Administración de Sistemas
Creación de una Aplicación con RabbitMQ en Docker y Kubernetes.

Gorka Cidoncha Marquiegui

4 de diciembre de 2022

Índice

1. Introducción

Para esta tarea se nos pide crear un cliente que ejecute como mínimo una tarea básica de la aplicación asignada, en este caso RabbitMQ. Además de encapsular la aplicación y el cliente como una imagen Docker. Como Tareas extras se pide crear una aplicación Kubernetes, usar volúmenes para guardar ficheros persistentes e integrar una tercera aplicación que funcione en relación a la aplicación asignada y a la cliente.

2. RabbitMQ

2.1. Descripción

RabbitMQ es un software que hace de Intermediario o *broker* en aplicaciones de mensajería. Este esta basado en colas, lo cual significa que es un servicio en el que se definen una serie de colas a las que se les puede enviar un mensaje, y a las cuales se puede conectar para recibir mensajes de la cola seleccionada. Además de poder mandar mensajes de texto puede mandar información de sistemas o tareas para iniciar otros procesos.

2.2. Funcionamiento

Los mensajes pasan por cinco estados en todo el proceso:

- Emisor:
  El Emisor o *Producer* es el encargado de publicar el menaje en un Intercambio, el cual tiene que tener unas características definidas desde el momento de su creación.

- Intercambio:
  El Intercambio o *Exchange* recibe el mensaje del emisor y se encarga de dirigir el mensaje a las diferentes Colas con las que esta conectada, dependiendo del tipo al que pertenezca este:

  - Directo:
    En el intercambio Directo o *Direct Exchange* el mensaje se enviara a las Colas conectadas con la misma clave de direccionamiento que el mensaje.
  - Tópico:
    El intercambio por Tópico o *Topic Exchange* envía el mensaje a las Colas conectadas que tengan una similitud con la clave de direccionamiento del mensaje.
  - Abanico:
    El intercambio por Abanico o *Fanout Exchange* manda el mensaje a todas las Colas conectadas.
  - Cabeceras:
    Los intercambios por Cabeceras o *Headers Exchange* usan directamente el direccionamiento del mensaje sin necesidad de ninguna conexión.

- Colas:
  Las Colas o *Queue* guardan el o los mensajes recibidos por los Intercambios hasta que el Consumidor los maneje.

- Consumidor:
  El Consumidor o *Consumer* se encarga de recibir el mensaje de la cola y manejarlo.

3. Objetivos Cumplidos

- Explicación del Funcionamiento de la aplicación asignada.
- Creación y encapsulación mediante Docker de una aplicación que funcione junto con la asignada.
- Publicación de la imagen en Docker Hub.
- Creación de un Docker Compose capaz de ejecutar las aplicaciones clientes y asignadas localmente.
- Creación de una aplicación Kubernetes para ejecutar las aplicaciones clientes y asignadas.
- Implementación de un Volumen Persistente para guardar fichero con Kubernetes.
- Integración de una tercera aplicación en la aplicación Kubernetes que funciona junto con las otras dos aplicaciones.

4. Aplicación Cliente

4.1. Descripción

Se usa el lenguaje Python, mas concretamente al versión 3.8, apoyándonos mayormente en la librería Flask para el desarrollo web, Pika para la implementación de RabbitMQ y Flask-MySQL para las conexiones con la base de datos. Además se usa HTML para la plantilla sobre la que trabaja Flask

4.2. Pseudocódigo

Para empezar la aplicación recibe vario valores mediante la variables del entorno las cuales se utilizan para hacer la conexiones pertinentes. Además con flask crea una ruta y ejecuta la plantilla index.html en ella. La aplicación recibe formularios a través de los botones y las cajas de texto de la platilla HTML. Dependiendo del botón seleccionado, Send o Recive, ejecuta diferentes partes del código.

