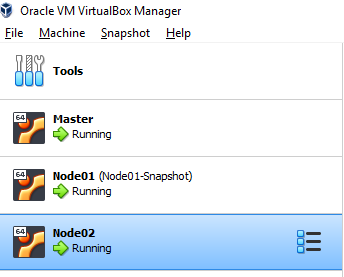
## START K8S CLUSTER

1. Start the master and worked nodes in the VM manager



1. SSH into **each** VM and execute the following command (as admin)

sudo swapoff –a

Credentials

Master: master/root

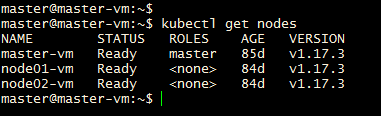
Node01: node01/root

Node02: node02/root

1. Go to master ssh session and use this command to check that the cluster is up:

kubectl get nodes

The result should “Ready” status for master node and two worker nodes.



## CONFIGURE KUBECTL ON WINDOWS VM (Developer Environment)

1. On the master node get the token of the cluster admin using the commands below. Save the token.

sudo kubectl create serviceaccount k8sadmin -n kube-system

sudo kubectl create clusterrolebinding k8sadmin --clusterrole=cluster-admin --serviceaccount=kube-system:k8sadmin

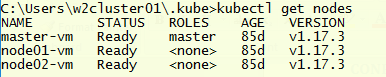
sudo kubectl -n kube-system describe secret $(sudo kubectl -n kube-system get secret | (grep k8sadmin || echo "$\_") | awk '{print $1}') | grep token: | awk '{print $2}'

1. Save the attached config file in USER\_HOME/.kube (if .kube is not created, use the window commands prompt to create one using mkdir .kube)



1. Open the config file and
   1. Replace the placeholder in clusters[0].cluster.server to IP address of master node.
   2. Replace placeholder in Users[0].user.token with the token saved in step 1.
2. Test the kubectl on windows VM now.

kubectl get nodes



## MSCHASSIS SERVICES ON K8S

git clone <https://github.ibm.com/kapil-raina/mschassis.git>.

If cloned already take the latest code by using git pull

## RabbitMQ

1. Go to mschassis\k8s\rabbitmq
2. Run the rabbit MQ deployment

kubectl apply -f rabbitmq-dep-k8s.yaml

kubectl apply -f rabbitmq-dep-w-k8s.yaml

1. Run RabbitMQ k8s service

kubectl apply -f rabbitmq-svc-k8s.yaml

1. Check that RabbitMQ pods are running

kubectl get pods



1. Check that RabbitMQ service is running

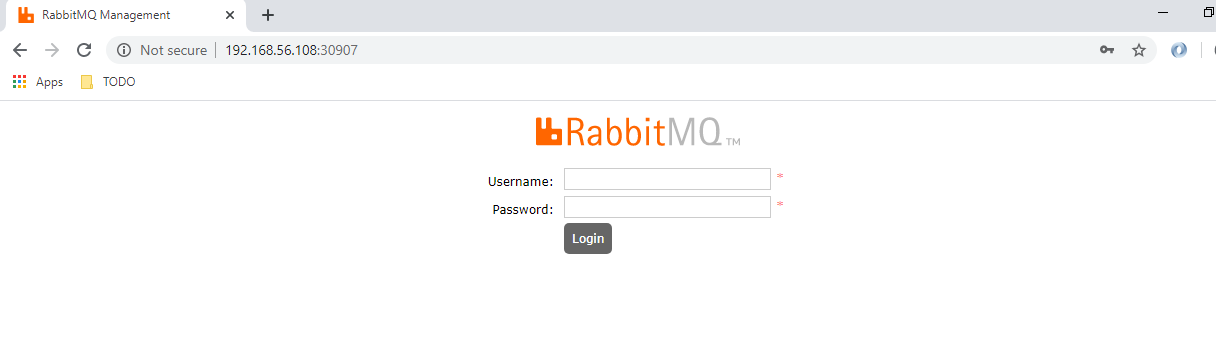
kubectl get svc



1. Since RabbitMQ service runs as NodePort service, it can be accessed externally via <node-ip:NodePort**>. Internally it can be accessed via 5672 with service name as rabbitmq-master, so rabbitmq-master:5672.**

