

Esta clase va a ser

- grabada

Clase 19. PROGRAMACIÓN BACKEND

Orquestación de contenedores & Práctica integradora

Temario

18 – Parte II

Clusters & escalabilidad

- ✓ [Módulo Cluster](#)
- ✓ [Docker](#)
- ✓ [Docker como PM](#)

19 – Parte I

Orquestación de contenedores

- ✓ [DockerHub](#)
- ✓ [Orquestación de contenedores](#)
- ✓ [Orquestación con Kubernetes](#)

19 – Parte II

Práctica integradora

- ✓ Skills
- ✓ Práctica integradora

Objetivos de la clase – Parte I

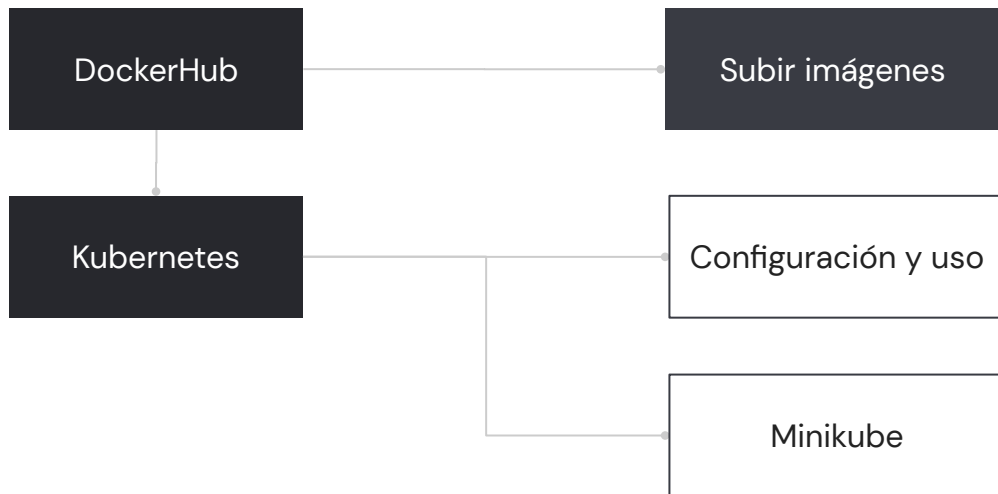
- Subir imágenes de servidores a DockerHub
- Conocer Kubernetes
- Realizar un deploy de Kubernetes local con Minikube

Glosario

Artillery: Es un toolkit de performance que prueba nuestro servidor y corrobora su fiabilidad en un entorno real.

Docker: es una plataforma que permitirá crear, probar e implementar aplicativos en unidades de software estandarizadas llamadas **contenedores**.

MAPA DE CONCEPTOS



Toda la clase es un proceso

DockerHub

¿Qué es DockerHub?

Dockerhub es una librería, o repositorio de imágenes en la nube.

Cuando hemos finalizado con el mantenimiento de nuestro aplicativo, necesitamos compartir la imagen resultante con nuestro equipo (Y enviar la imagen por correo electrónico no resulta óptimo).

Al subir nuestra imagen en la nube, todos los miembros autorizados podrán descargarla y utilizarla.



¿Por qué debería tener mis imágenes en la nube?

Existen múltiples razones para tener nuestra imagen en un repositorio:

- Podemos compartir nuestra imagen con nuestro equipo de desarrollo.
- Tenemos un sistema mejor controlado de nuestra imagen, ya que cada cambio de ésta puede significar una nueva tag con otra versión.
- Permite que otros softwares descarguen la imagen

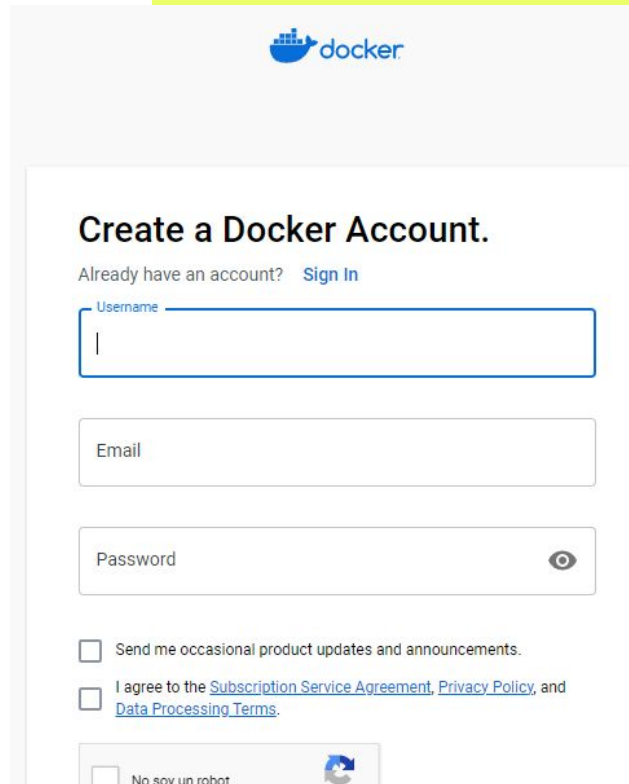
La razón de que éste se encuentre en la nube y permita que otros softwares lo descarguen, significa que al momento de configurar procesos más complejos (como deployment), podemos obtener la imagen directamente, sin necesidad de tener que estar cambiando el archivo local.


