MASTER UNIVERSITARIO EN DESARROLLO ÁGIL DE SOFTWARE PARA LA WEB

**ASIGNATURA: CLOUD COMPUTIING Y CONTENEDORES**

CURSO 2021-22

**TRABAJO FINAL**

**JUAN ANTONIO MUÑOZ GÓMEZ (71704175L)**

[1. Justificación de proyectos y herramientas. 3](#_Toc107186934)

[2. Estructura del proyecto 4](#_Toc107186935)

[3. Microservicios 5](#_Toc107186936)

[3.1 Versiones: Diferencia entre ellas 5](#_Toc107186937)

[4. Despliegue 7](#_Toc107186938)

[4.1 Local 7](#_Toc107186939)

[4.2 Contenedores 7](#_Toc107186940)

[4.3 Kubernetes 8](#_Toc107186941)

[4.4 Esquema de la estructura del proyecto 9](#_Toc107186942)

[5. Proceso y Resultado. 11](#_Toc107186943)

# Justificación de proyectos y herramientas.

Para la realización de la práctica final de esta asignatura se ha desarrollado un proyecto de temática personal, cuyo despliegue debe ser implementado con las tecnologías Docker y Kubernete, Istio y Kiali para una correcta automatización y monitorización del proyecto.

La temática elegida consiste en la implementación de un sistema de cambios de precios en bebidas. Este sistema pensado para la hostelería simula un entorno bursátil, donde los precios sufrirán quiebras durante X minutos y así los clientes se verán llamados a adquirir el producto rebajado. Una vez termine dicha quiebra, la cuál denominaremos “crack” en el documento, el precio volverá a su estado inicial.

En este documento explicaremos brevemente la arquitectura del proyecto, la relación entre los diferentes microservicios y las diferencias entre las versiones implementadas de estos.

# Estructura del proyecto

La entrega contará con un total de tres carpetas encontradas en el primer nivel accesible. La primera de ellas es la herramienta *istio*, colocada aquí por comodidad a la hora de trabajar en el desarrollo de la prueba. La segunda de ellas, *microservices*, contendrá todas las carpetas de los microservicios utilizados. Cada carpeta indica el nombre del microservicio y su correspondiente versión. La última carpeta, llamada *kubernetes*, contiene todos aquellos ficheros .yml necesarios para realizar el despliegue automático del proyecto. Por último, en este mismo nivel se pueden encontrar dos scripts bash que recogen los comandos utilizados tanto en la fase de dockerización de los microservicios (paso de local a Docker) como en el proceso de despliegue automático de kubernetes (paso de Docker manual a Kubernetes). Estos se llaman *Docker\_manual.sh* y *comandos.sh* respectivamente.

La organización es la siguiente (los directorios acaban en el carácter ‘/’):

.

├── comandos.sh

├── docker\_manual.sh

├── istio-1.14.1/

│ └── ...

├── kubernetes/

│ ├── enrouting\_50-50.yaml

│ ├── enrouting\_specific.yaml

│ ├── enrouting\_v1.yaml

│ ├── enrouting\_v2.yaml

│ ├── gateway.yaml

│ ├── routing\_v1.yaml

│ ├── virtual\_service\_frontend.yaml

│ ├── virtual\_service\_router.yaml

│ ├── deployments/

│ │ └── ...

│ └── services/

│ └── ...

└── microservicios/

├── Cracks-microservice-v1/

├── Cracks-microservice-v2/

├── Drinks-microservice-v1/

├── Drinks-microservice-v2/

├── Frontend-microservice-v1/

├── Frontend-microservice-v2/

├── Router-microservice-v1/

└── Router-microservice-v2/

# Microservicios

Se han implementado un total de 4 microservicios:

* **Frontend**: Es el encargado de pintar visualmente la lista de precios. No se implementa ninguna funcionalidad PUT/POST, ya que el sistema está pensado para que se vea en una pantalla de bar, y las acciones se realicen con un botón físico (implementado con un Arduino y relacionado por MQTT o un Websocket, por ejemplo) de manera que sea mucho más visual y atractivo al cliente.
  + **Puerto**: 4005
  + **Métodos**: Get (llama al router)
* **Router**: Encargado de relacionar los demás microservicios. Es el puente de unión entre el front y el back, así como la relación entre los distintos microservicios del back. Su función básicamente es enrutar las tramas a los diferentes receptores y devolver los resultados que reciba de estos.
  + **Puerto**: 4002
  + **Métodos**: CRUD de Drinks, CRUD de Cracks. Algunos de los métodos más especiales que encontramos son los de crear/eliminar crack, ya que debe almacenar la información comunicándose con cracks, y actualizar los precios comunicándose con drinks. Estos métodos se comunican con *DRINKS* y *CRACKS* respectivamente.
* **Drinks**: Microservicio que podemos tratar como Model de la base de datos que almacena las bebidas (MariaDB). Sus funciones son recuperar, crear, actualizar y eliminar [CRUD] las bebidas de forma que se cumpla con la funcionalidad pensada para el proyecto. Contiene un método para instanciar una serie de bebidas por defecto y guardarlas en la DB.
  + **Puerto**: 4000
  + **Métodos**: CRUD de Drinks
  + **DB relacionada**: MariaDB
* **Cracks**: También se puede tratar como Model, en este caso de la base de datos de quiebras o “cracks” (Mongo). Al igual que su hermano gemelo, se encarga de cumplir la funcionalidad por medio de un CRUD asociado a las quiebras.
  + **Puerto**: 4000
  + **Métodos**: CRUD de Cracks
  + **DB relacionada**: Mongo

Estos microservicios, unidos a las bases de datos, conforman el ecosistema del proyecto realizado. En capturas posteriores veremos la organización de estos de manera visual, pero para ello antes debemos explicar un par de puntos más.

## Versiones: Diferencia entre ellas

Se ha realizado una segunda versión para cada uno de los microservicios. Estos son los cambios entre las diferentes versiones:

* **Frontend**
  + **V1**: Pinta los precios en rojo y verde dependiendo del valor inicial.
  + **V2**: Cambio en los colores: ahora son azul y amarillo.
* **Router**
  + **V1**: Su funcionamiento normal distribuye la comunicación hacia el microservicio requerido sin añadir ningún campo nuevo.
  + **V2**: En esta versión, se añade un campo *isPerc* en algunas respuestas, indicando que se realizará la quiebra de forma porcentual y no de forma “flat”.
* **Drinks**
  + **V1**: A la hora de hacer un get de la lista de bebidas, solo trae la lista. Para aplicar un crack, aplicará el precio en plano (flat): con esto se quiere decir que si recibe 2.5 como dato, pondrá todos los precios a 2.50€, independientemente del valor que tuvieran antes.
  + **V2**: En esta ocasión, al hacer el get de la lista de bebidas se trae tanto la lista como el número total de las mismas en el objeto json. Los cracks en este caso se aplican de manera porcentual: si en esta ocasión se vuelve a recibir 2.5, se aplicará un descuento del 2,5% del precio original.
* **Cracks**
  + **V1**: A la hora de hacer un get de la lista de cracks, solo trae la lista. Al almacenar el crack se pone el campo *isPerc = false*.
  + **V2**: En esta ocasión, al hacer el get de la lista de cracks se trae tanto la lista como el número total de las mismas en el objeto json. Al almacenar el crack se pone el campo *isPerc = true*.

# Despliegue

Inciso: Cuando se habla de un microservicio, se omite decir que se habla de las dos versiones. Quiero dejar claro esto ya que, para saltar de un paso a otro, se han probado antes las dos versiones.

## Local

Por supuesto, la hoja de ruta y por tanto la manera de desplegar los microservicios primero pasa por desarrollarlos y desplegarlos de manera local. Utilizando la extensión Live Server de VSCode así como la funcionalidad de despliegue del propio npm.

Una vez probado y refinado el funcionamiento de los microservicios (abreviados “MS”) en local, se debe dar el siguiente paso: contendorizar. Primero se hizo este proceso con las bases de datos (abreviados “DB”), probando que la comunicación entre los MS y las DB fueran correctas. Una vez hecho esto, pasamos al siguiente paso.