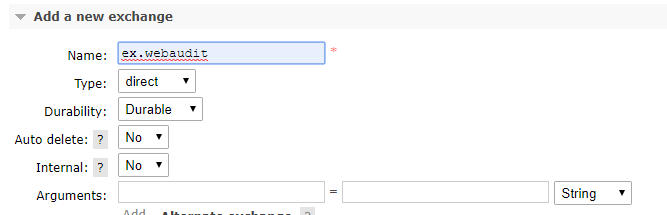
Take node of the nodeport mapped to 15672. In this case it is 30907. So RabbitMQ console can be accessed via Node01’s IP: 30907or Node02’sIP: 30907

Accessing over Node01’s IP- <http://192.168.56.108:30907/> (Use the node ip from your VM)

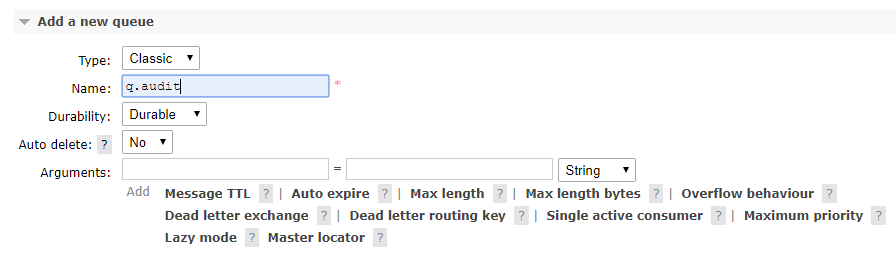


Access using guest/guest as credentials.

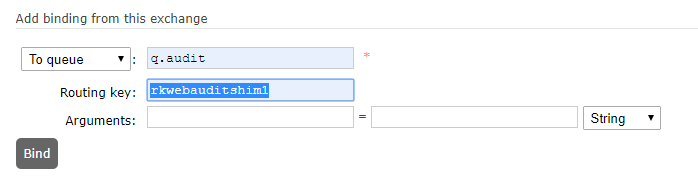
1. Create a new exchange “ex.webaudit” of type direct.