¡Disponible en todo momento y fácilmente actualizable!

Obtengamos nuestro Docker ID

Para comenzar a tener nuestras imágenes en nuestro repositorio, vamos a crear primero una cuenta.

Es importante recordar que el username que elijamos, se convertirá en nuestro Docker ID (lowercase)

A screenshot of the Docker website's account creation page. At the top right is the Docker logo. The main heading is "Create a Docker Account." Below it is a link "Already have an account? Sign In". There are three input fields: "Username" (with a blue border and a cursor), "Email", and "Password" (with an eye icon for toggling visibility). Below the fields are two checkboxes: "Send me occasional product updates and announcements." and "I agree to the Subscription Service Agreement, Privacy Policy, and Data Processing Terms." At the bottom is a CAPTCHA area with a checkbox labeled "No soy un robot" and a small robot icon.


 docker

Create a Docker Account.

Already have an account? [Sign In](#)


Username

Email

Password 

☐ Send me occasional product updates and announcements.

☐ I agree to the [Subscription Service Agreement](#), [Privacy Policy](#), and [Data Processing Terms](#).

☐ No soy un robot 

Panel principal

Una vez que confirmemos nuestra cuenta a partir del correo que recibimos, podemos entrar al panel principal de dockerhub

A partir de aquí, ya tenemos una cuenta activa para poder subir una imagen.

A pesar de que ya aprendimos el proceso para subir una imagen, nos falta aplicarlo a un proyecto más sólido, donde podamos aprovechar el potencial de la contenerización.

Subir una imagen a DockerHub



Comencemos con un proyecto para subir.

Con el fin de no afectar al flujo de desarrollo de un proyecto completo, el profesor cuenta con un proyecto que cuenta con las siguientes características

- ✓ Una vista para crear un usuario
- ✓ La creación de usuarios se hace vía mongoose, por lo cual se utiliza una variable de entorno MONGO_URL
- ✓ Además, se tiene una ruta test para generar usuarios (La cual se utilizará más adelante)

Se creará una imagen de docker para poder subir a DockerHub



Creando Dockerfile

Crearemos un Dockerfile con la configuración necesaria para tener una imagen ejecutable de nuestro proyecto.

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, showing the file structure of a project. The 'OPEN EDITORS' section shows a 'Dockerfile' file. The 'PLANTILLAPROYECTODOCKERIZAC...' section shows a 'src' directory, a '.gitignore' file, and a 'Dockerfile' file. The 'package-lock.json' and 'package.json' files are also visible. On the right, the Dockerfile editor is open, showing the following content:

```
1 FROM node
2 COPY . .
3 RUN npm install
4 EXPOSE 8080
5 CMD ["node", "src/app.js"]
```



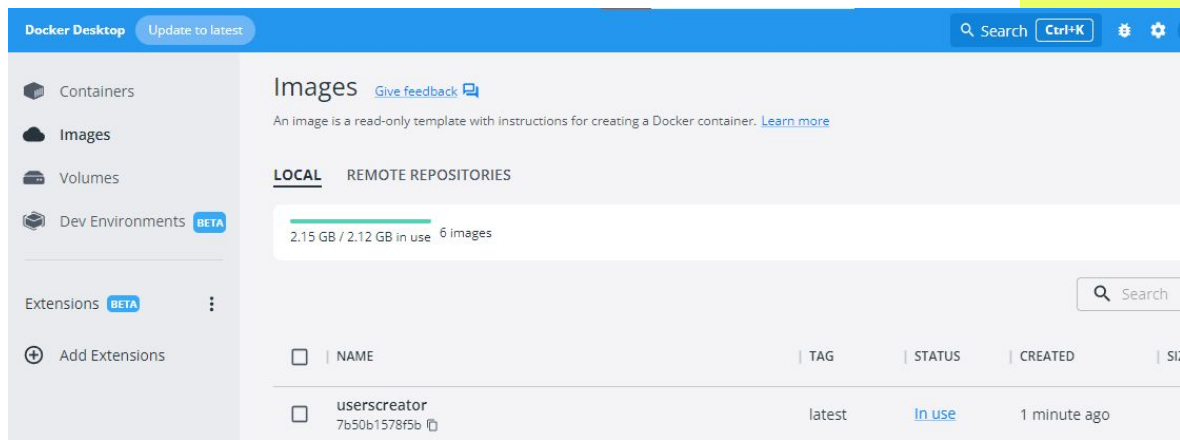
APROXIMACIÓN AL PROCESO

Creando imagen

Ejecutamos el build de Docker y colocamos el nombre que deseemos (**no olvides el punto al final**)

```
docker build -t userscreator .
```

Deberá aparecer en nuestro dockerDesktop







Levantemos un contenedor para probar la imagen

Vamos a ejecutar la imagen indicando el puerto donde deseamos conectar el puerto expuesto del contenedor.

Además, nota cómo colocamos la url de mongo para que el contenedor cuente con la variable necesaria para poder crear a los usuarios.

Debemos corroborar que el contenedor esté corriendo correctamente.

 **Run a new container**
userscreator:latest

Optional settings 

Container name

A random name is generated if you do not provide one.

Ports

Host port

80

Container port

8080/tcp

Volumes

Host path

...

