Kubernetes Hands-On

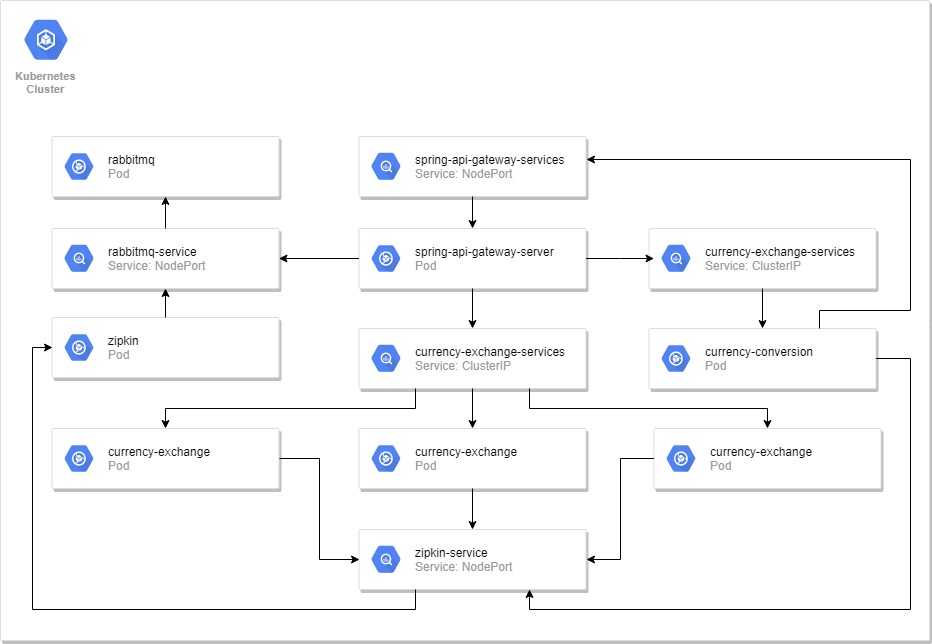
Service Discovery, Distributed Tracing, and Centralized Logging

In this tutorial, we will have practical labs related to Service Discovery, Distributed Tracing, and Deployment in Kubernetes. The technology we will use are :

1. Spring Boot & Cloud
2. Feign => Spring Cloud Library for REST Client
3. Sleuth => Spring Cloud Library for distributed tracing
4. Zipkin => Distributed Tracing Server
5. RabbitMQ => Message Queue Server
6. Zuul API Gateway => Spring Cloud library for API Gateway

# Architecture

The architecture we will use is describe below:



# Connecting to Cluster

You can choose option below:

1. Using Powershell

We can use Windows Powershell to connecting to our kubernetes cluster at 172.20.103.122, and before that, you need to install Docker for Windows first for building images later. Docker for Windows installer located here: \\cx-id2-file01\Temp\Docker

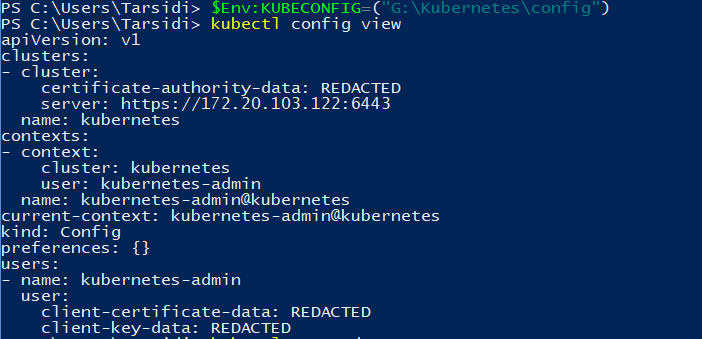
You can get config file from <https://raw.githubusercontent.com/sidie88/service-discovery-in-kubernetes/master/kube-config/config>.

1. Set kubernetes config

$Env:KUBECONFIG=("<YOUR\_CONFIG\_FILE\_LOCATION>\config")

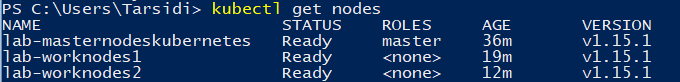
1. Check kubernetes config

kubectl config view

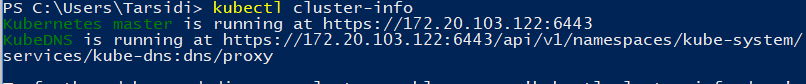


1. Check connection to kubernetes cluster

kubectl get nodes



kubectl cluster-info



1. Using Kubernetes Control Plane

Login into control plane using your credential via SSH at IP Address 172.20.103.122 and port 2220, you can ask me to create your credential to accessing kubernetes cluster.

1. Create kubernetes config cluster connection

mkdir .kube

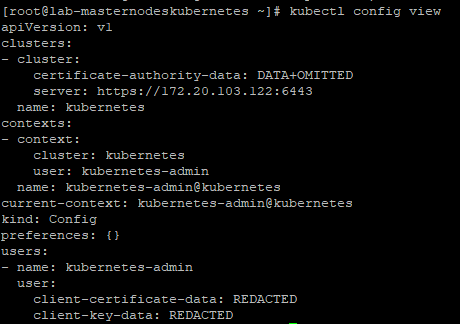
cd .kube/

curl -O <https://raw.githubusercontent.com/sidie88/service-discovery-in-kubernetes/master/kube-config/config>

export KUBECONFIG=$KUBECONFIG:$HOME/.kube/config

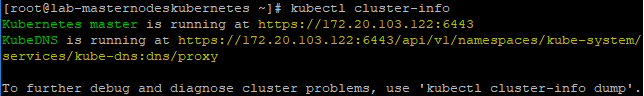
1. Checking your config file

kubectl config view



1. Checking your connection to cluster

kubectl cluster-info



# Preparation Step & Notes

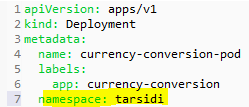
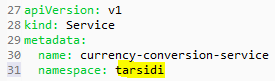
1. clone source code by following this command below:

git clone <https://github.com/sidie88/service-discovery-in-kubernetes.git>

1. Deploy pod for DNS/Service endpoint Lookup

kubectl apply -f <https://k8s.io/examples/admin/dns/busybox.yaml>

1. Please take a note, you need to add namespace to every yaml file to prevent deployment collision

1. Create your namespace

kubectl create namespaces YOUR\_NAME\_SPACE

1. Please take a note at DNS name, you need to change “default” to your namespace.

Example :

currency-exchange-service.default.svc.cluster.local **changed to** currency-exchange-service.YOUR\_NAMESPACE.svc.cluster.local

