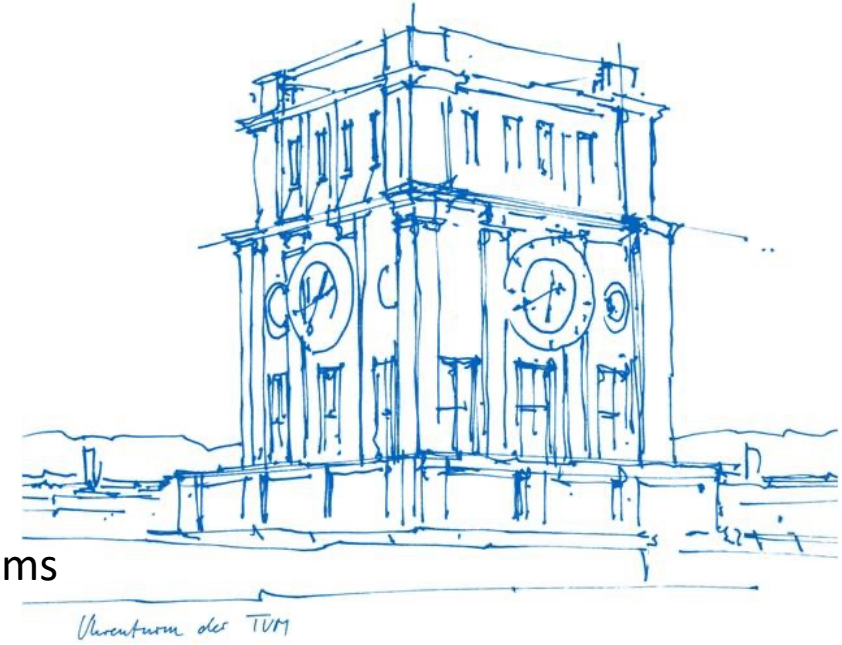
Cloud Computing Exercise – 5

Application Deployment using OpenWhisk



Anshul Jindal (M.Sc. Informatics)