Container path

+

Environment variables

Variable

MONGO_URL

Value

AQUÍ EL VALOR DE LA URL

+

Cancel

Run



busy_gates

982305a2806a 

[userscreator:latest](#)

Running

[80:8080](#) 

26 seconds ago





Preparamos la imagen para su subida: Login

Lo primero es conectar nuestro CLI a la cuenta de DockerHub que recién creamos. Para ello, llamaremos al comando

```
docker login
```

Se nos solicitará el username (lowercase) y contraseña para poder acceder. Para corroborar que todo esté en orden, debemos llegar al siguiente mensaje:

```
Login Succeeded
```

```
Logging in with your password grants your terminal complete access to your account.
```



Preparamos la imagen para su subida: tagname

La estructura para cargar la imagen en dockerhub será:

<username>/imagen:version

Entonces, si nuestra imagen actualmente tiene el nombre **userscreator**. Bien podemos cambiar el nombre de la tag a partir de:

```
docker tag userscreator <username>/userscreator:1.0.0
```

Podremos ver la nueva imagen generada con el nombre indicado.

```
docker push <username>/userscreator:1.0.0
```




Finalmente, subimos la imagen a DockerHub

Muy similar a Github, para poder subir nuestra imagen ejecutaremos:

```
docker push <username>/userscreator:1.0.0
```

Ahora, en nuestro DockerHub, debemos poder acceder a nuestros repositorios y visualizar la imagen cargada. éste contará con la sintaxis **<username>/imagen** previamente mencionada.

 / userscreator
Contains: Image | Last pushed: a minute ago

 Not Scanned

 0

 0

 Public

Orquestación de contenedores

Ya ganamos una batalla

Logramos ganarle a la sobrecarga de performance con la clusterización de Node. Además, conseguimos hacer nuestra primera virtualización de hardware, creando contenedores e incluso subiéndolos a la nube.

Sin embargo, queda mucho por resolver. Una de estas cosas es: si cada contenedor al final es un aplicativo corriendo de manera aislada, ¿cómo haremos para que todos los contenedores hagan armonía y se apoyen en las consultas de un único punto?

El entorno de producción es una guerra... ¡En verdad!

Cada vez nos hacemos más a la idea de que nuestro servidor no funciona "tan idealmente" como quisiéramos, cuando se trata de atender a un número realmente elevado de usuarios, con algunas peticiones de un nivel más complejo.

Es por ello que, si tenemos un elevado número de usuarios que consumen nuestros servicios, necesitamos responder con un número elevado de instancias para que no nos afecte en el rendimiento general.

¡Divide y conquistarás, hora de trabajar múltiples instancias de nuestra imagen, hora de ejecutar multi-contenedores!



PARA RECORDAR

Clusterización

Recuerda que con node podemos generar múltiples workers dirigidos por un Primary process. **Esta fue la primera alternativa de división de tareas que habíamos implementado.**

Lógica de clusterización para contenedores

Cuando hablamos de contenedores, muy seguramente escucharás acerca del término **Orquestación**. La orquestación es un término muy similar a la clusterización (a veces tomados por iguales), sin embargo, en esta clase haremos una ligera distinción.

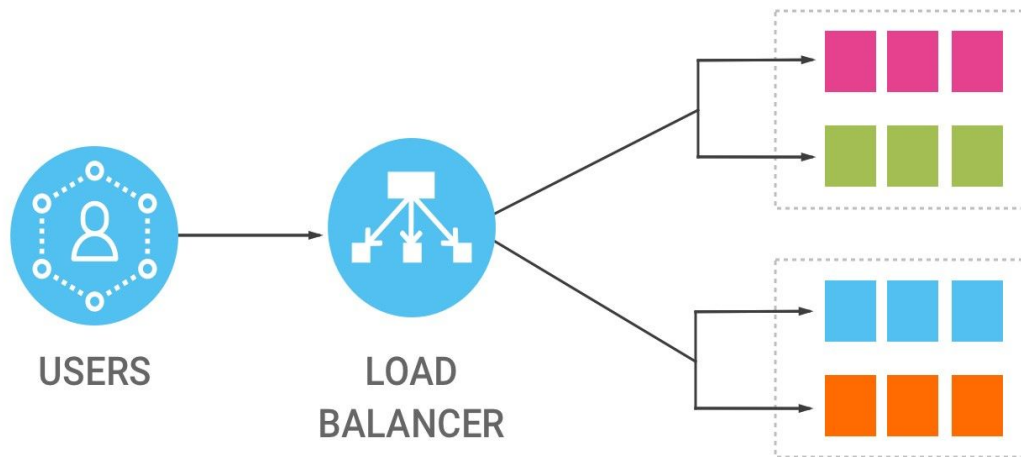
Al realizar una clusterización, estamos instanciando un modelo Primary-worker, en la cual los procesos workers pueden reiniciarse una vez caídos. ¿Y la orquestación? también se trata de tener un proceso principal, el cual se encargue de cargar con workers, sólo que ésta vez cada worker será en esencia un contenedor por sí solo.

¿En qué son diferentes?

La diferencia radica en que la orquestación es un proceso más profundo, ya que no sólo se trata de hacer la división, sino que permite realizar la gestión de éstos de manera más controlada, permitiendo::

- ✓ Control de reinicio para contenedores que presenten algún fallo
- ✓ **Balanceador de carga**, el cual permitirá que las peticiones puedan llegar a los contenedores de manera más distribuida, a diferencia de en cola (como lo hace la clusterización base).
- ✓ Manejo de rollout updates, los cuales nos permiten seguir un flujo organizado para poder actualizar o reconfigurar contenedores en un cluster

Balanceador de cargas





Break

¡10 minutos y volvemos!

