**KUBERNATES EXERCISES**

**Exercise – Pod**

InfyTel application can be hosted using the following Docker images as per the application architecture discussed in the previous exercise.

* infytel-mysq-img
* infytel-eureka-img
* infytel-customer-img
* Infytel-calldetail-img
* infytel-plan-img
* infytel-angular-img

Ensure that the repository name of each image is properly tagged as shown below in your Hyper-V environment.

* registry.example.com:5000/infytel-mysq-img:v1
* registry.example.com:5000/infytel-eureka-img:v1
* registry.example.com:5000/infytel-customer-img:v1
* registry.example.com:5000/infytel-calldetail-img:v1
* registry.example.com:5000/infytel-plan-img:v1
* registry.example.com:5000/infytel-angular-img:v1

Note: Place these images in all the worker nodes of the Kubernetes cluster environment

**Exercise 2 - Create MySQL Pod**

Create a MySQL pod and view the databases, tables and its' data available for InfyTel application.

**Step 1:** Create a Pod named "mysqldb-pod" using MySQL image with the following specification

* Container name: any
* Namespace: infytel-ns
* Container port: 3306 (default)
* Root user password: root
* Volume mount path: /var/lib/mysql

**Step 2:** View the pod

**Step 3:** Login to the newly created MySQL container and perform the following operations

* Login to mysql prompt as root user with the correct password
* View the tables and its' data that are present in the InfyTel related databases such as infytel-calldetails, infytel-customer and infytel-plan

Note: You can perform any database operations that are related to MySQL.

**Step 4:** Logout from the container

**Step 5:** Delete the "mysqldb-pod" pod

**KUBERNATES DEMOS**

**Kubernetes Objects - Labels and Selectors**

**Demo - Define Labels**

In this demonstration, three pods will be created to deploy web applications on nginx web server. These pods will be used for development, QA and production purpose. After creating the required Pods, it can be viewed and filtered based on the need using labels.

Below are the steps you need to follow to complete the above task.

Step 1: Create three yaml configurations files with different definition, especially labels.

*[root@k8s-master|/root/yaml]# ls*

*dev\_nginx\_pod.yaml prod\_nginx\_pod.yaml qa\_nginx\_pod.yaml*

Below are the configurations for the above listed yaml files.



Step 2: Create all the pods by running the respective yaml as shown below.

*[root@k8s-master|/root/yaml]# kubectl apply -f prod\_nginx\_pod.yaml*

*pod/label-demo-prod created*

*[root@k8s-master|/root/yaml]# kubectl apply -f dev\_nginx\_pod.yaml*

*pod/label-demo-dev created*

*[root@k8s-master|/root/yaml]# kubectl apply -f qa\_nginx\_pod.yaml*

*pod/label-demo-qa created*

Step 3: View the pods that are created. Only pods have been listed using "kubectl get pods --show-labels" command to view the labels

*[root@k8s-master|/root/yaml]# kubectl get pods --show-labels*

*NAME READY STATUS RESTARTS AGE LABELS*

*label-demo-dev 1/1 Running 0 17s app=nginx,environment=development*

*label-demo-prod 1/1 Running 0 31s app=nginx,environment=production*

*label-demo-qa 1/1 Running 0 5s app=nginx,environment=quality*

**Demo - Filter View (Selectors)**

Follow the steps mentioned below to filter the pods based on the need. Three pods have been already created for production, development and QA purpose.

**Example 1: Equality-based**

*[root@k8s-master|/root/yaml]# kubectl get pods --selector environment=quality*

*NAME READY STATUS RESTARTS AGE*

*label-demo-qa 1/1 Running 0 42m*

*[root@k8s-master|/root/yaml]# kubectl get pods -l environment=quality,app=nginx # --selector or -l*

*NAME READY STATUS RESTARTS AGE*

*label-demo-qa 1/1 Running 0 42m*

First command shows the pods that has the key "environment" with the value "quality". Consider there are multiple pods in the cluster for different applications such as nginx, mysql, apache. You can use apply multiple selectors using comma operator (similar to AND)

Second command shows the pods that has the key "environment" with the value "quality" and key "app" with the value "nginx".

**Example 2: Set-based**

If you wish to provide multiple values on the same key then you can use set-based operator. Below example shows all the pods that has the key "environment" with the values "production" or "development". You have to enclose the set-based selector with in a single quote.

*[root@k8s-master|/root/yaml]# kubectl get pods --selector 'environment in (production,development)'*

*NAME READY STATUS RESTARTS AGE*

*label-demo-dev 1/1 Running 0 53m*

*label-demo-prod 1/1 Running 0 54m*

**Example 3: Set-based with equality-based**

Below command shows the pods that have the key "environment" with the values "production" or "development" and the key "app" with the value "nginx" which as both set-based and equality-based selectors.

*[root@k8s-master|/root/yaml]# kubectl get pods --selector 'environment in (production,development),app=nginx'*

*NAME READY STATUS RESTARTS AGE*

*label-demo-dev 1/1 Running 0 54m*

*label-demo-prod 1/1 Running 0 54m*

**Demo - Add Annotations**

Below example yaml configuration file has the metadata of the objects through annotations and labels as well. Remember, you can select the objects using labels only. Annotations are added for defining additional information (metadata) about the Kubernetes object.