- Send:
  Al recibir el formulario por Send, se ejecuta la función *send_msg* la cual recibe el valor de la caja del texto del formulario, esta función se encarga de conectarse con el servidor de RabbitMQ, para crear una cola en caso de no estar ya creada y enviar el mensaje de la caja de texto a dicha cola.
- Recive:
  En el caso de pulsar el botón Recive se ejecutara la función *recive_msg* hace una conexión con el servidor RabbitMQ, para crear una cola en caso de no estar anteriormente creada y consumir los mensajes de esa cola para mandarlos a la plantilla y que se muestren en esta.
  - Uso de la BD:
    En caso de que se este también haciendo uso de la versión con la base de datos, también se ejecutara *add_msg* que hace la conexión con la base de datos y guarda el mensaje en una tabla previamente creada.

5. Dockerfile

- Coge la imagen de python, concretamente python:3.8-alpine.
- Copia el fichero requiriments.txt dentro de la imagen, el cual tiene la lista de programas necesarios para el funcionamiento de la aplicación.
- Cambia el directorio de de trabajo a la carpeta app.
- Instala el software especificado en requiriments.txt.
- Copia todo el contenido de la carpeta actual al directorio /app.
- Ejecuta el fichero view.py con python.

*El fichero requiriments.txt tiene mas programas en la versión appRabbitMQ+MySQL con Base de Datos.

6. DockerCompose

6.1. RabbitMQServer

- El nombre del contenedor sera rabbitmq.
- Se coge la imagen de rabbitmq, concretamente rabbitmq:3.6-management-alpine.
- Se abre el puerto 5672 en el puerto 5672.
- Se crean las variables de entorno de el usuario y contraseña de rabbitmq.
- Se crean los para guardar los logs y la carpeta lib de rabbitmq.
- Se conecta con la network rabbitnetwork.

6.2. Web

- El nombre del contenedor sera web.
- Se coge la imagen de la aplicación cliente, concretamente docker.io/gorka0501/asrabbitmq:latest.
- Se abre el puerto 5000 en el puerto 5000.
- Se crean las variables de entorno de el usuario y contraseña de rabbitmq.
- Se conecta con la network rabbitnetwork.

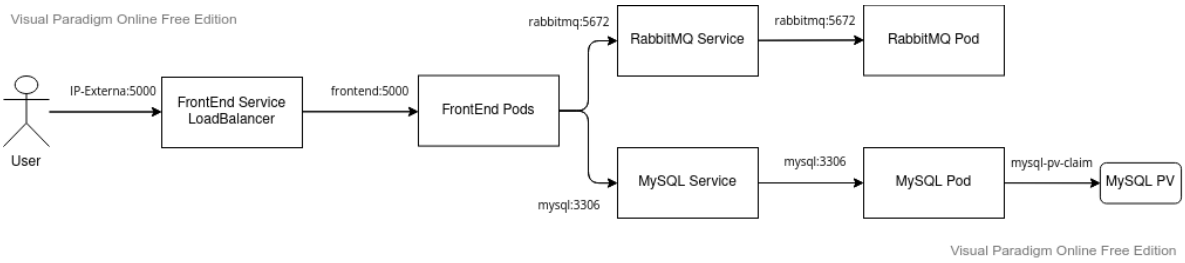
*Hay una versión llamada ASDockerRabbiMQ, en la que en vez de descargar la imagen de DockerHub se tiene la aplicación en una carpeta local y se crea la imagen desde esta.

6.3. Network

Es una red que se usa de puente para conectar las dos imágenes.

7. Kubernetes

La aplicación de Kubernetes consta de tres aplicaciones, Cliente, MySQL y RabbitMQ las cuales se comunican entre ellas, además de todos los servicios y volúmenes para que esto suceda. La aplicación RabbitMQ se comunica con la aplicación Cliente a través del servicio que abre el puerto 5672 a todo el cluster. La aplicación Cliente se abre al exterior mediante un Balanceador de Carga que proporciona una ip externa con la que mediante el puerto 5000 poder acceder a dicha aplicación. Además MySQL se comunica a través de la ip 3306 con los demás contenedores del cluster, en este caso con la aplicación cliente para guardar los mensajes recibidos. Por ultimo MySQL también tiene un Volumen montado para guardar toda la Base de Datos.



Para crear una aplicación en Kubernetes se necesitan varios ficheros yaml:

7.1. mysql-secrets

Contiene la contraseña del root en base64 del la Base de Datos.

7.2. rabbitmq-secrets

Contiene el usuario y contraseña en base64 para poder usar RabbitMQ.