# Zipkin Distributed Server

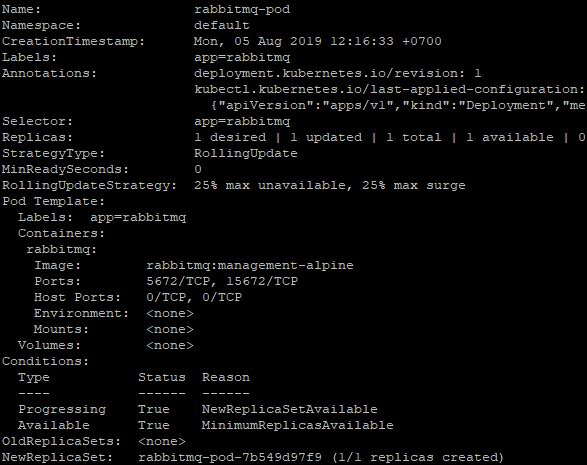
Before deploy zipkin, we need to deploy RabbitMQ first. Zipkin using RabbitMQ to tracing request from other services.

1. Deploy RabbitMQ Server
2. Navigate to distributed-tracing folder
3. Deploy RabbitMQ to Kubernetes cluster

kubectl apply –f rabbitmq.yaml

1. Check deployment status

kubectl describe deployments rabbitmq-pod –n YOUR\_NAMESPACE

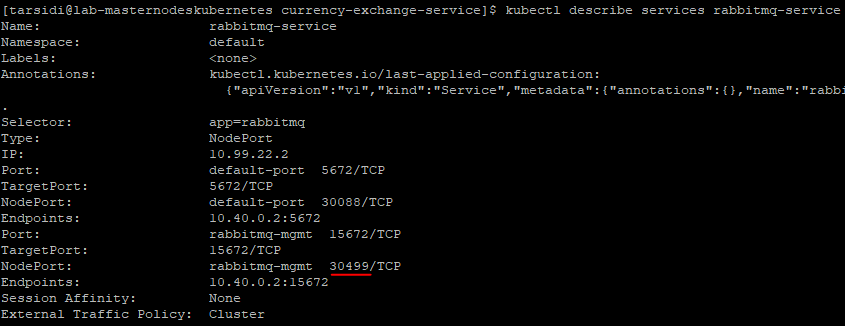


1. Check service endpoint

kubectl get services –n YOUR\_NAMESPACE



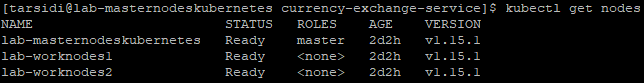
kubectl describe services rabbitmq-service –n YOUR\_NAMESPACE



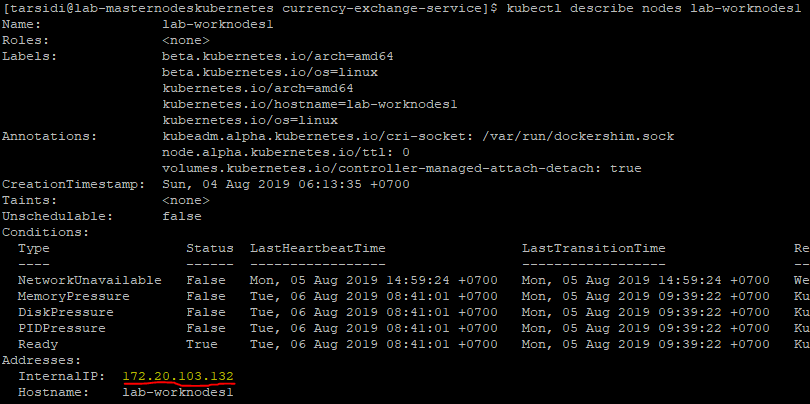
1. Check RabbitMQ Management site

You can access it using IP address from one of worker nodes and NodePort highlighted at previous step (Port: 30499). You can check worker node IP address using this command:

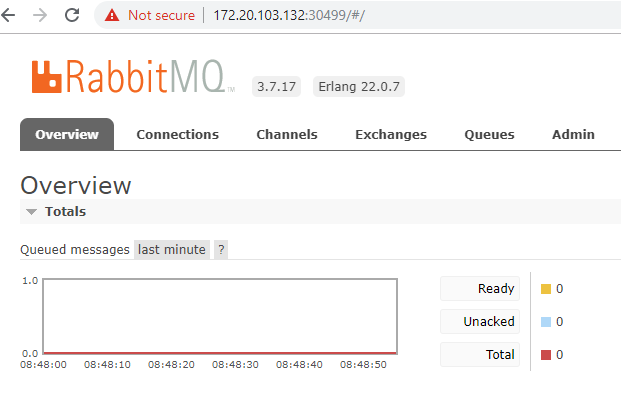
kubectl get nodes



kubectl describe nodes lab-workernodes1



After you get the IP address and port, you can open it using browser and use username “guest” and password “guest” to login.