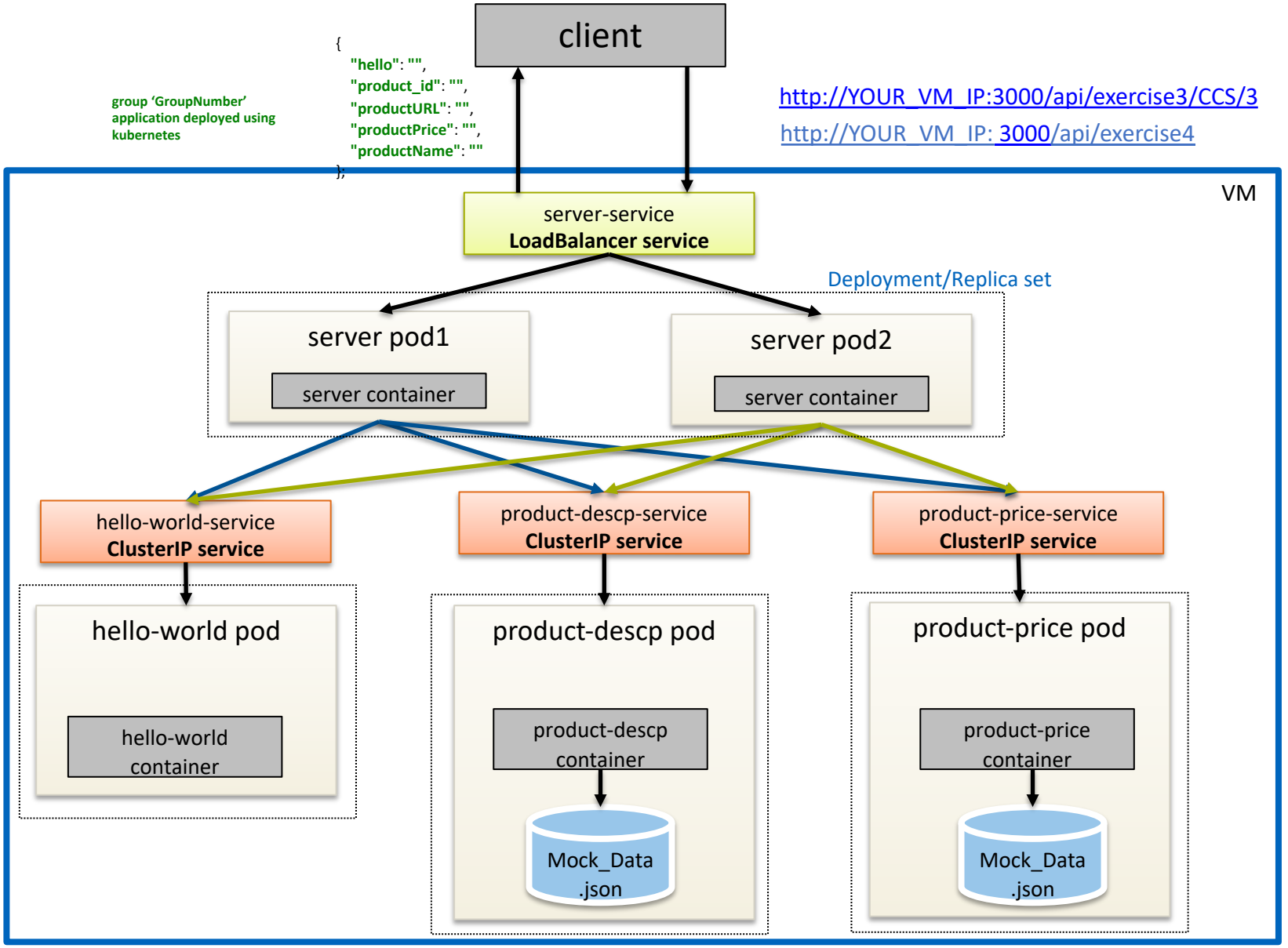
anshul.jindal@tum.de

Chair of Computer Architecture and Parallel Systems

Technical University of Munich (TUM), Germany

Exercise 4 Solution

Exercise 4 Application Architecture



kubernetes_files/deployments/product-descp.yml

- kubernetes_files/deployments/product-descp.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-descp-deployment
  labels:
    app: product-descp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product-descp
  template:
    metadata:
      labels:
        app: product-descp
    spec:
      containers:
        - name: product-descp
          image: HUB_ID/microservice:productdescp
          ports:
            - containerPort: 9002
```

Type of workload

Name of deployment

Labels for reference

Specification about pod

Number of replicas

Template of the pod

Specification of the container

Container Name

Image name

Container Port

Kube-Services – product-descp.yml

- kubernetes_files/services/product-descp.yml

```
apiVersion: v1
kind: Service
metadata:
  name: product-descp-service
spec:
  selector:
    app: product-descp
  ports:
    - protocol: TCP
      port: 9002
      targetPort: 9002
```

Service Type

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

Name of the pod to connect this with.

Container Port and VM port

