K8s lab part 7.1: Running a real Cloud Native App (CNA)

In order to run a CNA that is comprised of multiple scale-out components, we need a load balancer. In the following lab section we will be adding MetalLB (also see <https://metallb.universe.tf/installation/> ):

1. Set strictARP to “true” in kube-proxy:

kubectl edit configmap -n kube-system kube-proxy

Now find and change the line:

strictARP = false

To: strictARP = true

Exit by typing <ESC>wq

1. Install MetalLB:

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/namespace.yaml

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/metallb.yaml

kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 128)"

1. Now we need to create a config that lists what IPs we want to use for our LoadBalancer

nano metallb.yaml (or use your text editor of choice)

Paste in the below and change the IP range:

apiVersion: v1

kind: ConfigMap

metadata:

namespace: metallb-system

name: config

data:

config: |

address-pools:

- name: default

protocol: layer2

addresses:

- 192.168.1.240-192.168.1.250

1. Add the configuration we created above into kubernetes:

kubectl create -f metallb.yaml

K8s lab part 7.2: Running a real workload

Next we are going to run a real working microservice Cloud Native Application workload! In this example we are going to run Yelb, a voting app consisting of a database, a redis caching layer and an application layer.

1. Install the Yelb deployment. First we install wget, then we download the Yelb yaml file:

yum install -y wget

wget https://github.com/mreferre/yelb/raw/master/deployments/platformdeployment/Kubernetes/yaml/yelb-k8s-loadbalancer.yaml

kubectl create -f yelb-k8s-loadbalancer.yaml

1. That’s it! Now check the deployment by running these commands:

kubectl get pods -o wide

kubectl get svc

1. Make sure all pods are in the “Running” state (this can take a few minutes)
2. The output from the second command in step 2) should be similar to:

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 7d1h

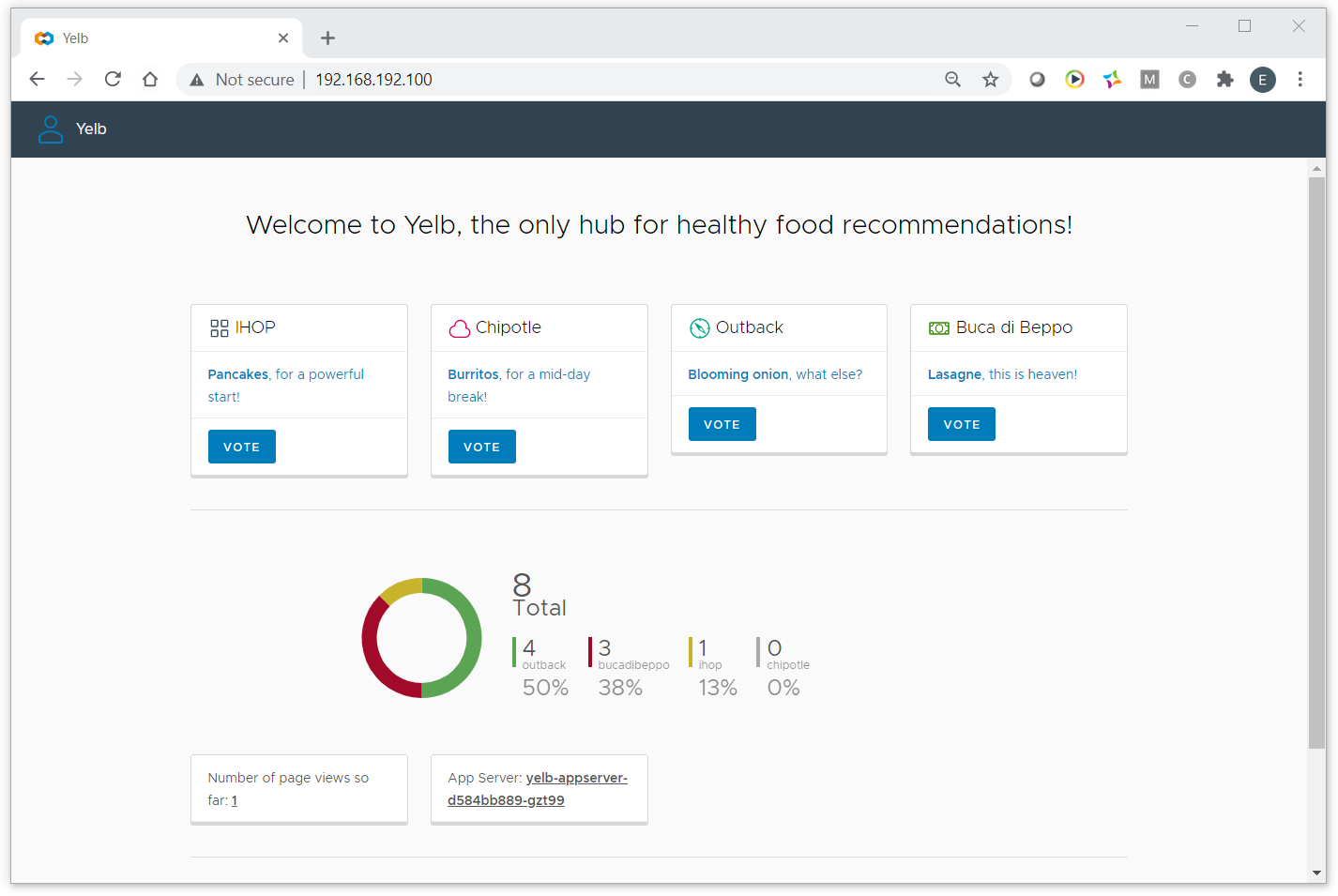
redis-server ClusterIP 10.97.171.218 <none> 6379/TCP 2m48s