1. Create a new queue “q.audit”



1. Bind this exchange to this queue using routing key “rkwebauditshim1”



## ZIPKIN

1. Go to mschassis\k8s\zipkin
2. Run Zipkin Deployment

kubectl apply -f zipkin-dep-k8s.yaml

1. Run Zipkin Service

kubectl apply -f zipkin-svc-k8s.yaml

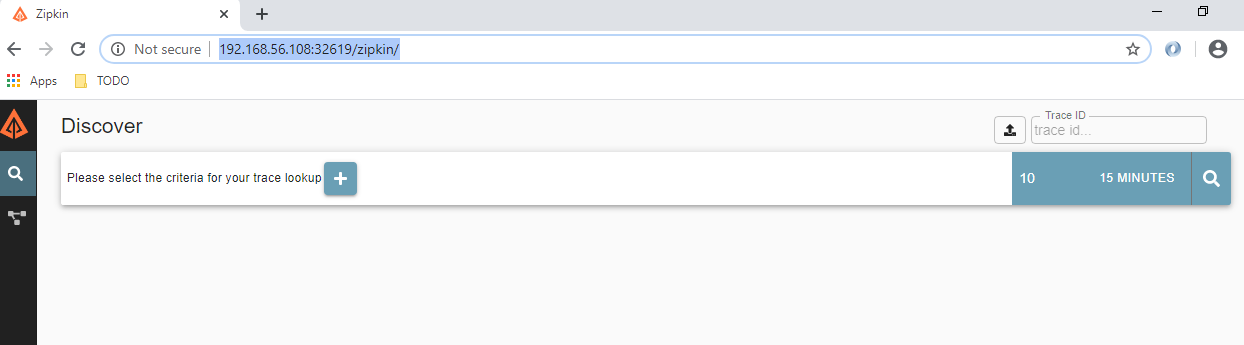
1. Check that Zipkin pods are running



1. Check that Zipkin service is running



1. Zipkin service is NodePort service. So it can be accessed externally to the NodePort mapped to 9411. Which in this case is 32619. **Internally it can accessed via service name i.e Zipkin. So internally the Zipkin URL would be Zipkin:9411.**
2. Accessing using node01’s IP(Use the node ip from your VM)



## EUREKA

1. Go to the <your-workspace>\EurekaServer(or your corresponding lab location).
2. Build the Eureka SpringBoot application

mvn clean package –DskipTests

1. Build the docker image for eureka (Use your own docker hub account name)

docker build -t kapilshekharraina/eurekaservervl01:1 .

1. Push the image to docker hub(Use your own docker hub account name)

docker push kapilshekharraina/eurekaservervl01:1

1. Deploy the Eureka pods (The K8S deployment refers to the image created in previous step)

kubectl apply -f eureka-dep-k8s.yaml

1. Run Eureka k8s service

kubectl apply -f eureka-svc-k8s.yaml

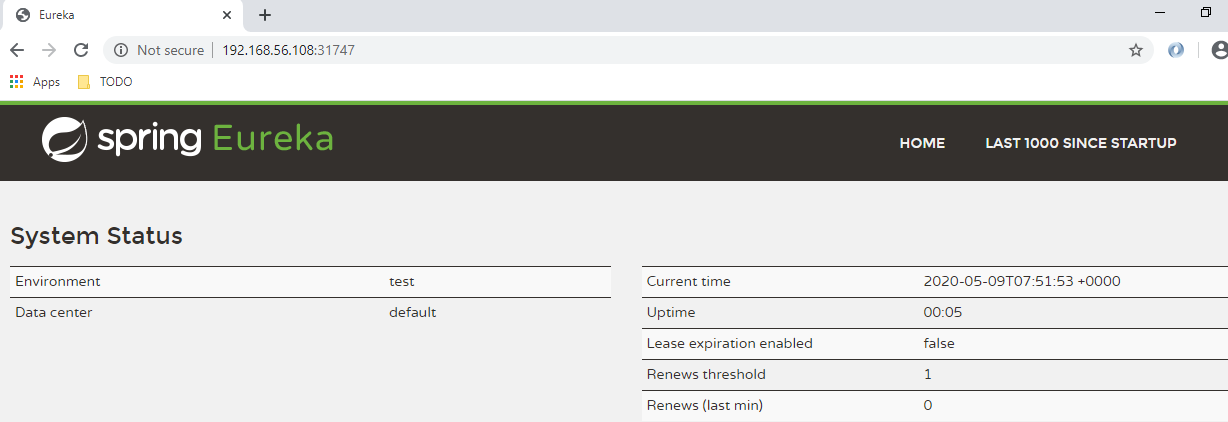
1. Check that eureka pods are running



1. Check the eureka service



1. Eureka service is NodePort type. So it can be accessed externally using the port mapped to 8761, which in this case is 31747**. Internally it can be accessed using service name eureka and 9411. So eureka:9411.**
2. Accessing over node01’s IP(Use the node ip from your VM)



## CONFIG SERVER

1. Go to <your-workspace>\ConfigServer(or your corresponding lab location)
2. Maven Build Config Server

mvn clean package –DskipTests

1. Build docker image for Config server(use your own docker account)

docker build -t kapilshekharraina/configservervl01:1 .