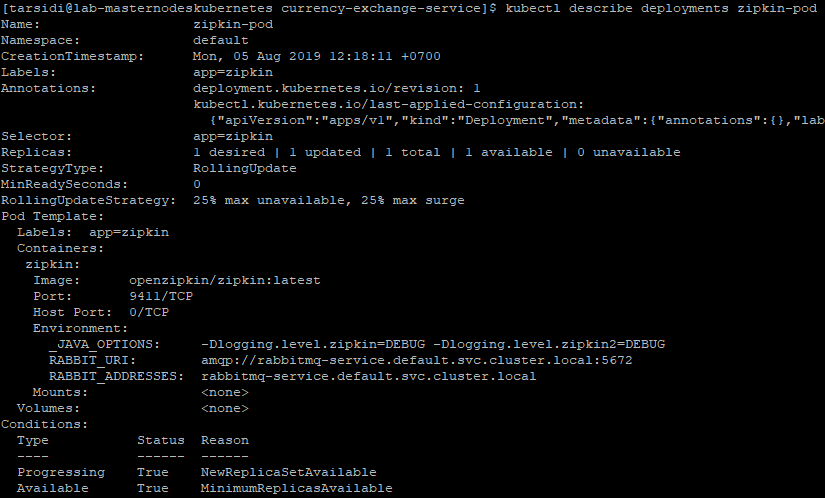


1. Deploy Zipkin Distributed Tracing Server
2. Navigate to distributed-tracing folder
3. Deploy zipkin using yaml file

kubectl apply -f zipkin.yaml

1. Check zipkin deployment

kubectl describe deployments zipkin-pod –n YOUR\_NAMESPACE

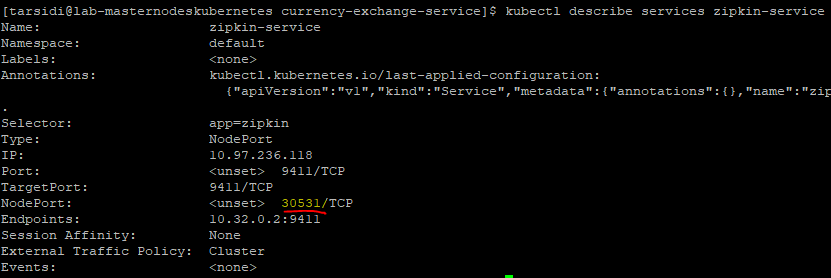


1. Check zipkin service endpoint

kubectl get services –n YOUR\_NAMESPACE

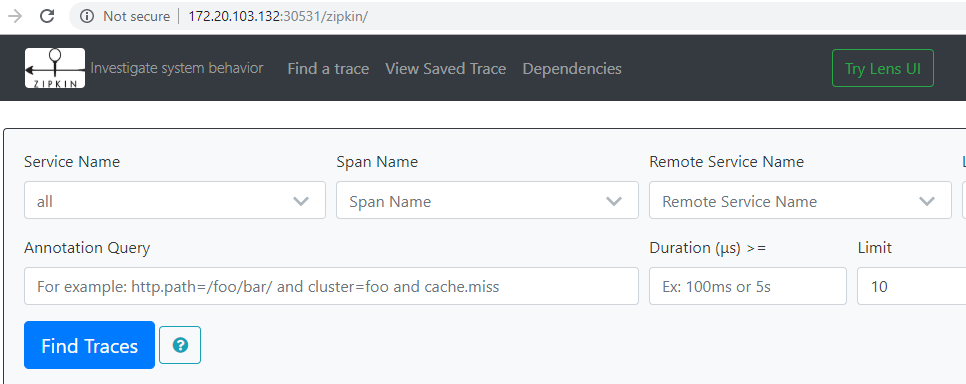


kubectl describe services zipkin-service –n YOUR\_NAMESPACE



1. Access zipking web tracing

Open your browser and use worker node IP and NodePort described at previous step.



# Currency Exchange Service

1. Navigate to currency-exchange-service folder
2. Deploy currency-exchange-service to cluster

kubectl apply -f currency-exchange-sevice.yaml

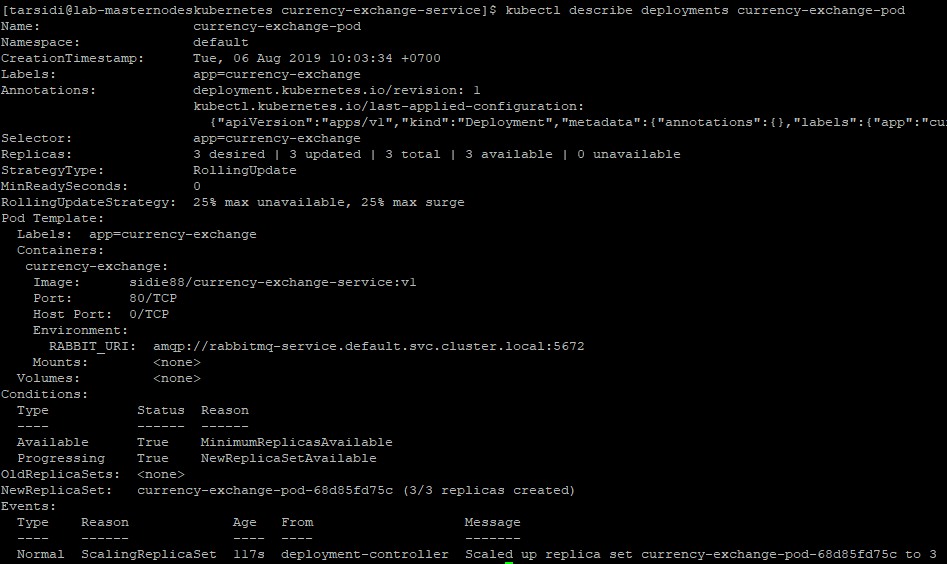


1. Check your deployment status

kubectl rollout status deployment.v1.apps/currency-exchange-pod



kubectl describe deployments currency-exchange-pod –n YOUR\_NAMESPACE



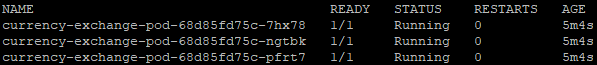
1. Check replica set

kubectl get rs --selector=app=currency-exchange –n YOUR\_NAMESPACE



1. Check pod status

kubectl get pods --selector=app=currency-exchange –n YOUR\_NAMESPACE

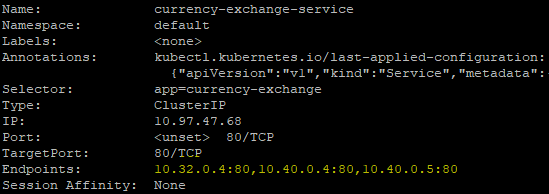


1. Check services endpoint

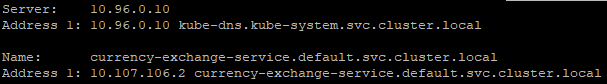
kubectl get services --field-selector metadata.name=currency-exchange-service –n YOUR\_NAMESPACE