## Contenedores

Los microservicios mencionados anteriormente se desplegarán en forma de contenedores para el resultado final de la práctica. Es necesario crear un archivo *Dockerfile* donde se le indique las instrucciones a la hora de crear las imágenes de los mismos. En este caso, se meterán los proyectos dentro de un contenedor originalmente de [Node@16.0](mailto:Node@16.0), añadiendo la capa personalizada de nuestro proyecto a dicha imagen para así poder desplegar contenedores personalizados con el proyecto. Las instrucciones le indican que ignore la carpeta *modules* de npm (.*dockerignore*), y por tanto que luego haga un *npm i* para obtener dichos paquetes. Esto se hace por comodidad a la hora de subir la práctica y compartirla de manera más ligera.

Una vez creados los cuatro contenedores, debemos cambiar los parámetros de entorno ya que la comunicación entre los contenedores ahora no puede ser llamando a localhost, si no que debe ser una relación a los propios contenedores utilizando el nombre que les hemos dado. De esta manera, al desplegarlos en una network que creemos, la relación no dependerá de estar en el localhost, y podremos incluso descentralizar la comunicación de estos sin depender de la misma máquina.

Se deja constancia de este paso en el fichero *docker\_manual.sh* donde se recogen los comandos que se utilizaban para este paso. Los apartados son los siguientes:

* **Reseting**: Limpieza de la terminal y borrado todos los contenedores actuales. Se comenta una línea en la que se eliminarían las imágenes inutilizadas, en caso de que quedara alguna así.
* **Docker Commands**
  + **Images**: Creación de las imágenes por medio del comando *Docker build*. Se comenta la versión que no se quiere crear en este momento para mayor rapidez a la hora de probar.
  + **Network inizalication**: Levantamiento de la network.
  + **Container Instances**: Levanta Databases y luego los microservicios de la versión que no esté comentada. También llama a los métodos de creación de datos en los microservicios *drinks* y *cracks*.
* **End & Clear Screen**: Limpieza de la terminal.

## Kubernetes

El gran salto del proceso se produce en este punto. Todo lo que se hacía de forma “manual” en el paso anterior, debe ser automatizado utilizando kubernetes (+ istio). Se omite explicar el proceso de instalación para centrarnos en detalle en las operaciones que deben ser realizadas.

Las operaciones detalladas se pueden consultar en el fichero *comandos.sh*, el cuál recoge los comandos finales (que también son los utilizados en el documento multimedia explicativo que se sube junto con este documento). Estas son las secciones que se pueden encontrar en el fichero:

* **Reseting**: Limpieza de la terminal y borrado todos los contenedores actuales. Se comenta una línea en la que se eliminarían las imágenes inutilizadas, en caso de que quedara alguna así. Una vez finalizado esto, se hace una limpieza de todos los pods creados (deployments + services), así como de los gateaway, virtual services y balanceadores utilizados.
* **Istio**
  + **Istio Configuration**: Descarga y habilitación de istio
  + **Ingress**: Configuración de Ingress para tener acceso a la red de pods desde fuera de ella por medio de una gateaway.
  + **Kiali**: Despliegue de la aplicación de monitorización del tráfico entre los distintos pods levantados.
* **Microservices**
  + **Docker Images V1**: Construcción de las imágenes de la versión 1
  + **Docker Images V2**: Construcción de las imágenes de la versión 2
* **Deployments**
  + **Database**: Despliegue de los pods de las databases Mongo y MariaDB llamando a los ficheros correspondientes (en *kubernetes/deployments/* los ficheros *deploy\_mariadb.yml* y *deploy\_mongo.yml*).
  + **Services**: Creación de los servicios de los microservicios llamando a los ficheros correspondientes (carpeta *kubernetes/services*, todos los *.yml*)
  + **Deployments - V1**: Despliegue de los pods de las versiones V1 de los microservicios (todos los ficheros terminados en *.yml* dentro de *kubernetes/deployments/version\_1*).
  + **Deployments – V2**: Despliegue de los pods de las versiones V2 de los microservicios (todos los ficheros terminados en *.yml* dentro de *kubernetes/deployments/version\_2*).
    - [ ! ] Se deja un breve comentario debajo de esta sección que se utilizará en el vídeo explicativo. No entra dentro de ninguna seccióncomo tal.
* **Configurations**
  + **Gateaway & Virutal Services**: Levantamiento del gateaway (puerto 80) y de los posibles virtual services (frontend o router) dependiendo si queremos acceder a probarlo desde la página web o desde la API (ficheros encontrados en *kubernetes/* con el mismo nombre que lo que representan).
  + **Enroutings**: Comandos encargados de distribuir el tráfico entre las distintas versiones de los servicios variando el peso que tienen cada uno en la red (ficheros encontrados en *kubernetes/* con el mismo nombre que lo que representan).