Orquestación con kubernetes

¿Qué es Kubernetes?

Kubernetes es una tecnología de orquestamiento de contenedores. En esencia, se trata de una plataforma que sirve para administrar cargas de trabajos y servicios.

Kubernetes tomará un conjunto de instrucciones y las ejecutará para poder distribuir **pods**, donde cada pod puede tener **n** contenedores, de esta forma **todos los contenedores que pertenezcan a un pod, podrán funcionar como una entidad única para comunicarse.**

¿Cómo comenzamos a utilizar Kubernetes?

Lo primero será comenzar a utilizar **kubectl**. El cliente de Kubernetes nos permitirá ejecutar comandos en nuestro CLI, de manera que lo necesitaremos instalado en nuestra computadora con Windows

Para poder instalar los binarios de kubectl directamente, ¿recuerdas que recientemente se te solicitó instalar curl?

Puedes hacer la instalación directamente corriendo este comando desde tu cmd:

```
curl.exe -LO "https://dl.k8s.io/release/v1.25.0/bin/windows/amd64/kubectl.exe"
```

Esto instalará directamente kubectl. Para poder probarlo, corre el comando **kubectl version --short**

Para instalar en Linux, puedes revisar [aquí](#). Para Mac, puedes revisar [aquí](#)

```
Client Version: v1.25.0  
Kustomize Version: v4.5.7  
Server Version: v1.25.2
```

CODERHOUSE

¿Cómo obtener Kubernetes?

Kubernetes está pensado para poder ser desplegado, eso quiere decir que podemos hacerlo desde alguna plataforma en la nube, aunque en este caso, por cuestión de costos, lo utilizaremos de manera local.

Hay múltiples posibilidades para poder correr un repositorio local:

- ✓ Minikube
- ✓ MicroK8s
- ✓ k3s

En el desarrollo de esta clase, utilizaremos Minikube



minikube

Minikube

Minikube es un software que permite levantar un cluster local de kubernetes, permitiendo hacer las pruebas que necesitemos dentro de nuestro contenedor de docker.

Funciona para cualquier sistema operativo y cuenta con un instalador directo.

- Para poder ejecutar el instalador, bastará con descargarlo de [este link](#)
No hace falta gran configuración, todo se instalará directamente en la computadora.
- Para comenzar a trabajar con Minikube, bastará con correr el comando ***minikube start***
- **Recuerda haber instalado previamente kubectl**, ya que al momento de iniciar minikube, buscará si está instalado, de otra forma lo tendremos que configurar manual

```
Complementos habilitados: storage-provisioner, default-storageclass, dashboard
kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Corroborando las instalaciones






Una vez teniendo **kubectl** y **minikube**, vamos a poder correr el comando

```
kubectl cluster-info
```

Podremos visualizar cómo tenemos Kubernetes corriendo en un puerto aleatorio:

```
Kubernetes control plane is running at https://127.0.0.1:54881
```

Además, en nuestro docker desktop, podemos ver cómo contamos con la imagen de minikube:

	gcr.io/k8s-minikube/kicbase 866c1fe4e3f2 	v0.0.36	In use	29 days ago	1.11 GB			
---	---	---------	------------------------	-------------	---------	---	---	---

Por último, ya deberíamos tener un contenedor de minikube corriendo

	minikube a9d31c06ee73 	gcr.io/k8s-minikube/kicbase:v0.0.36	Running	57824:22 
				57825:2376 
				57826:32443 
				57827:5000 
				57828:8443 

Elementos importantes antes de comenzar

Comandos utilizados comúnmente por kubectl

- ✓ **gets:** Son comandos utilizados para obtener información acerca de los recursos de kubernetes. Los gets más comunes realizados son los **kubectl get pods**, **kubectl get deployments** y **kubectl get services**
- ✓ **apply:** Nos servirá para poder crear nuestro primer recurso. El modelo de creación es declarativo, de manera que necesitaremos primero tener un conjunto de instrucciones que configuren el **pod** a crearse, además de los contenedores con los que se generará.
- ✓ **edit:** Servirá para cambiar la configuración de alguno de nuestros pods.

Comandos utilizados comúnmente por kubectl

- ✓ **delete:** Sirve para eliminar recursos, nuevamente, podemos realizar el borrado de pods y deployments completos.
- ✓ **config:** Sirve para comandos generales de configuración, por ejemplo, al activar kubernetes en nuestro docker desktop, reconocerá el contexto de nuestro kubectl automáticamente en minikube:

```
>kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	docker-desktop	docker-desktop	docker-desktop	
	minikube	minikube	minikube	default

Principales conceptos de Kubernetes

- ✓ **pod:** Unidad mínima de ejecución de kubernetes, éste consiste en un espacio donde podemos alojar nuestros contenedores. Podemos relacionarlos directamente con los **nodos** de un cluster. Si ejecutamos **kubectl get pods** obtenemos:

```
No resources found in default namespace.
```

- ✓ **replicas:** Refiere al número de pods a generarse, donde cada pod podrá contener una o n instancias de un contenedor.

