Under limit mS , We have dependency , For this you need to add property called this indicates where our config server are running or else application will not come up. Optional means if config server is not mandatory. Below Dependency in Limit mS will help you to connect to spring-cloud-config-server mS

spring:

config:

import:

optional:configserver:http://localhost:8888

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

In cnfig server, we can map the git repo to properties to Config server. Below properties will map git repo to config mS

Spring

cloud:

config:

server:

git:

uri: [file://C:/Users/hs3957/gi](about:blank)

**Step by Step instructions** is provided in the **troubleshooting guide** to help you troubleshoot frequently occurring problems

<https://github.com/in28minutes/spring-microservices-v2/blob/main/03.microservices/01-step-by-step-changes/microservices-v2-1.md#spring-cloud-config-server---steps-01-to-08>

Under the limit service application name should be same as git repository property file. So that when limit ms call with the ID by application will fetch the property detail from git repository.

In JPA by default the data.sql file under resources will be executed before tables are created.

We can set the property spring.jpa.defer-datasource-initialization=true to fix the issue but in Spring boot 2.4.4 I did not see this issue data.sql file is loaded with out setting the above mentioned property.

Below is the dependency for feign to be added under POM.xml to call other mS

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

Naming server or service registry is similar to load balancing it will distribute the load. In order to register all mS with Eureka client we need to add the following dependency present in naming server.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

To trigger request in UNIX for specific URI use

curl <http://localhost:8000/currency-exchange/USD/to/INR>

Below URL for configuring the circuit breaker properties

<https://resilience4j.readme.io/docs/getting-started-3>

Used to create an images for each micro service. It contains everything that is needed for mS to run .

docker run -p 5000 5000 in28min/to-do-rest-api-h2:1.0.0.RELEASE -- this is

When we run the the above command image will be downloaded from docker registory . Registory contains multiple repositoy .

The default registory is (hub.docker.com)

Image is static version it contains bytes

Container is running version

docker run -p 5000 5000 -d in28min/to-do-rest-api-h2:1.0.0.RELEASE --> -d will run the applicaiton in back ground

docker logs <full ID of container or first 5 digits > will provide the log

docker logs -f <full ID of container or first 5 digits > to tail the log

docker container ls --> will list all containers in registory

docker images --> will show images local to us

docker container -a -- this container will show all containes which are running and exited .

docker container stop <full ID of container or first 5 digits >

When we type command it is called Docker Client and it will pass it to Docker Daemon then it goes to container or Images or Repository.

Docker tag in28min/to-do-rest-api-h2:1.0.0.RELEASE:latest will tag same image id with latest tag name here it is latest

docker tag bgwms/bgwms/naming-server:0.0.1-SNAPSHOT bgwms/bgwms/naming-server:naming-server -- Rename the

docker images history <full ID of container or first 5 digits> will provide history

docker image remove <full ID of image or first 5 digits> will remove image from local

docker image remove -f 05e013711b8f -- forecefully remove

docker ps -a -f status=exited – Filter with status exited

docker search mysql , will search the mysql image is present

docker container pause <full ID of container or first 5 digits>

docker container unpause <full ID of container or first 5 digits>

All the Docker images on a system can be listed by adding -a to the docker images command. Once you’re sure you want to delete them all, you can add the -q flag to pass the image ID to docker rmi:

docker rmi $(docker images -a -q) – Will remove all images

docker container prune will stop all stopped containers from local repostory

docker run -p 5000 5000 -d --restart =always in28min/to-do-rest-api-h2:1.0.0.RELEASE --> Will restart always when docker is started

docker run -p 5000 5000-d -m 512m --cpu-quota 5000 in28min/to-do-rest-api-h2:1.0.0.RELEASE --> -m will allocate the memory to be used and --cpu-quota is CPU space allocation total cpu allocation 100000

docker system df will provide information about containers, images, local volumes

docker stats --> will provide the informaiton on CPU utilization , Memory utilization etc