kubernetes_files/deployments/server.yml

- kubernetes_files/deployments/server.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-deployment
  labels:
    app: server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: server
  template:
    metadata:
      labels:
        app: server
    spec:
      containers:
        - name: server
          image: HUB_ID/microservice:server
          ports:
            - containerPort: 3000
```

Type of workload

Name of deployment

Labels for reference

Specification about pod

Number of replicas

Template of the pod

Specification of the container

Container Name

Image name

Container Port

Kube-Services – server.yml

- kubernetes_files/services/server.yml

```
apiVersion: v1
kind: Service
metadata:
  name: server-service
spec:
  selector:
    app: server
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: LoadBalancer
```

Service Type

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

Exercise 5: Introduction

Sample Serverless Function

- Creating a sample serverless function (hello-world):

```
/**
 * Hello world as an OpenWhisk action.
 */
function main(params) {
  var name = params.name || 'World';
  return {payload:  'Hello, ' + name + '!'};
}
```

- Save to a file : hello.js

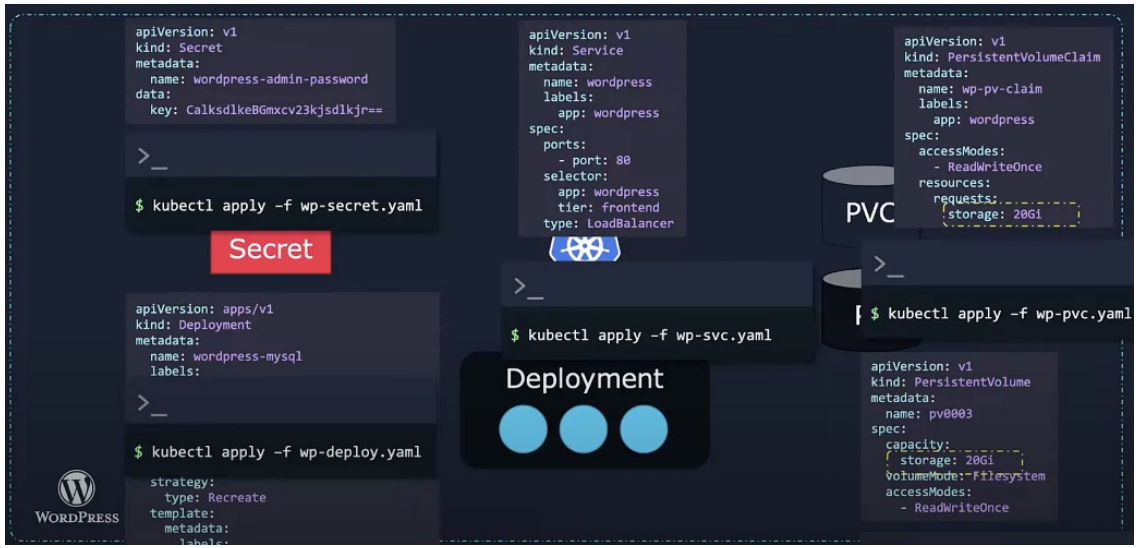
OpenWhisk CLI (wsk)