[root@k8s-master|/root/yaml]# cat prod\_nginx\_pod.yaml

*apiVersion: v1*

*kind: Pod*

*metadata:*

*name: label-demo-prod*

*labels:*

*environment: production*

*app: nginx*

*annotations:*

*buildVersion: 1.25*

*owner: CISRUN*

*contact: cisrun@example.com*

*spec:*

*containers:*

*- name: nginx*

*image: nginx*

*ports:*

*- containerPort: 80*

**Exercise 1 - Define Labels**

Create MySQL and Eureka pods with the labels.

**Step 1:** Create a Pod named "mysqldb-pod" using MySQL image with the following specification

* Container name: any
* Namespace: infytel-ns
* app: db-tier, env=test
* Container port: 3306 (default)
* Root user password: root
* Volume mount path: /var/lib/mysql

**Step 2**: Create a Pod named "eureka-pod" using Eureka image with the following specification

* Container name: any
* Namespace: infytel-ns
* app: eureka-server, env=prod
* Container port: 2222

**Step 3:** View all the pods with the labels defined

Filter the pods using selector. Perform the below operations on the pods that are already created in the previous exercise.

* View the pod that is created for testing purpose
* View the pod that is created for production purpose
* View the pods that are created for both testing and production purpose
* Delete the pods named "mysqldb-pod" and "eureka-pod"

**Kubernetes Objects – Services**

**Demo - Create ClusterIP Service**

Below is an example for ClusterIP Service. As mentioned, it makes a pod or group of pods accessible to other pods within the internal network of the Kubernetes cluster.

**Step 1**: Create a yaml file as shown below

*[root@k8s-master|/root/yamls]# cat clusterip-svc.yaml*

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: db-cip-svc # Name of the service*

*spec:*

*type: ClusterIP*

*ports:*

*- targetPort: 3306 # Container Port*

*port: 3306 # Port on the Service itself, requests first comes to this port and is forwarded to the targetPort*

*selector:*

*app: db-tier*

**Step 2**: Create a Service using kubectl create command

*[root@k8s-master|/root/yamls]# kubectl create -f clusterip-svc.yaml*

*service/db-cip-svc created*

**Step 3:** View the Service that is created

*[root@k8s-master|/root/yamls]# kubectl get svc*

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*db-cip-svc ClusterIP 10.110.14.44 <none> 3306/TCP 5s*

*kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 74m*

*simple-service NodePort 10.105.125.168 <none> 80:30180/TCP 8m*

**Step 4:** Finally, delete all the Services that are created

*[root@k8s-master|/root/yamls]# kubectl delete svc db-cip-svc simple-service*

*service "db-cip-svc" deleted*

*service "simple-service" deleted*

Note: You can edit or describe the Service using kubectl edit/describe command.

**Exercise – Services**

**Exercise 1** - Create Services for Microservices

In this exercise, you will create the services for deploying customer, plan, eureka, mysql and angular containers. Calldetail microservice will be deployed through roll out update as version 2.

**Step 1:** Create the services in the namespace infytel-ns for customer and plan microservices as mentioned below.

Customer: Create a node port service named "customer-np-svc" with the following specification

* Target port: 8200
* Port: 8200
* Node port: 30012
* Add the appropriate selector to match the respective deployment

Plan (Node port): Create a node port service named "plan-np-svc" with the following specification

* Target port: 8400
* Port: 8400
* Node port: 30014
* Add the appropriate selector to match the respective deployment

Plan (Cluster IP): Create a cluster IP service named "plan-cp-svc" with the following specification

* Target port: 8400
* Port: 8400
* Add the appropriate selector to match the respective deployment

**Step 2:** View all the services that are created.

**Exercise 2** - Create Services for Angular, Eureka and MySQL

Create the services in the namespace infytel-ns for all the below mentioned functionalities.

Angular: Create a node port service named "angular-np-svc" with the following specification

* Target port: 80
* Port: 4200
* Node port: 30015
* Add the appropriate selector to match the respective deployment

Eureka (Node port): Create a node port service named "eureka-np-svc" with the following specification

* Target port: 2222
* Port: 2222
* Node port: 30010
* Add the appropriate selector to match the respective deployment

Eureka (Cluster IP): Create a cluster IP service named "eureka-cp-svc" with the following specification

* Target port: 2222
* Port: 2222
* Add the appropriate selector to match the respective deployment

MySQL: Create a cluster IP service named "db-cp-svc" with the following specification

* Target port: 3306
* Port: 3306
* Add the appropriate selector to match the respective deployment

View all the services that are created.

**Kubernetes Objects – ReplicaSet**

**Demo - Create ReplicaSet**

Let us create a replicaset named "myapp-rc" with 3 nginx pods for hosting frontend web application. Follow the steps mentioned below to create and manage the replicaset.