kubectl describe services currency-exchange-service –n YOUR\_NAMESPACE



kubectl exec -ti busybox -- nslookup currency-exchange-service.YOUR\_NAMESPACE.svc.cluster.local



1. Access currency exchange service endpoint

We cannot access this endpoint directly because service type is ClusterIP, so we need to use port forwarding to access the endpoint using command below.

kubectl port-forward svc/currency-exchange-service –n YOUR\_NAMESPACE 8000:80



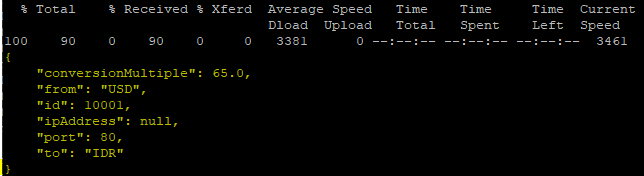
Open other terminal, and execute this command

curl localhost:8000/currency-exchange/from/USD/to/IDR

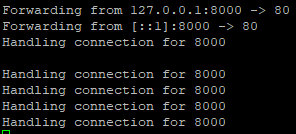


You can use pretty print to format json view

curl localhost:8000/currency-exchange/from/USD/to/IDR | python -m json.tool



At the previous terminal should be printed like this:



# API Gateway Server

For API Gateway server, we will use Netflix Zuul API Gateway Server. Every request should be through this server to centralize request and make tracing request easier.

1. Register all services to zuul routes in application.properties, line below is sample how to add service to zuul routes.

* Route path

Route path is URI used to call service through zuul api gateway. It’s format config is zuu.routes.<SERVICE\_NAME>.path=api path.

Example :

zuul.routes.currencyexchange.path=/currency-exchange-service/\*\*

Configuration above will set currency-exchange-service as URI at api gateway, so url will be: http://<ZUUL\_API\_GATEWAY\_SERVICE\_ENDPOINT>:8762/currency-exchange-service

* Route URL

Route url is the redirection from route path above.

Example:

zuul.routes.currencyexchange.url=http://currency-exchange-service.YOUR\_NAMESPACE.svc.cluster.local

Configuration above will redirect request from http://<ZUUL\_API\_GATEWAY\_SERVICE\_ENDPOINT>:8762/currency-exchange-service to <http://currency-exchange-service.default.svc.cluster.local>

You can also add service route inside kubernetes deployment file (yaml file), so when you already deploy the api gateway server, you don’t need to rebuild your docker image and update deployment.

1. Build docker image

Make sure you already inside spring-api-gateway-server folder and run this command below:

docker image build –t sidie88/spring-api-gateway-server:v1 .

docker image push sidie88/spring-api-gateway-server:v1

Please change sidie88 with your docker hub username

1. Change image name inside spring-api-gateway-server.yaml file with your image name
2. Deploy to kubernetes cluster

kubectl apply –f spring-api-gateway-server.yaml

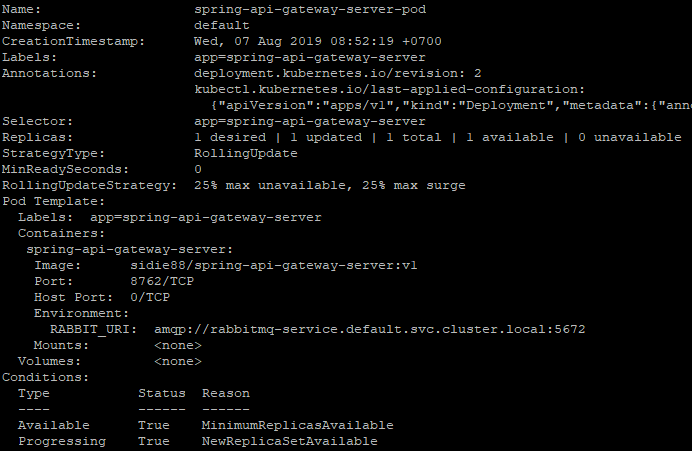


1. Check deployment status

kubectl rollout status deployment.v1.apps/spring-api-gateway-server-pod –n YOUR\_NAMESPACE



kubectl describe deployments spring-api-gateway-server-pod –n YOUR\_NAMESPACE



1. Check replica set

kubectl get rs --selector=app=spring-api-gateway-server –n YOUR\_NAMESPACE



1. Check pod status

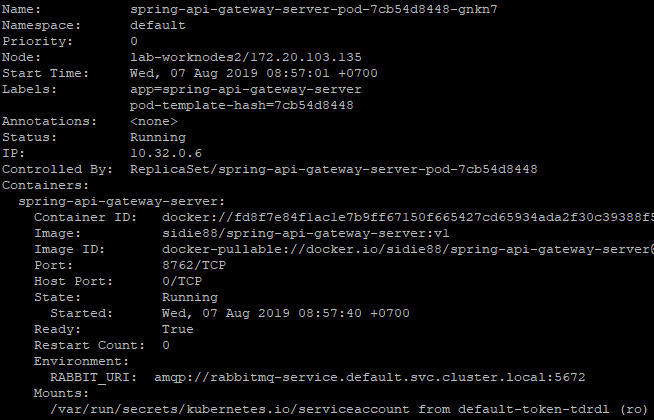
kubectl get pods --selector=app=spring-api-gateway-server –n YOUR\_NAMESPACE



More detail about pods:

kubectl describe pods spring-api-gateway-server-pod-7cb54d8448-gnkn7 –n YOUR\_NAMESPACE

Highlighted text is your pod’s name, you need to change it to your pod name which is printed in previous command.