7.3. rabbitmq

7.3.1. Deployment

- El nombre sera rabbitmq.
- Solo crearemos una replica.
- El nombre del contenedor sera rabbitmq.
- Se coge la imagen de rabbitmq, concretamente rabbitmq:3.6-management-alpine.
- Se le piden 100m para CPU y 100Mi de memoria al cluster.
- Se abre el puerto 5672.
- Se crean las variables de entorno de el usuario y contraseña de rabbitmq con los valores de rabbitmq-secrets.

7.3.2. Service

Se abre el puerto de 5672 rabbitmq en el puerto 5672 para el resto del cluster.

7.4. mysql

7.4.1. Deployment

- El nombre sera mysql.
- Solo crearemos una replica.
- El nombre del contenedor sera mysql.

- Se coge la imagen de rabbitmq, concretamente mysql:8.0.
- Se le piden 5000m para CPU y 1G de memoria al cluster.
- Se abre el puerto 3306.
- Se crea la variable de entorno de la contraseña de root de mysql con el valor de mysql-secrets.
- Se monta el el volumen mysql-pv en /var/lib/mysql.
- Se asigna el volumen reclamado por mysql-pv-claim como mysql-pv.

7.4.2. Service

Se abre el puerto de 3306 rabbitmq en el puerto 3306 para el resto del cluster.

7.5. volumen-persistente

Se reclama un volumen de 5Gi para escribir y leer desde un pod con el nombre mysql-pv-claim, que al estar utilizando gke-autopilot crea el volumen pedido automáticamente.

7.6. app

7.6.1. Deployment

- El nombre sera frontend.
- Crearemos tres replicas.
- El nombre del contenedor sera rabbitmq-webapp.
- Se coge la imagen de asrabbitmqmysql, concretamente docker.io/gorka0501/asrabbitmqmysql:latest.
- Se crea la variable de entorno de la contraseña de root de mysql y las de usuario y contraseña para el uso de RabbitMQ, con el valor de mysql-secrets y rabbitmq-secrets y la variable dbname con el valor myData.
- Se le piden 100m para CPU y 100Mi de memoria al cluster.
- Se abre el puerto 5000.

7.6.2. Service

Se crea un Service de tipo Balanceador de Carga *LoadBalancer* el cual abrirá el puerto 5000 de frontend al exterior y dividirá las peticiones a través de los tres containers de frontend.

8. Archivos

8.1. Apps

Aplicaciones Cliente desarrolladas.

8.1.1. appRabbitMQ

Aplicación Cliente con la funcionalidad de enviar y recibir mensajes mediante RabbiMQ:
appRabbitMQ

8.1.2. appRabbitMQ+MySQL

Aplicación Cliente con la funcionalidad de enviar y recibir mensajes mediante RabbiMQ y guardar los mensajes recibidos en una base de datos previamente creada:
appRabbitMQ+MySQL

8.2. DockerCompose

DockerCompose para la puesta en marcha del todo el entorno la funcionalidad de enviar y recibir mensajes mediante RabbiMQ.

8.2.1. ASDockerHubRabbitMQ

DockerCompose para poner en marcha el entorno, cogiendo la imagen de la web:
ASDockerHubRabbitMQ

8.2.2. ASDockerRabbitMQ

DockerCompose para poner en marcha el entorno, construyendo la imagen desde la maquina local:
ASDockerRabbitMQ

8.3. Kubernetes

Carpeta con los YAML necesarios para crear una aplicación Kubernetes un exec.sh para ponerlo en marcha y un redme.md con la instrucciones para un correcto funcionamiento.

8.3.1. ASKubernetesRabbitMQ

Crea una aplicación con las mismas funcionalidades que los DockerCompose:
ASKubernetesRabbitMQ

8.3.2. ASKubernetesRabbitMQ+MySQL

Crea una aplicación con con las mismas funcionalidades que los DockerCompose mas la inclusión de una base datos y un volumen persistente para guardar esta:
ASKubernetesRabbitMQ+MySQL

Referencias

[1] GitHub

[2] RabbitMQ

[3] DockerHub

[4] GoogleCloud

[5] Kubernetes

[6] MySQL