- It is a unified tool that provides a consistent interface to interact with OpenWhisk services.
- GitHub: https://github.com/apache/openwhisk-cli
- There are two required properties to configure in order to use the CLI:
  - **API host (name or IP address)** for the OpenWhisk deployment you want to use.
  - **Authorization key (username and password)** which grants you access to the OpenWhisk API.
- Example:

```
wsk property set --apihost 10.11.11.11:31001
wsk property set --auth 23bc46b1-71f6-4ed5-8c54-816aa4f8c502:123zO3xZCLrMN6v2BKK1dXYFpXlPkccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIwP
```

- Other Commands:
  - **Create action :** `wsk -i action create hello hello.js`
  - **Invoke action :** `wsk -i action invoke hello --result`
  - **Invoke action with parameter:** `wsk -i action invoke hello --result --param name CCS`
- More Details: https://github.com/apache/openwhisk/blob/master/docs/cli.md

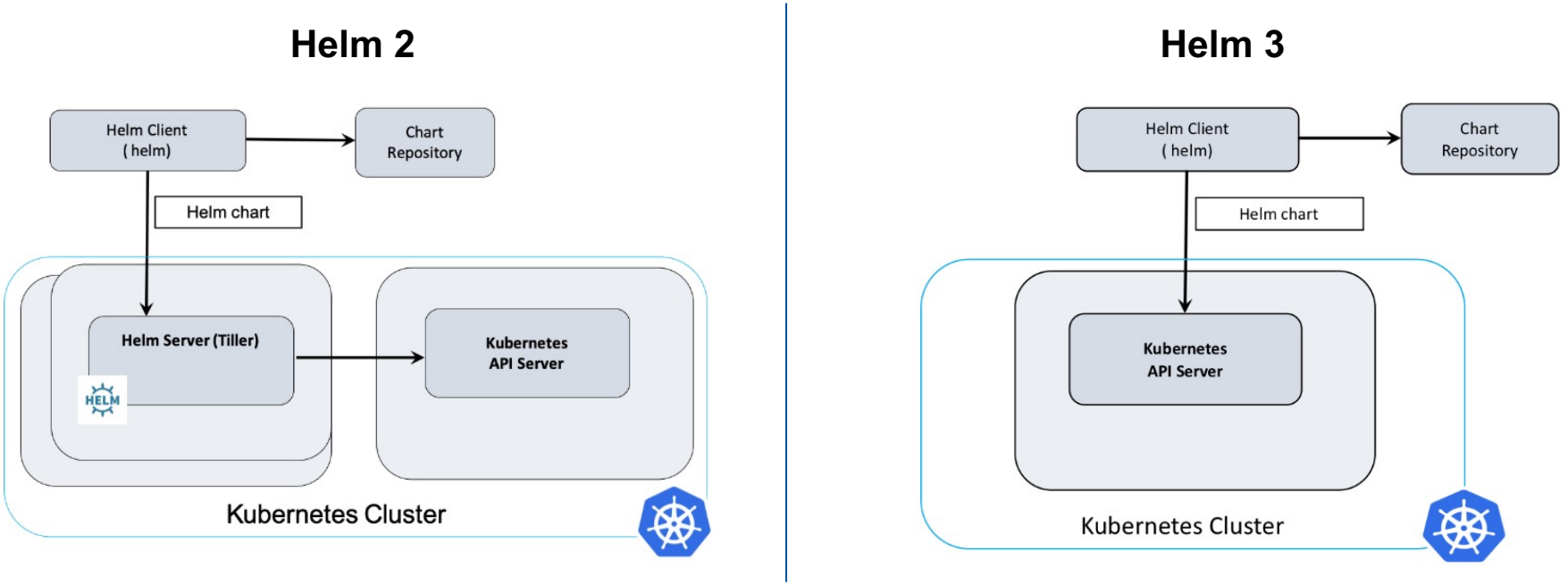
Helm



- Helm is a package manager for Kubernetes that simplifies the management of Kubernetes applications
- Helm is organized around several key concepts:
  - **The chart** is a bundle of information necessary to create an instance of a Kubernetes application.
  - **The config** contains configuration information that can be merged into a packaged chart to create a releasable object.
  - **A release** is a running instance of a chart, combined with a specific config.

Helm Architecture

- Helm consists of two main components:
  - **Helm Client (helm)** – allows developers to create new charts, manage chart repositories, and interact with the tiller server.



Source: https://developer.ibm.com/blogs/kubernetes-helm-3/

  - **The Helm Library** provides the logic for executing all Helm operations. It interfaces with the Kubernetes API server.

OpenWhisk Conductor Actions

- Conductor actions make it possible to build and invoke a series of actions, similar to sequences.

triple.js

```
function main({ value }) { return { value: value * 3 } }
```

```
wsk action create triple triple.js
```

increment.js

```
function main({ value }) { return { value: value + 1 } }
```

```
wsk action create increment increment.js
```

tripleAndIncrement.js

```
function main(params) {
    let step = params.$step || 0
    delete params.$step
    switch (step) {
        case 0: return { action: 'triple', params, state: { $step: 1 } }
        case 1: return { action: 'increment', params, state: { $step: 2 } }
        case 2: return { params }
    }
}
```

```
wsk action create tripleAndIncrement tripleAndIncrement.js -a conductor true
```

OpenWhisk Conductor Actions Cont..

```
wsk action invoke tripleAndIncrement -r -p value 3
```

```
{
    "value": 10
}
```

- Activations list

```
Datetime            Activation ID                  Kind      Start Duration   Status   Entity
2019-03-16 20:03:00 8690bc9904794c9390bc9904794c930e nodejs:6  warm  2ms        success  guest/tripleAndIncrement:0.0.1
2019-03-16 20:02:59 7e76452bec32401db6452bec32001d68 nodejs:6  cold  32ms       success  guest/increment:0.0.1
2019-03-16 20:02:59 097250ad10a24e1eb250ad10a23e1e96 nodejs:6  warm  2ms        success  guest/tripleAndIncrement:0.0.1
2019-03-16 20:02:58 4991a50ed9ed4dc091a50ed9edddc0bb nodejs:6  cold  33ms       success  guest/triple:0.0.1
2019-03-16 20:02:57 aee63124f3504aefa63124f3506aef8b nodejs:6  cold  34ms       success  guest/tripleAndIncrement:0.0.1
2019-03-16 20:02:57 22da217c8e3a4b799a217c8e3a0b79c4 sequence  warm  3.46s      success  guest/tripleAndIncrement:0.0.1
```

- The primary activation record is the last one in the list because it has the earliest start time. The five additional activations are:
  - one activation of the **triple** action with input { value: 3 } and output { value: 9 },
  - one activation of the **increment** action with input { value: 9 } and output { value: 10 },
  - three **secondary activations** of the **tripleAndIncrement** action.
- The secondary activations of the conductor action are responsible for orchestrating the invocations of the component actions.

More Detail : https://github.com/apache/openwhisk/blob/master/docs/conductors.md

OpenWhisk Composer

- Composer is a programming model for composing functions built on OpenWhisk.
- Composer synthesizes OpenWhisk conductor actions to implement compositions.
- Composer is distributed as Node.js package. To install this package, use the NPM: `npm install -g openwhisk-composer`
- Defining a composition:

```
const composer = require('openwhisk-composer')

module.exports = composer.if(
    composer.action('authenticate', { action: function ({ password }) { return { value: password ===
    composer.action('success', { action: function () { return { message: 'success' } } }),
    composer.action('failure', { action: function () { return { message: 'failure' } } }))
```

  - This example composition composes three actions named **authenticate**, **success**, and **failure** using the **composer.if combinator**, which is the conditional construct.
  - It invokes the first one and, depending on the result of this invocation, invokes either the second or third action.
  - This composition includes the definitions of the three composed actions. If the actions are defined and deployed elsewhere, the composition code can be :

```
composer.if('authenticate', 'success', 'failure')
```

OpenWhisk Composer Cont..