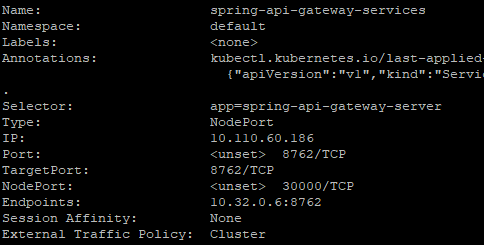


1. Check services endpoint

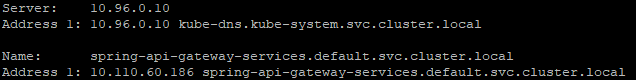
kubectl get services --field-selector metadata.name=spring-api-gateway-services –n YOUR\_NAMESPACE



kubectl describe services spring-api-gateway-services –n YOUR\_NAMESPACE



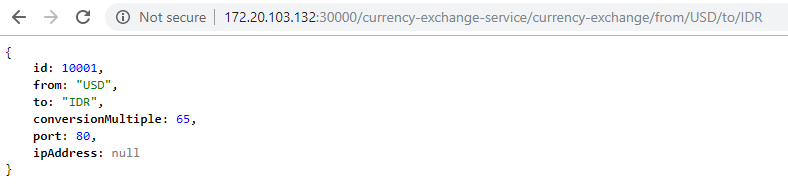
kubectl exec -ti busybox -- nslookup spring-api-gateway-services.default.svc.cluster.local



1. Test calling currency-exchange-service from api gateway server

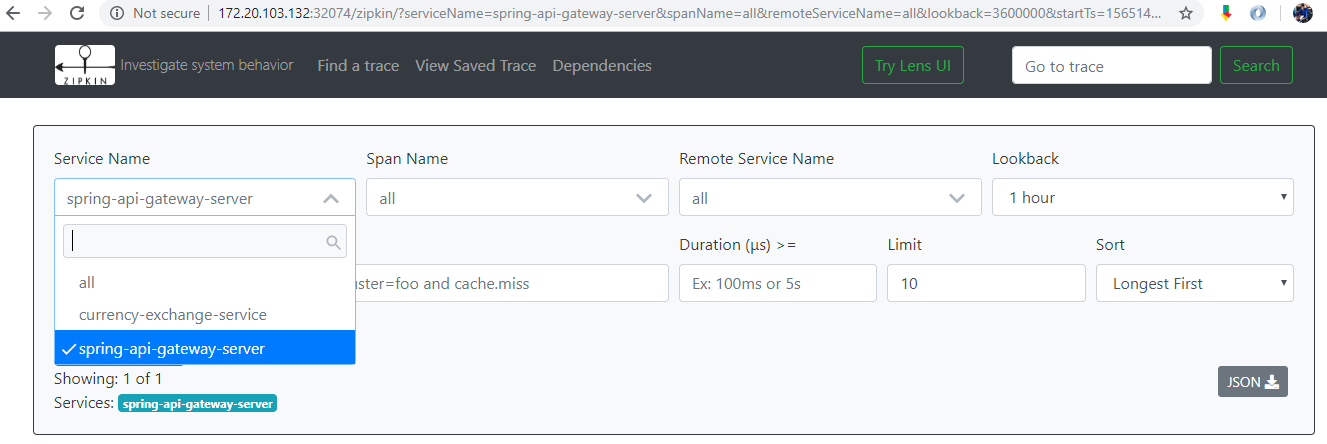
Open your browser and use worker node IP and NodePort described in yaml file.

<http://172.20.103.132:30000/currency-exchange-service/currency-exchange/from/USD/to/IDR>

