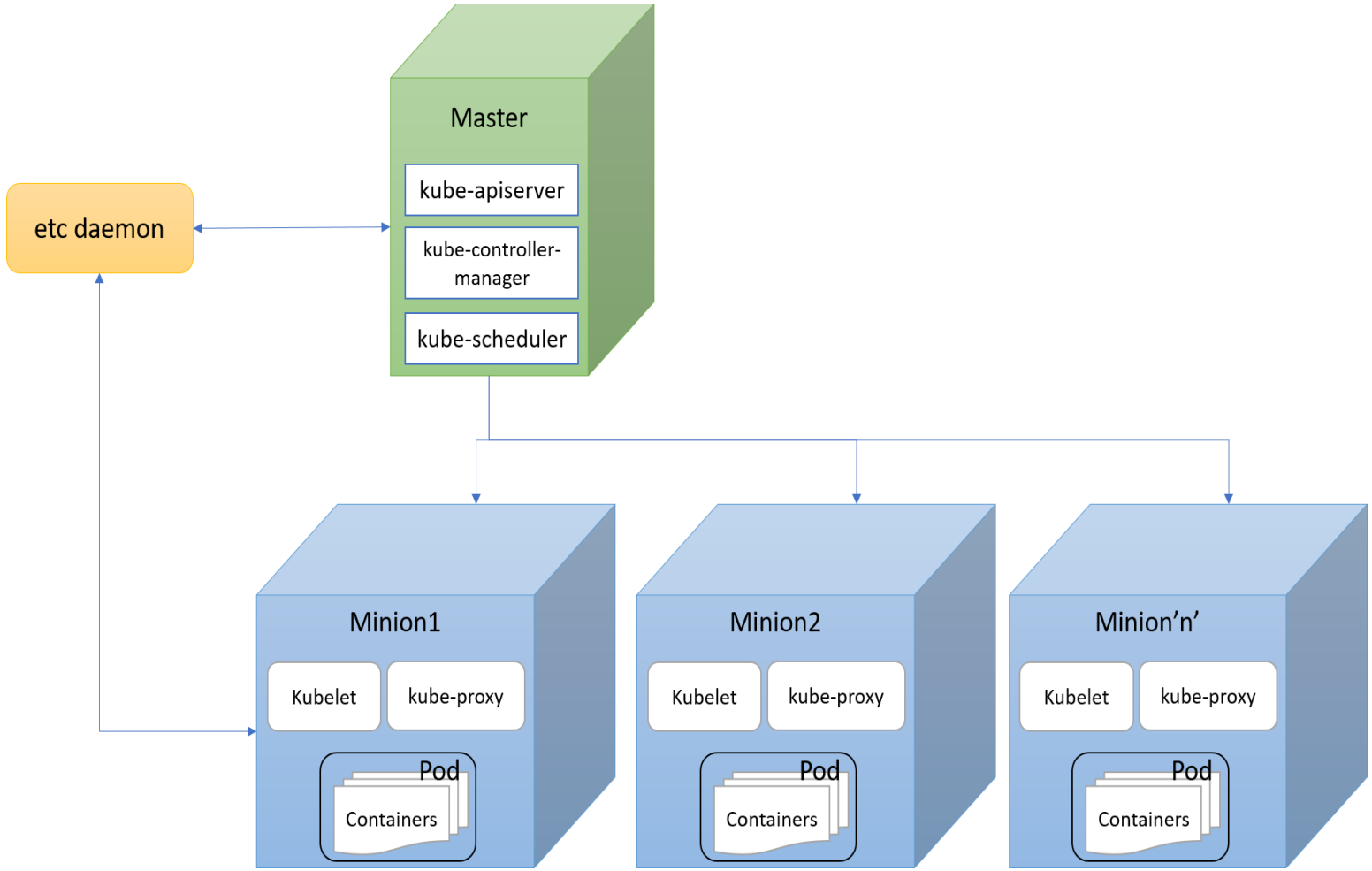
Introduction: Kubernetes Architecture

kube-scheduler: kube-scheduler watches newly created pods that have no node assigned, and selects a node for them to run on.

Kubelet: The node-level manager in Kubernetes; it runs on a minion.