Principales conceptos de Kubernetes

- ✓ **pod template:** Si sabemos que un pod contendrá uno o más contenedores, entonces necesitamos una plantilla para poder generar dichos contenedores, un template contendrá información sobre la replicación de los contenedores que deberá generar una vez aplicada la configuración.
- ✓ **service:** Un recurso de servicio hace que los Pods sean accesibles para otros Pods o usuarios fuera del cluster. Sin un servicio, es imposible acceder a un pod. Gracias a éste, se podrá recibir un request y se redigirá a los pods a partir de un balanceador de carga.

Principales conceptos de Kubernetes

- ✓ **deployment:** Es un creador y actualizador de **pods** y sets. A partir de un **manifiesto**, éste sabrá cómo crear el pod, qué contenedores colocar dentro de él, cuántas réplicas obtener de un contenedor
- ✓ **manifest:** El manifiesto es el archivo json o yml que cuenta con todas las instrucciones para realizar el deployment. Escribir correctamente un manifiesto llevará a un deployment exitoso y desembocará en la creación de nuestro primer pod.

Haciendo nuestro primer deploy en kubernetes

Ya tenemos el cluster ejecutándose

Recuerda que podemos visualizarlo con el comando *kubectl cluster-info*.

Ahora, para poder levantar nuestros pods de kubernetes primero crearemos un **archivo de deployment**.

Éste contendrá toda la configuración de los pods a generar y los respectivos contenedores que tendrán dentro.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

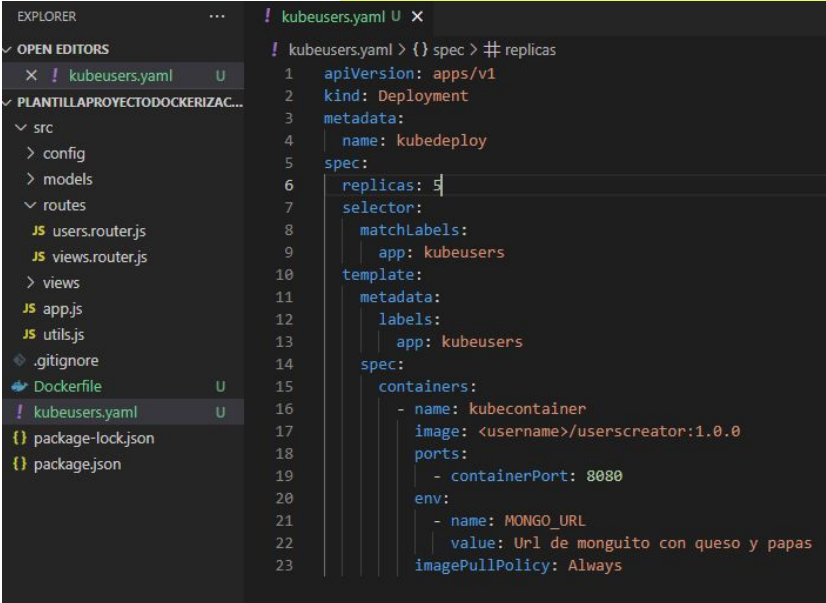
Ejemplo de un deploy de Nginx

CODERHOUSE

kubeUsers.yaml

Escribiremos entonces la configuración del cluster en un archivo con el nombre kubeusers (El nombre del app y labels debe ser en minúsculas), el cual tendrá el creador de usuarios que hemos trabajado.

Sin embargo, este archivo parece ligeramente más complejo que los que hemos visto, así que profundizaremos en las secciones



```
! kubeusers.yaml U X
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: kubedeploy
5  spec:
6    replicas: 5
7    selector:
8      matchLabels:
9        app: kubeusers
10   template:
11     metadata:
12       labels:
13         app: kubeusers
14     spec:
15       containers:
16         - name: kubecontainer
17           image: <username>/userscreator:1.0.0
18           ports:
19             - containerPort: 8080
20           env:
21             - name: MONGO_URL
22               value: Url de monguito con queso y papas
23           imagePullPolicy: Always
```

Entendiendo un archivo de deploy: Base

- ✓ apiVersion: la versión del recurso que estamos trabajando:
- ✓ kind: El tipo de recurso, aquí hacemos saber que el recurso que estamos configurando es el deploy del aplicativo.
- ✓ metadata.name : El nombre con el cual será reconocida esta aplicación, este nombre aparecerá en cada pod como prefijo del pod

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeusers
```

Entendiendo un archivo de deploy: spec

- ✓ spec: describe las especificaciones de lo que queremos que se implemente en este recurso.
- ✓ replicas: el número de pods que se generarán en el cluster de kubernetes.
- ✓ selector.matchLabels: hace referencia a que seleccione las instancias del template que cuenten con la label indicada. Podemos entonces leer: Genera 5 réplicas del template que cumpla con esta label.
- ✓ template: es la “plantilla” de nuestro pod, de manera que aquí describiremos los detalles de lo que contendrá dicho pod

```
spec:  
  replicas: 5  
  selector:  
    matchLabels:  
      app: kubeusers  
  template:  
    metadata:  
      labels:  
        app: kubeusers
```

Entendiendo un archivo de deploy: template (1)

- ✓ spec (nuevamente): refiere a las especificaciones, pero esta vez hace referencia a las especificaciones de nuestro pod
- ✓ containers: Aquí definimos la característica del (o los contenedores) que vamos a meter en el pod
- ✓ name: Nombre del contenedor
- ✓ image: Imagen sobre la cual se basará el contenedor.
- ✓ ports.containerPort: hace referencia al puerto que el contenedor tenga expuesto