Distributed Tracking - All microservices will be connected to distrubuted tracking then to data base .ZIPKIN acts as

docker run -p 9411:9411 openzipkin/zipkin:latest ---> to get zipkin image from registory and and run it

Docker Compose is a tool for defining and running multi-container Docker applications

Refer to link :https://docs.docker.com/compose/

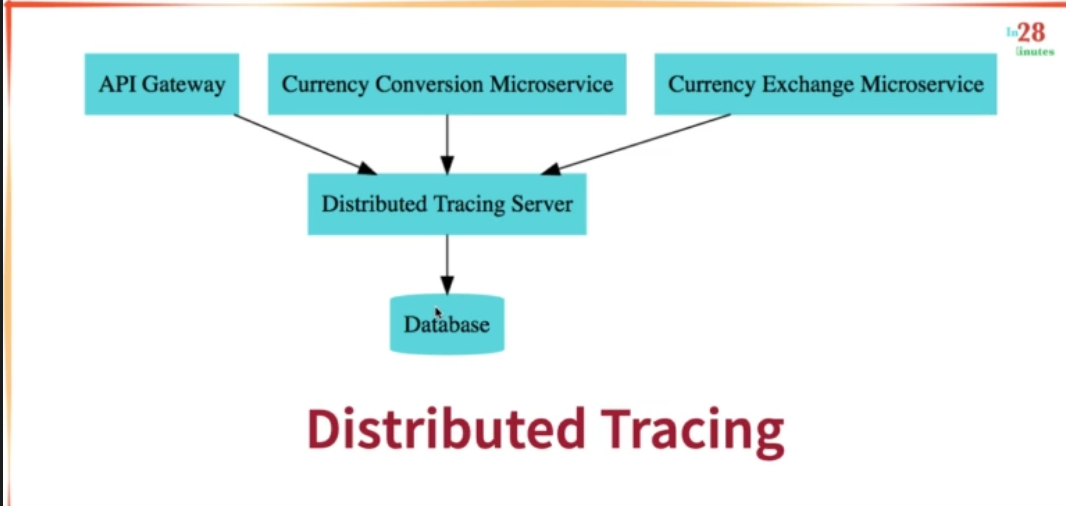
Docker container stop will gracefully shutdown the connection like disconnecting the entity , complete pending transaction etc but kill command will immediately shut down the connection

Docker can be installed from below link

Docker events command will show the event happening in docker

<https://docs.docker.com/>

Distributed tracing diagram is below



Zipkin is one of the distributed tracking system zipkin is used to track all transaction across the multiple ms . Use below command to launch the zipkin in docker

docker run -p 9411:9411 openzipkin/zipkin:2.23.

Below dependencies is needed for generating the unique transaction id and add traction to Zipkin. These dependency needed to be added under pom.xml of each mS

<!-- Below dependency will create an unique ID for each transaction for each multiple mS -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-sleuth</artifactId>

</dependency>

<!-- this dependency will add the tranctions to ZIPKIN distributed tracking system-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

We need to have Rebbit MQ in place before distributed tracking server , In case distributed tracking server is down then mS can write it to Rabmit mq from there it can be written later to distributed tracking server.

In order to create the image in docker , mS should contain the following dependency

<!-- below configaration is to build the image in docker -->

<configuration>

<image>

<name> bgwms/bgwms/${project.artifactId}:${project.version}</name>

</image>

<pullpolicy>IF\_NOT\_PRESENT</pullpolicy> <!—This will create the image if does not exists -->

</configuration>

If you are in STS, Then you need to right click on project 🡪 Run configuration. Under goals enter “spring-boot:build-image” . If you are running from command prompt then go to project folder where pom.xml exists and run the command

./mvnw spring-boot:build-image

Docker Compose is a tool for defining and running multi-container Docker applications

Refer to link :https://docs.docker.com/compose/

In order to add Rabit mq to the mS , We need to add following dependency to the pom.xml.