1. Push docker image for Config server(use your won docker account)

docker push kapilshekharraina/configservervl01:1

1. Deploy Config server pods (The K8S deployment refers to the image created in previous step)

kubectl apply -f configserver-dep-k8s.yaml

1. Run Config Server service

kubectl apply -f configserver-svc-k8s.yaml

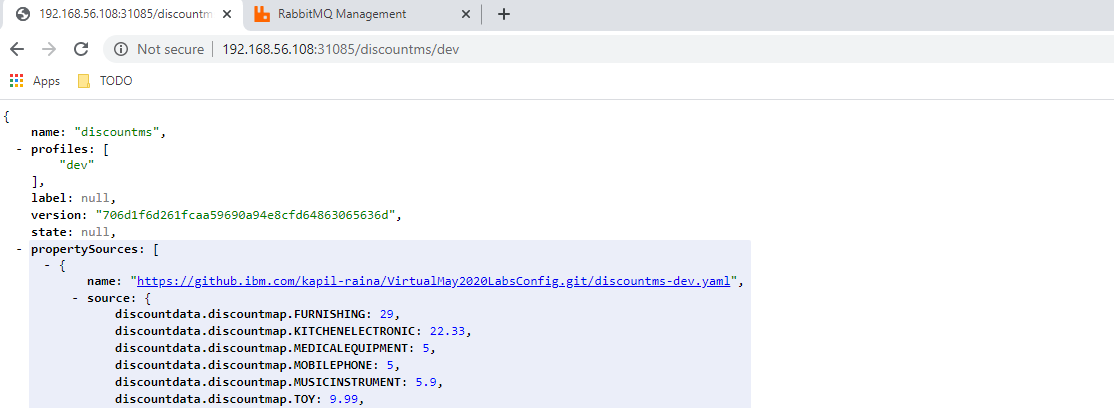
1. Check that Config server pods are running



1. Check the Config Server service



1. Config server service is NodePort service, so it is accessible externally over port mapped to 8888, which in this case is 31085**. Internally Config server can be accessed over service name i.e. configserver:8888.**
2. Accessing Config server externally over node01’s IP(Use the node ip from your VM)



## API GATEWAY

1. Go to <your-workspace>\ApiGateway.
2. Maven build apigateway

mvn clean package –DskipTests

1. Create docker image for apigateway(use your own docker hub account)

docker build -t kapilshekharraina/apigatewayvl01:1 .

1. Push the docker image for apigateway(use your own docker hub account)

docker push kapilshekharraina/apigatewayvl01:1

1. Deploy apigateway pods (The K8S deployment refers to the image created in previous step)

kubectl apply -f cloudapigateway-dep-k8s.yaml

1. Run apigateway service

kubectl apply -f cloudapigateway-svc-k8s.yaml

1. Check Config server pods



1. Check apigateway service



1. API Gateway service is NodePort and since it will always be accessed from outside only, the node port mapped to 9092 can be used with node01/02’s IP. The mapped NodePort in this case is 32366
2. Configurations would be retrieved from the Config Server for this ms.
3. Access Api Gateway externally using node01’s IP(Use the node ip from your VM)

<http://192.168.56.108:32366/product/v5/1>

## PRODUCTMS

1. Go to <your-workspace>\Productms
2. Maven build productms

mvn clean install –DskipTests

1. Create docker image for productms(use your own docker hub account)

docker build -t kapilshekharraina/productmsvl01:1 .