Kube-proxy: kube-proxy enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding

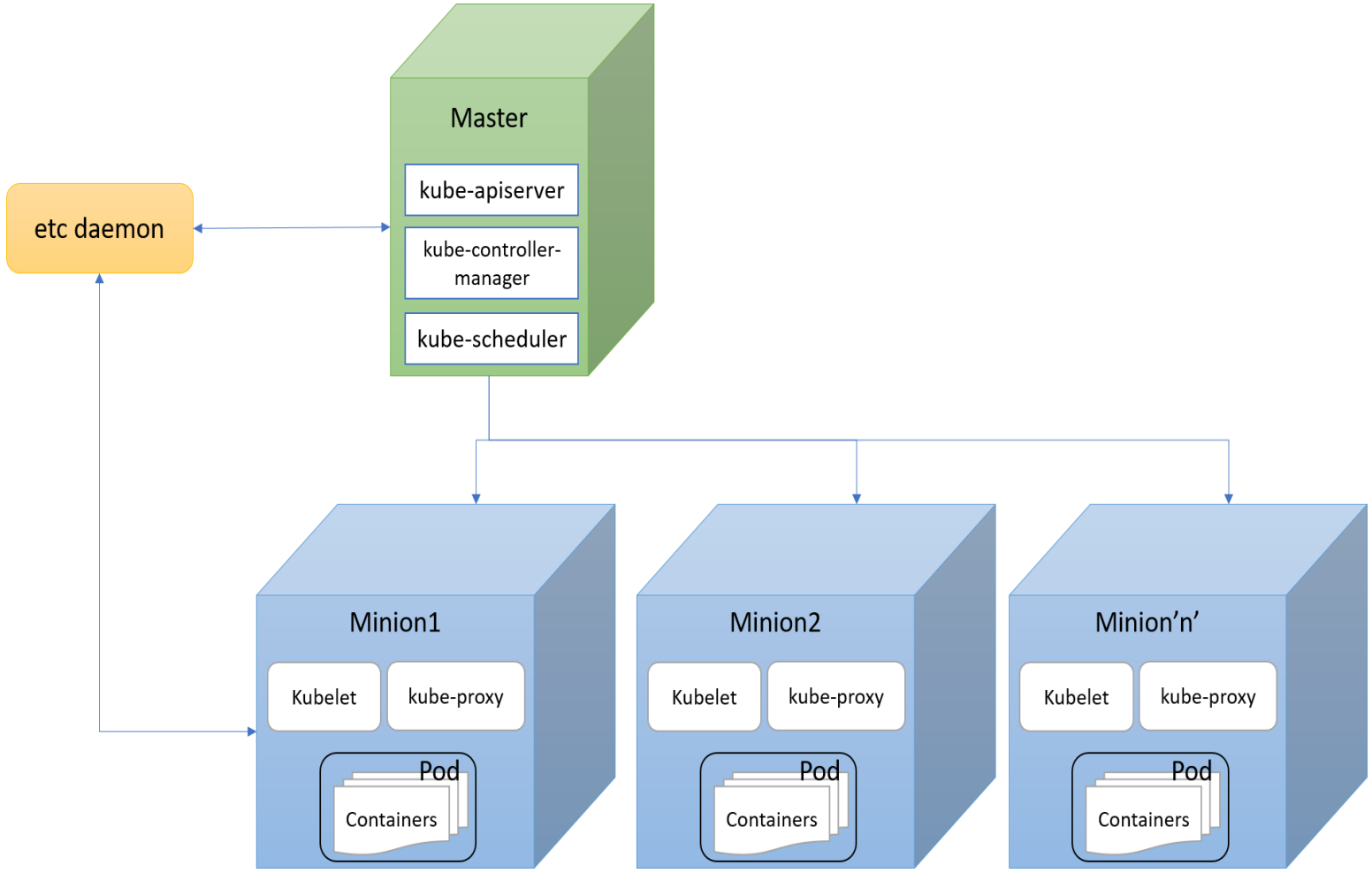
Pod(logical Grouping): The collection of containers deployed on the same minion.