```
template:
  metadata:
    labels:
      app: kubeusers
  spec:
    containers:
      - name: kubecontainer
        image: <username>/userscreator:1.0.0
        ports:
          - containerPort: 8080
        env:
          - name: MONGO_URL
            value: Url de monguito con queso y papas
        imagePullPolicy: Always
```

Entendiendo un archivo de deploy: template (2)

- ✓ env: variables de entorno, en este caso sólo tenemos configurada la variable de entorno MONGO_URL para nuestro proyecto.
- ✓ imagePullPolicy: sirve para determinar en qué momento debería obtener la imagen indicada de internet o del entorno local
 - IfNoPresent: Sólo hará pull de DockerHub si no está la imagen ya en el entorno local.
 - Always: Siempre pullear la imagen de DockerHub, independientemente de que se encuentre local.
 - Never: Nunca obtener de DockerHub, fuerza a que la imagen ya esté localmente (peligroso para minikube)

```
template:
  metadata:
    labels:
      app: kubeusers
  spec:
    containers:
      - name: kubecontainer
        image: <username>/userscreator:1.0.0
        ports:
          - containerPort: 8080
        env:
          - name: MONGO_URL
            value: Url de monguito con queso y papas
        imagePullPolicy: Always
```

El proceso aún no termina

```
! kubeusers.yaml U x
! kubeusers.yaml > {} spec > {} template >
1  ∨ apiVersion: v1
2    kind: Service
3  ∨ metadata:
4    | name: kubeshervice
5  ∨ spec:
6    ∨ selector:
7      | app: kubeusers
8    ∨ ports:
9      ∨ - port: 80
10       | targetPort: 8080
11       type: LoadBalancer
12    ---
13  ∨ apiVersion: apps/v1
14    kind: Deployment
15  ∨ metadata:
16    | name: kubedeploy
17  ∨ spec:
18    replicas: 5
19  ∨ selector:
20    ∨ matchLabels:
21      | app: kubeusers
```

Recuerda las definiciones importantes de kubernetes, comentamos que un servicio es aquel que realmente permitirá conectar con los pods de un cluster. Sin un servicio, nuestros pods quedarán ocultos en el cluster y nunca podremos realmente conectar un request con ellos.

El proceso aún no termina

```
! kubeusers.yaml U x
! kubeusers.yaml > {} spec > {} template >
1  ∨ apiVersion: v1
2    kind: Service
3  ∨ metadata:
4    | name: kubeshervice
5  ∨ spec:
6    ∨ selector:
7      | app: kubeusers
8    ∨ ports:
9      ∨ - port: 80
10       | targetPort: 8080
11       type: LoadBalancer
12  ---
13  ∨ apiVersion: apps/v1
14    kind: Deployment
15  ∨ metadata:
16    | name: kubedeploy
17  ∨ spec:
18    replicas: 5
19  ∨ selector:
20    ∨ matchLabels:
21      | app: kubeusers
```

Ahora escribiremos en el mismo archivo, el bloque superior indicado en la captura (nota cómo lo separamos del recurso de deploy).

Nota que el nuevo recurso es de tipo **Service**, además tiene el nombre **kubeshervice**, y selecciona todos los pods kubeusers.

Por último, el tipo **LoadBalancer**, hace referencia a la distribución de requests entre los diferentes pods

¡Ha llegado el momento!

Toda nuestra configuración ahora tendrá sus frutos: Es hora de aplicar nuestra configuración para tener nuestro servidor en el cluster de kubernetes local de Minikube.

Para ello, colocaremos el comando:

```
kubectl apply -f kubeusers.yaml
```

Éste nos permitirá ejecutar el archivo yaml que estuvimos configurando. En cuanto lo ejecutemos, deberíamos poder ver:

```
service/kubeservice created  
deployment.apps/kubedeploy created
```

¡Vamos a revisar qué pasó!

Vamos a ejecutar un conjunto de comandos que permitirán revisar la integridad de lo que acabamos de deployar. A simple vista parece que no se ha ejecutado nada, sin embargo, hay que entender lo que se ejecutó

Primero está el ***kubectl get deployments***. El cual me ayudará a corroborar si el deploy se realizó satisfactoriamente:

```
kubectl get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
kubedeploy	5/5	5	5	90s

¡Ahí está el nombre de nuestro deploy! además, indica que tenemos 5 pods “READY”. ¡Hay que ver dónde se localizan!

Visualizar pods

El siguiente comando será ***kubectl get pods***

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
kubedeploy-78f8d65d6b-6p84w	1/1	Running	0	3m17s
kubedeploy-78f8d65d6b-7hzl7	1/1	Running	0	3m17s
kubedeploy-78f8d65d6b-7js9c	1/1	Running	0	3m17s
kubedeploy-78f8d65d6b-kzd48	1/1	Running	0	3m17s
kubedeploy-78f8d65d6b-nkrgrn	1/1	Running	0	3m17s

Podemos visualizar las 5 réplicas (pods) que se encuentran ahora ejecutándose (STATUS => Running).

Esto significa que estos 5 pods están corriendo un contenedor de Docker cada uno, lo cual me está permitiendo tener una fuerza mayor de procesamiento, además de estar controladas por un cluster y son tratados como una unidad.