1. Push docker image for productms(use your own docker hub account)

docker push kapilshekharraina/productmsvl01:1

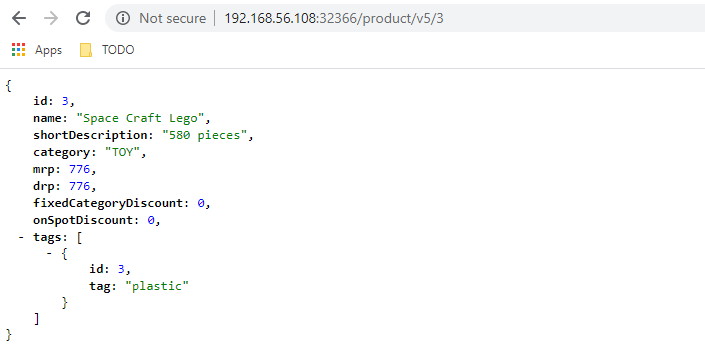
1. Deploy productms pods

kubectl apply -f productms-dep-k8s.yaml

1. Check productms pods



1. Since productms would be accessed via apigateway and apigateway employs eureka lookp, there is no need to create a k8s service for productms. The H2 DB associated with productms can be accessed via endpoint registered with Eureaka server(Eureka server would have internal pod IP , so the H2 console would need to be accessed on one of the cluster nodes).
2. Configurations would be retrieved from the Config Server for this ms.
3. Access productms using apigateway now:(Use the node ip from your VM)



Note that there is no discount applied since discountms is not available yet.

## DISCOUNTMS

1. Go to <your-workspace>\Discountms
2. Maven build discountms

mvn clean package –DskipTests

1. Create discountms docker image(use your own account)

docker build -t kapilshekharraina/discountmsvl01:1 .

1. Push discountms docker image(use your own account)

docker push kapilshekharraina/discountmsvl01:1

1. Deploy discountms pods. (The K8S deployment refers to the image created in previous step)

kubectl apply -f discountms-dep-k8s.yaml

1. Check discount ms pods



1. Since discountms would be accessed by productms via eureka service lookup, it employs client side service discovery, so a K8s service is not needed for discountms.
2. Configurations would be retrieved from the Config Server for this ms.
3. Access productms using apigateway :(Use the node ip from your VM)



Note the discount applied now.

## AUDITMS

1. Go to<your-workspace>\SimpleWebAuditMS
2. Maven build audit service

mvn clean package –DskipTests

1. Create auditms docker image(Use your own docker hub account)

docker build -t kapilshekharraina/webauditmsvl01:1 .