**Step 1:** create the yaml configuration file as per the requirement.

[root@k8s-master|/root/yaml]# cat rs.yaml

*apiVersion: apps/v1*

*kind: ReplicaSet*

*metadata:*

*name: myapp-rc*

*labels:*

*app: myapp*

*tier: frontend*

*spec:*

*replicas: 3*

*selector:*

*matchLabels:*

*app: myapp*

*tier: frontend*

*template:*

*metadata:*

*name: myapp-pod*

*labels:*

*app: myapp*

*tier: frontend*

*spec:*

*containers:*

*- name: myapp-nginx-cont*

*image: nginx*

**Step 2:** Create the replicaset using kubectl command with the configuration file. Replicaset "myapp-rc" is created. You can also "kubectl apply" to create a replicaset

*[root@k8s-master|/root/yaml]# kubectl create -f rs.yaml*

*replicaset.apps/myapp-rc created*

**Step 3:** View the replicaset and pods created as shown below. ReplicaSet "myapp-rc" contains 3 pods 2 in worker-node2 and 1 in worker-node1.

*[root@k8s-master|/root/yaml]# kubectl get replicaset*

*NAME DESIRED CURRENT READY AGE*

*myapp-rc 3 3 3 4s*

*[root@k8s-master|/root/yaml]# kubectl get pods -o wide*

*NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES*

*myapp-rc-55nzh 1/1 Running 0 37m 10.244.1.6 worker-node1.pod2.example.com <none> <none>*

*myapp-rc-5fm6n 1/1 Running 0 37m 10.244.2.10 worker-node2.pod2.example.com <none> <none>*

*myapp-rc-z54r2 1/1 Running 0 37m 10.244.2.11 worker-node2.pod2.example.com <none> <none>*

**Step 4:** You can also filter the pods with the selectors as shown below. You can view all the labels assigned to the Pod using "kubectl get pods --show-labels" command

*[root@k8s-master|/root/yaml]# kubectl get pods --selector app=myapp*

*NAME READY STATUS RESTARTS AGE*

*myapp-rc-55nzh 1/1 Running 0 38m*

*myapp-rc-5fm6n 1/1 Running 0 38m*

*myapp-rc-z54r2 1/1 Running 0 38m*

*[root@k8s-master|/root/yaml]# kubectl get pods --selector tier=frontend*

*NAME READY STATUS RESTARTS AGE*

*myapp-rc-55nzh 1/1 Running 0 38m*

*myapp-rc-5fm6n 1/1 Running 0 38m*

*myapp-rc-z54r2 1/1 Running 0 38m*

**Demo - Manage ReplicaSet**

Assume that for some reason pod or the node in which it runs crashes, then the replicaset automatically create the pods to guarantee the availability of a specified number of identical pods.

**Step 1:** View the existing pods. There are 3 replicas available now

*[root@k8s-master|/root/yaml]# kubectl get pods*

*NAME READY STATUS RESTARTS AGE*

*myapp-rc-55nzh 1/1 Running 0 38m*

*myapp-rc-5fm6n 1/1 Running 0 38m*

*myapp-rc-z54r2 1/1 Running 0 38m*

**Step 2:** For this demo, pod "myapp-rc-55nzh" (runs in worker-node1) is deleted intentionally using "kubectl delete pod" command. Observe that there is a new pod named "myapp-rc-x8mfd" created in worker-node1 immediately to ensure 3 pods running always.

*[root@k8s-master|/root/yaml]# kubectl delete pod myapp-rc-55nzh*

*pod "myapp-rc-55nzh" deleted*

*[root@k8s-master|/root/yaml]# kubectl get pods -o wide*

*NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES*

*myapp-rc-5fm6n 1/1 Running 0 49m 10.244.2.10 worker-node2.pod2.example.com <none> <none>*

*myapp-rc-x8mfd 1/1 Running 0 7s 10.244.1.7 worker-node1.pod2.example.com <none> <none>*

*myapp-rc-z54r2 1/1 Running 0 49m 10.244.2.11 worker-node2.pod2.example.com <none> <none>*

**Kubernetes Objects – ReplicaSet**

**Demo - Scale ReplicaSet (Using configuration file)**

Follow the below steps to scale the replicaset using yaml file.

**Step 1:** Create the yaml file replicaset named "myapp-rc" with 3 replicas for nginx container as shown below.

*[root@k8s-master|/root/yaml]# cat rs.yaml*

*apiVersion: apps/v1*

*kind: ReplicaSet*

*metadata:*

*name: myapp-rc*

*labels:*

*app: myapp*

*tier: frontend*

*spec:*

*replicas: 3*

*selector:*

*matchLabels:*

*app: myapp*

*tier: frontend*

*template:*

*metadata:*

*name: myapp-pod*

*labels:*

*app: myapp*

*tier: frontend*

*spec:*

*containers:*

*- name: myapp-nginx-cont*

*image: nginx*

*[root@k8s-master|/root/yaml]# kubectl apply -f rs.yaml*

*replicaset.apps/myapp-rc created*

**Step 2:** View the replicaset and pods

*[root@k8s-master|/root/yaml]# kubectl get replicaset*

*NAME DESIRED CURRENT READY AGE*

*myapp-rc 3 3 3 6s*

*[root@k8s-master|/root/yaml]# kubectl get pods -o wide*

*NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES*

*myapp-rc-6h2qd 1/1 Running 0 14s 10.244.2.14 worker-node2.pod2.example.com <none> <none>*