<dependency>

<groupId>org.springframework.amqp</groupId>

<artifactId>spring-rabbit</artifactId>

</dependency>

Kubernetes

Cluster in Kubernetes means combination of Master node and worker node. Node is where the application runs. Master nodes manages nodes

Command to deploy the image into Kubernetes is

Kubectl create deployment <application name this can be any name> --image=<path of the of the dockethub image along with tag name>

Expose the deployment command is Kubectl expose deployment <application name> --type=LoadBalancer –-port=8080

When we execute the deployment command Kubernetes creates pods, replicaset & deployment

When we execute the expose deployment Kubernetes creates the service

**PODS: It is most important thing is Kubernetes and pod is smallest deployable units**

**Pods contains the container. Each PODS has unique IP address. kubectl explain pod command will explain about the**

**Replica set basically will monitor the number of PODS running and if number PODS less than the desired PODS then it will automatically createsthe PODS. Example if you have one POD running and if you delete the POD it will automatically creates the new POD.**

**Below command will create the desired PODS for the application**

**kubectl scale deployment hello-world-rest-api --replicas=3**

**Service: The role of the service is to provide always unique URL to client in case PODS are deleted,recreated etc**

**To get the Kubernetes yaml file in to your local use below command. Get Deployment and service yaml then merge it locally**

**kubectl get deployment currency-excchange -o yaml >> deployment.yaml**

**kubectl get deployment currency-excchange -o yaml >> deployment.yaml**

**To deploy use below command , ignore warning**

**kubectl apply -f deployment.yaml**

**kubectl logs <PODID> Example kubectl logs currency-conversion-7cbcccd4c6-cz62f**

**to get current log for current transaction is**

**kubectl logs -f currency-conversion-7cbcccd4c6-cz62f**

**Look in to 208 video to enable the logging on cloud**

**To delete the pods use the command kubectl delete all -l app=currency-excchange**

**Config map acts as centralized repository where you can configure multiple environments. Get the yaml configuration from Kubernetes and add it to deployment yml.**

**In case wrong deployment is done using wrong image and it is not success, It can be reverted using below command. If you deployed the wrong image then previous deployment will continue to work.**

**kubectl rollout undo deployment currency-conversion --to-revision=2**

**kubectl rollout history deployments- Command will give the history of deployment**

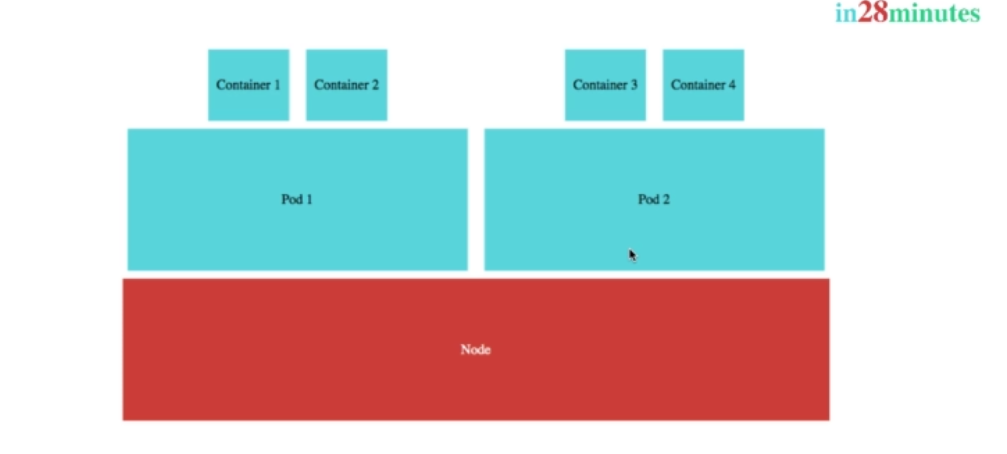
**Readyless and Liveness will be used to make sure when new code is deployed there will be fraction of seconds we may get Internal server error .In order to fix the same need to configure Readyless and Liveness in deployment.yml file**