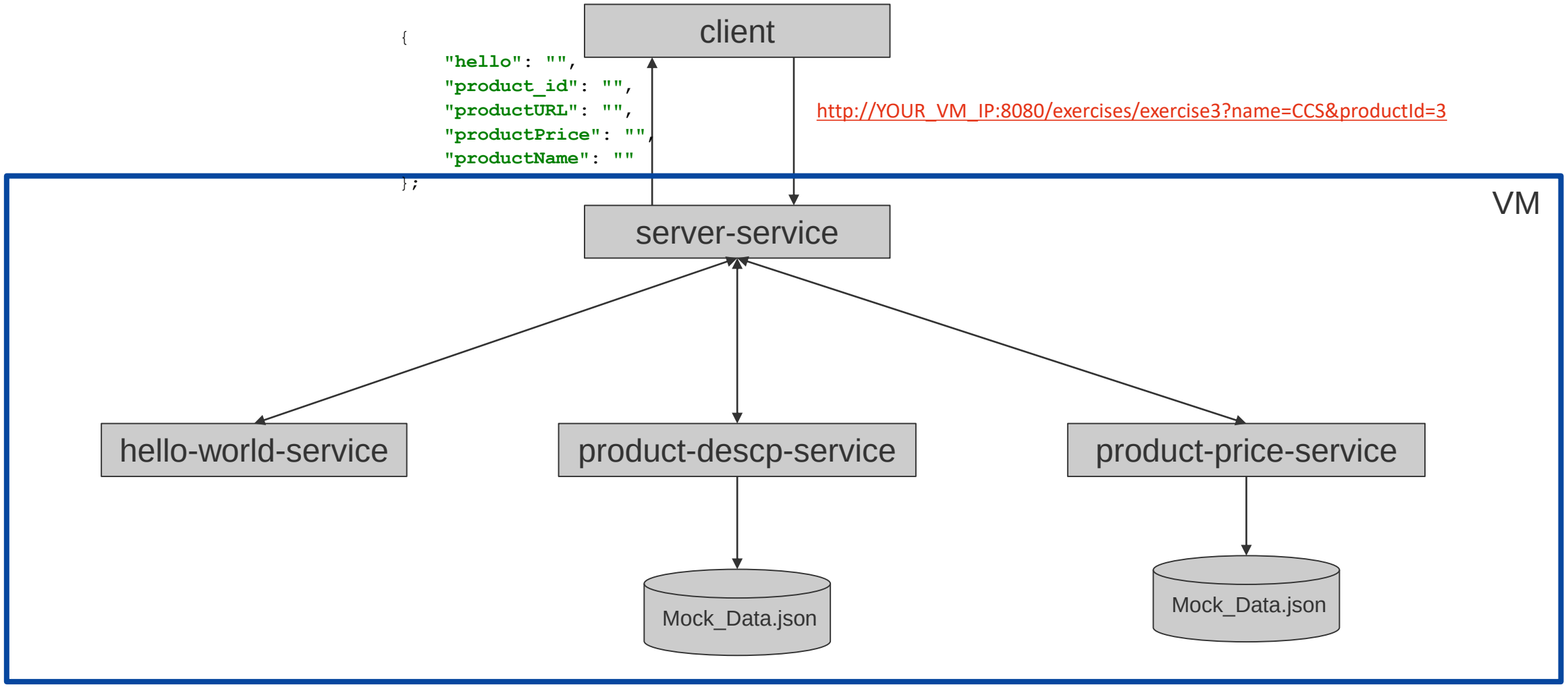
Introduction: Kubernetes Architecture

Services: Acts as front end and load balancer for pods, providing a floating IP for access to the pods.

Labels: key-value tags (e.g. “Name”) that you and the system use to identify pods, replication controllers and services.