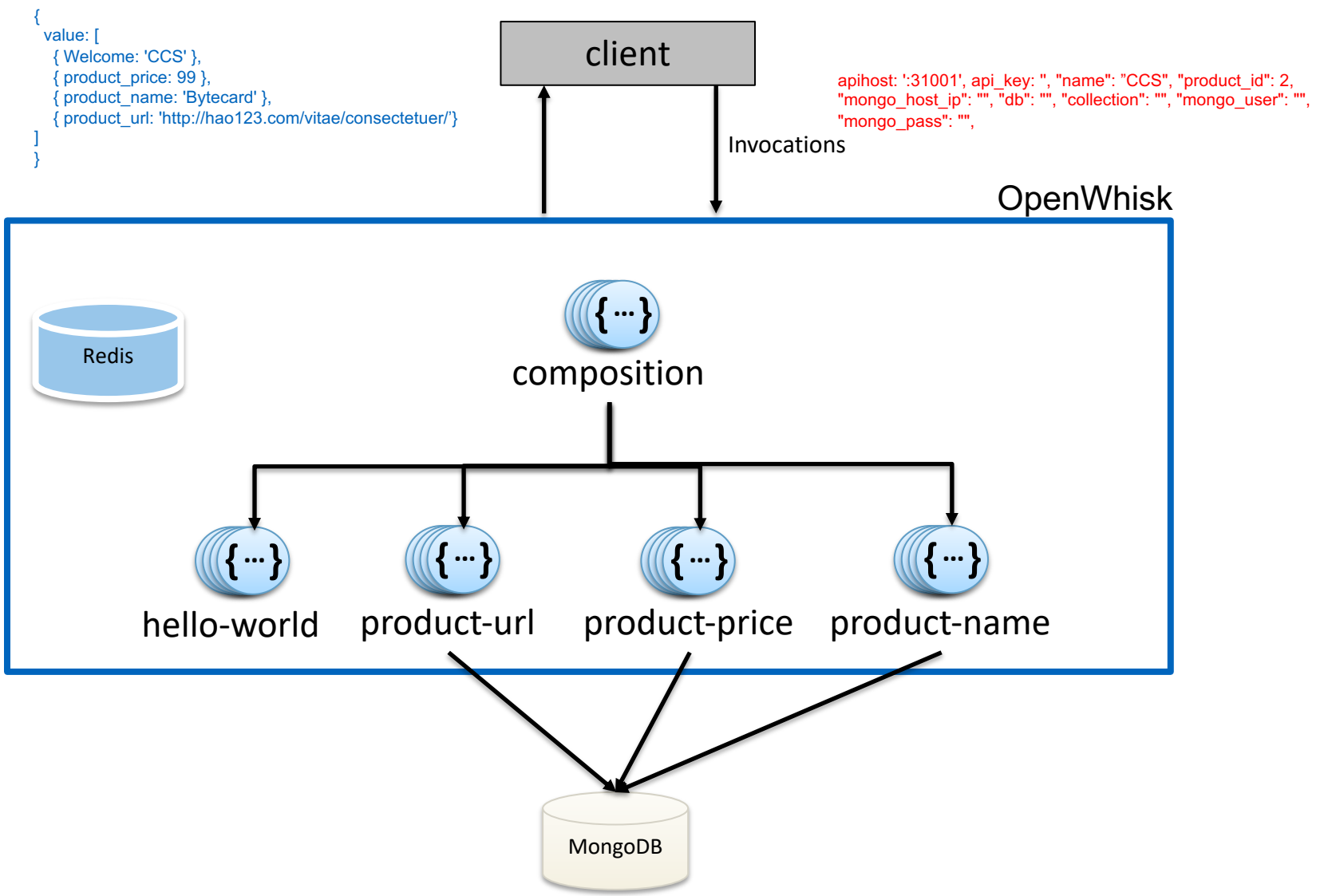
- Deploying a composition:

```
compose demo.js > demo.json
deploy demo demo.json –w -i
```