**If readiness probe is not ready then no traffic is sent**

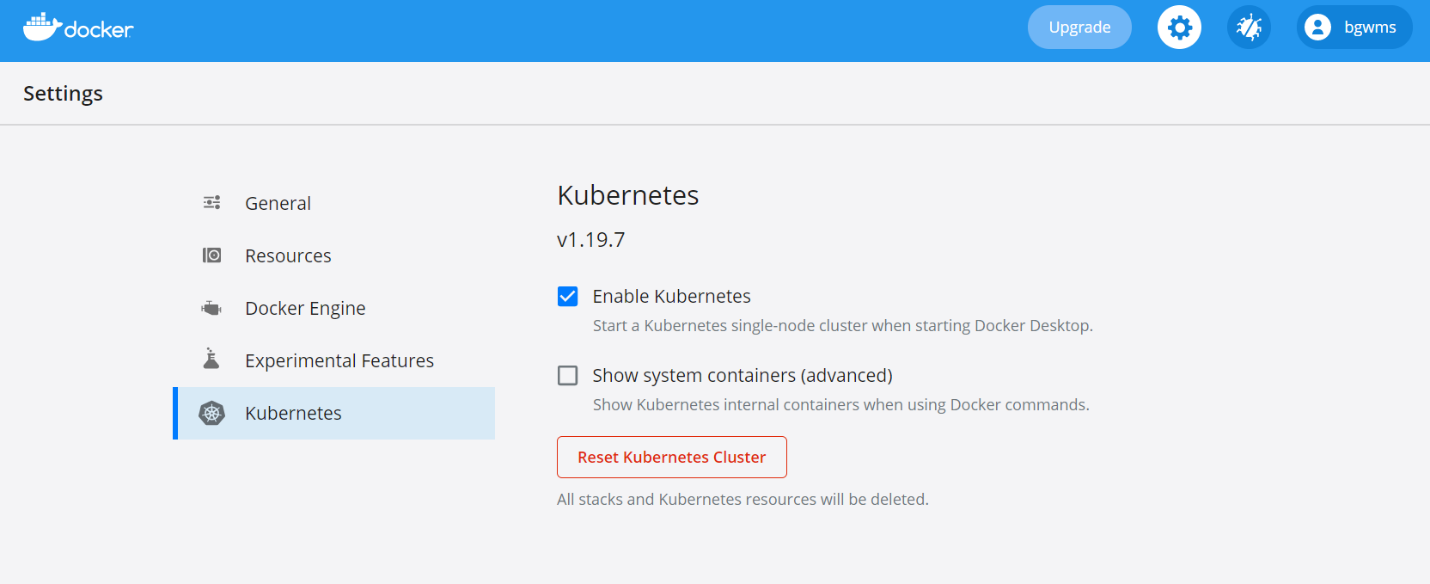
**If lieveness prope is not successful , pod is restarted**

Below command is used to auto scale the deployment basis on CPU utilization

kubectl autoscale deployment currency-exchange --min=1 --max=3 --cpu-percent=5

****

T Install Kubernetes in to your system, go to you Docker setting in lower right corner of the system and select Kubernetes and select Enable Kubernetes



After that you can download the kubectl.exe from docker hub you can set local environment variable under path by providing the exe path of kubectl.