yelb-appserver ClusterIP 10.105.62.189 <none> 4567/TCP 2m48s

yelb-db ClusterIP 10.109.220.138 <none> 5432/TCP 2m48s

yelb-ui LoadBalancer 10.111.38.147 **192.168.192.100** 80:30552/TCP 2m47s

1. The highlighted IP address above is our load balancer at work! It is successfully using an IP address from the pool we specified, and it is allowing access into the app server.
2. Now access the application from a browser which you point to the EXTERNAL-IP that you found in step 4). You should now be looking at a working Yelb voting app!



1. Now let’s do a scale-out on the yelb-app microservice:

kubectl scale deployment yelb-appserver --replicas=3

kubectl get pods -o wide

Output should be something similar to this, adding two extra yelb-appserver instances:

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

redis-server-74556bbcb7-rncq6 1/1 Running 1 28m 10.244.2.51 k8s-worker2 <none> <none>

snapshot-controller-0 1/1 Running 1 47m 10.244.2.50 k8s-worker2 <none> <none>

yelb-appserver-d584bb889-dqs2p 0/1 ContainerCreating 0 1s <none> k8s-worker1 <none> <none>

yelb-appserver-d584bb889-rvth5 0/1 ContainerCreating 0 1s <none> k8s-worker1 <none> <none>

yelb-appserver-d584bb889-vq4pc 1/1 Running 1 12m 10.244.2.53 k8s-worker2 <none> <none>

yelb-db-694586cd78-spmsz 1/1 Running 1 28m 10.244.2.55 k8s-worker2 <none> <none>

yelb-ui-798667d648-z4bfk 1/1 Running 0 28m 10.244.1.49 k8s-worker1 <none> <none>

1. Now access the application from a browser once again. Check the Apps Server name at the bottom as you force a refresh in the browser by hitting CTRL-SHIFT-R. You should see different appserver instances serving the web page, indicating we have successfully scaled out (and load balanced) the instances of this microservice!
2. Scale the number of appservers back to 1 by editing the configuration:

kubectl edit deployments.apps yelb-appserver

Change the number of replicas back to “1” and write/quite (:wq)

1. Cast a few votes to fill the database. Note how they are being registered and depicted in the system. Now we “accidentally” destroy DB server:

kubectl delete pod yelb-db-xxxxxxxx-yyyyy

kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

redis-server-74556bbcb7-rncq6 1/1 Running 1 37m 10.244.2.51 k8s-worker2 <none> <none>