1. Push auditms docker image(Use your own docker hub account)

docker push kapilshekharraina/webauditmsvl01:1

1. Deploy auditms pods. (The K8S deployment refers to the image created in previous step)

kubectl apply -f webauditms-dep-k8s.yaml

1. Deploy auditms service

kubectl apply -f webauditms-svc-k8s.yaml

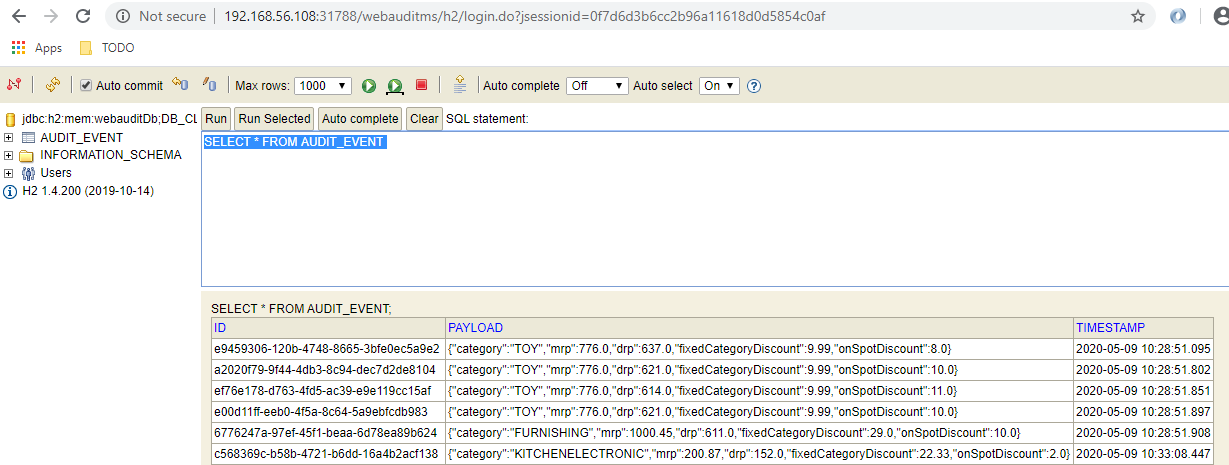
1. Check auditms pods



1. Check auditms service

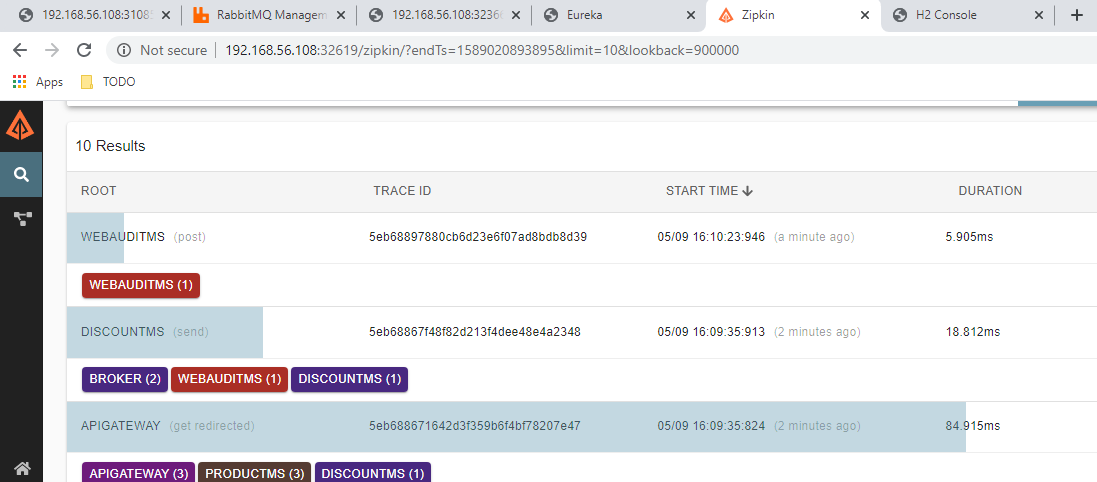


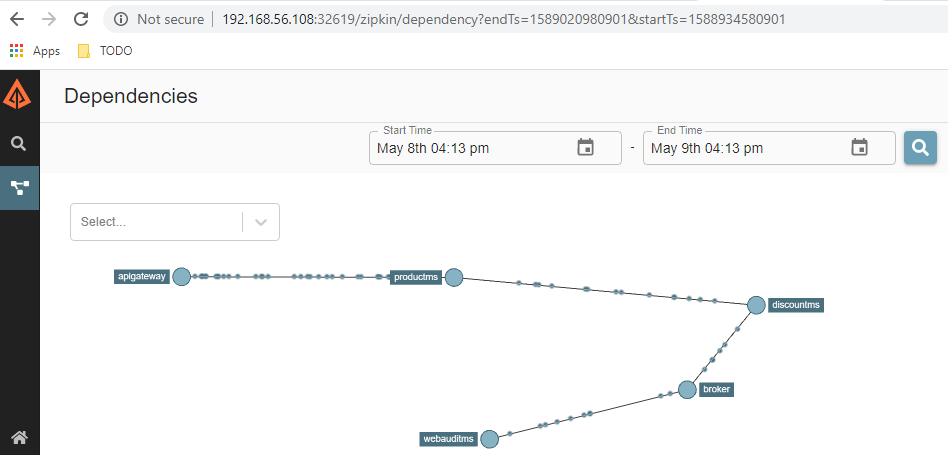
1. The webaudit service is not accessed directly for auditing standpoint. Auditing happens asynchronously. This service is exposed as NodePort to access the H2 web console from outside.
2. Configurations would be retrieved from the Config Server for this ms.
3. Access auditms H2 console from node01’s IP and NodePort 31788:(Use the node ip from your VM)



### Zipkin Test

Now Zipkin will show traces across services(Use the node ip from your VM)





### Spring Cloud Bus Test

Actuator bus-refresh would work on the configserver deployed in K8s Cluster (Use the node ip from your VM)

curl -X POST <http://192.168.56.108:31085/actuator/bus-refresh>

EFK