To install the Kubernetes UI ,

1. kubectl apply -f <https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml>
2. To access Dashboard from your local workstation you must create a secure channel to your Kubernetes cluster. Run the following command:

kubectl proxy

1. After above command , Kubernetes should be running on your localhost and you can access using below URL

<http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>

1. Create An Authentication Token (RBAC)

For each of the following snippets for ServiceAccount and ClusterRoleBinding, you should copy them to new manifest files like dashboard-adminuser.yaml and use kubectl apply -f dashboard-adminuser.yaml to create them.

dashboard-adminuser.yaml file content

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kubernetes-dashboard

ClusterRoleBinding.yml file content below

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

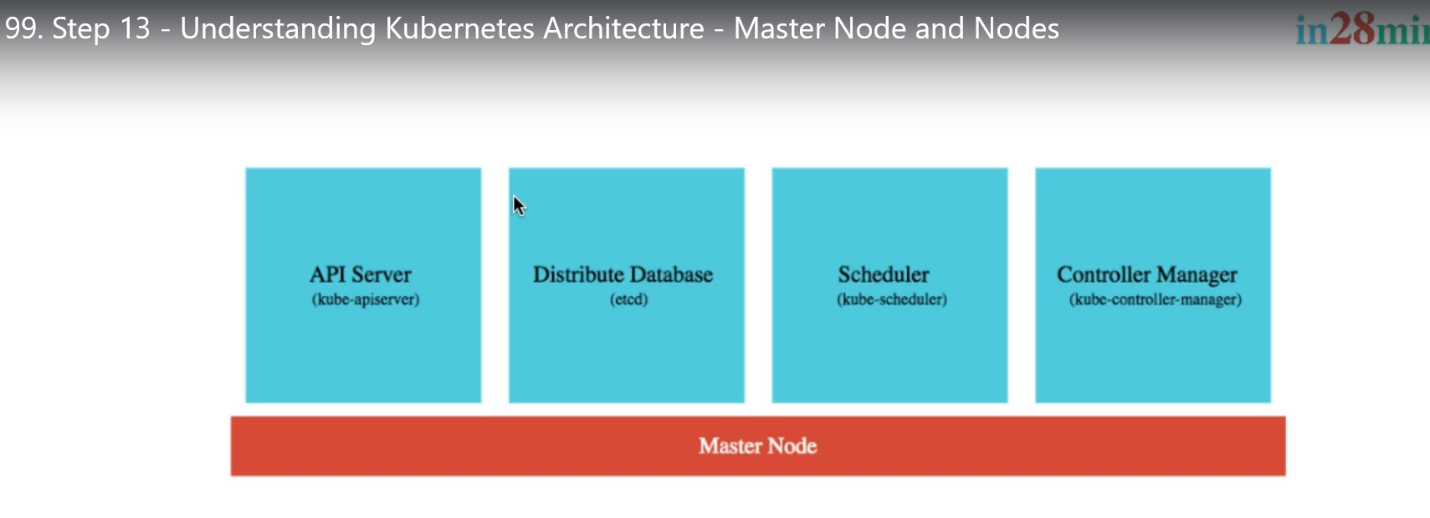
- kind: ServiceAccount

name: admin-user

namespace: kubernetes-dashboard

1. Run the command kubectl get secrets
2. kubectl describe secret <name of the dashboard> from previous command
3. Enter the token in Kubernetes UI

Kubenetes Architecture



To log in to docker from command prompt type **docker login**

Important commands

<https://github.com/in28minutes/spring-microservices-v2/tree/main/05.kubernetes#commands>

docker run -p 8080:8080 in28min/hello-world-rest-api:0.0.1.RELEASE

kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0.1.RELEASE

kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080

kubectl scale deployment hello-world-rest-api --replicas=3

kubectl delete pod hello-world-rest-api-58ff5dd898-62l9d

kubectl autoscale deployment hello-world-rest-api --max=10 --cpu-percent=70

kubectl edit deployment hello-world-rest-api #minReadySeconds: 15

kubectl set image deployment hello-world-rest-api hello-world-rest-api=in28min/hello-world-rest-api:0.0.2.RELEASE

gcloud container clusters get-credentials in28minutes-cluster --zone us-central1-a --project solid-course-258105

kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0.1.RELEASE

kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080

kubectl set image deployment hello-world-rest-api hello-world-rest-api=DUMMY\_IMAGE:TEST