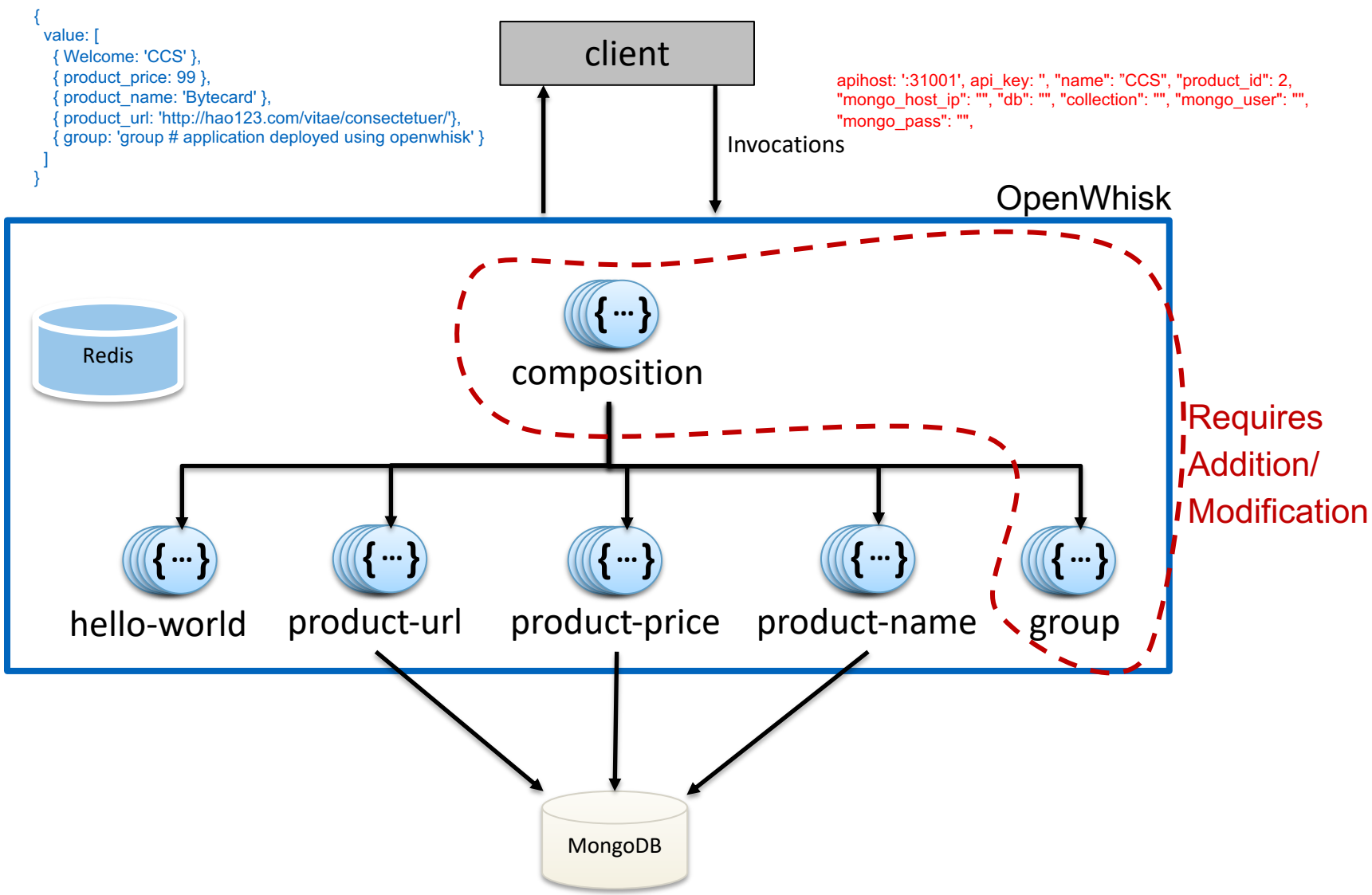
  - The compose command compiles the composition code to a portable JSON format.
  - The deploy command deploys the JSON-encoded composition creating an action with the given name. It also deploys the composed actions if definitions are provided for them.
  - The -w option authorizes the deploy command to overwrite existing definitions.
- Invoking a composition: `wsk action invoke demo -p password passw0rd`
- Parallel Composition
  - Composer offers parallel combinators that make it possible to run actions or compositions in parallel: `composer.parallel('checkInventory', 'detectFraud')`
  - These combinators require **access to a Redis instance** to hold intermediate results of parallel compositions.
- More Combinators: https://github.com/apache/openwhisk-composer/blob/master/docs/COMBINATORS.md

Exercise 5

Exercise 5 Application Architecture



Exercise 5 Application Architecture



Application

- Application source:
  https://github.com/ansjin/openwhisk_parallel_composer.git

```
|____app
|  |____product-name.js
|  |____product-url.js
|  |____composition.js
|  |____product-price.js
|  |____hello-world.js
|  |____group.js
|____README.md
|____test_ow_function.js
```

Modify composition.js to include group function.