*myapp-rc-vk98l 1/1 Running 0 14s 10.244.1.9 worker-node1.pod2.example.com <none> <none>*

*myapp-rc-vtglk 1/1 Running 0 14s 10.244.1.10 worker-node1.pod2.example.com <none> <none>*

**Step 3:** In the yaml file, update "replicas" field value to scale the replicaset to maintain 5 identical replicas of the pods all the time. Then, run "kubectl replace" command scale up the replicaset

*[root@k8s-master|/root/yaml]# cat rs.yaml*

*apiVersion: apps/v1*

*kind: ReplicaSet*

*metadata:*

*name: myapp-rc*

*labels:*

*app: myapp*

*tier: frontend*

*spec:*

*replicas: 5*

*selector:*

*matchLabels:*

*app: myapp*

*tier: frontend*

*template:*

*metadata:*

*name: myapp-pod*

*labels:*

*app: myapp*

*tier: frontend*

*spec:*

*containers:*

*- name: myapp-nginx-cont*

*image: nginx*

*[root@k8s-master|/root/yaml]# kubectl replace -f rs.yaml*

*replicaset.apps/myapp-rc replaced*

**Step 4:** View the replicaset and pods

*[root@k8s-master|/root/yaml]# kubectl get replicaset*

*NAME DESIRED CURRENT READY AGE*

*myapp-rc 5 5 5 2m49s*

*[root@k8s-master|/root/yaml]# kubectl get pods -o wide*

*NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES*

*myapp-rc-6h2qd 1/1 Running 0 2m52s 10.244.2.14 worker-node2.pod2.example.com <none> <none>*

*myapp-rc-jn48p 1/1 Running 0 7s 10.244.2.16 worker-node2.pod2.example.com <none> <none>*

*myapp-rc-l9t2d 1/1 Running 0 7s 10.244.2.15 worker-node2.pod2.example.com <none> <none>*

*myapp-rc-vk98l 1/1 Running 0 2m52s 10.244.1.9 worker-node1.pod2.example.com <none> <none>*

*myapp-rc-vtglk 1/1 Running 0 2m52s 10.244.1.10 worker-node1.pod2.example.com <none> <none>*

**Demo - Scale ReplicaSet (HPA)**

Follow the steps mentioned below to create and test HPA on replicaset.

**Step 1:** To demonstrate Horizontal Pod Autoscaler, custom docker image php-apache is used. First, you start a deployment running the image and expose it as a service. Service will be discussed in the next section.

*[root@k8s-master|/root]# kubectl run php-apache --image=gcr.io/google\_containers/hpa-example:run --requests=cpu=200m --expose --port=80*

*service/php-apache created*

*deployment.apps/php-apache created*

**Step 2:** View the pod and the service that is created

*[root@k8s-master ~]# kubectl get pods*

*NAME READY STATUS RESTARTS AGE*

*php-apache-69df9d46d5-cqdqt 1/1 Running 0 21s*

*[root@k8s-master ~]# kubectl get service*

*NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*

*kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 38d*

*php-apache ClusterIP 10.102.193.18 <none> 80/TCP 21s*

**Step 3:** Now, you create the autoscaler using "kubectl autoscale" command. Below command will create a Horizontal Pod Autoscaler that maintains between 1 and 10 replicas of the Pods controlled by the php-apache deployment

*[root@k8s-master|/root]# kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10*

*horizontalpodautoscaler.autoscaling/php-apache autoscaled*

HPA will increase and decrease the number of replicas (via the deployment) to maintain an average CPU utilization across all pods of 50%.

**Step 4:** Check the current status of autoscaler by running below command

*[root@k8s-master|/root]# kubectl get hpa*

*NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE*

*php-apache Deployment/php-apache <unknown>/50% 1 10 1 97s*

**Step 5:** Let us see how the autoscaler reacts to increased load. You start a container, and send an infinite loop of queries to the php-apache service in a different terminal as shown below

*[root@k8s-master|/root]# kubectl run -i --tty load-generator --image=busybox:run /bin/sh*

If you don't see a command prompt, try pressing enter.

# while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done

OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!

OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!

OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!

**Step 6:** Check the status of the hpa in the first terminal. As there is no much of load on the application, only one replica is running now

*[root@k8s-master ~]# kubectl get hpa*

*NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE*

*php-apache Deployment/php-apache 77% / 50% 1 10 1 3m*

**Step 7:** Wait for the load to increase and view the status of the hpa and deployment after few more minutes. You can see that CPU consumption is increased to 355%, hence the number of replicas is also increased to 8 as shown below

*[root@k8s-master ~]# kubectl get hpa*

*NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE*

*php-apache Deployment/php-apache 355% / 50% 1 10 8 15m*

*[root@k8s-master ~]# kubectl get deployment php-apache*

*NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE*

*php-apache 8 8 8 8 15m*

*[root@k8s-master ~]# kubectl get pods | grep php*

*php-apache-69df9d46d5-bnqbr 1/1 Running 0 3m*

*php-apache-69df9d46d5-cqdqt 1/1 Running 0 15m*

*php-apache-69df9d46d5-ftxvh 1/1 Running 0 7m*

*php-apache-69df9d46d5-jd2mq 1/1 Running 0 11m*

*php-apache-69df9d46d5-r99pp 1/1 Running 0 3m*

*php-apache-69df9d46d5-s2qgg 1/1 Running 0 3m*

*php-apache-69df9d46d5-wftmp 1/1 Running 0 3m*

*php-apache-69df9d46d5-wllrd 1/1 Running 0 7m*

Based on the CPU utilization of the nodes, HPA is automatically scaling up/down the replicas of the pods.