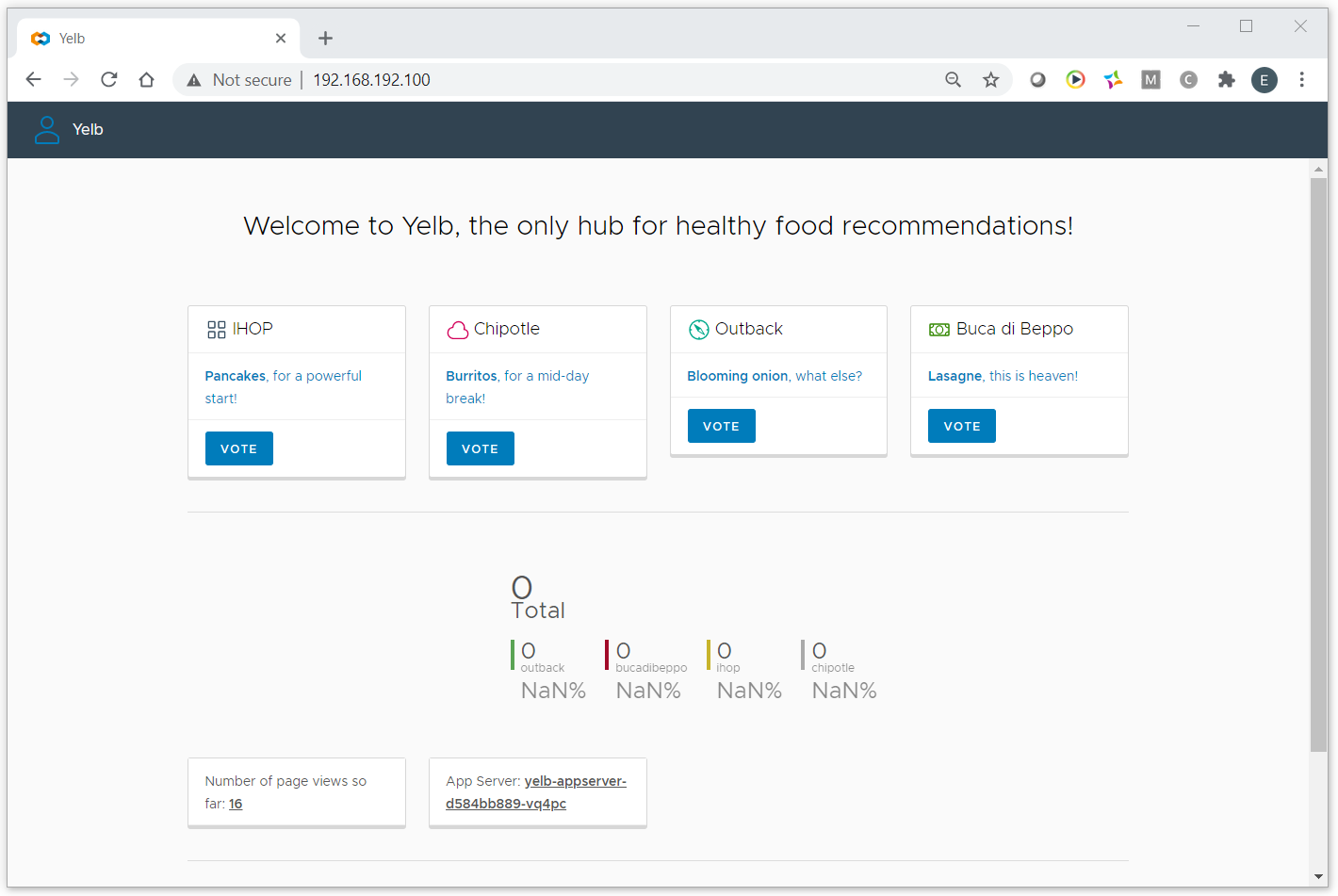
snapshot-controller-0 1/1 Running 1 56m 10.244.2.50 k8s-worker2 <none> <none>

yelb-appserver-d584bb889-vq4pc 1/1 Running 1 21m 10.244.2.53 k8s-worker2 <none> <none>

yelb-db-694586cd78-r7qgw 0/1 ContainerCreating 0 26s <none> k8s-worker1 <none> <none>

yelb-ui-798667d648-z4bfk 1/1 Running 0 37m 10.244.1.49 k8s-worker1 <none> <none>

1. As you can see k8s detects that the pod disappeared. It will automatically spawn a new instance, but as these are containers the Db will be empty again. Verify this by refreshing your browser:



All values are back down to zero! We need to figure out somehow how to get persistency into this storage layer… 😉