Exercise 3: Microservice Architecture



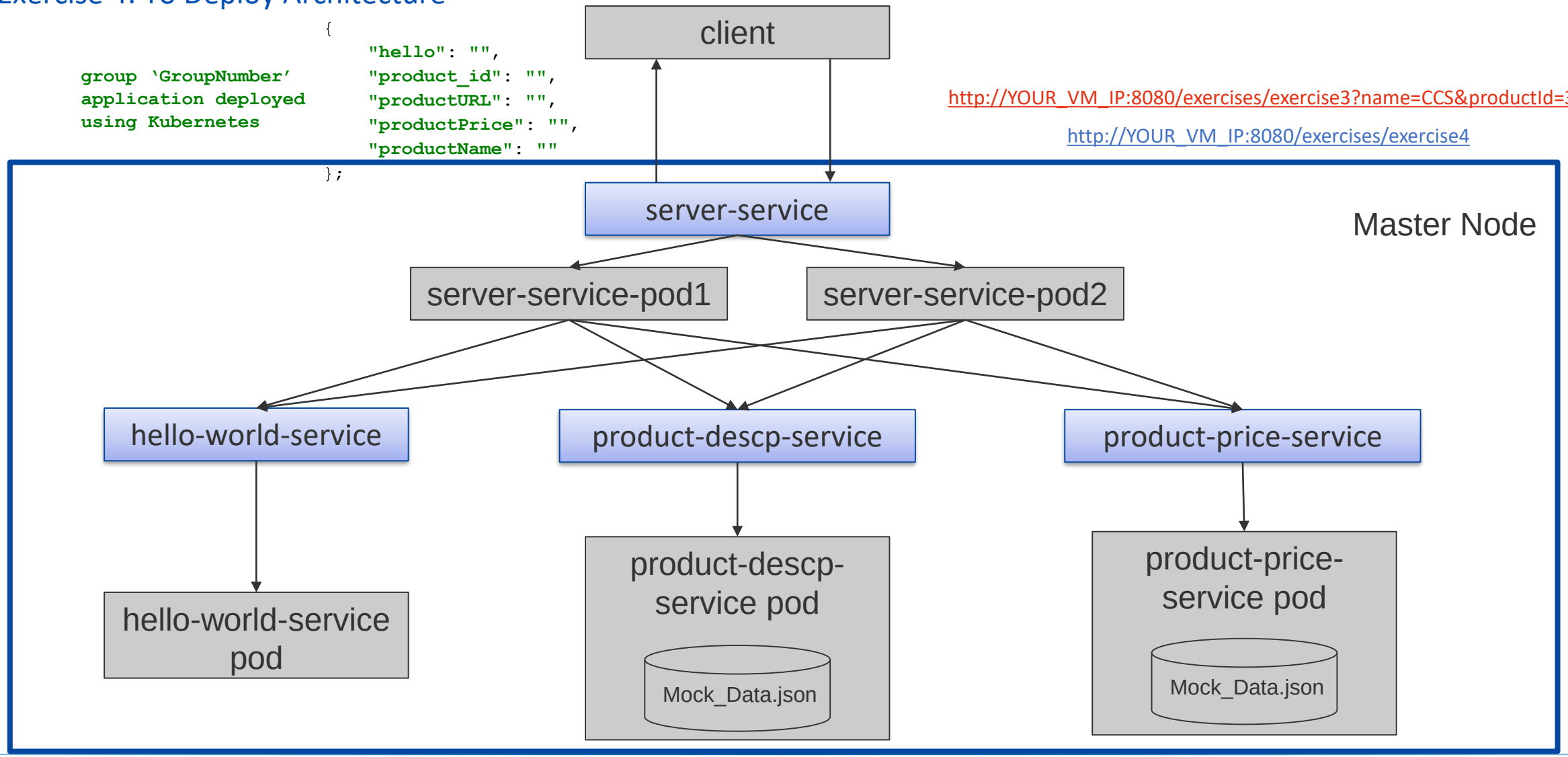
Exercise 4: To Deploy Architecture

```
{
    "hello": "",
    "product_id": "",
    "productURL": "",
    "productPrice": "",
    "productName": ""
};
```

group 'GroupNumber'
application deployed
using Kubernetes

http://YOUR_VM_IP:8080/exercises/exercise3?name=CCS&productId=3

http://YOUR_VM_IP:8080/exercises/exercise4



Exercise 4: 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 4: 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 on the slave nodes for joining the kubernetes cluster

Exercise 4: 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 4: 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 4: 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
```

Exercise 4: Step 5 – **Accessing Kubernetes Dashboard**

- To access the dashboard of kubernetes, firstly access control permissions need to be provided. These access permissions can be studied here.
- But for the exercise we will be providing full admin access control by using the **dashboard-admin.yaml** file (provided in the source code) to access dashboard.
- Run the following command from inside the source directory.