**Step 8:** Finally, you can reduce the load on the application by terminating the infinite loop (press Ctrl + C) and check the status of the hpa and pods after few mins

*[root@k8s-master ~]# kubectl get hpa*

*NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE*

*php-apache Deployment/php-apache 0% / 50% 1 10 1 25m*

*[root@k8s-master ~]# kubectl get deployment php-apache*

*NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE*

*php-apache 1 1 1 1 25m*

As there is no load on the application, number of replicas is scaled down automatically.

**Replication Controller**

A ReplicationController (RC) also ensures that a specified number of pod replicas are running at any one time. If there are too many pods, the ReplicationController terminates the extra pods. If there are too few, the ReplicationController starts more pods. With respect to working principle, RC is similar to replicaset (RS).

**Step 1:** Create the RC yaml file as shown below to create a RC named "nginx-rc" with 3 pods

*[root@k8s-master|/root]# cat rc.yml*

*apiVersion: v1*

*kind: ReplicationController*

*metadata:*

*name: nginx-rc*

*spec:*

*replicas: 3*

*selector:*

*app: nginx-app*

*template:*

*metadata:*

*name: nginx-pod*

*labels:*

*app: nginx-app*

*spec:*

*containers:*

*- name: nginx-container*

*image: nginx*

*ports:*

*- containerPort: 80*

**Step 2:** Create a replication controller using kubectl command

*[root@k8s-master|/root]# kubectl apply -f rc.yml*

*replicationcontroller/nginx-rc created*

**Step 3:** View the RC and it's pods

*[root@k8s-master|/root]# kubectl get rc*

*NAME DESIRED CURRENT READY AGE*

*nginx-rc 3 3 3 7s*

*[root@k8s-master|/root]# kubectl get pods*

*NAME READY STATUS RESTARTS AGE*

*nginx-rc-2ptvd 1/1 Running 0 9s*

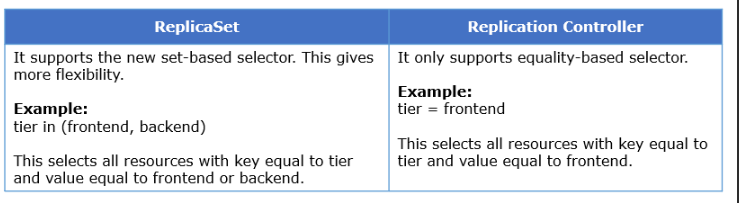
*nginx-rc-bxb2f 1/1 Running 0 9s*

*nginx-rc-r2kxx* 1/1 Running 0 9s

You can further scale up/down the RC as shown in RS.

**ReplicaSet Vs Replication Controller**

RS is the next-generation RC that supports the new set-based label selector. Let us understand this with an example.



**Demo - Delete ReplicaSet**

If you wish decommission the replicaset environment, you can delete the entire replicaset as deleting a pod will automatically create another pod.

Any of the below commands can be used to delete the replicaset.

**Command 1:**

*[root@k8s-master|/root/yaml]# kubectl delete -f rs.yaml*

*replicaset.apps "myapp-rc" deleted*

**Command 2:**

*[root@k8s-master|/root/yaml]# kubectl delete replicaset myapp-rc*

*replicaset.apps "myapp-rc" deleted*

**Exercise – ReplicaSet**

**Exercise 1 - Create ReplicaSet**

Create a MySQL replicaset for InfyTel application to ensure that the database is highly available.

Step 1: Create a replicaset named "mysql-rs" in the namespace infytel-ns using MySQL image with the following specification

* Container name: any
* Namespace: infytel-ns
* Container port: 3306 (default)
* Root user password: root
* Volume mount path: /var/lib/mysql
* replicas: 3

Step 2: View the replicaset and the pods along with the node in which it runs

Step 3: Delete any one of the pods and observe what happens to the number of replicas

**Exercise 2 - Scale ReplicaSet**

Assume that the demand for InfyTel application increases during the specific period and you need to scale up the replicaset. Similarly, once the demand is over, you can scale down the replicaset.

Step 1: Scale the replicaset "mysql-rs" with 2 more pods

Step 2: View the replicaset and the number of pods created for replicaset along with the node in which it runs

Step 3: Delete any one of the pods and observe what happens to the number of replicas

Step 4: You can scale down the replicaset with 4 pods once the demand is over

Step 5: Finally, delete the replicaset

**Exercise – Deployment**

**Exercise 1 - Create Deployments for InfyTel Application**

Create the following deployments for InfyTel application in infytel-ns namespace.