Add group.js

Modify to include parameters values

Application – hello-world.js

```
'use strict';

function main(params) {
  var name = params["name"];

  if(name){
    return {"Welcome": name};
  }
  else {
    return {"Welcome": "DEFAULT VALUE"};
  }
}
```

Read the name parameter

Return Message

Application – product-price.js

```
[…]

    let url = 'mongodb://' + mongo_user + ':' + mongo_pass + '@' + mongo_host_ip +
':27017?authMechanism=SCRAM-SHA-1&authSource=admin';
    let client, db;
    try {
      client = await MongoClient.connect(url, {useNewUrlParser: true});
      db = client.db(db_name);
      let dCollection = db.collection(collection_name);


      let result = await dCollection.findOne({product_id: product_id_given});
      return {"product_price": result.product_price};

    } catch (err) {
      console.log("err", err);
      return {"product_price_err": "error, check logs"};
    } // catch any mongo error here
    finally {
      client.close();
    } // make sure to close your connection after

  } else {
    return {"product_name_err": "Id or mongo host IP address or db or collection name or
mongo_user or mongo_pass  is not provided"};
  }
}
```

Read the parameters

Configure MongoDB

Query Mongodb

Return the price if found

Return error if not found

Application – composition.js

```
const composer = require('openwhisk-composer');

module.exports = composer.parallel(
  composer.action('hello-world'),
  composer.action('product-price'),
  composer.action('product-name'),
  composer.action('product-url'),
);
```

Parallel combinator

All the actions are invoked in parallel

Application – test_ow_function.js

```
function main(params) {
  // eslint-disable-next-line global-require, import/no-extraneous-dependencies
  const options = {
    apihost: ':31001',
    api_key: '23bc46b1-71f6-4ed5-8c54-
816aa4f8c502:123zO3xZCLrMN6v2BKK1dXYFpXlPkccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIwP',
    ignore_certs: true
  }
  const ow = require('openwhisk')(options);

  let data = {
    "$composer": {
      "redis": {
        "uri": "redis://owdev-redis.openwhisk.svc.cluster.local:6379"
      },
      "openwhisk": {
        "ignore_certs": true
      }
    },
    "name": "anshul",
    "product_id": 2,
    "mongo_host_ip": "",
    "db": "ccs",
    "collection": "products",
    "mongo_user": "root",
    "mongo_pass": "",

  }
  const invoke = (actionName, params) => ow.actions.invoke({ actionName, params, blocking: true });

  return invoke('composition', data)
    .then(res => res.response.result);
```

OpenWhisk Parameters

Composer Parameters

Application Parameters

Invoking composition

OpenWhisk Installation
and
Running the application

Step1: Creating a Kubernetes Cluster

- Create two new VM on GCP (select instance > = n1-standard-2)
- Create a Kubernetes cluster using these two machines as described in last exercise
- Kubernetes cluster operate on the below mentioned ports, so enable these.
  - 30000-32767 (node port range)
  - 8001, 443, 6443 (for Kubernetes communication)
- Check if both the nodes are in ready status using command:

```
kubectl get nodes
```

- Check if all the pods are running: `kubectl get pods --all-namespaces`
- Don‘t forget to taint master node:

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

Step2: Install Helm and wsk (on Control-Plane Node)

- Installing Helm
  1. Download it: `wget https://get.helm.sh/helm-v3.5.0-linux-amd64.tar.gz`
  2. Unpack it : `tar -zxvf helm-v3.5.0-linux-amd64.tar.gz`
  3. Find the helm binary in the unpacked directory, and move it to its desired destination : `sudo mv linux-amd64/helm /usr/local/bin/helm`

- Installing wsk **(you can configure this on your local laptop as well):**
  1. Download it: `wget https://github.com/apache/openwhisk-cli/releases/download/latest/OpenWhisk_CLI-latest-linux-386.tgz`
  2. Unpack it : `tar -xvf OpenWhisk_CLI-latest-linux-386.tgz`
  3. Move it to desired destination: `sudo mv wsk /usr/local/bin/wsk`

Step3: Configure OpenWhisk (on Control-Plane Node)

- Create a **mycluster.yaml** file

```
controller:
  replicaCount: 1
whisk:
  ingress:
    type: NodePort
    apiHostName: <CLUSTER_IP>
    apiHostPort: 31001
k8s:
  persistence:
    enabled: false
nginx:
  httpsNodePort: 31001
invoker:
  containerFactory:
    impl: "kubernetes"
```