```
kubectl create -f dashboard-admin.yaml
```

- Install the dashboard add on by running the following command

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

- The easiest way to access Dashboard is to use kubectl.

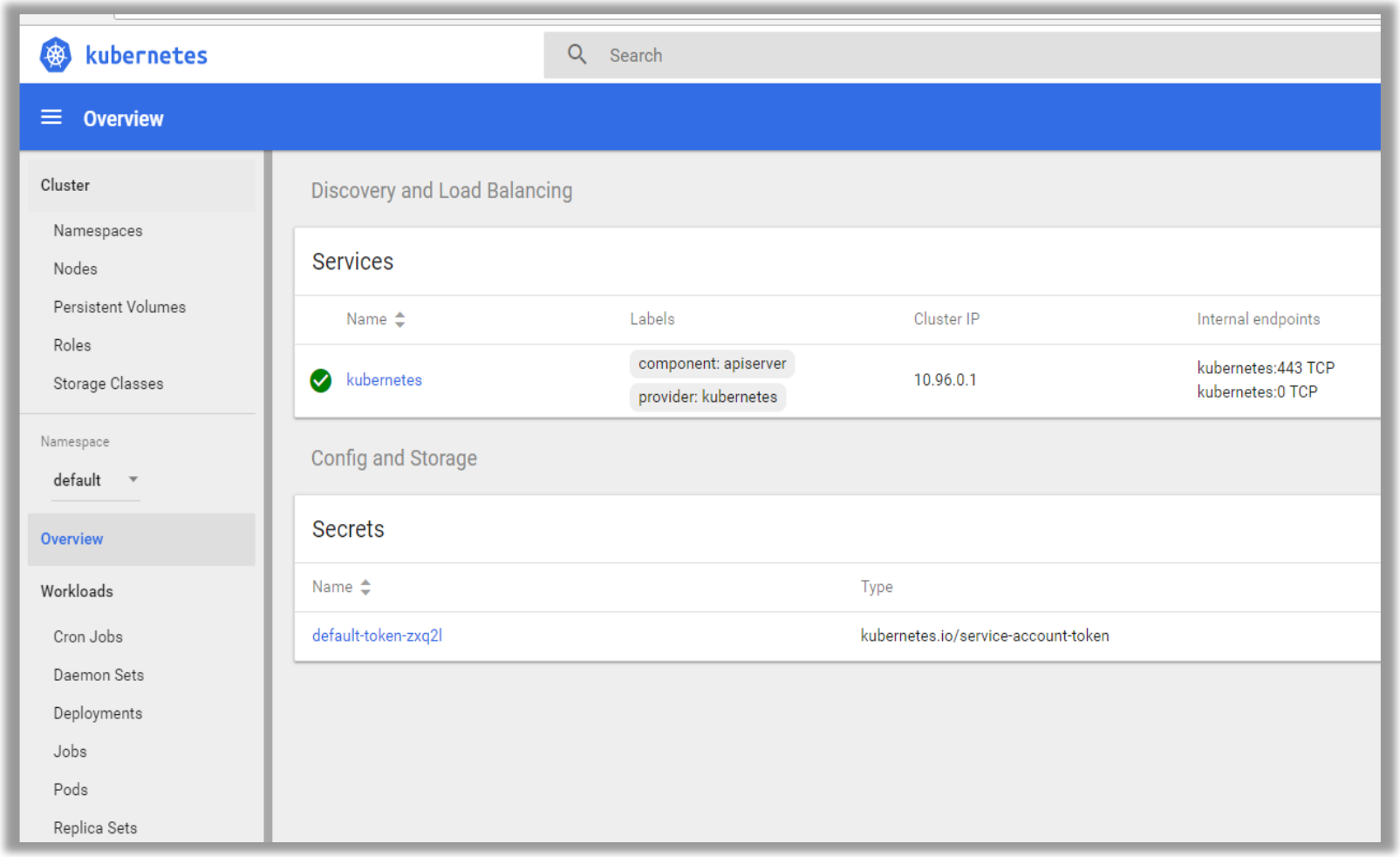
```
sudo kubectl proxy --address='0.0.0.0' --port=8001 --accept-hosts='^*$'&
```

- Dashboard is available at

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

Exercise 4: Step 5 – **Accessing Kubernetes Dashboard**

- Click on the skip as we are have provided admin access control.



Exercise 4: Step 6 – **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-
```

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

Exercise 4: Step 7 – **Running your containerized image**

- Deploy the images of microservices from previous exercise using the following command:

kubectl run SERVICENAME --image=docker.io/IMAGEADDRESS --port=PORT

```
ansjin@vm-10-155-208-213:/$ kubectl run hello-world-service --image=docker.io/ansjin/microservice:hello --port=
9001
deployment "hello-world-service" created
ansjin@vm-10-155-208-213:/$ kubectl run product-descp-service --image=docker.io/ansjin/microservice:productdesc
p --port=9002
deployment "product-descp-service" created
ansjin@vm-10-155-208-213:/$ kubectl run product-price-service --image=docker.io/ansjin/microservice:productpric
e --port=9003
deployment "product-price-service" created
ansjin@vm-10-155-208-213:/$ kubectl run server --image=docker.io/ansjin/microservice:server --port=8080
deployment "server" created
ansjin@vm-10-155-208-213:/$
```

Exercise 4: Step 7 – **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 4: Step 7 – **Running your containerized image**