**Step 1:** Create MySQL deployment named "mysql-deployment" with the following specification

* Pod name: any
* Container name: any
  + Image name: mysql
  + Container port: 3306
  + Root user password: root
  + Volume mount path: /var/lib/mysql
* Replicas: 1
* Add the appropriate labels and selector to match the respective service

**Step 2:** Create Eureka deployment named "eureka-deployment" with the following specification

* Pod name: any
* Container name: any
  + Image name: eureka
  + Container port: 2222
* Replicas: 1
* Add the appropriate labels and selector to match the respective service

**Step 3:** View the deployments and the pods that are created,

**Exercise 2 - Create Deployments for InfyTel Application**

Create the following deployments for InfyTel application in infytel-ns namespace.

**Step 1:** Create Customer deployment named "customer-deployment" with the following specification

* Pod name: any
* Container name: any
* Image name: customer
* Container port: 8200
* Environment variables:
  + HOSTNAME\_ERK=<Cluster IP name of eureka service>
  + HOSTNAME\_CON=<Cluster IP name of mysql service>
  + HOSTNAME\_PLAN=http://<Cluster IP name of plan service>:<Target port of plan service>
* Replicas: 1
* Add the appropriate labels and selector to match the respective service

**Step 2:** Create Plan deployment named "plan-deployment" with the following specification

Pod name: any

* Container name: any
* Image name: plan
* Container port: 8400
* Environment variables:
  + - HOSTNAME\_ERK=<Cluster IP name of eureka service>
    - HOSTNAME\_CON=<Cluster IP name of mysql service>
  + Replicas: 1
  + Add the appropriate labels and selector to match the respective service

**Step 3:** Create Angular deployment named "angular-deployment" with the following specification

* Pod name: any
* Container name: any
* Image name: angular
* Container port: 80
* Environment variables:
  + API\_URL1=http://<Machine IP of customer pod>:<Node port of customer service>
  + API\_URL2=http://<Machine IP of plan pod>:<Node port of plan service>
* Replicas: 1
* Add the appropriate labels and selector to match the respective service

**Step 4:** View the deployments and the pods that are created

**Exercise 3 - Access InfyTel Application**

Now, you can access the applications through any browser of your base machine. Ensure that all the services and pods are running before accessing the application.

* Check the instances that are currently registered with Eureka. Ensure that both customer and plan microservices are registered with this (Use this URL to view the instances, http://<Machine IP of eureka pod>:<Node port of eureka service>)
* Access the customer microservice. Ensure that you are able to view the details of the customer 9009009001 (Use this URL to access customer service, http://<Machine IP of customer pod>:<Node port of customer service>/customers/9009009001)
* Access the plan microservice. Ensure that you are able to view the details of the plans (Use this URL to access plan service, http://<Machine IP of plan pod>:<Node port of plan service>/plans/2)
* Finally, access the InfyTel application UI using http://<Machine IP of angular pod>:<Node port of angular service>/Infytel-master-UICode/
  + - Login to the application using Phone No "9009009001" and the password "infy@123"
    - Access "View Profile" and "View Plans" options
    - Log out from the application

**Exercise 4 - Rollout New Microservice for InfyTel Application**

Now, you have to rollout another microservice named calldetails which will list the call details of the logged in customer. Perform the following to roll out the new changes using rollout feature of deployment.

Step 1: Create Calldetails deployment named "calldetail-deployment" with the following specification

* Pod name: any
* Container name: any
* Image name: calldetail
* Container port: 8100
* Environment variables:
* HOSTNAME\_ERK=<Cluster IP name of eureka service>
* HOSTNAME\_CON=<Cluster IP name of mysql service>
* Replicas: 1
* Add the appropriate labels and selector to match the respective service

Step 2: Create Angular deployment named "angular-deployment" with the following specification

* Pod name: any
* Container name: any
* Image name: angular
* Container port: 80
* Environment variables:
* API\_URL1=http://<Machine IP of customer pod>:<Node port of customer>
* API\_URL2=http://<Machine IP of plan pod>:<Node port of plan>
* API\_URL3=http://<Machine IP of calldetails pod>:<Node port of calldetails>
* Replicas: 1
* Add the appropriate labels and selector to match the respective service

Step 3: Apply the changes

Step 4: View the status and history of the rollout to ensure that it is successful

Step 5: Launch the application through browser and access the calldetails service along with customer and plan

Note: Once you access the application through browser, if you wish, you can roll back the update and test the application again.

**Exercise - Create ConfigMap**

There are various configuration details you are passing to InfyTel application deployments. Now, all those details are passed through the respective deployment manifest file itself.

As a best practice, create the following ConfigMaps in the namespace infytel-ns for microservices and recreate the respective deployments.

**Step 1:** Create a configmap named "customer-config" and pass the below parameters to customer container

* HOSTNAME\_ERK=<Cluster IP name of eureka service>
* HOSTNAME\_CON=<Cluster IP name of mysql service>
* HOSTNAME\_PLAN=http://<Cluster IP name of plan service>:<Target port of plan service>

**Step 2:** Create a configmap named "plan-config" and pass the below parameters to plan container

* HOSTNAME\_ERK=<Cluster IP name of eureka service>
* HOSTNAME\_CON=<Cluster IP name of mysql service>

Step 3: Pass the configmap to customer and plan deployment manifest files and recreate the deployments

**Step 4:** Login to customer and plan containers and view the host environment variables that are added

**Step 5:** Finally, access the application through browser and test the services

**Exercise – Secret**

**Exercise - Create Secret**

Password of the MySQL root user has been passed to the container through environment variable MYSQL\_ROOT\_PASSWORD with the value "root". As it is not a best practice for handling the confidential data, create a secret named "mysql-secret" in the namespace infytel-ns and store the root user password in it.