Los ficheros yaml utilizados para la ejecución de las distintas tareas se pueden encontrar todos dentro de la carpeta *kubernetes/* como bien se indica en cada uno de los apartados. En resumen, para los de ficheros relacionados con las **bases de datos** se crea un único fichero que contenga las credenciales (*secret*), el *servicio*, el *deployment* y el *volumen de almacenamiento persistente*. Para cada uno de los **microservicios** se crea un fichero encargado de levantar el *service* y luego para cada una de las versiones un fichero para el *deployment*. Los demás ficheros que se pueden encontrar son los relacionados con la comunicación entre los microservicios o el acceso a ellos: primeramente se mencionan los utilizados para el **gateaway** que son el fichero que recibe este mismo nombre sumado a los ***virtual services*** del frontend y el router. Por último, se crea una serie de ficheros de **enrutamiento** encargados de balancear el tráfico hacia una u otra versión. Estos ficheros comienzan por la nomenclatura *enrouting*.

## Esquema de la estructura del proyecto

En definitiva, en la siguiente imagen se representa un esquema de lo que se ha implementado. Se recoge la información de los microservicios (imagen original, puerto, versiones, nombre) así como la relación entre ellos. Por último, se muestran los diferentes virtual services creados (representados con un punto rojo de borde negro), los cuales serán accesibles gracias al gateaway de ingress levantado en el puerto 80. Se puede visualizar lo que quedaría dentro y fuera de la propia estructura creada por kubernetes gracias a una línea tenue de color gris que engloba todo aquello que no sería accesible desde fuera, o bien que sí lo sería (virtual services).

Este es el esquema de la estructura del proyecto (sig. Página):

Diagrama

Descripción generada automáticamente

# Proceso y Resultado.

En las siguientes capturas se muestra el proceso de desarrollo de la aplicación, pasando del simple levantamiento de los servicios y el posterior despliegue, a la comunicación entre diferentes versiones, pasando por la creación de la comunicación simple entre ellos.

Estas son algunas capturas del proceso:

1. Comunicación del router con la base de datos de mariadb al hacer un get de bebidas. Cracks no había sido relacionado todavía, pero se había accedido por terminal para probar la comunicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Comunicación simple de la aplicación entera realizada. Acceso desde el virtual service del router:

Diagrama

Descripción generada automáticamente

1. Implementación de las versiones 2 y prueba de tráfico:

Diagrama

Descripción generada automáticamente

1. Fuerce de fallo 5xx (sin acceso, pero sin crashear el microservicio) en distintas etapas del proceso (Apagando servicios o sobresaturándolos). El resultado es que los microservicios siguen funcionando pero algunas flechas indican con color rojo que alguna trama ha sido denegada o no resuelta satisfactoriamente (se aprecia que en drinks se habría perdido, pero en cracks ha seguido funcionando bien, ósea que Router no se ha caido):

Diagrama

Descripción generada automáticamente con confianza baja

1. Prueba con el Frontend:

Diagrama

Descripción generada automáticamente

1. Prueba con la versión 2 del Frontend:

Diagrama

Descripción generada automáticamente

1. Diferencia entre el frontend V1 y V2 (color solamente):

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

1. Despliegue correcto de todas las versiones de todos los microservicios, incluido el frontend (se aprecia cómo el gateaway de ingress permite acceder tanto al frontend como al router):

Diagrama

Descripción generada automáticamente