kubectl get events --sort-by=.metadata.creationTimestamp

kubectl set image deployment hello-world-rest-api hello-world-rest-api=in28min/hello-world-rest-api:0.0.2.RELEASE

kubectl get events --sort-by=.metadata.creationTimestamp

kubectl get componentstatuses

kubectl get pods --all-namespaces

kubectl get events

kubectl get pods

kubectl get replicaset

kubectl get deployment

kubectl get service

kubectl get pods -o wide

kubectl explain pods

kubectl get pods -o wide

kubectl describe pod hello-world-rest-api-58ff5dd898-9trh2

kubectl get replicasets

kubectl get replicaset

kubectl scale deployment hello-world-rest-api --replicas=3

kubectl get pods

kubectl get replicaset

kubectl get events

kubectl get events --sort.by=.metadata.creationTimestamp

kubectl get rs

kubectl get rs -o wide

kubectl set image deployment hello-world-rest-api hello-world-rest-api=DUMMY\_IMAGE:TEST

kubectl get rs -o wide

kubectl get pods

kubectl describe pod hello-world-rest-api-85995ddd5c-msjsm

kubectl get events --sort-by=.metadata.creationTimestamp

kubectl set image deployment hello-world-rest-api hello-world-rest-api=in28min/hello-world-rest-api:0.0.2.RELEASE

kubectl get events --sort-by=.metadata.creationTimestamp

kubectl get pods -o wide

kubectl delete pod hello-world-rest-api-67c79fd44f-n6c7l

kubectl get pods -o wide

kubectl delete pod hello-world-rest-api-67c79fd44f-8bhdt

gcloud container clusters get-credentials in28minutes-cluster --zone us-central1-c --project solid-course-258105

docker login

docker push in28min/mmv2-currency-exchange-service:0.0.11-SNAPSHOT

docker push in28min/mmv2-currency-conversion-service:0.0.11-SNAPSHOT

kubectl create deployment currency-exchange --image=in28min/mmv2-currency-exchange-service:0.0.11-SNAPSHOT

kubectl expose deployment currency-exchange --type=LoadBalancer --port=8000

kubectl get svc

kubectl get services

kubectl get pods

kubectl get po

kubectl get replicaset

kubectl get rs

kubectl get all

kubectl create deployment currency-conversion --image=in28min/mmv2-currency-conversion-service:0.0.11-SNAPSHOT

kubectl expose deployment currency-conversion --type=LoadBalancer --port=8100

kubectl get svc --watch

kubectl get deployments

kubectl get deployment currency-exchange -o yaml >> deployment.yaml

kubectl get service currency-exchange -o yaml >> service.yaml

kubectl diff -f deployment.yaml

kubectl apply -f deployment.yaml

kubectl delete all -l app=currency-exchange

kubectl delete all -l app=currency-conversion

kubectl rollout history deployment currency-conversion

kubectl rollout history deployment currency-exchange

kubectl rollout undo deployment currency-exchange --to-revision=1

kubectl logs currency-exchange-9fc6f979b-2gmn8

kubectl logs -f currency-exchange-9fc6f979b-2gmn8

kubectl autoscale deployment currency-exchange --min=1 --max=3 --cpu-percent=5

kubectl get hpa

kubectl top pod

kubectl top nodes

kubectl get hpa

kubectl delete hpa currency-exchange

kubectl create configmap currency-conversion --from-literal=CURRENCY\_EXCHANGE\_URI=http://currency-exchange

kubectl get configmap

kubectl get configmap currency-conversion -o yaml >> configmap.yaml

watch -n 0.1 curl http://34.66.241.150:8100/currency-conversion-feign/from/USD/to/INR/quantity/10

docker push in28min/mmv2-currency-conversion-service:0.0.12-SNAPSHOT

docker push in28min/mmv2-currency-exchange-service:0.0.12-SNAPSHOT