Once the secret is created, pass the respective secret parameters to MySQL deployment and recreate it.

Finally, access the application through browser and test the services. Also, login to MySQL container and view the host environment variable that is added.

Exercises – Solution

4

Kubernetes - Basics

Module: Basics of Container and Docker

Exercise 1:

The Infytel application can be dockerized as per the application architecture. There

are few Docker images in your Hyper-V environment which are used to successfully

do the same. These Docker images have to be tagged appropriately with its

repository name for further use.

Tag the Docker images using the information below:

• infytel-mysq-img - registry.example.com:5000/infytel-mysq-img:v1

• infytel-eureka-img - registry.example.com:5000/infytel-eureka-img:v1

• infytel-zipkin-img – registry.example.com:5000/infytel-zipkin-img:v1

• infytel-customer-img - registry.example.com:5000/infytel-customer-img:v1

• infytel-calldetail-img - registry.example.com:5000/infytel-calldetail-img:v1

• infytel-plan-img - registry.example.com:5000/infytel-plan-img:v1

• infytel-angular-img - registry.example.com:5000/infytel-angular-img:v1

Solution:

$ docker tag infytel-mysq-img registry.example.com:5000/infytel-mysq-img:v1

$ docker tag infytel-eureka-img registry.example.com:5000/infytel-eureka-img:v1

$ docker tag infytel-customer-img registry.example.com:5000/infytel-customer-img:v1

$ docker tag infytel-calldetail-img registry.example.com:5000/infytel-calldetail-

img:v1

$ docker tag infytel-plan-img registry.example.com:5000/infytel-plan-img:v1

$ docker tag infytel-angular-img registry.example.com:5000/infytel-angular-img:v1

Exercise 2:

Since the application is fragmented and each segment is deployed in a different

container, we have to make arrangements so that the containers can communicate

with each other. In this exercise, create a new network in the name “infytel-docker-

networkMS” and add the containers to the same.

Hint: Since you want all the containers related to the Infytel application to

communicate with each other, you will create the network with “bridge” network

driver.

Solution:

$ docker create network infytel-docker-networkMS

Exercise 3:

One of the containers required for the application is the mysql container. Run the

mysql container in the background using the image infytel-mysql-img in the

“infytel-docker-networkMS” network . Also, mount the volume “infytel-mysql-

volume” in the path /var/lib/mysql. Use appropriate docker commands to do the

same.

Also, provide the environment variable for “MYSQL\_ROOT\_PASSWORD” as root

Hint: Environment variables can be given in the docker run command using the -e

option.

Solution:

$ docker container run --volume infytel-mysql-volume:/var/lib/mysql --

network=infytel-docker-networkMS --name=infytel-mysqlMS -e MYSQL\_ROOT\_PASSWORD=root

-d registry.example.com:5000/infytel-mysq-img:v1

Exercise 4:

In this exercise, you are going to create an eureka container in the background

using the “infytel-eureka-img” image. This container must be accessible from the

Docker host through port no.2222. Note the following information about the

container:

Name: infytel-eureka-con

Port: The port no. 2222 of the container must be matched to the port no.2222 of

the Docker host.

Network: The container must belong to infytel-docker-networkMS network.

Solution:

$ docker container run --network=infytel-docker-networkMS -p 2222:2222 --

name=infytel-eureka-con -d registry.example.com:5000/infytel-eureka-img:v1

Exercise 5:

In this exercise, you are going to create a Zipkin container in the background using

the “infytel-zipkin-img” image. This container must be accessible from the Docker

host through port no.4444. Note the following information about the container:

Name: infytel-zipkin-con

Port: The port no. 4444 of the container must be matched to the port no.4444 of

the Docker host.

Network: The container must belong to infytel-docker-networkMS network.

Solution:

$ docker container run --network=infytel-docker-networkMS -p 4444:4444 --

name=infytel-zipkin-con -d registry.example.com:5000/infytel-zipkin-img:v1

Exercise 6:

In this exercise, you are going to create a Customer container in the background

using the “infytel-customer-img” image.

Note the following information about the container:

Name: infytel-customer-con

Network: The container must belong to infytel-docker-networkMS network.

Solution:

$ docker container run --network=infytel-docker-networkMS --name=infytel-customer-con

-d registry.example.com:5000/infytel-customer-img:v1

Exercise 7:

In this exercise, you will run the container for Angular UI using the image “infytel-

angular-img”. The container must be accessible from the port 4200 of the host in

the port 80 of the container.

Also, verify the deployment of the application by accessing

http://localhost:4200/infytel-master-UI-Code/.

Fill the required fields with inputs of your choice to verify how the application

works.

Solution:

docker container run --name infytel-angular-con -d -p 4200:80

registry.example.com:5000/infytel-angular-img:v1

Module: Working with Kubernetes Objects

Microservices can be implemented in a simple way using Spring microservices.