- As all the services are running in different pods so we need to create services for each of them to complete the interaction. This can be done by following command:

  kubectl expose deployment <deploymentName> --port=<portNumber>

- We want the server service to be accessed from outside the Cluster. So that need to be exposed to Internet.

  kubectl expose deployment <deploymentName> --port=<portNumber> --type=LoadBalancer

```
ansjin@vm-10-155-208-213:/$ kubectl expose deployment  product-descp-service --port=9002
service "product-descp-service" exposed
ansjin@vm-10-155-208-213:/$ kubectl get svc --all-namespaces
NAMESPACE     NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
default       hello-world-service    ClusterIP      10.101.168.134   <none>        9001/TCP         3m
default       kubernetes             ClusterIP      10.96.0.1        <none>        443/TCP          3h
default       product-descp-service  ClusterIP      10.99.204.194    <none>        9002/TCP         5s
default       server                 LoadBalancer   10.104.32.226    <pending>     8080:32400/TCP   2h
kube-system   kube-dns               ClusterIP      10.96.0.10       <none>        53/UDP,53/TCP    3h
kube-system   kubernetes-dashboard   ClusterIP      10.102.132.97    <none>        443/TCP          3h
ansjin@vm-10-155-208-213:/$ kubectl expose deployment  product-price-service --port=9003
```

Exercise 4: Step 7 – **Running your containerized image**

- 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 4: Step 8 – **Scaling your deployment**

- This can be done by following command:

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

```
ansjin@vm-10-155-208-213:/$ kubectl get deployment server
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
server    2         2         2            2           2h
```

Exercise 4: Step 9 – **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

```
kubeadm reset
```

Short Demo

Tasks To be Completed

Tasks to be completed

As part of the exercise3, following are the tasks to be completed:

1. Install docker on the VM.
2. Install Kubernetes as explained in detail document.
3. Run the Kubernetes Cluster
4. Install the Pod network
5. Enable pod Scheduling on Master as explained in **section 2.2.3 of detailed document.**
6. Add a API endpoint in your Server Microservice with the name
    a. **/exercises/exercise4:** Send a message "group ‘GroupNumber’ application deployed using Kubernetes".
7. Run all 4 Microservices inside Kubernetes

Deadline for submission: 8th January 2018 23:59

Tasks to be completed

8. Create services for all the 4 Microservices
   a. Expose **3 Microservices** (Hello_service, Product_Descp, Product_Price) inside the Cluster as explained in **section 2.3 4th Point.**
   b. Expose **4th Microservice** (Server), so that it can be accessed from outside the Cluster.
9. Scale your **Server Microservice** to have **2 Replicas**.
10. 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.
11. 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
12. 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: 8th 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 **exercise3**
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☺

A Chance to Earn a Extra Point

For those who have missed the first exercise or those who already knew Kubernetes and of course for others too!

Create a **single YAML file** for the whole deployment procedure(including dashboard deployment with admin access) stated above and check if it works exactly the same as the above commands. If yes then send me the YAML file, you may get a extra point if it is found correct.

*Per Group only one chance to send the file.

Thank you for your attention! ☺

Questions?

Appendix

Install the node and npm

- Get yourself a more recent version of Node.js by adding a PPA (personal package archive) maintained by NodeSource.

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
```

- You can now install the Node.js package.

```
sudo apt-get install nodejs
```

The nodejs package contains the nodejs binary as well as npm, so you don't need to install npm separately.

- For some npm packages to work (such as those that require building from source), you will need to install the build-essentials package

```
sudo apt-get install build-essential
```

- Test Node: `node -v` (This should print a version number, so you'll see something like this v0.10.35)
- Test NPM: `npm -v` (This should print NPM's version number so you'll see something like this 1.4.28)

Installation of required modules

1. As part of this application some modules need to be installed. For installing them run the following command from inside the directory of application.

   npm install

   This command will install all the dependent modules mentioned in the package.json file.

2. If you need some other modules you can install them by running the command

   npm install “module name” –save

   This will automatically add that module in the package.json file and now you can use it inside you development file.

Node.js Client Application Deployment

1. Now your application is ready to be deployed on VM. Run the following command to start the application:

   node clientApplication.js

   After running this, on console you will see

   “Server started and listening on port 8080”

2. Now your application is deployed on the server. Open your browser on your local machine and enter the address as

   http://IP_ADDRESS_VM:8080/exercises

```
'Welcome to Cloud Computing Exercises API
```

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