



MÁSTER UNIVERSITARIO EN TECNOLOGÍAS WEB,
COMPUTACIÓN EN LA NUBE Y
APLICACIONES MÓVILES

ASIGNATURA:
COMPUTACIÓN EN LA NUBE

5 DE JUNIO DE 2023

TRABAJO FINAL

ANDONI SALCEDO NAVARRO

1. Introducción

Este es un proyecto conjunto para las materias de Computación en la Nube, Desarrollo Basado en Componentes Distribuidos y Servicios, y Persistencia Relacional y No-Relacional de Datos. En el curso de Persistencia, se ha requerido establecer la capa de persistencia, tanto relacional como no relacional, basándose en ciertas especificaciones. Para el curso de DBCDS, se necesita diseñar e implementar servicios utilizando Spring Boot a través de una API REST que facilite el acceso a los datos. En cuanto a la materia de Computación en la Nube, se solicita que el despliegue se lleve a cabo utilizando contenedores que se ejecuten en Kubernetes. El diagrama 1 (también incluido en la entrega para mejorar la legibilidad) se muestra la arquitectura de despliegue de la aplicación en Kubernetes.

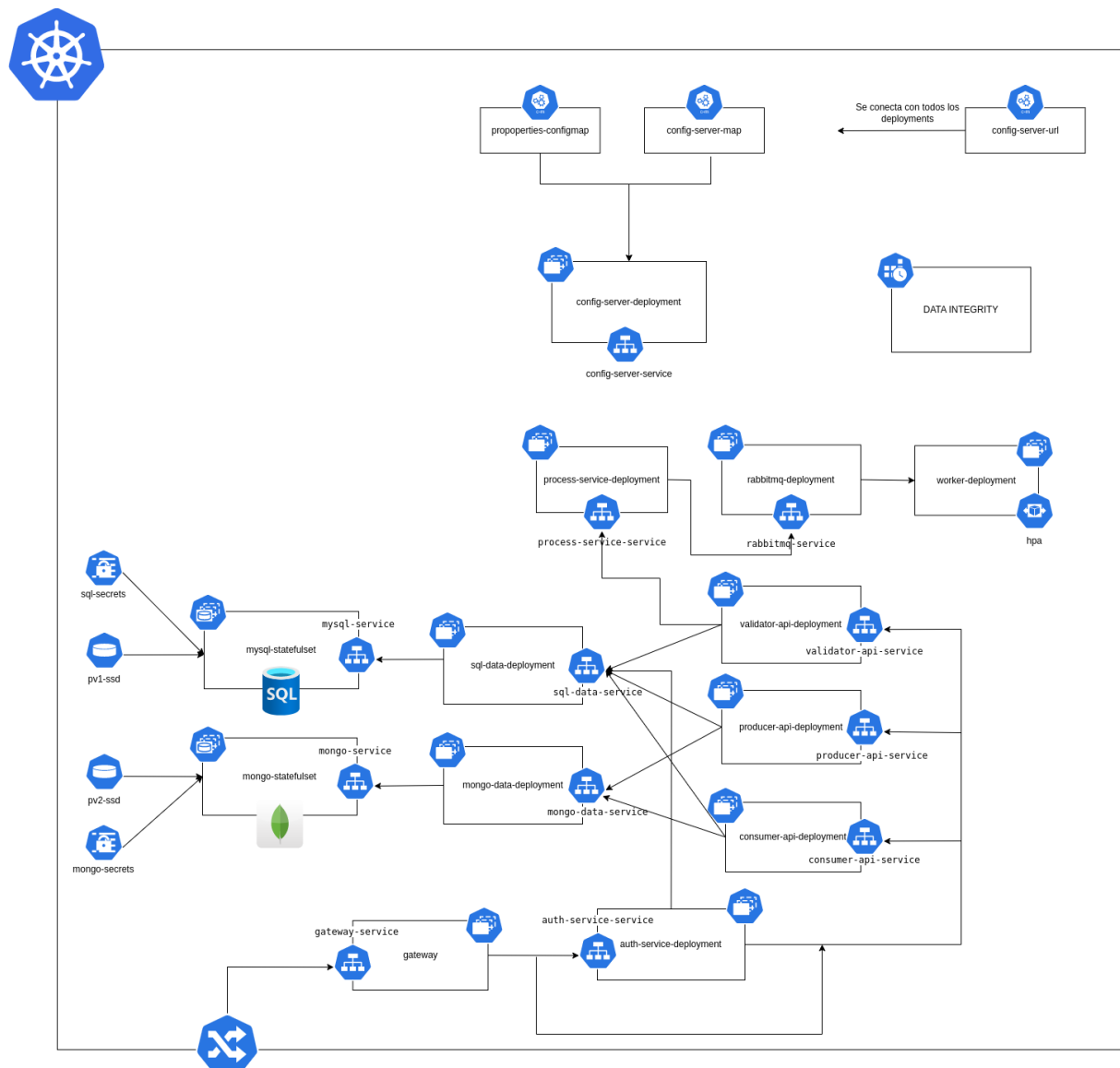


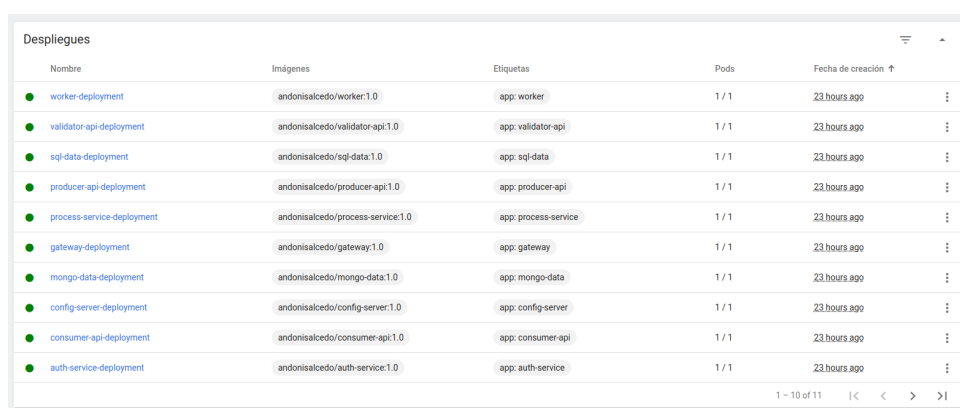
Figura 1: Diagrama de despliegue de la aplicación

En la entrega se expone que todos los recursos de yaml han de estar ubicados en el directorio **recursos**. Sin embargo, se ha decidido poner los recursos de los componentes dentro de cada componente de la aplicación, con la finalidad de automatizar el proceso de despliegue y construcción de imágenes. Para ello se ha utilizado el plug-in de maven que permite crear imágenes personalizadas para el proyecto dentro de maven. El resto de recursos se encuentran en la carpeta **RESOURCES/deploy/k8s**.

2. Recursos que se han utilizado

Se han utilizado los siguientes recursos para realizar el despliegue de la aplicación.

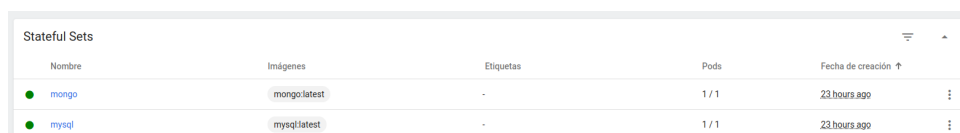
- Deployments: Es el recurso más utilizado para el despliegue se ha usado para la mayor parte de componentes, apis y servicios