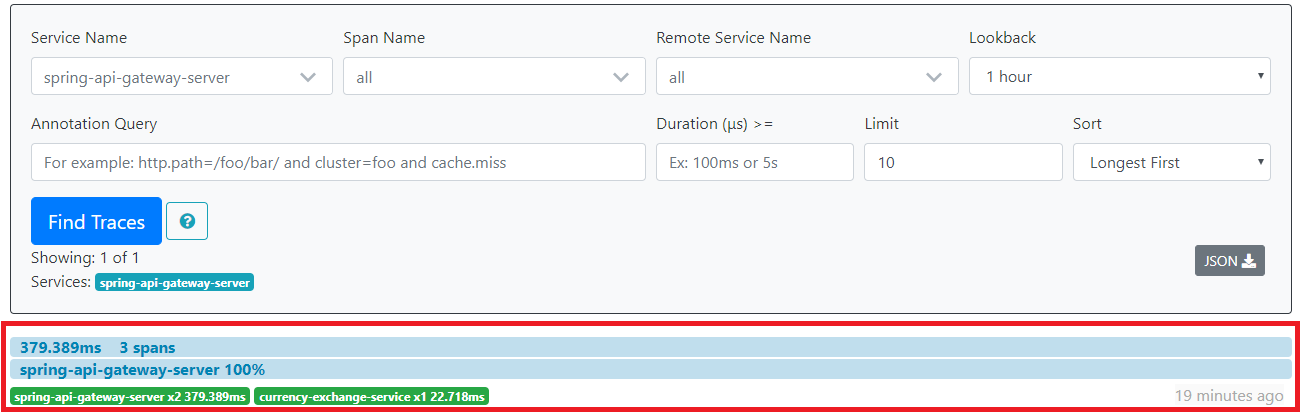


1. Trace your request in zipkin web tracing

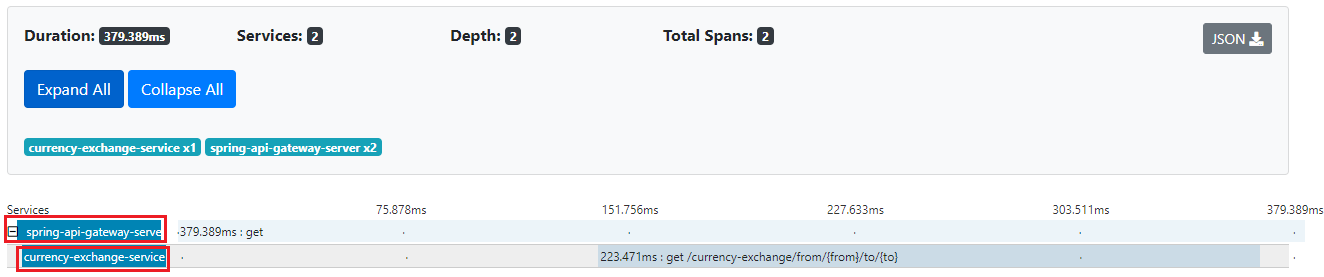
* Open this url in your browser <http://172.20.103.132:32074/>
* Select spring-api-gateway-server at Service Name drop down



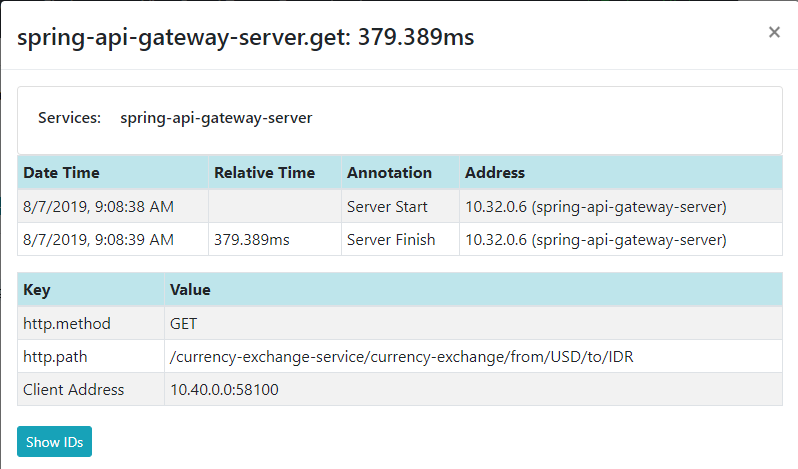
* Click “Find Traces” button

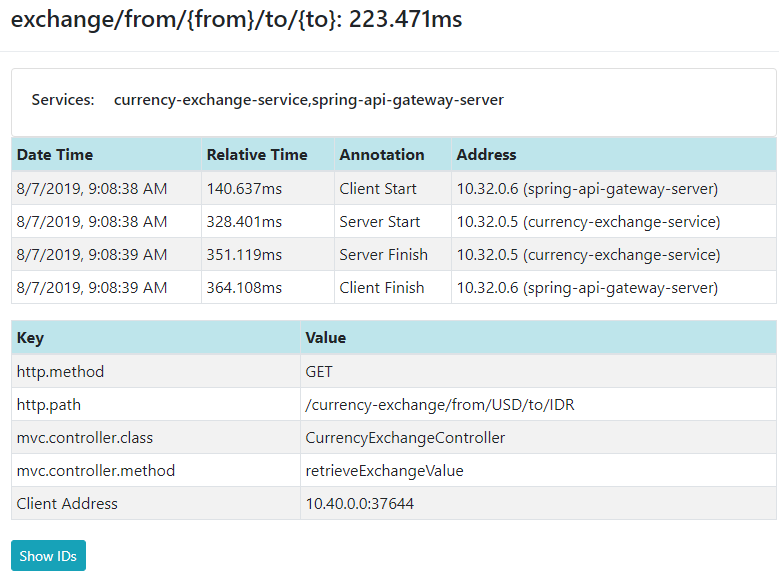


* Click on highlighted part to see details of request



* For more details, you also can click on highlighted services





# Currency Conversion Service

This pod will be calling currency-exchange-service through spring-api-gateway-server using feign client.

1. Deploy to kubernetes cluster

kubectl apply –f currency-conversion-sevice.yaml

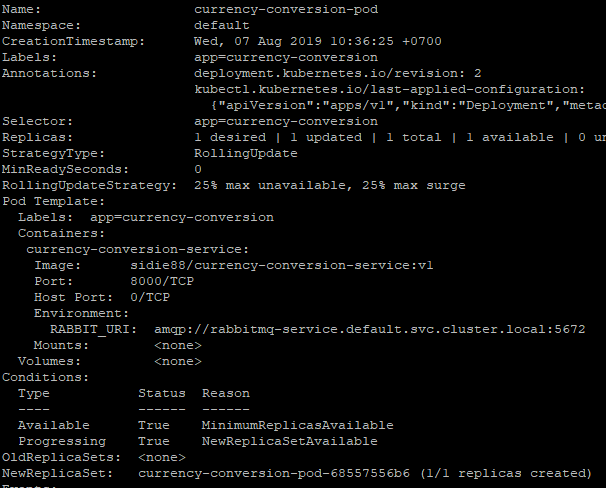


1. Check deployment status

kubectl rollout status deployment.v1.apps/currency-conversion-pod –n YOUR\_NAMESPACE



kubectl describe deployments currency-conversion-pod –n YOUR\_NAMESPACE



1. Check replica set

kubectl get rs --selector=app=currency-conversion –n YOUR\_NAMESPACE



1. Check pod status

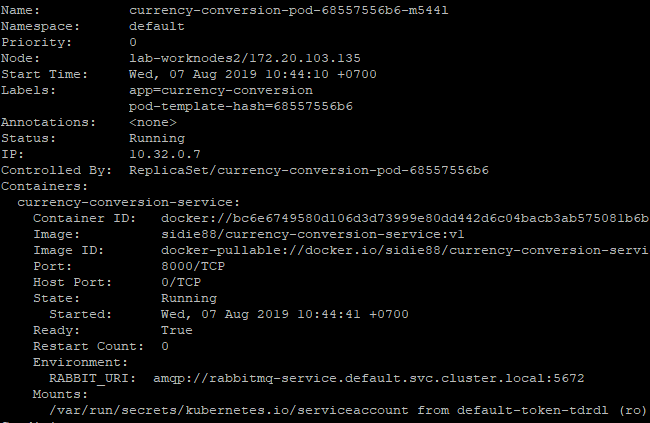
kubectl get pods --selector=app=currency-conversion –n YOUR\_NAMESPACE