Visualizar servicio (1)

El tercer comando a correr es ***kubectl get services***. Éste nos devolverá el servicio que debimos haber levantado al hacer apply.

```
kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	7d22h
kubeservice	LoadBalancer	10.97.148.45	<pending>	80:31599/TCP	6m21s

Ahí aparece el buen **kubeservice**, que es el que configuramos en nuestro .yaml

Recuerda que este servicio es el que permite conectar con los pods del cluster. Echa un vistazo a la columna PORT(s). ¿Notas cómo está haciendo el vínculo?

Significa que, el proceso que nosotros vinculamos a un puerto 80, está siendo procesado de manera interna en un puerto de escucha 31599.

Visualizar servicio (2)

Ahora, recordemos que al final nuestro cluster local al final vive en minikube, así que es momento de ponerlo a trabajar.

Primero, ejecutaremos ***minikube service list***

```
minikube service list
```

NAMESPACE	NAME	TARGET PORT	URL
default	kubernetes	No node port	
default	kubeservice	80	http://192.168.49.2:31599
kube-system	kube-dns	No node port	
kubernetes-dashboard	dashboard-metrics-scraper	No node port	
kubernetes-dashboard	kubernetes-dashboard	No node port	

Uno de esos servicios debe tener el nombre del servicio que levantamos al aplicar el archivo de deploy. Ahora, para que pueda salir a la luz en una url de localhost, ejecutaremos ***minikube service <nombre del servicio>***

Luego minikube buscará en su tabla el servicio solicitado y lo ejecutará para poder utilizarlo.

Ejecutando el servicio con minikube