Nombre	Imágenes	Etiquetas	Pods	Fecha de creación ↑
worker-deployment	andonisalcado/worker:1.0	app: worker	1 / 1	23 hours ago
validator-api-deployment	andonisalcado/validator-api:1.0	app: validator-api	1 / 1	23 hours ago
sql-data-deployment	andonisalcado/sql-data:1.0	app: sql-data	1 / 1	23 hours ago
producer-api-deployment	andonisalcado/producer-api:1.0	app: producer-api	1 / 1	23 hours ago
process-service-deployment	andonisalcado/process-service:1.0	app: process-service	1 / 1	23 hours ago
gateway-deployment	andonisalcado/gateway:1.0	app: gateway	1 / 1	23 hours ago
mongo-data-deployment	andonisalcado/mongo-data:1.0	app: mongo-data	1 / 1	23 hours ago
config-server-deployment	andonisalcado/config-server:1.0	app: config-server	1 / 1	23 hours ago
consumer-api-deployment	andonisalcado/consumer-api:1.0	app: consumer-api	1 / 1	23 hours ago
auth-service-deployment	andonisalcado/auth-service:1.0	app: auth-service	1 / 1	23 hours ago

Figura 2: Deployments

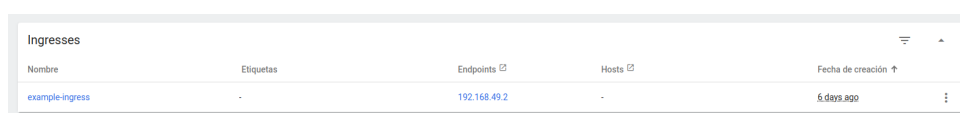
- StatefulSets: Se ha empleado para el despliegue de las bases de datos de mongodb y mysql.



Nombre	Imágenes	Etiquetas	Pods	Fecha de creación ↑
mongo	mongo:latest	-	1 / 1	23 hours ago
mysql	mysql:latest	-	1 / 1	23 hours ago

Figura 3: Statefulsets

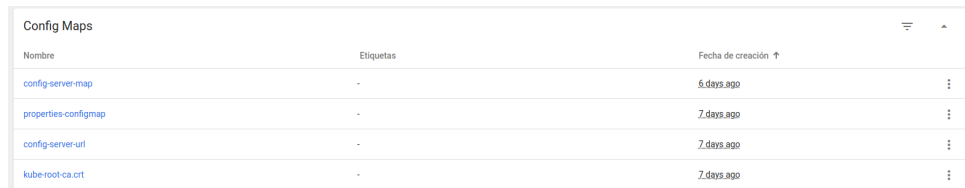
- Ingress: Se ha utilizado como proxy para redirigir peticiones de fuera del clúster a dentro del cluster.



Nombre	Etiquetas	Endpoints	Hosts	Fecha de creación ↑
example-ingress	-	192.168.49.2	-	6 days ago

Figura 4: Ingress

- **ConfigMaps:** Se ha utilizado para configurar la mayor parte de recursos de kubernetes. En primer lugar, se utiliza configmaps de tipo fichero para mostrarlo sobre el servidor de configuración este es el que guarda la configuración de todas las aplicaciones. Para ello, las aplicaciones han de tener la dirección de este servidor para cuando arranquen acceder a su configuración con lo que se ha creado otro configmap que define la url y las credenciales para acceder a este servidor. Por último, se ha creado otro configmap para configurar el propio servidor de configuración y configurarlo de tal manera que busque los ficheros de configuración en el directorio donde se ha montado los ficheros de configuración de todas las aplicaciones.



Nombre	Etiquetas	Fecha de creación ↑
config-server-map	-	6.days.ago
properties-configmap	-	7.days.ago
config-server-url	-	7.days.ago
kube-root-ca.crt	-	7.days.ago

Figura 5: ConfigMaps

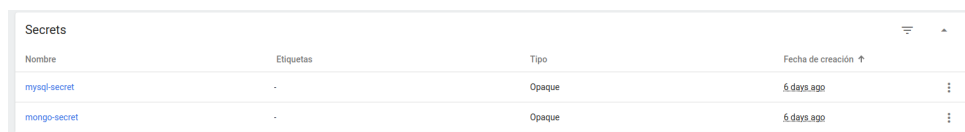
- **PersistentVolume:** Se han creado dos volúmenes persistentes de la clase *ssd*, sirven para persistir los datos de las bases de datos estos volúmenes serán asociados a las bases de datos en los recursos de StatefulSets.