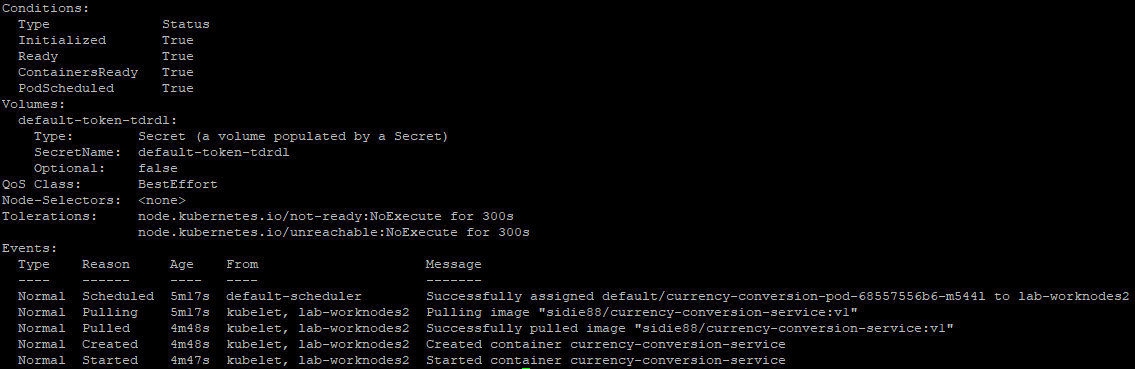


More detail about pods:

kubectl describe pods currency-conversion-pod -68557556b6-m544l –n YOUR\_NAMESPACE

Highlighted text is your pod’s name, you need to change it to your pod name which is printed in previous command.



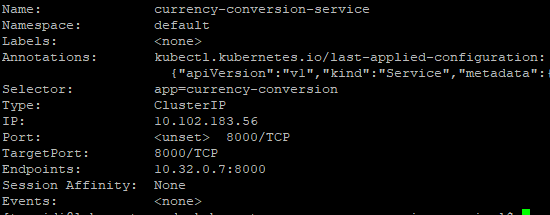


1. Check services endpoint

kubectl get services --field-selector metadata.name=currency-conversion-service –n YOUR\_NAMESPACE



kubectl describe services currency-conversion-service –n YOUR\_NAMESPACE



kubectl exec -ti busybox -- nslookup currency-conversion-service.YOUR\_NAMESPACE.svc.cluster.local

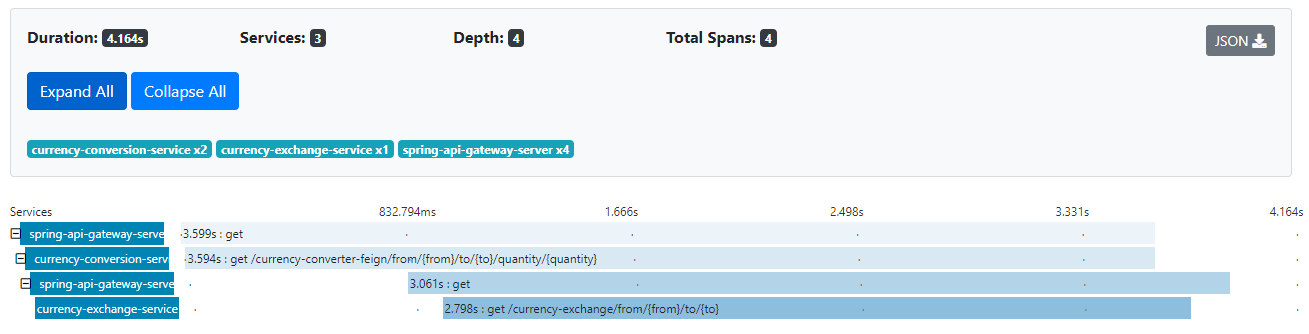
1. Test calling currency-conversion-service from api gateway server

Open your browser and use worker node IP and NodePort described in yaml file.

<http://172.20.103.132:30000/currency-conversion-service/currency-converter-feign/from/USD/to/IDR/quantity/10>

1. Trace your request in zipkin web tracing

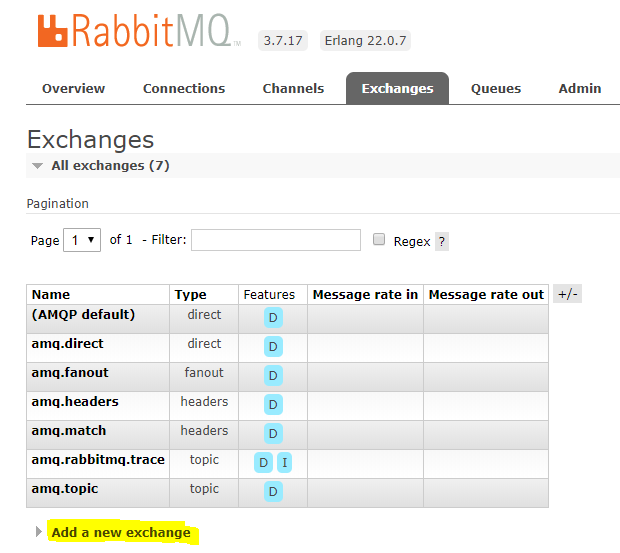
Follow step 10 at API Gateway Server section



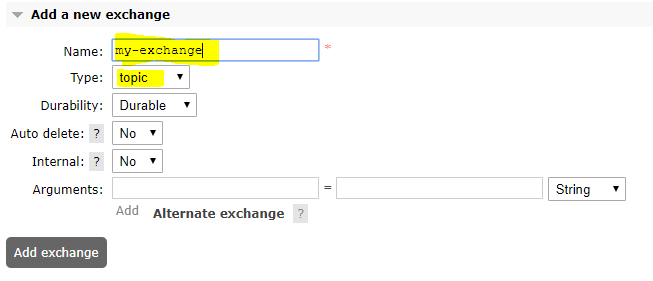
# Centralized Logging

In this section, we will centralized logging for currency-exchange-service application.