```
minikube service kubeservice
```

NAMESPACE	NAME	TARGET PORT	URL
default	kubeservice	80	http://192.168.49.2:31599
🏃 Starting tunnel for service kubeservice.			
NAMESPACE	NAME	TARGET PORT	URL
default	kubeservice		http://127.0.0.1:52786



🚫 127.0.0.1:52786

¡Bienvenido!, crea algunos usuarios

En conclusión...

La orquestación de contenedores y el acercamiento de configuraciones de nivel más complejo como son docker, kubectl, minikube y kubernetes, puede ser intimidante al principio. Sin embargo, es un stack de herramientas sumamente útil para que tengamos el primer acercamiento de una clusterización del servidor que hemos trabajado.

Estos procesos los irás aprendiendo sobre la marcha. ¡Ánimo y a profesionalizarlo a tu ritmo!

¡Atención!

La próxima clase será una **Práctica Integradora**.
Te recomendamos prepararte con la guía de temas disponible
en la carpeta de tu comisión



¿Preguntas?

Resumen de la parte I

- ✓ DockerHub
- ✓ Orquestación de contenedores con Kubernetes
- ✓ Manejo de cluster local con Minikube



Break

¡30 minutos y volvemos!

Temario

19 – Parte I

Orquestación

- ✓ DockerHub
- ✓ Orquestación de contenedores
- ✓ Orquestación con Kubernetes

19 – Parte II

Tercera Práctica Integradora

- ✓ [Skills](#)
- ✓ [Práctica integradora](#)

20 – Parte I

Seguridad

- ✓ Cultura de seguridad
- ✓ OWASP
- ✓ OWASP Top 10

Glosario

DockerHub: es una librería, o repositorio de imágenes en la nube.

Kubernetes: es una tecnología de orquestamiento de contenedores, se trata de una plataforma que sirve para administrar cargas de trabajos y servicios.

Minikube: es un software que permite levantar un cluster local de kubernetes y hacer las pruebas necesarias.

Objetivos de la clase – Parte II

- Hacer una integración práctica de todos los conceptos vistos hasta el momento, bajo el desarrollo de un proyecto paralelo a nuestro proyecto final.

¿Cómo organizar la clase?

Estructura general de la clase	Tiempo de dedicación	Enfoques
Skills	10 min	<ul style="list-style-type: none">✓ No detenerse a explicar cada skill✓ Utilízalo como motivador, no como stopper.
Práctica integradora	1hr 40 min (contemplando break a tu elección)	<ul style="list-style-type: none">✓ Recuerda que estás retomando un proyecto de práctica integradora previo. No pierdas tiempo repasando cosas de la práctica integradora anterior✓ Centralizarse en la reestructura del proyecto en cuanto a su arquitectura✓ El Dao puede ser algo desafiante según cómo lo armemos, se recomienda tomarse el tiempo para explicar a detalle su estructura.✓ Además, se complementarán elementos de roles, variables de entorno, un rol de admin y envío de correos para diversas situaciones.
Presentación del desafío	10 min	

¿Qué estamos por ver?

Práctica integradora

La mejor forma de repasar los temas vistos hasta el momento, es hacer un repaso integrado de todos los elementos.

Si bien es correcto repasar el código parte por parte, es importante que como desarrolladores comencemos a trabajar en nuestra **lógica de integración**, es decir, tenemos que tener siempre contemplado el cómo vamos a juntar todo lo aprendido, para tener un proyecto sólido



Elementos a integrar

Elementos a integrar

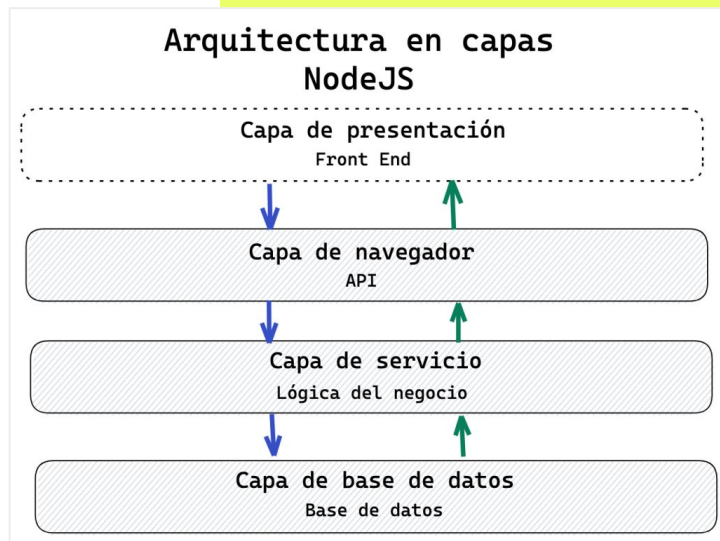
En el desarrollo de esta práctica de integración, repasarás y conectarás los siguientes conceptos:

- ✓ Arquitectura por capas
- ✓ Patrones de diseño
- ✓ Mailing
- ✓ Manejo de variables de entorno

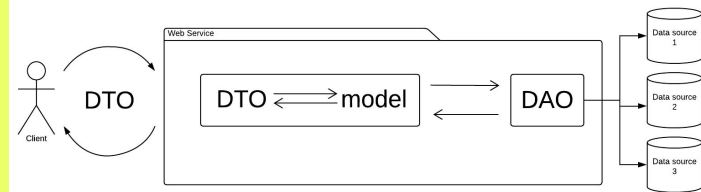
Skills para esta práctica integradora

Skills para Arquitectura por capas

- ✓ Comprender el manejo de arquitectura por capas
- ✓ Entender e isolar la responsabilidad de ruteo
- ✓ Entender e isolar la responsabilidad de controlador
- ✓ Entender e isolar la responsabilidad de servicio
- ✓ Entender e isolar la responsabilidad de persistencia



Skills para patrones de diseño



- ✓ Entender e implementar el patrón de diseño Data Access Object (DAO)
- ✓ Entender e implementar el patrón de diseño Data Transfer Object (DTO)
- ✓ Entender e implementar el patrón de diseño Repository
- ✓ Entender e implementar el patrón de diseño Factory

Skills para Mailing

- ✓ Comprender sobre el protocolo SMTP
- ✓ Estructurar correos básicos a partir de una cuenta configurada de gmail.
- ✓ Implementar attachments en un correo electrónico.



Skills para entornos

- ✓ Entender la diferencia entre un entorno de desarrollo y un entorno de producción.
- ✓ Configurar entornos a partir de .dotenv
- ✓ Setear variables de entorno y un archivo general de configuración del aplicativo.

Importante

La práctica hace al maestro, si consideras que aún necesitas practicar algún aspecto de los skills mencionados, siempre puedes repasar las clases previas. Sin embargo, no significa que no puedas avanzar.

¡Esta práctica integradora te ayudará a sentirte más seguro en estos temas!

Desarrollo de esta práctica integradora



1

Reestructuración de arquitectura del servidor para trabajar por capas

2

Reestructuración del modelo de persistencia para ajustarse al modelo de capas y funcionando con patrones de diseño

3

Configuración de envío de correos y seteo de variables de entorno.

¡Comenzamos!



Práctica de integración sobre tu ecommerce



DESAFÍO COMPLEMENTARIO

Consigna

Con base en el proyecto que venimos desarrollando, toca solidificar algunos procesos

Aspectos a incluir

- ✓ Realizar un sistema de recuperación de contraseña, la cual envíe por medio de un correo un botón que redirija a una página para restablecer la contraseña (no recuperarla).
 - link del correo debe expirar después de 1 hora de enviado.
 - Si se trata de restablecer la contraseña con la misma contraseña del usuario, debe impedirlo e indicarle que no se puede colocar la misma contraseña
 - Si el link expiró, debe redirigir a una vista que le permita generar nuevamente el correo de restablecimiento, el cual contará con una nueva duración de 1 hora.
- ✓ Establecer un nuevo rol para el schema del usuario llamado "premium" el cual estará habilitado también para crear productos
- ✓ Modificar el schema de producto para contar con un campo "owner", el cual haga referencia a la persona que creó el producto
 - Si un producto se crea sin owner, se debe colocar por defecto "admin".
 - El campo owner deberá guardar sólo el correo electrónico (o _id, lo dejamos a tu conveniencia) del usuario que lo haya creado (Sólo podrá recibir usuarios premium)
- ✓ Modificar los permisos de modificación y eliminación de productos para que:
 - Un usuario premium sólo pueda borrar los productos que le pertenecen.
 - El admin pueda borrar cualquier producto, aún si es de un owner.



DESAFÍO COMPLEMENTARIO

Aspectos a incluir

- ✓ Además, modificar la lógica de carrito para que un usuario premium NO pueda agregar a su carrito un producto que le pertenece
- ✓ Implementar una nueva ruta en el router de `api/users`, la cual será **`/api/users/premium/:uid`** la cual permitirá cambiar el rol de un usuario, de "user" a "premium" y viceversa.

Formato

- ✓ Link al repositorio de GitHub con el proyecto completo (No incluir `node_modules`).

Sugerencias

- ✓ Te recomendamos testear muy bien todas las políticas de acceso. ¡Son la parte fuerte de este entregable!

¿Preguntas?

Muchas gracias.

Opina y valora
esta clase

#DemocratizandoLaEducación