Nombre	Capacidad	Modos de acceso	Política de reclamación	Estado	Petición	Clase de almacenamiento	Motivo	Fecha de creación ↑
pv1-ssd	storage: 2Gi	ReadWriteOnce	Retain	Bound	default/data-mongo-0	ssd	-	7.days.ago
pv2-ssd	storage: 2Gi	ReadWriteOnce	Retain	Bound	default/data-mysql-0	ssd	-	7.days.ago

Figura 6: PersistentVolumes

- **Secrets:** Se han configurado dos secrets con las credenciales de acceso de usuario para las bases de datos.



Nombre	Etiquetas	Tipo	Fecha de creación ↑
mysql-secret	-	Opaque	6.days.ago
mongo-secret	-	Opaque	6.days.ago

Figura 7: Secrets

- **HPA:** El Horizontal Pod Autoscaler se ha configurado para el componente worker por ello se ha limitado el uso de cpu de los workers para cuando este supere su capacidad cree un nuevo worker para manejar la carga.
- **Services:** Se han configurado la mayor parte de los servicios de kubernetes como servicios Headless, ya que no se requiere un acceso directo a los puertos desde fuera de kubernetes, pero si se necesita que el DNS configure su nombre para la resolución de la IP. Si se han configurado dos servicios de tipo nodeport con un puerto expuesto para las bases de datos, de esta manera se ha podido configurar la base de datos con herramientas como *mysql workbench* o *mongodb compass*.

Servicios						
Nombre	Etiquetas	Tipo	IP cluster	Endpoints Internos	Endpoints Externos	Fecha de creación ↑
worker-service	-	ClusterIP	10.107.197.74	worker-service:8080 TCP worker-service:0 TCP	-	23 hours ago
validator-api-service	-	ClusterIP	10.105.241.248	validator-api-service:8080 TCP validator-api-service:0 TCP	-	23 hours ago
sql-data-service	-	NodePort	10.96.126.64	sql-data-service:8080 TCP sql-data-service:31819 TCP	-	23 hours ago
producer-api-service	-	ClusterIP	10.99.2.168	producer-api-service:8080 TCP producer-api-service:0 TCP	-	23 hours ago
process-service-service	-	ClusterIP	10.111.62.123	process-service-service:8080 TCP process-service-service:0 TCP	-	23 hours ago
gateway-service	-	ClusterIP	10.109.247.217	gateway-service:8080 TCP gateway-service:0 TCP	-	23 hours ago
mongo-data-service	-	ClusterIP	10.96.30.123	mongo-data-service:8080 TCP mongo-data-service:0 TCP	-	23 hours ago
consumer-api-service	-	ClusterIP	10.101.65.214	consumer-api-service:8080 TCP consumer-api-service:0 TCP	-	23 hours ago
config-server-service	-	ClusterIP	10.100.208.226	config-server-service:8080 TCP config-server-service:0 TCP	-	23 hours ago
mongo-service	-	NodePort	10.110.248.119	mongo-service:27017 TCP mongo-service:52623 TCP	-	23 hours ago

Figura 8: Services

- CronJob: Como forma opcional se ha creado una aplicación es springboot que gestiona la consistencia de los datos entre ambas bases de datos, esta aplicación se debe ejecutar periódicamente consultando ambas bases de datos para comprobar que las entidades de una y los documentos de la otra coinciden, por ello se ha utilizado un cronjob que ejecuta cada 12 horas esta comprobación.

Cron Jobs							
Nombre	Imágenes	Etiquetas	Planificar	Suspender	Activo	Última ejecución	Fecha de creación ↑
data-integrity-cronjob	andonisalcedo/data-integrity:1.0	app: data-integrity	0 */12 *	false	0	-	9 seconds ago

Figura 9: CronJob

Adicionalmente, se han configurado InitContainers, dado que la mayoría de recursos dependen de que el servidor de configuración este activo se ha decidido comprobar que este esta activo antes de arrancar la aplicación por ello se ha configurado todos los deployments de la siguiente forma:

```
initContainers:
- name: health-check
  image: curlimages/curl
  command: ["sh", "-c", "until curl -s -o /dev/null -w '%{http_code}']
  ↪ ${SPRING_CLOUD_CONFIG_URI}/actuator/health | grep -q 200; do sleep 1; done"]
  env:
  - name: SPRING_CLOUD_CONFIG_URI
    valueFrom:
      configMapKeyRef:
        name: config-server-url
        key: SPRING_CLOUD_CONFIG_URI
```