Here replace the <CLUSTER_IP> with the Kubernetes cluster IP which you can get by running the command (here 10.128.0.13):

kubectl cluster-info

```
Kubernetes master is running at https://10.128.0.13:6443
KubeDNS is running at https://10.128.0.13:6443/api/v1/na
```

Step4: Deploy OpenWhisk (on Control-Plane Node)

1. Label **all nodes** as part of the cluster as **Invoker** nodes by running this command :

```
kubectl label nodes --all openwhisk-role=invoker
```

2. Deploy openwhisk using helm

```
helm repo add openwhisk https://openwhisk.apache.org/charts

helm repo update

helm install owdev openwhisk/openwhisk -n openwhisk --create-namespace -f
mycluster.yaml
```

The process will take around 5–10 minutes. This will create all the required Kubernetes components (containers, networks, volumes, etc) required by OpenWhisk.

1. Check the status of the deployment :

```
helm status owdev -n openwhisk
```

Step5: Configure wsk (on Control-Plane/Local Laptop)

- Configure by setting the auth and API host properties as follows:

```
wsk property set --apihost <master_node_public_ip>:31001

wsk property set --auth 23bc46b1-71f6-4ed5-8c54-
816aa4f8c502:123zO3xZCLrMN6v2BKK1dXYFpXlPkccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIw
P
```

- Check if it working or not by listing the installed packages:

```
wsk -i package list /whisk.system
```

1. -i to ignore SSL certificate checks
2. The output of above command will be:

```
packages
/whisk.system/messaging                                    shared
/whisk.system/cloudant                                     shared
/whisk.system/alarms                                       shared
/whisk.system/github                                       shared
/whisk.system/utils                                        shared
/whisk.system/samples                                      shared
/whisk.system/slack                                        shared
/whisk.system/weather                                      shared
/whisk.system/websocket                                    shared
```

Step6: App Deployment (on Control-Plane/Local Laptop)

1. Install openwhisk-composer:

```
npm install -g openwhisk-composer
```

2. Clone the application repository:

```
git clone https://github.com/ansjin/openwhisk_parallel_composer.git
```

3. Edit test_ow_functions.js file, to include following:

```
const options = {
  apihost: 'MASTER_IP:31001',
  api_key: '23bc46b1-71f6-4ed5-8c54-
816aa4f8c502:123zO3xZCLrMN6v2BKK1dXYFpXlPkccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIwP',
  ignore_certs: true
}
```

```
"name": "anshul",
 "product_id": 2,
 "mongo_host_ip": " FILL THIS ",
 "db": "ccs",
 "collection": "products",
 "mongo_user": "root",
 "mongo_pass": "FILL THIS",
```

Check exercise5 page on cloudcom server to replace FILL THIS

Step6: Application Deployment Cont..

4. Deploy the functions:

```
cd app
```

wsk -i action create hello-world hello-world.js
wsk -i action create product-url product-url.js --docker openwhiskansjin/action-nodejs-v14:mongo
wsk -i action create product-name product-name.js --docker openwhiskansjin/action-nodejs-v14:mongo
wsk -i action create product-price product-price.js --docker openwhiskansjin/action-nodejs-v14:mongo

```
compose composition.js > myCompose.json
deploy composition myCompose.json -i
```

wsk -i action create test_ow_functions test_ow_functions.js

5. Test Composition:

```
wsk -i action invoke test_ow_functions --result
```


```
{
    "value": [
        {
            "Welcome": "anshul"
        },
        {
            "product_price": 90
        },
        {
            "product_name": "Overhold"
        },
        {
            "product_url": "https://goo.ne.jp/commodo/placerat/
        }
    ]
}
```

Tasks to be Completed

Tasks to be completed

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

1. Create a new function called “group” which returns:

   { group: 'group # application deployed using openwhisk'}

2. Modify composition.js and test_ow_functions.js .
3. After modification, run this application on OpenWhisk:
   a. Start Kubernetes Cluster(using 2 nodes).
   b. Install Helm, wsk and openwhisk-composer.
   c. Create OpenWhisk configuration file.
   d. Deploy OpenWhisk
   e. Configure wsk CLI.
   f. Create all the functions (product-price, product-name, product-url, group, test_ow-functions) using wsk.
   g. Create composition using compose and deploy command of openwhisk-composer.
   h. Test the composition by invoking `test_ow_functions.`
   i. Expose the port 31001.

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 **exercise5**
4. Click on it and a form will come up where you must provide
   - VM Master ip on which your application is running

Example:

10.0.23.1

5. Then click submit.
6. You will get the correct submission from server if everything is done correctly.

Deadline for submission: Check the submission server

Thank you for your attention!