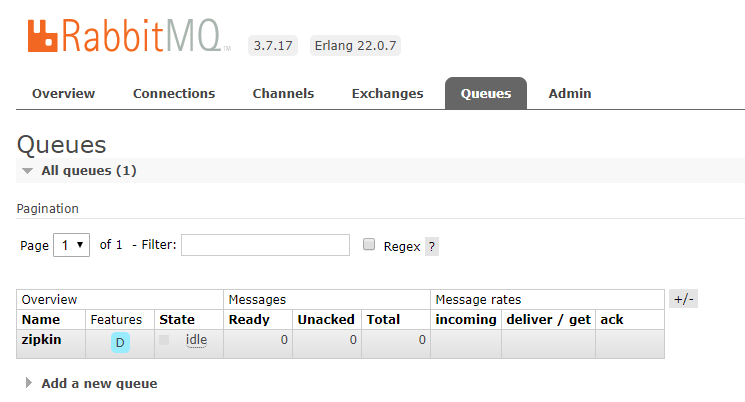
1. Login to RabbitMQ Management Web at <http://172.20.103.132:31791/> use guest as user and guest as password
2. Go to Exchanges tab, click Add a new exchange



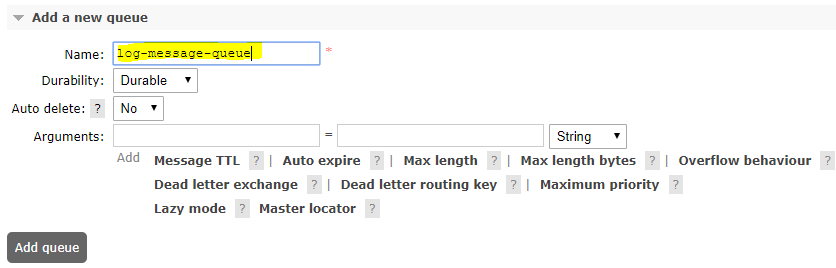
1. Type your exchange name and select “topic” on “Type” dropdown, click Add exchange



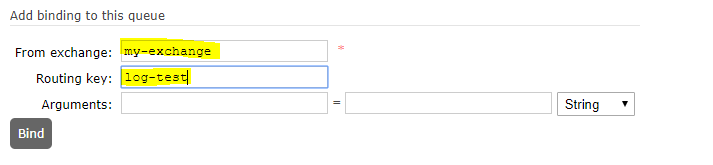
1. Go to Queues tab, click Add a new queue



1. Type your queue name, click Add queue



1. Bind queue to exchange, type From exchange with your exchange name, and routing key with “log-test”, click Bind



1. Navigate to currency-exchange-service project, and create a logback.xml file in the src\main\resource folder with the following content:

<?xml version="1.0" encoding="UTF-8"?>

<configuration debug="true">

<include resource="org/springframework/boot/logging/logback/defaults.xml"/>

<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>${HOSTNAME} ${CONSOLE\_LOG\_PATTERN}</pattern>

<charset>utf8</charset>

</encoder>

</appender>

<appender name="amqp"

class="org.springframework.amqp.rabbit.logback.AmqpAppender">

<host>rabbitmq-service.default.svc.cluster.local</host>

<port>5672</port>

<virtualHost>/</virtualHost>

<username>guest</username>

<password>guest</password>

<exchangeType>queue</exchangeType>

<exchangeName>my-exchange</exchangeName>

<applicationId>AmqpAppenderTest</applicationId>

<routingKeyPattern>logs-test</routingKeyPattern>

<generateId>true</generateId>

<contentType>text/plain</contentType>

<maxSenderRetries>2</maxSenderRetries>

<charset>UTF-8</charset>

<layout>

<pattern><![CDATA[ %d %p %t [%c] - <%m>%n ]]></pattern>

</layout>

</appender>

<logger name="com.cxrus.microservices.currencyexchangeservice"

level="DEBUG">

<appender-ref ref="amqp" />

<appender-ref ref="CONSOLE" />

</logger>

</configuration>

1. Build docker image

docker image build -t sidie88/currency-exchange-sevice:v2 .

docker image push sidie88/currency-exchange-sevice:v2

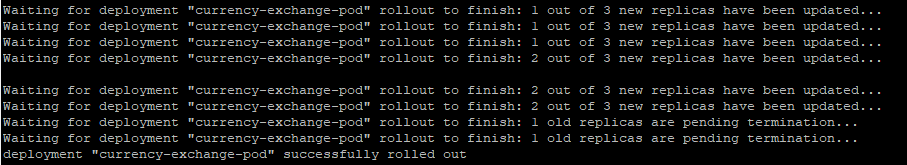
1. Update image in currency-exchange-service deployment

kubectl --record deployment.apps/currency-exchange-pod set image deployment.v1.apps/currency-exchange-d currency-exchange=sidie88/currency-exchange-sevice:v2 –n YOUR\_NAMESPACE



1. Monitor deployment process

kubectl rollout status deployment.v1.apps/currency-exchange-pod –n YOUR\_NAMESPACE



1. Test call currency-exchange from currency-conversion
2. Open RabbitMQ management web, and you can check log in Queues tab > Get messages and click Get Messages(s) button