Spring Boot helps in quickly creating a microservice with required functionalities.

InfyTel is a telecom application which provides following services to the customers.

• infytel-customer: The customer microservice deals with customer

functionality like login, view profile, register

• infytel-calldetails: The call details microservice deals with call details of a

given customer

• infytel-plan: The plan microservice deals with plan related functionality of

getting all plans and getting a specific plan

The application architecture is shown below.

• Eureka: It is a naming server. All the running microservices are registered

with this naming server

• UI application: You can access the microservices through front-end which is

developed using Angular

• MySQL database has been used with each microservice having its own

database schema

Note: The above application is deployed in Hyper-V environment.

Application architecture in non-Hyper-V environment:

As we deploy this InfyTel application in Infosys Hyper-V environment, there is a

restriction in accessing open internet by Hyper-V machines. Hence we are unable to

use the below functionalities in this isolated environment.

• Config server: It fetches the properties of InfyTel application from GIT.

Below is the sample application properties stored in GIT

Why to use this service?

If there are any changes in the database configuration, we need not update it

in all the microservices. It can be modified only in GIT and the microservices

will get the same from Cloud Config.

Change in Hyper-V environment:

As this cloud service can’t be accessible in this restricted environment, it has

been configured with all the microservices.

• Zuul server: It is a gateway to access all 3 microservices

Change in Hyper-V environment:

Zuul config file contains the pod IP address of all the microservices. Zuul

can’t access microservices using this pod IP through Angular or exposed

outside of the cluster environment. Hence, it has been removed.

Microservice node port and it’s machine IP address where it runs is passed as

an environment variables to Angular container.

Ideally, it is not a recommended solution as the pod will be created in any

worker node dynamically. Hence, Zuul will be used with Ingress controller to

fix this issue.

Note: Ingress controller is discussed in Foundation course.

spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

eureka.client.service-

url.defaultZone=http://localhost:2222/eureka

Kubernetes - Namespace

Exercise 1:

As you know that you can logically isolate all the Kubernetes objects according to

your logical categorization within a namespace. Here, create (use declarative

method) a namespace named infytel-ns to isolate all the Kubernetes objects

(Service, Deployment, ConfigMap, Secret) that you will create in this module

further.

Note: Create all the Kubernetes objects that are related to InfyTel application only

in this namespace further.

Solution:

Kubernetes - Pod

Exercise 1:

InfyTel application can be hosted using the following Docker images as per the

application architecture discussed in the previous exercise.

• infytel-mysq-img

• infytel-eureka-img

• infytel-customer-img

• infytel-calldetail-img

• infytel-plan-img

• infytel-angular-img

Ensure that the repository name of each image is properly tagged as shown below

in your Hyper-V environment.

• registry.example.com:5000/infytel-mysq-img:v1

• registry.example.com:5000/infytel-eureka-img:v1

• registry.example.com:5000/infytel-customer-img:v1

$ cat infytel-ns.yaml

apiVersion: v1

kind: Namespace

metadata:

name: infytel-ns

$ kubectl create -f infytel-ns.yaml

$ kubectl get ns

• registry.example.com:5000/infytel-calldetail-img:v1

• registry.example.com:5000/infytel-plan-img:v1

• registry.example.com:5000/infytel-angular-img:v1

Note: Place these images in all the worker nodes of the Kubernetes cluster

environment

Solution:

Exercise 2:

In this exercise, you will create a MySQL pod and view the databases, tables and its

data available for InfyTel application.

Step 1: Create a Pod named "mysqldb-pod" using MySQL image with the following

specification

• Container name: any

• Namespace: infytel-ns

• Container port: 3306 (default)

• Root user password: root

• Volume mount path: /var/lib/mysql

Solution:

$ docker tag infytel-mysql-img registry.example.com:5000/infytel-mysql-img:v1

$ docker tag infytel-eureka-img registry.example.com:5000/infytel-eureka-img:v1

$ docker tag infytel-calldetail-img registry.example.com:5000/infytel-calldetail-img:v1

$ docker tag infytel-customer-img registry.example.com:5000/infytel-customer-img:v1

$ docker tag infytel-plan-img registry.example.com:5000/infytel-plan-img:v1

$ docker tag infytel-angular-img registry.example.com:5000/infytel-angular-img:v1

Step 2: View the pod

Step 3: Login to the newly created MySQL container and perform the following

operations

• Login to mysql prompt as root user with the correct password

• View the tables and its' data that are present in the InfyTel related databases

such as infytel-calldetails, infytel-customer and infytel-plan

Note: You can perform any database operations that are related to MySQL

$ cat mysql-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: mysqldb-pod

namespace: infytel-ns

spec:

containers:

- name: mysql-container

image: registry.example.com:5000/infytel-mysql-img:v1

imagePullPolicy: IfNotPresent

ports:

- containerPort: 3306

env:

- name: MYSQL\_ROOT\_PASSWORD

value: root

volumeMounts:

- name: infytel-mysql-volume

mountPath: /var/lib/mysql

readOnly: False

volumes:

- name: infytel-mysql-volume

hostPath:

path: /var/lib/mysql

type: Directory

$ kubectl create -f mysql-pod.yaml

$ kubectl get pod -n infytel-ns