Al iniciarse los componentes ocurriría lo siguiente:

NAME	READY	STATUS	RESTARTS	AGE
pod/auth-service-deployment-7785db5478-dtv55	0/1	Init:0/1	0	12s
pod/config-server-deployment-5f8c57b964-z9t65	1/1	Running	0	11s
pod/consumer-api-deployment-574cd54c65-99thz	0/1	Init:0/1	0	11s
pod/gateway-deployment-5788f4768d-lcmzf	0/1	Init:0/1	0	10s
pod/mongo-0	1/1	Running	0	12s
pod/mongo-data-deployment-5bdf5b4c5-6mgb7	0/1	Init:0/1	0	9s
pod/mysql-0	0/1	Terminating	0	67s
pod/process-service-deployment-6c44b9bfc7-8jtrw	0/1	Init:0/1	0	8s
pod/producer-api-deployment-6f5c786c-pps5f	0/1	Init:0/1	0	7s
pod/rabbitmq-deployment-6bcbf4ff47-tk9kx	1/1	Running	0	12s
pod/sql-data-deployment-68bfcf78dc-znlzm	0/1	Init:0/1	0	6s
pod/validator-api-deployment-5c44475f8f-rgh2k	0/1	Init:0/1	0	6s
pod/worker-deployment-5bd88dc995-rzjjq	0/1	Init:0/1	0	5s

Figura 10: InitContainer ejemplos

3. Pruebas

Para mostrar el funcionamiento del *Horizontal Pod Autoescaler* se va a someter al sistema a una alta carga de trabajo, para que la CPU de los workers límite y se tengan que crear más pods para gestionar las peticiones.

En una situación inicial el sistema se encontraría de la siguiente manera:

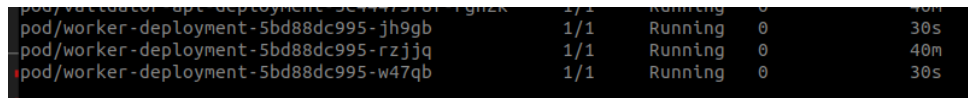
pod/sql-data-deployment-68bfcf78dc-znlzm	1/1	Running	0	55s
pod/validator-api-deployment-5c44475f8f-rgh2k	1/1	Running	0	71m
pod/worker-deployment-5bd88dc995-rzjjq	1/1	Running	0	71m

Figura 11: tras la ejecución de *kubectl get all*

validator-api-deployment-5c44475f8f-rgh2k	3m	153Mi
worker-deployment-5bd88dc995-rzjjq	2m	108Mi

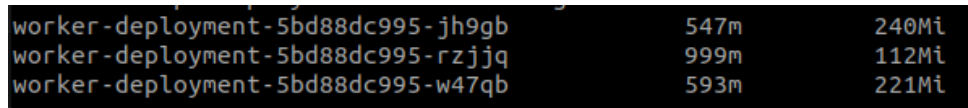
Figura 12: tras la ejecución de *kubectl top pod*

Cuando se somete al sistema a altas cargas de trabajo el sistema se encontraría de la siguiente manera:



pod/worker-deployment-5bd88dc995-jh9gb	1/1	Running	0	30s
pod/worker-deployment-5bd88dc995-rzjjq	1/1	Running	0	40m
pod/worker-deployment-5bd88dc995-w47qb	1/1	Running	0	30s

Figura 13: tras la ejecución de *kubectl get all*



worker-deployment-5bd88dc995-jh9gb	547m	240Mi
worker-deployment-5bd88dc995-rzjjq	999m	112Mi
worker-deployment-5bd88dc995-w47qb	593m	221Mi

Figura 14: tras la ejecución de *kubectl top pod*

De esta manera se muestra como los workers son capaces de adaptarse a la carga de trabajo si son sometidos a altas cargas de trabajo.